НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.9:004.77

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету	Завідувач кафедри
інформаційних технологій	комп'ютерних наук
Глазунова О.Г., д.п.н., професор	Голуб Б.Л., к.т.н., доцент
____ _____ 2023 р.	**____ _____ 2023 р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему **«Методи оптимізації ігрового ПЗ засобами Unity»**

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Програмне забезпечення інформаційних систем»**

Орієнтація освітньої програми **освітньо-професійна**

Гарант освітньої програми

к.т.н., доцент _____ Голуб Белла Львівна
(підпис)

Керівник магістерської кваліфікаційної роботи

к.т.н., доцент _____ Ткаченко Олексій Миколайович
(підпис)

Виконав _____ Глуховський Максим Костянтинович
(підпис)

КИЇВ-2023

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) Інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

____к.т.н., доцен_____ _____ Голуб Б. Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)

"____" _____ 2023 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ
Глуховському Максиму Костянтиновичу

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Програмне забезпечення інформаційних систем»**

Орієнтація освітньої програми **освітньо-професійна**

Тема магістерської кваліфікаційної роботи **«Методи оптимізації ігрового ПЗ засобами Unity»**

затверджена наказом ректора НУБіП України від **"30" грудня 2022р. №1939 – "С"**

Термін подання завершеної роботи на кафедру **"03" листопада 2023 р.**

Вихідні дані до магістерської кваліфікаційної роботи

Аналіз сучасного ринку розробки відеоігор, огляд основних принципів оптимізації відеоігрового програмного забезпечення, що дозволить проводити максимально продуктивні оптимізації відеоігор.

Перелік питань, що підлягають дослідженню:

1. Дослідження інструментів які використовуються розробниками відеоігор для аналізу та пошуку проблем з оптимізацією.
2. Дослідження актуальних методів які використовуються для оптимізації графіки, фізики та кодової бази відеоігор на рушію Unity.
3. Використання методів оптимізації на ігровому проекті та порівняння отриманих результатів оптимізації.

Дата видачі завдання "_____"_____20___ р.

Керівник магістерської кваліфікаційної роботи _____ Ткаченко О.М.
(підпис)

Завдання прийняв до виконання _____ Глуховський М.К.
(підпис)

РЕФЕРАТ

Кваліфікаційна магістерська робота містить: 71 с., 50 рис., 1 таблиця, 12 джерел, 1 додаток. В роботі представлено 3 розділи.

Об'єкт дослідження - це ігрове програмне забезпечення, створене на платформі Unity.

Предмет дослідження – методи і програмні засоби оптимізації графіки в при розробці відеоігрового ПЗ.

Методи дослідження: загальнонаукові методи, методи динамічної оптимізації, методи оптимізації завантаження графіки, методи оптимізації освітлення, методи оптимізації створення об’єктів.

Мета дослідження полягає у визначенні і розробці ефективних методів оптимізації ігрового ПЗ на платформі Unity з метою покращення продуктивності і якості ігрового досвіду.

Результатом дослідження є методи які можна ефективно впроваджувати в відеоігрові проекти за для отримання хороших показників оптимізації.

ABSTRACT

A Master's qualification thesis contains: 71 pages, 50 figures, 1 table, 12 sources, 1 appendix. The thesis is divided into 3 sections.

The object of the research is gaming software created on the Unity platform.

The subject of the research is methods and software tools for optimizing graphics in the development of video game software.

Research methods: general scientific methods, methods of dynamic optimization, methods for optimizing graphics loading, methods for optimizing lighting, methods for optimizing object creation.

The aim of the research is to identify and develop effective methods for optimizing gaming software on the Unity platform to improve performance and the quality of the gaming experience.

The research results include methods that can be effectively implemented in video game projects to achieve good optimization results.

Зміст

Перелік умовних позначень 6

Вступ 8

Розділ 1: Теоретичні аспекти оптимізації ігрового ПЗ 12

1.1 Огляд ігрової індустрії та її важливість 12

1.2 Основні принципи оптимізації ігрового ПЗ 16

Розділ 2: Дослідження інструментів і технологій Unity для оптимізації 20

2.1 Unity як платформа для розробки ігор 20

2.2 Типи проблем продуктивності 22

2.3 Інструменти Unity для аналізу продуктивності 24

2.4 Інструменти Unity для оптимізації графіки 30

2.5 Інструменти Unity для оптимізації фізики 38

2.6 Інструменти Unity для оптимізації коду 46

Розділ 3: Практичні аспекти оптимізації ігрового ПЗ в Unity 57

3.1 Аналіз та вибір ігрового проекту для оптимізації 57

3.2 Аналіз продуктивності і виявлення проблем 58

3.3 Реалізація оптимізаційних заходів 59

3.4 Аналіз результатів та оцінка досягнутих покращень 64

Висновоки 67

Список літератури 71

Додаток А 73

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Unity – ігровий рушій

Game Object – об’єкт, ігрова одиниця на сцені.

Сцена – безкінечний простір в якому будується відеогра.

Префаб – об’єкт в якому заздалегідь зібрано певний набір вкладених об’єктів і компонентів, вони можуть дублюватися на сцені, але коли потрібно щось змінити у всіх префабах, що вже використовуються достатньо змінити лише головний, а всі інші автоматично отримають оновлення. Прикладом може слугувати передавання об’єкта класу за посиланням, зміни в одному місці змінять увесь файл.

Компонент – скрипт з візуальною оболонкою який виконує конкретний набір функцій і має візуальні елементи керування для зручного управління ним.

Скрипт – C# клас, який виконує певний набір функцій, зазвичай вони пишуться розробником відеогри.

ПК – персональній комп’ютер.

ПЗ – програмне забезпечення.

ВВП – внутрішній валовий продукт.

CDPR – CD Project Red, відеоігрова студія.

FPS – (анг . Frame per second) частота кадрів в секунду.

RTS – (англ. Real-time strategy) – стратегія в реальному часі.

Frametime – (англ. Frame time) – час кадру.

LOD - (англ. Level of Detail) – характеризує деталізацію 3D-моделі, підхід в оптимізації коли модель при віддаленні від камери підміняється менш деталізованою моделлю, але через велику відстань гравець цього не помічає.

Batching (баччинг) – об’єднання графічних об’єктів у батчі.

Батч – пакет об’єктів що відправляються на відмалювання.

Draw Call – виклик для відмалювання кадру.

Колізія – перетин двох фізичних об’єктів.

GPU – графічний процесор.

RPG – жанр відеоігор, де основна частина ігрового процесу полягає в управлінні персонажем чи групою персонажів.

рис. – рисунок.

API – (англ. application programming interface) набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

Дерево квадрантів - (також квадродерево, 4-дерево, англ. quadtree) — дерево, в якому у кожного внутрішнього вузла рівно чотири нащадки.

Колайдер – межі фізичного об’єкта.

GC – збірник сміття в C#

API - набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення

ВСТУП

Визначення актуальності теми.

Актуальність відеоігрової індустрії в сучасному світі знаходиться на високому рівні. Відеоігри мають важливе значення як для економіки країни так і рухають розвиток електронного обладнання та розробників програмного забезпечення, що проявляється у співпраці розробників рушіїв та компаній по розробці відеокарт та процесорів.

Розглянемо більш конкретні приклади актуальності теми оптимізації відеоігор. Для гравців покращення продуктивності відеоігор дозволяє в повній мірі поринути в проект, насолодитися створеним віртуальним світом, яким бачили його розробники, без лагів і провисань.

У світі де ігрова індустрія стає все більш конкурентною, добре оптимізовані ігри привертають більше уваги гравців і завойовують ринок. Гравці з більшим ентузіазмом купують проекти які мають гарну оптимізацію та високу якість графіки та геймплею. Нажаль останнім часом прослідкується тенденція на погану оптимізацію великих ігор, що дуже шкодить репутації та фінансовому становищу студій розробників, які спішать випустити проект в поставлені терміни, зрештою з часом студії допрацьовують проект, але наслідки для студії вже отримано і в кращому випадку репутаційні. Одним із таких гучних скандалів був із компанією CD Project Red з проектом Cyberpunk 2077, що компанії довелося зробити повне повернення грошей гравцям які купили або перед замовили гру через погану оптимізацію і продуктивність. У своєму фінансовому звіті CDPR визнає, що витратила на всі повернення Cyberpunk 2077 близько 51 млн. доларів.

Оптимізація дозволяє зменшити споживання ресурсів пристрою таких як енергія та обчислювальна потужність. Це особливо важливо на мобільних

пристроях та ігрових консолях, де є досить невеликі потужності в порівнянні персональними комп’ютерами.

Швидкі та оптимізовані ігри можуть більш успішно використовувати нові технології такі як віртуальна реальність та штучний інтелект, що потребують досить високу обчислювальну потужність для своєї роботи.

Враховуючи вище описані фактори оптимізація ігор залишається досить актуальною, та навіть набирає оберти у цьому, бо жоден ігровий проект, який би він цікавий не був без хорошої оптимізації не знайде великої полярності у гравців і не принесе розробникам популярності і прибутку.

Об'єкт та предмет дослідження. Об'єкт дослідження - це ігрове програмне забезпечення, створене на платформі Unity.

Предмет дослідження - це методи та програмні засоби оптимізації продуктивності коду відеогри, роботи фізичного рушію та графічного модуля ігрового ПЗ в Unity.

Мета дослідження. Мета дослідження полягає у визначенні і розробці ефективних методів оптимізації ігрового ПЗ на платформі Unity з метою покращення продуктивності і якості ігрового досвіду.

Зміст поставлених завдань.

1. Аналіз існуючих проблем оптимізації ігрового ПЗ в Unity.
2. Вивчення методів та інструментів, доступних в Unity для аналізу і оптимізації.
3. Розробка стратегій оптимізації для різних аспектів гри, включаючи графіку, фізику та код.

4. Проведення експериментів для визначення оптимальних рішень для оптимізації.
5. Оцінка результатів та формулювання рекомендацій для розробників ігор.

Методи дослідження. У цьому дослідженні будуть використовуватися такі методи:

1. Аналіз літературних джерел для збирання інформації про сучасні методи оптимізації ігрового ПЗ та їх ефективність.
2. Для пошуку проблем оптимізації використовуються засоби Unity такі як Unity Profiler, Frame Debugger, Unity Memory Profiler та Unity Profile Analyzer
3. Для проведення оптимізації графіки використовується метод LOD та метод Occlusion Culling.
4. Для оптимізації завантаження графіки використовується метод Batching.
5. Для оптимізації створення об’єктів використовується Object Pooling.

Наукова новизна. Підчас виконання роботи було запропоновано методи оптимізації графіки які значно покращують продуктивність відеогри, а саме параметри FPS та кількість пакетів відмалювання. Також було запропоновано використання пулів для перевикористання ігрових об’єктів.

Апробація результатів дослідження провадилася шляхом публікації тез на «XIV Міжнародній науково-практичній конференції молодих вчених» тема якої «Інформаційні технології: економіка, техніка, освіта» 26-27 жовтня 2023 р.

Структура роботи. Кваліфікаційна магістерська робота містить: 71 с., 50 рис., 1 таблиця, 12 джерел. В роботі представлено 3 розділи.

Перший розділ є більш узагальнюючим в якому оглянуто загальну важливість ігрової індустрії та основні принципи та метрики якими керуються розробники при оптимізації відеоігор.

В другому розділі проводиться дослідження інструментів та технологій за допомогою яких розробники проводять оптимізацію відеоігор та їхню ефективність.

У третьому розділі виконується практичне застосування та збір даних для порівняння підходів оптимізації відеоігор.

РОЗДІЛ 1: ТЕОРЕТИЧНІ АСПЕКТИ ОПТИМІЗАЦІЇ ІГРОВОГО ПЗ

1.1 Огляд ігрової індустрії та її важливість

В наш час ігрова індустрія займає одну з важливих позицій у світовій економіці, а її прибутки вираховуються в мільярдами доларів. За масштабами вона починає випереджати своїх найближчих конкурентів із сфери розваг таких, як музична індустрія, кіноіндустрія та шоу-бізнес. Ігрова індустрія постійно розробляє нові підходи і інструменти для покращення та прискорення розробки відеоігор, а також постійно кидає виклик розробникам електроніки спонукаючи їх вдосконалювати свої чіпи та технології задаючи нові стандарти.

Ігрова індустрія - це сектор економіки в якому пов’язані виробництво просування і продаж відеоігор які в свою чергу складаються з великої кількості задіяних спеціальностей, а це в свою чергу – робочі місця. З її розвитком та розвитком комп’ютерної техніки ростуть і її можливості, а з появою мобільних телефонів з’явилися і мобільні ігри, що в свою чергу привело до масовості відеоігор. Поява ігрових консолей найпопулярнішими з яких є X-Box та Sony PlayStation додала ринку ще більшу популярність завдяки різноманіттю способів грати у відеоігри. Сучасні портативні комп’ютери багато в чому завдячують саме відеоіграм.

У сучасному світі відеоігри стали не просто дитячою забавкою, справжнім глобальним ринком в якому задіяні мільйони людей, які їх створюються, обслужують і поживають та створюють контент навколо них. Саме ком’юніті відеоігор не обмежене лише дитячою аудиторією, а зібрало навколо себе людей різного віку і статі. Це доводять різноманітні дослідження аудиторій гравців (рис. 1). Це і не дивно тому, що різноманіття відеоігор та їх жанрів зацікавить будь-яку людину.

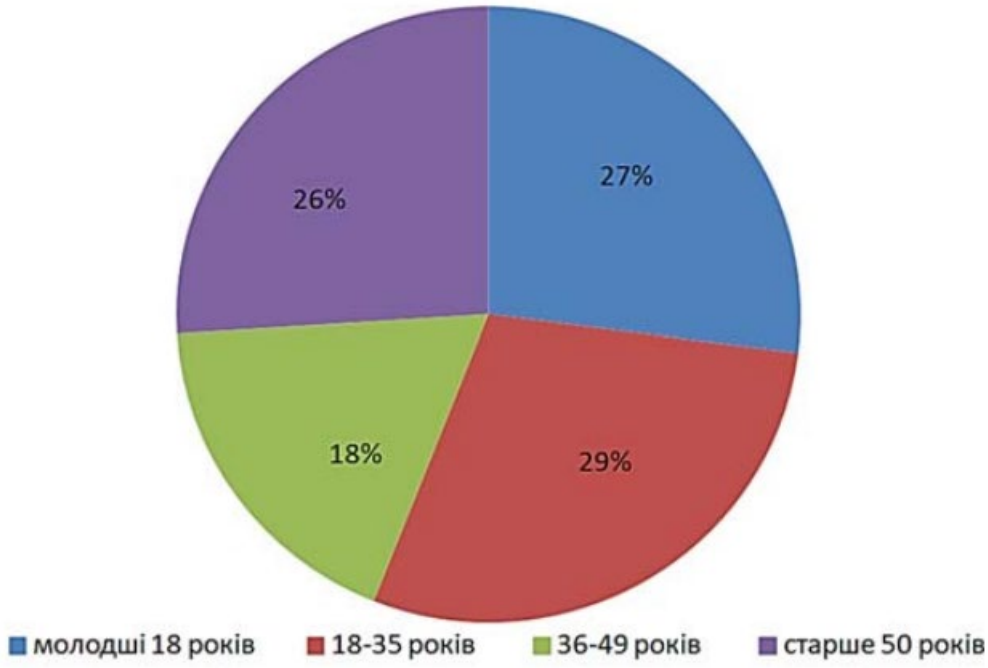

Рис. 1 Розподіл гравців за віковою категорією [1]

На початковому етапі зародження індустрії собівартість ігор була не значною загалом через не великі розміри команд, а також простіші самі відеоігри в плані графіки в порівнянні з сучасними, що в свою чергу приносило досить солідні прибутки. З розвитком технологій зросли і витрати, адже потрібен був більший штат, а проекти ставали більш масштабніші і глибше пропрацьовані.

Дохід від відеоігор з кожним роком зростає, що пов’язано із зростанням популярності у світі (рис. 2). Розробники вкладають у розробку проектів мільйони доларів що в свою чергу стимулює на пошук нових способів покращення графіки, оптимізації проектів, підходів для створення анімацій та інших частин розробки, а значні кошти вкладаються в ринок реклами, щоб привернути більше уваги до своїх проектів.

За приблизними підрахунками у світі понад 2,5 млрд геймерів, що витратили 152,1 млрд доларів на ігри в 2019 році, що показує зростання прибутків ігрових компаній на 9,6% в порівнянні з попереднім роком.

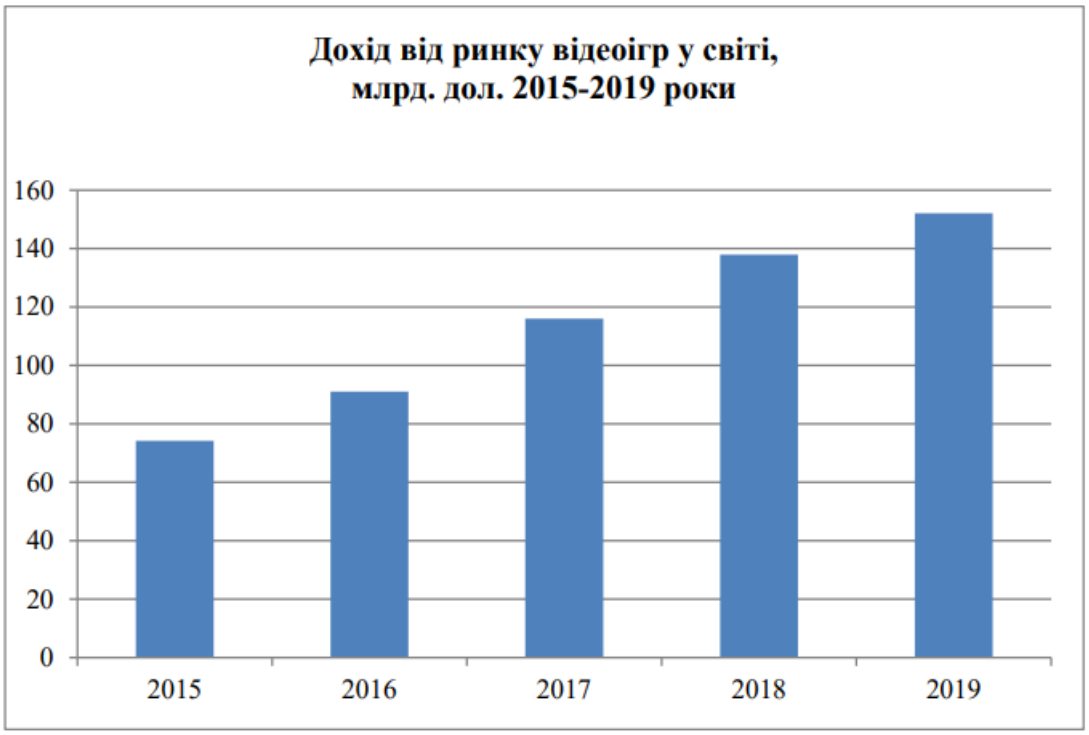

Рис. 2 Дохід від ринку відеоігор у світі, млрд дол., 2015-2019 рр.

Найбільш швидкозростаючим сегментом ринку відеоігор це консоль. Декілька років поспіль зростання консольних відеоігор випереджає зростання мобільних відеоігор. Але ринок мобільних відеоігор все ще залишається найбільшим сегментом у 2019 році і становить 45% світового ринку ігор (рис. 3).

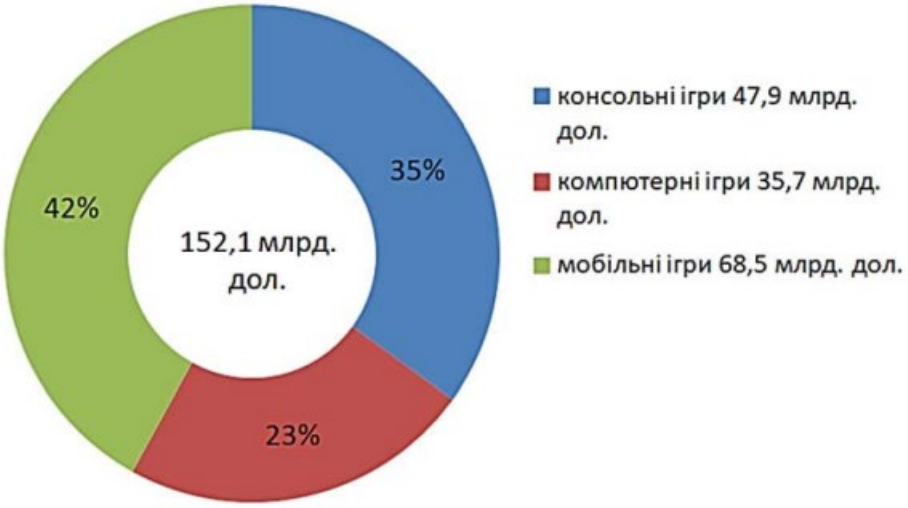

Рис. 3 Дані розподілу доходу світового ігрового ринку за видами, 2019р.

