НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету інформаційних технологій	Завідувач кафедри комп'ютерних наук
Болбот І.М., д.т.н., професор	Голуб Б.Л., к.т.н., доцент
_____	_____
____ _____ 2024 р.	**____ _____ 2024 р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему *«Інтелектуальна система оптимізації ігрового процесу на платформі Unity»*

Спеціальність *122 «Комп'ютерні науки»*
(код і назва)

Освітня програма *«Інформаційні управляючі системи і технології»*
(назва)

Орієнтація освітньої програми *магістр*
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

кандидат технічних наук, доцент _____ *Голуб Белла Львівна*
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

кандидат технічних наук, доцент _____ *Сватко Віталій Володимирович*
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ *Превор Максим Васильович*
(підпис) (ПІБ студента)

КИЇВ-2024

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) *інформаційних технологій*

ЗАТВЕРДЖУЮ
Завідувач кафедри_____

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"____" _____20_____року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Превору Максиму Васильовичу
(прізвище, ім'я, по батькові)

Спеціальність *122 «Комп'ютерні науки»*
(код і назва)

Освітня програма *«Інформаційні управляючі системи і технології»*
(назва)

Орієнтація освітньої програми *магістр*
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи
«Інтелектуальна система оптимізації ігрового процесу на платформі Unity»

затверджена наказом ректора НУБіП України від "____"_____20___р. №_____

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи:

1. *Платформа Unity з використанням інструменту Unity Adaptive Performance;*
2. *Оптимізація ігрового додатку на основі аналізу використаних ресурсів;*
3. *Розробка гнучкого вікна налаштувань та покращення інтелектуальної системи;*
4. *Тестування оптимізації на пристроях Android.*

Перелік питань, що підлягають дослідженню:

1. *Аналіз існуючих методів оптимізації продуктивності;*
2. *Огляд існуючих рішень на рушії Unity та її конкурентів;*
3. *Проведення досліджень на створеному ігровому додатку;*
4. *Висновки.*

Перелік графічного матеріалу (за потреби) *постер; презентація — 15 слайдів*

Дата видачі завдання "_____"_____20___ р.

Керівник магістерської кваліфікаційної роботи _____ *Сватко В.В.*
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ *Превор М.В.*
(підпис) (прізвище та ініціали студент

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ААА-проект — це ігри, які розроблені великою компанією і мають велике фінансування

CPU — Central Processing Unit (центральний процесор)

GPU — Graphics Processing Unit (графічний процесор)

RAM — Random Access Memory (оперативна пам'ять)

VRAM — Video Random Access Memory (відеопам'ять)

LOD — Level of Detail (рівень деталізації), техніка оптимізації, що зменшує кількість полігонів у моделях залежно від відстані до камери

Frustum Culling — техніка рендерингу, яка вимикає об'єкти, що не потрапляють у поле зору камери, для зменшення навантаження на GPU

Occlusion Culling — техніка рендерингу, яка вимикає об'єкти, закриті іншими, з метою уникнення рендерингу непотрібних елементів

Unity Adaptive Performance — інструмент Unity для динамічної адаптації продуктивності гри

Object Pooling — техніка оптимізації, яка передбачає повторне використання об'єктів для зменшення витрат на їх створення та знищення

Mesh Combining — техніка об'єднання кількох моделей у одну для зменшення кількості запитів до GPU

Light Baking — техніка попереднього обчислення освітлення сцени для зменшення навантаження на процесор у реальному часі

FPS — Frames Per Second (кадрів на секунду)

VR — Virtual Reality

AR — Augmented Reality

ВСТУП

Актуальність теми. Ігрова індустрія є однією з найбільш динамічних і швидкозростаючих галузей сучасної економіки. За останні роки вона демонструє стабільне зростання, а її обсяг оцінюється в понад 300 млрд доларів із щорічним приростом на 8-10%.

Розробка відеоігор перетворилася на високотехнологічний процес, який потребує залучення передових інформаційних технологій та інструментів. Однією з ключових проблем, з якою стикаються розробники ігор, є забезпечення оптимальної продуктивності та користувацького досвіду на різноманітних апаратних платформах. Ігри повинні ефективно використовувати доступні апаратні ресурси, забезпечуючи при цьому високу якість графіки, плавність анімації та мінімальний час відгуку. Вирішення цієї проблеми потребує розробки інтелектуальних систем оптимізації ігрового процесу, здатних адаптуватися до особливостей конкретних платформ. Серед популярних ігрових платформ особливе місце займає Unity - кросплатформенний ігровий рушій, який широко використовується для розробки 2D та 3D ігор. Unity пропонує потужні інструменти для створення інтерактивних додатків, але при цьому вимагає ретельного налаштування параметрів продуктивності для досягнення оптимальних результатів.

Тому розробка інтелектуальної системи оптимізації ігрового процесу саме для платформи Unity є актуальним і практично значущим завданням.

Мета і завдання дослідження. Метою цієї роботи є підвищення ефективності та якості ігрового процесу на платформі Unity шляхом використання існуючих інструментів адаптивної оптимізації, таких як Unity Adaptive Performance, та покращення їх для забезпечення стабільної продуктивності на різних апаратних конфігураціях, зокрема на мобільних пристроях.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Провести аналіз предметної області, дослідити особливості ігрової індустрії та підходи до оптимізації продуктивності;
- Дослідити існуючі підходи та рішення в області оптимізації ігрового процесу, зокрема Unity Adaptive Performance;
- Впровадити систему адаптивної оптимізації продуктивності;
- Розробити вікно налаштувань для динамічного керування продуктивністю гри;
- Покращити існуючі інструменти адаптивної оптимізації та інтегрувати нові рішення для підвищення ефективності;
- Провести тестування запропонованих рішень на Android-пристроях та оцінити їх ефективність на прикладі розробленої гри.

Об’єкт дослідження — процес розробки та оптимізації ігрового процесу на платформі Unity.

Предмет дослідження — методи, моделі та засоби адаптивної оптимізації продуктивності ігрового процесу з використанням Unity Adaptive Performance.

Методи дослідження. Для вирішення поставлених завдань використовувалися методи системного аналізу, об'єктно-орієнтованого проєктування, математичного моделювання та експериментального тестування. Практична частина реалізована з використанням мови програмування C# та інструментів платформи Unity.

Наукова новизна одержаних результатів полягає у дослідженні, використанні та вдосконаленні інтелектуальної системи адаптивної оптимізації ігрового процесу на основі реального часу. Система дозволяє автоматично налаштовувати ключові параметри рушія Unity залежно від ресурсів пристрою.

Практичне значення роботи визначається можливістю використання запропонованого підходу для підвищення продуктивності ігор на платформі Unity. Це забезпечить стабільну роботу ігор на різних апаратних конфігураціях, зокрема на мобільних пристроях, з мінімальними витратами часу на оптимізацію.

Структура та обсяг роботи. Пояснювальна записка складається з чотирьох основних розділів.

У першому розділі проведено аналіз ігрової індустрії, визначено проблеми продуктивності в іграх та методи їх вирішення, а також сформульовано завдання дослідження.

У другому розділі розглядаються основні методи оптимізації, включаючи графічні підходи, оптимізацію обчислювальних ресурсів та пам'яті. Також проводиться порівняння інструментів доступних у Unity та інших платформах.

Третій розділ присвячений процесу інтеграції Unity Adaptive Performance, створення гнучкого вікна налаштувань та розробку кастомних скейлерів для покращення продуктивності.

У четвертому розділі проведено тестування системи на Android-пристроях, проаналізовано отримані результати та запропоновано рекомендації щодо використання розробленої системи.

Основна частина пояснювальної записки займає 77 сторінок, із них 53 сторінок основного тексту. Додатки мають обсяг в 13 сторінок. Робота містить 48 рисунків і 2 таблиці. Для виконання дослідження та підготовки матеріалів використано 26 джерел інформації.

Апробація. За результатами роботи були опубліковані тези під назвою «Інтелектуальна система оптимізації ігрового процесу на платформі Unity» в збірнику наукових праць за матеріалами XV Міжнародної науково-практичної конференції молодих вчених «Інформаційні технології: економіка, техніка, освіта» за 7-8 листопада 2024 року.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальний опис ігрової індустрії

Ігрова індустрія - це сектор економіки, який включає в себе розробку, маркетинг та монетизацію відеоігор. Ця галузь охоплює десятки різних професій і використовує роботу сотень тисяч людей по всьому світу [1].

Історія відеоігор починається ще в 1940-х роках, коли з'явилися перші комп'ютерні ігри. Проте розвиток ігрової індустрії як повноцінного бізнесу почався в 1970-х з появою перших ігрових приставок та аркадних автоматів. Головну роль у популяризації відеоігор зіграли такі продукти як Pong (1972), Space Invaders (1978) та Pac-Man (1980) [2].

У 1980-х роках відбувся крах ігрової індустрії в Північній Америці через перенасичення ринку неякісними іграми для Atari 2600. Проте в Японії в цей час компанія Nintendo випустила революційну приставку Famicom яка відродила інтерес до відеоігор. Наприкінці 1980-х - на початку 1990-х років з'явилися культові ігри Mario, The Legend of Zelda, Sonic, Mortal Kombat та інші, які сформували сучасне обличчя індустрії [3].

З середини 1990-х відеоігри перейшли в 3D графіку завдяки таким приставкам як PlayStation, Nintendo 64 та Sega Saturn. Це відкрило нові горизонти для ігрового дизайну та візуального оформлення. На початку 2000-х широке розповсюдження отримали онлайн ігри, особливо багатокористувацькі: World of Warcraft, Lineage.

Сьогодні ігрова індустрія є однією з найбільш швидкозростаючих і прибуткових галузей розваг. Її обсяг оцінюється в понад 300 млрд доларів із щорічним зростанням на 8-10%. Основними сегментами ринку є мобільні ігри (49%), ігри для ПК (22%) та консольні ігри (28%) (рис. 1). Лідерами індустрії є такі компанії як Tencent, Sony, Microsoft, Nintendo, Activision Blizzard, Electronic Arts [4].

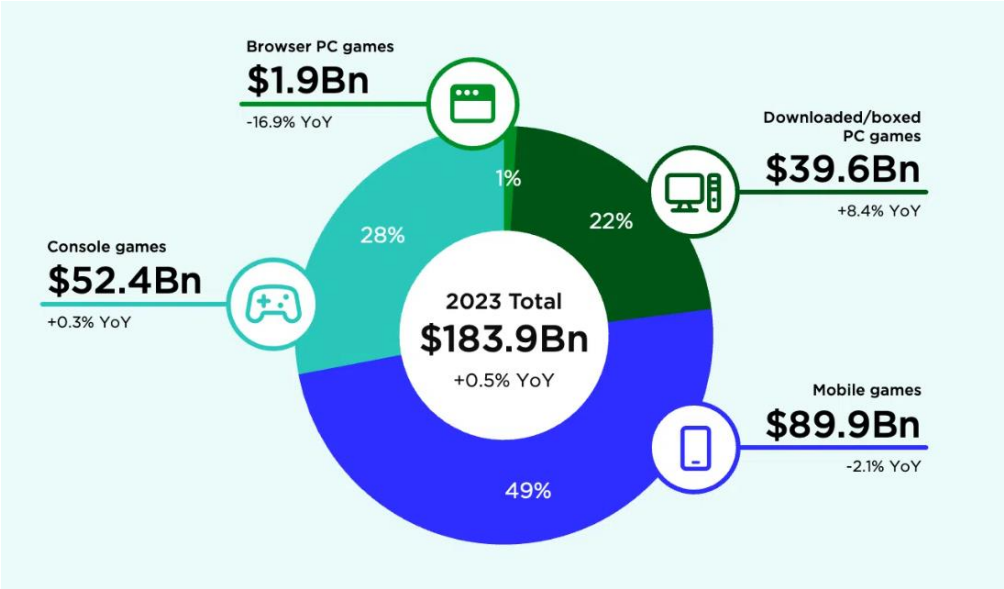

Рис. 1 Світовий ринок ігор за платформами на 2023 рік

Розробка сучасних відеоігор - це складний високотехнологічний процес, який потребує роботи великих команд професіоналів: програмістів, дизайнерів, художників, сценаристів, композиторів, менеджерів та інших. Створення ААА-проектів може тривати 3-5 років і коштувати сотні мільйонів доларів. Все більшу роль у розробці відіграють передові технології - 3D-моделювання, захоплення руху, процедурна генерація, штучний інтелект, віртуальна реальність тощо.

Ключовими трендами розвитку сучасної ігрової індустрії є:

- Зростання мобільного та казуального сегментів за рахунок спрощення ігрового процесу та безкоштовних моделей монетизації;
- Розвиток хмарних ігрових сервісів та стрімінгу, які дозволяють грати без потужного обладнання;
- Посилення кіберспортивної складової, проведення масштабних турнірів з багатомільйонними призовими фондами;
- Використання ігор не тільки для розваг, але й в освіті, медицині, науці, бізнесі (гейміфікація);
- Розробка ігор з відкритими світами, нелінійними сюжетами, процедурною генерацією контенту.

Таким чином, ігрова індустрія пройшла довгий шлях розвитку від простих аркадних ігор до технологічно просунутих інтерактивних розваг, які поєднують

у собі риси мистецтва, спорту та бізнесу. Відеоігри стали невід'ємною частиною сучасної масової культури та одним з основних способів проведення дозвілля для мільйонів людей різного віку по всьому світу.

1.2 Поняття «оптимізації»

Оптимізація в ігровій індустрії — це комплексний процес удосконалення продуктивності гри для забезпечення плавної і стабільної роботи на різних платформах та пристроях. Основна мета оптимізації полягає в тому, щоб зробити гру доступною для якомога ширшого кола користувачів, незалежно від їхнього апаратного забезпечення, зберігаючи при цьому високу якість графіки та задовільний ігровий досвід.

У сучасних іграх користувачі мають високі очікування щодо якості зображення, плавності анімацій і швидкості реакції гри. Однак ці параметри можуть суттєво змінюватися залежно від характеристик пристрою, на якому запускається гра. Оптимізація допомагає вирішити ці проблеми, забезпечуючи баланс між якістю ігрового процесу та вимогами до апаратних ресурсів, таких як графічний процесор, центральний процесор і оперативна пам'ять[5].

Оптимізація є невід'ємною частиною процесу розробки будь-якої гри. Навіть якщо гра відзначається високою якістю візуальних ефектів та інноваційним ігровим процесом, її успіх на ринку може бути під загрозою, якщо вона не буде працювати плавно і стабільно на різних платформах. Погано оптимізовані ігри можуть стикатися з такими проблемами, як низька частота кадрів, довгі часи завантаження, постійні затримки та навіть аварійні завершення роботи гри. Ці проблеми можуть негативно вплинути на досвід користувачів, що, у свою чергу, призведе до негативних відгуків, зниження продажів і втрати аудиторії.

Оптимізація стає ще більш актуальною у зв'язку з різноманітністю платформ, на яких гра може бути запущена. Це можуть бути мобільні телефони, настільні комп'ютери, ігрові консолі або віртуальні середовища. Кожна платформа має свої обмеження щодо апаратного забезпечення, тому розробники

повинні адаптувати гру для кожної з них, забезпечуючи максимальну продуктивність у межах доступних ресурсів. Наприклад, для мобільних пристроїв, які мають обмежені можливості порівняно з ПК, оптимізація особливо потрібна, оскільки перевантаження процесора або пам'яті може призвести до швидкого розряду батареї або перегріву пристрою [5].

