Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Рівень вищої освіти другий (магістерський)

Метод оптимізації ігрового додатку на платформі Unity

(тема)

Виконав:

студент II курсу, групи СПм-22-1

Воробйов А.А.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Фесенко Т.Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Воробйову Антону Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Метод оптимізації ігрового додатку на платформі Unity

затверджена наказом по університету від " 06 " листопада 2023 р. № 1299 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2024р.

3. Вхідні дані до роботи 1) Платформа Unity; 2) Оптимізація коду; 3)Оптимізація ігрового додатку з допомогою розроблених методів

4. Перелік питань, що потрібно опрацювати у роботі
1)Огляд існуючих рішень (методів), для оптимізації ігрового додатку;
2)Вибів існуючих методів дослідження;
3)Реалізація та розвиток методів у програмному застосунку;
4)Проведення експериментальних досліджень;
5)Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 13 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд методів дослідження	07.11.23-13.11.23	
2	Вибір та обґрунтування методів дослідження	14.11.23-20.11.23	
3	Вибір програмного застосунку	21.11.23-23.11.23	
4	Розробка ігрового додатку	24.11.23-06.12.23	
5	Проведення експериментів	07.12.23-23.12.23	
6	Оформлення матеріалів кваліфікаційної роботи	26.12.23-02.01.24	
7	Подання кваліфікаційної роботи керівникові та	03.01.24-06.01.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	09.01.24-12.01.24	
	вання		

Дата видачі завдання 06 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Фесенко Т.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Звіт з кваліфікаційної роботи містить – 61 сторінок, 25 рисунка, 13 джерел посилання, 3 таблиці.

UNITY, .NET, C#, OOP, GAMEDEV, VISUAL STUDIO, OBJECT POOLING, DEPENDENCY INJECTION, ENTITY COMPONENT SYSTEM, OCCLUSION CULLING, DYNAMIC LOADING, DRAW CALL, PROFILLER.

Метою кваліфікаційної роботи є оптимізація розробленої гри в середовищі Unity з підвищенням продуктивності та ефективності, дослідити існуючі методи оптимізації та обрати найкращі які підходять до проекту та реалізувати їх.

Результатом роботи були обрані методи оптимізації, з допомогою яких був пришвидшений ігровий додаток.

ABSTRACT

The report on the qualification work contains 61 pages, 25 figures, 13 reference sources, 3 spreadsheet.

UNITY, .NET, C#, OOP, GAMEDEV, VISUAL STUDIO, OBJECT POOLING, DEPENDENCY INJECTION, ENTITY COMPONENT SYSTEM, OCCLUSION CULLING, DYNAMIC LOADING, DRAW CALL, PROFILLER.

In order to qualify the work to optimize the fragmented game in the middle of Unity with increased productivity and efficiency, follow the common methods of optimization and reverse how to approach the project and implement them.

The result of the work was a software application, a game that can be installed on any device from a computer to a phone, with an interesting and simple gameplay you can spend your free time enjoying.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ООП – об'єктно-орієнтоване програмування

SDK – комплект для розробки програмного забезпечення

VR – віртуальна реальність

LOD – рівень деталізації

API – інтерфейс програмування програми

GPU – графічний процесор

FPS – частота кадрів

SSAO – навколишнє затінення в екранному просторі

Mesh – набір вершин та багатокутників, що визначають форму тривимірного об'єкта

API – інтерфейс програмування додатків, програмний інтерфейс програми

ПК – персональний комп’ютер

ВСТУП

Початково комп'ютери використовувалися лише для наукових цілей. Однак із розвитком технологій вони стали доступнішими, і їх почали використовувати для вирішення більш широкого спектру завдань. Зрештою, ентузіасти почали використовувати комп'ютери для розваг.

Сучасна індустрія геймдеву пов'язана з постійними викликами у сфері оптимізації гри з метою забезпечення кращої продуктивності та найкращого геймплею. Однак, з урахуванням різноманітності платформ і обмежень обчислювальних ресурсів, виникає потреба у ефективної оптимізації гри для різних пристроїв та платформ.

Платформа Unity стала однією з найпопулярніших та потужних інструментів для розробки ігор, проте оптимізація ігор на цій платформі є складним завданням і вимагає глибокого розуміння принципів її функціонування.

Ігрові додатки є одними з найскладніших програмних продуктів. Вони часто вимагають значних ресурсів комп'ютера, таких як процесор, пам'ять та графічний процесор. Це може призвести до того, що ігрові додатки будуть працювати повільно, що може негативно позначитися на досвіді гравців.

Оптимізація ігрового додатку – це процес підвищення його продуктивності шляхом зменшення використання ресурсів комп'ютера. Оптимізація може бути проведена на різних рівнях, від коду до самого движка Unity.

Об’єктом дослідження – є процес оптимізації продуктивності ігрового додатку на платформі Unity.

Предмет дослідження – методи оптимізації ігрового додатку на платформі Unity, які включають:

-оптимізацію ресурсів, таких як текстури, моделі, скрипти та звуки;

-оптимізацію коду, включаючи використання ефективних алгоритмів, оптимізацію пам'яті та продуктивності;

-оптимізацію графіки, включаючи використання технологій, таких як теселяція, LOD і запікання світла.

Метою кваліфікаційної роботи – розробка алгоритмів оптимізації шляхом застосування методу оптимізації коду.

Завдання роботи – проаналізувати існуючі методи оптимізації ресурсів, графіки та коду, дослідити вплив оптимізації на продуктивність ігрового додатку, розробка методу до оптимізації ігрового додатку. Розглянути методи оптимізації ігрового додатку на платформі Unity. У дослідженні будуть розглянуті основні методи оптимізації, а також їх ефективність.

Методи дослідження. Метод Object Pooling (для оптимізувати продуктивність ігрового додатку на платформі Unity шляхом повторного використання вже створених об'єктів). Метод Dependency Injection (допомагає оптимізувати продуктивність ігрового додатку на платформі Unity шляхом зменшення кількості циклів залежності). Метод Occlusion Culling (дозволяє виключати з рендерингу об'єкти, які не видно з точки зору камери).

Результати кваліфікаційної роботи були апробовані на науковій конференції Modernization of science and its influence on global processes: collection of scientific papers «SCIENTIA» with Proceedings of the IV International Scientific and Theoretical Conference, Bern, Swiss Confederation: International Center of Scientific Research.[13].

Результатом кваліфікаційної роботи та ігровий додаток, проаналізовані метод оптимізації ігор у жанрі Roguelike на движку Unity.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблем і переваг розробки ігрового додатку на платформі Unity

Unity — це кросплатформний ігровий движок, розроблений компанією unity technologies, який в основному використовується для розробки відеоігор і симуляцій для комп'ютерів, консолей і мобільних пристроїв. Вперше було оголошено лише для OS X на всесвітній конференції розробників Apple у 2005 році, згодом його було розширено до 27 платформ.

Unity — це ігровий движок, який використовується для розробки 2D- та 3D-ігор для широкого спектру платформ, включаючи ПК, мобільні пристрої, консолі та віртуальну реальність. Він пропонує широкий спектр функцій і інструментів, які роблять його популярним вибором для розробників ігор усіх рівнів досвіду.

Unity особливо популярний для розробки мобільних ігор, так як вона зосереджена на мобільних платформах. Двовимірний конвеєр Unity3D є нещодавнім доповненням до механізму та менш зрілим, ніж 3D-конвеєр. Незважаючи на це, Unity є сучачною платформою для розробки 2D-ігор навіть у порівнянні з іншими 2D-движками.

Ця платформа є хорошим вибором для розробки віртуальної реальності, хоча на даний момент ринок віртуальної реальності є дуже малим. Ринки мобільних пристроїв і PSVR на сьогодні є найбільшими у VR, Unity має хороші можливості для портування ігор на багатьох платформах, таких як PS4 і ПК.

Механізм націлений на такі графічні API: Direct3D у Windows і Xbox One; OpenGL в Linux, macOS і Windows; OpenGL ES на Android та iOS; WebGL в Інтернеті; і власні API на ігрових консолях.

Крім того, він підтримує низькорівневі API Metal на iOS і macOS і Vulkan на Android, Linux і Windows, а також Direct3D 12 на Windows і Xbox One. У

2D-іграх Unity дозволяє імпортувати спрайти та розширений засоби візуалізації 2D-світу.

Для 3D-ігор Unity дозволяє специфікувати параметри стиснення текстури та роздільної здатності для кожної платформи, яку підтримує ігровий движок, а також забезпечує підтримку відображення рельєфу, паралакса, екранного простору ambient occlusion (SSAO), динамічних тіней за допомогою карт тіней, візуалізації до текстури та ефекти постобробки на весь екран.

Unity — це набір програмного забезпечення для розробки за замовчуванням (SDK) для платформи консолі відеоігор Wii U Nintendo, причому Nintendo надає безкоштовну копію до кожної ліцензії розробника Wii U. Unity Technologies називає це об’єднання стороннього SDK «першим у галузі».

Unity став одним із найпопулярніших інструментів для розробки відеоігор, і його застосування настільки розширилося, що ігри, створені на цьому движку, доступні на різних платформах – від мобільних пристроїв до особистих комп'ютерів. В цьому контексті оптимізація стає життєво важливою для розробників, які мають на меті забезпечити плавний геймплей та відмінну якість гри на різних пристроях.[13]

Головні переваги движка:

-підтримка багатьох платформ: легко переносити додаток або гру з Android на iOS або навпаки, випускати їх для ПК, консолей або будь-якої іншої платформи;

-велика бібліотека готових ресурсів і плагінів значно прискорює процес розробки в Unity і зменшує фінансові витрати. Заощаджуйте на створенні рівнів, моделей персонажів та навіть шаблонів поведінки штучного інтелекту завдяки безкоштовним шаблонам і заготовкам;

-реалістична фізика взаємодії твердих тіл, високорозвинений ragdoll, ефективні колізії об'єктів та інструменти для створення складних анімацій;

-для використання движка достатньо знань мови C#, що, в свою чергу, зменшує необхідність у висококваліфікованих програмістах і, відповідно,

економить кошти;

-модульна система компонентів движка, яка дозволяє конструювати ігрові об'єкти.