Сучасний ігровий ринок зазнав найменших втрат, а місцями навіть отримав приріст, особливо під час коронавірусної кризи. Так як значна частина населення знаходилася в умовах повного локдауну, багато хто шукав спосіб провести вільний час і наслідком цього кількість гравців виросла на 26-43%, що означає і приріст доходів у ігровій індустрії. [1]

Річний дохід відеоігрового ринку України на початок пандемії у 2019 році становив 203 млн доларів, а у наступному році вже зріс на 13%, тобто на 26 млн доларів, У 2021 році цей показник приросту склав ще 14,4% і на кінець календарного року річний дохід уже складав 260 млн доларів. По прогнозам експертів український ринок буде зростати, що пояснюється стабільним щорічним приростом фінансових показників та розвитком українського геймдеву. [2]

Війна в Україні також специфічним чином вплинула на ситуацію відеоігрової індустрії. З одного боку деякі видавці ігор за результатами 2022 року показують зниження попиту на ігри, але з іншого Україна перестала були лише частиною СНГ регіону, а набула суб’єктності, що добре показується тим що більшість відеоігор та навколо ігрових сервісів отримали українську локалізацію.

Підсумовуючи все вище сказане можу зробити висновок що ігри та ігровий ринок є новим етапом у розвитку економічних та соціальних питань. Він є досить перспективним та важливим для світу та України як новий сектор економіки, що набуває обертів та здатен принести велику частку зростання ВВП та розвитку культури. Великі прибутки а також розвиток технологій дає можливість країні розвиватися, що є дуже важливою складовою для підвищення рівня життя.

1.2 Основні принципи оптимізації ігрового ПЗ

Відеоігри за свою історію пережили стрімкий технологічний розвиток від простої купки пікселів на екрані здалеку нагадувавших м'яч до фотореалістичних відкритих світів які вражають своїми масштабами. Такому розвитку також сприяв потужний розвиток і комп'ютерної інженерії так і розвиток самої ігрової індустрії.

Для того щоб задовольнити зростаючих запитів від гравців та реалізації творчих задумів розробників індустрія постійно потребує все потужніших комп'ютерних систем, що в свою чергу позитивно впливає на розвиток комп'ютерних технологій. Але в зараз навіть найпотужніші комп'ютери не здатні задовольнити всі творчі амбіції розробників та запити гравців. Також історія відеоігор знає багато прикладів фінансового провалу проектів через погану технічну реалізацію, яка не давала гравцям насолодитися процесом гри.

Саме через це перед розробниками стоїть актуальне завдання по оптимізації своїх продуктів, а саме реалізації гри максимально оптимізованою під конкретні комп'ютерні пристрої такі як телефон, консоль чи персональний комп'ютер. Сама оптимізація має таке ж важливе значення як і розробка ігрового процесу, цікавий сюжет чи привабливість графіки та звуків.

Оптимізація - це процес знаходження оптимального рішення поставленої задачі, і при цьому якість рішення повинна відповідати меті, яку потрібно досягти. Основні критерії за якими оптимізують більшість відеоігор це продуктивність та об'єм зайнятого місця в пам'яті пристрою, а все тому що їх простіше подавати в кількісній оцінці.

Частіше за все для оцінювання продуктивності відеоігор використовують частоту кадрів в секунду що називається FPS (англ. Frame per second – частота кадрів в секунду), але іноді помилково припускають що низький FPS означає низьку продуктивність і навпаки, але в реальності в залежності від жанру відеогри вимоги в частоті кадрів можуть відрізнятися. Наприклад для жанру

шутер частота кадрів відіграє важливішу роль ніж для стратегій тобто RTS (англ. Real-time strategy – стратегія в реальному часі). Для комфортної гри мінімальний FPS складає 30 кадрів у секунд, але для комфортної плавності відеогри рекомендованим є 60 FPS. На персональних комп'ютерах частота кадрів обмежується лише самими характеристиками системи, а от на консолях і мобільних пристроях частота кадрів обмежується в 30 і 60 FPS через технічні обмеження пристроїв.

Для оцінки продуктивності додатково до FPS використовують Frametime (англ. Frame time – час кадру) – це час який витрачається на відображення одного кадру. За допомогою Frametime у вигляді часової залежності можна отримати фактичну продуктивність відеогри. На рисунку 4 зображено приклад залежності для FPS і Frametime, зліва видно що час на відтворення кадру досить великий на що вказують пікові ділянки графіку, а в грі це супроводжується затримками і лагами, такі графіки свідчать про досить низьку продуктивність відеогри. З права графіки відповідають добре оптимізованому проекту до якого варто прагнути в розробці відеоігор. Цей графік показує однаковий час на завантаження кадрів та стабільний FPS що в грі відображається плавністю картинки.

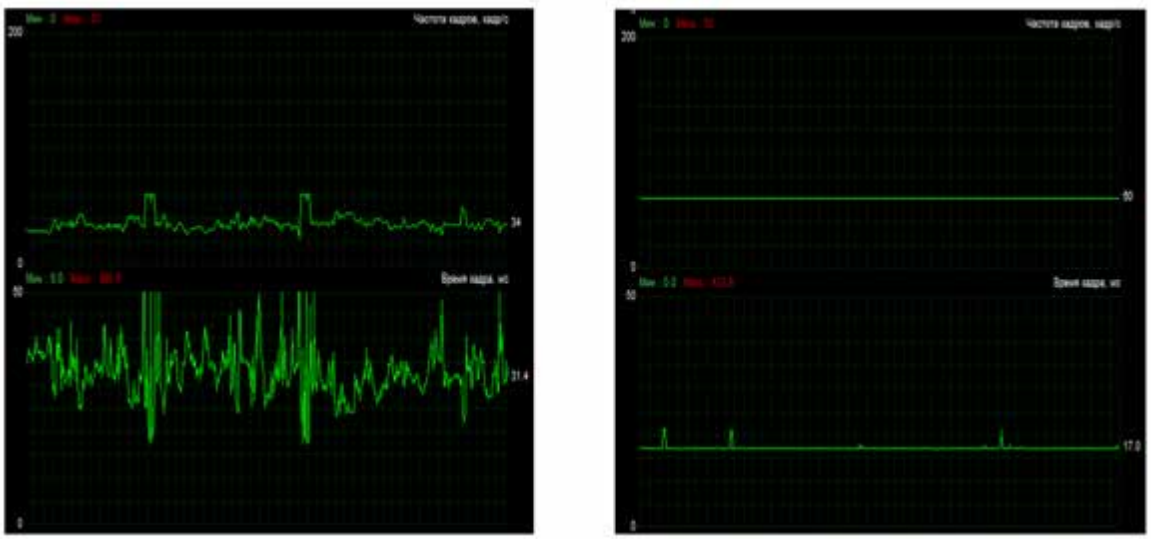

Рис. 4 Графік FPS (зверху) та Frametime (знизу) погано оптимізованого проекту (з ліва) та добре оптимізованого (з права)

Оптимізація ресурсів гри, тобто місця у пам’яті пристрою більш гостро стоїть лише на консолях та мобільних пристроях, зважаючи на те що на ПК часто встановлюються більш об’ємні жорсткі диски. Мобільні пристрої зазвичай мають меншу кількість пам’яті, також ще одним аспектом що Google Play та App Store які є найбільш популярними додатками для поширення відеоігор існує обмеження на розмір файлу відеогри яку можна туди завантажити, але це не поширюється на те що можна до завантажити додаткові файли після встановлення відеогри, що є своєрідною оптимізацією.

Розберемо конкретні аспекти оптимізації відеоігор такі як графіка, фізика, код та ресурси. Для їх оптимізації використовуються різні методи та підходи.

Для оптимізації графіки використовують такий підхід як редукція полігонів – це означає зменшення кількості полігонів у 3D-моделях, що може суттєво покращити продуктивність. Unity має вбудовані інструменти для спрощення геометрії моделей. Ще один підхід це використання лодів (LOD - Level of Detail). Це використання різних рівнів деталізації для 3D-моделей на різних відстанях від камери, що дозволяє зменшити навантаження на графічний процесор. Також можна оптимізувати текстури такими способами як використання стиснених текстур, обмеження розміру текстур і використання асинхронного завантаження, що допоможе зменшити споживання пам’яті та покращити продуктивність. Також можна використати батчінг – об’єднання графічних об’єктів у батчі, що може значно зменшити кількість викликів до графічного процесора.

Оптимізацію фізики можна провести використанням простих геометричних фігур при обрахунках колізій замість складних колізійних об’єктів, що значно зменшить обчислювальне навантаження на фізичний рушій. Також можна зменшити саму їх кількість виключивши ті, що не беруть безпосередньої участі у відеогрі.

Код можна оптимізувати шляхом профілювання та оптимізації викликів функцій. Використавши інструменти для профілювання визначаються функції які забирають найбільше часу і оптимізуються їх робота. Використання пулу об’єктів для уникнення створення та знищення об’єктів на льоту що може привести до великого обсягу сміття. Також можна використати паралельну обробку за допомогою розгалуження коду на паралельну роботу на різних потоках, що значно прискорить обчислення на багатоядерних процесорах.

Також код можна оптимізувати правильною архітектурою коду, а саме використання патернів. Одним із таких варіантів є використання Entity-Component-System (ECS). Його переваги в тому що він дає можливість ефективно оптимізувати відеогру, оскільки системи обробляють дані без надмірних перевірок та переключень стану. Компоненти і системи легко замінюються і розширяються, що робить розробку більш модульною. Дає можливість окремо тестувати компоненти і системи, що полегшує пошук багів. Також ECS дозволяє використовувати паралельність та багатопоточність, що покращує продуктивність на багатоядерних процесорах.

Оптимізувати ресурси можна завантажуючи їх асинхронно, що дозволяє уникнути підвисань під час завантаження великих ресурсів. Використання стиснених для текстур, аудіо та іншого допомагає зменшити їх обсяг та зменшити вимоги до пам’яті. Також важливо правильно керувати пам’яттю та вчасно видаляти непотрібні ресурси, що допомагає уникнути утримання зайвої пам’яті.

Варіанти оптимізації підбираються під кожен проект індивідуально і тут описані лише базові які використовуються. Важливо зрозуміти слабкі місця вашого конкретного проекту і проводити оптимізацію для покращення продуктивності.

РОЗДІЛ 2: ДОСЛІДЖЕННЯ ІНСТРУМЕНТІВ І ТЕХНОЛОГІЙ UNITY ДЛЯ ОПТИМІЗАЦІЇ

2.1 Unity як платформа для розробки ігор

Unity – це ігровий рушій, що дозволяє розробляти відеоігри під різні платформи. При створенні цього рушія автори намагалися вкласти в нього всі інструменти, що потрібні розробникам для комфортної роботи в ньому, і перелік інструментів постійно зростає. Рушій став універсальним рішенням як для малих команд або інді розробників так і для великих компаній.

За роки існування рушію навколо нього зібралось велике ком'юніті, що в свою чергу створило багато онлайн-ресурсів та навчальних матеріалів, що значно знижують поріг входу для нових користувачів, а сам інтерфейс програми розроблений таким чином щоб бути інтуїтивно зрозумілим. Ще одною перевагою рушія є об'єктно-орієнтована мова програмування C#, яка теж має багато навчальних матеріалів та є мовою високого рівня, що облегшує написання коду.

Кросплатформеність рушію Unity дозволяє розробляти ігрові проекти на різні платформи, такі як Android, iOS, Windows, macOS, консолі та веб, що в свою чергу дозволяє охопити дуже широку аудиторію.

Unity має потужні вбудовані засоби для роботи з 2D і 3D графікою, а також зручний інтерфейс для створення та редагування анімацій. Окрім графіки в рушію підтримується обробка аудіо, що дозволяє реалізовувати 3D звукову обробку.

Фізичне моделювання в іграх відіграє важливу роль, тому в Unity також має вбудовану систему фізики, яка дозволяє моделювати реалістичні фізичні ефекти, колізії та симуляції поведінки фізичних об'єктів.

Unity має не аби які засоби для розширення інструменту для розробки. Якщо розробнику не вистачає базових інструментів або йому потрібні

специфічні інструменти під його задачі, розробник може сам розробити додаткові інструменти такими як йому потрібно. Часто такі інструменти створюють щоб дати можливість гейм дизайнерам, які не дотичні до розробки налаштовувати параметри проекту, або спростити для себе виконання рутинних завдань. Маючи велике ком’юніті розробників дуже важливо мати місце де розробники можуть ділитися своїми власними розробками і це місце називається Unity Asset Store. Тут розробники можуть поширювати свої власні розробки як безкоштовно так і за гроші.

Рушій Unity дуже універсальна річ, яка постійно вдосконалюється і намагається запропонувати щось своє для вирішення завдань які виникають у розробників під час створення відеогри, а якщо чогось не вистачає розробник може створити це сам.

2.2 Типи проблем продуктивності

В Unity відеоігри можна оптимізувати різними способами в залежності від характеру проблем, а для їх коректного виявлення потрібно знати що шукати. Розглянемо типи проблем з продуктивністю залежно від їх характеру.

Спайк (з англ. Spike шип) - це раптове падіння частоти кадрів в грі. Його видно коли гра раптово завмирає на декілька секунд. Це погано впливає на сприйняття відеогри, втрачається плавність і гра перетворюється на по кадрову анімацію. Також це може вплинути на швидкість реакції гравця та може змусити його зробити помилку, якої б він міг уникнути при стабільному ігровому процесі.

Такі пікові навантаження можна побачити на графіку в профайлері, приклад такого графіку зображено на рис. 5.

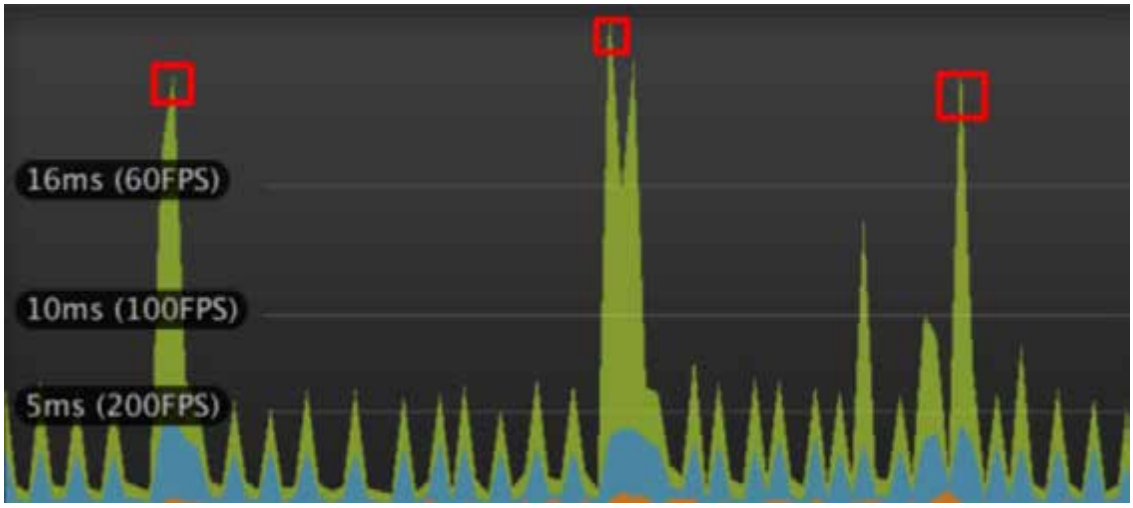

Рис. 5 Раптове падіння частоти кадрів

Ці скачки зазвичай викликані складними розрахунками системи що проходять під час розрахунку одного кадру. Бажано оптимізувати такі місця щоб графік виглядав більш однорідним.

Ще однією причиною таких падінь може бути збирання сміття в Unity. Коли GC починає видаляти непотрібні об’єкти вони і виникають, особливо якщо дуже багато зібралося в пам’яті сміття.

Багато сміття генерується коли ми постійно створюємо та знищуємо об’єкти. Для його зменшення використовують різні техніки наприклад кешування об’єктів для перевикористання або патерн Object Pooling.

Однією з проблем відеогри може бути надмірне споживання пам’яті. Пам’ять RAM – це пам’ять на яку звантажується відеогра під час роботи, саме там зберігається все необхідне для роботи гри. VRAM – це пам’ять яка знаходиться на відеокарті та призначена для відтворення графічних об’єктів, тут зберігаються текстури та моделі які відтворюються відеокартою. При нестачі пам’яті може виникнути збій та гра може зависнути або закритися з помилкою.

2.3 Інструменти Unity для аналізу продуктивності

Досить важливою річчю в розробці є оптимізація та управління ресурсами відеогри. Unity має різні інструменти що спрощують розробникам роботу по оптимізації проекту. До цих інструментів відноситься:

- Profiler;
- Unity Frame Debugger;
- Unity Profile Analyzer;
- Unity Memory Profiler.

Unity Profiler – це інструмент, який можна використовувати для отримання інформації про продуктивність відеогри (рис. 6). Його можна підключити до пристроїв підключених до комп'ютера або пристроїв, у вашій мережі, щоб перевірити, як програма працює на запланованій платформі випуску. Можна також запустити його в редакторі, щоб отримати огляд розподілу ресурсів під час розробки відеогри.

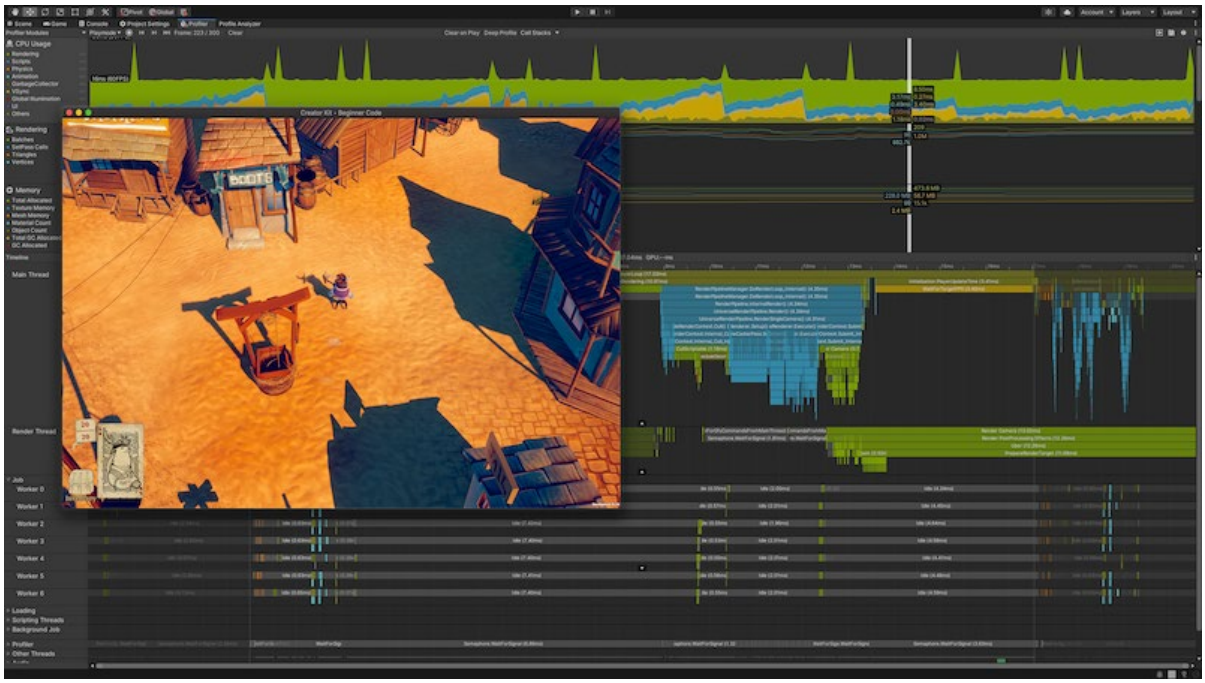

Рис. 6 Загальний вигляд вікна Unity Profiler

Profiler збирає та відображає дані про продуктивність відеогри в таких областях, як:

- Використання центрального процесора окремо кожним компонентом рушія;
- Витрати на візуалізацію;
- Використання графічного процесора GPU на різних етапах робото графічного конвеєра;
- Використання пам’яті;
- Затрати на відтворення звукових ефектів;
- Використання фізичного рушія.