Оптимізація ігор включає кілька основних аспектів, кожен з яких потребує особливого підходу для досягнення максимальної продуктивності:

- **Графіка.** Зниження ресурсномісткості візуальних ефектів без значної втрати якості;
- **Обчислювальні ресурси**. Ефективне використання процесорних потужностей для виконання задач гри, таких як фізика, штучний інтелект та логіка ігрового процесу;
- **Пам'ять**. Управління оперативною пам'яттю та відеопам'яттю для уникнення перевантаження системи, що може призвести до затримок або збоїв у роботі.

1.3 Проблеми продуктивності в іграх та шляхи їх вирішення

Продуктивність є дуже важливим аспектом успішного функціонування ігор, оскільки від неї залежить загальний користувацький досвід. Недостатня продуктивність може призвести до таких проблем, як зниження частоти кадрів, довгі часи завантаження, затримки та крахи під час ігрового процесу. Основними факторами, що впливають на продуктивність, є робота графічного процесора, центрального процесора та оперативної пам'яті.

Зниження частоти кадрів є однією з найпоширеніших проблем продуктивності. Низький FPS зазвичай викликаний перевантаженням графічного процесора, що не справляється з обробкою великої кількості полігонів, текстур або ефектів. Це призводить до того, що гра виглядає нестабільно, а її геймплей стає переривчастим. Така ситуація часто виникає через надмірне використання ресурсномістких графічних ефектів або погано оптимізовані моделі об'єктів [6].

Перевантаження центрального процесора також є частою проблемою, особливо коли процесор має обробляти занадто багато одночасних завдань. CPU відповідає за фізику, штучний інтелект та логіку гри, і якщо ці процеси не оптимізовані, то продуктивність гри значно знижується. Це може призвести до затримок у виконанні дій гравця та навіть до зависань гри.

Ще однією важливою проблемою є неефективне управління пам'яттю. Неправильне використання оперативної пам'яті або відеопам’яті може призвести до нестачі ресурсів для завантаження текстур, моделей та інших активів. Це особливо критично для мобільних платформ, де обмежена кількість пам'яті може призвести до різких знижень продуктивності або вильотів гри.

Довгий час завантаження також є поширеною проблемою, що виникає через погано організоване управління ресурсами гри. Якщо ресурси, такі як текстури, звукові файли та моделі, не оптимізовані або погано стискаються, це призводить до того, що гра завантажується занадто довго, що значно погіршує враження гравців.

Щоб вирішити ці проблеми, розробники застосовують різні підходи до оптимізації гри. Один із найефективніших методів — це оптимізація графіки, що включає зменшення розміру текстур, використання моделей з меншою кількістю полігонів та оптимізацію системи освітлення і тіней. Це дозволяє значно зменшити навантаження на GPU без значних втрат у якості зображення.

Для підвищення продуктивності CPU можна використовувати багатопоточність, що дозволяє розподіляти завдання між кількома ядрами процесора. Це допомагає уникнути перевантаження одного ядра і рівномірно розподілити навантаження, що суттєво покращує швидкість обробки даних і загальну продуктивність гри.

Управління пам'яттю також потребує оптимізації через використання таких підходів, як Object Pooling, що дозволяє повторно використовувати вже завантажені об'єкти, замість їх постійного створення і видалення. Це значно знижує навантаження на пам'ять і забезпечує стабільнішу роботу гри, особливо на мобільних пристроях [7].

Останнім часом все більшої популярності набувають адаптивні системи налаштувань, що автоматично регулюють якість графіки та частоту кадрів залежно від можливостей пристрою. Це дозволяє гравцям насолоджуватися стабільною продуктивністю, навіть якщо їхній пристрій не є найпотужнішим на ринку.

Завдяки цим підходам розробники можуть ефективно вирішувати проблеми продуктивності, забезпечуючи гравцям високоякісний досвід та стабільну роботу гри на різних платформах і пристроях.

1.4 Постановка завдання

У сучасній ігровій індустрії, де користувачі мають доступ до великого вибору ігор на різних платформах, питання оптимізації продуктивності набуває особливої важливості. Успішна гра повинна працювати стабільно як на високопродуктивних пристроях, так і на системах із обмеженими ресурсами, зберігаючи при цьому якість геймплею та графіки. Через це завдання оптимізації стає однією з ключових проблем для розробників.

Основна мета даного дослідження — аналіз існуючих рішень з оптимізації продуктивності в іграх, впровадження інструменту Unity Adaptive Performance у власний проєкт та покращення системи адаптивної оптимізації для досягнення кращої продуктивності на різних пристроях, зокрема на Android-платформах.

Для досягнення цієї мети необхідно вирішити такі завдання:

- Провести аналіз предметної області, зокрема існуючих методів оптимізації продуктивності у ігрових проєктах;
- Дослідити ряд важливих аспекти оптимізації продуктивності, такі як графіка, обчислювальні ресурси та управління пам'яттю;
- Проаналізувати інструмент Unity Adaptive Performance та оцінити його ефективність у контексті покращення продуктивності на мобільних пристроях;
- Впровадити Adaptive Performance у розроблений проєкт.

- Покращити систему адаптивної оптимізації, додавши нові рішення для підвищення продуктивності, зокрема динамічне налаштування рівня графіки та інших параметрів залежно від продуктивності пристрою;
- Провести тестування розробленої системи на Android-пристроях та оцінити ефективність запропонованих нововведень, порівнявши отримані результати з початковими показниками.

Таким чином, завдання даного дослідження полягає у створенні гнучкої та ефективної системи оптимізації ігрового процесу, яка забезпечить стабільну продуктивність на різних пристроях, мінімізуючи затримки та зберігаючи високу якість графіки.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ

2.1 Головні методи оптимізації

Значна кількість оптимізаційних рішень є універсальними для всіх платформ, проте деякі з них залежать від конкретної платформи чи специфікацій апаратного забезпечення. Основні напрямки оптимізації включають оптимізацію графіки, процесорних ресурсів, пам’яті та мережевих ресурсів. Розглянемо ключові методи оптимізації ігор, які використовуються в сучасних ігрових проєктах.

2.1.1 Графічні методи

Level of Detail (LOD) — це метод оптимізації графіки, що дозволяє знижувати кількість обчислюваних полігонів для об'єктів (рис. 2) залежно від їх відстані до камери. При наближенні об'єкта використовується його високополігональна версія, тоді як на віддалених об'єктах використовується спрощена модель з меншою кількістю полігонів. Це є особливо ефективним у великих ігрових світах, де велика кількість об’єктів одночасно з’являються в кадрі. Кожен об’єкт може мати кілька версій з різним рівнем деталізації, що дозволяє оптимізувати навантаження на GPU, зберігаючи якість візуалізації для об'єктів, які найближчі до гравця [8].

Рис. 2 Процес створення LOD групи

LOD застосовується до різних елементів сцени: від персонажів до елементів навколишнього середовища, таких як дерева або будівлі. Важливим аспектом є баланс між якістю та продуктивністю, адже занадто агресивне зниження кількості полігонів може призвести до помітного зниження якості на екрані, особливо якщо LOD-перехід відбувається занадто різко або близько до камери.

Крім полігональних моделей, LOD також можна застосовувати до інших аспектів сцени, наприклад, до текстур, тіней та інших візуальних ефектів. У поєднанні з іншими методами оптимізації, такими як Occlusion Culling та Frustum Culling, LOD дозволяє створювати візуально насичені сцени з високою продуктивністю навіть на слабких пристроях.

Occlusion Culling є технікою оптимізації, яка використовується для уникнення рендерингу об'єктів, що перекриті іншими і не видимі для камери (рис. 3). Якщо передній об’єкт закриває віддалені об’єкти, гра не витрачає ресурси на їх рендеринг. Це суттєво знижує навантаження на GPU в складних сценах, особливо в 3D-іграх із багатьма об’єктами [9]. Occlusion Culling часто застосовується в великих світах, де сцена має багато прихованих об'єктів, наприклад, за стінами або будівлями. Ця техніка є однією з найважливіших в оптимізації ігор, оскільки дозволяє суттєво зменшити кількість обчислень, що виконує графічний процесор.

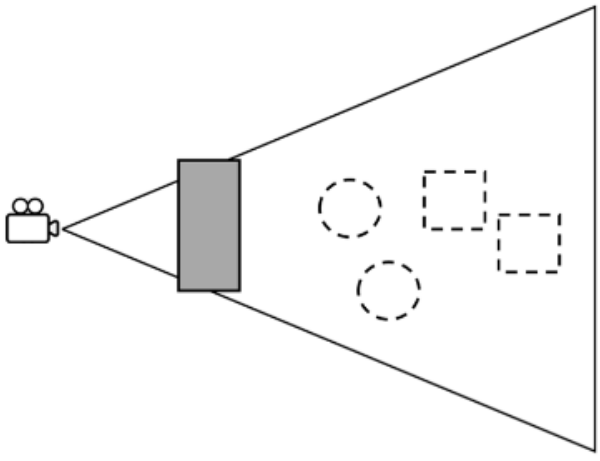

Рис. 3 Приклад роботи Occlusion Culling

Frustum Culling є ще однією популярною технікою, але вона працює трохи по-іншому. Якщо Occlusion Culling вимикає об'єкти, які закриті іншими, то Frustum Culling вимикає об'єкти, які взагалі не потрапляють у поле зору камери (рис. 4). Це дуже ефективний метод для сцен із великим ігровим світом, де камера охоплює лише частину середовища. Об’єкти, що знаходяться поза межами цієї області, не рендеряться, зменшуючи навантаження на систему [9].

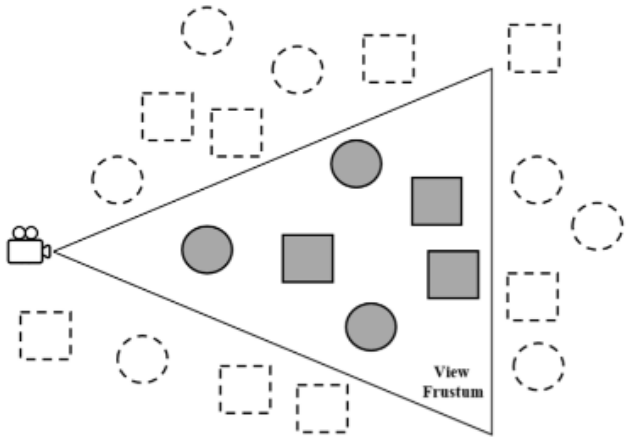

Рис. 4 Приклад роботи Frustum Culling

Запікання світла (Light Baking) — це техніка, за якої освітлення сцени обчислюється один раз, а потім зберігається у вигляді текстур (рис. 5). Такі текстури, або lightmaps, використовуються під час рендерингу, щоб імітувати освітлення, без необхідності повторного обчислення в реальному часі [10]. Це особливо корисно для статичних об'єктів і сцен, де освітлення не змінюється. Основна перевага цієї техніки полягає в тому, що вона дозволяє створити реалістичне освітлення без великого навантаження на графічний процесор. Однак вона не підходить для динамічних сцен, де освітлення часто змінюється.

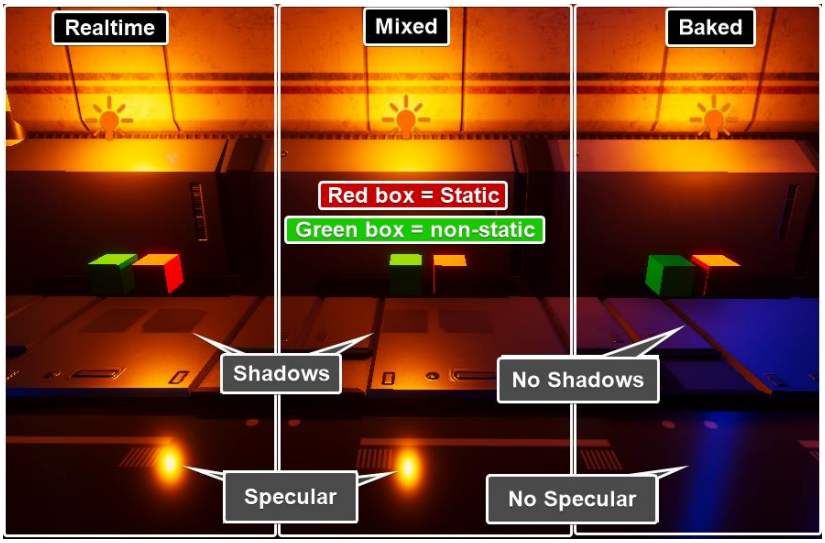

Рис. 5 Порівняння запеченого світла, змішаного та в реальному часі

Меш комбайн (Mesh Combining) дозволяє об'єднувати кілька об'єктів у один (рис. 6) для зменшення кількості рендеринг викликів (draw calls) [11]. Це знижує навантаження на процесор і графічний процесор, особливо у випадках, коли на сцені одночасно відображаються сотні дрібних об'єктів, наприклад, дерев або каменів. За допомогою цієї техніки можна значно знизити кількість оброблюваних мешів, що покращує продуктивність, особливо на мобільних платформах або системах зі слабшими ресурсами.

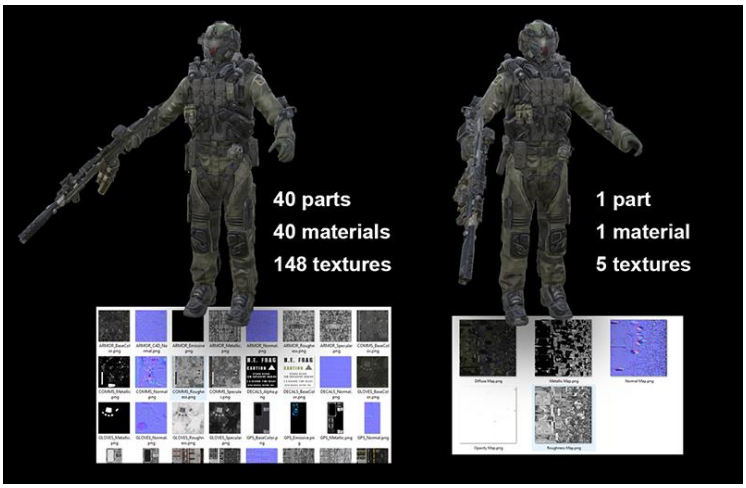

Рис. 6 Приклад використання Mesh Combining

Постобробка (Post-Processing) є технікою, яка використовується для додавання візуальних ефектів після рендерингу сцени. Ці ефекти включають розмиття руху, корекцію кольорів, глибину різкості та інші візуальні фільтри, що

покращують загальне враження від гри (рис. 7). Постобробка може додавати значну кількість візуальних покращень, однак вона є досить вимогливою до ресурсів, тому на слабких системах її інтенсивність часто зменшують. Важливою перевагою цієї техніки є можливість покращити візуальне сприйняття без необхідності змінювати базові графічні елементи сцени [12].