Оптимізація є процесом, спрямованим на максимізацію вигідних характеристик та співвідношень, а також на мінімізацію витрат. Завдання оптимізації визначається наступним чином, якщо надані:

-критерій оптимальності, який включає економічні та технологічні вимоги, такі як вихід продукту, вміст домішок та інші параметри;

-варіюючи параметри, такі як температура, тиск, обсяг вхідних потоків у процесах переробки сировини, що можуть впливати на ефективність процесу;

-математична модель процесу;

-обмеження, пов'язані з економічними та конструктивними умовами, можливостями апаратури, вимогами вибухо-небезпеки та інше.

Методи оптимізації застосовуються для пошуку та розрахунку оптимальних технологій, геометричних конструкцій, оптимальних часів для технологічних процесів та подібних задач.

Оптимізація гри в Unity є важливою складовою розробки ігрового проекту, особливо для забезпечення плавності гри та оптимальної продуктивності але ми можемо зустріти проблеми пов’язані з оптимізацію на движку. А саме:

-потужність обчислень: Графічні ефекти, штучний інтелект, фізика та інші обчислення можуть бути дуже витратними з точки зору ресурсів. Потрібно здійснити оптимізацію коду та використання ресурсів для зменшення навантаження на процесор і GPU.

-завелика кількість об'єктів: Велика кількість ігрових об'єктів на сцені може призвести до зниження продуктивності. Важливо використовувати рівні деталізації, об'єднувати об'єкти, видаляти непотрібні та використовувати різні техніки оптимізації рендерингу.

-неефективне використання пам'яті: Пам'ять може стати обмежуючим фактором для продуктивності. Важливо уникати витоків пам'яті,

використовувати текстури та ресурси оптимально та звільняти ресурси, які вже не потрібні;

-неналежний рендеринг: Рендеринг може бути дуже витратним. Важливо використовувати різні техніки оптимізації, такі як Level of Detail (LOD), маскування частин сцени та оптимізація матеріалів;

-неефективний код: Неоптимізований код може суттєво вплинути на продуктивність гри. Важливо оптимізувати код, уникати зайвих обчислень та використовувати належні структури даних;

-недооптимізовані асети: Великі текстури, моделі та інші асети можуть збільшити час завантаження гри та використовувати багато пам'яті. Важливо оптимізувати асети та використовувати формати, які підходять для конкретної гри;

-неефективне управління анімаціями: Анімації можуть бути дуже витратними. Важливо оптимізувати управління анімаціями та використовувати механізми, які зменшують навантаження;

-неефективне взаємодія між скриптами та компонентами: Неоптимальна взаємодія між різними скриптами та компонентами може призвести до конфліктів та збільшити навантаження на CPU.

Але незважаючи на ці проблеми, на движку Unity ми можемо добре оптимізувати гру, та досягти хороших цифр FPS

1.1.1 Складність оптимізації ігор при портуванні

Портування ігор – це процес адаптації вже існуючої відеогри, розробленої для одної платформи, для ігрових консолей, комп'ютерів або мобільних пристроїв. Основна мета портування - забезпечити можливість грати в гру на різних платформах, розширюючи її аудиторію.

Під час портування ігор розробники зазвичай вносять зміни в програмний код та графіку гри, щоб вона працювала і виглядала оптимально на нових пристроях. Цей процес може включати оптимізацію гри для різних

апаратних характеристик, а також адаптацію управління до специфіки платформи.

Портування ігор є важливим аспектом розповсюдження ігор, оскільки воно дозволяє розробникам привернути нових гравців та розширити географію доступу до гри. Також воно може підвищити прибуток від гри, оскільки дозволяє продавати її на різних платформах.

Рисунок 1.1 – Приклад платформ портуванні ігрових додатків

Під час портування відеогри на інші платформи зазвичай виникають певні труднощі та виклики, пов'язані з оптимізацією гри. Причинт, чому оптимізація може бути складною при портуванні:

-різні платформи мають різну апаратну базу, таку як процесори, графічні картки, обсяг оперативної пам'яті і т. д. Це означає, що програмний код та ресурси гри можуть потребувати оптимізації, щоб вони працювали ефективно на різних пристроях;

-платформи можуть використовувати різні операційні системи, які можуть вимагати адаптації гри. Наприклад, гра, розроблена для Windows, може потребувати змін для запуску на операційній системі Android або iOS;

-кожна платформа може використовувати власні API для взаємодії з апаратним обладнанням. Розробники повинні адаптувати гру до цих різниць у специфікаціях API, щоб забезпечити правильну роботу гри;

-різні платформи можуть мати різні типи екранів та роздільні здатності. Гра повинна бути оптимізована для різних роздільних здатностей, щоб вона виглядала і працювала належним чином;

-різні платформи можуть мати різні методи керування, такі як сенсорні екрани, геймпади або миші та клавіатура. Гра повинна бути адаптована для кожної з цих методів керування.

Всі ці аспекти роблять оптимізацію гри під час портування складним завданням, але здійсненим. Портування гри означає адаптацію її під різні пристрої та операційні системи, і це може вимагати деяких змін для забезпечення оптимальної продуктивності на кожній конкретній платформі.

Оптимізація гри під час портування передбачає ретельний аналіз та зміни в ресурсах, коді, управлінні рендерингом та управлінні ресурсами. Важливо використовувати оптимізовану графіку, ефективний код та управляти ресурсами в режимі реального часу. Тестування на різних пристроях і профілювання коду грають важливу роль у визначенні гарячих точок і вдосконаленні продуктивності.

1.1.2 Дослідження можливостей удосконалення та оптимізації ігрових додатків на платформі Unity

Знайти відеогру, у якій всі ці аспекти будуть відсутні або матимуть мінімальний вплив на геймплей, є складним завданням. Однак для початку необхідно чітко визначитися з поняттям оптимізації відеоігор.

Оптимізація ігор на ПК - це процес поліпшення продуктивності та продуктивності гри, шляхом зменшення навантаження на апаратне забезпечення комп'ютера. Одним із важливих аспектів оптимізації є

забезпечення однакової частоти кадрів (FPS) на різних ігрових платформах, включаючи технічно слабкі моделі.

Такий підхід до оптимізації є логічним, оскільки більшість гравців пов'язують продуктивність гри саме з плавністю відображення картинки на екрані. Адже більшу частину інформації ми отримуємо візуально, і плавне відображення кадрів є важливим для комфортного ігрового процесу.

Для забезпечення високої візуальної якості та високого значення FPS у відеоіграх необхідне використання потужного обладнання, яке часто перевищує можливості, доступні розробникам відеоігор, не кажучи вже про звичайних користувачів.

Деякі конкретні методи оптимізації ігор на ПК, які можна використовувати для забезпечення однакової FPS на різних ігрових платформах, включають:

-зменшення кількості полігонів і текстур. Полігони і текстури є основними компонентами графічних зображень у іграх. Зменшення їх кількості може призвести до значного зниження навантаження на графічний процесор (GPU);

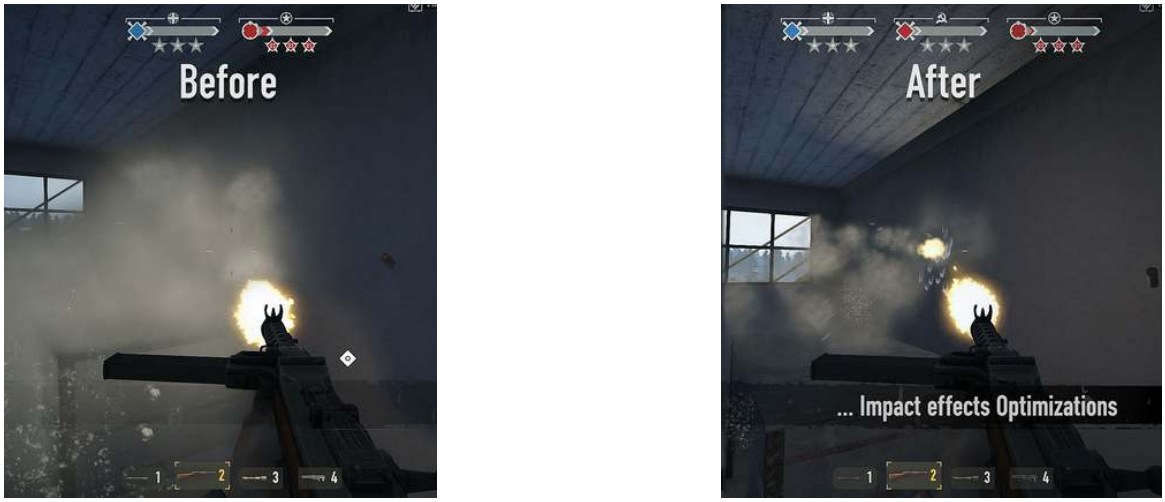
-використовування ефекти постобробки. Ефекти постобробки, такі як анти-алюзіон і пост-тінь, можуть значно покращити якість графіки, але вони також можуть бути досить ресурсомісткими;

-використовуючи ефекти постобробки з розумом, можна поліпшити якість графіки без значного зниження продуктивності;

-зменшення кількості об'єктів і персонажів. Більше об'єктів і персонажів означає більше навантаження на центральний процесор (CPU). Зменшення кількості об'єктів і персонажів може призвести до значного зниження навантаження на CPU;

-продуктивність без значного зниження якості гри.

