МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)

(рівень вищої освіти)

Дослідження інструментів та засобів оптимізації 3D-графіки в комп'ютерних іграх та їх застосування до ігор у жанрі "First-person Shooter"

Виконав:
студент 2 курсу групи ІПЗм-20-4

Гриневич К. В.

(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова

Керівник проф. Власенко Л.А.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ З.В. Дудар

2022 р.

Факультет Комп’ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121– Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«__»_____202_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Гриневич Кирило Вікторович
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження інструментів та засобів оптимізації 3D-графіки в комп'ютерних іграх та їх застосування до ігор у жанрі "First-person Shooter"»

затверджена наказом університету від «__»_____202___р. №____

2. Термін подання студентом роботи до екзаменаційної комісії «__» _____202__р.

3. Вихідні дані до роботи *інтеграція даних, optimization, jenject, game architecture pattern, 3D-graphics, пояснювальна записка.*

4. Перелік питань, що потрібно опрацювати в роботі *аналіз предметної галузі та постановка задачі, дослідження та аналіз існуючих двигунів, дослідження методі оптимізації ігор, реалізація*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	17.01.2022	виконано
2	Аналіз існуючих методів	31.01.2022	виконано
3	Проектування та розробка програмної системи	21.02.2022	виконано
4	Підготовка пояснювальної записки	14.03.2022	виконано
5	Підготовка презентації та доповіді		
6	Перевірка на академічний плагіат		
7	Нормоконтроль		
8	Рецензування		
9	Занесення диплома в електронний архів		
10	Попередній захист		
11	Допуск до захисту		

Дата видачі завдання 17 січня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Власенко Л.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 62 с., 26 рис. 11 джер.

OPRIMIZATION, FIRST-PERSON SHOOTER, ZENJECT, GAME, ARCHITECTURE PATTERN

Об’єктом дослідження є засобів оптимізації 3D-графіки в комп'ютерних іграх та їх застосування у жанрі «First-person Shooter»

Метою роботи є аналіз засобів оптимізації 3D-графіки в комп'ютерних іграх та їх застосування у жанрі «First-person Shooter».

Результатом практики є аналіз літературних джерел, уточнення постановки задачі і збір матеріалу для узагальнення теоретичної частини за темою архітектурних та графічних рішень для 3D-графіки у жанрі «First-person Shooter».

OPRIMIZATION, FIRST-PERSON SHOOTER, ZENJECT, GAME, ARCHITECTURE PATTERN

The object of research is the means of optimizing 3D graphics in computer games and their application in the genre of "First-person Shooter"

The aim of the work is to analyze the means of optimizing 3D graphics in computer games and their application in the genre of "First-person Shooter".

The result of the practice is the analysis of literature sources, clarification of the problem and collection of material for generalization of the theoretical part on the topic of architectural and graphic solutions for 3D-graphics in the genre of "First-person Shooter".

ЗМІСТ

Вступ 7
1 Аналіз предметної галузі та постановка задачі 9
1.1 Аналіз предметної галузі 9
1.2. Аналіз існуючих програмних рішень на ринку 15
1.3. Постановка задачі 20
2 Дослідження та аналіз існуючих ігрових двигунів 22
2.1. Дослідження ігрового двигуну 22
2.2. Аналіз ігрового двигуна Unreal Engine 24
2.3. Аналіз ігрового двигуна Unity 27
3. Дослідження методів оптимізації ігор 31
3.1 Дослідження оптимізації графіки 31
3.2 Дослідження Entity Component System 38
3.3. Дослідження Dependency Injection 40
3.4. Дослідження використання Model-View-Controller у двигуні Unity 43
4. Реалізація 48
Висновки 50
Перелік джерел посилання посилань 51
Перелік джерел посилань за науковими напрямами науковців кафедри програмної інженерії 52
ДОДАТОК А Апробація результатів роботи 53
ДОДАТОК Б Звіт ркзультатів перевірки на унікальність тестку в мережі інтернет та бізу ХНУРЕ 55
ДОДАТОК В Слайди презентації 56

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ЕЦП – електронний цифровий підпис;

ПЗ – програмне забезпечення;

FPS – First-person Shooter;

TPS – third person shooter;

3D –3 Dimension graphics;

ВР – віртуальна реальність;

ПК – персональній комп’ютер.

ВСТУП

З розвитком комп'ютерів та його популізацією в бізнес сегмент людство розпочало активне використання їх потужностей для повсюдних завдань. Спочатку це були найбільш спеціалізовані наукові завдання, але як тільки розробка та створення персонального комп'ютера стала масовим явищем і навіть звичайні люди змогли дозволити собі раніше небачену розкіш, ентузіасти почали замислюватися про впровадження цієї технології в сегмент розваг.

Першою грою, що вийшла в тираж, по праву можна вважати Spacewar. Гра була випущена в 1961 групою студентів ентузіастів на комп'ютері університету, після в 1971 через цілих 10 років оптимізована для приміщення на маленьку плату з можливістю підключення до телевізора. Таким чином був створений перший ігровий автомат і відмінний приклад того, що оптимізацією ігор почали займатися від початку їх поширення у маси.

Першою грою у жанрі стрілялка від першої особи стала "Maze War". Оточення гри було побудовано за допомогою тайлової графіки. Це була гра, де гравець спостерігав за діями персонажа з боку його очей. Суть була простою, користувач рухався лабіринтом і міг стріляти у ворогів які були перед ним, за вбивство нараховувалися окуляри, за смерть забиралися. Якщо подивитися на більшість сучасних стрілялок, можна побачити схожість системи нагороди гравця у матчі та загальної концепції гри.

З моменту свого створення ринок відеоігор отримав стрімкий розвиток. Так на сьогоднішній день цей сегмент займає велику нішу ринку розваг. Під кінець 2021 року цей ринок приніс загального доходу вдвічі більше ніж усі касові збори від відеопрокату у світі та перевершив дохід топ-3 спортивних клубів світу разом узятих та склав 175 мільярдів доларів США. Близько півмільярда людей дивилися трансляції кіберспортивних турнірів за 2021 рік, що є однією шістнадцятою частиною населення планети, що явно відображає зацікавленість людей у даному сегменті розваг.

На даний момент FPS є одним із найпопулярніших жанрів ігор. Дані ігри в основі своїй не вимагають від гравців великих витрат для освоєння в грі і часто пропонують швидкі ігрові сесії, які мають високу концентрацію активностей і веселощів. Саме тому такий мастодонт індустрії як "Activision blizzard" щороку випускають неймовірно успішну серію гри "Call of Duty", яка продається величезним тиражем і щороку мільйони фанатів з нетерпінням чекають на нову частину серії.

Саме своєчасний розвиток та оптимізація ігрової фізики та графіки є одним із факторів успішності серії. Більшість гравців із середніми та слабкими комп'ютерами мають можливість насолодитися одночасно сучасною графікою та гарним геймплеєм завдяки даним процесам розробки сучасної гри.

В ході атестаційної роботи магістра була розроблена презентація, досліджені та проаналізовані методи оптимізації 3D ігор у жанрі FPS на популярному ігровому двигуні Unity, та різні жанри графіки в цілому.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

У світі комп'ютерні ігри давно закріпили себе як окремий вид мистецтва, який поєднує у собі безліч інших, класичних жанрів цієї науки. Саме тому на сьогоднішній день існує безліч ігрових продуктів, кожен з яких має свої унікальні риси, різнобічні світи, особливості геймплею, свій стиль візуальної подачі, який може бути сумісний з різним відображенням світу для гравець (2д графіка, 2.5д графіка, 3д графіка та їх суміші що утворюють із себе віртуальні світи).

Дані властивості та критерії створюють новий виток у розвитку мистецтва та його подальшого розширення додаючи нові базові поняття такі як ігрова інтерактиву – можливість персонажа (гравця) взаємодіяти з елементами оточення, безпосередньо впливаючи на них та змінювати сам світ, в якому він знаходиться, в окремих моментах повністю перебудовуючи його, створюючи вже зовсім новий світ. Саме тому було прийнято рішення класифікувати ігри, визначити жанри, які будуть об'єднувати їх за схожою концепцією. У цьому блоці ми розглянемо основні жанри для подальшого розуміння процесу дослідження.

Для початку давайте дізнаємося, що таке комп'ютерна гра - це інтерактивне програмне забезпечення, яке будуватися на базі певного ігрового світу управління або вплив на який проводиться за допомогою дій гравця або самого світу внаслідок чого користувач може спостерігати зміни в додатку, зображенні або звук (часто все це відбувається одночасно), в результаті чого користувач отримує новий досвід і певні емоції. Виходячи з цієї гри повинні впливати на емоції і психічно залучати гравця в гру, внаслідок чого гравець більше вірить у реальність того, що відбувається, і сповнюється симпатією до свого ігрового персонажа. З огляду на сказане раніше гра повинна мати добре прописану систему і методи за допомогою яких впливатиме на стан користувача, що грає в неї, але не переступаючи певну межу, даючи гравцеві відчуття ігрового процесу.

Комп'ютерна гра має як спільні риси з програмним забезпеченням, а й особливості, які змушують винести їх у окрему категорію ПЗ.

Спочатку розглянемо спільні особливості розробки. Основною загальною рисою є сама методологія розробки гри, більшість циклів розробки що застосовуються до ПЗ можна застосувати і до відеогри, так само багато і технології можуть застосовуватися, хоч і в деяких випадках вийдуть зовсім інший результат. Будь-яка програма має орієнтацію на певну аудиторію, нішу людей і має всі процеси необхідні для бізнес процесів такі як маркетинг, реклама модель реалізації та доходу. При розробці відеоігри так само необхідна обчислювальна техніка, за допомогою якої розробник вироблятиме необхідні дії в очікуванні результату.

Тепер ми можемо перейти до характеристик розробки які відрізняють комп'ютерну гру від ПЗ. Найголовніше — це призначення гри, зазвичай ПЗ спрямоване на вирішення певної проблеми, чи то створення оптимального маршруту, чи оптимізація раціону людини тощо. Ігри ж насамперед призначені для задоволення внутрішніх вимог людини, таких як моральне та психологічне, та загальної потреби у проведенні часу та розваги. Коли сучасна компанія замислюється про розробку гри, вона не дивитиметься на гру як на продукт для вирішення проблеми певного осередку спільноти, насамперед розглядатиметься необхідність даного жанру в суспільстві в цілому, потреба людей у такому вигляді розваги та насиченість ринку на даний момент такими світами. . Комп'ютерна гра в основі свого творчого процесу може активно змінюватися зі зміною потреби суспільства на момент розробки і саме тому не завжди вдається стандартизувати і планувати повний цикл розробки. Також для створення успішного продукту, деякі частини програми можуть вимагати досліджень і консультацій з фахівцями в даній сфері та у великій ААА грі таких компонентів може бути не один десяток, і роль фахівця буде динамічною у міру зміни вимог світу гри. Виходячи з цього процес розробки вимагає гнучких та адаптивних методів, розрахунку безлічі факторів ризику, що будуватиметься на суб'єктивному сприйнятті з боку гравця кожного прийнятого рішення, такого як:

– інтерфейс;

– стиль гри;

– вибрана колірна гама;

– основні ігрові механіки;

– сюжет.