Це корисний інструмент для визначення областей для покращення продуктивності відеогри і без його використання не обходиться жоден проект.

Профайлер має багато різних методів для виявлення проблем в продуктивності відеоігор. Одним із таких є режим відображення ієрархії в якому разом із назвою функції класу відображаються додаткові параметри, які надають додаткову інформацію про роботу конкретного методу. Також там можна побачити кількість викликів методу, скільки сміття генерує та час який витрачається на функцію. Також якщо ця функція викликала іншу функцію то поруч з назвою буде стрілочка розгорнувши яку можна переглянути ці функції (рис. 7)

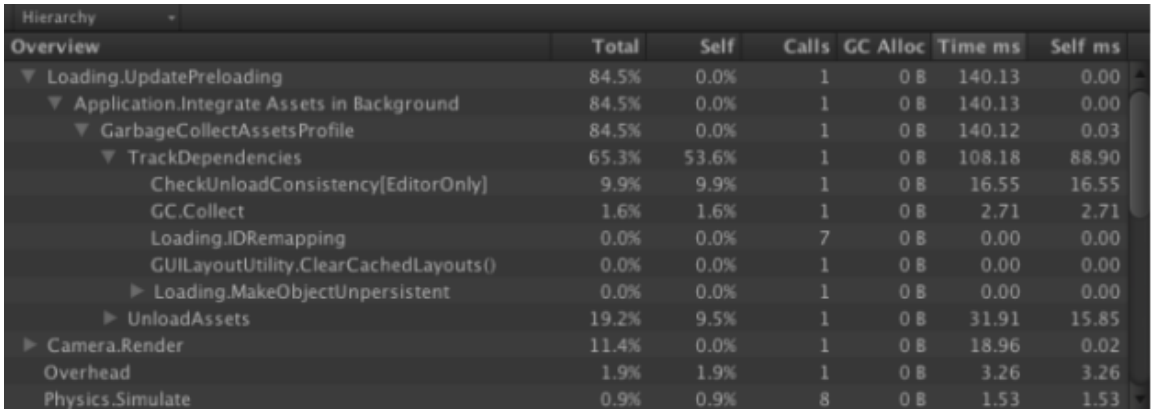

Рис. 7 Режим ієрархії в профайлері

Frame Debugger — це зручний інструмент, який дозволяє зупинити відтворення запущеної гри на певному кадрі, щоб переглянути окремі виклики відображення, які використовуються для візуалізації цього кадру. Він також дозволяє переглядати кадри один за одним, щоб можна було більш детально побачити, як побудована сцена. Вікно Frame Debugger зображено на рисунку 8.

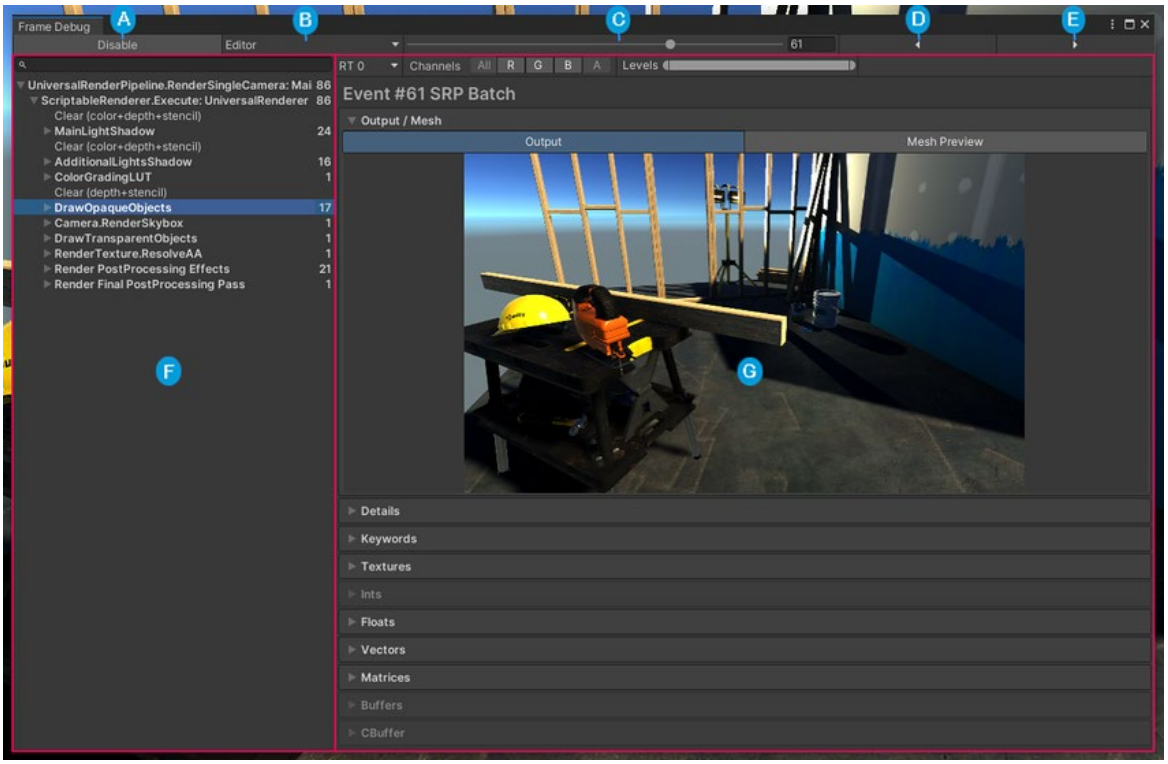

Рис. 8 Загальний вигляд вікна Frame Debugger

Вікно Frame Debugger складається з таких частин:

- A: увімкнути або вимкнути Frame Debugger;
- B: селектор цілі – визначає процес, до якого потрібно приєднати Frame Debugger. За замовчуванням це редактор Unity, але ви можете використовувати його, щоб приєднати Frame Debugger до вбудованих програм;
- C: переміщувач подій - повзунок, який можна використовувати для лінійного переміщення подій візуалізації в поточному кадрі.

- D: попередня подія - вибирає подію, яка передувала поточній вибраній;
- E: наступна подія - вибирає подію після поточної вибраної.
- F: ієрархія подій - перераховує послідовність подій візуалізації, які складають кадр;
- G: панель інформації про подію : відображає інформацію про подію.

Unity Profile Analyzer – це інструмент агрегує та візуалізує дані кадрів і маркерів із набору кадрів Unity Profiler, щоб допомогти вам зрозуміти їхню поведінку в багатьох кадрах, доповнюючи аналіз одного кадру, який уже доступний у Unity Profiler. Це корисно, коли важливо мати більш широкий погляд на роботу проекту

Він аналізує дані CPU фрейму і маркера, які витягуються з активного набору кадрів, завантажених в даний момент в Unity Profiler або завантажених з раніше збереженої сесії. Ці дані підсумовуються і відображаються за допомогою боксових графіків гістограм, та інших.

Приклад роботи Unity Profile Analyzer зображений на рис. 9

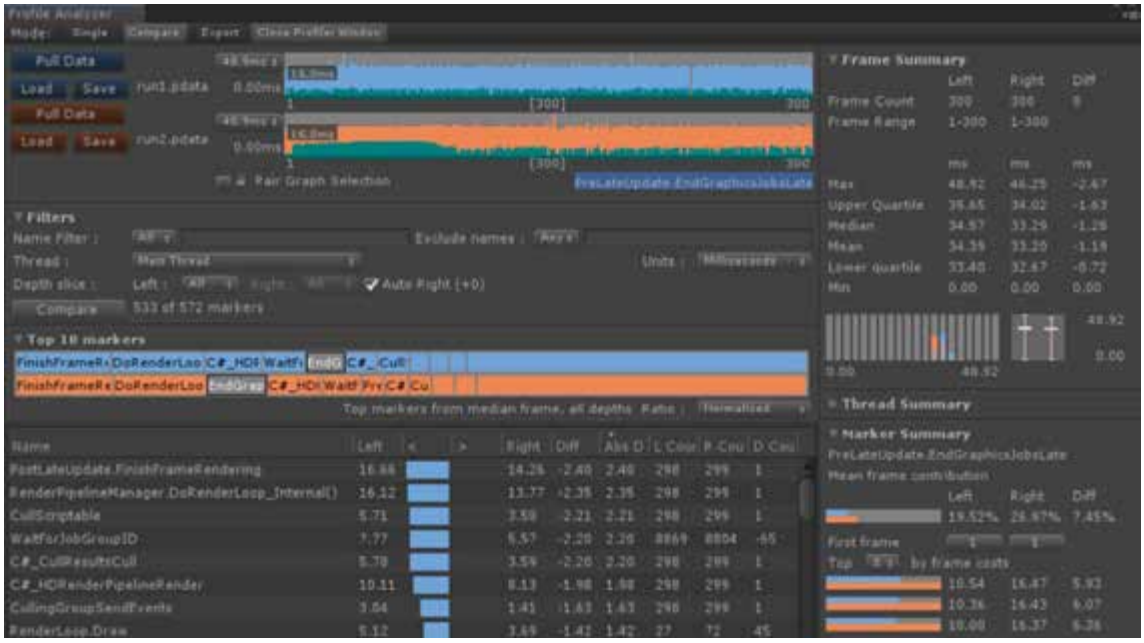

Рис. 9 Вікно Profile Analyzer

Unity Memory Profiler - це інструмент, який надається Unity для аналізу використання пам'яті в ігрових проектах. Він дозволяє розробникам відслідковувати, як використовується оперативна пам'ять під час виконання гри, виявляти витік пам'яті та оптимізувати використання ресурсів для підвищення продуктивності і забезпечення стабільності гри. На рис. 10 зображено вікно Unity Memory Profiler.

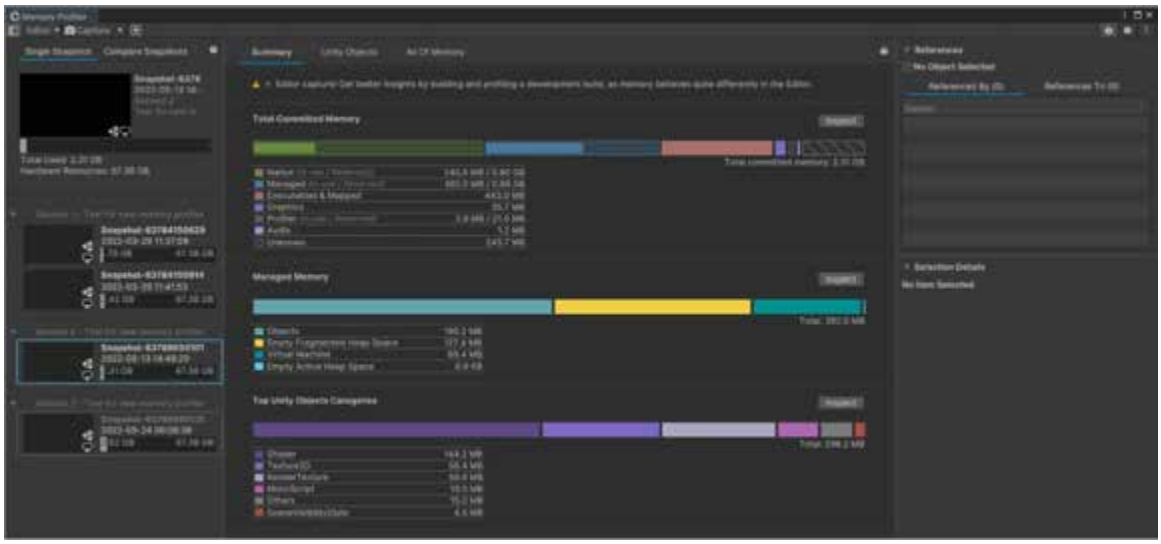

Рис. 10 Вікно Memory Profiler

Профілювання пам'яті: Unity Memory Profiler здатен відстежувати, як ваша відеогра використовує оперативну пам'ять під час її виконання. Він фіксує розмір пам'яті, який виділяється для об'єктів, ресурсів та інших компонентів відеогри.

Виявлення витоків пам'яті: Інструмент допомагає виявляти витік пам'яті, коли пам'ять не вивільняється після завершення виконання об'єкта чи компонента. Це важливо для запобігання переповненню пам'яті та збереження стабільності гри.

Аналіз ресурсів: Unity Memory Profiler дозволяє вам переглядати, які ресурси завантажені в пам'ять, і визначати їхнє використання. Це допомагає вам знайти зайві ресурси та уникнути зайвого використання пам'яті.

Інтерфейс інтерактивного аналізу: Інструмент надає інтерфейс, що дозволяє вам взаємодіяти з відстежуваними об'єктами та компонентами, щоб отримати більше інформації про їхнє використання пам'яті.

Зручний аналіз даних: Unity Memory Profiler відображає дані у зручному та зрозумілому форматі, що допомагає вам швидко знайти проблемні місця та приймати рішення щодо оптимізації.

Unity Memory Profiler є важливим інструментом для розробників відеоігор, оскільки дозволяє виявляти та усувати проблеми з пам'яттю, що можуть впливати на продуктивність та стабільність відеогри. Оптимізоване використання пам'яті допомагає гарантувати, що ваша гра працює ефективно та без збоїв на різних платформах.

2.4 Інструменти Unity для оптимізації графіки

Unity надає різні інструменти та можливості для оптимізації графіки в відеоіграх. Оптимізація графіки дозволяє забезпечити плавний ігровий досвід на різних пристроях і платформах, зменшуючи навантаження на графічну підсистему.

Level of Detail (LOD): LOD дозволяє розробникам створювати декілька версій моделей і змінювати їх деталізацію залежно від віддаленості від об'єкта. На рис. 11 можна побачити як змінюється деталізація 3D моделі в залежності від відстані до неї, зліва найменш деталізована, з права найбільше деталізована. Це зменшує кількість полігонів, які потрібно обробити в графіці, коли об'єкт знаходиться подалі, що покращує продуктивність.

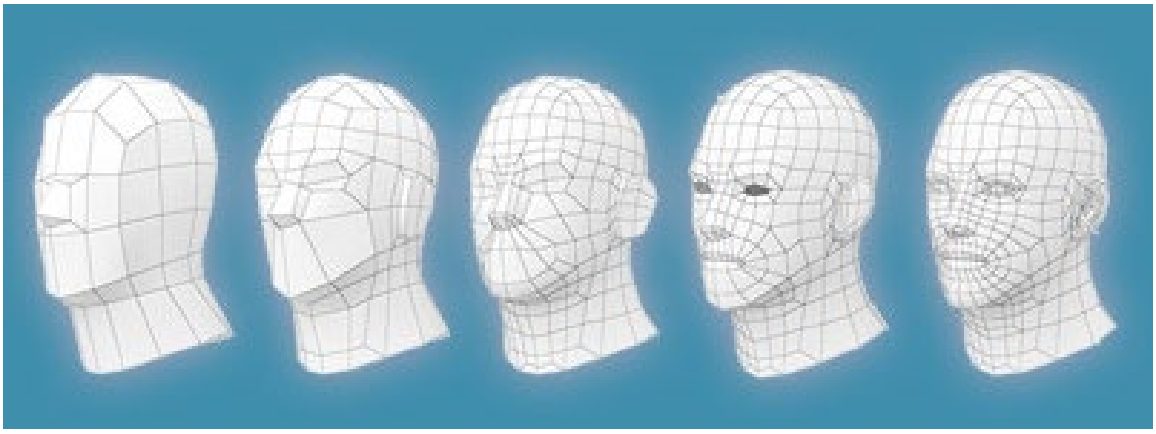

Рис. 11 Деталізація моделей для в залежності від відстані до неї

Для підключення LOD в Unity потрібно додати на Game Object компонент LOD Group та налаштувати його. Зовнішній вигляд цього компонента зображено на рис. 12.

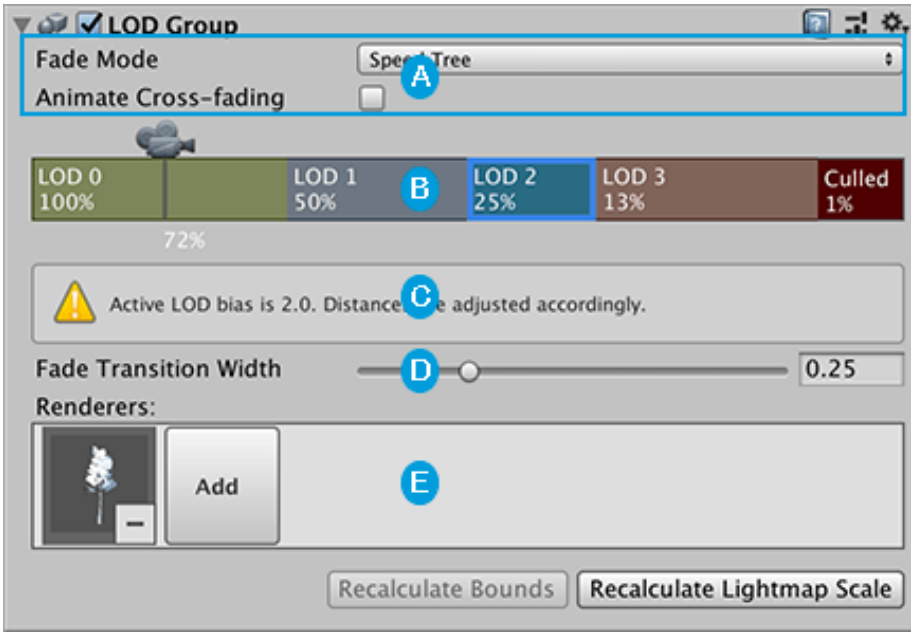

Рис. 12 Зовнішній вигляд компонента LOD Group

Компонент складається з таких частин:

- A Елементи керування для переходу між рівнями LOD
- B Панель вибору групи LOD для перемикання між рівнями LOD і попереднього перегляду візуалізації LOD
- C Інформація про параметр Lod Bias Quality. Це повідомлення з’являється, якщо для властивості Lod Bias встановлено значення, не дорівнює 1.
- D Параметр Fade Transition Width для вибраного рівня LOD. Ця властивість з’являється, лише якщо ви вимкнете властивість Animate Cross-fading , тобто коли ви вирішите встановити зону переходу за шириною, а не за часом.
- E Mesh Renderers , встановлений для вибраного рівня LOD

Крім того, у нижній частині компонента є дві кнопки: Recalculate Bounds повторно обчислює обмежувальний обсяг усіх LOD Mesh Game Objects після додавання нового рівня LOD, та Recalculate Lightmap Scale, щоб оновити

властивість Scale in Lightmap для всіх LOD Меш-рендерерів, на основі змін, які були внесені до меж рівня LOD.

Occlusion Culling – ця техніка дозволяє визначити, які частини сцени або об'єкти приховані і не відображаються на екрані, та приховати їх для зменшення навантаження на GPU. Кожен кадр камера відмальовує всі об’єкти які потрапляють в фруструм камери, і при цьому бувають об’єкти які потрапляють у фруструм але які ми не бачимо, тому що вони можуть перекриватися іншими об’єктами. Тут нам на допомогу і приходить Occlusion Culling. Він використовуючи віртуальну камеру проходить сценою для побудови ієрархії потенційно видимих об’єктів. Ці дані використовуються підчас гри для того щоб камера розуміла бачить вона об’єкт чи ні. Спираючись на цю інформацію відмальовуються лише ті об’єкти які безпосередньо потрапляють у поле видимості камери.

Дані для Occlusion Culling складаються з осередків. Кожен осередок - частинка сцени, а самі вони утворюють бінарне дерево. Occlusion Culling використовує два дерева. Перше View Cells (для статичних об'єктів), друге Target Cells (для об'єктів, що рухаються). View Cells містить список індексів, який визначає видимість статичних об'єктів із вищою точністю.

На рис. 13-15 зображена робота Occlusion Culling.