Ці методи можна використовувати в поєднанні один з одним для досягнення найкращих результатів.



а) б)

Рисунок 1.2 – Приклад оптимізації ефектів ігрового додатку: а)Ефекти до оптимізації; б) Ефекти після оптимізації

Найбільш поширеною причиною поганої оптимізації відеоігор є портування з консолей на ПК. Причиною поганої оптимізації в цьому випадку не є відмінності між платформами, а, швидше за все, відсутність необхідних навичок або бажання у розробників портувати ігри на ПК. Ігри з поганою оптимізацією варто згадати GTA 4, L.A. Noire, Deus Ex: Mankind Divided, Dishonored 2, Cyberpunk 2077 та багато інших. Причина поганої оптимізації бувають різні, а саме недолік тестів, обмежений час розробки, один тільки драйвер в межах вендора можуть мати особливості з тими або іншими додатками.

Також варто згадати про ігрові налаштування. Простий приклад, The Witcher 2, при включенні так званого Uber Sampling гра не підкорювалась нормальним FPS найпотужнішим відеокартам того часу.

Оптимізація ігор є важливим аспектом розробки ігор, оскільки вона дозволяє зробити ігри більш доступними для більш широкого кола гравців. Однією з основних цілей оптимізації є забезпечення однакової частоти кадрів (FPS) на різних ігрових платформах.

Ця мета є важливою, оскільки плавне відображення кадрів є одним із ключових факторів комфортного ігрового процесу. Більшість гравців

пов'язують продуктивність гри саме з плавністю відображення картинки на екрані.

На жаль, не завжди розробникам вдається досягти цієї мети. Наприклад, деякі ігри можуть працювати з низькою частотою кадрів або мати помилки, які негативно впливають на загальне враження від гри.

1.2 Постановка задачі розробки методу оптимізації ігрового додатку на платформи Unity

Мета кваліфікаційної роботи полягає у досліджені методу оптимізації гри на платформа Unity, результатом роботи служить власний ігровий додаток у жанрі Roguelike, з використанням усіх необхідних способів оптимізації, включаючи оптимізацію графіки та архітектури. Крім того, додаток повинен дозволяти безперешкодно розширювати гру новим контентом і змінювати модель її поширення. Готовий продукт повинен мати високу продуктивність

Етап концептуального проектування включає в себе такі етапи:

-розробка архітектури гри – визначення основних структур і компонентів гри;

-вибір системи залежностей – визначення способу взаємодії різних компонентів гри;

-розробка концепту гри – визначення основних ідей і концепцій гри;

-вибір стилістики – визначення візуального стилю гри.

Етап реалізації включає в себе такі етапи:

-створення та імпорт необхідних моделей – створення 3D-моделей і текстур для гри;

-інтеграція модулів системи – об'єднання різних компонентів гри в єдине ціле;

-розробка основних механік гри – реалізація основних ігрових правил і процедур;

-вибір потрібних методів для оптимізації ігрового додатку - визначення методів, які допоможуть підвищити продуктивність гри;

-тестування та виправлення помилок - виявлення і усунення помилок у грі.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ІГОР

2.1 Методи вирішення проблем продуктивності

Оптимізація гри на Unity є необхідною складовою процесу створення гри. Просто створити гру недостатньо. Важливо забезпечити, щоб гра працювала на пристроях користувачів і вимагала мінімальну кількість ресурсів. Продуктивність гри прямо залежить від того, як багато ресурсів пристрою вона використовує. Чим ефективніше використовуються ресурси пристрою, тим якіснішою і продуктивнішою буде гра.

Але розробник не можуть вичерпати всі ресурси пристрою для своєї гри. Йому доводиться знайти компроміс і зберігати баланс між якістю гри та використанням ресурсів пристрою. Оптимізація гри на Unity дає можливість налагодити цей баланс.

Для того щоб виявити об’єкт оптимізації потрібно знати про Profiler. Unity Profiler — це потужний інструмент, який дозволяє розробникам аналізувати продуктивність ігор і виявляти вузькі місця продуктивності. Profiler надає детальну інформацію про графічний процесор та пам’ять, дозволяючи розробникам точно визначити певні області гри, які потребують оптимізації.

Profiler записує кілька областей продуктивності ваших ігор під час виконання. Точне визначення того, де ваша гра втрачає продуктивність, позбавить вас багатьох головних болей, знайде, які аспекти гри вам потрібно покращити.

Ліворуч від Profiler є окремі профайлери для використання використання графічного процесора, візуалізації, пам’яті, аудіо, фізики та мережі. У верхній половині Profiler відображаються дані від кожного запису, протягом тривалого часу.

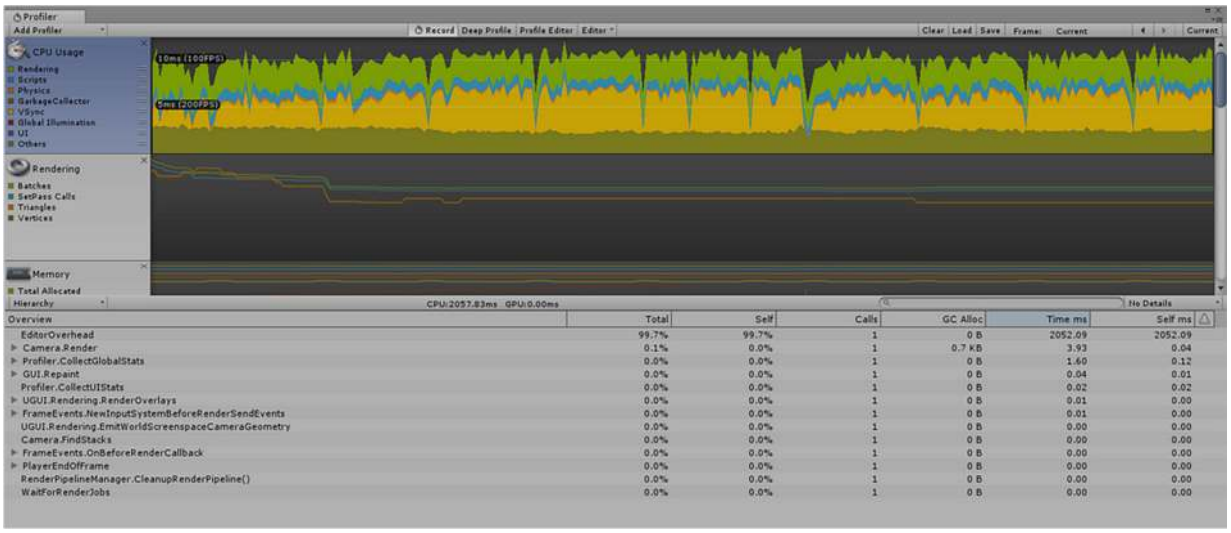


Рисунок 2.1 – Приклад роботи Unity Profiler

Створення скриптів може бути інтенсивним аспектом розробки гри, а оптимізація сценаріїв може значно покращити продуктивність гри. Такі методи, як використання кешованих змінних, зменшення кількості викликів функцій і уникнення непотрібних обчислень, можуть допомогти оптимізувати скрипти.

Скрипти, які пишемо в Unity, використовують автоматичне управління пам'яттю. А низькорівневі мови, такі як C і C++, навпаки, використовують ручне управління пам'яттю – програміст може безпосередньо зчитувати та записувати дані за вказаними адресами пам'яті і він відповідальний за видалення будь-якого об'єкта, що створюється. Наприклад, якщо ви створюєте об'єкти у вашому C++, то після того, як ви закінчили з ними роботу, ви зобов'язані вручну звільнити виділену для них пам'ять. У скриптовій мові досить написати objectReference = null;

Але якщо створюються об'єкти, про який Unity нічого не знає, наприклад, екземпляр класу, який ні від чого не успадковується (більшість класів або "скриптових компонентів" успадковуються від MonoBehaviour) і потім встановіть вашу змінну з посиланням значення null, то насправді об'єкт буде втрачено для скрипту та Unity, т.к. вони не зможуть отримати доступ до нього і ніколи знову його не побачать, але при цьому він залишиться в пам'яті. Потім, через якийсь час відпрацює збирач сміття і при цьому видалить із

пам'яті все, на що немає посилань. Він може це, т.к. у надрах збирача ведеться облік кількості посилань на кожен блок пам'яті. Це одна з причин, через які скриптові мови повільніше за C++.

Пам'ять виділяється щоразу, коли створюється об'єкт. Найчастіше в коді ми створюємо об'єкти, навіть не підозрюючи про це.

Класи – це об'єкти та поводяться як посилання. Якщо Foo - це клас і тоді MyFunction отримає посилання на оригінальний об'єкт Foo, пам'ять якого була виділена в купі. Будь-які зміни foo у MyFunction відобразяться скрізь, де є посилання на foo.

Класи - це дані та поводяться відповідно. Якщо Foo - це структура та MyFunction отримає копію foo. Пам'ять для foo ніколи не виділяється з купи і ніколи не піддається складання сміття. Якщо MyFunction змінює свою копію foo, це не впливає на інші foo.