Таких факторів може бути безліч, ми привели лише основні з них. Саме тому ризики розробки комп'ютерної гри майже завжди вищі ніж звичайний ПЗ і така розробка має труднощі з визначенням шансу на провал даного продукту.

Комп'ютерні ігри в основі своїй осмислено поділяють на різновиду за певними ознаками з допомогою жанрів. Найчастіше критерії оцінки приналежності до певного жанру часто не визначені однозначно, класифікація комп'ютерних ігор має слабку систематизацію унаслідок чого кожна з команд розробників найчастіше трактують жанр своєї гри по-своєму. Однак світова спільнота гравців і розробників змогла прийти до загальних констант для кожного жанру, внаслідок чого визначити приналежність будь-якої гри до жанру не важко для звичайного користувача.

Давайте розглянемо основні жанри відеоігри, головні із них такі:

Action (3D-shooter) – в іграх даної категорії гравці в основі своїй керують одним персонажем від очей гравця або від іншого розташування камери (наприклад за спиною гравця). Основний геймплей будувати на знищення різних ворогів за допомогою всіляких видів озброєння або підручних засобів наодинці або групі союзників якщо йдеться про розраховану на багато користувачів гру. Є дві найпопулярніші категорії в даному жанрі:

– FPS, де гравець керує своїм персонажем від очей персонажа. Відомі представники жанру серія гри «Call of Duty» та «Battlefield»;

– TPS де часто гравець керує своїм ігровим персонажем з боку спини або іншого ракурсу, спрямованого на нього. Одним із популярних представників поджанру є гра «Fortnite», яка приносить мільярди доходу своїм розробникам щороку.

Файтинг – час проведення гравця будуватися на поєдинку з одним або більше опонентів, якими управляє сама гра або інший гравець. Найчастіше в цьому жанрі гравець може використовувати обмежений набір зброї, який прив'язаний до

обраного ігрового персонажа або інтерактивних елементів оточення, якщо є на карті. Популярним представником жанру є серія ігор «Mortal combat» та «Tekken». Дані ігри характеризують себе рідкісною зміною базових механік жанру і складним порогом входу для гравців-початківців.

Аркада – тип ігор, у яких гравцю доводиться приймати швидкі рішення, спираючись на свою реакцію та швидкість рефлексів. Даному жанру характерна добре розвинена система заохочення гравця за успішні дії як бонусів або інших ігрових предметів, атрибутів.

Симулятори – імітація моделі поведінки певного об'єкта чи підгрупи об'єктів, світів узята з реального світу, де гравець має максимально наближену реальності модель поведінки об'єкта шляхом відтворення їх у грі. Симулятори діляться на аркадні, де основна симуляція полягає в основних правилах роботи категорії, популярний приклад серія гоночного арканного симулятора «Forza horizon» де основний упор зроблений на швидкість і динамічність гри, полегшуючи процес керування автомобілем. Реалістичні симулятори – тип ігор, у яких максимально детально описується робота певної системи, акцент робиться на точну передачу поведінки об'єктів. Популярний представник гоночний симулятор «Forza motosport» і військовий симулятор «Arma 3».

Стратегії - дані гри розраховані на стимуляцію у гравця стратегічного або логічного мислення для вирішення низки внутрішньоігрових проблем, наприклад знищення ворожого загону, що переважає союзний чисельністю використовуючи особливості дизайну карти або інших ігрових механік. У більшості випадків гравець може керувати не одним персонажем, а цілою групою різних юнітів, призначених для вирішення вузького чи широкого спектру завдань. У особливо масштабних іграх цього жанру гравець може управляти цілими містами чи державами. Ігри даної серії діляться на покрокові, де кожен гравець (команда) виконують дії з певної гри черги та маю сильні обмеження в діях поза своїм ходом, дані ігри мають монотонних геймплей та довгі ігрові сесії. Популярний представник даного жанру серія ігор «Might & Magic Heroes», яка мала неймовірну популярність на початку 20-го століття. Стратегії в реальному часі, де всі дії

відбуваються в реальному часі, і кожен гравець приходить контролювати безліч процесів одночасно. Дані ігри маю більш динамічний геймплей і складність входу для гравців-початківців. Популярний представник жанру гра «Starcraft 2». Є так само ігри, що змішують обидва жанри такі як «Total War».

Пригоди – в основному оповідальні ігри, де гравцеві необхідно вирішувати головоломки різного ступеня складності взаємодіючі з ігровим світом за допомогою сюжетних предметів. Популярний представник "Space Quest".

Рольові ігри - дані ігри мають певні риси, які їх виділяють, а саме:

– персонажі гри (як дружні так і ворожі) мають певний набір параметрів, які визначають їх характер, силу та здібності під час бою;

– рівнева система, що дозволяє визначити загальний рівень сили персонажа від сукупності всіх його характеристик;

– параметри, що були вказані в попередніх розділах, можна покращувати шляхом виконання певних ігрових умов (завдання, знищення ворогів, захоплення території тощо);

– загальний світ гри має сильне опрацювання основних аспектів, які мають вплив на світ або сюжет;

– Розвинена система комунікації між персонажами;

– великий вибір різних елементів екіпірування та предметів взаємодії, які так само мають свої характеристики і в деяких ситуаціях можливість поліпшення шляхом виконання умов або завдань.

Популярний представник "Dragon Age".

Традиційні (настільні) – реалізація класичних ігор реального світу як комп'ютерної інтерпретації.

Тепер коли ми знаємо основну інформацію про те, що таке комп'ютерна гра і яка вона може бути, настав час дізнатися, що таке 3D графіка.

Вже понад 26 років з моменту виходу першої справжньої тривимірної гри «Quake» у 1996 році, 3D розвивається неймовірними темпами, як і графічні процесори, які дозволяють розробникам дедалі ближче наблизити графіку до реалістичного рівня та не відлякував велику аудиторію системними вимогами. У 2010 році вийшов «DirectX 8» який дав можливість розробникам використовувати шейдери за допомогою яких центральному процесору більше не було потреби займатися обробкою графіки і цю рутинну для нього роботу вдалося перевести на графічний процесор.

3D – вид комп'ютерної графіки, в якому зображення візуалізується у тривимірному світі або об'єкті та за допомогою рендерингу проводиться зображення на двовимірному дисплеї пристрою у вигляді тривимірного об'єкта.

У 3D всі об'єкти є моделями, вони ж у свою чергу складаються з міша та матеріалу, даний об'єкт можна уявити як безліч плоских фігур, кожна з яких складається з двомірного малюнка.

Меш - це фізична форма об'єкта, яка складається з полігонів, вони ж у свою чергу є трикутником або чотирикутником (використовуються для тесселяції та згладжування). Зазвичай в ігрові двигуни працюють з трикутниками, бо з ними простіше здійснювати розрахунки, але більшість моделей будуються з чотирикутників, вони без проблем перетворюються в трикутники і при їх створенні менше шансів на появу артефактів. Полігони можуть бути як односторонніми, так і двосторонніми, перший видно тільки з одного боку, тоді як з іншого він прозорий, інший видно відразу з двох сторін. Полігон має вектор нормалі, що визначає перпендикулярність площини грані. Даний вектор використовується для розрахунку світла, яке потрапляє на полігон, чим кут між світлом і полігоном менше, тим більше світла приймає полігон. У кожного меша є текстурні координати за допомогою яких відеокарта визначає, де саме розташувати меш у світовій координаті гри. Матеріал – є на текстуру меша, тобто її візуальне розфарбування та інформацію для кожної грані як саме вона має виглядати.

В ігрових реаліях всі моделі можуть розташовуватися на різній від гравця дистанції, і вона може динамічно змінюватись, але може змінюватись навіть дистанція між двома окремими полігонами в одній моделі. Саме тому сучасні відеокарти мають апаратні блоки для розрахунку та коректного відображення даної дистанції.

Вершинні шейдери - це об'єкти, які зберігають у собі всю необхідну інформацію для відеокарти, щоб вона змогла коректо відмалювати цю модель в ігровому світі, враховуючи всі джерела світла, їх потрапляння на полігони і так само координати кожного з них та іншу інформацію.

За допомогою 3D конвеєра відеокарта обробляє дані моделі та створює зображення у три робочі етапи. Перший це перетворення всього в мозаїчну модель для розбиття на полігони, далі відбувається геометричне перетворення, і розстановка освітлення у світі та в заключному етапі створюється двомірне зображення на дисплеї за допомогою математичних моделей та формул, розраховуючи все необхідне.

Сучасна оптимізація 3D є постійним пошуком компромісу між збільшенням кількістю полігонів, що підвищує якість зображення або знаходження їх оптимальної кількості зі світловою корекцією для оптимальної частоти кадрів у грі. Саме тому більша частина сучасної оптимізації ігор відбувається на стороні розробника відеоігри в самому ігровому движку за допомогою знаходження необхідної архітектури та методів вирішення проблеми навантаження системи.

1.2. Аналіз існуючих програмних рішень на ринку

Оскільки ми розроблятимемо і оптимізуватимемо свою гру в жанрі FPS буде доречно розглянути кілька успішних прикладів ігор у даному жанрі, проаналізувати їх модель монетизації, загальне стилістичне оформлення гри, основний базис на якому побудований геймплей гри, технічну реалізацію та її

оптимізацію. Виходячи з необхідного набору даних і наближеності сеттинга, були обрані такі ігри: Call of Duty Vanguard, Pavlov VR.

Call of Duty Vanguard була випущена в тираж восени 2021 з тісною інтеграцією з іншою грою даної серії Call of Duty Warzone і є 19 номерною частиною серії. Розробником виступала команда Sledgehammer Games, а видавцем всім відома Activision. Гра мала трирічний цикл розробки, як і всі минулі ігри серії, починаючи з другої її частини. Побудована на ігровому движку власного виробництва IW Engine 8.0, який не мав великих оновлень з моменту виходу 17 великої частини серії.

Модель монетизації. Ця гра була випущена за класичною для цієї серії моделі одноразового платежу. Купив одного разу – грай завжди. Але також у гру було інтегровано модель ігрового пропуску, що вже стала класикою для сучасних ігор. Дана система, призначена для сезонної купівлі ігрового абонементу, який гравець може покращувати, виконуючи ігрові місії або просто граючи в гру, а нагороду отримувати внутрішньо ігрові предмети візуалізації. Перепустку необхідно купувати кожен новий ігровий сезон, але для найвідданіших гравців, за проходження всіх рівнів дається достатня кількість внутрішньо ігрової валюти, для покупки наступного, залишок валюти гравець може витратити у внутрішньо ігровому магазині для покупки предметів візуалізації або наборів для зброї. Дана модель ігрових перепусток є вдалим рішенням для монетизації гри та продовження її життєвого циклу, нові гравці отримують сезонне оновлення контенту та ігрових режимів, що робить покупку гри актуальною весь життєвий цикл, старі гравці отримують безліч внутрішньо ігрових активностей, які вони можуть виконувати, що стимулює їх повернуться у гру та зіграти ще один матч. Ігровий абонемент не дає жодних переваг у самому геймплей крім візуалізації, що зменшує негативне ставлення гравців до нього.