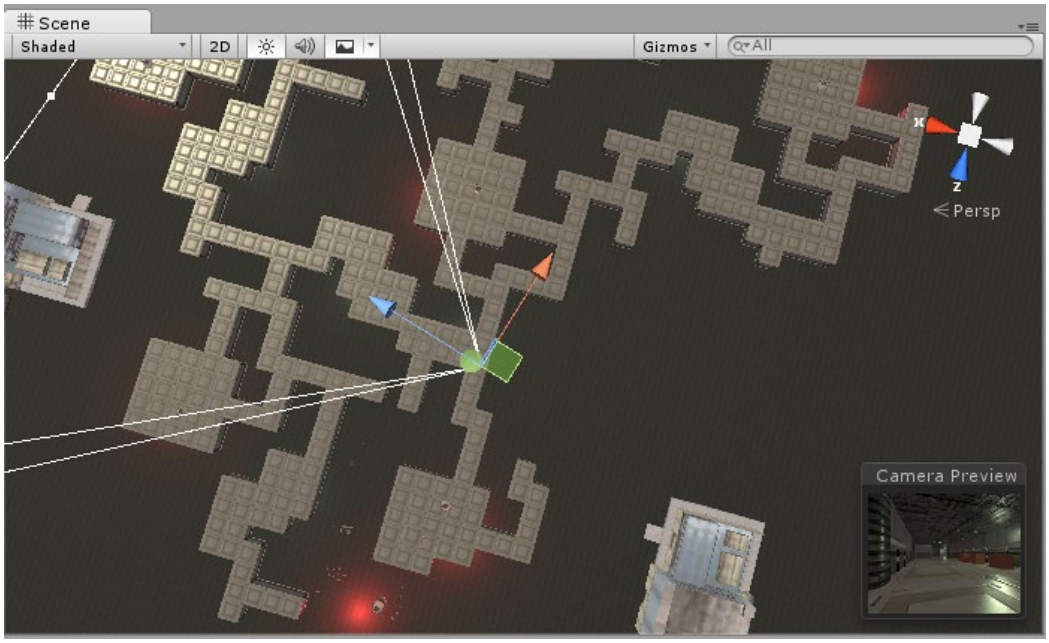

Рис. 13 Загальний вигляд сцени з усіма наявними на ній об’єктами

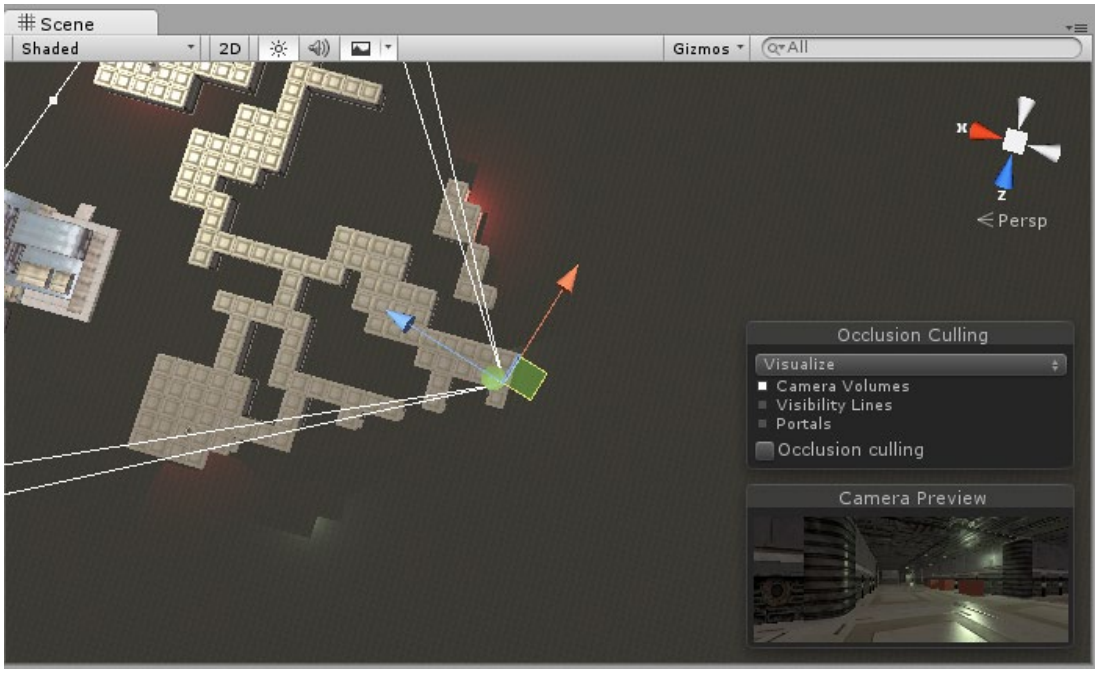

Рис. 14 Сцена з об’єктами які потрапляю у фруструм камери

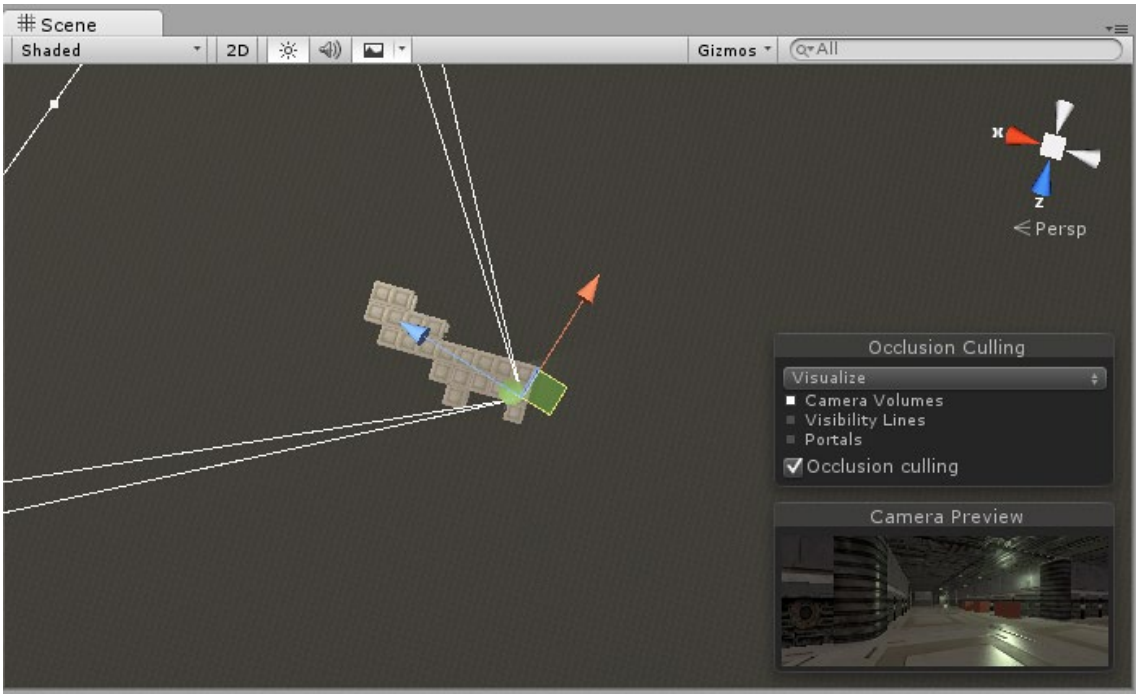

Рис. 15 Сцена з об’єктами, які безпосередньо потрапляють у кадр

На рис. 13 зображено всю сцену з наявними на ній об’єктами. На рис. 14 ми бачимо лише ті об’єкти які будуть відмальовуватися при виключеному Occlusion Culling, тобто всі які потрапляють у фруструм. Як видно на рис. 15 використання Occlusion Culling дає змогу значно зменшити навантаження на систему значно зменшуючи кількість об’єктів які потрібно відмалювати.

Batching. Для відтворення об’єкту на екрані рушій має виконати виклик відмалювання до графічного API наприклад до OpenGL або Direct3D. Batching (баччинг) - це техніка оптимізації графіки в Unity, яка дозволяє об'єднувати графічні об'єкти в один батч (групу) і малювати їх одним обчисленням, що покращує продуктивність гри. Існує два основних типи баччингу в Unity: Static Batching (статичний баччинг) і Dynamic Batching (динамічний баччинг).

Static Batching (статичний баччинг): Цей тип баччингу застосовується до статичних об'єктів, тобто тих, які не рухаються протягом гри. Unity об'єднує їх великі групи (меші), і ці меші рендеряться більш ефективно, оскільки вони вимагають менше обчислень. Це особливо корисно для великих ігрових рівнів зі

статичними об'єктами, такими як стіни, дороги тощо. Static Batching робиться автоматично Unity під час обробки об'єктів на сцені, які відзначені як статичні.

Dynamic Batching (динамічний баччинг): Динамічний баччинг використовується для маленьких мешів, які рухаються або змінюють свій стан під час гри. Unity трансформує їх вершини на центральному процесорі, об'єднує схожі вершини і малює їх разом одним викиданням. Це допомагає зменшити навантаження на графічний процесор, оскільки він малює одну велику групу об'єктів замість багатьох окремих.

Загальна ідея баччингу полягає в тому, щоб мінімізувати кількість викликів до графічного процесора та ефективно організувати рендеринг об'єктів. Використання баччингу є важливим етапом у процесі оптимізації графіки, що допомагає досягти плавної та продуктивної гри в Unity.

Оптимізація освітлення. Освітлення в Unity процес який споживає велику долю ресурсів пристрою. Щоб краще оптимізувати освітлення важливо розуміти як воно працює в Unity і що для цього використовується.

Освітлення в Unity можна розглянути з двох сторін, як «попередньо розраховане» (запечене) і як «освітлення в реальному часі». Обидва ці варіанти можуть використовуватися в проектів для отримання максимально природнього освітлення. Зазвичай для статичних об’єктів використовують запечене світло, так як таке освітлення вже прораховане, воно потребує мінімум потужностей для його відображення. І навпаки світло в реальному часі потребує більше потужності, так як для нього потрібно зробити безліч прорахунків, наприклад таких як тіні або відбиття.

У Unity є 5 типів джерел світла:

- Directional Light. Найпростіший, імітує сонячне світло. Воно представляє собою багато паралельних один одному променів.

- Point Light. Точкове джерело світла, тобто промені розходяться на всі боки з однієї точки. Хорошим прикладом такого джерела світла буде звичайна лампочка.
- Area Light. Джерело світла, що має площу. Прикладом може бути прямокутна панель, з якої виходить світло, зазвичай такі панелі використовуються в офісах.
- Ambient Light. Заповнююче світло, що не має джерела. Приклади використання: висвітлити занадто темні тіні; додати атмосферності підземелля, заповнивши його ледь помітним світлом біолюмінісцентних рослин.
- Light Probes . Особливе джерело світла, що впливає виключно динамічні об'єкти.

Розглянемо детальніше точкове джерело світла. По замовчуванню в Unity освітлення спрямоване, конусне, точкове і розраховується в реальному часі, а саме воно випромінює пряме світло на сцену і оновлюється в кожному кадрі. На рис. 16 зображено приклад направленого конусного світла.

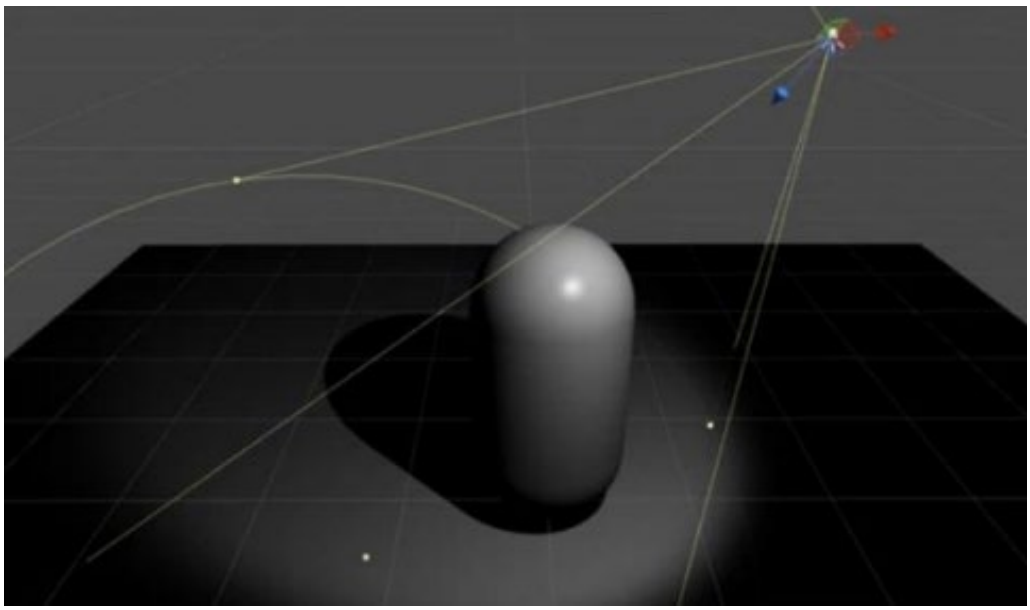

Рис. 16 Направлене конусне освітлення.

Освітлення в реальному часі є найпростішим рішенням для рухомих об’єктів і персонажів. Але в Unity для створення більш реалістичних сцен потрібно використовувати глобальне освітлення, але для оптимізації найкращим методом буде запікання глобального освітлення.

Запікання світла (Light Baking) - це процес попереднього обчислення освітлення та збереження його у текстурі чи карті для подальшого використання в реальному часі в грі. Цей процес використовується для оптимізації відображення освітлення у відеоіграх, особливо для сцен, які вимагають великої кількості обчислень.

Основні кроки запікання світла включають:

- Геометричний облік: Спочатку ігрова сцена обробляється для обчислення геометричної інформації, такої як положення об'єктів, їхні форми та матеріали. Це допомагає визначити, як світло взаємодіє з поверхнею об'єктів.
- Обчислення освітлення: Система обчислює, як світло від джерел світла впливає на об'єкти в сцені. Цей крок включає в себе розрахунок тіней, розсіяння світла та інших параметрів освітлення.
- Запікання у текстури (Lightmaps): Обчислені дані про освітлення зберігаються у вигляді текстур, які називаються lightmaps. Lightmaps містять інформацію про те, як світло розподіляється по поверхнях об'єктів.
- Застосування lightmaps: У реальному часі, під час гри, графічний рушій читає lightmaps та застосовує освітлення до об'єктів на основі їхньої геометрії та позиції.

Після запікання світла сцена може бути рендерена у реальному часі зі значно меншим навантаженням на процесор та графічний прискорювач, оскільки більшість обчислень освітлення вже виконано попередньо. Це особливо корисно

для відеоігор на мобільних пристроях та відеоігор з великою кількістю об'єктів на сцені. Слід зауважити при запеченому світлі нам не можна переміщати статичні об’єкти, тому що світло не буде переміщене, натомість потрібно буде заново перезапікати світло, але продуктивність при такому підході значно вище. Приклад запеченого світла зображено на рис.17.

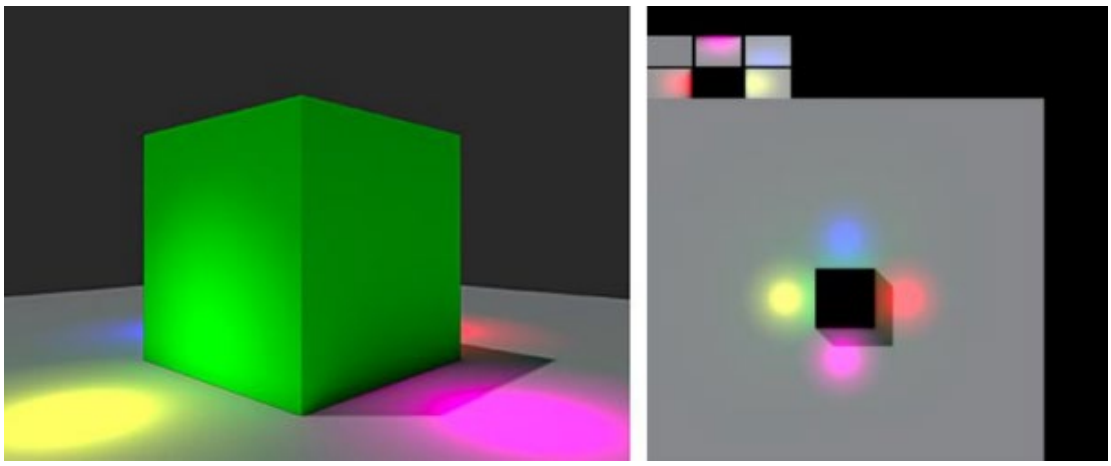

Рис. 17 Приклад запеченого в Unity світла

Unity надає можливість використовувати систему Baked Global Illumination для запікання освітлення. Ця система дозволяє налаштовувати параметри запікання, такі як роздільність lightmaps, рівень деталізації, розгортання UV-координат для об'єктів тощо. Це допомагає створити хороше освітлення у відеоіграх і забезпечити високу продуктивність на різних платформах.

2.5 Інструменти Unity для оптимізації фізики

Фізику в Unity можна оптимізувати багатьма способами, але перш за все як важливо подумати чи взагалі вона потрібна в проекті. Якщо відеогра не потребує фізичних симуляцій або реалістичної поведінки об’єктів у грі, то від фізики можна взагалі відмовитися, а всі процеси у грі такі наприклад як рух або колізію об’єктів можна симулювати іншими більш дешевими в плані продуктивності варіантами. Наприклад колізію об’єктів можна симулювати за допомогою дерева квадрантів.

Вибір колайдерів. Якщо ж вам все таки доводиться використовувати фізику то першим кроком в її оптимізації буде використання простих колайдерів. На рис. 18 зображені прості колайдери в Unity в порядку простоти вираховування колізій.

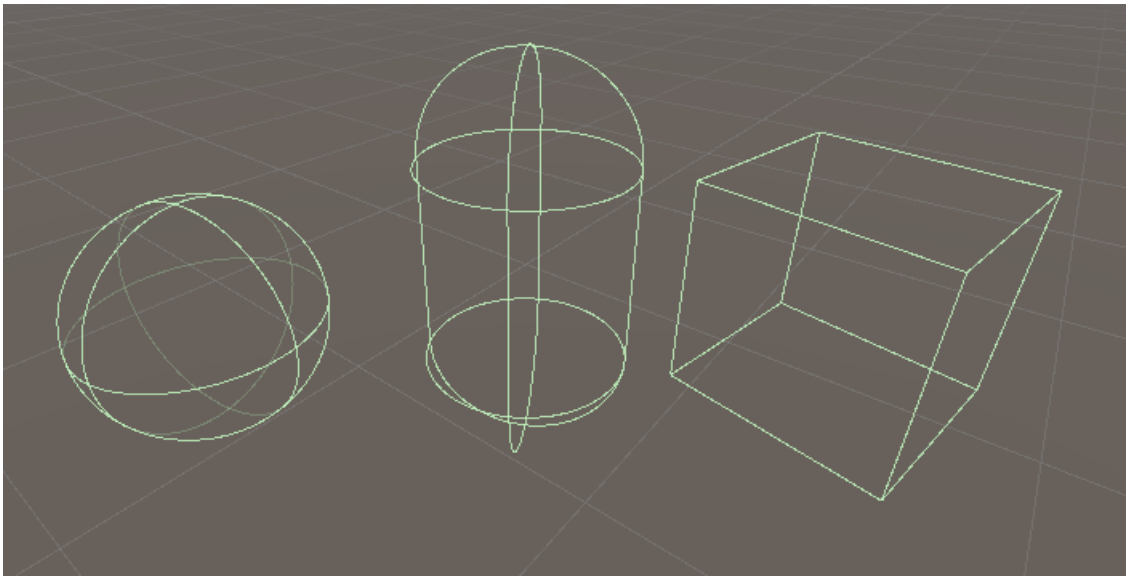

Рис. 18 Прості колайдери в Unity

В Unity також є і Mesh Collider (рис. 19). Його особливість в тому що він повністю повторює 3D модель об’єкта, але коли діло доходить до вираховування колізії з іншими об’єктами це сильна навантажує процесор і як наслідок страждає оптимізація відеогри. Якщо ваш проект не зав’язаний на детальній колізії об’єктів використання простих колайдерів зменшить навантаження при цих обрахуваннях, а продуктивність відеогри збільшиться.