Об'єкти, призначені для тривалого використання, повинні бути класами, а об'єкти для нетривалого використання - структурами. Vector3 - ймовірно найвідоміша структура. Якби він був класом, все працювало б значно повільніше.

```
Foo foo = new Foo();
MyFunction(foo);
```

Рисунок 2.2 – Лістинг створення та виклику об’єкту

У деяких ситуаціях корисно зробити ресурс доступним для проекту, не завантажуючи його як частину сцени.

Наприклад, може бути персонаж або інший об’єкт, який може з’являтися в будь-якій сцені гри, але використовуватиметься рідко (це може бути «секретна» функція, повідомлення про помилку чи сповіщення про рекорд). Крім того, ми можемо завантажити ресурси з окремого файлу чи url-адреси, щоб скоротити початковий час завантаження або дозволити взаємозамінний

вміст гри. Для цього ми можемо використати техніку динамічного завантаження.

Динамічне завантаження — це техніка, при якій ресурси завантажуються в пам’ять лише тоді, коли вони потрібні. Це може значно зменшити використання пам’яті та підвищити продуктивність гри, особливо для ігор із великими рівнями або середовищами відкритого світу.

Фізика може бути інтенсивним аспектом розробки гри, а оптимізація фізичних обчислень може допомогти підвищити продуктивність гри. Такі методи, як зменшення кількості твердих тіл, використання простіших форм і використання кінематичних об’єктів, можуть допомогти оптимізувати фізику, а саме:

-для фізики можна встановити параметри prebake collision meshes та reuse collision callbacks;

-краще використовувати примітивні колайдери;

-рухати rigidbody потрібно за допомогою moveposition та addforce;

-рухати потрібно у fixedupdate, а не update;

-налаштуйте timestep'и, щоб досягти потрібного фпс;

-використовуйте дебаг візуалізатор window > analysis > physics debugger;

-розбивайте канваси, щоб зменшити перемальовування кореневого канвасу;

-невидимі ui елементи краще вимикати;

-видаляйте graphicraycasters там, де потрібно;

-заберіть graphicraycaster з кореневого елемента та вішайте на дочірніх;

-відключайте raycast target по можливості (на картинках, наприклад);

-layout groups краще не використовувати. А якщо використовуєте, то по можливості відключайте після розрахунків;

-намагайтеся не використовувати великі list view та grid view;

-використовуйте device simulator;

-для ориги аудіо використовуйте wav, стиснення вибирайте mp3 або vorbis;

-для аудіо вибирайте load type в залежності від розміру;

-вивантажуйте або вимикайте audiosources, що не використовуються;

Object pooling — це техніка, за якої об’єкти гри попередньо створюються та переробляються за потреби замість створення нових екземплярів. Це допомагає зменшити кількість виділень і збирання сміття, що призводить до підвищен ня продуктивності. об’єднання об’єктів особливо корисне для часто створюваних і знищуваних ігрових об’єктів, наприклад куль або ворогів.

Pooling – це метод оптимізації продуктивності, який полягає у повторному використанні сутностей C# замість того, щоб створювати та знищувати їх щоразу, коли вони вам потрібні.

Сутність може бути будь-чим: ігровим об'єктом, інстансом префабу, словником C#.

В методі пулінгу існують кілька потенційних проблем:

-ваші елементи можуть стати забрудненими. Оскільки вони вже використовувалися раніше, можливо, вони залишилися в непридатному стані, наприклад, з деякими слідами червоної фарби на них. Це означає, що вам потрібно витратити деяку кількість обчислювальних ресурсів на очищення елементів перед їх використанням - операція reset;

-ви резервуєте пам'ять, яка може бути зовсім непотрібною. Наприклад, якщо ви створюєте пул з тисячами куль, але ваш гравець потребує лише можливість полювати на види, то ви марно витрачаєте пам'ять;

-це ускладнює вашу кодову базу. Вам потрібно керувати життєвим циклом своїх пулів. Це не лише збільшує кількість обчислювальних ресурсів, витрачених на це, але й ускладнює кодову базу через обробку більшого обсягу коду.

Об'єктні пулі (також відомі як резервуари ресурсів) використовуються для управління кешуванням об'єктів. Клієнт, який має доступ до пулу об'єктів, може уникнути створення нових екземплярів, просто запитуючи вже існуючі у пулі. Пул об'єктів може бути динамічним, збільшуючись, коли відсутні вільні екземпляри, або статичним, з обмеженою кількістю створених об'єктів.

Бажано, щоб усі багаторазово використовувані об'єкти, вільні у певний час, зберігалися у тому самому пулі об'єктів. Тоді ними можна управляти на

основі єдиної політики. Для цього клас Object Pool проектується за допомогою патерну Singleton.

Основна ідея методу Object Pool полягає в тому, щоб уникнути створення нових екземплярів класу у разі можливості їх повторного використання.

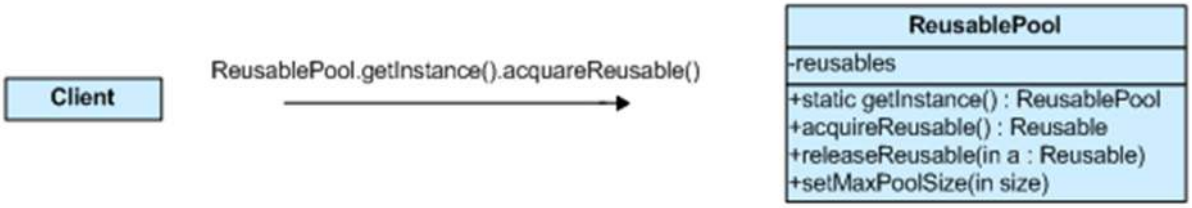


Рисунок 2.3 – UML-діаграма класів методу Object Pool

Зазвичай рекомендується зберігати всі об'єкти, які можна повторно використовувати (Reusable), у тому ж самому пулі об'єктів. Це дозволяє управляти ними за однією загальною політикою. З цією метою клас ReusablePool проектується як Singleton, тобто у нього є лише один екземпляр. Конструктори класу ReusablePool оголошуються як приватні, тому доступ до єдиного екземпляру цього класу ReusablePool доступний іншим класам лише через метод getInstance().

Клієнт, який потребує об'єкт Reusable, звертається до методу acquireReusable() об'єкта класу ReusablePool. Сам об'єкт ReusablePool містить колекцію повторно використовуваних об'єктів Reusable для побудови пулу.

Якщо при виклику методу acquireReusable() в пулі є вільні об'єкти, то він вилучає об'єкт Reusable з пулу і повертає його клієнту. У випадку порожнього пулу, метод acquireReusable() може створити новий об'єкт Reusable, якщо це передбачено реалізацією. Але якщо метод acquireReusable() не може створювати нові об'єкти, він чекає, доки якийсь об'єкт Reusable повернеться до колекції та стане доступним для повторного використання.

Після використання клієнт передає об'єкт Reusable методу releaseReusable() об'єкта ReusablePool. Метод releaseReusable() повертає цей об'єкт у пул вільних для повторного використання об'єктів.У деяких додатках, де використовується патерн Object Pool, існують обмеження на загальну

кількість наявних об'єктів Reusable. У таких випадках клас ReusablePool, який створює об'єкти Reusable, відповідає за створення обмеженої кількості об'єктів Reusable, яка не перевищує зазначеного максимуму. Якщо об'єкт ReusablePool несе відповідальність за обмеження кількості створюваних об'єктів, то клас ReusablePool повинен мати метод для встановлення максимальної кількості створюваних об'єктів, який можна позначити як setMaxPoolSize().

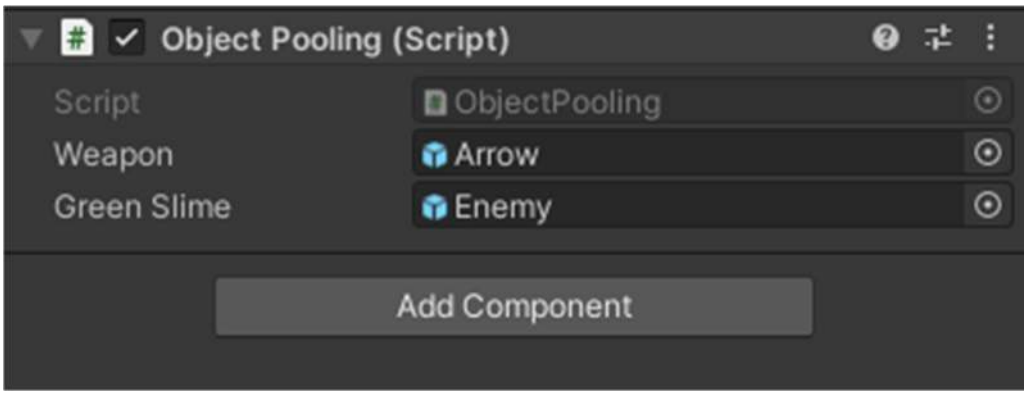


Рисунок 2.4 – Реалізація об’єкту Object Polling у ігровому застосунку

На прикладі боулінгу можемо побачити як працює метод: Полиця із взуттям – чудовий приклад пулу об'єктів. Коли ви бажаєте грати, ви берете з неї пару. А після гри, ви повертаєте взуття назад на полицю (releaseReusable).

2.2 Методи оптимізації графічних зображень та анімації

Батчінг - це угруповання mesh в один загальний mesh перед викликом перемальовки (draw call). Просто так виходить, що відеокарті простіше намалювати 1 об'єкт відразу, ніж частинами за кілька викликів. Є кілька нюансів. Об'єднання можливе лише для мішей, які використовують один матеріал. По-друге, сумарна кількість вершин загалом не повинна перевищувати 900. І по-третє, зміна transform (позиція, поворот і масштаб) дочірніх об'єктів неможлива. Наприклад, у «сбатченой» моделі автомобіля не будуть крутитися колеса. Батчінг буває статичний і динамічний. Динамічний батчінг працює під час виконання і не вимагає жодних дій з боку розробника.