Загальний стиль оформлення. Події гри відбуваються до кінця Другої світової війни, тому загальний візуальний стиль ігрової графіки є реалізмом, а сеттинг середина 50-х років минулого століття. Гра спрямована на зображення

спецпідрозділів, які виконують свої завдання у різних куточках світу. Приклад візуальної складової гри можна спостерігати на рисунку 1.1.

Рисунок 1.1 – візуальне зображення гри “Call of Duty Vanguard”

Геймплей. Основний геймплей гри не змінився з 2019 року, коли вийшла 17 частина серії, яка привнесла найбільші зміни в серії, до неї входило нова система очок, новий ігровий движок, чудова оптимізація та візуальна супроводжуюча ігри, чудовий звуковий супровід та детальна анімація кожної зброї, наступні частини тільки намагалися досягти поставленої планки, але не могли це зробити, основною проблемою була технічна частина гри, про що буде сказано пізніше. Основні механіки залишилися без змін, динамічні бої маленьких команд до 10 осіб на невеликій карті, що підходять під необхідну кількість гравців. Системою бонусів за серії перемоги над ворогам у матчі та різних тимчасових нагород за них. Прокачуванням зброї за досвід і т. д.

Технічний стан гри. На момент виходу гри її технічний стан був задовільним, графічна частина була на рівні минулих частин серії, технічна

реалізація не набула великих змін. Але весь жахала друга частина серії “Warzone”, гра була випущена в 2019 році і мала хороші технічні характеристику та оптимізацію, на той час люди могли грати навіть на досить слабких системах, але вихід кожної нової частини серії та її інтеграція погіршувала ситуацію і на сьогоднішній день люди деякі користувачі, що могли раніше грати в цю гру при задовільній частоті кадрів, не мають такої можливості. Основна причина такого різкого погіршення оптимізації є не прагнення розробників покращити графічну частину гри, а погано побудована архітектура минулої частини гри, яка не розглядала можливості такого різкого зростання гри, зі збільшенням життєвого циклу програми в кілька разів, що призвело до безлічі внутрішніх конфліктів у системі.

Саме з останніх причина дана гра була розглянута як приклад поганої архітектури, що мала сильний вплив на загальну продуктивність та оптимізацію гри. Якщо ж прибрати цей фактор, дана гра є чудовим прикладом класичної FPS з відполірованим роками геймплеєм, що пройшла багаторічне випробування системою монетизації та гарний приклад важливості оптимізації гри на підтримку її популярності.

Игра “Pavlov VR” была выпущена и издана студией “Vankrupt Games” в 2017 году на персональные компьютеры с последующим импортом на автономный шлем виртуальной реальности “Oculus Quest 2”.

Модель монетизації. У момент виходу гри студія розробник була не великою компанією, тому гра поширюється за схемою купив одного разу, граєш завжди.

Загальний стиль оформлення. Даний продукт, розрахований для відтворення на шоломах віртуальної реальності, тому має так само, як і минулий зразок реалістичну стилістику, хороші анімації для руху персонажа, щоб не викликати огиди у гравців при різноманітних рухах тілом персонажа. Загальний рівень графіки є середнім. Ми можемо спостерігати загальний стиль гри та якість її графіки рисунку 1.2.

Рисунок 1.2 – одна із ігрових мап гри “Pavlov VR”.

Геймплей. У сам жанр гра не привнесла нічого нового, навіть перенесення FPS у світ ВР не був новинкою, гравці так само борються невеликими командами один проти одного, використовуючи заздалегідь приготовлений набір зброї. За загальним планом ігрової логіки цю гру можна було б назвати відсталою для свого часу, але її порт у ВР та грамотне перенесення всіх основних механік такої серії ігор як “Counter-Strike” зробив її неймовірно популярною у ВР спільноті та підняв загальний рівень інтерактивності в даному жанрі ігор на платформі ВР.

Технічний стан гри. Не одним захоплюючим і в той же час легким в освоєнні геймплеєм ця гра зайняла безліч топів продажів і залишила своє місце в серцях багатьох гравців. Хоч і візуальна частина гри і виглядає на жорсткий середнячок, дане рішення дало можливість розробникам максимум можливостей движка гри і дозволило підвищити загальну продуктивність гри, оптимізувавши її для досить слабких за мірками ВР персональних машин.

Виходячи зі всього сказаного дана гра є чудовим прикладом грамотної оптимізації у ВР і продуктом, що дозволив підняти інтерактивність FPS на новий рівень, давши своїм користувачам відчути себе в ролі справжнього бійця контр терористичного загону або переміститися за інший бік барикади. Також не варто забувати про грамотно реалізовану архітектуру, яка дозволила користувачам впроваджувати в гру свої особисті карти без особливих проблем із сумісністю. Можливо, саме останній пункт послужив фінальним поштовхом у популяризації гри, але її хороша оптимізація і масштабованість для розширення контенту відіграла чималу роль у цьому процесі.

1.3. Постановка задачі

Мета наукової роботи полягає у розробці First-Person Shooter, з використанням усіх необхідних способів оптимізації, включаючи оптимізацію графіки та вибором вірною архітектурою для програми. Цей продукт буде самостійною сутністю в закритому середовищі, оскільки це навчальний проект. Також додаток повинен мати можливості для без шкідливого розширення гри та переходу на іншу модель поширення. Фінальний продукт повинен мати високу продуктивність для комфортного отримання більшої групи людей.

Основний акцент у дослідженнях буде покладено в оптимізацію об'єктів в ігровому движку та загальному стилю архітектури та коду, так як провівши дослідження ринку було виявлено, що оптимізація графіки полягає у пошуку оптимальної кількості полігонів для досягнення балансу між продуктивністю та візуальною складовою. Найбільший фактор у загальну картину оптимізації гри вносить архітектура та методи реалізації програми.

Платформою для гри були обрані персональні комп'ютери з мінімальною підтримкою шоломів віртуальної реальності, мінімальна операційна система буде Windows 10. Користувачеві буде потрібний шолом віртуальної реальності.

Середовищем для роботи зі шоломом буде Steam VR. Установка має бути інтуїтивно зрозумілою для кожного користувача та не мати зайвих етапів.

Продукт матиме серверну складову для можливості гри з іншими гравцями.

Виходячи зі специфіки розробки комп'ютерних ігор з використанням технології ВР, персональний комп'ютер користувача має відповідати мінімальним системним вимогам до техніки, а саме:

– графічний процесор GTX 1060 6 GB або RX470 8 GB або вище;

– мінімальний обсяг оперативної пам'яті 8 гігабайт;

– процесор Intel Core i5-4590 або AMD FX 8350 або нова модель;

– наявність карти для обробки звуку;

– жорсткий диск з більш ніж 4 гігабайт вільного місця.

Розробка та оптимізація FPS гри із застосуванням технології ВР має такі етапи: проектування архітектури гри, вибір системи залежностей, вибір мережного рішення (архітектури сервера або готового рішення), розробка концепту гри, вибір стилістики, створення та імпорт необхідних 3D моделей, інтеграція необхідних модулів системи , розробка основної механіки гри, тестування та виправлення помилок.

2 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ІСНУЮЧИХ ІГРОВИХ ДВИГУНІВ

2.1. Дослідження ігрового двигуну

Як тільки ігри стали більш масовим явищем, розробники стали помічати, що кожна наступна гра містить загальні компоненти, такі як:

- графічні рішення, які необхідні для відображення зображення;
- загальні ігрові механіки що переносяться від гри до гри (наприклад переміщення персонажа, де обходять розрахунки його позиції у світі та фізичне перенесення моделі);
- пост процеси, які необхідні в певних ситуаціях.

Всі подібні рішення стали виносити в окремі бібліотеки, які відповідали за вирішення тієї чи іншої проблеми, але назвати це ігровим двигуном все ще було неможливо. Здебільшого це було неможливо через серйозні апаратні відмінності систем, бо часів загального технічного рішення ще не було. Також свою роль вносила нерозбірливість в уніфікації ігрових жанрів бо кожен із них міг вимагати свого рішення.

Ситуація кардинально змінилася, коли розробники з компанії ID Software використовували напрацювання своїх колег з гри Wolfenstein 3D і створили нову гру Doom на базі повноцінного ігрового двигуна в 1993-му році. З точки зору технічної складової це був по-справжньому прорив для свого часу, вони визначили концепт модульності для ігрових двигунів і додали безліч необхідних можливостей. Одним із ключових моментів було створення свого модуля математичного обчислення для роботи з тривимірними об'єктами, яке брало за основу двовимірне та представляло його у тривимірному вигляді. Кожен модуль системи відповідав за завдання і мав особистий набір даних. Саме ця система дозволила створити низку ігор на своїй основі та покласти основу для такого терміна як “Ігровий движок” у своєму нинішньому вигляді.

Ігровий движок - це вузькоспеціалізованою операційною системою, оскільки включає всі модулі останньої. До нього входять: система управління пам'яттю,

графічна підсистема, система введення, аудіо підсистема, штучний інтелект, фізична підсистема, мережна підсистема, редактор ігрових рівнів та інше. Крім того ядро двигуна може надавати особливий підхід до роботи з файлами - файлову (ресурсну) систему, а так само від основної операційної системи засоби роботи з багатопоточністю. Сучасний ігрові двигуни також включають інтерпретатор скриптової мови, заточеного для опису ігрової логіки, а часто й цілком візуальний її редактор. Використання даного рішення дозволяє залишити опис низькорівневих команд та інструкцій на систему, а сконцентруватися на геймплей.

Ігровий двигун в першу чергу служить для спрощення та прискорення розробки. Тому включає засоби для створення ігрового світу – дизайну рівнів, імпорту необхідних матеріалів, текстурування об'єктів, завантаження або створення анімації, створення візуальних ефектів, налаштування фізичних правил світу та інших завдань.

Наступною значущою метою розробки двигуна є можливість додавання кросплатформенності для гри, що розробляється. Можливість запускати її на всіх необхідних платформах з мінімальними змінами в логіці гри. Звичайно, здійснити запуск програми без змін на іншій платформі у своєму первозданному вигляді, часто не вдасться через низку можливих апаратних відмінностей, у тому числі: розмірів екрана, засобів і способів управління та ін. Але в таких випадках як портування гри з Android на iOS можна зробити портування без особливих зусиль.

Розвиток ігрових движків відбувається разом або під впливом розвитку апаратних та програмних платформ, а також вимог гравців та розробників до більшої гнучкості та оптимізації самої системи, разом із появою нових ігрових жанрів та змінами смаків користувачів. Іншими словами, вони розвиваються разом із розвитком ігрової промисловості загалом. Надалі ми розглянемо два найпопулярніші ігрові движки з умовно безкоштовною моделлю використання.