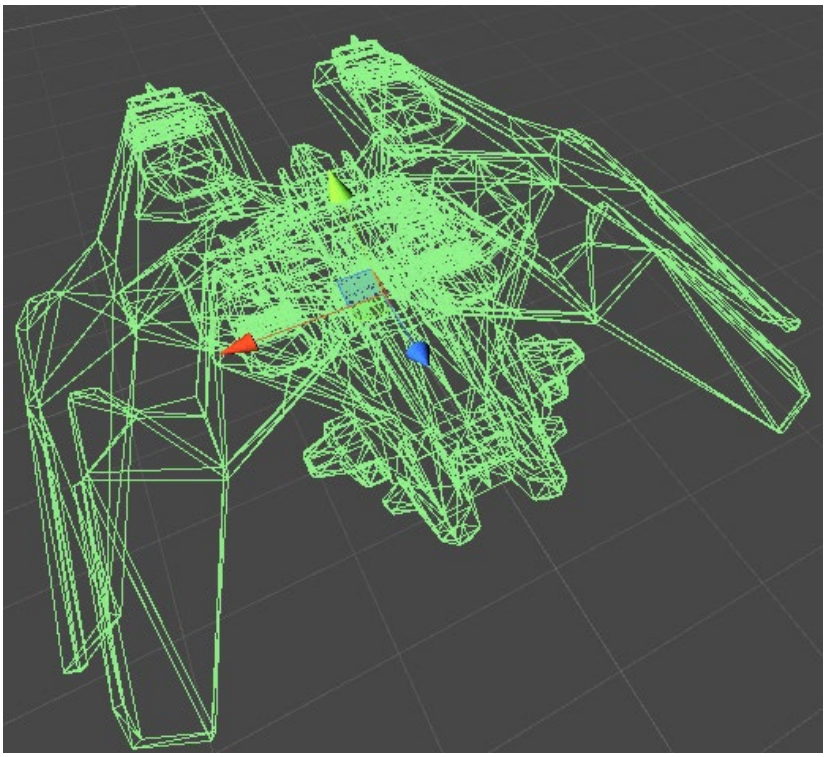

Рис. 19 Mesh Collider 3D об’єкта робота.

При використанні простих колайдерів на персонажах зазвичай вистачає лише одного або двох простих колайдерів (рис. 20) , щоб обрахувати колізії з персонажем.

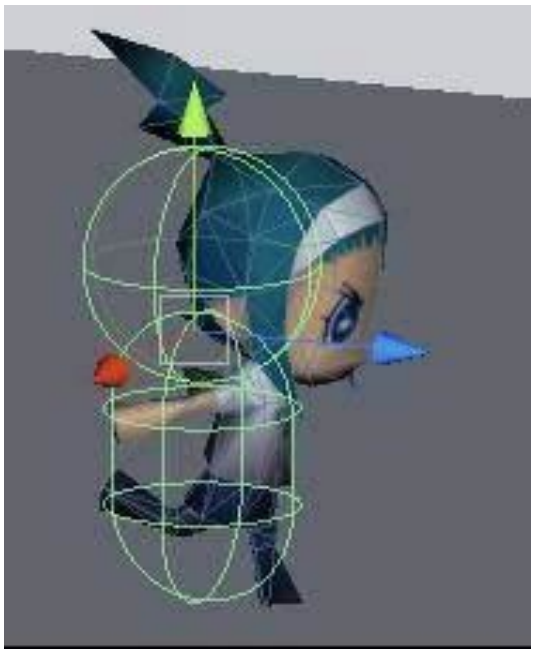

Рис. 20 Комбінування двох простих колайдерів для персонажу відеогри.

Для більш складних об’єктів або коли гра вимагає детального обрахування колізій кількість простих колайдерів можна збільшувати. Наприклад на рис. 21 зображено автомат який складається з декількох простих колайдерів, що є значно більш оптимізованим варіантом чим використання меш колайдеру.

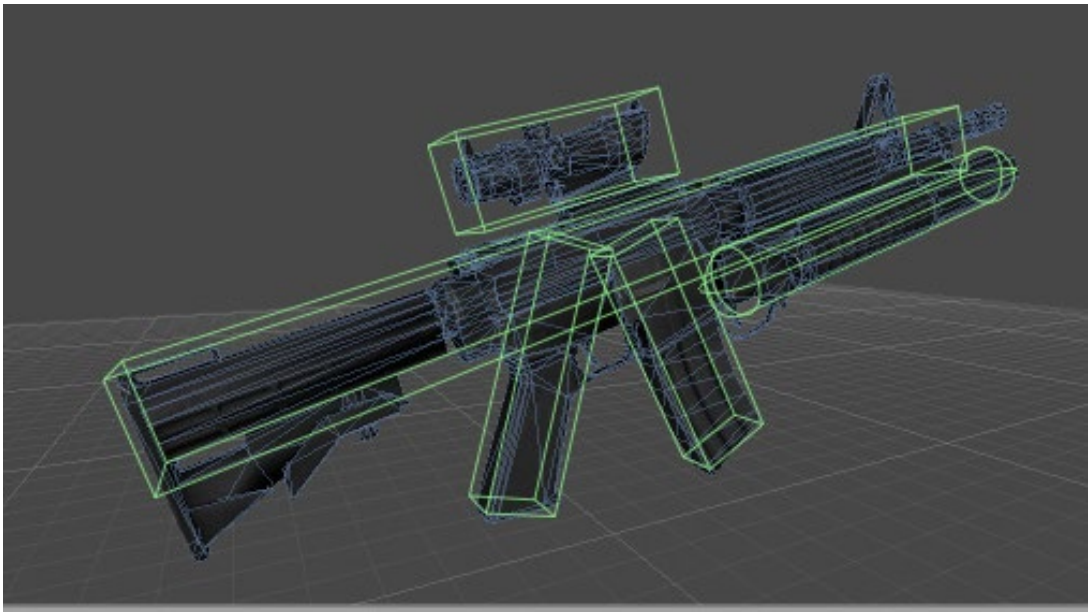

Рис. 21 Комбіновані прості колайдери для моделі автомату.

Кількість обрахунків фізики. Важливим аспектом оптимізації є також баланс кількості колайдерів та фізичної взаємодії між ними. Чим менше на сцені об’єктів що використовують фізику тим більш оптимізованим буде відеогра. Зменшити кількість обрахунків фізики можна декількома способами.

Перший самий очевидний – зменшити кількість об’єктів які одночасно взаємодіють у грі, це можна зробити видаливши зайві колайдери або колайдери які не виконують ніяких функцій наприклад в недоступних для гравця місцях.

Другий спосіб це використання шарів та матриці колізій (рис. 22). Кожен об’єкт на сцені знаходиться на певному шарі. Якщо цей шар не вказано то по замовчуванню встановлений шар Default. Правильно буде розмістити групи об’єктів на відповідних шарах. Зазвичай можна виділити такі основні шари як гравець, оточення та противники. В залежності від гри цих шарів може бути багато.

В самій матриці колізій для кожного шару додається відповідний стовпчик і рядок, який відповідає колізії між цими шарами. Виконавши налаштування матриці колізій ми можемо уникнути непотрібних обрахунків між шарами які не повинній пересікатись, що в свою чергу зменшить навантаження на систему.

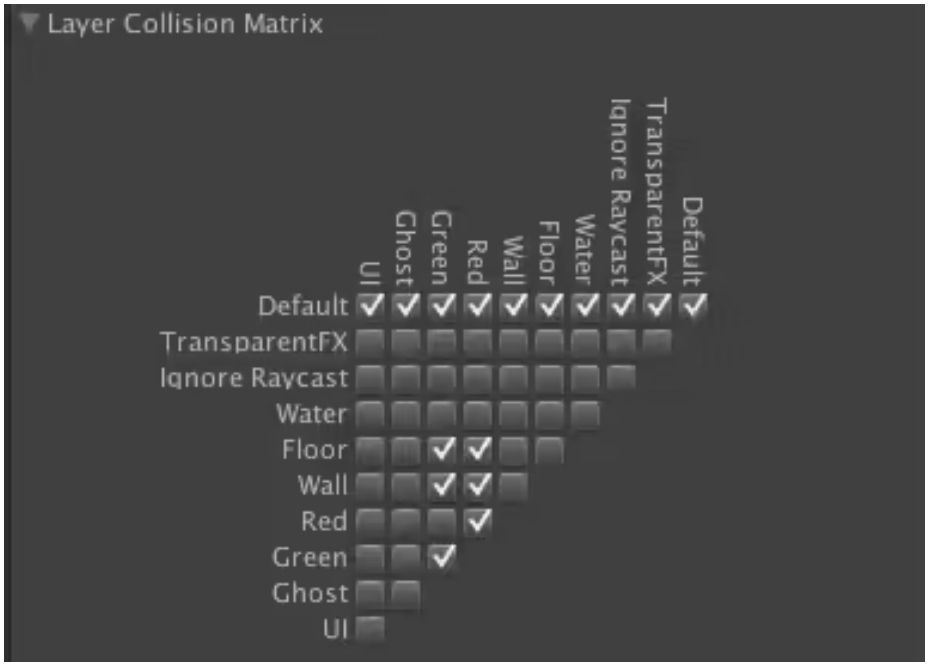

Рис. 22 Матриця колізій

Рейкастинг. Коли нам потрібно визначити чи є перед нами якийсь об’єкт або потрібно дізнатися відстань до нього використовується рейкастинг. Він дозволяє направити промінь в потрібному напрямку і перевірити чи проходить якась колізія з ним. Але це досить важке обчислення. При його використанні важливо оптимізувати його роботу. Для цього можна врахувати такі поради, як:

- Потрібно використовувати оптимальну кількість променів, щоб не перенавантажувати систему його обчисленнями;
- Не робити промені занадто довгими, на приклад якщо персонаж має певну дальність огляду то промінь повинен відповідати цій відстані;
- Не проводити обрахунок променів у методах FixedUpdate() або Update(). Ці методи виконуються дуже часто, і використання важких операцій в цих методах не бажане з точки зору оптимізації;

- Важливим чинником є те які колайдери зустрічаються на шляху променю. Як зазначалося вище колізія з Mesh колайдером досить сильно навантажує систему;
- Також важливим є використання маски шарів що промінь перевіряв колізію лише з тими колайдерами які його цікавлять, а всі інші ігнорував.

Для прикладу було проведено порівняння використання Mesh колайдеру та звичайного Box колайдеру на тестовій сцені. Один об’єкт направляє промені навколо себе у всі сторони, а навколо нього знаходяться інші об’єкти червоного та зеленого кольору (рис. 23) з простим Box колайдером. Колізія проходить лише з об’єктами зеленого кольору, а червоного ігноруються.

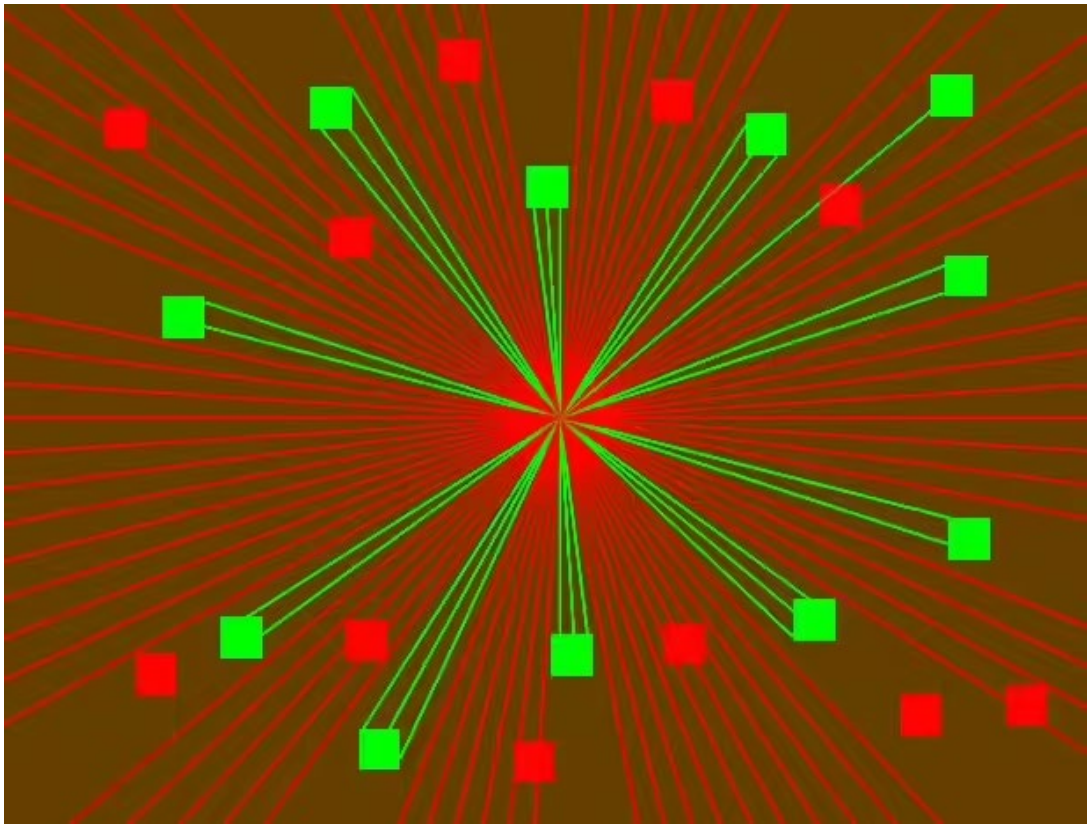

Рис. 23 Скріншот сцени з тестом колізії променів з Box колайдерами

Після зняття показників продуктивності профайлера, Box колайдери були замінені на Mesh колайдери (рис. 24). Кожен Mesh колайдер містить 110 полігонів.

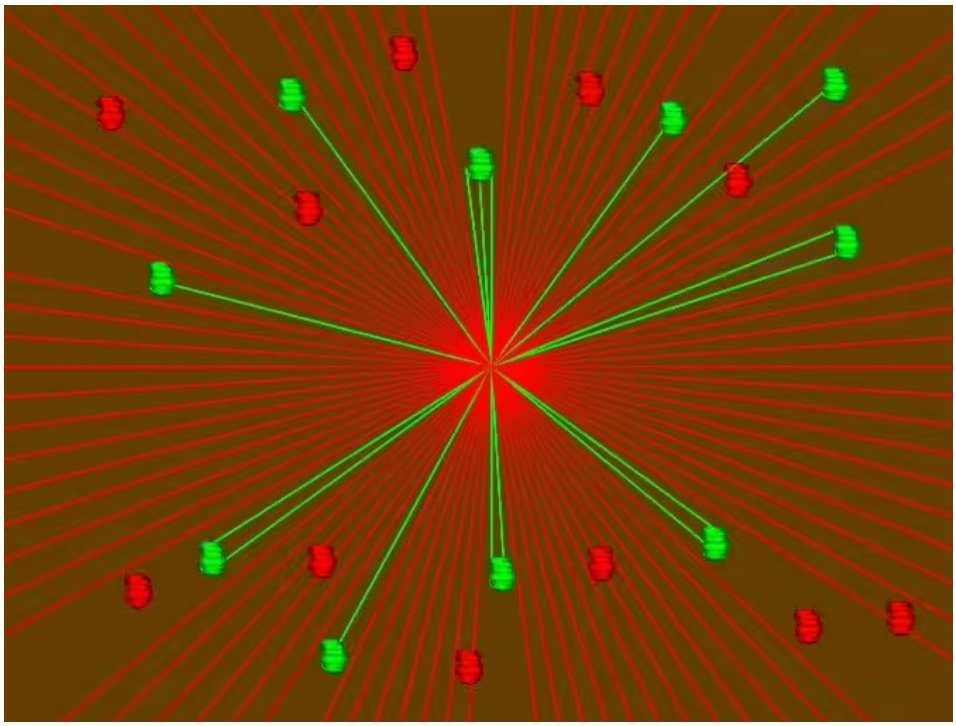

Рис. 24 Скріншот сцени з тестом колізії променів з Box колайдерами

Протестувавши яке навантаження на виникає при використанні Mesh колайдерів можемо порівняти дані. Дані профайлера зображені на рис. 25, зліва результати використання примітивних колайдерів, справа використання Mesh колайдерів. Графік чітко показує, що рейкаст Mesh колайдерів обходиться системі значно дорожче ніж використання Box колайдера.

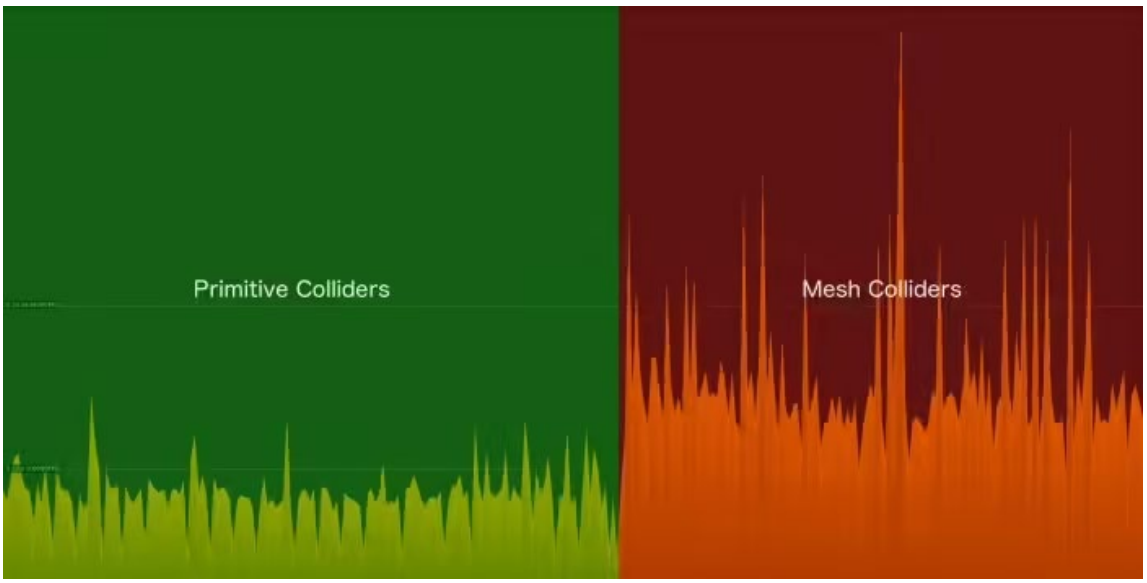

Рис. 25 Діаграма профайлера по навантаженню рейкасту для Box колайдерів зліва, для Mesh колайдерів справа.

Rigidbody. Колайдери використовуються в фізиці для того, щоб фізичний рушій мав уявлення про форму фізичного об’єкту. Але використання лише колайдера сигналізує лише про те що об’єкт повинен бути статичним, тобто нерухомим. Якщо ми будемо рухати такий об’єкт наприклад із коду то фізичний рушій почне заново перераховувати положення всіх фізичний об’єктів на сцені, що буде додатково навантажувати систему. Якщо ж таке буде відбуватися то профайлер буде видавати попередження про переміщення статичних колайдерів (рис. 26).

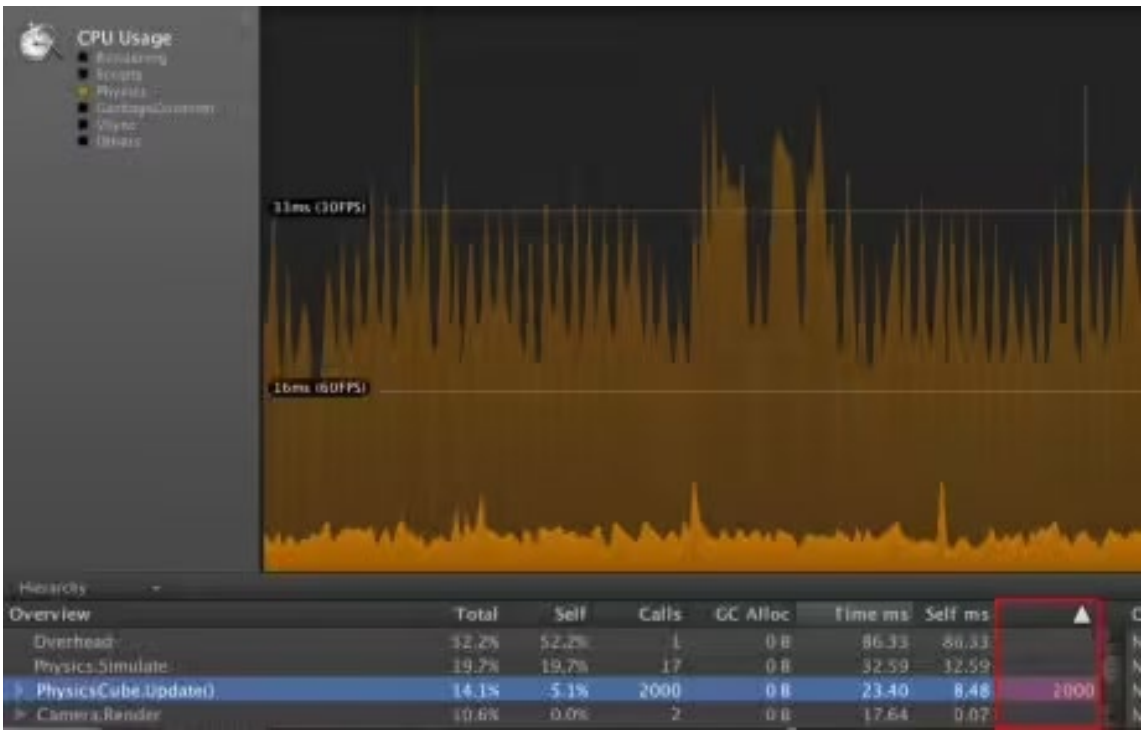

Рис. 26 зображення профайлеру Unity з попередженнями.