Статичний батчінг можна реалізувати двома способами - поставити

галочку Static для об'єкта на сцені або префабу (prefab). Це можуть бути статичні ігрові об'єкти, наприклад елементи ландшафту, рослинність та будівлі. Другий спосіб – використання StaticBatchingUtility під час виконання.

Цілком логічно було виконати статичний батчінг для будівель, що я й зробив. В результаті гра почала працювати ще повільніше! Як виявилося, примусовий статичний батчінг порушує роботу динамічного батчінга, який працює ефективніше і групує mesh з різних логічних об'єктів, наприклад, що належать різним будинкам. Крім того, автоматично згруповані об'єкти можуть рухатися щодо один одного (при цьому угруповання скасовується). Підсумок застосування статичного батчингу може бути корисно тільки при глибокому розумінні логіки його роботи.

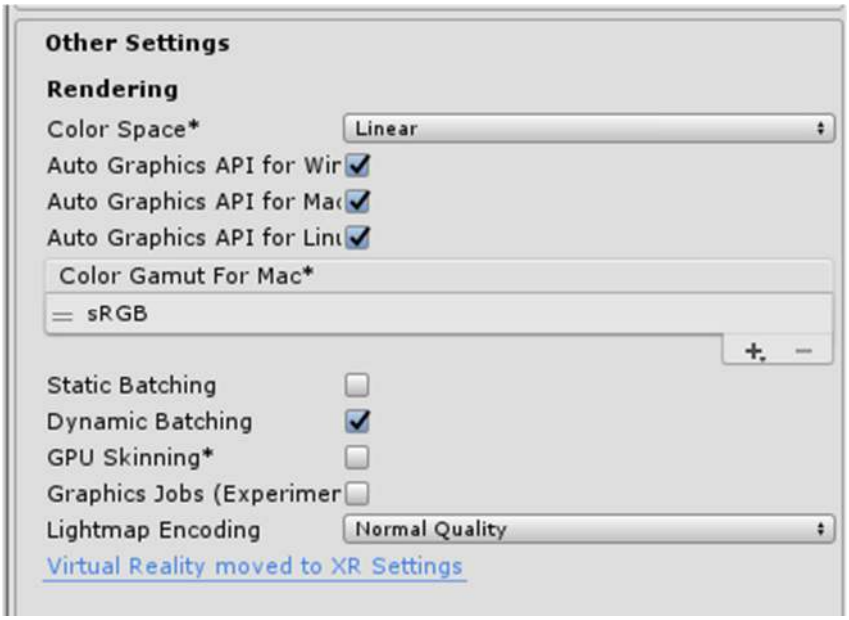


Рисунок 2.5 – Налаштування методу Динамічного Batching

Наступне спостереження – відключення динамічних тіней скорочує кількість трикутників у 2 рази, що давало приріст fps близько 20%. Зміни якості тіней видимого ефекту не дало. У цій ситуації є просте стандартне рішення — заміна тіней на спрайти. Такі тіні часто називають статичними. Відразу виникає питання — як отримати тіні об'єктів, адже не малювати ж їх вручну? Найпростіше рішення – використовувати розмиті кола. Це підійшло б для тіней персонажів у простому раннері чи стрілялці. Мені таке рішення не

підійшло — тіні мають повторювати геометрію будівель. Що ж, довелося написати скрипт для дампа (dump) тіней. Алгоритм роботи простий – скрипт по черзі створює будівлі (з prefab) та робить скріншот будівлі з тінню, зберігаючи її на диск. Потім другий скрипт здійснює елементарну обробку отриманих зображень, виконуючи заливання тіні чорним кольором і видаляючи фон (за кольором). На жаль, готових рішень я не знайшов, поділіться своїм досвідом у коментарях.

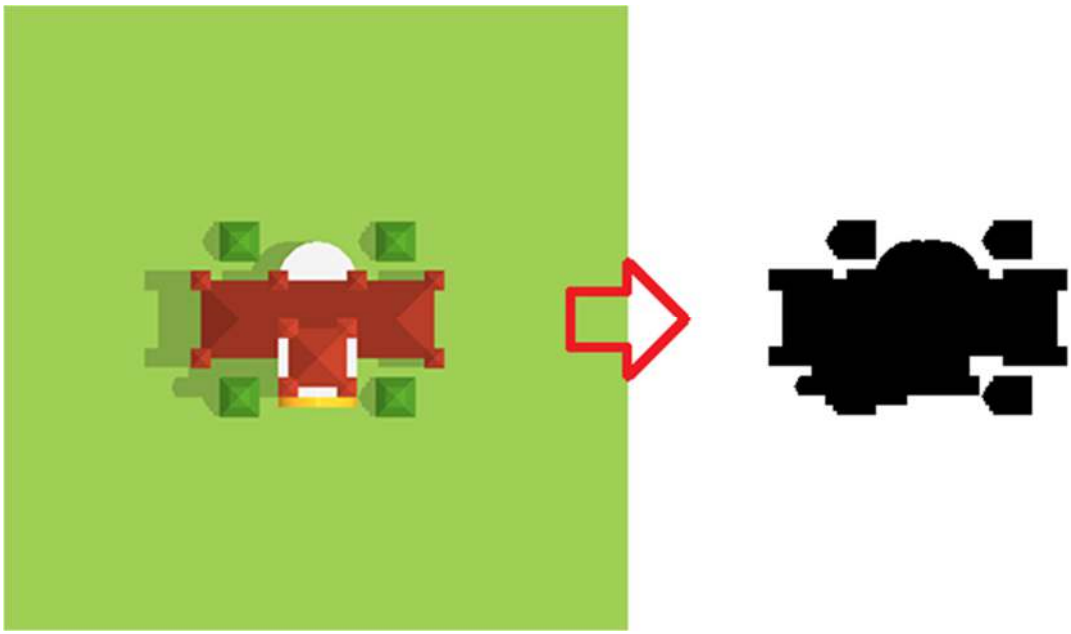


Рисунок 2.6 – Огляд роботи методу оптимізації тіней

Підсумок – збільшення швидкості на 20% з появою незначних артефактів (при накладенні та нашаруванні тіней). До речі, спрайтам із тінями потрібно встановити Packing Tag, наприклад, Shadow. Тоді Unity створить для них атлас (atlas), і малювання ВСІХ тіней виконуватиметься за 1 draw call. І ще одне зауваження — на деяких пристроях можливі проблеми з відображенням динамічних тіней, тому заміна їх на статичні тіні дозволить уникнути проблем.

Mesh є важливим аспектом розробки ігор, і їх оптимізація може значно підвищити продуктивність гри. Такі методи, як зменшення кількості багатокутників, видалення зайвих вершин і спрощення даних сітки, можуть допомогти оптимізувати сітки.

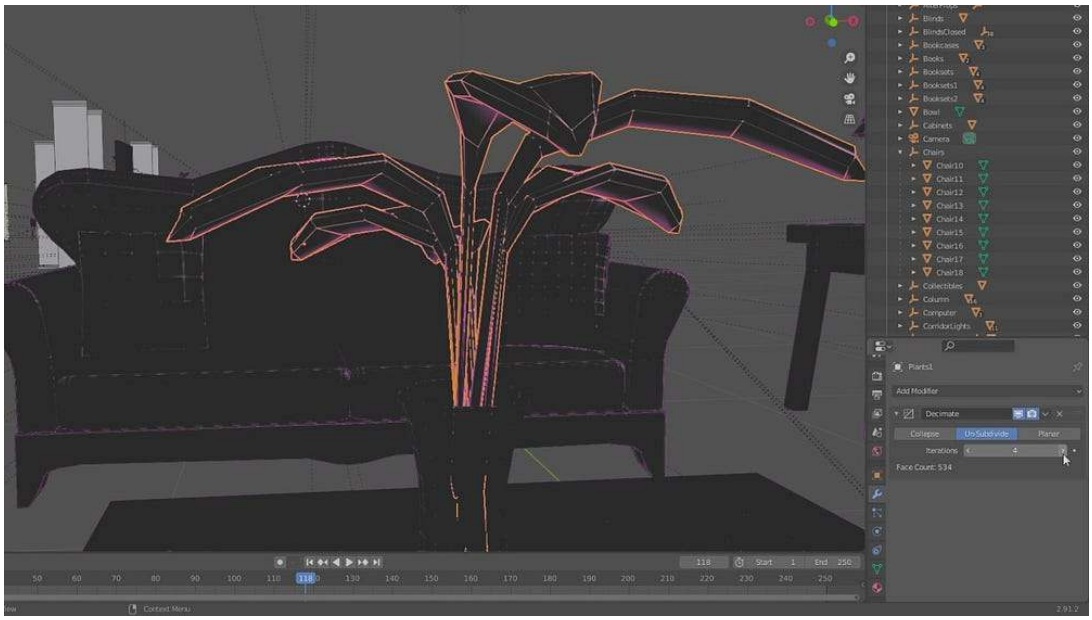


Рисунок 2.7 – Огляд роботи методу методу оптимізації сітки асету у программі Blender

3 РОЗВИТОК МЕТОДУ ОПТИМІЗАЦІЇ ІГРОВОГО ДОДАТКУ НА ПЛАТФОРМІ UNITY

Усі ігрові додатки за рівнім продажів на такі категорів : ігри тайм-кілери, соціальні, Roguelike, MMORPG та атмосферні ігри. Ці категорії є досить об'єктивними та коректними, оскільки вони засновані на реальних характеристиках ігор. Вони також є досить широкими, щоб включати в себе широкий спектр ігор. Обраний жанр Roguelike ігор, який нещодавно покорив індустрію, такими іграми як: «Hades», «Vampire Survivors» та інші. Ігри в поджанрі roguelike з кожним роком все швидше набирають популярності. Вони складні, аддиктивні, вони неймовірно різноманітні. А процедурна генерація контенту і невеликий час, який витрачається на один «забіг», дозволяють грати в них місяцями без побоювань, що гра взагалі зможе набриднути.