2.2. Аналіз ігрового двигуна Unreal Engine

Ігровий двигун був розроблений компанією Epic Games, і на сьогоднішній день відомий як Unreal Engine. Основна мова програмування C++. Вперше він був показаний у 1998 році у шутері від першої особи Unreal. Хоча він був розроблений в основному для шутерів від першої особи, він ефективно використовувався в різних жанрах, включаючи стелс, файтинг, MMORPG та інші рольові ігри. Unreal Engine має великий спектр можливостей і в даний час використовується багатьма розробниками ігор. На рисунку 2.1 ми можемо спостерігати логотип даного двигуна.

Рисунок 2.1 – логотип Unreal Engine

Unreal Engine - це велика група інструментів для розробки ігор, створена для всіх, хто працює з технологіями в реальному часі. Починаючи з корпоративних додатків та кінематографічних демонстрацій і закінчуючи високоякісними

ігровими продуктами для ПК, консолей, мобільних пристроїв, віртуальної та доповненої реальності. Даний двигун дає вам все необхідне для запуску, випуску, розвитку та виділення з маси.

Набір інструментів для всіляких завдань і простота доступу до них і внутрішньої структури дозволяють розробникам швидко відтворювати ідеї та бачити миттєві результати, не заблукаючи в написанні коду, в той час як повне право доступу до основного коду дає кожному в спільноті Unreal Engine свободу налаштування та розширення можливостей двигуна під свої потреби.

Давайте розглянемо основні плюси цього рішення:

– має чудовий набір для створення різних типів графіки високої якості. Один із найкращих на ринку. Таким чином, цей двигун відомий як засіб для створення високотехнологічної гри AAA, в які славляться високим рівнем якості та опрацювання світу;
– має схеми на кшталт блоків, так що навіть художники без значних знань мови програмування зможуть використовувати двигун у цьому напрямку. Також ця функція підходить для прототипування. Але для основної роботи все одно доведеться використовувати C++ для досягнення кращого результату;
– має великий список інструментів та функцій для різних ситуацій, яких часто немає в інших рішеннях такого типу. У Unreal Engine більше візуальних редакторів, ніж у конкурентів;
– завдяки різним функціям технологія рендерингу працює ефективніше та швидше, ніж в інших рішеннях. Так рендеринг світла в сцені Unity може займати значно більше часу;
– має інструменти візуальним налагодженням, за допомогою якого ви можете керувати процесом розширення та кількістю витраченого на нього часу;
– мова програмування З++ який дає розробнику широкі можливості дій.

Тепер розглянемо основні мінуси цього рішення:

- для роботи Unreal Engine вам потрібна ліцензійна копія і так само необхідно заплатити п'ятивідсотковий податок після того, як ваша гра стане прибутковою;
- для створення простих ігор це рішення не підходить. Навіть використання для вирішення короткострокових завдань буде недоречним і ресурсно-витратним. Висока сумісність з продуктами розрахованими на довгий життєвий цикл, але вартість такої роботи буде набагато вищою, ніж у іншого двигуна;
- для людини без команди це рішення є надто затратним і складним;
- необхідно чітко відібрана команда, кожен член якої працюватиме у своїй сфері, має бути організований правильний процес відбору з урахуванням усіх основних моментів системи, а експертів у даного рішення значно менше;
- пороговий вхід, хоч C++ і є чудовим і великим інструментом, складність його вивчення все ще дуже висока і стати хорошим фахівцем дуже складно.

Виходячи з усього перерахованого вище можна дійти невтішного висновку що хоч це рішення і надає великий спектр можливостей, як і дає можливість створити високоякісну графіку з подальшою її оптимізацією, двигун все ще є мало застосовним у великих масштабах через дорожнечу в обслуговуванні і складності входу початківцям розробника, унаслідок чого утворюється дефіцит високоякісних фахівців.

2.3. Аналіз ігрового двигуна Unity

Unity - це ігровий двигун, що дозволяє створювати ігри під більшість популярних платформ за мінімально короткий проміжок часу. За допомогою цього двигуна розробляються ігри, що запускаються, на всі можливі сучасні платформи. Основний пріоритет даного двигуна, мобільні ігор або інді ігор (низькобюджетні ігри з невеликою командою розробників, але чудовою ідеєю). На рисунку 2.2 ми можемо спостерігати логотип даного двигуна.

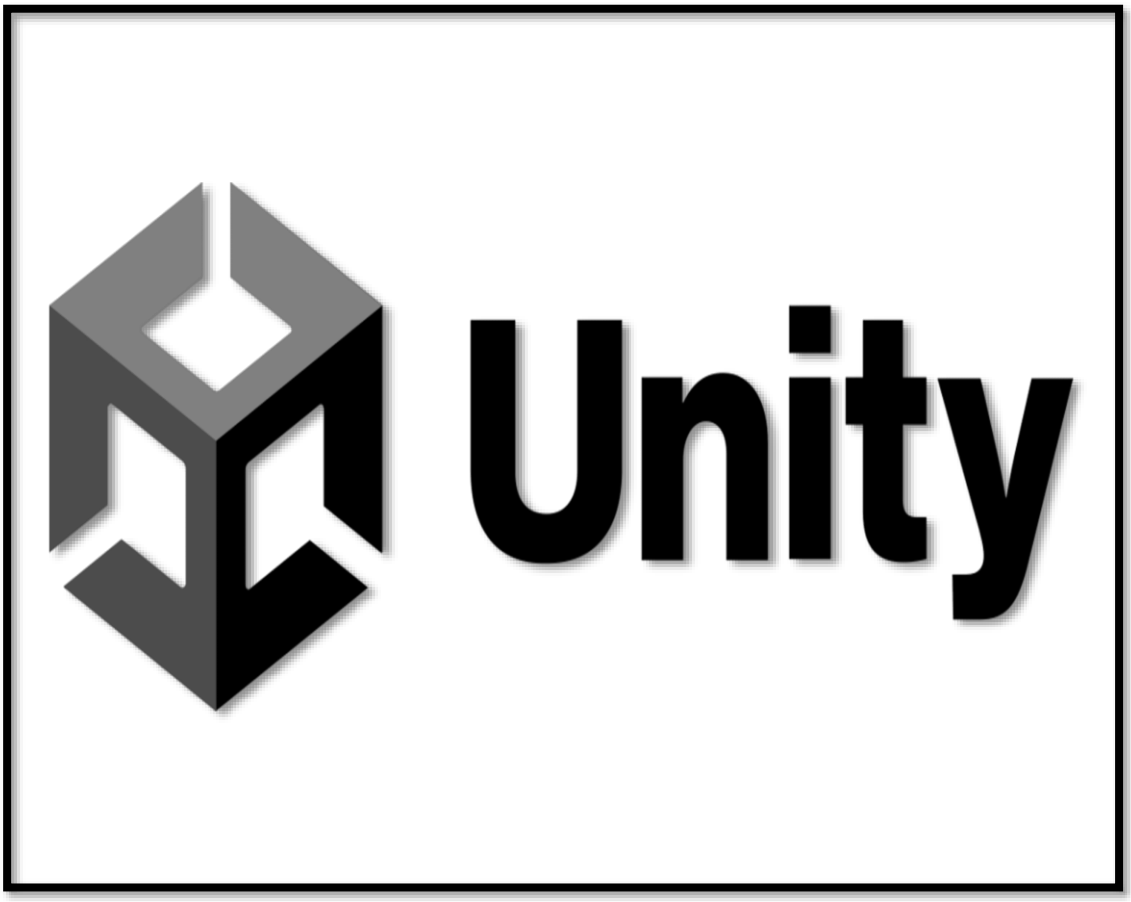

Рисунок 2.2 – логотип двигуна Unity

Раніше розробка мобільних ігор викликала у розробників тихий жах, оскільки не було готових рішень для створення цього типу продукту. Тому на створення мобільної гри могло піти купа грошей, часу і людських ресурсів. Проте

ситуація кардинально змінилася, коли у 2005 році на всесвітній конференції розробників від компанії Apple був представлений двигун Unity, який надавав весь необхідний набір інструментів для розробки мобільних ігор.

Unity виявився дуже надійним кросплатформним двигуном, який дозволив створювати ігрові програми в зручному середовищі для розробки одночасно для 27 різних платформ і пристроїв. Було представлено безліч ресурсів та готових інструментів, які були прості у використанні, докладну документацію та навчальні посібники що дозволили знизити поріг входження для нових розробників, а найголовніше, що весь цей чудовий мобільний 3D контент для ігор був абсолютно безкоштовним (навчальні інструкції та документація).

Рендеринг, сценарії, відстеження ресурсів та фізичні функції Unity допомагають суттєво скоротити час та витрати на розробку ігрових програм та запускати їх на різних пристроях без безлічі змін. Двигун Unity зараз одним з найкращих рішень для розробки кросплатформових ігор. Все через його переваги перед іншими інструментами і низького порога входження для нових розробників.

Дослідження показують, що "движок Unity займає 45% світового ринку ігрових двигунів і обраний 47% розробників ігор як основний інструмент розробки". Це суттєві цифри. Але у двигуна також є деякі недоліки, через які деякі розробники неохоче використовують двигун для розробки ігрових програм наступного покоління і дивлячись на нього в негативному ключі.

Давайте поглянемо на переваги і недоліки даного рішення, почнемо с позитивних моментів:

- двигун дуже кращий через його розширену підтримку до 27 платформ. Розроблений та розгорнутий додаток можна легко використовувати на ПК, у браузері та на мобільних платформах. Крім того, гнучка методологія дозволяє швидко створювати прототипи та випускати стабільні версії, що дозволяє прискорити процес розробки гри;
- Код пишеться у вбудованому текстовому редакторі IDE, однак розробники можуть також віддати перевагу окремому редактору коду, щоб уникнути плутанини. Крім того, вбудований редактор розробки

підтримує JavaScript для створення браузерних розширень і C# для написання сценаріїв та передбачуваних важливих функцій, що ідеально підходить для розробки ігор;

– двигун підтримує високоякісні звукові та візуальні ефекти, що спрощують процес розробки ігор. Візуальні ефекти можна легко адаптувати до будь-якого екрану та пристрою без будь-яких спотворень чи збитків для якості зображення;
– цей аспект добре розвинений і наданий самими розробниками двигуна. Розробникам-початківцям завжди потрібна зрозуміла документація, в якій міститься кожна дрібна деталь про двигун Unity;
– налагодження та налаштування стають неймовірно простими при використанні Unity, оскільки всі ігрові змінні відображаються під час гри. Це здорово, тому що дозволяє розробникам налагоджувати процес під час виконання використовуючи внутрішні інструменти самого двигуна;
– мова програмування C# дозволяє комфортно будувати високорівневі системи та використовувати всі сучасні патерни без особливих зусиль.