В новіших версіях Unity цей недолік виправили, а попереджень в профайлері більше немає, але все ж рекомендація не рухати статичні об’єкти залишилась, то му що фізичний рушій намагається їх оптимізувати [4].

Rigidbody – це компонент який забезпечує фізичну взаємодію між об’єктами. Він вказує фізичному рушію, що об’єкт може рухатися і на нього повинні діяти фізичні закони. Тому на всіх об’єктах які будуть рухатися у відеогрі він повинен бути. Якщо рух об’єкта відбуватиметься через компонент Transform, то компонент Rigidbody потрібно перевести у стан Kinematic.

Fixed Timestep. Вираховування фізики не залежить від кількості кадрів, а проводиться через певний проміжок часу що називається Timestep. Саме тому фізичні обчислення виконуються однаково не залежно від частоти кадрів на конкретному пристрої, а симуляція відбувається майже однаково. Цей час впливає на те як часто буде вираховуватись фізичні взаємодії між об'єктами, для більшої оптимізації цей час можна збільшити, але потрібно бути з ним обережним щоб відеогра встигала точно прораховувати всі взаємодії та не виникало не потрібних артефактів. По замовчуванню цей параметр відповідає 0.02 секунди. Щоб змінити цей параметр потрібно перейти в Project Settings -> Teme -> Fixed Timestep (рис. 27).

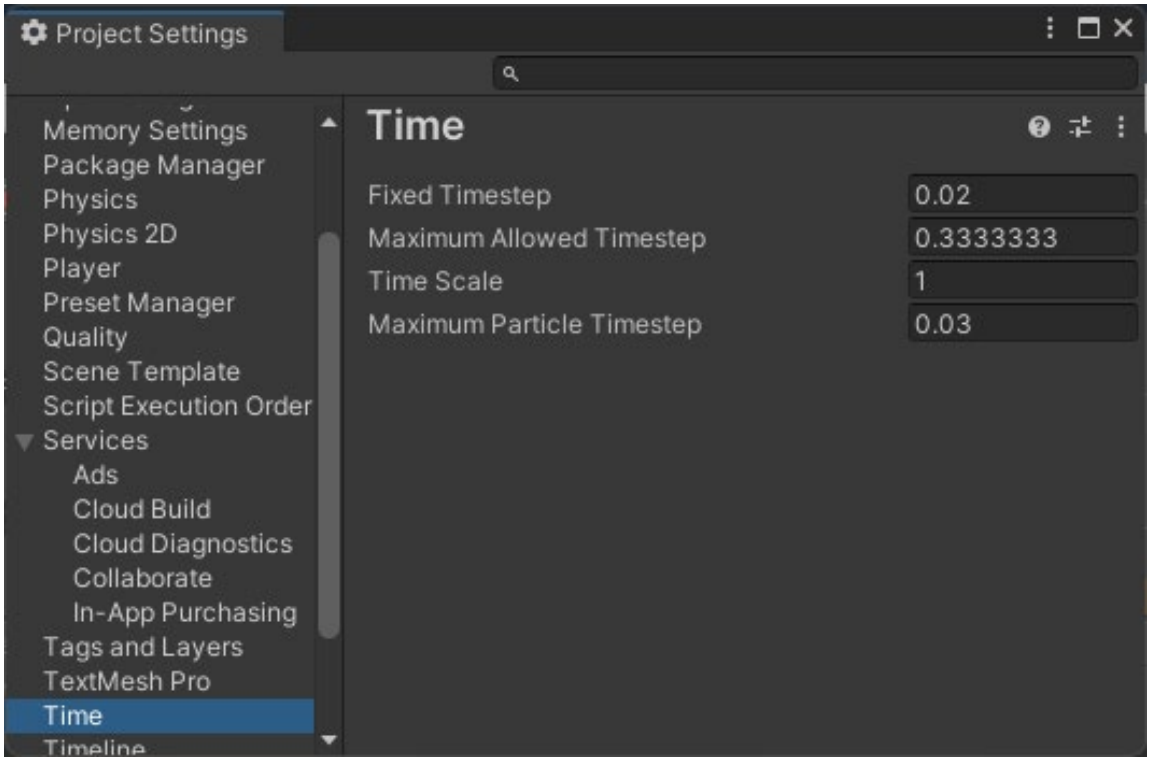

Рис. 27 Вікно для зміни Fixed Timestep

2.6 Інструменти Unity для оптимізації коду

Оптимізація коду в Unity дуже важлива, оскільки не оптимізований код може спричинити погіршення продуктивності та падіння кадрів у відеогрі. Оптимізація коду - це процес постійного вдосконалення та пошуку шляхів зниження навантаження на систему. Ретельний аналіз та оптимізація коду допомагають забезпечити плавну та ефективну гру в Unity.

Використання Profiler. Profiler – це ключовий інструмент для аналізу продуктивності відеогри. Він надає інформацію про використання ресурсів, таких як CPU, GPU та пам'ять. Profiler виводить дані в реальному часі, дозволяючи вам виявляти проблеми з продуктивністю, такі як графічні переривання або завантаження пам'яті. Більшість з проблем пов’язаних з поганою продуктивністю коду легко можна знайти через Profiler, адже він відображає повну картину по кожному методу та класу.

Мінімізація вкладених циклів. Вкладені цикли можуть значно збільшити кількість обчислювальних операцій і збільшити навантаження на CPU. Зменшення вкладених циклів допомагає покращити продуктивність. Для кращої продуктивності краще обмежити кількість ітерацій додатковими перевірками або знаходячи більш ефективні методи для обробки даних.

Використання асинхронності. Використання асинхронних функцій дозволяє виконувати операції паралельно з головним потоком відеогри, що зменшує блокування та покращує продуктивність. В Unity для такий цілей можна використовувати Coroutine та async/await щоб розділити та виключити блокування основного потоку гри.

Використання швидких структур даних. Використання швидких структур даних, таких як масиви або List<T>. В Unity є власна структура List<T> яка побудована на основі динамічного масиву, тому вона є більш ефективною ніж звичайний List<T> який є в мові C#.

Кешування результатів обчислень та об’єктів. Кешування – це збереження результатів обчислень для подальшого використання, замість повторного обчислення. Це особливо корисно для обчислювально важких операцій, які не змінюються з кадру на кадр.

Також важливо кешувати об’єкти та компоненти які ви використовуєте при написанні скрипта. Зазвичай в MonoBehaviour скриптах кешування проводиться в методі Awake або Start (рис 28). Метод GetComponent бере компонент який знаходиться в тому ж об’єкті на якому висить скрипт, а тип компонета передається в кутових дужках. Важливо пам’ятати що метод GetComponent важкий для обробки тому використання його в методах які обробляються кожен кадр, таких як Update або FixedUpdate, дуже не вигідне з токи зору оптимізації.

```
C# MoveObject.cs ×
1   using UnityEngine;
    No asset usages
2   public class MoveObject : MonoBehaviour
3   {
4       private Transform _transform;
        Event function
5       private void Start()
6       {
7           // Кешування об'єкта
8           _transform = GetComponent<Transform>();
9       }
        Event function
10      private void Update()
11      {
12          // Використання об'єкта
13          _transform.Rotate( eulers: Vector3.one * 1000);
14          _transform.position += Vector3.one;
15      }
16  }
```

Рис. 28 Кешування об’єкта в методі Start

Ще одним варіантом кешування є передавання об’єкту в скрипт через інспектор (рис. 29).

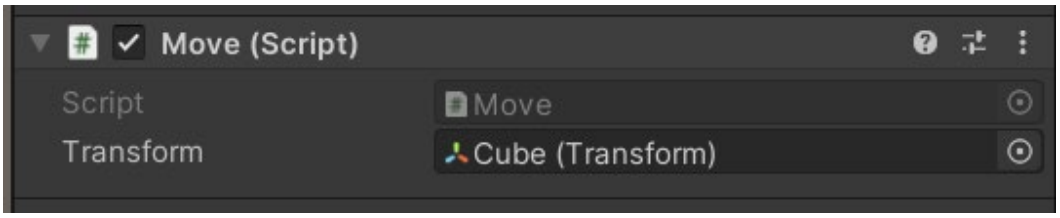

Рис. 29 Передавання об’єкту в скрипт через інспектор

Для того щоб передати об’єкт через інспектор поле повинно бути публічним, але публічні поля це погана практика, тому в Unity зазвичай використовуються атрибути в квадратних дужках, і для цього випадку є атрибут SerializeField (рис. 30). При його використанні поле залишається приватним, але в інспекторі доступним для передачі туди об’єкта.

```
C# Move.cs ×
1   using UnityEngine;
2
    No asset usages
3   public class Move : MonoBehaviour
4   {
5       // Кешування об'єкта
6       [SerializeField] private Transform _transform;  Unchanged
7
        Event function
8       private void Update()
9       {
10          // Використання об'єкта
11          _transform.Rotate( eulers: Vector3.one * 1000);
12          _transform.position += Vector3.one;
13      }
14  }
15
```

Рис. 30 Кешування об’єкту через передачу його в інспекторі

Пул об'єктів (Object Pooling). Об'єкт-пулінг - це метод оптимізації, при якому об'єкти створюються перед грою та повторно використовуються під час гри. Це допомагає уникнути надмірного створення та знищення об'єктів, що допомагає уникнути навантаження на збирач сміття.

Для створення пулу об’єктів використовують різні структури даних такі як списки, черги або стеки. На їх основу будується клас який має методи отримання об’єкту з пулу та повернення в пул. З додаткових налаштувань в пул об’єктів можуть передаватися такі параметри як стартова кількість об’єктів яку потрібно створити при запуску гри, та кількість додаткових об’єктів які потрібно створити якщо в пулі не залишилося об’єктів. Також в пул передається префаб об’єкта який потрібно створювати або список об’єктів якщо пул використовується для створення більше ніж один тип об’єктів.

Розглянемо практичну причину використання Object Pooling. Наприклад у нас є аркадна відеогра в якій космічний корабель знищує астероїди та інші кораблі (рис. 31). Для стрільби він використовує умовні кулі, які є однаковими і створюються з одного префабу.

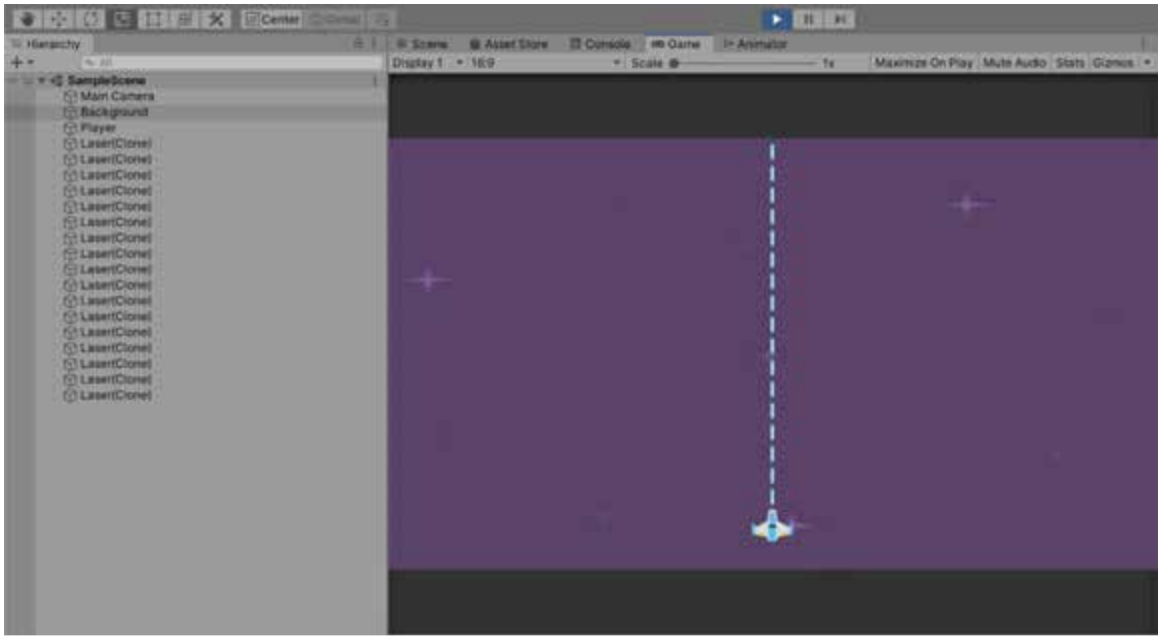

Рис. 31 Скріншот аркадної відеогри про космічний корабель

Без використання пулу нам потрібно кожен раз створювати нову кулю при кожному пострілі та знищувати її коли вона вийшла за екран або досягла цілі. Це в свою чергу з часом навантажить збирач сміття і в якийсь момент, особливо на слабших пристроях, ми можемо зловити зависання на декілька секунд, до того ж

і саме створення об’єкту вимагає постійних затрат на створення об’єкту. Це не ефективне використання ресурсів.

Натомість використаємо пул об’єктів, для цього створимо скрипт ObjectPool який матиме таку реалізацію створення пулу як на рис. 32.

```
public static ObjectPool SharedInstance;
public List<GameObject> pooledObjects;
public GameObject objectToPool;
public int amountToPool;

void Awake()
{
    SharedInstance = this;
}

void Start()
{
    pooledObjects = new List<GameObject>();
    GameObject tmp;
    for(int i = 0; i < amountToPool; i++)
    {
        tmp = Instantiate(objectToPool);
        tmp.SetActive(false);
        pooledObjects.Add(tmp);
    }
}
```

Рис. 32 Код скрипта ObjectPool для ініціалізації пулу

Наступним кроком потрібно повісити наш скрипт на об’єкт на сцені, та передати префаб нашої кулі в скрипт та встановити стартову кількість об’єктів (рис. 33).

Рис. 33 Скрипт на об’єкті з встановленими параметрами

Далі нам потрібно додати метод для отримання об’єкту з пулу. Код даного методу показано на рис. 34, а сам код потрібно розмістити в класі ObjectPool.

```
public GameObject GetPooledObject()
{
    for(int i = 0; i < amountToPool; i++)
    {
        if(!pooledObjects[i].activeInHierarchy)
        {
            return pooledObjects[i];
        }
    }
    return null;
}
```

Рис. 34 Код для отримання об’єкту з пулу

Тепер нам потрібно замінити в скрипті в якому ми раніше створювали кулі на виклик цього методу та встановити йому потрібні налаштування такі як позиція та поворот, далі залишиться активувати кулю (рис. 35).

```
GameObject bullet = ObjectPool.SharedInstance.GetPooledObject();
  if (bullet != null) {
    bullet.transform.position = turret.transform.position;
    bullet.transform.rotation = turret.transform.rotation;
    bullet.SetActive(true);
  }
```

Рис. 35 Код використання пулу

Після цього нам потрібно замінити в місцях де ми знищували кулю на її повернення в пул. В даному прикладі нам потрібно лише деактивувати об’єкт кулі (рис. 36).

```
gameobject.SetActive(false);
```

Рис. 36 Деактивація об’єкту кулі після використання

Наведений приклад демонструє простоту та ефективність такого підходу використання пулів. Варіантів реалізації пулів досить багато, і варто підібрати той, який буде більш ефективним в конкретному проекті.

В Unity версії 2021 були додані свої пули, та реалізовані варіанти з використанням різних структур даних. Одним із таких є ObjectPool, щоб його використовувати потрібно підключити простір імен UnityEngine.Pool. Далі створити поле з типом IObjectPool<T> і в якості типу пулу передати тип який будемо використовувати. В Awake в конструктор передаємо потрібні параметри такі як:

- метод створення елементу пулу коли пул порожній;
- метод який буде викликаний коли екземпляр береться з пулу;
- метод який буде викликаний коли екземпляр повернеться до пулу;
- метод який виконується при знищенні екземпляра;
- булева змінна чи буде виконуватись перевірка колекції, вона виконується тільки в редакторі;
- ємність за замовчуванням;
- максимальний розмір, Коли пул досягає максимального розміру, будь-які подальші екземпляри, що повертаються до пулу, ігноруватимуться та можуть бути зібрані як сміття. Це можна використовувати, щоб запобігти зростанню пулу до дуже великих розмірів.

Після реалізації потрібних методів пул готовий до використання. Щоб отримати елемент з пулу потрібно використати метод Get(), а для повернення в пул метод Release().

Entity Component System. Структура коду має важливе значення в оптимізації проекту. Для покращення коду було розроблено багато патернів проектування та архітектурних підходів. Одним із таких є ECS.

Entity Component System (ECS) - це парадигма програмування та архітектура, яка використовується в Unity для розробки відеоігор та оптимізації продуктивності. Вона спрямована на поліпшення продуктивності і оптимізацію ігрових застосунків, дозволяючи розробникам краще контролювати використання ресурсів.

ECS складається з таких компонентів:

- Entity (Сутність): Сутність в ECS - це абстракція для відеогри, яка представляє ігровий об'єкт чи об'єкти. Кожна сутність складається з набору компонентів.
- Component (Компонент): Компоненти - це дрібні частини інформації, що містять дані та функції для визначеної функціональності. Наприклад, компонент може представляти інформацію про позицію, рух, графіку об'єкта тощо.
- System (Система): Системи відповідають за обробку компонентів. Вони представляють логіку гри, таку як наприклад переміщення об’єкта, роботу фізики, логіку штучного інтелекту, рендеринг тощо. Кожна система обробляє певний набір компонентів для певної функціональності.

ECS має ряд переваг над стандартним підходом MonoBehaviour, який використовується по замовчуванню в Unity.

Висока продуктивність: ECS дозволяє ефективно перерозподіляти компоненти, що робить обробку даних більш ефективною та забезпечує високу продуктивність гри, зокрема на мобільних пристроях та інших обмежених платформах.

Складність коду: ECS розділяє код на компоненти та системи, що спрощує розробку та підтримку коду. Кожен компонент відповідає за конкретний аспект функціональності, що робить код більш структурованим та легко зрозумілим. Зазвичай компоненти роблять з типом даних структура, та мітить вона лише поля для зберігання даних, наприклад компонент для переміщення може мати такі поля як поточна позиція та швидкість, а компонент ціль (таргет) матиме координати позиції куди потрібно рухатися. Система яка буде обробляти рух буде брати ці два компоненти та з плином часу та залежно від швидкості перемістить об’єкт з поточної позиції в позицію цілі.

Легкість масштабування: ECS спрощує масштабування гри. Ви можете додавати нові компоненти та системи, не перероблюючи всю архітектуру. Через відсутність зв’язаності систем та компонентів один з одним можна без додаткових труднощів додавати та видаляти системи і компоненти. Якщо вимкнути якусь окрему систему, то це ніяк не вплине на інші системи, а у відеогрі лише не буде працювати те що виконує ця система.

Паралельна обробка: ECS робить можливою паралельну обробку компонентів та систем, що покращує продуктивність на багатоядерних процесорах. Через такий підхід з роботою систем вони легко розподіляються на окремі потоки, що в свою чергу позитивно впливає на оптимізацію на багатоядерних процесорах.