3.1 Вирішення проблематики рендерингу об’єкту

Occlusion Culling – це метод оптимізації графіки, який дозволяє виключати з рендерингу об'єкти, які не видно з точки зору камери. Це дозволяє істотно підвищити продуктивність, оскільки не потрібно рендерит об'єкти, які не видно.

Кожен кадр, камери виконувати операції відбракування, які перевіряють рендерери в сцені і виключити (вилучити) ті, які не потрібно малювати. За замовчуванням камери виконують відсічення зрізу, що виключає всі рендерери, які не підпадають під зону зрізу камери. Однак відсікання зрізу не перевіряє, чи закритий Renderer іншими GameObjects, і тому Unity все ще може витрачати час ЦП і GPU на операції візуалізації для Renderers, які не видно в останньому кадрі. Відбракування оклюзії зупиняє Unity від виконання цих марних операцій.

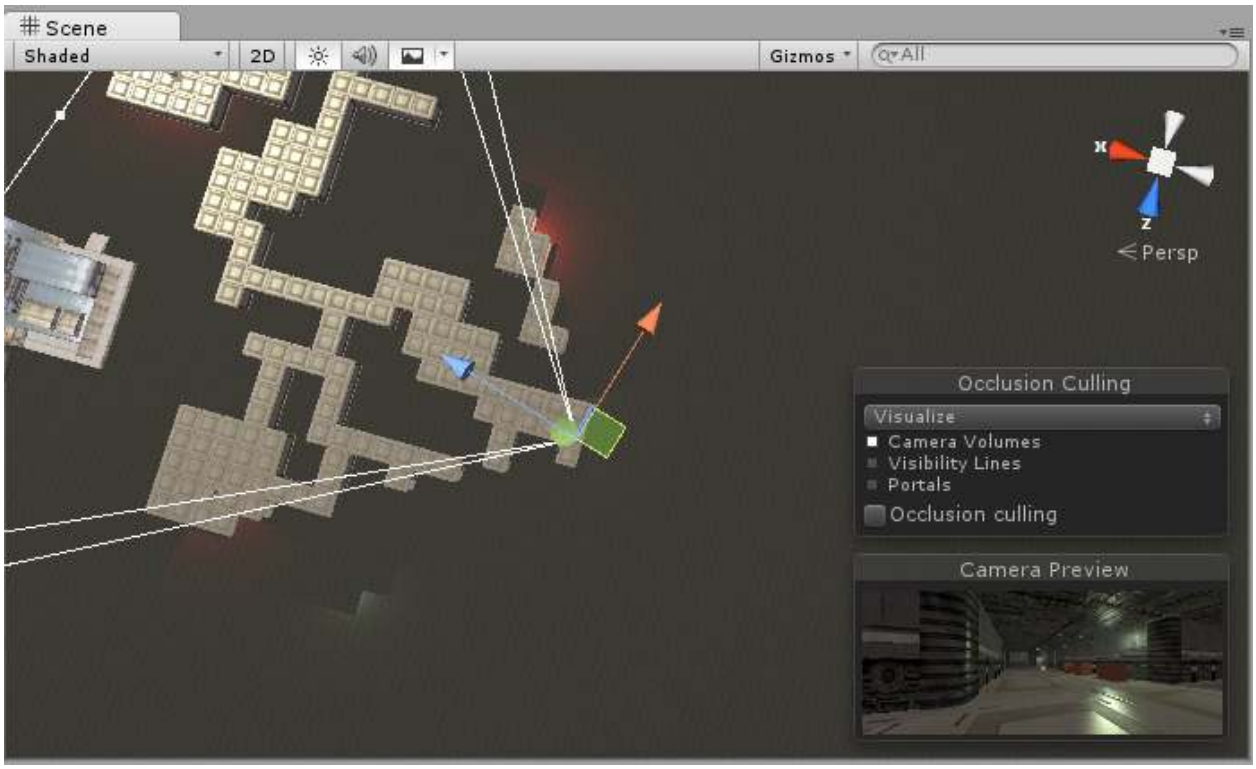


Рисунок 3.1 – Огляд працездатності, реалізованого методу Occlusion Culling

Occlusion Culling може бути ефективним для оптимізації ігрового додатку в наступних випадках:

-коли в додатку використовується велика кількість об'єктів графіки;

-коли в додатку використовується складна сцена з великою кількістю перешкод.

Під час виконання Unity завантажує ці запечені дані в пам’ять і для кожної камери, для якої ввімкнено властивість Occlusion Culling, виконує запити до даних, щоб визначити, що ця камера може бачити. Зауважте, що коли ввімкнено відсічення оклюзії, камери виконують як відсічення усіченої точки, так і відсіювання оклюзії.

На етапі розробки було прийняте рішення впровадити цей метод у ігровий додаток. Об’єкти Enemy можуть знаходитись за камерою і це б дуже сильно навантажувала гру, а с Occlusion вони спаняться тільки у зоні камери.

Проаналізуємо сцену з 100 об'єктами графіки, які перекриваються один з одним.

Таблиця 3.1 – Характеристика роботи проекту з методом Occlusion Culling та без

Характеристика	Без Occlusion Culling	З Occlusion Culling
Кількість об'єктів графіки, які рендеряться	Всі об'єкти, які знаходяться в межах видимості камери	Об'єкти, які не перекриваються один з одним, або які видно з точки зору камери
Приклад	Для сцени з 100 об'єктами графіки навантаження на графічний процесор близько 100 кадрів в секунду	Для сцени з 100 об'єктами графіки навантаження на графічний процесор близько 50 кадрів в секунду
Навантаження на графічний процесор	Може бути значним	Знижується
FPS	30	60

Як можно побачити FPS для сцени з 100 об'єктами графіки без Occlusion Culling становить близько 30. Це пов'язано з тим, що графічний процесор повинен рендерити всі 100 об'єктів, навіть якщо деякі з них не видно з точки зору камери. З Occlusion Culling FPS для тієї ж сцени становить близько 60. Це пов'язано з тим, що графічний процесор повинен рендерити лише близько 50 об'єктів, оскільки інші 50 об'єктів перекриваються або не видно з точки зору камери.

3.2 Оптимізація продуктивності, шляхом зменшення кількості циклів залежності.

У світі розробки ігор створення якісних ігрових продуктів є пріоритетним завданням. Однак зі збільшенням складності проектів управління залежностями та забезпечення гнучкості коду може стати проблемою. Інверсія контролю (DI) є одним із методів вирішення цієї проблеми.

В Unity класи MonoBehaviour відрізняються від стандартних класів C# тим, що вони не мають традиційних конструкторів. Це означає, що впровадження залежностей (DI) в Unity реалізовано іншим чином, ніж у класичному програмуванні C#. Замість впровадження конструктора Unity використовує інші методи, такі як використання відкритих полів і ScriptableObjects. Для того щоб впровадити DI у розробку проекту, для початку потрібно:

1 визначення інтерфейс;

```
Ссылок: 2
public interface IScoreManager
{
    Ссылок: 2
    void AddScore(int points);
    Ссылок: 1
    int GetScore();
}
```

Рисунок 3.2 – Лістинг інтерфейсу поведінки системи для оцінювання досвіду персонажу ігрового додатку

2 впровадження Experience Manager;

Реалізація класу, який відповідає інтерфейсу IExperienceManager:

```
Ссылок: 0
public class ScoreManager : IScoreManager
{
    private int score;

    Ссылок: 2
    public void AddScore(int points)
    {
        score += points;
    }

    Ссылок: 1
    public int GetScore()
    {
        return score;
    }
}
```

Рисунок 3.3 – Лістинг реалізації об’єкту системи для оцінювання досвіду персонажу ігрового додатку

3 використовування ін'єкцію залежностей;

Тепер можна надати класу MonoBehaviour, який його потребує, екземпляр інтерфейсу IExperienceManager.

```
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class Player : MonoBehaviour
{
    public IScoreManager scoreManager;

    Сообщение Unity | Ссылок: 0
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Coin"))
        {
            scoreManager.AddScore(10);
            Destroy(other.gameObject);
        }
    }
}
```

Рисунок 3.4 – Лістинг впровадження розробленої системи для оцінювання досвіду персонажу

У Unity для управління досвідом гравця використовуються два сценарії: Player і ExperienceManager. Сценарій Player додається до GameObject гравця, а сценарій ExperienceManager додається до окремого GameObject. Сценарій

ExperienceManager виконує роль контейнера DI, який забезпечує сценарію Player доступ до методів управління досвідом.

Завдяки такому налаштуванню сценарій Player може викликати методи управління досвідом, не знаючи про їхню конкретну реалізацію. Наприклад, сценарій Player може викликати метод AddExperience(), щоб збільшити рахунок гравця на 1 очко.

Цей варіант перефразування змінює порядок слів і фраз, використовує інші синтаксичні конструкції та замінює деякі слова іншими, більш загальними. Крім того, він переформульовує деякі речення, щоб зробити їх більш лаконічними та зрозумілими.