З плюсами закінчили, тепер розглянемо основні мінуси:

– це явно відстає, коли справа доходить до графіки. Порівняно кажучи, йому не вистачає широкого спектра можливостей створення візуально красивої графіки. Звичайно і використовуючи цей двигун можна досягти оптимальної графіки, але не як не випереджаюче час;
– у двигуні Unity вбудована підтримка фізичного двигуна PhysX, який має деякі проблеми з продуктивністю і не має важливих функцій, необхідних для створення відмінної ігрової програми. Але з роками прийшов розвиток і була додана підтримка фізики Havok, яку ще належить вивчити;
– розробники повинні мати ліцензії на кращу графіку, розгортання та покращення продуктивності. Ці ліцензії коштують дорого. Крім того, використання рендерингу, підтримки буферів, підтримки трафаретів та багатьох інших функцій збільшує загальні витрати на розробку через дорогі ліцензії;

– код стабільний в Unity, на відміну від інших двигунів, і має нормальну архітектуру, що підвищує продуктивність ігрової програми. Але недоступність вихідного коду ускладнює пошук, усунення та усунення проблем із продуктивністю;
– гра, розроблена з використанням движка Unity, споживає більше пам'яті, що, у свою чергу, призводить до помилок OOM та проблем налагодження у додатках.

З усього перерахованого вище це рішення підходить для розробки середньобюджетних ігор від великих студій, комп'ютерних малобюджетних ігор від команд-початківців і займає лідируючу позицію з розробки мобільних ігор. Дане рішення ідеально підходить, якщо у вас маленька команда і є чудова ідея, яку ви хочете втілити в життя. Звичайно, за допомогою даного рішення можна створити і високобюджетну гру, але в даному випадку краще розглянути інші варіанти реалізації.

3. ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ІГОР

3.1 Дослідження оптимізації графіки

Основне джерело витрат продуктивності графічного процесора є самі фізичні моделі об'єктів на ігровій сцені, оскільки кожен кадр повинні проводитися розрахунки їхньої позиції, відображення та вплив джерел світла на них. Найбільш ресурсно витратними вважається математичний розрахунок об'єкта, відображення матеріалів, розрахунок динамічних джерел світла, об'ємний туман та постобробка. Прорахувати вплив кожного з них на систему і побудувати рівномірне вкладення ресурсів у кожен з них для дотримання балансу по суті своєю вважається неможливою, тому що завжди потрібно враховувати людський фактор та можливі помилки при роботі з кожною з окремих систем. Далі давайте розглянемо основні способи вирішення цих проблем задля уникнення втрат у продуктивності.

Математичні розрахунки в 3D є з'єднанням між собою полігонів. Сучасні апаратні рішення можуть проводити розрахунки одночасно мільйонів вершин трикутників. Тому ця проблема легко вивчається. Зі збільшенням числа моделей та полігонів у них зростає і складність їх обчислення. Тому чим більше розробник додасть полігонів для розрахунку в ігрову сцену, тим витратнішою вона вийде.

Найпростіший і очевидніший спосіб вирішення даної проблеми, зменшення кількості полігонів у сцені до прийнятного значення. Базовий спосіб найнижчого рівня полягає у фізичному зменшенні щільності сітки об'єкта в 3D редакторі (Blender тощо), але складні фігури при зменшенні їх сітки сильно страждатимуть як вихідне зображення, тому слід дотримуватися балансу між якістю і величиною полігональної сітки об'єкта.

Дане рішення вимагає багато людини годинника, оскільки часто проводиться вручну, інструменти, які можуть у цьому допомогти є такими:

– Налаштування рівнів деталізації, створити кілька рівнів для вибору бажаного рівня оптимізації або їх вибору залежно від дальності гравця;

– Occlusion Culling, даний інструмент часто вбудований в ігровий двигун і виконується автоматично, але за бажання або необережності його можна відключити, вся суть роботи полягає у відключенні розрахунку геометрії об'єктів, на які зараз не спрямований погляд гравця, або будь яка інша модель загороджує даний предмет;

– precomputed visibility – інструмент обмеження дальності розрахунку ігрових об'єктів на сцені, як і вбудований у основні ігрові движки.

Ми розглянули всі основні інструменти та методи полегшення розрахунку геометрії для графічного процесора, тепер варто перейти до вивчення матеріалів, які завжди йдуть у зв'язці з нею.

Матеріали, тобто шкіри самих об'єктів завжди зважають на геометрію і найочевидніше питання до них полягає в їх складності. У спільноті розробників та дизайнерів ігор прийнято розділити матеріали на категорії, давайте розглянемо їх:

Непрозорі без розрахунку світла. Даний підвид є сам економним у плані ресурсних витрат системи, оскільки він передає тільки колір матеріалу без розрахунків відбиття світла або відблисків, тому його використання слід ставити в першорядний план і якщо умови дозволяють вибирати його;

Непрозорі з розрахунком світла. Цей підвид вважається більш ресурсно витратним, але якщо ми хочемо використовувати освітлення в нашій грі, обійтися без нього ми не зможемо. Один із способів обмеження навантаження у разі використання даного матеріалу, відключення всіляких променів, які може відображати даний матеріал, простими словами зробити його матовим.

Замасковані непрозорі матеріали. Їх основна риса - це два стани, в яких може бути піксель, видимий і невидимий. З використанням безлічі шарів невидимих пікселів збільшується складність розрахунків матеріалу. Але вирішення цієї проблеми є у всіх сучасних ігрових двигунах і полягає у включенні додаткової

буферизації глибини об'єктів. Цей спосіб полягає у створенні шарів глибини для камери і приміщення кожного матеріалу, на свій рівень, унаслідок чого якщо більш високий рівень перекриває відображення матеріалу нижчого рівня, рівень нижче не візуалізується.

Прозорі матеріали. Є найзаразнішими і найскладнішими в роботі матеріалами. Основна складність така ж, як і в минулому підтипі, але таке рішення не застосовується до цього типу. Практична порада в оптимізації їх може здатися банальною, виключить їх використання, але якщо вам все-таки необхідно їх використати, спробуйте не допустити накладання шарів та зменшить кількість інструкцій у матеріалі шляхом використання успадкування матеріалів.

Виходячи з усього вище сказаного, якщо ваша основна мета є створення добре оптимізованої гри, виключіть використання прозорих і невидимих матеріалів і намагайтеся якнайбільше зменшити кількість джерел світла та матеріалів що з ними взаємодіють.

Ситуація з джерелами світла виглядає набагато простіше. Усього буває три модифікації для джерел світла, динамічна, статична, і змішаний режим. Спостерігати вибір, даний режимів в ігровому движку Unity, можна на рисунку 3.1.

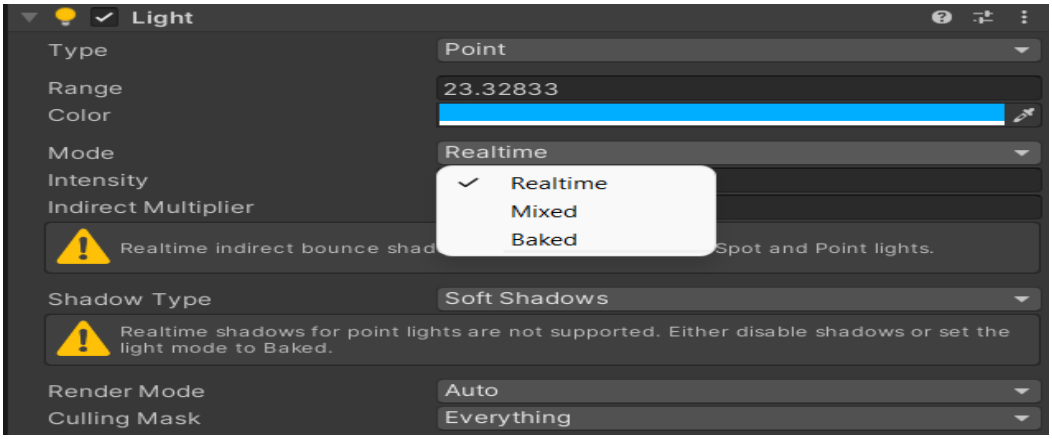

Рисунок 3.1 – меню вибору режиму освітлення.

Статичні джерела світла розраховуються при запіканні сцени в ігровому середовищі, тому вимагають лише часу у розробника та збільшують загальний час,

який потрібен на перехід у цю сцену. Тому ці джерела світла можна використовувати без сильного побоювання за вашу продуктивність.

Динамічні джерела світла несуть серйозне навантаження через постійний прорахунок тіней, які мають виходити від них. Однією з рішень обмежити дальність відображення тіней в ігровому середовищі або взагалі відключати джерела світла на певних дистанціях від гравця. Так само можна відключити розрахунок тіней, що істотно знизить накладні витрати системи. Якщо вам важлива продуктивність, найкращим рішенням буде мінімізувати застосування динамічних джерел світла. Всі ці методи сумісні один з одним.

Змішані джерела світла є проміжним між минулими режимами, що можна зрозуміти з назви режиму. Відмінності від статичних джерел світла є можливістю впливати на них в ігровому процесі, відключаючи або включаючи їх, тому кожне таке джерело світла має свій набір світлових карт. Вплив на кожен вид об'єкта чи то статичний, чи динамічний об'єкт залишається таким самим, як і в його батьків. Варто уникнути близького розміщення даних джерел один з одним. Загальні рекомендації з оптимізації зберігаються такими ж, як і з динамічними джерелами.

Туман сам по собі є досить недорогим інструментом, тому що прораховується на рівні, тому використання простого туману є допустимим та нескладним процесом. Основне навантаження може викликати лише об'ємний туман. При використанні даного туману варто бути дуже обережним або виключити його появу у вашій грі вибором інших дизайнерських рішень.

Постобробці також заслуговує на окрему увагу. У більшості рішень є можливість включення режиму HDR, який активує постобробку, але цей режим буде знижувати продуктивність, навіть на порожній ігровій сцені. Таким чином, слід розглядати включення цієї технології виключно для ПК. На малюнку 3.2 можна спостерігати включення цієї опції.