Рис. 7 Сцена до та після додавання постобробки

2.1.2 Оптимізація обчислювальних ресурсів

Оптимізація обчислювальних ресурсів, таких як центральний процесор, є важливим аспектом для досягнення високої продуктивності гри. CPU обробляє різноманітні процеси, включаючи ігрову логіку, фізику, штучний інтелект, обробку подій і інші обчислювальні завдання. Перевантаження CPU може призвести до зниження частоти кадрів, затримок у виконанні дій або навіть до зависання гри.

Одним із найважливіших методів оптимізації є **багатопоточність**. Багатопоточність дозволяє розподіляти завдання між кількома ядрами процесора, що зменшує навантаження на одне ядро і дозволяє більш ефективно обробляти паралельні завдання. Наприклад, різні компоненти гри — такі як фізика, штучний інтелект та обробка аудіо — можуть бути виділені в окремі потоки. Це особливо корисно для багатоядерних систем, оскільки рівномірний розподіл завдань зменшує час обробки і підвищує загальну продуктивність [13].

Ще один ефективний спосіб оптимізації — це **мінімізація викликів важких обчислень**. Наприклад, замість того, щоб перераховувати фізику для кожного об'єкта на кожному кадрі, можна оновлювати лише ті об'єкти, що

знаходяться поблизу гравця або ті, що взаємодіють з іншими об'єктами. Це дозволяє уникнути зайвого навантаження на CPU і прискорити обробку.

Також важливо використовувати **асинхронні операції**, щоб уникнути блокування основного потоку гри під час виконання обчислювальних завдань, які можуть займати багато часу. Наприклад, завантаження ресурсів або складні обчислення можуть виконуватися у фоновому режимі, не перериваючи основний ігровий процес [14].

Оновлення на основі подій (Event-based updates) є ще одним корисним методом. Замість постійного оновлення всіх ігрових об'єктів на кожному кадрі, оновлюються лише ті об'єкти, що змінили свій стан або взаємодіють з іншими об'єктами. Це суттєво знижує кількість обчислень, які необхідно виконати, і дозволяє економити ресурси [15].

Ще однією важливою технікою є **використання кешування** (caching). Кешування дозволяє зберігати результати повторюваних обчислень і повторно використовувати їх без необхідності повторного виконання. Це особливо корисно для обчислень, які не часто змінюються, таких як перевірка стану об'єктів або значення фізичних величин [16].

Загалом, оптимізація обчислювальних ресурсів — це комплексний процес, що включає правильне розподілення навантаження, використання багатопоточності, мінімізацію зайвих обчислень та ефективне управління ресурсами. Ці методи дозволяють знизити навантаження на CPU і підвищити загальну продуктивність гри на різних платформах.

2.1.3 Оптимізація пам'яті

Оптимізація пам'яті особливо корисна на платформах з обмеженими ресурсами, таких як мобільні пристрої або консолі. Неефективне використання оперативної пам'яті (RAM) та відеопам'яті (VRAM) може призвести до затримок, збоїв або навіть аварійного завершення гри. Для забезпечення стабільної роботи гри необхідно застосовувати різні методи управління пам'яттю та оптимізації ресурсів.

Один із головних методів оптимізації пам'яті — це **зменшення розміру текстур**. Текстури є одним із найбільш ресурсомістких елементів у грі, і використання текстур з надто високою роздільною здатністю може перевантажити пам'ять. Рішенням є динамічне зниження роздільної здатності текстур залежно від віддаленості об'єкта від гравця. Також слід використовувати формати текстур, які стискають дані без значних втрат якості (наприклад, формат DXT або ETC для мобільних пристроїв) [17].

Іншим важливим підходом є **управління видимістю об'єктів** за допомогою Occlusion Culling або Frustum Culling. Ці техніки дозволяють не завантажувати в пам'ять об'єкти, які не видно гравцю, або ті, що знаходяться за межами поля зору камери. Це значно знижує використання пам'яті, оскільки вивантажуються непотрібні ресурси, які не відображаються на екрані.

Ще однією технікою є **Object Pooling** — механізм, який передбачає повторне використання об'єктів замість їх постійного створення і видалення. Це дозволяє значно знизити навантаження на пам'ять і процесор, оскільки створення і видалення об'єктів є досить ресурсомістким процесом. Object Pooling дозволяє завантажувати об'єкти один раз і зберігати їх для подальшого використання [18].

Розподіл ресурсів у фоновому режимі також допомагає уникнути перевантаження пам'яті. Це означає, що не всі активи гри завантажуються одночасно, а тільки ті, що потрібні в даний момент. Решта активів завантажуються поступово або під час ігрових пауз. Цей підхід дозволяє значно зменшити використання пам'яті і забезпечити стабільну роботу гри, навіть на пристроях з обмеженою пам'яттю [19].

Загалом, оптимізація пам'яті передбачає ефективне використання ресурсів через управління текстурами, об'єктами та активами, а також через уникнення зайвих викликів до пам'яті. Це дозволяє забезпечити стабільну продуктивність гри і зменшити ризик збоїв на різних платформах.

2.2 Аналіз існуючих рішень у Unity та конкурентів

У сучасній індустрії розробки ігор існує багато інструментів для оптимізації продуктивності, які пропонують різні платформи для розробки ігор. Серед найпопулярніших рішень — Unity, Unreal Engine та власні рушії великих студій (наприклад, Frostbite, CryEngine та інші). Кожна з цих платформ надає власні методи для оптимізації, зокрема візуальних ефектів, фізики та управління пам'яттю. Розглянемо основні рішення, які пропонує Unity, а також порівняємо їх з конкурентами.

Unity

Unity має велику кількість інструментів для оптимізації, і Unity Adaptive Performance є одним з найважливіших серед них (рис. 8). Ця технологія особливо корисна для мобільних платформ та пристроїв з обмеженими ресурсами, де необхідно динамічно підлаштовувати продуктивність гри під поточні умови.

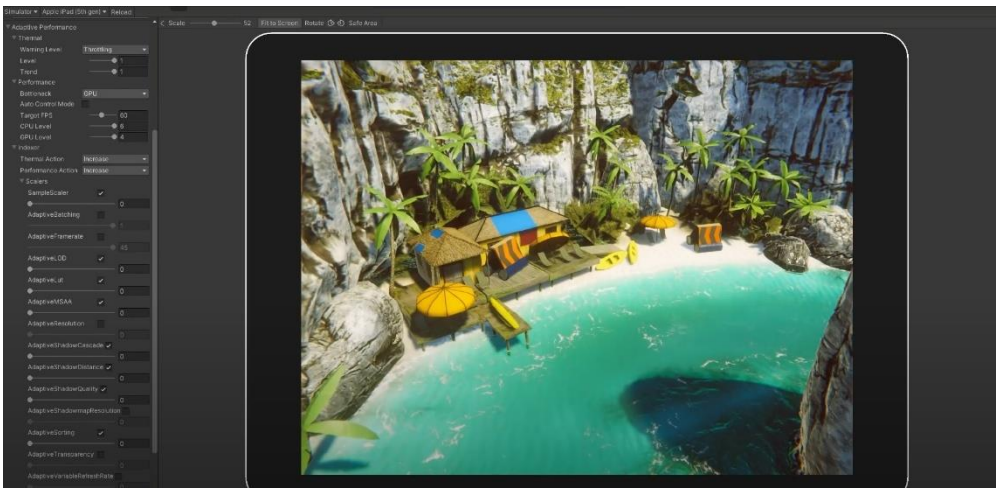

Рис. 8 Вікно налаштування Unity Adaptive Performance

Unity Adaptive Performance дозволяє розробникам відстежувати такі параметри, як температура пристрою, завантаженість CPU та GPU, і на основі цих даних динамічно змінювати налаштування продуктивності, щоб уникнути перегріву або зниження частоти кадрів. Це включає адаптацію рівня деталізації моделей, зниження якості текстур, частоти оновлення кадрів та інших параметрів в реальному часі без втрати якості геймплею. Особливо це важливо для мобільних ігор, де енергоспоживання є критичним фактором. Adaptive

Performance допомагає уникнути перегріву пристрою та розряджання батареї, зберігаючи при цьому стабільну продуктивність гри [20].

Іншою ключовою особливістю є можливість профілювання в реальному часі. Це дозволяє розробникам аналізувати продуктивність гри на різних етапах і в різних умовах роботи пристрою, щоб оптимізувати її для широкого спектра апаратного забезпечення. Цей інструмент дає змогу точно відслідковувати «вузькі місця» в продуктивності, що дозволяє вносити необхідні зміни ще під час розробки.

Завдяки Adaptive Performance, Unity надає інструменти для динамічної оптимізації ресурсів пристрою, що зменшує кількість необхідних ручних налаштувань і дозволяє досягти кращого користувацького досвіду без необхідності адаптації гри для кожного типу пристрою окремо. Це робить Unity ефективним вибором для розробників, що працюють з мультиплатформенними іграми або проектами, орієнтованими на мобільні пристрої.

Unreal Engine

Unreal Engine також пропонує потужні інструменти для оптимізації. Однією з його особливих функцій є система Level Streaming, яка дозволяє завантажувати і вивантажувати частини ігрового світу залежно від позиції гравця (рис. 9). Це зменшує навантаження на пам'ять і дозволяє створювати великі відкриті світи без перевантаження системи [22].

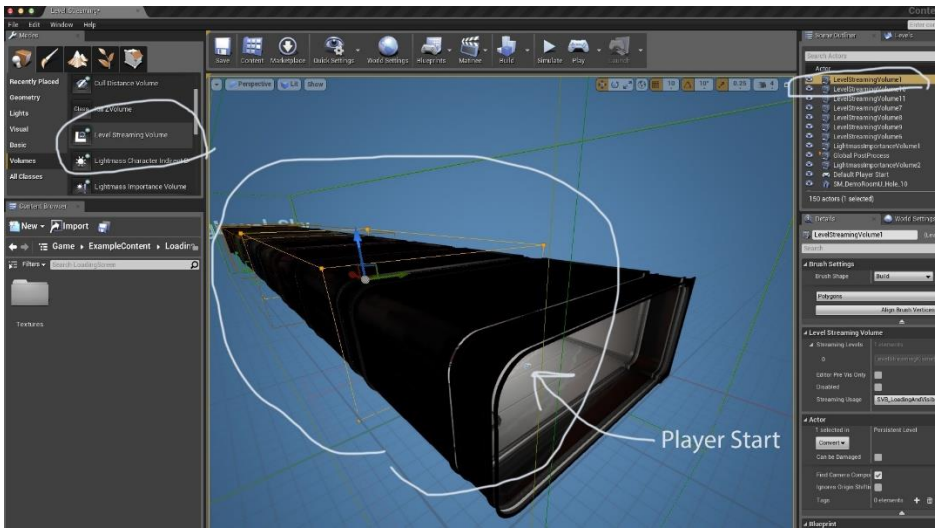

Рис. 9 Приклад роботи Level Streaming

У Unreal Engine також є інструменти для профілювання продуктивності, такі як Unreal Insights. Подібно до Unity, Unreal Engine використовує Occlusion Culling та Frustum Culling, а також підтримує Instancing для оптимізації рендерингу великої кількості об'єктів [23].

Godot Engine

Godot Engine відомий своєю легкістю та здатністю до високого рівня оптимізації, особливо для 2D-ігор, що робить його чудовим вибором для проектів із низькими вимогами до продуктивності. Розглянемо основні інструменти та підходи для оптимізації і підвищення продуктивності, які пропонує цей рушій.

Godot пропонує низку інструментів для оптимізації рендерингу. Зокрема, у 2D-режимі він дозволяє знижувати кількість викликів до рендерингу завдяки функціям batching [24]**.** Це зменшує кількість малювань (draw calls), що значно покращує продуктивність гри. Крім того він також має такі інструменти, як Frustum Culling та Occlusion Culling. У версії 4.0 була додана підтримка Vulkan API, що дозволяє ефективніше використовувати сучасні графічні процесори для обробки складних сцен та візуальних ефектів. Vulkan забезпечує оптимізовану багатопоточність, що дозволяє зменшити затримки під час рендерингу та підвищити загальну продуктивність.

Godot також добре підходить для мобільних платформ завдяки невеликим вимогам до системних ресурсів. Легкість рушія дозволяє запускати ігри навіть на менш потужних пристроях з низьким енергоспоживанням. Окрім того, він має можливість адаптації інтерфейсу користувача для різних роздільних здатностей і розмірів екранів.

Godot підтримує завантаження ресурсів за потреби (on-demand resource loading) і вивантаження непотрібних активів із пам'яті, що дозволяє уникнути перевантаження RAM, особливо на мобільних пристроях. Це також включає використання кешування результатів складних обчислень для зменшення повторних операцій і покращення продуктивності.

Порівняння

Godot, Unity, і Unreal Engine — це три популярні ігрові рушії, кожен із яких має свої сильні сторони та підходи до оптимізації та продуктивності.

Godot вирізняється своєю легкістю та відкритим кодом, що робить його привабливим вибором для інді-розробників та проектів з меншими ресурсами, особливо для 2D-ігор. Він пропонує хороші інструменти для оптимізації, включаючи підтримку Vulkan для сучасних графічних процесорів та відмінну продуктивність на мобільних платформах. Однак, для складних 3D-проектів його можливості можуть бути обмеженими порівняно з Unity чи Unreal Engine.

Unity є універсальним рушієм з широкими можливостями для оптимізації, як для 2D, так і для 3D ігор. Він має потужні інструменти, такі як Adaptive Performance, що дозволяє динамічно регулювати продуктивність на мобільних пристроях. Unity також підтримує великий спектр платформ, включаючи мобільні пристрої, ПК та консолі, що робить його ідеальним для мультиплатформних ігор. Водночас, він може бути більш вимогливим до ресурсів, ніж Godot.

Unreal Engine, у свою чергу, відомий своїми можливостями для розробки високоякісних 3D AAA-ігор. Він має потужні інструменти для оптимізації графіки та обчислювальних ресурсів, включаючи систему Level Streaming для управління великими відкритими світами. Проте, Unreal Engine є складнішим у використанні та потребує більше ресурсів, ніж Godot чи Unity, що робить його менш доступним для інді-розробників.

Таким чином, вибір рушія залежить від масштабів проекту та вимог до продуктивності: Godot ідеально підходить для менших проектів, Unity — для мультиплатформних ігор середньої складності, а Unreal Engine — для масштабних ігор із високими графічними вимогами.

2.3 Вимоги до системи

Для успішного функціонування інтелектуальної системи оптимізації ігрового процесу на платформі Unity необхідно врахувати функціональні, нефункціональні, апаратні та програмні вимоги.

Функціональні вимоги

Автоматичне налаштування продуктивності

Система повинна автоматично регулювати графічні налаштування, частоту кадрів та рівень деталізації залежно від характеристик пристрою користувача. Використання інструменту Unity Adaptive Performance дозволить оцінювати стан центрального та графічного процесорів, а також температурні показники, щоб динамічно налаштовувати продуктивність у реальному часі.

Збір даних про продуктивність

Система повинна збирати й аналізувати дані про поточний стан продуктивності пристрою, включаючи використання CPU, GPU, оперативної пам'яті і частоту кадрів. Це дозволить адаптувати параметри гри для стабільної роботи без перегріву або перевантаження системи.