Unity введе підтримку ECS через Unity DOTS (Data-Oriented Technology Stack), який надає засоби для розробки високопродуктивних ігор. ECS і DOTS дозволяють розробникам отримувати більшу продуктивність, особливо для відеоігор з великою кількістю об'єктів і комплексною фізикою.

Але до того як з’явилася реалізація ECS від компанії Unity розробники реалізували окремі рішення архітектури ECS. Розглянемо деякі з них.

Entitas – один з найстаріших фреймворків для Unuty (рис. 37) був і залишається одним із най популярніших. Із плюсів має велике ком’юніті та

хорошу документацію, має пропрацьований WorldViewer в редакторі Unity та добре читабельний стиль коду. До мінусів можна віднести погану продуктивність відносно інших ECS фреймворків, але яка все ж краще ніж в MonoBehaviour, багато алокацій, що навантажує збірку сміття, потребує кодогенерації на кожну зміну структури компонентів та на великих проектах виростає розмір відеогри через кодогенерацію.

Рис. 37 Логотип фреймворку Entitas

LeoECS – мінімалістичний ECS фреймворк (рис. 38), простий в навчанні з відкритим вихідним кодам, другий за популярністю і один із най продуктивніших. Також має спрощену версію LeoEcsLite яка ще більш продуктивна, але має менш зручне API. До плюсів можна віднести мінімілістичність і легкість фреймворку, просте API, висока продуктивність, велике ком’юніті та може використовуватись без Unity на чистому C#, а як мінус через це доведеться багато елементів реалізовувати вручну.

Рис. 38 Логотип фреймворку LeoECS

DefaultECS – близький по духу до LeoECS фреймворк (рис. 39), побудований за принципом Engine-Agnostic, без будь-якої інтеграції з Unity та не поступається продуктивністю LeoECS та має власний аналог Jobs. Принцип Engine-Agnostic вказує на те що його можна використовувати на будь-якому рушію, але підключення до рушію потрібно реалізовувати самостійно.

default(ecs)

Рис. 39 Логотип фреймворку DefaultECS

Використання ECS може бути вимогливим в плані вивчення та розробки, але для деяких ігор і симуляцій він може бути надзвичайно корисним для досягнення високої продуктивності та складної функціональності.

РОЗДІЛ 3: ПРАКТИЧНІ АСПЕКТИ ОПТИМІЗАЦІЇ ІГРОВОГО ПЗ В UNITY

3.1 Аналіз та вибір ігрового проекту для оптимізації

Оптимізація важлива частина кожної відеогри. Для того щоб якісніше дослідити процес оптимізації та її ефективність, потрібно вибрати проект який повинен мати дуже погану оптимізацію.

Для дослідження процесів оптимізації відеоігор було обрано демо проект від компанії Unity Technologies який називається 3D Game Kit [5]. Гра розроблена на рушію Unity в жанрі RPG, та цільовою платформою для неї є ПК.

На рис. 40 зображено скріншот ігрового процесу гри. Сюжет гри описує як інженер Еллен здійснила аварійну посадку на таємничій планеті. Її мета дослідити стародавні руїни та перемагати ворогів які їй зустрічатимуться в руїнах невідомої цивілізації.

Рис. 40 Скріншот гри 3D Game Kit

3.2 Аналіз продуктивності і виявлення проблем

Для проведення аналізу використовується ноутбук ACER Predator PH315-51. Короткі характеристики ноутбука:

- Процесор: Intel, Core i5, i5-8300H, 2.30 GHz
- Розширення екрану: 1920 x 1080
- Відеокарта: NVIDIA, GeForce GTX 1060, 6 GB
- Оперативна пам’ять: 16 GB, DDR4 SDRAM
- Операційна система: Windows 11

Почнемо з того що запустимо гру та перевіримо кількість кадрів на секунду. Пройшовши деякий шматок першого рівня, я дійшов до висновку що середнє значення FPS знаходиться в межах 56-58, але в деяких місцях підчас битви значення просідало до 25-30 кадрів за секунду, що говорить про досить погану оптимізацію бойової частини гри. На рис. 41 зображено скріншот з гри з даними поточного FPS і та іншими параметрами графіки що відображають поточне значення оптимізації гри.

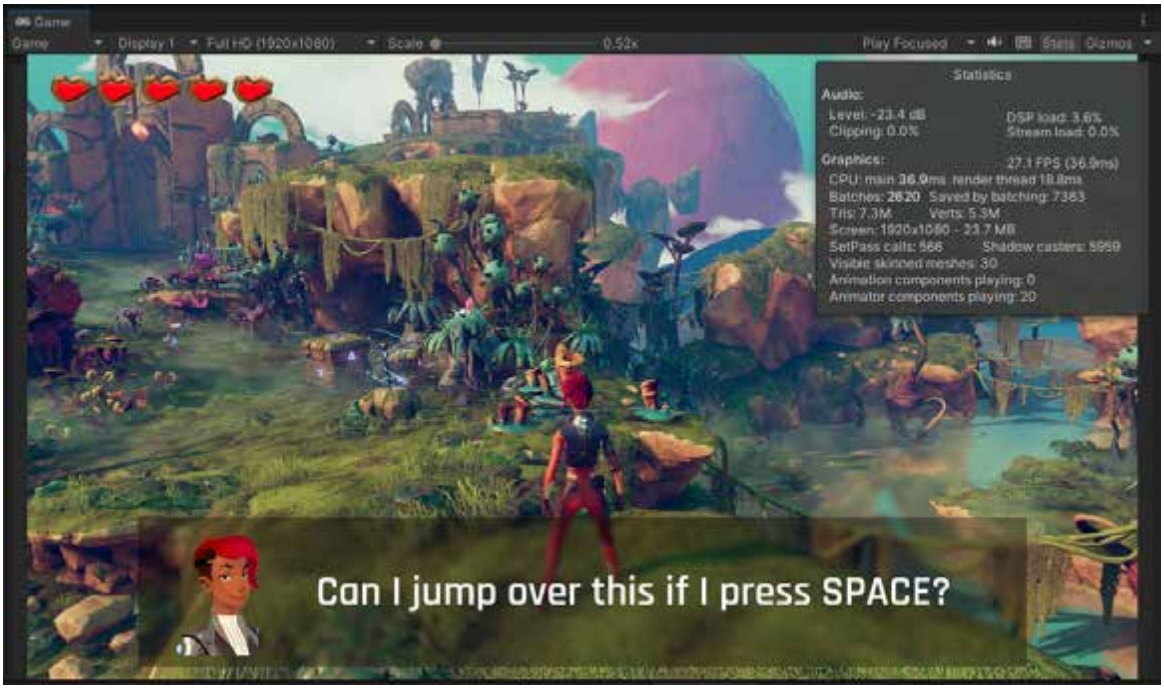

Рис. 41 Скріншот гри з параметрами оптимізації графіки.

3.3 Реалізація оптимізаційних заходів

Occlusion Culling. Почнемо з того що використаємо Occlusion Culling для нашої ігрової сцени. Так як в нас сцена досить великого розміру тому нам не потрібно постійно відмальовувати всі наявні на ній об’єкти. Використання Occlusion Culling дасть нам змогу відмальовувати лише ті об’єкти які знаходяться в полі зору камери.

Першим етапом нам потрібно позначити статичні об’єкти на сцені такі як навколишнє середовище як Occlluder Static та Occludee Static (рис 42).

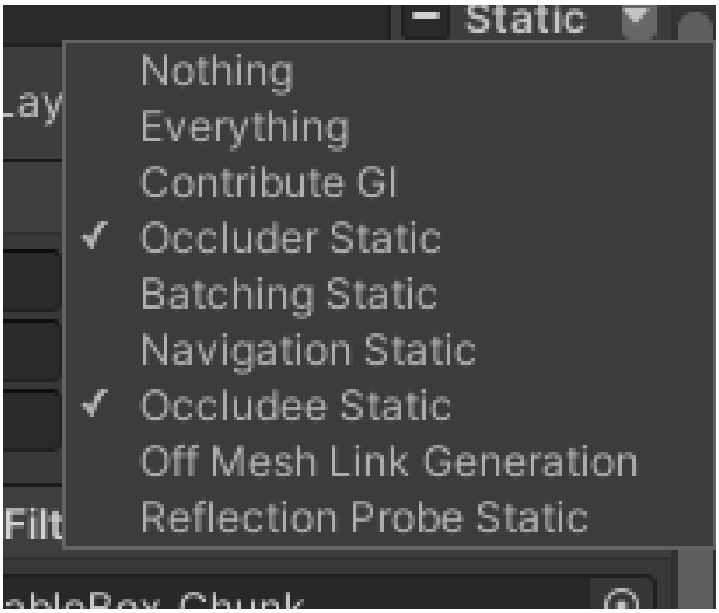

Рис. 42 Позначення для статичних об’єктів як Occlluder Static та Occludee Static.

Наступним етапом йде створення файлу який буде зберігати дані про порядок наших об’єктів на сцені щоб правильно їх відображати. На рис. 43 зображено панель для роботи з Occlusion Culling. Після запікання сцени ми бачимо розмір файлу який було створено, в нашому випадку для запікання нашої сцени файл займає 19,6 Mb. Тепер нам залишається лише перевірити ефективність Occlusion Culling.

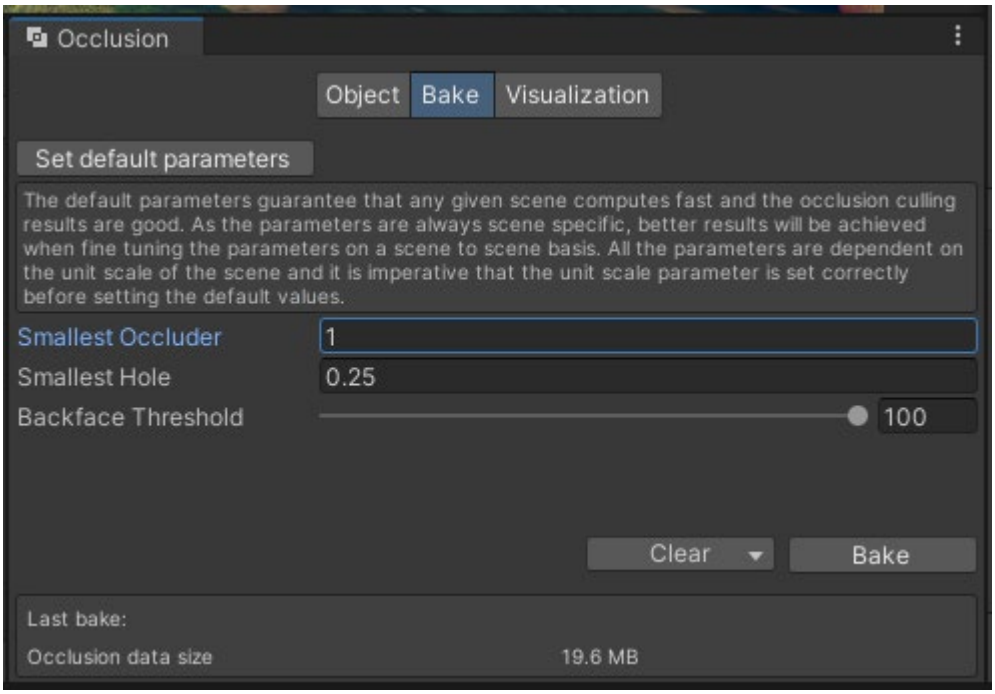

Рис. 43 Вікно для роботи з Occlusion Culling

LOD. Наступною нашою оптимізацією проекту буде додавання LOD. Для цього нам знадобиться додавання до кожного візуального елемента на сцені відповідного компонента LOD Group (рис. 44) та наявність ще двох моделей які мають меншу деталізацію. На певній відстані від камери основна модель буде підмінятися моделлю меншої якості, так як вони будуть знаходитись далеко підміна 3D моделі буде непомітною, а продуктивність відеогри при цьому зросте.

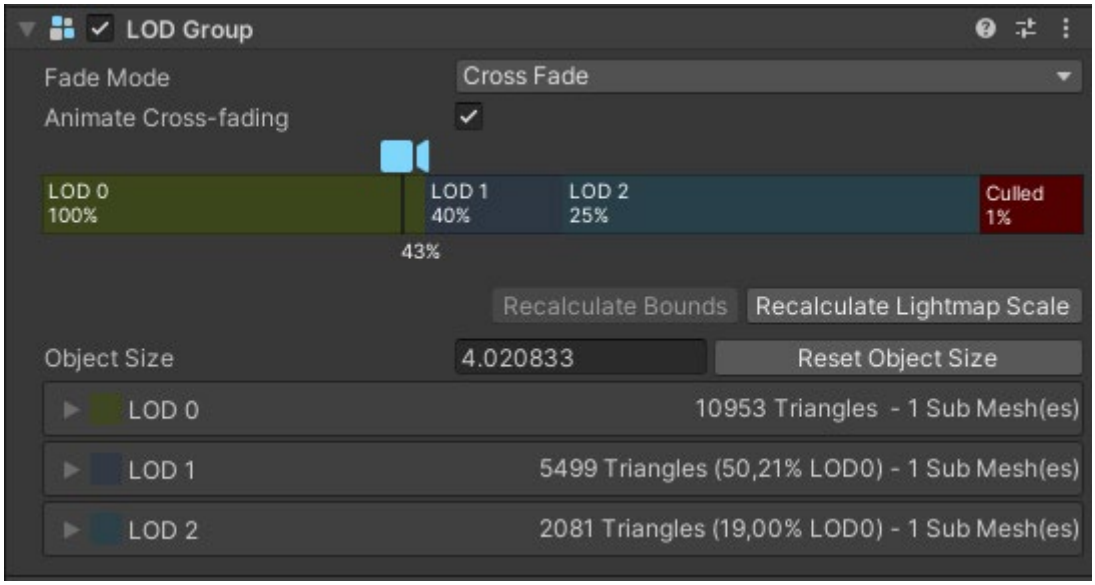

Рис. 44 Компонент LOD Group

Так як всі моделі об’єктів знаходяться в префабах, тому нам потрібно обновити лише їх, а всі об’єкти на сцені які дублюються з префабів автоматично підтягнуть ці зміни.

Batching. Для зменшення навантаження на графічний процесор підключимо до нашого проекту батчинг. Для цього нам потрібно перейти в Project Settings на вкладку Player, тут вибираємо розділ Other Settings і вмикаємо Static Batching і Dynamic Batching (рис. 45)

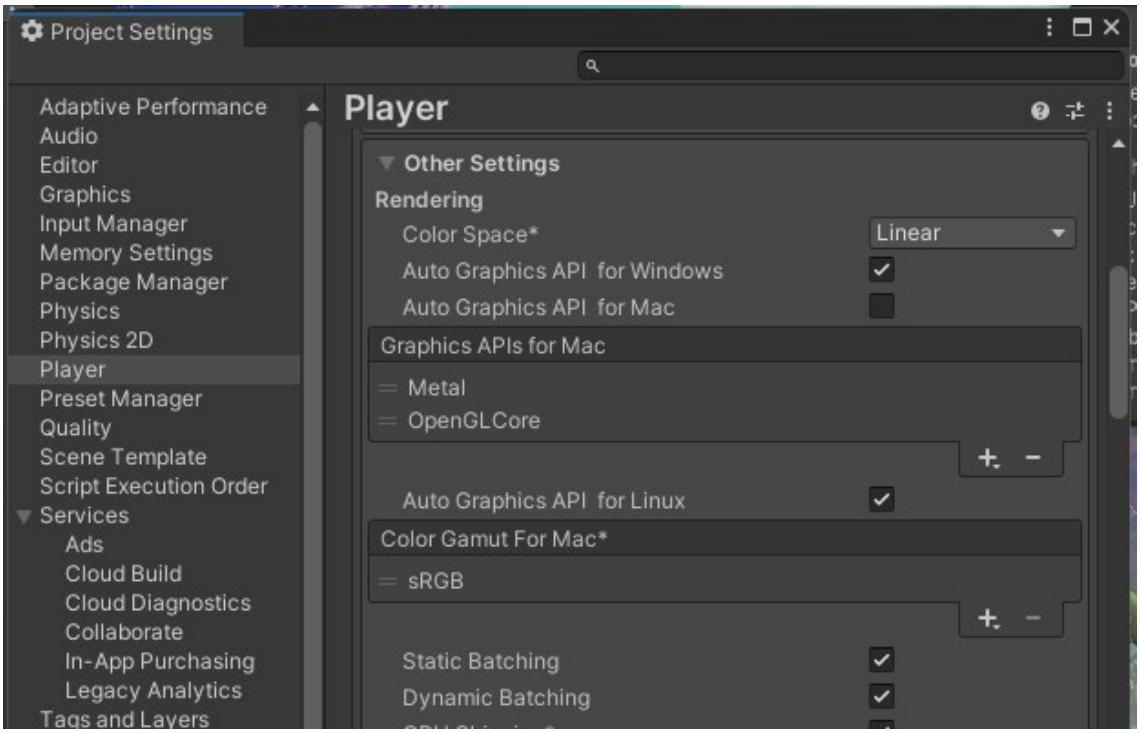

Рис. 45 Налаштування батчингу в Unity

Наступним кроком є позначення всіх нерухомих об’єктів на сцені як Static. Це потрібно для того щоб програма розділяла рухому і нерухомі об’єкти і використовувала на них відповідний батчинг Так як статичний батчинг є більш продуктивним, важливо як найбільше об’єктів позначити як Static. На рис. 46 показано приклад префабу позначеного як Static (верхній правий кут).

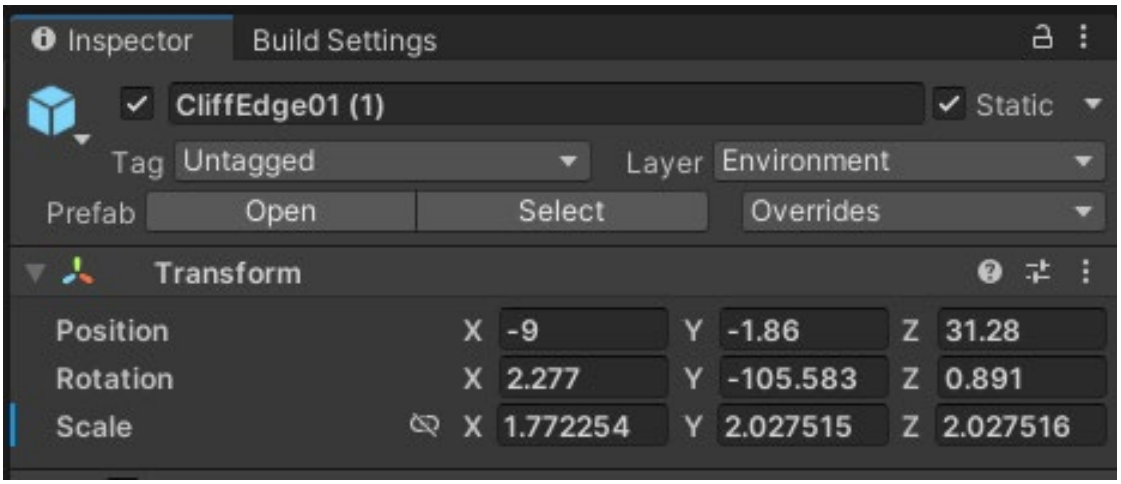

Рис. 46 Приклад префабу позначеного як Static

Також потрібно поставити галочку в панелі яка знаходиться справа від Static, щоб включити статичний батчинг для цього об’єкта (рис. 47)

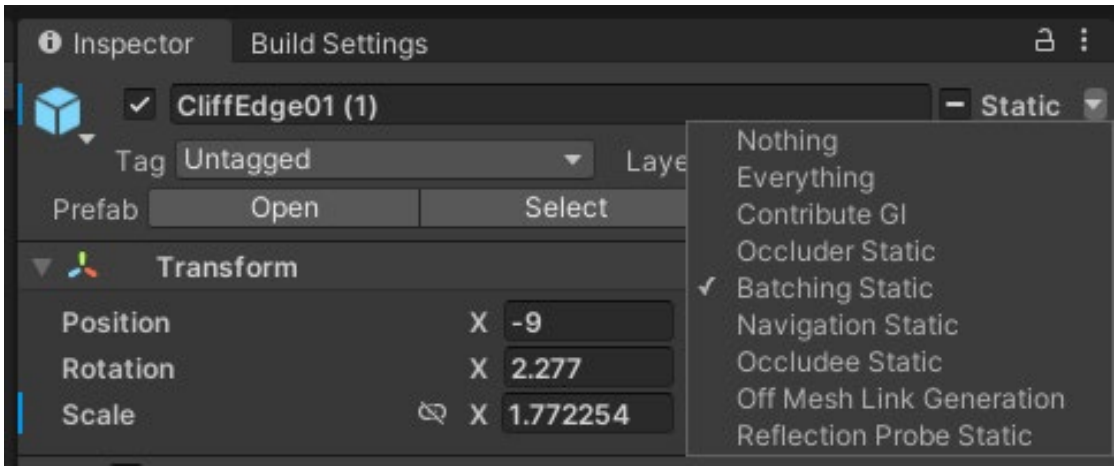

Рис. 47 Включення статичного батчингу на об’єкті

Для підключення динамічного батчингу нам потрібно налаштувати матеріали об’єктів. Відкривши налаштування матеріалу нам потрібно знайти пункт Enable GPU Instancing та включити його (рис. 48). Тепер всі не статичні об’єкти з однаковим матеріалом при певних умовах будуть одночасно батчиться, що сильно зменшить кількість викликів відмалювання.