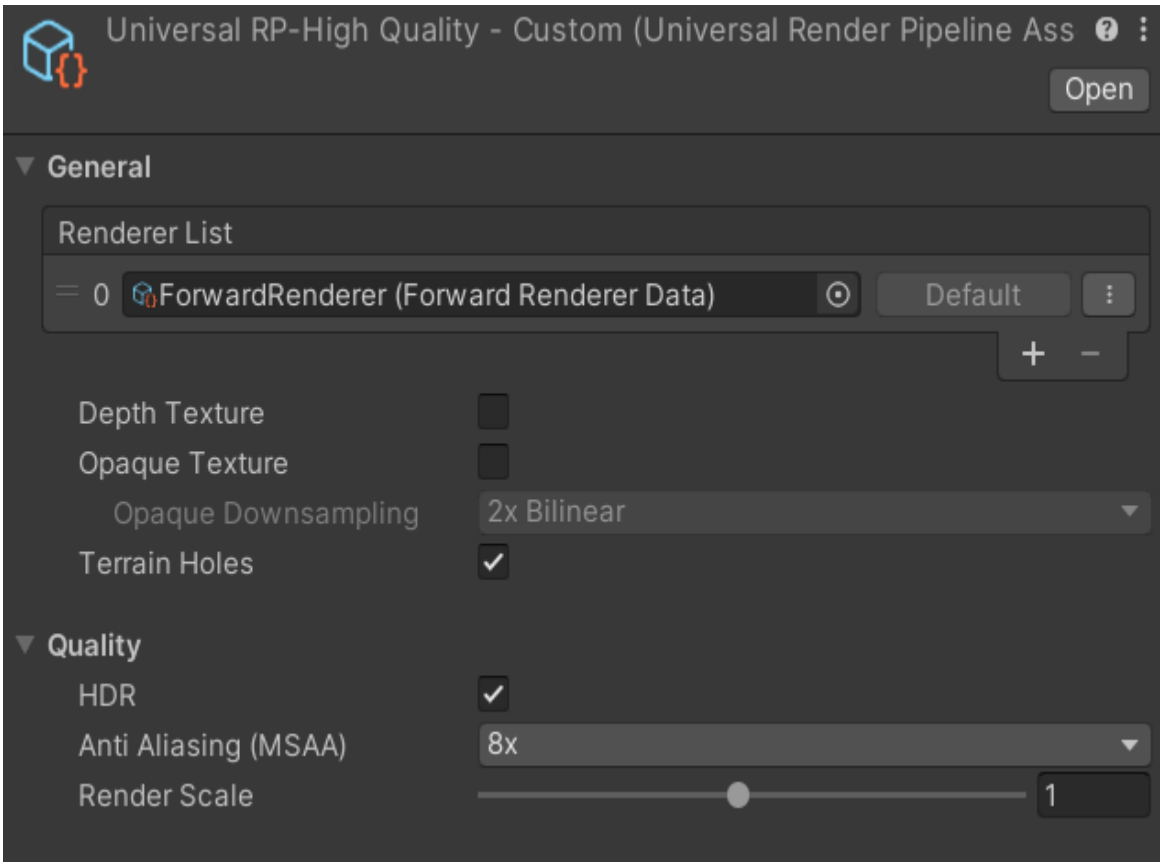

Рисунок 3.2 – ілюстрація вмикання опції HDR.

Посторобка, як випливає з назви, накладає ефекти поверх промальованої сцени, використовуючи для цього пропуск рейдеру. Наприклад, буфер глибини чи нормалі у світових координатах. Свічення є недорогим ефектом під час використання їх у стандартному режимі. На малюнку 3.3 зображено застосування ефекту свічення для створення освітлення на сцені.

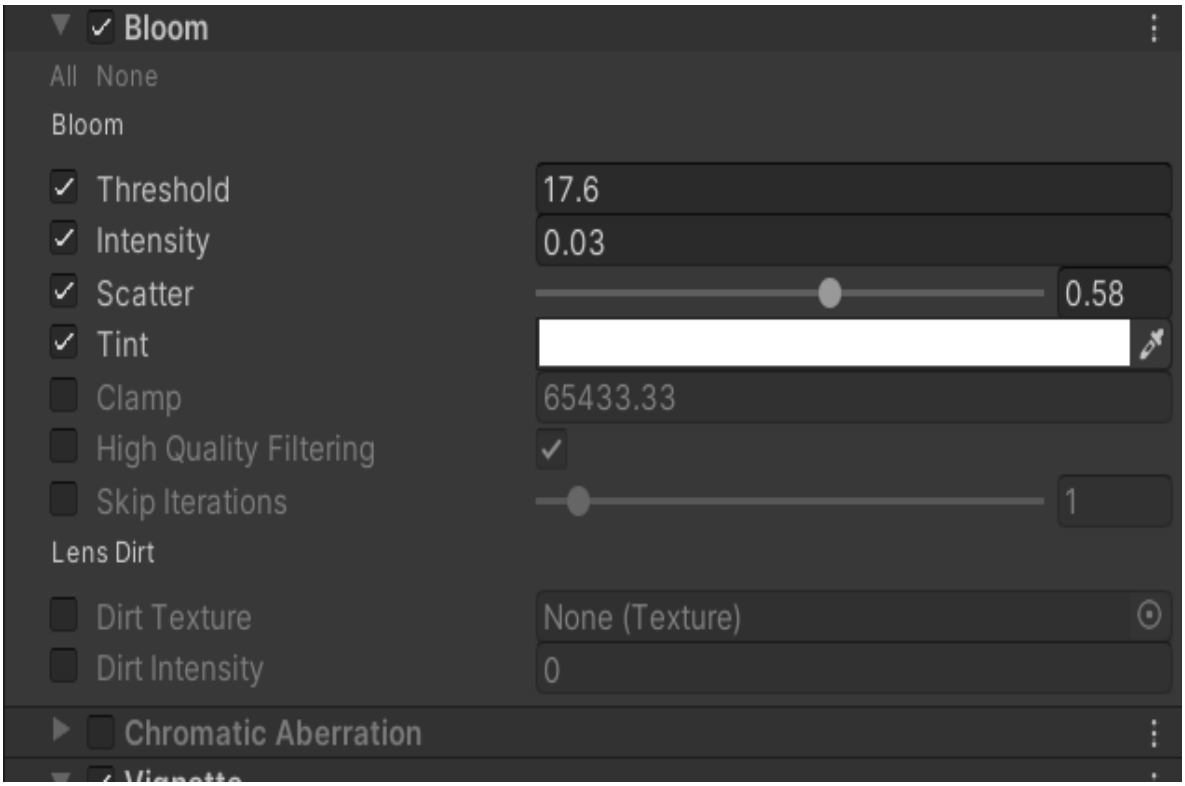

Рисунок 3.3 – пример використання ефекту свічення.

Матеріали постобробки можна вільно використовувати, але не слід забувати, що тут працюють правила, що застосовуються до інших матеріалів. Так будь-який матеріал можна перевантажити кількістю використаних текстур і складними набором команд, внаслідок чого дешевий за продуктивністю інструмент може стати дуже дорогий. Ambient Occlusion (Глобальне затінювання) є потенційно дорогим ефектом. Використовувати його можна при зниженні рівня якості даного ефекту або достатньому запасі продуктивності. Screen Space Reflections (відображення у просторі екрана) успадковує правила використання глобального затінювання. Його теж небажано, але можна використовувати при зниженні якості цього ефекту. Інші ефекти постобробки досить недорогі, тому їх використання не повинно викликати проблем із продуктивністю при дотриманні всіх попередніх рекомендацій.

Крім всього вищесказаного, окремо слід звернути увагу на роботу з пам'яттю. У ПК це може бути некритично, але слід враховувати, що не всі користувачі будуть запускати тільки одну програму і тому слід проводити роботу над оптимізацією пам'яті, щоб уникнути проблем із витоком пам'яті. На кожному етапі розробки проекту слід орієнтуватися на максимальний обсяг оперативної пам'яті кінцевого пристрою і постійно стежити за обсягом пам'яті, що використовується, залишаючи запас в 40-50% в ідеальній ситуації. Цей запас необхідний при масштабуванні проекту навіть в умовах повторного використання матеріалів та текстур, тому що в оперативну пам'ять завантажуватимуться текстури запеченого освітлення, вага яких проблематично оцінити заздалегідь та розрахунком на завантаженість системи звичайного користувача.

Оптимізація 3D ігор є багатозадачним і важким процесом, що вимагає безліч людей і динамічно змінювати під час розробки гри. Так само варто розуміти, що хоч і були розглянуті основні проблеми з якими може зіткнутися команди при проведенні даних процесів і вказані найбільш раціональні методи їх вирішення для кожної з проблем, у купі всіх прийнятий напівзахід можуть виникнути проблеми, бо кожен метод вимагає своєї реалізації знаходження балансу може знадобитися час, дуже багато часу, що викличе непомірні витрати на розробку проекту.

Тепер коли були розглянуті всі основні методи оптимізації 3D гри з боку графічних технологій, варто розглянути архітектурні рішення, які дозволять покращити загальну масштабованість системи та її баланс у плані використання рівномірного використання всієї обчислювальної сили системи користувача, задля уникнення помилок та зайвих витоків. Оскільки для розробки було прийнято використовувати ігровий двигун Unity, всі наступні методи будуть більше описані для реалізації саме в цьому середовищі розробки.

3.2 Дослідження Entity Component System

Система сутність-компонент (Entity Component System, ECS, EC) - шаблон проектування, де насамперед визначається ієрархія елементів, з яких складається додаток (сутності), а потім вказуються логіка та дані для кожного з них (компоненти). У деяких інтерпретаціях терміну сутність може бути об'єктом з масивом з нуля і більше компонентів.

EC хороший шаблон для нівелювання проблем множинного успадкування, коли заплутана структура класів може породжувати проблеми по типу діаманту (the diamond problem): клас D, успадкованих від B і C із загальним класом A, може містити конфлікти через різні перевизначені можливостей класами A та C.

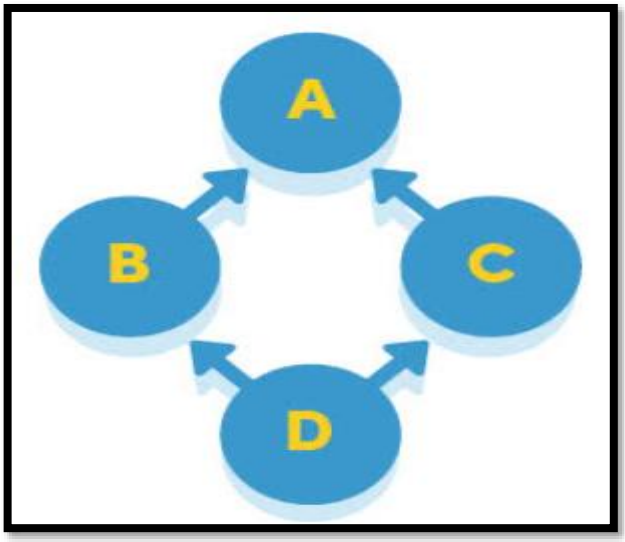

Рисунок 3.4 - the diamond problem

Подібні проблеми часто трапляються при активному використанні успадкування. Якщо розбити завдання та обробники даних на невеликі компоненти, вони можуть приєднуватися до сутностей та перевикористовуватися без множинного успадкування, якого немає в C#, основній мові програмування для Unity.

Давайте розглянемо основні особливості ECS щодо стандартної компонентної моделі Unity. Почнемо за традицією з плюсів:

– підвищена гнучкість та масштабованість (додавання нових, видалення старих систем та компонентів не викликає сильного скрути);
– ефективне використання пам'яті (особливість реалізації, ми можемо перевикористувати об'єктів C#-класів як завгодно на відміну від "MonoBehaviour", де під кожен клас необхідно створити свій компонент, що збільшить витрати на пам'ять);
– простий доступ до об'єктів (вибірка сутностей з певними компонентами виконується ядром ECS без втрати швидкості та алокацій пам'яті - цього аспекту не вистачає стандартної компонентної системи Unity);
– простіший і зрозуміліший інструмент для поділу логіки та даних;
– простіше тестувати (легко відтворювати тестове оточення);
– простіше створювати ігрову логіку сервера без прив'язки її до ігрового двигуна, що зменшує зв'язність і збільшує гнучкість системи.

Тепер варто розглянути основні мінуси цього рішення.