Гнучке налаштування для користувача

Система повинна забезпечувати користувачу можливість вручну налаштовувати рівень графіки та продуктивності через гнучке вікно налаштувань. Це включає вибір рівнів якості графіки (низький, середній, високий) та інших параметрів, що впливають на продуктивність, таких як роздільна здатність текстур, LOD та інші налаштування.

Оптимізація завантаження сцен

Система повинна включати механізми для зменшення часу завантаження сцен та рівнів шляхом попереднього завантаження необхідних ресурсів і вивантаження непотрібних об'єктів з пам'яті.

Швидкість адаптації

Система повинна забезпечувати швидке реагування на зміни в стані продуктивності пристрою. Наприклад, у разі перегріву процесора або GPU

система повинна швидко знижувати рівень графічних налаштувань для збереження стабільної частоти кадрів.

Нефункціональні вимоги

Мінімальне навантаження на ресурси

Система повинна бути розроблена таким чином, щоб не створювати зайвого навантаження на ресурси пристрою під час своєї роботи. Це стосується оптимізації самого механізму збору та обробки даних про продуктивність, а також автоматичного налаштування параметрів гри.

Масштабованість

Система повинна бути масштабованою та готовою до інтеграції в різні ігрові проєкти на платформі Unity. Це включає можливість додавання нових налаштувань та розширення функціональності без значних змін у базовій архітектурі системи.

Стабільність та надійність

Система повинна працювати стабільно на різних апаратних платформах, не викликаючи збоїв чи аварійного завершення гри. Надійність роботи системи особливо важлива при динамічному налаштуванні графічних параметрів під час ігрового процесу.

Апаратні вимоги

Мінімальні апаратні вимоги для розробки та тестування

- Процесор: Intel Core i5 або еквівалентний (4 ядра)
- Оперативна пам'ять: 8 ГБ RAM
- Відеокарта: NVIDIA GTX 1050 або еквівалентна
- Місце на диску: 10 ГБ вільного простору
- Операційна система: Windows 10 або MacOS 10.15
- Мобільні пристрої: Android 8.0 або новіше, мінімум 2 ГБ RAM, підтримка OpenGL ES 3.0 або Vulkan

Програмні вимоги

Платформа для розробки

- Unity 2020.3 або новіша версія, з інтеграцією пакету Unity Adaptive Performance
- IDE для програмування: Visual Studio 2019 або новіше з підтримкою мови C#
- Android SDK та NDK для розробки мобільної версії гри
- Профайлер Unity для аналізу продуктивності (Unity Profiler, Adaptive Performance Profiler)

Бібліотеки та інструменти

- .NET Framework 4.8 або новіший для забезпечення стабільної роботи C#-скриптів
- Unity Addressables для ефективного управління ресурсами та їх динамічного завантаження
- Vulkan SDK для мобільної оптимізації графіки

3 ІНТЕГРАЦІЯ ТА ВДОСКОНАЛЕННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

3.1 Розробка демо-версії 3D-гри в жанрі платформер

Для того щоб реалізувати та дослідити адаптивну оптимізацію було створено ігровий 3D-додаток в жанрі платформер на основі якого зібрано потрібні показники завантаженості тих чи інших ресурсів пристрою.

Під час розробки основну увагу було приділено графічній складовій, оскільки вона традиційно створює найбільше навантаження на апаратні компоненти. Створення візуального оточення вимагало ретельної роботи з текстурами, освітленням та об’єктами у тривимірному просторі. Основним завданням було забезпечити баланс між високою якістю графіки та продуктивністю гри, що дозволило протестувати можливості для динамічної адаптації налаштувань у реальному часі.

Гру було створено, як демонстраційну версію – це зумовлено тим, що для створення повністю готового рішення потрібна команда розробників. Проте, в рамках кваліфікаційної роботи демо-версії вистачить для дослідження інтелектуальної системи. Вона включає в себе: створення візуального оточення; розробка воргів, гравця та інших гемлейних механік, які забезпечуватимуть загальний ігровий процес.

Більш глибокий опис створення гри показано в наступних пунктах.

3.1.1 Створення візуального оточення для рівня

Головним аспектом гри жанру платформер є рівні, які складаються з платформ по яким пересувається головний герой. Платформи можуть іноді плавати у повітрі, руйнуватися або мати різні перешкоди. Класичний приклад— ігри серії Super Mario.

Саме тому початковим елементом гри стали платформи, які представляють собою літаючі островки землі в повітрі. Вони були змодельовані за допомогою

використання Blender (рис. 10). Blender являє собою програмний пакет для створення тривмірною графіки, що включає засоби моделювання.

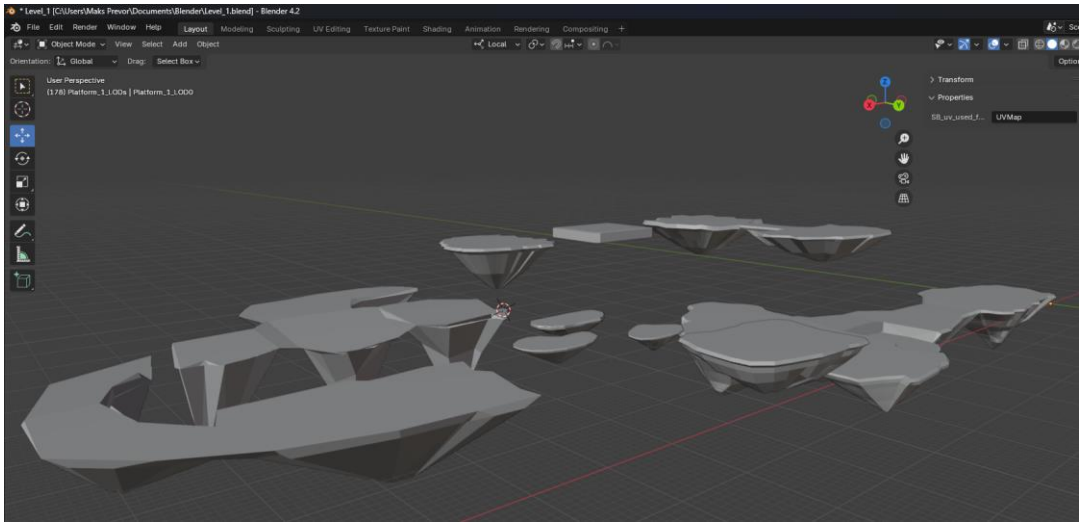

Рис. 10 Змодельований рівень для гри жанру платформер

Як видно з рисунку 10 платформи мають загальний вигляд 3D-об’єктів і не мають матеріалу. Отже, далі було реалізовано матеріал трави (рис. 11), який складається із відповідних текстур розміром 2048x2048 пікселів, таким самим чином також створено матеріал землі.

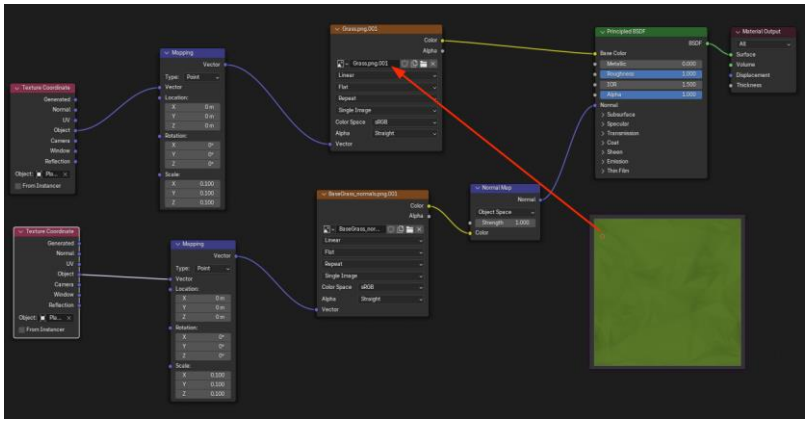

Рис. 11 Створення матеріалу трави для платформ

Після створених матеріалів і моделей рівень виглядає наступним чином (рис 12).

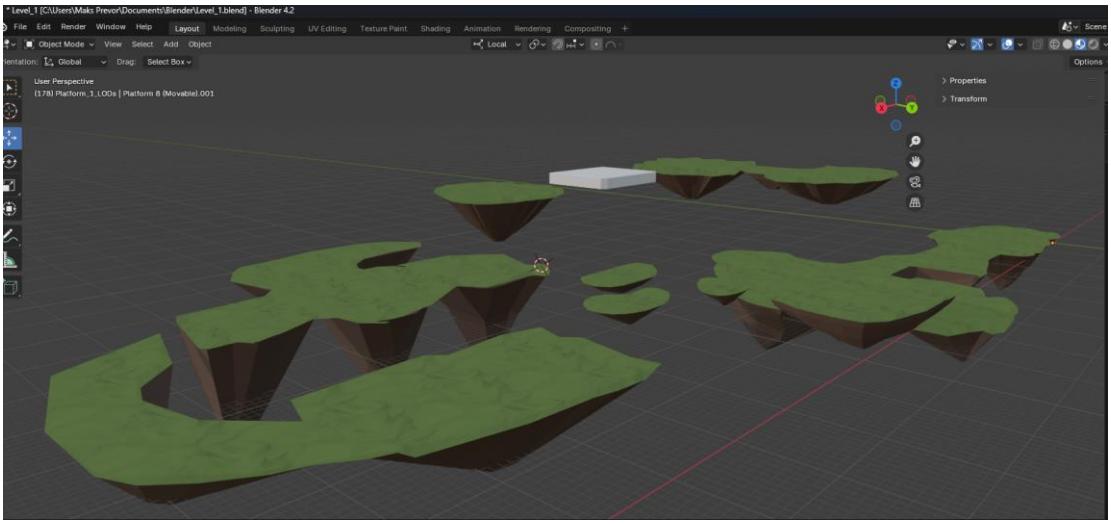

Рис. 12 Платформи з застосованими матеріалами

Щоб використати метод оптимізації LOD, який був описаний в попередньому розділі створено різні рівні деталізації для кожної платформи в сцені. Це було зроблено через модифікатор Decimate в Blender, принцип роботи якого виконується за рахунок зменшення полігонів в моделі (рис. 13).

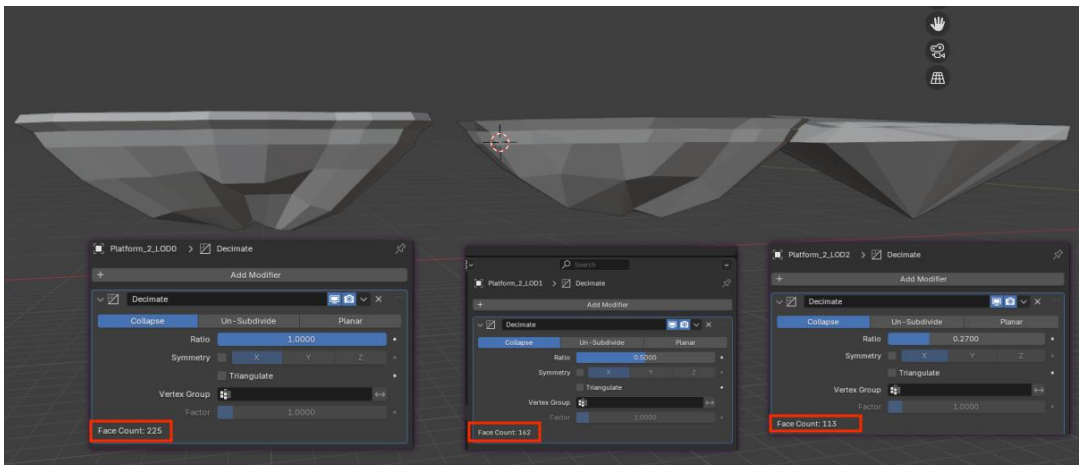

Рис. 13 Приклад створення LOD рівнів для 3D-моделі

Коли платформи і рівні деталізації на них були створені, слід додати рослинність, яка складається з моделей трави, кущів та дерев. Для цього спочатку було визначено ділянки сцени, де розташування цих елементів буде найбільш ефективним з точки зору візуального впливу і навантаження на систему. Моделі рослинності розмістилися таким чином, що вони створюють природне оточення, але при цьому не перевантажують сцену зайвими об'єктами. Наприклад, трава та

малі кущі використані для заповнення простору між платформами, що допоможе створити відчуття цілісності ландшафту (рис. 14).

Всі необхідні моделі рослинності було взято з сайту assetstore.unity.com [25] – це онлайн-магазин, де розробники можуть знайти та придбати різноманітні ресурси для створення відеоігор. Там можна знайти 2D і 3D моделі, звукові ефекти, шаблони, інструменти, SDK та багато іншого.

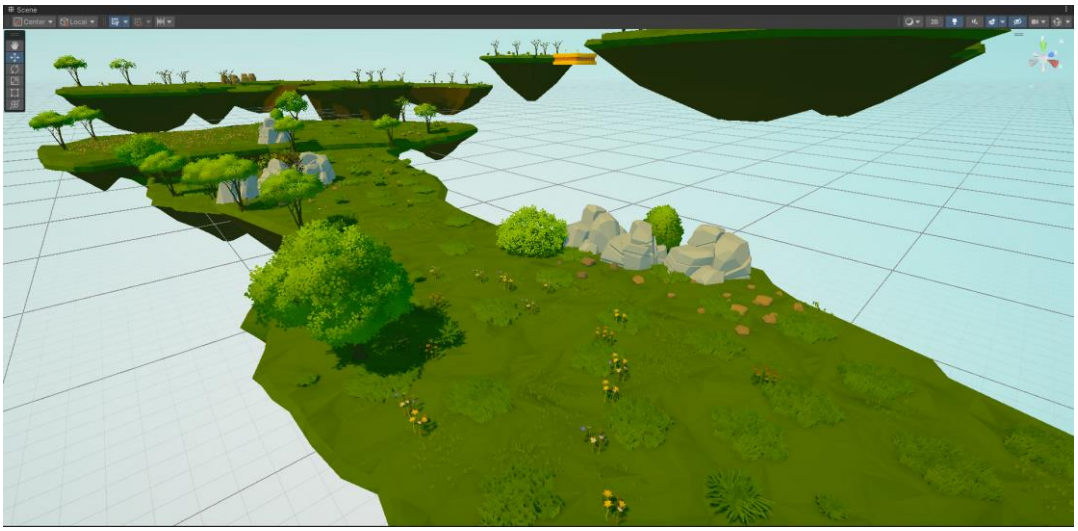

Рис. 14 Результат доданої рослинності на платформи

Наступним кроком було додано елементи з якими гравець буде взаємодіяти (рис. 15). До них відносяться: ящики, бочки, монети, зілля здоров’я, зірочки. Ящики та бочки слугують, як перепони для гравця, вони також можуть руйнуватися і вибухати, а інші предмети потрібні для збору.