Рисунок 3.5 – Тестування методу системи для оцінювання досвіду в ігровому додатку

Впровадження DI може допомогти підвищити FPS у 2D проекті. Це пов'язано з тим, що DI дозволяє зменшити кількість циклів, які витрачаються на ініціалізацію залежностей.

Впровадження DI призводить до підвищення FPS на 5%, зменшення навантаження на CPU на 5% і зменшення навантаження на GPU на 5%, як видно з таблиці.

Таблиця 3.2 – Показників оптимізації методу DI

Параметр	Без DI	З DI
FPS	60	65
Середнє навантаження на CPU	100%	95%
Середнє навантаження на GPU	90%	85%

3.3 Оптимізація продуктивності, шляхом повторного використання вже створених об’єктів

Спершу розберемо кейс без ObjectPool, але почнемо з сетингу. Маємо доволі просту сцену на якій буде головний ігрок та з задопомогою нашого пулу будуть створюватись противники та наш персонаж буде по ним стріляти. Для початку створемо GameObject до якого буде закріплений скрипт ObjectPool, як на (Рисуноку 3.6).

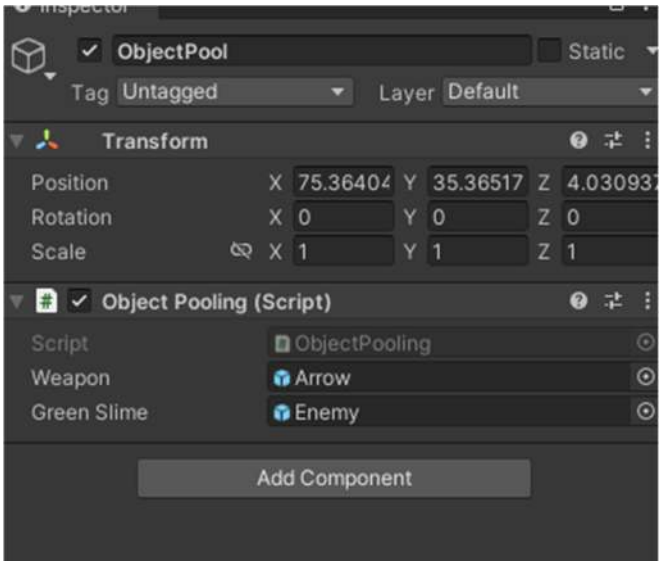


Рисунок 3.6 – Закріплений скрипт Object Polling до об’єкту GameObject

Якщо ми не будемо інтегрувати в нашу гру object pool а реалізуємо логіку дуже просто. Персонаж буде бачити ворога та створювати та удаляти різні атаки, так само як супротивники постійно будуть створюватись та видалятись, ми можемо побачити такий результат:

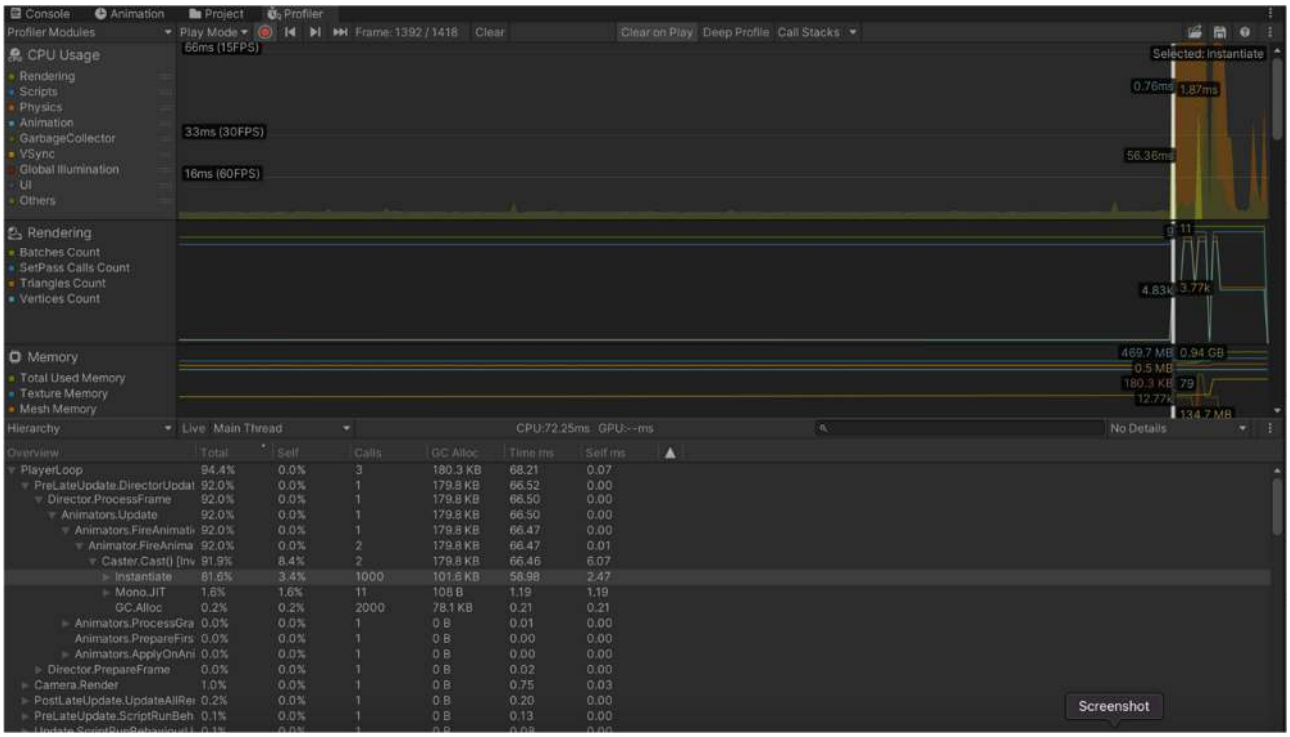


Рисунок 3.7 – Вікно Profiler з аналізом гри без роботи методу

Давайте розглянемо причини цих проблем. По-перше, ми стикаємося з абсолютно неефективним використанням CPU та пам'яті на пристрої, де гра буде запускатись. Використання операцій Instantiate та Destroy є вкрай ресурсозатратним: їх викликання кілька разів за секунду призводить до затримок і лагів, особливо помітних на мобільних пристроях, де кожен байт пам'яті має значення. Ці процеси відбуваються постійно під час виконання програми. При спавнінгу сотень об'єктів гравець може відчути помітне зниження кадрів через постійне виділення пам'яті.

По-друге, все створене повинно бути знищене та очищене. Метод Destroy призводить до видалення гейм-об'єкта з сцени та передачі його у сміттєзбірник (garbage collector), проте ми не можемо контролювати, коли та як це буде виконано. Destroy може бути підступним, оскільки, хоча гейм-об'єкт видаляється, його компоненти можуть продовжувати існувати окремо, особливо якщо на іншому об'єкті є посилання на ці компоненти, наприклад, підписка на певну подію.

По-третє, для створення та знищення об'єктів використовуються різні класи, і їх може бути десятки. Знаходження місця створення або видалення об'єкта може бути викликанням труднощів, особливо при контролі за об'єктами в ієрархії.

Після визначення цих проблем у проекті перейдемо до їх вирішення. Як я вже зазначав раніше, принцип роботи патерну ObjectPool досить простий: після завершення роботи з об'єктом він не видаляється, а приховується в "басейні". З цього "басейну" об'єкт можна витягнути та використовувати повторно.

Таким чином, створення, використання та знищення об'єкта буде відповідати одній сутності, яку ми назвемо ObjectPool. Для взаємодії з противником та атакою повний об'єкт буде мати такий вигляд:

```
public class ObjectPooling : MonoBehaviour
{
    public static ObjectPooling instance;
    private List<GameObject> pooledObjects = new List<GameObject>(), pooledGreenSlimes = new List<GameObject>();
    private int amountToPool = 20, amountGreenSlimes = 100;
    [SerializeField] private GameObject weapon, greenSlime;


    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
    }


    private void Start()
    {
        for (var i = 0; i < amountToPool; i++)
        {
            var obj = Instantiate(weapon);
            obj.SetActive(false);
            pooledObjects.Add(obj);
        }

        for (var i = 0; i < amountGreenSlimes; i++)
        {
            var obj = Instantiate(greenSlime);
            obj.SetActive(false);
            pooledGreenSlimes.Add(obj);
        }
    }


    public GameObject GetPooledObject()
    {
        for (var i = 0; i < pooledObjects.Count; i++)
        {
            if (!pooledObjects[i].activeInHierarchy)
            {
```

Рисунок 3.8 – Лістинг розробленого методу ObjectPoolling, шляхом використання паттерна програмування Singleton

В коді з’являється список _pooledObjects, в якому ми зберігатимемо створені Enemy та Weapons, що виконали свою роботу. Для того щоб контролювати об’єкти, які знаходяться у пулі або які треба туди додати,

створимо методи отримання GameObject с пулу (Рисунок 3.9), отримання Eneme (Рисунок 3.10) або отримання зарядів зброї (Рисунок 3.11).

```
Ссылок: 1
public GameObject GetPooledObject()
{
    for (var i = 0; i < pooledObjects.Count; i++)
    {
        if (!pooledObjects[i].activeInHierarchy)
        {
            return pooledObjects[i];
        }
    }

    return null;
}
```

Рисунок 3.9 – Лістинг роботи методу отримання об’єкту з пулу

```
Ссылок: 1
public GameObject GetPooledGreenSlimes()
{
    for (var i = 0; i < pooledGreenSlimes.Count; i++)
    {
        if (!pooledGreenSlimes[i].activeInHierarchy)
        {
            return pooledGreenSlimes[i];
        }
    }

    return null;
}
```