Більше коду. На перший погляд це здається не настільки суттєвим на тлі плюсів, але варто заглибитися у створенні ігрової системи на ECS, як тут же дізнаєшся про те, що стандартні компоненти Unity не підтримуються в ECS рішеннях, які представлені на даний момент часу і тобі потрібно буде писати обгортки для кожної. абстракції або модуля, в якому необхідний “MonoBehaviour”.

Нема офіційного рішення від Unity. Хоча і всі наступні рішення так само не матимуть офіційних інструментів для впровадження в середовище Unity, дана система є винятком, основний концепт є прямим конкурентом і наступним етапом розвитку поточної компонентної системи Unity, цей факт визнають навіть розробники Unity. Що у свою чергу спонукало їх почати створювати свою реалізацію цього шаблону, але вона все ще перебуває на етапі розробки.

На закінчення необхідно сказати, що дана система безумовно буде майбутнім розвитком середовища Unity, але її час ще не настав і створення зараз гри на її основі вимагає детального вивчення реалізацій та методів вирішення існуючих

конфліктів для оптимального використання всіх плюсів технології. Дані дослідження будуть виходити за рамки поточної роботи, тому було прийнято рішення застосувати інші архітектурні методи оптимізації додатків у Unity.

3.3. Дослідження Dependency Injection

Використання залежностей (DJ) - це шаблон проектування та реалізація принципу інверсії елемента управління. Це допомагає спростити та автоматизувати підключення компонентів у складних програмах. Це допомагає досягти ізоляції компонентів, що важливо для модульного тестування.

Наступна цитата добре визначає суть цієї концепції.

“Коли ви йдете та дістаєте речі з холодильника для себе, ви можете створити проблеми. Ти можеш залишити двері відчиненими, ти можеш отримати щось, чого мама чи тато не хочуть, щоб ти давала. Можливо, ви навіть шукаєте щось, чого у нас навіть немає або термін придатності минув.

Те, що ви повинні зробити, це заявити про потребу: «Мені потрібно щось випити за обідом», а потім ми подбаємо про те, щоб у вас було щось, коли ви сядете за стіл.

John Munsch, 28 жовтня 2009 [N]

Преимущества внедрения зависимостей:

- проблеми використання залежностей та вирішення залежностей розділені. Поділ інтересів є гарною практикою для будь-якої системи;
- програмні компоненти підключаються один до одного автоматично, тому ви витратите набагато менше часу на підключення системи та порядок ініціалізації. Наприклад, у системі легко створювати нові об'єкти, які мають зручний доступ до всіх необхідних компонентів логіки;
- це зменшує виникнення проблем із порядком ініціалізації. Внаслідок чого відпаде проблема з постійним прокиданням залежностей у самому

інспекторі та дозволить виключити помилки порожніх об'єктів у першій ініціалізації сцени, оскільки весь процес буде виконано заздалегідь;

– впровадження залежностей зменшує жорстке підключення в додатку, це одна з речей, яка призводить до поломки можливих критичних помилок системи, коли ви вирішуєте змінити код, активи або ієрархію Unity. Системи, які мають сильну пов'язаність, дуже крихкі. Це має велике значення для додатків з тривалою підтримкою і зростанням;
– використання залежностей спрощує переналаштування вашої програми у міру її розвитку. Залежно можна перемикати на інші залежності, що реалізують той самий інтерфейс. Головний компонент реалізації не знає і не дбає про це, поки його залежність якимось чином задоволена;
– використання залежностей спрощує модульне тестування та розробку через тестування. Впроваджені залежності можна легко замінити на фіктивні автономні об'єкти, що забезпечує ізоляцію коду, необхідну для чистого модульного тестування;
– даний механізм використовуватиме централізації або екстерналізації конфігурації вашої системи. Існують інструменти для використання вільного API для налаштування установки в коді. Інакше можна впровадити налаштування залежності через файл конфігурації. Як приклад підключення залежностей програми можна налаштувати з файлу json, який вантажиться з віддаленого сервера. Можна подумати, чи це хороша практика проектування, але є ситуації коли таке рішення має право на життя і є необхідністю. Як, наприклад, віддалені налаштування деяких графічних показників для оптимізації клієнта без заходження в код гри або додавання даного функціоналу користувачеві, який може все поламати.

Цей метод перекидання залежностей має один мінус. Сильне зменшення зв'язків між компонентами. У плані архітектури програми та загального дизайну

гри це звичайно позитивна риса. З точки зору статистичного аналізу програми для нового розробника або якщо старому буде необхідно повернутись до давно розроблених модулів буде складно зрозуміти загальну структуру додаток та відстежити місця вихідного запуску програми разом з усіма точками ініціалізації компонентів. Від чого сильно підвищується загальна складність розуміння структури програми.

Сінглтон є дуже популярною і жахливою звичкою у розробників двигуна Unity, але DJ допомагає уникнути основних проблем, які створює даний патерн, даючи можливість користуватися зручністю синглтонів без проблем. Самі програмні компоненти просто вимагають залежності, їм все одно, чи є ці залежності синглтонами чи ні. Це дозволяє проводити модульне тестування компонентів, що використовують синглтони, що інакше було б важко. Це також дає розробникам деяку гнучкість для подальшого видалення синглетонів із системи у міру розвитку архітектури ігрового програмного забезпечення з мінімальним перериванням (або взагалі без) для класів, які залежали від одинаків.

Один із популярних реалізацій є Zenject - контейнер застосування залежностей з відкритим вихідним кодом, орієнтований на застосування з ігровим двигуном Unity, що забезпечує роботу на більшості платформ, підтримуваних Unity. Варто зазначити, що Zenject можна застосовувати і для програм, розроблених без Unity. Цей контейнер активно підтримується та розвивається. Крім того, Zenject володіє всім необхідним контейнеру DI функціоналом.

Використання залежностей допомагає тим більше, що більше стає додаток. Якщо реалізація впровадження залежностей повністю автоматизовано, то завжди є розумієте, як система буде зашита, неважливо, в якій частині програми ви опинилися. По-друге, все буде підключено послідовно, тому що це автоматизоване рішення, яке використовується і на 100% однаково в усіх напрямках.

3.4. Дослідження використання Model-View-Controller у двигуні Unity

Модель-Представлення-Контролер (MVC) - розділяє програму на три основні компоненти: Модель (CRUD), Представлення (Інтерфейс/Детектування) та Контролер (Рішення/Дія). MVC досить гнучкий і реалізується поверх EC та OOP.

Для ігор або розробки Unity у нас можуть бути різні імена для кожного модуля, але це практично однакові концепції:

Модель. Представляє дані, база даних, елемент чи будь-які класи-оболонки даних.

View. Інтерфейс користувача, будь-які класи, які управляють поданням інтерфейсу користувача і прослуховують вхідні дані гравця.

Контролер. Являє собою менеджер, контролер або будь-який логічний клас, який управляє моделлю.

Будь-який розробник що брав участь у рефакторинг проектів від маленьких студій, не раз бачив сценарій з більш ніж 500 рядками, керуючими даними, інтерфейсом користувача і логікою одночасно. Сценарій дуже важко читати, і було б катастрофою, якби мені потрібно було знайти та налагодити певну функціональність у сценарії. Тому дуже важливо організувати проект із меншою залежністю від коду, особливо коли над проектами працює команда. Запозичуючи концепцію шаблону MVC, ми можемо створити щось подібне до MVC і досягти кращої архітектури коду.

Основні причини, через які MVC може бути тут корисним:

Менша залежність. Найбільша перевага відокремлення уявлення від контролера полягає у зменшенні залежності та ймовірності відсутності помилок посилань, кожен модуль системи матиме можливість автономної роботи. Наприклад, порушення чогось з інтерфейсу користувача не вплине на логіку основного контролера.

Легше створювати інтеграційні тести. Інтерфейс користувача відділений від контролера, ми можемо створити інтеграційний тест з макетом контролера, що не спотворює інтерфейс користувача і введення гравця.

Кілька сторінок інтерфейсу користувача. Іноді може потребуватися створити кілька інтерфейсів для однієї функції в грі, наприклад, два різних екрани збору карт з меню і з гри. Ми можемо додавати або оновлювати певний інтерфейс користувача, не заважаючи контролеру, коли як виклик їх обробки буде викликатися загальним інтерфесом.

Менший конфлікт злиття. Як приклад, поки технічний художник оновлює сценарій інтерфейсу користувача і призначає посилання на нього в Unity, розробник може незалежно працювати зі сценаріями контролера.

Найкраща читабельність. У додатку буде легше знайти код кожного модуля за його функціональністю.

Тепер давайте детальніше розглянемо кожен компонент даного патерну в реаліях ігрового двигуна Unity.

Почнемо с модуля даних. У розробці ігор зазвичай використовуються два типи даних:

Попередньо встановлені значення — або з бази даних, або з локального CSV-файлу, або з ScriptableObject файлу ресурсів Unity. Наприклад, HP гравця на кожному рівні, базові пошкодження зброї, навички супротивника і т. д.

Внутрішньоігрові цінності - дані про прогрес гравця, що зберігаються в PlayerPrefs або будь-якому іншому рішенні для зберігання. Наприклад, рівень гравця, рівень зброї, статус ворога тощо. Іноді використовується з встановленими значеннями, але не обов'язково.

Незалежно від того, як зберігаються необроблені дані, нам завжди будуть потрібні класи-оболонки для легкого доступу, це стосується модуля Model.

Щоб ідентифікувати різні типи даних, ми використовуємо різні імена:

попередні дані – *Data;

внутрішньоігрові дані - *Item.

Наприклад, модель даних необхідна для розрахунку таблиці лідерів може бути:

```
public class ScoreboardData : ScriptableObject
{
    [SerializeField] private GameObject redPlayerPrefab;
    [SerializeField] private GameObject bluePlayerPrefab;

    public int CoinsForKill = 100;
    public int CoinsForDeath = 0;

    public GameObject RedPlayerPrefab => redPlayerPrefab;
    public GameObject BluePlayerPrefab => bluePlayerPrefab;
}
```

В даному прикладі таблиця лідерів має заздалегідь готові осередки для кожної команди, в які поміщається гравець і встановлену нагороду за дії в грі. Поточний блок поміщений в ScriptableObject для більш зручної зміни значень редактора.

Наступним на черзі йде View. Це інтерфейс користувача. У даному модулі, залежно від управління зовнішнім виглядом інтерфейсу користувача, він також включає введення гравця, наприклад, натискання кнопки. Основна концепція MVC в Unity - відокремити логіку та елементи інтерфейсу користувача від скрипта контролера. Іншими словами, контролер або майстер-скрипт НЕ ПОВИННІ робити жодних прямих викликів для взаємодії з інтерфейсом користувача. У цьому випадку зворотні дзвінки є нашим рішенням.