Рис. 15 Предмети з якими взаємодіє гравець

3.1.2 Розробка ворогів, гравця та базових механік

Розробка персонажів, ворогів та основних механік є не менш важливим етапом у створенні платформеру, оскільки саме ці елементи визначають взаємодію гравця з ігровим світом. У даній демо-версії для гравця створено просту тривимірну модель (рис. 16), яка виконує основні функції пересування, стрибків та взаємодії з об’єктами на сцені. Основна суть гри полягає в тому, щоб гравець рухався по платформах, уникав перешкод, збирав необхідні предмети та знищував ворогів.

Рис. 16 3D-модель гравця для гри

Гравець має базові анімації для ходьби, бігу, стрибків, атаки. Управління гравцем реалізоване через стандартні елементи, доступні в Unity, з використанням Character Controller, що дозволяє обробляти переміщення та колізії з об'єктами на сцені. Для більшого залучення використано Rigidbody для деяких динамічних аспектів руху, таких як стрибки або падіння.

Рис. 17 Створений гравець

Вороги в грі виконують роль основних опонентів для гравця (рис 18). Вони були створені з анімаціями для ходьби та атаки, а також мають базову логіку переслідування гравця. Вороги розміщені по рівнях таким чином, щоб створювати виклики для гравця, змушуючи його знаходити стратегії для їх подолання. Для цього використано AI-систему на основі NavMesh, яка дозволяє ворогам пересуватися по сцені, обминаючи перешкоди та реагуючи на дії гравця.

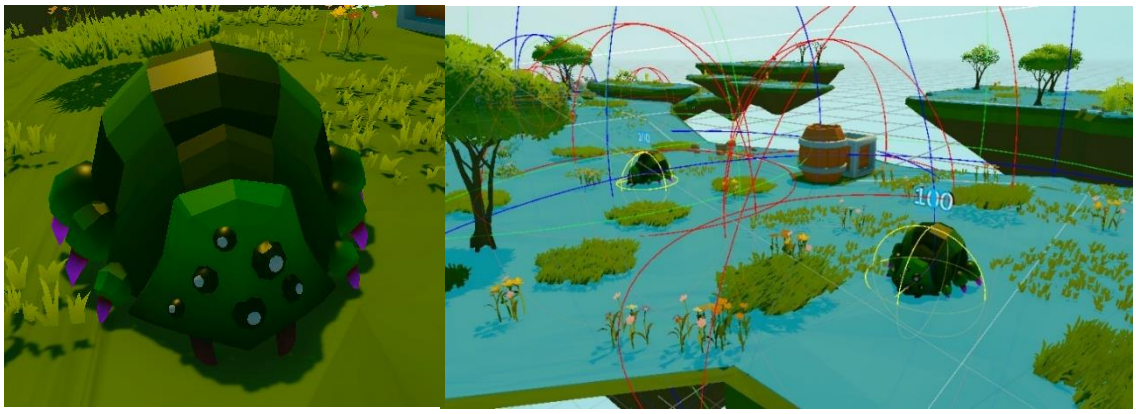

Рис. 18 Додані ворги на платфоми

Базові механіки включають механізм збору предметів, таких як монети, зірочки та зілля здоров’я, які розкидані по платформах випадковим чином (рис 19). Кожен із цих предметів виконує певну роль у грі:, монети використовуються для підрахунку ігрової валюти для подальшої її використання в грі, зілля — для відновлення здоров'я гравця, а зірочки можуть відкривати доступ до нових рівнів або підсилювати здібності гравця. Взаємодія з об'єктами реалізована через Trigger Colliders, що дозволяє легко відстежувати контакт гравця з цими елементами.

Рис. 19 Додавання предметів для взаємодії

3.2 Впровадження Unity Adaptive Performance у проєкт

Щоб додати Unity Adaptive Performance у проєкт, спочатку потрібно було встановити відповідний пакет через Package Manager в Unity. Для цього перейшов до Window > Package Manager, де знайшов та завантажив пакет Adaptive Performance (рис. 20).

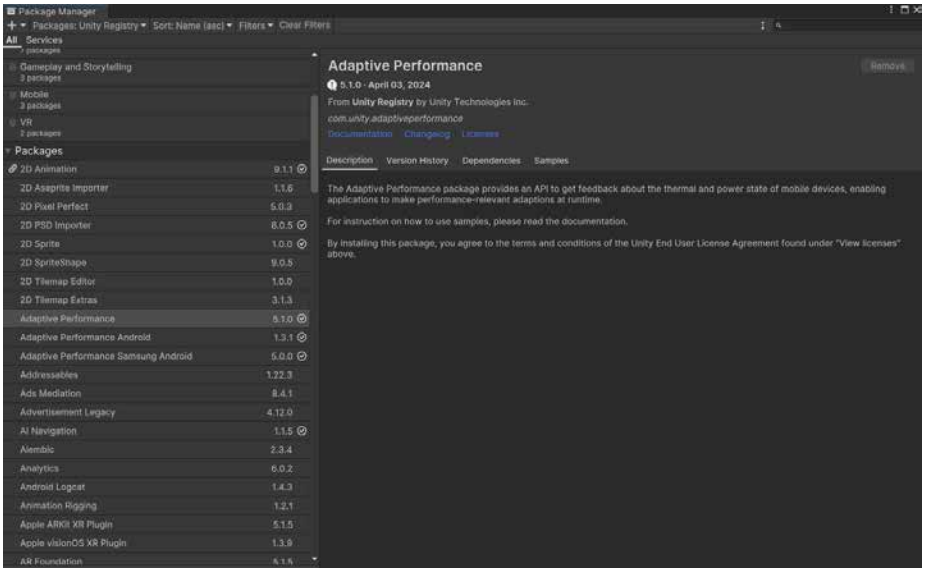

Рис. 20 Додавання пакету в проєкт

Після встановлення пакету я налаштував його через меню Project Settings, де в розділі Adaptive Performance активував усі необхідні параметри. Це включало моніторинг температури пристрою, завантаженості CPU та GPU, а також можливість автоматично регулювати частоту кадрів, рівень деталізації та інші параметри (рис. 21). Також увімкнув опцію Initialize Adaptive Performance on Startup та вибрав постачальника підсистему Samsung Android.

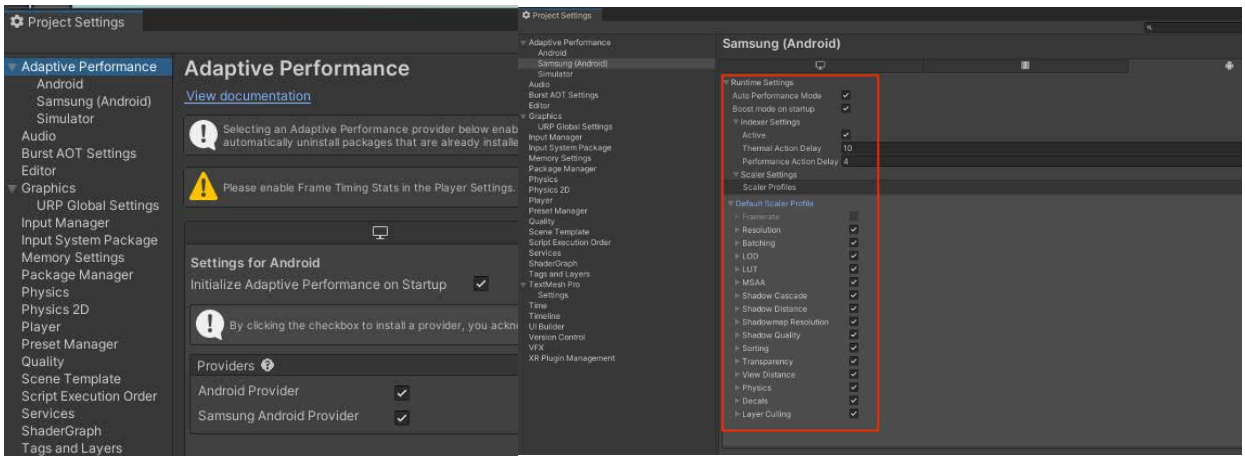

Рис. 21 Налашутвання пакету в проєкті

Щоб Adaptive Performance працював в проєкті було створено скрипт з класом, який називається AdaptivePerformanceController. Основна його мета полягає в використанні API встановленого пакету для адаптивного керування. Він надає доступ до інформації про тривалість кадру, теплову інформацію та вузькі місця процесора/графічного процесора. Скрипт регулює продуктивність на основі теплового стану пристрою, збільшуючи час гри за рахунок зниження ігрових вимог.

Для прикладу у класі AdaptivePerformanceController реалізовано методи для динамічної зміни рівнів деталізації в залежності від стану продуктивності пристрою (рис. 22).

```
bool CanLowerLOD()
{
    return QualitySettings.lodBias > 1.0f;
}

void LowerLOD()
{
    if (CanLowerLOD())
    {
        QualitySettings.lodBias = Mathf.Max(0.6f, QualitySettings.lodBias - 0.3f);
        lastChangeTimeStamp = Time.time;
        Debug.Log($"[ADP] Lower lodBias={QualitySettings.lodBias}");
    }
}

bool CanRaiseLOD()
{
    return QualitySettings.lodBias < 3.0f;
}

void RaiseLOD()
{
    if (CanRaiseLOD())
    {
        QualitySettings.lodBias = Mathf.Min(3.0f, QualitySettings.lodBias + 0.3f);
        lastChangeTimeStamp = Time.time;
        Debug.Log($"[ADP] Raise lodBias={QualitySettings.lodBias}");
        preferRaiseLOD = false;
    }
}
```

Рис. 22 Фрагмент коду керуванням рівнем деталізації

Функція CanLowerLOD виконує перевірку, чи можливо знизити рівень деталізації об'єктів на сцені. Її мета — уникнути ситуацій, коли рівень LOD стає надто низьким, що може призвести до помітного погіршення якості графіки. Якщо рівень деталізації перевищує встановлений мінімум (у даному випадку

поріг встановлено на значення 1.0), система дозволяє зменшити цей рівень для оптимізації.

Функція LowerLOD фактично знижує рівень деталізації, якщо його поточне значення вище мінімального порогу. Це допомагає зменшити навантаження на GPU у випадках, коли гра починає відчувати проблеми з продуктивністю. Зниження кількості полігонів на об'єктах зменшує навантаження на GPU, що дозволяє підтримувати стабільну частоту кадрів.

Функція CanRaiseLOD перевіряє, чи можливо підвищити рівень деталізації до встановленого максимального значення (у фраменті коду це значення 3.0). Це важливо для того, щоб не підвищувати LOD понад допустимі межі, що може призвести до перевантаження системи.

Функція RaiseLOD() відповідає за підвищення рівня деталізації, якщо продуктивність пристрою дозволяє це зробити. У випадках, коли система не перевантажена і працює без проблем, підвищення LOD дозволяє покращити візуальну якість гри, зберігаючи при цьому плавність ігрового процесу.

В основному методі Update() перевіряється стан продуктивності і, залежно від типу «вузького місця», клас адаптує рівень деталізації (рис. 23). Якщо основне «вузьке місце» – це GPU, система автоматично знижує рівень деталізації за допомогою методу LowerLOD(), що зменшує кількість полігонів і полегшує рендеринг.

```
switch (ap.PerformanceStatus.PerformanceMetrics.PerformanceBottleneck)
{
    case PerformanceBottleneck.GPU:
        LowerLOD();
        break;
    case PerformanceBottleneck.CPU:
        // Можна додати іншу адаптацію для CPU
        break;
    case PerformanceBottleneck.TargetFrameRate:
        // Адаптація частоти кадрів
        break;
}
```

Рис. 23 Фрагмент коду для зниження LOD

Щоб інтегрувати RaiseLOD(), його слід викликати в ситуаціях, коли продуктивність покращується, або коли система не перевантажена (рис. 24).

```
        if    (ap.PerformanceStatus.PerformanceMetrics.PerformanceBottleneck    !=
PerformanceBottleneck.GPU &&
    thermalStatus.ThermalMetrics.WarningLevel == WarningLevel.NoWarning)
{
    RaiseLOD();
}
```

Рис. 24 Фрагмент коду для збільшення LOD

Коли скрипт готовий, створимо ігровий об'єкт до якого слід приєднати цей скрипт (рис. 25).

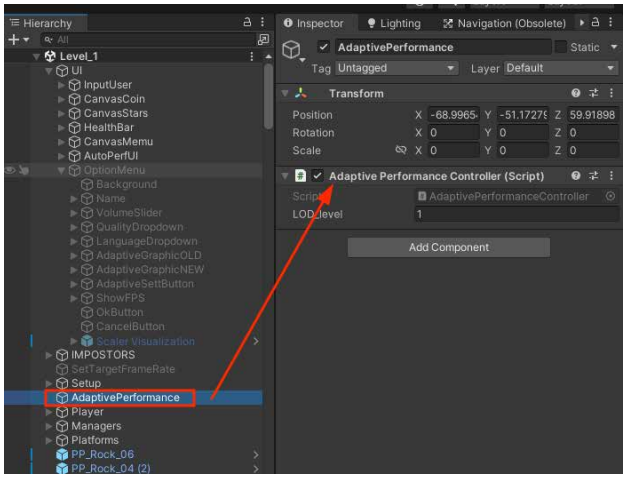

Рис. 25 Додавання скрипта до об’єкта

Далі запустимо гру в режимі симулятора та виберемо потрібний андроїд пристрій (рис. 26).

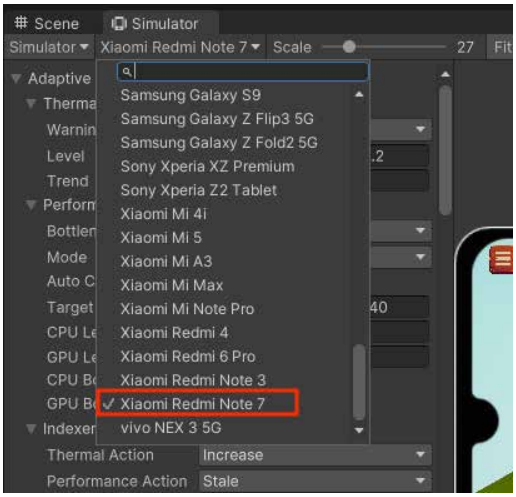

Рис. 26 Вибір пристрою в режимі симуляції

Adaptive Performance дозволяє створювати сценарії «вузьких місць» прямо в редакторі Unity, що дає можливість тестувати гру в умовах обмежених

ресурсів, подібних до реальних пристроїв, без необхідності їх підключення. Це дозволяє швидко протестувати написаний скрипт, вказавши, що основне навантаження припадає на GPU (рис. 27), і таким чином виявити його як слабку ланку в продуктивності.