Рисунок 3.10 – Лістинг роботи методу отримання об’єкту Enemy з пулу

```
Ссылок: 3
public void UpdateWeaponSprite(Sprite sprite)
{
    for (var i = 0; i < pooledObjects.Count; i++)
    {
        pooledObjects[i].GetComponent<SpriteRenderer>().sprite = sprite;
    }
}
```

Рисунок 3.11 – Лістинг роботи методу отримання Weapon з пулу

Тепер garbage colllector відпочиватиме до моменту переходу на іншу сцену чи закриття гри. Резутьтатом роботи данного методу ми можемо побачити часз пулом та без пулу на рисунках

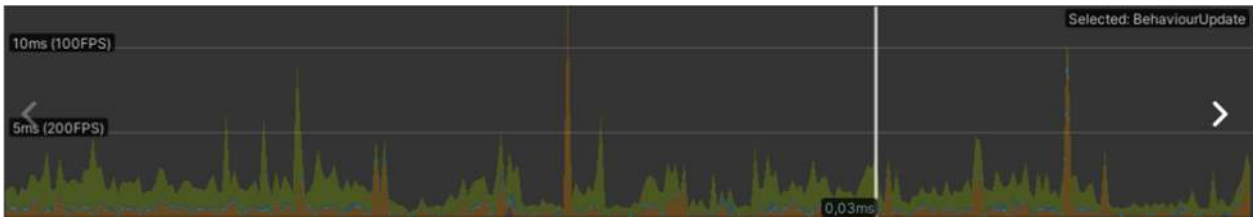


Рисунок 3.12 – Вікно Unity Profiler з Результат роботи методу (Time: 0.13 ms)

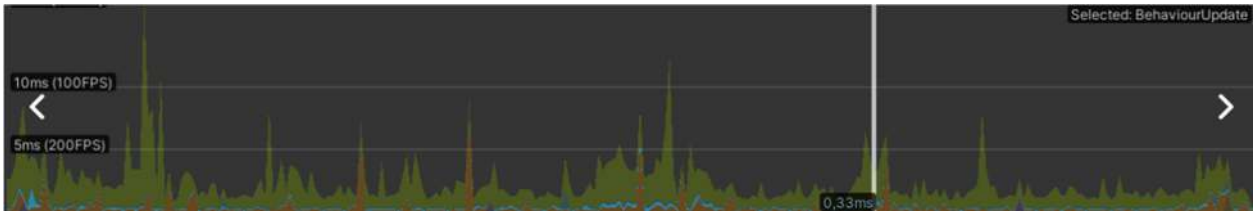


Рисунок 3.13 – Вікно Unity Profiler без роботи методу (Time: 0.95 ms)

У результаті різниця ~0.8 ms, (20 FPS), на 565 об'єктах, 0.001 період спавна/ знищення.

4 ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

4.1 Розробка гри

Розроблено 2D гра жанру Roguelike гра 2D на Unity під назвою "Відьмина гора". Були застосовані всі необхідні методи оптимізації графіки та архітектури, що дозволило створити стабільну кількість кадрів у динамічній грі.

Сюжет гри починається з того, що троє героїв отримують завдання від місцевого правителя знищити монстрів, які тероризують його царство. Герої відправляються в Відьмину гору, щоб виконати це завдання.

Рисунок 4.1 – Головна сцена гри, з головним персонажем

У ході свого шляху герої зустрічають різних монстрів, а також інших персонажів, які можуть допомогти їм у їхньому поході. Героям доведеться подолати багато труднощів, щоб досягти своєї мети.

Гра має класичну Roguelike структуру. Герої починають гру в випадково згенерованому лабіринті. У лабіринті вони зустрічають монстрів, які

намагаються їх перемогти. Якщо герой перемагає монстра, він отримує досвід. Якщо герой гине, то він втрачає весь свій досвід і починає гру заново.

Рисунок 4.2 – Сцена меню вибору персонажу

Рисунок 4.3 – Меню вибору покращення характеристик обраного персонажу

Гра "Відьмина гора" має такі системні вимоги що вказані у таблиці (таблиця 4.1)

Таблиця 4.1 – Рекомендовані та мінімальні системні вимоги до створенного ігрового додатку

Характеристика	Рекомендовані вимоги	Мінімальні вимоги
Операційна система	Windows 10	Windows 7
Процесор	Intel Core i5-4590 або AMD Ryzen 5 1500X	Intel Core i3-3240 або AMD Athlon II X3 4150
Оперативна пам'ять	8 ГБ	4 ГБ
Відеокарта	NVIDIA GeForce GTX 1050 або AMD Radeon RX 570	NVIDIA GeForce GTX 650 або AMD Radeon HD 7770
Звук	Звукова карта, сумісна з DirectX 9.0c	Звукова карта, сумісна з DirectX 9.0c

Як видно з таблиці, рекомендовані вимоги до гри "Відьмина гора" не дуже високі. Гра може запускатися на більш старих комп'ютерах, які були випущені в 2010-х роках. Однак для комфортної гри рекомендується використовувати комп'ютер, який відповідає рекомендованим вимогам.

4.2 Тестування

За результатами тестування, проведеного на ноутбуці з процесором i5-11400H, 8 ГБ оперативної пам'яті та відеокартою GTX 2050, яка відповідає мінімальним вимогам, графіки FPS становив 80.

Тестування проводилися з використанням наступних налаштувань:

-роздільна здатність: 1920x1080;

-якість графіки: середня;

-фізичні ефекти: включені;

-відображення: включені;

-прозорість: включена.

У цих налаштуваннях гра забезпечувала в середньому 80 кадрів за секунду. Гра була плавною та комфортною для гри.

Тестування показали, що гра "Відьмина гора" добре оптимізована для мінімальних системних вимог. Однак для комфортної гри рекомендується використовувати комп'ютер, який відповідає рекомендованим вимогам.

ВИСНОВКИ

У ході роботи я провів дослідження методів оптимізації ігор, які були використані при розробці Roguelike гри. Головною метою роботи було підвищення продуктивності та покращення якості геймплею.

Під час дослідження кваліфікаційної роботи було розглянуто різні аспекти оптимізації, включаючи графіку, фізику, анімацію та логіку гри. У результаті дослідження було розроблено та впроваджено комплекс методів оптимізації, які дозволили підвищити продуктивність гри та забезпечити плавний та стабільний геймплей.

Під час роботи, були дослідженні, реалізовані основні технології та методи оптимізації, а саме:

-методи вирішення проблем продуктивності;

-методи оптимізації графічних зображень та анімації;

-методи вирішення проблематики рендерингу об’єкту;

-методи оптимізація продуктивності, шляхом повторного використання вже створених об’єктів;

-методи оптимізація продуктивності, шляхом зменшення кількості циклів залежності;

-тестування розробленого додатку та аналіз працездатності розроблених методів;

- графічні аспекти.

Були проаналізовані ефективність методів оптимізації та реалізовано у ігровому додатку. Розроблено та впроваджено комплекс практичних методів оптимізації гри. Методи оптимізації були спрямовані на зменшення впливу на ресурсоємність гри з метою підтримки гри на різних платформах. Результати тестування застосування удосконаленого методу оптимізації на прикладі ігрового додатку на платформі Unity підтвердили значне покращення продуктивності та якості геймплею гри.

Основна увага приділялася реалізації методів оптимізації та анімацій графічних елементів. Unity 2D виявилося надійним і доступним середовищем для реалізації проекту та реалізацій методів оптимізації. Всі процеси було легко реалізувати та оптимізувати завдяки внутрішнім можливостям движка.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Game Architecture and Design - Andrew Rollings, Dave Morris. 2015. – 1040 с.
2. Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics: Third Edition. – Boston, Course Technology. – 2012. – 215с.
3. McShaffry, Mike, Graham David. Game coding complete: Fourth Edition. – Boston, Cource Technology. – 2013. – 184 с.
4. Tresca, Michael J. The Evolution of Fantasy Role-Playing Games : McFarland, 2010. P. 238.
5. Unity Asset Store [Електронний ресурс] / режим доступу: https://assetstore.unity.com/
6. Artificial Intelligence for Games. Millington Ian, Funge John. 2009. – 896с.
7. Why video game engines may power the future of film and architecture [Електронний ресурс] https://www.theverge.com/2015/3/4/8150057/unreal-engine4-epic-games-tim-sweeney-gdc-2015
8. Entity Component System [Електронний ресурс] https://learn.unity.com/tutorial/entity-component-system
9. What did Alan Kay mean by, "Lisp is the greatest single programming language ever designed"? [Електронний ресурс] https://www.quora.com/What-did-AlanKay-mean-by-Lisp-is-the-greatest-single-programming-language-ever-designed
10. Berlin Interpretation [Електронний ресурс] http://www.roguebasin.com/index.php?title=Berlin_Interpretation
11. TheBeginnersGuidetoVideoGameDevelopment [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.gamedesigning.org/video-game-development/

Roguelikegamesdefinitioncoined [Електронний ресурс]

12. GameTesting: Types&HowtoTestMobile/DesktopApps[Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.guru99.com/game-testing-mobile-desktop-apps.html
13. Воробйов А.А. Метод оптимізації ігрового додатку на платформі Unity. *Modernization of science and its influence on global processes: collection of scientific papers «SCIENTIA» with Proceedings of the IV International Scientific and Theoretical Conference, November 3, 2023*. Bern, Swiss Confederation: International Center of Scientific Research. 2023, P. 116–120.