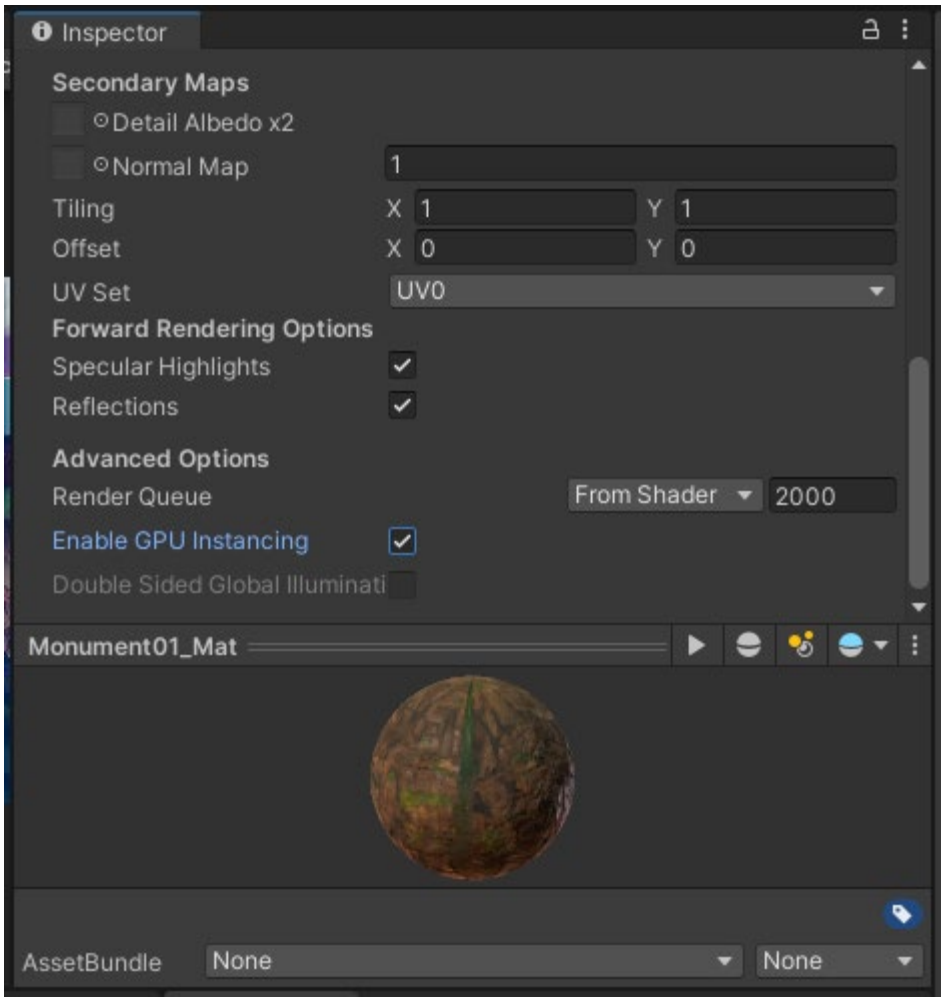

Рис. 48 Приклад матеріалу з включеним Enable GPU Instancing

Object Pooling. Слабким місцем у коді було створення ворогів які постійно створювалися та видалялися при знищенні, що в свою чергу було створювало сміття яке за допомогою GC доводилося чистити, що час від часу могло викликати підвисання відеогри. Цю проблему було вирішено впровадженням пулів об’єктів. Для цього було написано скрипт який генерував пул об’єктів які зберігалися, а при необхідності їх можна було дістати, після знищення об’єкту у грі він повертався у пул для подальшого перевикористання. Код пулу представлено у додатку.

3.4 Аналіз результатів та оцінка досягнутих покращень

В нашому дослідженні було проведено оптимізацію відеогри засобами Unity. Основна зосередженість оптимізації була на графічній частині проекту, тому що вона дає значний приріст при її оптимізації. На рис. 49 зображено скріншот з не оптимізованої відеогри.

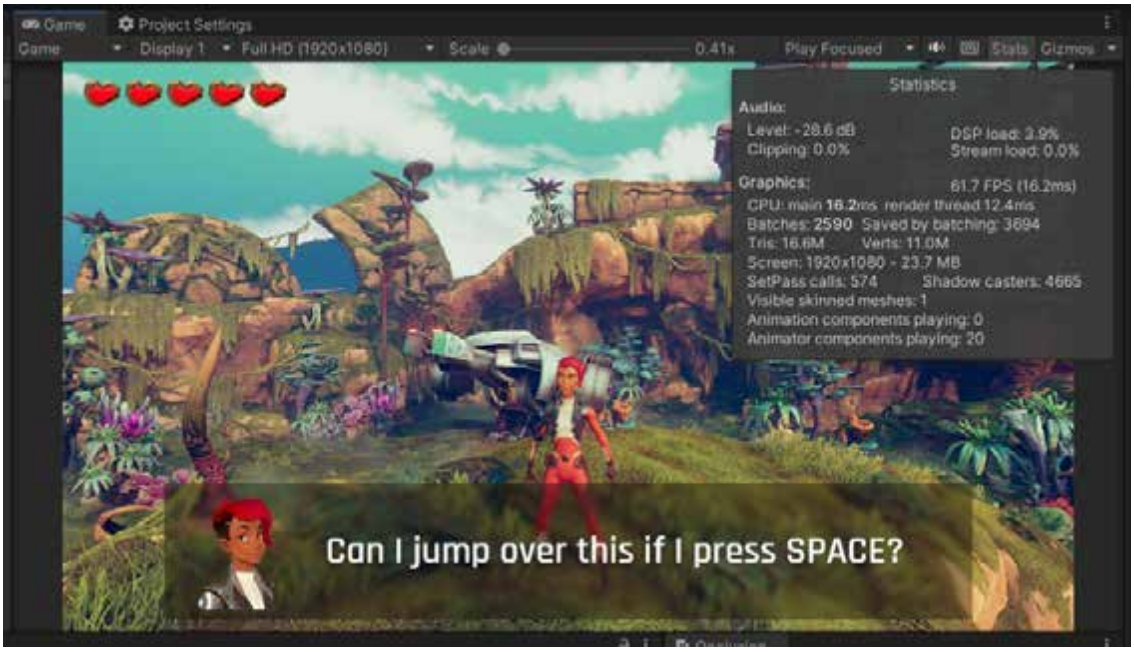

Рис. 49 Скріншот з не оптимізованої відеогри

Перевірка оптимізації була в однакових умовах для отримання як найбільш точних даних. Отримані результати представлені в табл. 1.

Табл. 1 Порівняння результатів оптимізації різними методами.

Тип оптимізації	**FPS**	**Батчі**
Без оптимізації	58-62	2590
Occusion Culling	64-66	2441
LOD	78-80	1350
Батчінг	65-68	1430
Object Pooling	65-67	2590
Об’єднана оптимізація	82-86	971

Оптимізація за допомогою Occusion Culling дала приріст в приблизно 2-8 FPS та зменшилась кількість батчів приблизно на 100-150, що є хоч і не значним, але приростом продуктивності. Зважаючи на те що ноутбук для тестування було обрано з доволі потужними характеристиками тому приріст не є великим, але для слабших ПК приріст може бути більшим.

Оптимізація за допомогою батчінгу підняла продуктивність не оптимізованого проекту трішки вище ніж Occusion Culling, що склало 3-10 FPS. Натомість кількість батчів впала значною мірою аж приблизно на 1000, що і було основною причиною оптимізації цим способом.

Най продуктивнішим виявився спосіб оптимізації за допомогою LOD. Оптимізація цим способом підвищила продуктивність гри на 16-20 FPS, та знизило кількість батчів на приблизно 1100. Головна причина такої потужної оптимізації полягає в тому, що більшість моделей в безпосередній близькості від камери мають досить деталізований вигляд, і чим далі ми від них відходимо нам стає складно їх детально розглянути і в цей момент 3D модель підміняється простішою, що не помітно для самого гравця. В порівнянній з Occusion Culling, ми можемо бачити перед собою багато об’єктів, які будуть знаходитися перед нами в полі зору, а Occusion Culling виключає об’єкти лише ті що ми не бачимо томі він мало ефективний для відкритих локацій і дуже ефективний у закритих кімнатах або тунелях. І протилежним є LOD, ми можемо бачити перед собою дуже багато об’єктів, але через те що вони можуть знаходитися далеко ми будемо бачити лише силует об’єкта, тобто спрощену 3D модель, що дуже слабо навантажуватиме наш ПК. Тому най ефективнішим рішенням буде комбінування даних підходів, що дасть максимальний приріст продуктивності для нашої відеогри.

Об’єднавши всі три способи оптимізації ми отримали максимальний приріст продуктивності. Деякі способи показали себе краще в поточних умовах, деякі гірше, але збільшення продуктивності досягається комбінуванням усіх

можливих способів. На рис. 50 зображено скріншот вже оптимізованого проекту засобами Unity.

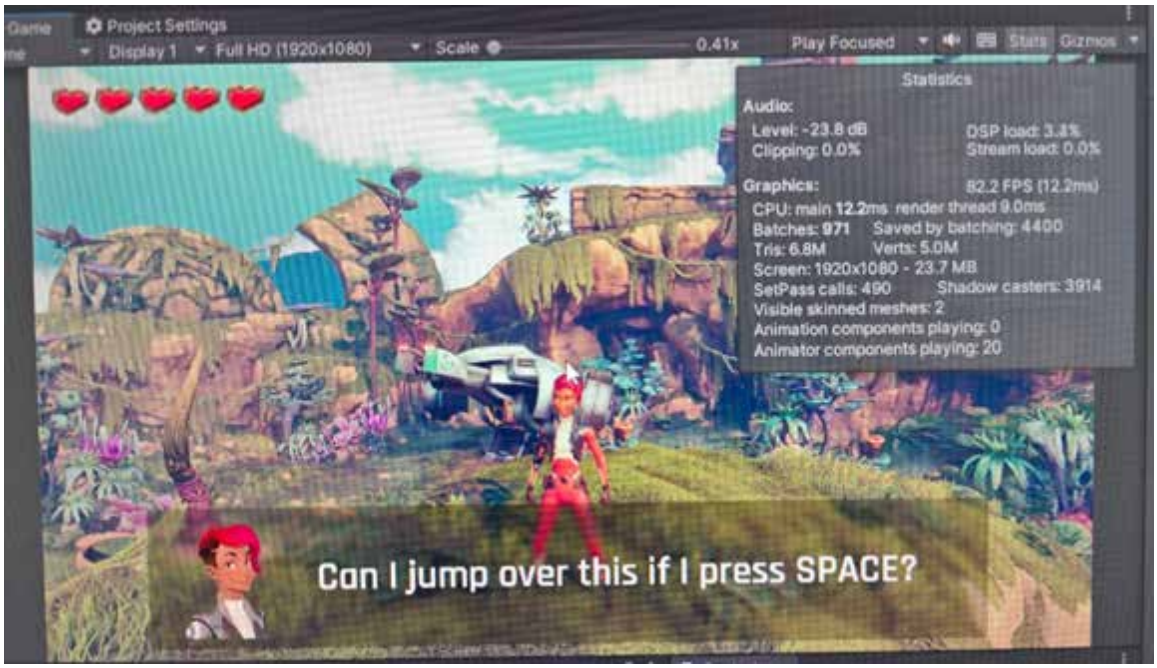

Рис. 50 Скріншот оптимізованої відеогри.

Використання пулів об’єктів не сильно покращили продуктивність відеогри, але зменшили кількість падінь FPS через те що об’єкти не створюються кожного разу нові, а перевикористовуються вже створені.

ВИСНОВОКИ

Оптимізація це важлива складова будь-якого відеоігрового проекту. Хороша оптимізація дозволяє гравцям проникнути в світ який створили автори, та відчути нові емоції. З іншого боку розробникам та інвесторам важливий успіх відеогри через те, що вони вклали туди гроші та сили і щоб отримати прибуток та мати можливість продовжувати створювати нові віртуальні світи. А погана оптимізація може зробити це не можливим.

Дослідивши інструменти та технології рушію Unity для оптимізації відеоігор було визначено ефективні методи та підходи для оптимізації відеоігор. Більшість підходів для оптимізації можуть бути використані не лише на рушію Unity але і на інших рушіях.

В першому розділі ми розглянули відеоігрову індустрію та важливість її у сучасному світі, вплив на сучасну економіку та охоплення аудиторії. Відеоігри мають дуже широкий розмах аудиторії.

Наступним кроком було розглянуто основні принципи оптимізації відеоігор, які існують метрики для оцінювання характеристик оптимізації та як їх розуміти.

Отримавши базове розуміння в оптимізації відеоігор в другому розділі ми переходимо до дослідження інструментів та технологій які використовуються безпосередньо в Unity.

В першій частині другого розділу розглянуто рушій Unity як платформу для розробки відеоігор, його особливості та переваги, такі як кросплатформність, велике ком’юніті, що в свою чергу дає велику кількість матеріалів для навчання та велику кількість вирішених завдань, потужний графічний рушій, фізичний рушій, робота з аудіо та візуальними ефектами та можливість розширення інструментів самими розробниками відеоігор.

В другій частині другого розділу розглянуто інструменти які використовуються розробниками для того, щоб знайти проблеми з оптимізацією в проекті. Цими інструментами є Profiler та Frame Debugger, перший відображує графіки використання ресурсів пристрою а другий використовується для отримання даних про відмалювання кожного кадру відеогри.

В третій частині другого розділу буди розглянуті інструменти та підходи для оптимізації графіки відеоігор на рушію Unity. До основних відходів відноситься Level of Detail, Occlusion Culling та Batching. Level of Detail – це використання моделей з різною деталізацією в залежності від відстані до камери, з метою зменшення навантаження на відмалювання сцени. Occlusion Culling – це вимикання відображення об’єктів які не потрапляють в поле зору камери. Batching – це оптимізація відмалювання статичних та динамічних об’єктів шляхом об’єднання їх у пакети що відмальовуються за один раз.

В четвертій частині було досліджено способи оптимізації фізики за допомогою рушію. Вона полягає у правильному виборі колайдера для фізичних об’єктів що знаходяться на сцені та взаємодіють між собою. Розглянуто способи зменшення навантаження на процесор шляхом зменшення кількості обчислень фізики та як на це впливає матриця колізій. Досліджено чому краще використовувати прості моделі колайдерів при роботі з променями та на прикладі показано на скільки це може бути ефективно для оптимізації. Розглянуто як та навіщо використовувати Rigidbody та які переваги він надає. Також було визначено як часто вираховується використання фізики та які параметри на це впливають.

В третьому розділі було використано на практиці інструменти та методи оптимізації відеоігор. Для цього було обрано відеогру-кандидата над яким проводилось впровадження оптимізаційних заходів.

В першій частині третього розділу описано вибір відеогри для оптимізації, яким стала демо гра від розробників Unity яка називається 3D Game Kit. Вона

розроблена для розробників та демонструє приклад за допомогою якого можна розробляти подібні ігри. Дана гра має жанр RPG.

В другій частині було проведено аналіз продуктивності та виявлення місць що потрібно оптимізувати в даному проекті. Найбільш пріоритетним місцем для оптимізації виявилась графіка проекту.

В третій частині було проведено впровадження оптимізаційних заходів у проект. Серед впроваджених заходів були Level of Detail, Occlusion Culling та Batching.

В четвертій частині після оптимізації графіки було проведено ряд тестів для виявлення результатів отриманих в ході оптимізації. Тести проводились в максимально ідентичних умовах, але варто враховувати присутність похибки в 2-3% в отриманих результатах. Оптимізація за допомогою Occusion Culling дала приріст в приблизно 2-8 FPS та зменшилась кількість батчів приблизно на 100-150, що є хоч і не значним, але приростом продуктивності. Оптимізація за допомогою батчінгу підняла продуктивність не оптимізованого проекту трішки вище ніж Occusion Culling, що склало 3-10 FPS. Натомість кількість батчів впала значною мірою аж приблизно на 1000, що і було основною причиною оптимізації цим способом. Най продуктивнішим виявився спосіб оптимізації за допомогою LOD. Оптимізація цим способом підвищила продуктивність гри на 16-20 FPS, та знизило кількість батчів на приблизно 1100.

Оптимізація важлива частина розробки будь-якого проекту, і від якості якої може напряму залежати успіх відеогри. Дослідивши та впровадивши оптимізаційні заходи в тестовий проект можна з впевненість говорити про їхню необхідність. Отримані результати від проведення всіх оптимізаційних заході дали досить серйозний приріст в продуктивності відеогри, що в свою чергу покращить відеоігровий досвід гравців та збільшить впевненість розробників у якості свого відеоігрового продукту.

СПИСОК ЛІТЕРАТУРИ

1. Подскребко, О. С., & Іванченко, Н. (2021). АНАЛІЗ РИНКУ ЦИФРОВИХ ВІДЕОІГОР ТА ЙОГО ВПЛИВ НА ЕКОНОМІКУ. *Економічний простір*, (175), 130-135. https://doi.org/10.32782/2224-6282/175-24
2. Harbuzova A. Що чекає на ігрову індустрію: прогнози й дослідження Google. ДОУ. URL: https://gamedev.dou.ua/news/global-game-industry-google-repor t-2021/ (дата звернення: 12.10.2023)
3. Chris Dickinson Unity 5 Game Optimization : навч. посіб. 1st edition, Birmingham : Packt Publishing, 2015, 296с.
4. Unity - Manual: Introduction to collision, URL: https://docs.unity3d.com/2021.3/Documentation/Manual/CollidersOverview.html (дата звернення: 14.10.2023)
5. Unity Technologies - 3D Game Kit. URL: https://assetstore.unity.com/packages/templates/tutorials/unity-learn-3d-game-kit-115747 (дата звернення: 15.10.2023)
6. Doran, John P.., Zucconi, Alan. Unity 2018 Shaders and Effects Cookbook: Transform Your Game Into a Visually Stunning Masterpiece with Over 70 Recipes, 3rd Edition. Велика Британія: Packt Publishing, 2018, 392с.
7. Sanjay Madhav Game Programming Algorithms and Techniques: A PlatformAgnostic Approach – Crawfordsville, Indiana: R.R Donnelay, 2013. – 350 с.
8. Artificial Intelligence for Games. Millington Ian, Funge John. 2009. – 896с.
9. Game Engine Architecture 2nd Edition. Jason Gregory. 2014 – 1052с.
10. Матвєєв, Д., & Лановий, О. (2020). ПРОБЛЕМИ ОПТИМІЗАЦІЇ ГРАФІКИ ПІД ПРИСТРОЇ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ. ΛΌГΟΣ.

ОНЛАЙН. вилучено із https://ojs.ukrlogos.in.ua/index.php/2663-4139/article/view/5033 (дата звернення: 18.10.2023)

11. Реутская Ю.Ю., Новиченко А.А. " Продуктивність та оптимізація програм. Популярні алгоритми " Вісник Національного технічного університету України Київський політехнічний інститут. Серія: Радіотехніка. Радіоапаратобудування, №. 41, 2010, 137-147с
12. Unity - Manual: Graphics. Unity - Manual: Unity User Manual 2022.3 (LTS). URL: https://docs.unity3d.com/Manual/Graphics.html (дата звернення: 19.10.2023).

ДОДАТОК А