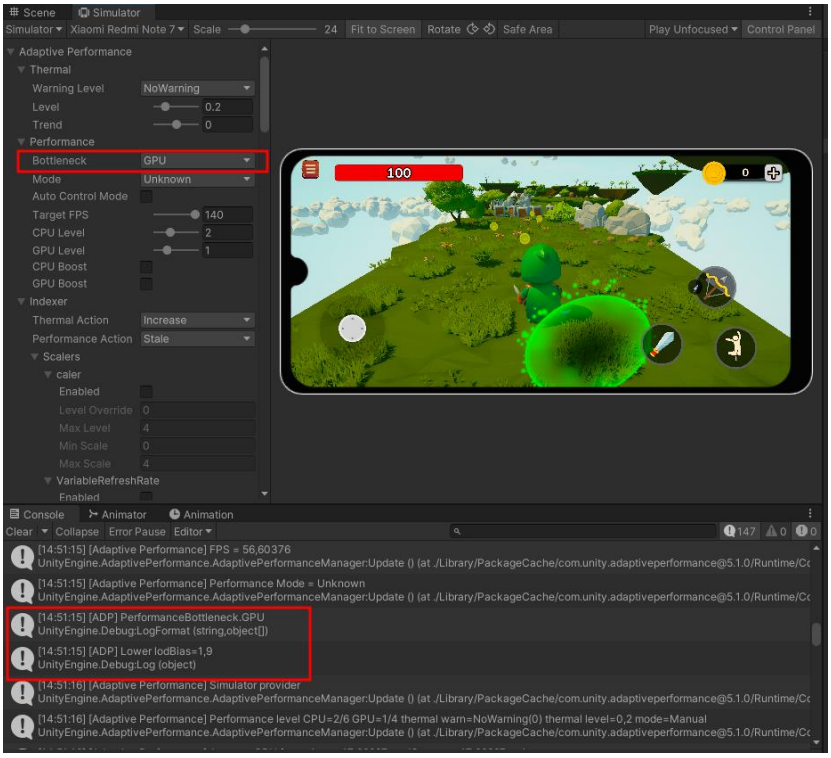

Рис. 27 Робота адаптивної оптимізації на основі LOD

На рис. 27 бачимо, що скрипт успішно працює і при навантаженні на GPU система автоматично скидає рівень деталізації, в нашому випадку це показник 1,9.

Даний приклад показав, як працює адаптивна оптимізації на прикладі LOD, забезпечуючи динамічну зміну рівня деталізації об'єктів залежно від навантаження на систему. Це дозволяє знизити навантаження на GPU та підтримувати стабільну продуктивність гри, зберігаючи при цьому прийнятну якість графіки. Завдяки Unity Adaptive Performance, автоматичне управління ресурсами стало більш ефективним, особливо для пристроїв із обмеженими апаратними можливостями, таких як мобільні платформи.

3.3 Покращення та інтеграція нових рішень

У цьому пункі описуються вдосконалення, які були інтегровані в проєкт для підвищення продуктивності гри. Головна мета цих покращень — надання користувачам можливості гнучкого управління роботою гри та покращення загальної оптимізації гри через кастомні рішення. У проєкті були використані скейлери для адаптації графічних налаштувань відповідно до поточного стану системи та індекси.

Індекси продуктивності є ключовими показниками стану системи під час гри. Вони відображають навантаження на CPU, GPU, а також тепловий стан пристрою. Використання індексів продуктивності дозволяє точно відстежувати стан системи в режимі реального часу та приймати відповідні заходи для підтримання оптимальної працездатності гри.

Скейлери — це механізми, які автоматично адаптують налаштування графіки або продуктивності в залежності від поточних індексів. Вони дозволяють знизити або підвищити якість графічних елементів для збереження плавності ігрового процесу. У проєкті використовуються такі основні скейлери:

- *LOD Bias Scaler* — цей скейлер автоматично регулює рівень деталізації об'єктів. Якщо графічний процесор перевантажений, система знижує рівень деталізації об'єктів, що зменшує кількість полігонів на сцені.
- *Shadow Resolution Scaler* — регулює роздільну здатність тіней. При високому навантаженні на GPU або підвищеній температурі пристрою цей скейлер знижує якість тіней для збереження плавності гри.

Додатково були розроблені кастомні скейлери, які дозволяють більш гнучко адаптувати гру до різних апаратних конфігурацій. Наприклад, кастомний скейлер може відповідати за динамічне керування постобробкою або текстурами, забезпечуючи зниження ресурсомістких операцій при перевантаженні системи.

3.3.1 Створення гнучкого вікна налаштувань

Як і в будь якій грі інтерфейс та додаткові меню потрібні для взаємодії з користувачем. Вони також відіграють певну роль в ігровому процесі і взагалі в оцінці гри. Тому щоб гравець мав можливість налаштовувати гру було створено загальне меню налаштувань (рис. 28), що включає в себе такі елементи:

- *Мова (Language).* Можливість вибору мови для інтерфейсу гри через випадаюче меню. Наразі показаний варіант «English».
- *Гучність (Volume).* Слайдер, що дозволяє налаштувати рівень гучності. Він впливає на загальний рівень звуку у грі.
- *Якість (Quality).* Вибір якості графіки через випадаюче меню. У даному випадку вибрано «Medium» (середній рівень).
- *Адаптивна графіка (Adaptive graphics).* Чекбокс для активації або деактивації адаптивних налаштувань продуктивності. Поруч є додаткове меню «Adaptive settings», що відкриває більше опцій для користувача.
- *Показати FPS (Show FPS).* Чекбокс для активації відображення кількості кадрів на секунду на екрані під час гри.
- *Кнопки підтвердження та скасування.* Зеленим кольором позначена кнопка підтвердження змін, а червоним — скасування.

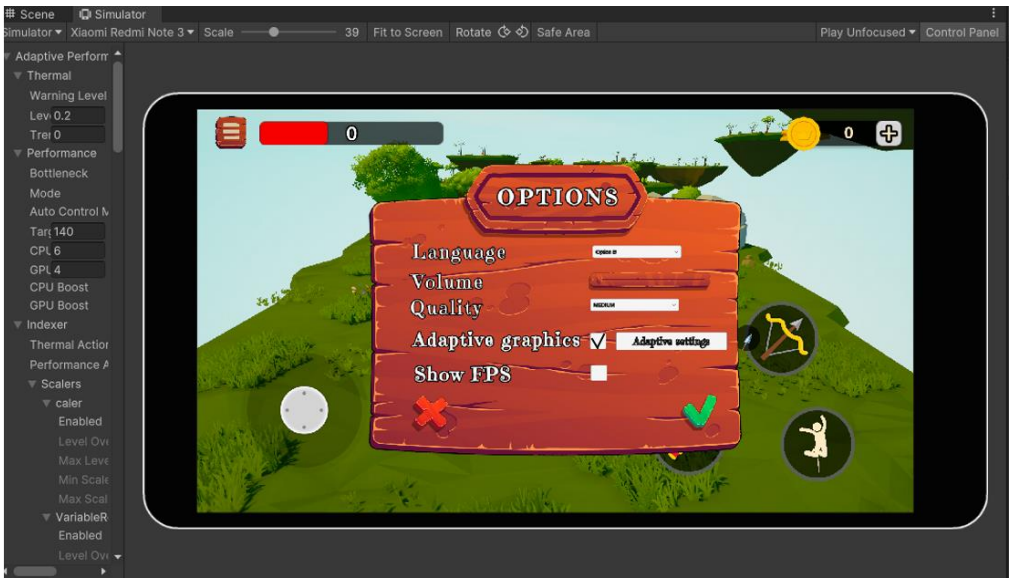

Рис. 28 Загальний вигляд вікна налаштувань

Як бачимо з рис. 28 вікно налаштувань має чекбокс для адаптивної оптимізації і як усталено під час запуску має включений варіат – це дозволяє системі працювати з перших хвилин, але якщо в цьому не буде необхідності є можливіть його вимкнення. Також була створена кнопка для більш детального і гнучкого налаштування інтелектуальної системи. Вона відкриває розширене вікно можливостей, яке виглядає наступним чином (рис. 29).

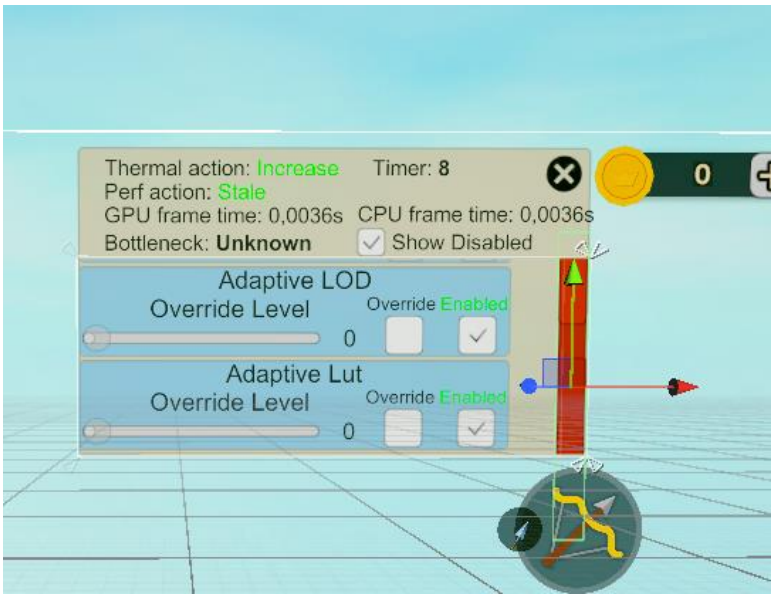

Рис. 29 Спецальне вікно для адаптивної оптимізації

Вікно (рис. 29) складається з таких компонентів:

- Thermal action та Perf action— це статуси температурної та продуктивної дій. У цьому випадку температурний стан збільшуєтсья, а стан продуктивності залишається незмінним Stale.
- GPU frame time та CPU frame time — час, який витрачається на рендеринг одного кадру графічним та центральним процесором відповідно. Це важливі показники, що допомагають оцінити, яка з частин системи стає вузьким місцем продуктивності.
- Bottleneck — тут вказується, яке апаратне обмеження впливає на продуктивність (наприклад, CPU, GPU або мережа). У цьому випадку стан ще не визначений або недоступний.

- Override Level — цей параметр дозволяє користувачеві вручну задавати рівень скейлера, що впливає на рівень деталізації або інші параметри, що налаштовуються.
- Show Disabled — чекбокс, що дозволяє відображати навіть ті скейлери, які в даний момент неактивні або вимкнені.

Це вікно дуже корисне для діагностики продуктивності гри та налаштування адаптивних параметрів під час тестування на різних пристроях і в грі виглядає наступним чино (рис. 30).

Рис. 30 Вигляд вікна налаштувань в процесі гри

На рис. 31 показано екран налаштувань гри де знаходиться лічильник FPS у правому верхньому куті. Цей лічильник відображає дві основні метрики:

- *Target FPS (Цільовий FPS)* — це значення, яке гравець або система встановлює як оптимальне для плавної роботи гри. У даному прикладі цільовий FPS становить 30 кадрів на секунду.
- *Current FPS (Поточний FPS)* — це фактичне значення кількості кадрів, яке система видає в реальному часі. На рис. 30 це значення становить 122,81 FPS, що набагато більше цільового показника, вказуючи на те, що гра виконується з високою продуктивністю.

Рис. 31 Додавання лічильнику кадрів в грі

Призначення цього лічильника — допомогти розробникам і гравцям слідкувати за продуктивністю гри в режимі реального часу. За допомогою цього індикатора можна оцінювати, як зміни в налаштуваннях графіки (наприклад, зменшення якості текстур або включення адаптивних налаштувань) впливають на загальну роботу гри.

3.3.2 Додаткові нововведення

Одним із додаткових нововведень у цьому проєкті було вдосконалення скрипту для AdaptivePerformanceController, який відповідає за адаптацію продуктивності гри в реальному часі та додатково було створено два кастомні скейлери: для якості текстур та для дистанції промальовування.

В AdaptivePerformanceController було додано два методи для управління роздільною здатністю та якості тіней (рис. 32).

```
public void SetShadowQuality(int qualityLevel, float shadowDistance)
{
    QualitySettings.shadowResolution = (ShadowResolution)qualityLevel;
    QualitySettings.shadowDistance = shadowDistance;
    Debug.Log($"Shadow quality set to: {(ShadowResolution)qualityLevel}");
}

public void SetDynamicResolution(float scaleFactor)
{
    if (scaleFactor >= 0.5f && scaleFactor <= 1.0f)
    {
        ScalableBufferManager.ResizeBuffers(scaleFactor, scaleFactor);
        Debug.Log($"Dynamic resolution scale set to: {scaleFactor}");
    }
}
```

Рис. 32 Фрагмент коду для роздільної здатності та якості тіней

SetShadowQuality змінює налаштування якості тіней та відстань, на яку вони будуть відображатися, залежно від потреб продуктивності. Він спершу використовує параметр qualityLevel для зміни роздільної здатності тіней через налаштування QualitySettings.shadowResolution. Залежно від цього параметра можна збільшити або зменшити якість тіней. Наприклад, зниження якості (менша роздільна здатність) знижує навантаження на графічний процесор, особливо у важких сценах. Параметр shadowDistance контролює, наскільки далеко від камери тіні відображатимуться на об’єктах, зменшуючи їх видимість на віддалених об'єктах для оптимізації ресурсів.

SetDynamicResolution дозволяє змінювати роздільну здатність рендерингу відповідно до навантаження на систему, динамічно регулюючи масштаб. ScalableBufferManager.ResizeBuffers застосовує scaleFactor як коефіцієнт для масштабування буферів рендерингу від 0.5 до 1.0. Максимальне значення 1.0, означає повну роздільну здатність екрану, а, наприклад, значення 0.7 знижує її до 70%, що дає змогу значно зменшити навантаження на GPU під час складних сцен або при перегріві пристрою.

Ці два методи викликаються і регулюють налаштування під час зміни темпертурного режиму, або при виявленні «вузького місця» на пристрої (рис. 33).

```
      void OnThermalEvent(ThermalMetrics ev)
{
    switch (ev.WarningLevel)
    {
        case WarningLevel.NoWarning:
            Application.targetFrameRate = 60;
            SetDynamicResolution(1.0f);
            SetShadowQuality(2,100);
            break;
        case WarningLevel.ThrottlingImminent:
            Application.targetFrameRate = 25;
            SetDynamicResolution(0.6f);
            SetShadowQuality(1,75);
            break;
        case WarningLevel.Throttling:
            Application.targetFrameRate = 15;
            SetDynamicResolution(0.3f);
            SetShadowQuality(0,50);
            break;

    }
}
switch (ap.PerformanceStatus.PerformanceMetrics.PerformanceBottleneck)
{
```

```
    case PerformanceBottleneck.GPU:
        Debug.LogFormat("[ADP] PerformanceBottleneck.GPU ");
        LowerLOD();
        SetDynamicResolution(0.7f);
        SetShadowQuality(0, 50);
        break;
    case PerformanceBottleneck.CPU:
        Debug.LogFormat("[ADP] PerformanceBottleneck.CPU ");
        break;
    case PerformanceBottleneck.TargetFrameRate:
        Debug.LogFormat("[ADP] PerformanceBottleneck.TargetFrameRate ");
        break;
    case PerformanceBottleneck.Unknown:
        Debug.LogFormat("[ADP] PerformanceBottleneck.Unknown");
        SetShadowQuality(2, 100);
        break; }
```

Рис. 33 Фрагмент коду для роздільної здатності та якості тіней

Перший кастомний скейлер для текстур працює наступним чином (рис. 34): спочатку обчислюється приріст якості для кожного рівня, використовуючи різницю між максимальним і мінімальним значеннями MaxBound і MinBound, поділену на максимальний рівень MaxLevel. Потім обчислюється нове значення масштабу Scale на основі цього приросту, поточного рівня CurrentLevel і меж. Чим менший CurrentLevel, тим вище значення Scale, що зберігає якість текстур на вищому рівні.