Наприклад наведений інтерфейс кнопки для віртуальної реальності:

```
public class ButtonPointerEvent : MonoBehaviour, ILaserPointEventer
    {
        [SerializeField] private Image[] buttonImages;

        [SerializeField] private Material inMaterial;
        [SerializeField] private Material outMaterial;

        public void PointClick() => SetMaterial(outMaterial);

        public void PointIn() => SetMaterial(inMaterial);

        public void PointOut() => SetMaterial(outMaterial);


    private void SetMaterial(Material material)
```

```
    {
        for (int i = 0; i < buttonImages.Length; i++)
            buttonImages[i].material = material;
    }
}
```

У даному прикладі ми можемо спостерігати реалізацію кнопки користувача, яка реалізує інтерфейс ILaserPointEventer який має основні методи зміни кнопки для користувача при різних випадках.

```
public class InputHandler : MonoBehaviour
{
    ***
    private void PointerIn(object sender, PointerEventArgs e)
    {
        e.target.TryGetComponent<ILaserPointEventer>(out var
laserPointEvent);

        if (laserPointEvent != null)
            laserPointEvent.PointIn();
    }
    ***
}
```

Клас є одним із модулів контролера, який відповідає за зчитування натискань клавіш на руці гравця та відправляє необхідні команди кожному спадкоємцю минулого інтерфейсу. Що б не змінилося в системі, кнопка завжди буде кнопкою виконуючи свій ряд завдань, а обробник натискань завжди оброблятиме завдання і сповіщатиме інтерфейс іншого рівня про завершення свого циклу. У результаті ці два компоненти не можуть перервати роботу іншого і при виході з ладу одного з них другий не постраждає. Це рішення підвищує загальну стабільність системи.

Последним на очереди, но не по значению идет Controller. Нарешті, ми у модулі контролера, де відбувається більшість логіки. У нашій угоді про імена ми називаємо *Manager, якщо сценарій має лише один екземпляр або статичний (наприклад, TeamManager), і *Controller, якщо сценарій має кілька екземплярів (наприклад, EnemyController, по одному для кожного ворога у грі с абстрактною реалізацією батька).

Як завжди розглянемо приклад с реального коду.

```
public class TeamSpawnPositionManager
 {
     private List<TeamShelterStartPosition> _startPositions;

     public TeamSpawnPositionManager()
     {
         _startPositions = new List<TeamShelterStartPosition>();
     }

     public void AddStartPosition(TeamShelterStartPosition
startPosition)
     {
         _startPositions.Add(startPosition);
     }

     public void RemoveStartPosition(TeamShelterStartPosition
startPosition)
     {
         _startPositions.Remove(startPosition);
     }

     public TeamShelterStartPosition GetStartPosition()
     {
         RemoveDeadPoints();
         return GetEmptyStartPosition();
     }

     private TeamShelterStartPosition GetEmptyStartPosition()
     {
         return _startPositions.FirstOrDefault(position =>
position.IsEmpty);
     }

     private void RemoveDeadPoints()
     {
         _startPositions.RemoveAll(t => t == null);
     }
 }
```

Цей приклад являє собою менеджер з додавання стартових позицій гравця та їх отримання для його появи на ігровій сцені.

Тепер ми знаємо як застосовувати патерн MVC для реалізації ігрової архітектури та розуміємо його загальний концепт.

4. РЕАЛІЗАЦІЯ

В результаті було розроблено 3D FPS гра з використанням технології віртуальної реальності. Були застосовані всі необхідні методи оптимізації графіки та архітектури, що дозволило створити стабільну кількість кадрів у динамічній грі та підтримати рівень графіки на прийнятному рівні.

На малюнку 4.1 можна побачити кількість кадрів у реалізованій грі на момент пострілу.

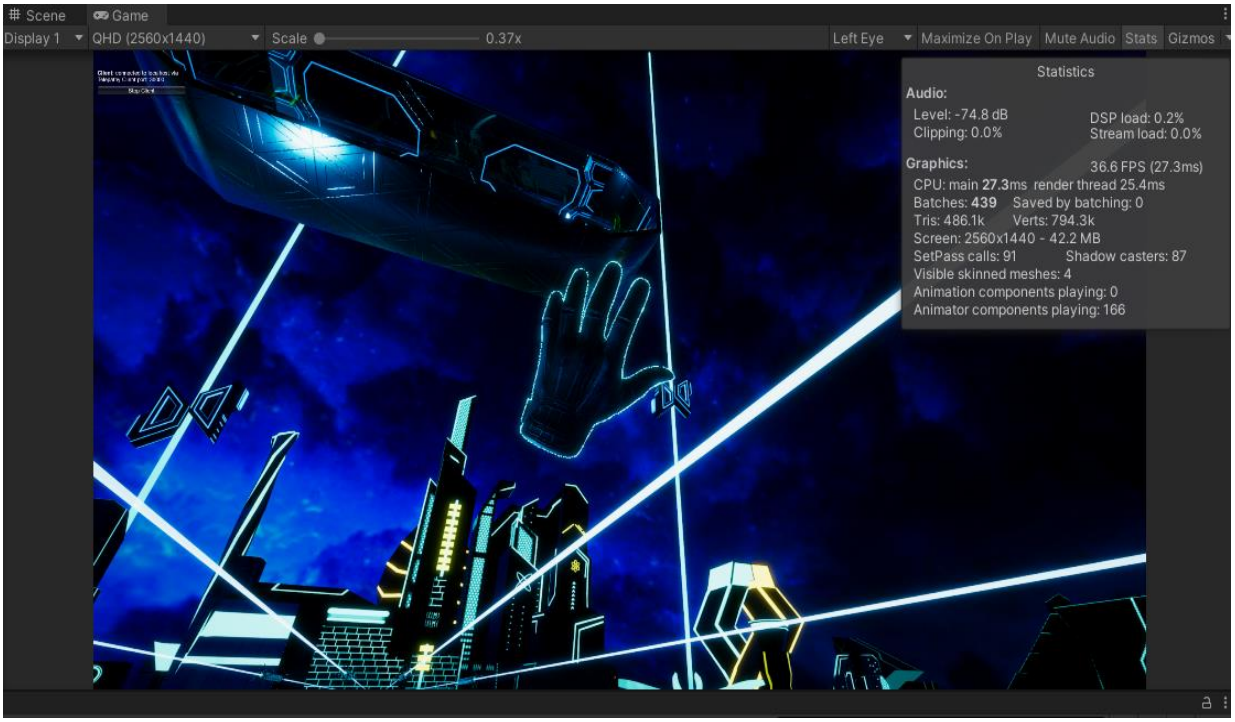

Рисунок 4.1 – зображення гри.

Тестовий стенд складався з процесора i5-8400H, 16 GB RAM, GTX 1050 та laptop model. У момент тестування на тестовій машині було запущено сервер для тестування і сама гра. Витрата пам'яті склала оперативної пам'яті склала близько 4 гігабайти, витрата пам'яті відеокарти 3 гігабайти. Як можна помітити тестовий стенд не підходити під мінімальні вимоги, але здатний видавати в середньому 36 кадрів за секунду, що є чудовим результатом для цієї конфігурації.

У ролі основного двигуна гри був обраний Unity, тому що він є найкращим середовищем для розробки гри в короткий термін невеликою командою розробників або самотужки. В основного інструмента для залежностей був обраний Zenject, який дозволив зменшити зв'язність коду і полегшив перехід гравців на нову сцену, так само за допомогою даного інструменту були прибрані критичні помилки з ініціалізацією гравця при створенні його моделі.

Сеттинг гри був обраний Фантастика, що дозволило мінімізувати кількість полігонів на моделях у сцені, зберігаючи прийнятний рівень графіки та стиль гри.

Основним патерном для побудови архітектури програми був обраний MVC, він дозволив грамотно розділити гру на модулі, кожен з яких виконує своє завдання і не впливає на роботу інших компонентів системи. Також це рішення дозволить масштабувати гру надалі.

Мережеву частину гри було реалізовано за допомогою умовно безкоштовного мережевого рішення для Unity Mirror. В результаті чого користувачі можуть грати один з одним у різних куточках світу. Логіка сервера була винесена в окремі ігрові модулі, а сама гра побудована на принципі клієнт-сервер, що дозволило прибрати накладні витрати з боку клієнта та збільшити загальну продуктивність та безпеку гри.

ВИСНОВКИ

В рамках виконання класифікаційної магістерської роботі було досліджено та проаналізовано методи оптимізації ігрової 3D графіки, які були використані при розробці FPS гри с використанням технології BP, та можливості використовувати ці методи в цілому.

Було розглянуто та проаналізовано предметну область. В результаті аналізу були визначені та проаналізовані основні аналоги двигунів Unity та Unreal Engine.

Були дослідженні основні методи та технології, які потрібні для розробки та оптимізації 3D гри, а саме:

- основні графічні аспекти, на які варто звертати увагу при розробці гри;
- практичні рекомендації щодо реалізації ігрової графіки для ресурсномістких ігор;
- основні методи вирішення проблем продуктивності при використанні тих чи інших графічних рішень;
- основні архітектурні рішення для створення продуктивної та якісної гри, з можливістю розширення в майбутньому за мінімальних ризиків та ресурсних витрат.

В останніх розділах був наведений аналіз обраних рішень та їх конкурентів, а також реальні приклади їх використання.

В результаті проведеної роботи було проаналізовано безліч літератури та проведено її аналіз. Було вирішено, що саме необхідно для оптимізації 3D FPS гри та її подальшої розробки та підтримки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ПОСИЛАНЬ

1. ОПТИМИЗАЦИЯ ИГР В UNITY 5 – ДМК Пресс, 2016. Крис Дикинсон. – 306 с

2. Unity URL: https://docs.unity3d.com/Manual/JobSystem.html

3. Game Architecture and Design - Andrew Rollings, Dave Morris. 2015. – 1040 с.

4. Github URL: https://github.com/modesttree/Zenject (дата звернення: 03.05.2022)

5. Unity в действии. Мультиплатформенная разработка на C#. Джозеф Хокинг, 2019. – 352с.

6. Sanjay Madhav Game Programming Algorithms and Techniques: A Platform-Agnostic Approach – Crawfordsville, Indiana: R.R Donnelay, 2013. – 350 с.

7. Artificial Intelligence for Games. Millington Ian, Funge John. 2009. – 896с.

8. Game Engine Architecture 2nd Edition. Jason Gregory. 2014 – 1052с.

9. Wikipedia URL: https://en.wikipedia.org/wiki/Dependency_injection (дата звернення: 11.05.2022)

10. URL: https://www.tutorialsteacher.com/mvc (дата звернення: 07.05.2022)

11. Матвєєв, Д., & Лановий, О. (2020). ПРОБЛЕМИ ОПТИМІЗАЦІЇ ГРАФІКИ ПІД ПРИСТРОЇ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ. ΛΌГOΣ. ОНЛАЙН. вилучено із https://ojs.ukrlogos.in.ua/index.php/2663-4139/article/view/5033

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ ЗА НАУКОВИМИ НАПРЯМАМИ НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

11. Матвєєв, Д., & Лановий, О. (2020). ПРОБЛЕМИ ОПТИМІЗАЦІЇ ГРАФІКИ ПІД ПРИСТРОЇ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ. ΛΌГOΣ. ОНЛАЙН. вилучено із https://ojs.ukrlogos.in.ua/index.php/2663-4139/article/view/5033