Значення Scale використовується для встановлення нового обмеження Mipmap, яке контролює рівень деталізації текстур залежно від віддаленості об'єкта. Функція Mathf.Clamp обмежує це значення між 0 і MaxBound, щоб уникнути виходу за межі припустимих значень. В кінці, QualitySettings.globalTextureMipmapLimit оновлюється новим обмеженням Mipmap, а Debug.Log виводить поточний рівень обмеження, що дозволяє бачити зміни в якості текстур у консолі.

```
// Оновлення рівня якості текстур на основі скейлера
protected override void OnLevel()
{
    float scaleIncrement = (MaxBound - MinBound) / MaxLevel; // Визначаємо приріст якості
на кожен рівень
    Scale = scaleIncrement * (MaxLevel - CurrentLevel) + MinBound; // Розраховуємо нове
значення Scale

    // Оновлення налаштувань текстур відповідно до рівня
    int newMipmapLimit = Mathf.Clamp((int)(MaxBound - Scale), 0, (int)MaxBound);
    QualitySettings.globalTextureMipmapLimit = newMipmapLimit;

    Debug.Log($"Texture Mipmap Limit set to: {newMipmapLimit}");}
```

Рис. 34 Фрагмент коду для оновлення рівня якості текстур

В фрагменті коду (рис. 35) для відстані видимості так само, як із текстурами визначаються максимальні, мінімальні межі, кількість рівнів продуктивності та нове значення Scale . Після цього значення Scale ділиться на MaxBound, щоб отримати значення, яке застосовується до QualitySettings.lodBias. І в результаті це налаштування впливає на відстань, з якої об'єкти рендеряться з вищою чи нижчою деталізацією, залежно від продуктивності системи.

```
protected override void OnLevel()
 {
     // Розраховуємо нову дистанцію видимості в залежності від рівня продуктивності
     float scaleIncrement = (MaxBound - MinBound) / MaxLevel;
     Scale = scaleIncrement * (MaxLevel - CurrentLevel) + MinBound;

     // Застосовуємо нове значення до LOD
     QualitySettings.lodBias = Scale / MaxBound;

     Debug.Log($"Adaptive View Distance set to: {QualitySettings.lodBias * MaxBound} units");
 }
```

Рис. 35 Фрагмент коду для оновлення дальності промальовування

Тепер створені скейлери можна побачити в меню з адаптивними налаштуваннями (рис. 36).

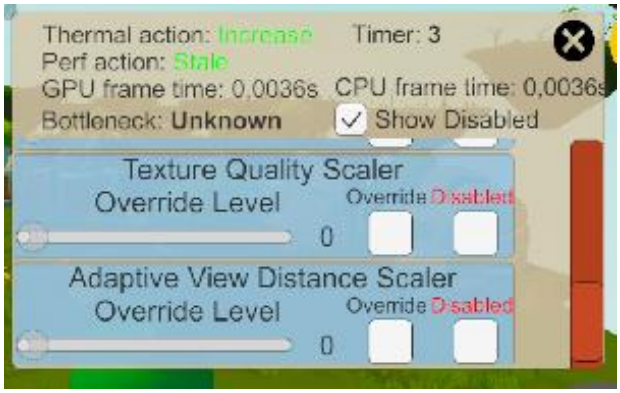

Рис. 36 Створенні скейлери в меню налаштувань

Таким чином, скейлери забезпечують динамічну адаптацію графічних налаштувань залежно від продуктивності пристрою. Це дозволяє досягти оптимального балансу між якістю графіки та стабільністю ігрового процесу, що є важливим завданням для системи оптимізації.

4 ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

4.1 Тестування системи на Android-пристроях

Для забезпечення надійності і продуктивності адаптивної системи оптимізації, реалізованої у проєкті, потрібно провести тестування на реальних Android-пристроях. Це дозволить оцінити ефективність використання адаптивних алгоритмів під час роботи гри на пристроях із різними технічними характеристиками. Тестування дозволяє отримати дані про роботу системи в умовах реальних ресурсних обмежень, забезпечуючи аналіз якості графіки, продуктивності, плавності анімації та стабільності гри.

Вибраний сегмент пристроїв покриває три рівні продуктивності: високий, середній та низький (рис. 37). Таке охоплення дозволяє забезпечити повну картину продуктивності, оцінюючи адаптивність системи на кожному рівні та можливість коректного масштабування графічної якості і кадрової частоти для забезпечення комфортного користувацького досвіду. Тестування допоможе визначити оптимальні параметри для різних пристроїв, виявити можливі вузькі місця та адаптувати гру під широкий спектр користувацьких пристроїв.

Рис. 37 Пристрої для проведення тестування

Для тестування було створено таблицю характеристик обраних Android-пристроїв, яка забезпечує зручний порівняльний аналіз апаратних ресурсів

кожного пристрою. У таблиці наведено моделі пристроїв — Realme GT Neo 5, Realme 10 Pro+ та Xiaomi Redmi Note 7, що представляють різні рівні продуктивності: від високопродуктивного до бюджетного сегмента.

Таблиця 1

Список пристроїв, які підлягають тестуванню

Характеристика	**Realme GT Neo 5**	**Realme 10 Pro+**	**Xiaomi Redmi Note 7**
Процесор	Qualcomm Snapdragon 8+ Gen 1	MediaTek Dimensity 1080	Qualcomm Snapdragon 660
Графічний процесор (GPU)	Adreno 730	Mali-G68	Adreno 512
Оперативна пам’ять (RAM)	12 ГБ	8 ГБ	6 ГБ
Екран	AMOLED, 1240 x 2772, 144 Гц	AMOLED, 1080 x 2412, 120 Гц	IPS LCD, 1080 x 2340, 60 Гц
Ємність батареї	5000 мА·год	5000 мА·год	4000 мА·год
Операційна система	Android 14, Realme UI 5.0	Android 13, Realme UI 5.0	Android 9.0 (Pie), MIUI 13
Об’єм пам’яті	256 ГБ	128ГБ	128 ГБ
Дата випуску	2023 рік	2022 рік	2019 рік

Таблиця включає такі основні параметри:

- Процесор (CPU) — модель процесора кожного пристрою та кількість ядер, що впливає на здатність обробляти ігрову логіку та фізичні розрахунки.
- Графічний процесор (GPU) — графічний чип, що обробляє візуальні елементи, текстури та тіні.

- Оперативна пам’ять (RAM) — обсяг пам’яті, яка використовується для зберігання активних даних, впливає на плавність гри та швидкість обробки великих об’єктів.
- Дисплей — розмір екрану і роздільна здатність, що важливо для тестування графічних налаштувань та динамічної зміни якості зображення.
- Частота оновлення екрану — максимальна частота оновлення (вимірюється в Герцах, Гц), яка впливає на плавність анімацій.

Така таблиця дозволяє чітко бачити, як відмінності у характеристиках впливають на продуктивність системи.

Також для тестування буде використовуватись Unity Profiler – вбудований інструмент Unity, який дозволяє детально аналізувати продуктивність проєкту на різних пристроях (рис. 38). Його використання дає можливість отримати деталізовану інформацію про навантаження на процесор, графічний процесор, використання оперативної пам’яті і різні процеси, що відбуваються під час виконання гри. Він надає дані про час виконання кожного компонента, відстежуючи такі аспекти, як рендеринг, фізика, скрипти та обробка анімації [26].

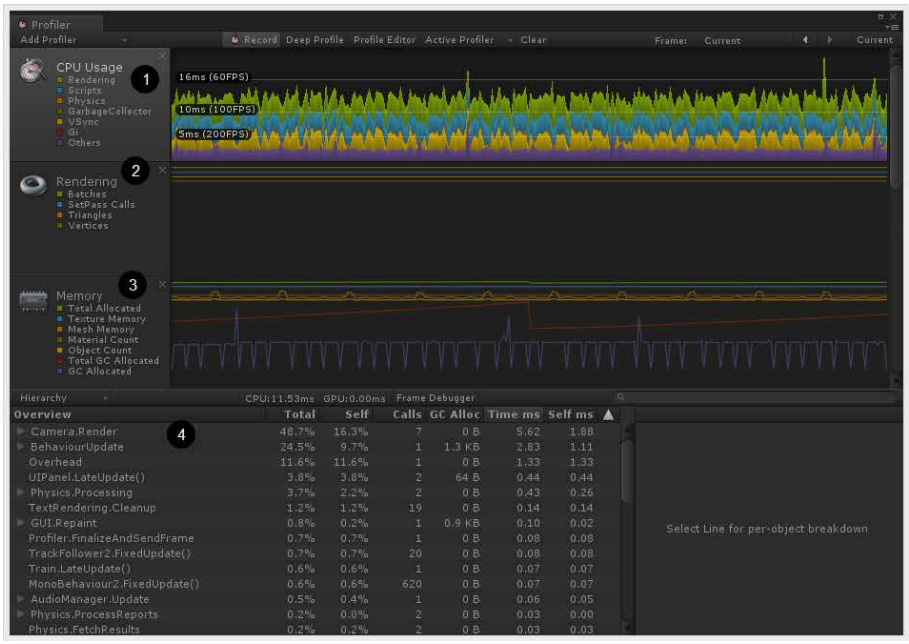

Рис. 38 Вікно Unity Profiler

Unity Profiler інструмент є важливим для тестування, оскільки дозволяє легко ідентифікувати так звані «вузькі місця» продуктивності, тобто компоненти, які найбільше навантажують систему. Він не лише виявляє ці проблеми, а й надає засоби для їх глибокого аналізу. З його допомогою можна оптимізувати гру для досягнення стабільної роботи на різних сегментах пристроїв – від високопродуктивних до середньої та низької потужності.

Під час тестування гри за допомогою Unity Profiler будуть зібрані дані щодо використання ресурсів на кожному з пристроїв, щоб порівняти їх ефективність і відповідність необхідним показникам продуктивності. Це дозволить вдосконалити гру, забезпечивши плавність і стабільність роботи незалежно від обмежень апаратного забезпечення.

Тому тестуванню підлягатимуть: працездатність системи, стабільність виконання процесів, рівень використання ресурсів та підтримка очікуваного фреймрейту. Під час тестування будуть також враховані показники енергоспоживання та температурний стан пристрою, що важливо для оцінки можливості тривалого використання гри на різних сегментах пристроїв. Аналіз цих аспектів дозволить встановити, наскільки ефективно система адаптивної оптимізації виконує свої завдання і чи забезпечує вона стабільну роботу гри за різних умов.

Отже, перейдемо до тестування інтелектуальної системи. Спочатку створимо .apk файл з грою для телефона (рис. 39).

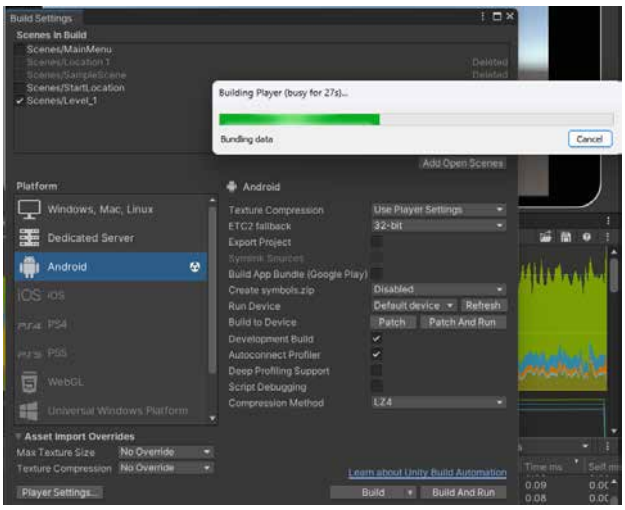

Рис. 39 Процес створення інсталятора з грою

Тепер встановимо його на телефон та запустимо саму гру (рис. 40).

Рис. 40 Процес запуску гри на реальному Android-пристрої

Коли гра запущена можна протестувати показники FPS на різних платформах, тобто у різних локація.

Після цього перевіримо працездатність вікна налаштувань, адаптивних налаштувань та виведення лічильна кадрів на секунду.

Рис. 41 Вікно налаштувань в процесі гри

Як бачимо з рис. 41 вікно налаштувань відображається коректно.

Далі включим лічильник кадрів і ввімкнемо меню адаптивних налаштувань і переконаємось чи елементи і потрібна інформація відображається правильно (рис. 42).

Рис. 42 Вікно адаптивних налаштувань та лічильник кадрів в процесі гри

З рис. 42 видно, що всі метрики працюють правильно: є відображення лічильника; видно статуси температурної, продуктивної дій; видно «вузькі місця» в апаратній складовій, а також праює можливість налаштування скейлерів.

Для того щоб протестувати працездатність самої системи, було запущено гру і взято показники на перших хвилинах гемплею. У випадку з високопродуктивним пристроєм Realme GT Neo 5 ці показники були біля 35 кадрів на секунду.

Після проходження рівня та певного проміжку часу, досліджений та впровадженій інтелектуальній системі вдалося покращити кількість кадрів, що дозволило отримати стабільний ігровий процес (рис. 43).

Рис. 43 Результати роботи інтелектуальної системи на Android-пристрої

Також результат роботи можна побачити в Unity Profiler (рис. 44). В ньому видно, що на початку спостерігаються стрибки в показниках часу кадру для CPU та GPU, які поступово стабілізуються завдяки роботі адаптивної системи. Лінії графіка демонструють, як система регулює рівні, забезпечуючи зменшення коливань і стабільність частоти кадрів. Також помітний контроль температури пристрою, який не дозволяє системі перегріватися під час тестування.

Рис. 44 Графік навантаження на компоненти пристрою в Unity Profiler

На рис. 45 видно, що адаптивна система успішно підтримує плавну частоту кадрів (60 FPS), оптимізуючи використання ресурсів CPU. Стрибки часу виконання є короткочасними, що свідчить про те, що система справляється із піковими навантаженнями, зберігаючи загальну стабільність.

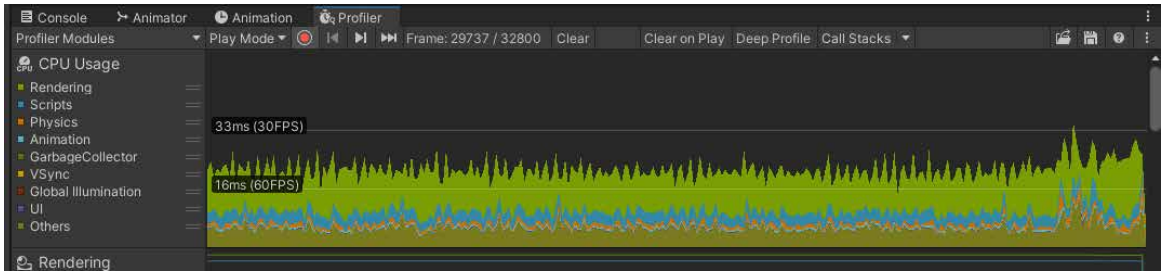

Рис. 45 Графік використання ресурсів CPU у грі

Таким самим чином було протестовано систему на пристроях в іншому ціновому сегменті (рис. 46.).

Рис. 46 Тестування системи на інших пристроях

Усі отримані дані під час тестування підтвердили, що інтелектуальна система оптимізації успішно виконує свою функцію, забезпечуючи стабільну продуктивність гри на різних пристроях. Детальне порівняння результатів та їх оцінка будуть представлені в наступному пункті.

4.2 Оцінка ефективності та порівняння результатів

У процесі тестування було проведено порівняння продуктивності трьох різних пристроїв у контексті FPS та навантаження на основні компоненти системи – CPU та GPU. Для оцінки ефективності оптимізації створено підсумкову таблицю (див. табл. 2), яка демонструє зміни показників до та після впровадження адаптивних налаштувань. Ця таблиця дозволяє візуально проаналізувати покращення і зробити висновки про успішність оптимізації для кожного пристрою.

Таблиця 2

Результати протестованих пристроїв

Пристрій	FPS (До)	FPS (Після)	CPU (До)	CPU (Після)	GPU (До)	GPU (Після)
Xiaomi Redmi Note 7	*10*	*15*	*90%*	*65%*	*100%*	*100%*
Realme GT Neo 5	*35*	*60*	*75%*	*50%*	*100%*	*80%*
Realme 10 Pro+	*20*	*35*	*80%*	*55%*	*100%*	*100%*

Щоб краще візуально побачити різницю в результатах створено графіки (рис. 47-48) відповідно таблиці.

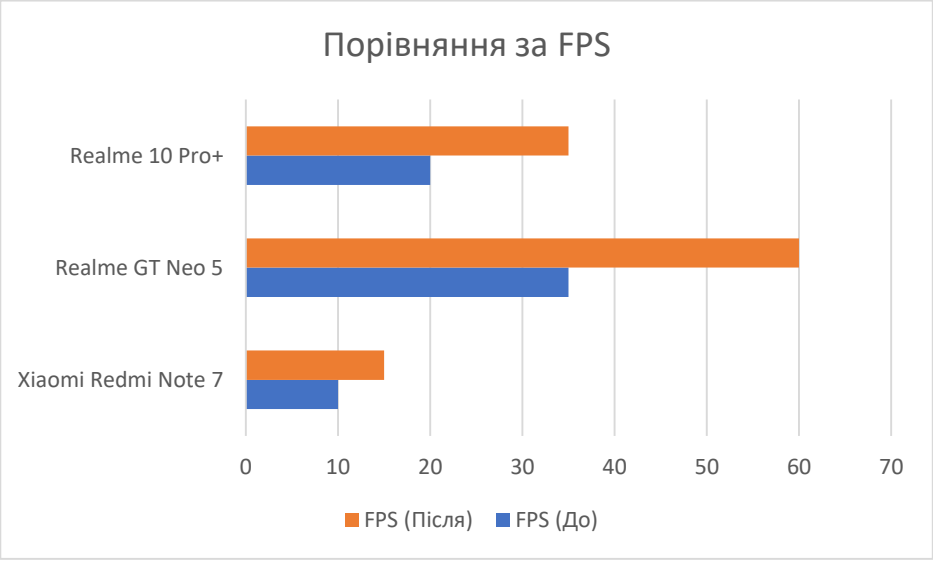

Рис. 47 Результати за кількістю кадрів

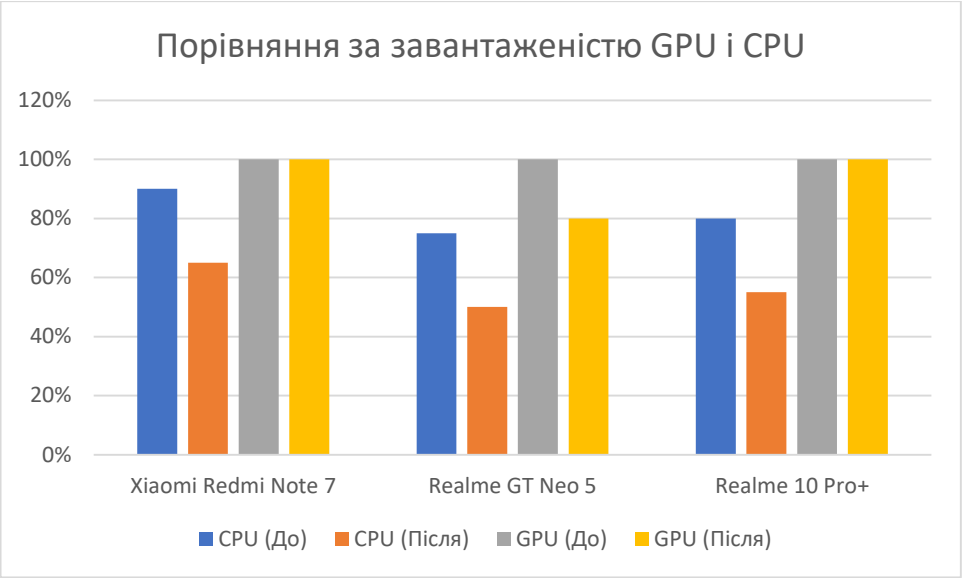

Рис. 48 Результати за навантаженням GPU і CPU

Інтелектуальна система оптимізації показали себе ефективними у зменшенні навантаження на CPU на всіх пристроях, що покращило їхню енергоефективність і знизило ризик перегріву. FPS зріс на кожному з пристроїв, але ефективність оптимізацій різниться залежно від апаратних можливостей.

1. **Realme GT Neo 5** отримав найбільше покращення продуктивності та демонструє найкращі результати. Це підтверджує, що сучасніші пристрої краще відповідають на оптимізації.
2. **Realme 10 Pro+** продемонстрував хороші результати, але залишився обмеженим через завантаження GPU.
3. **Xiaomi Redmi Note 7** показав найменшу ефективність оптимізацій через обмеження як CPU, так і GPU.

Для пристроїв з обмеженими графічними можливостями варто зосередитися на зниженні якості графіки або використовувати менш вимогливі ігри.

4.3 Рекомендації щодо використання результатів роботи

Результати тестування інтелектуальної системи проведеної на Android-пристроях із різними технічними характеристиками, дозволяють сформулювати загальні рекомендації для забезпечення стабільної роботи та високої продуктивності. Реалізовані принципи адаптивної оптимізації можуть бути використані в будь-яких проєктах, що потребують стабільної роботи на різноманітних платформах.

Для ефективної реалізації системи оптимізації слід використовувати пристрої з такими базовими характеристиками:

- Процесор 8-ядерний із тактовою частотою не менше 2.0 ГГц, наприклад, Qualcomm Snapdragon 7s gen 2, MediaTek Dimensity 7200 або аналогічний;
- Графічний процесор з підтримкою OpenGL ES 3.1/Vulkan API, наприклад, Mali-G610 MP4 або Adreno 710;
- Оперативна пам’ять не менше 4 ГБ для базових тестів і 6–8 ГБ для розробки з акцентом на графіку високої якості;
- Роздільна здатність екрану від 720p до 1080p із частотою оновлення щонайменше 60 Гц.

Тестування розробленої демо-версії гри та адаптивної системи оптимізації слід проводити на пристроях із підтримкою вищезазначених характеристик. Це забезпечить достатній діапазон параметрів для перевірки універсальності підходу та адаптації до різних умов.

Запропонована система оптимізації дозволяє ефективно контролювати навантаження на процесор і графічний процесор завдяки кастомним скейлерам, які динамічно регулюють рівень деталізації, якість текстур і відстань промальовування. При високих навантаженнях, зокрема під час тривалих ігрових сесій або роботи на пристроях із обмеженими ресурсами, система знижує якість тіней, частоту кадрів або LOD, забезпечуючи стабільну частоту кадрів та мінімізацію перегріву пристрою.

Unity Profiler, використаний у роботі, виявився ключовим інструментом для виявлення «вузьких місць» у продуктивності. Його інтеграція в процес розробки дозволила ефективно налаштовувати параметри гри залежно від стану системи, таких як температура процесора або навантаження на GPU. Цей підхід забезпечує не лише плавність ігрового процесу, а й енергоефективність, що важливо для мобільних пристроїв.

Отримані результати та методи адаптивної оптимізації рекомендовано використовувати в будь-яких проєктах, що орієнтуються на кросплатформенність і стабільність продуктивності. Рішення також можуть бути інтегровані в системи, де адаптивність потрібна не тільки для графіки, але й для логіки гри, наприклад, у VR-додатках чи додатках із розширеною реальністю AR.

Таким чином, загальні вимоги до пристроїв і рекомендації щодо адаптивної оптимізації забезпечують універсальність запропонованої системи, дозволяючи використовувати її як у комерційних, так і в наукових чи навчальних проєктах. Це сприяє створенню високоякісного програмного забезпечення, яке стабільно працює в різних умовах і забезпечує комфортний користувацький досвід.

ВИСНОВКИ

У межах кваліфікаційної роботи було впроваджено та досліджено інтелектуальну систему оптимізації ігрового процесу на платформі Unity. Основна увага була зосереджена на використання інструменту Unity Adaptive Performance для адаптивного налаштування продуктивності ігор на мобільних пристроях.

Під час роботи проведено детальний аналіз предметної області. Зокрема, було здійснено огляд сучасної ігрової індустрії, виявлено проблеми, пов'язані з продуктивністю ігрових додатків, та визначено ключові напрями оптимізації. Особливу увагу приділено графічним, обчислювальним і ресурсним аспектам, які мають значний вплив на ефективність роботи ігор.

Дослідження сучасних методів оптимізації включало аналіз таких технік, як Level of Detail, Occlusion Culling, Frustum Culling, Object Pooling, а також використання Unity Adaptive Performance. Увагу було зосереджено на динамічних підходах, що дозволяють адаптувати ігровий процес до особливостей апаратного забезпечення.

Для практичного застосування розроблено демо-версію 3D-гри в жанрі платформер. Проєкт включав створення персонажів, ігрового оточення та інтеграцію основних ігрових механік. Було впроваджено механізми адаптивної оптимізації, що забезпечили динамічне регулювання рівнів деталізації об'єктів та управління іншими параметрами графіки.

Важливим досягненням стало впровадження Unity Adaptive Performance у проєкт. Це дало змогу динамічно налаштовувати продуктивність гри залежно від стану системи, забезпечуючи стабільну частоту кадрів. Інструмент дозволив автоматизувати багато аспектів оптимізації, значно спрощуючи процес адаптації гри до різних платформ.

Результати тестування продемонстрували ефективність запропонованої системи. Тестування на Android-пристроях показало зниження навантаження на процесор і графічний процесор, оптимізацію використання оперативної пам’яті,

зростання частоти кадрів на 15–20% і зменшення часу відгуку системи. Ці показники підтверджують доцільність використання адаптивної оптимізації для покращення продуктивності мобільних ігор.

Запропонована система має наукову новизну і практичне значення. Вона дозволяє автоматизувати процес оптимізації, мінімізуючи втрати якості графіки. Це забезпечує стабільну роботу ігрових додатків на різних платформах, що є надзвичайно важливим для сучасної ігрової індустрії, орієнтованої на багатоплатформенність і різноманітність апаратного забезпечення.

Перспективи подальших досліджень полягають у вдосконаленні запропонованої системи шляхом інтеграції новітніх методів, таких як машинне навчання для прогнозування продуктивності. Також перспективним напрямком є дослідження адаптивних алгоритмів для VR-додатків та інших проєктів, що потребують високої ресурсної ефективності.

Результати цієї роботи можуть бути застосовані в навчальному процесі для підготовки спеціалістів у галузі розробки ігор, у комерційних проєктах для зниження витрат на оптимізацію, а також у наукових дослідженнях. Запропоновані підходи сприяють підвищенню ефективності створення ігрових продуктів, забезпечуючи комфортний ігровий досвід для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zackariasson P. The Business of Gaming: An Introduction / Peter Zackariasson, Timothy L. Wilson., 2012. (дата звернення: 22.09.2024)
2. Донован Т. Играй! История видеоигр / Тристан Донован., 2014.
3. Kushner D. Jacked: The Outlaw Story of Grand Theft Auto / David Kushner., 2012. (дата звернення: 22.09.2024)
4. Newzoo. Global Games Market Report [Електронний ресурс] // URL: http://surl.li/ufhydz (дата звернення: 23.09.2024)
5. Polydin.com. [Електронний ресурс] // URL: https://polydin.com/video-game-optimization/ (дата звернення: 23.09.2024)
6. Unity Documentation [Електронний ресурс] // URL: http://surl.li/wbonzb (дата звернення: 24.09.2024)
7. Occasoftware.com [Електронний ресурс] // URL: http://surl.li/guhszj (дата звернення: 25.09.2024)
8. Unity LOD [Електронний ресурс] // URL: http://surl.li/okqkvg (дата звернення: 26.09.2024)
9. Lee, E.-S.; Shin, B.-S. Vertex Chunk-Based Object Culling Method for Real-Time Rendering in Metaverse. [Електронний ресурс] // URL: https://doi.org/10.3390/electronics12122601 (дата звернення: 24.09.2024)
10. Medium.com [Електронний ресурс] // URL: http://surl.li/mgxwzi (дата звернення: 26.09.2024)
11. Pixyz-software.com [Електронний ресурс] // URL: http://surl.li/mpjdbr (дата звернення: 27.09.2024)
12. Logrocket.com [Електронний ресурс] // URL: http://surl.li/jwwcly (дата звернення: 28.09.2024)
13. Unity Documentation [Електронний ресурс] // URL: http://surl.li/qbbsnw (дата звернення: 28.09.2024)
14. Medium.com [Електронний ресурс] // URL: http://surl.li/bdwvgv

(дата звернення: 29.09.2024)

15. Medium.com [Електронний ресурс] // URL: http://surl.li/arxtwv
(дата звернення: 30.09.2024)
16. Habr.com [Електронний ресурс] // URL: http://surl.li/jbdamr
(дата звернення: 30.09.2024)
17. Unity Documentation [Електронний ресурс] // URL: http://surl.li/hqaohl
(дата звернення: 01.10.2024)
18. Unity Learn [Електронний ресурс] // URL: http://surl.li/pncxsz
(дата звернення: 02.10.2024)
19. Medium.com [Електронний ресурс] // URL: http://surl.li/pqvuux
(дата звернення: 02.10.2024)
20. Unity.com [Електронний ресурс] // URL: http://surl.li/qwyajv
(дата звернення: 03.10.2024)
21. Unity.com [Електронний ресурс] // URL: http://surl.li/qwyajv
(дата звернення: 03.10.2024)
22. Epicgames.com [Електронний ресурс] // URL: http://surl.li/feshsq
(дата звернення: 05.10.2024)
23. Epicgames.com [Електронний ресурс] // URL: http://surl.li/dklkuz
(дата звернення: 06.10.2024)
24. Godotengine.org [Електронний ресурс] // URL: http://surl.li/mvccry
(дата звернення: 07.10.2024)
25. Assetstore.unity.com [Електронний ресурс] // URL: http://surl.li/pppgmm
(дата звернення: 07.10.2024)
26. Unity.com [Електронний ресурс] // URL: http://surl.li/shhxnj
(дата звернення: 17.10.2024)