Міністерство освіти і науки України

Ніжинський державний університет імені Миколи Гоголя

Навчально-науковий інститут природничо-математичних, медико-біологічних наук та інформаційних технологій

Кафедра інформаційних технологій, фізико-математичних та економічних наук

Освітня програма: Комп’ютерні науки

Спеціальність:122 Компʼютерні науки

**КВАЛІФІКАЦІЙНА РОБОТА**

на здобуття освітнього ступеня *бакалавр*

**Оптимізація коду при створені ігрового застосунку в жанрі "hypercasual" в Unity**

студента **Довди Нікіти Олександровича**

**Науковий керівник:**

Лисенко Ірина Миколаївна,

кандидат фізико-математичних наук, доцент

**Рецензент**:

Бугаєць Наталія Олександрівна,

к. пед. н., доцент

**Допущено до захисту:** \_\_\_\_ \_\_\_\_\_\_ 2024 р.

Завідувач кафедри

проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Казачков І.В.

Ніжин – 2024

АНОТАЦІЯ

Метою даної кваліфікаційної роботи є дослідження методів оптимізації коду при створенні ігрового застосунку в жанрі "hypercasual" за допомогою інструментів та підходів, наданих Unity. Об'єктом дослідження є процес оптимізації ігрового застосунку. Предмет дослідження – методи оптимізації коду при створені ігрового застосунку на платформі Unity.

Кваліфікаційна робота включає в себе вступ, основну частину, висновки, список використаних джерел.

У першому розділі описуються теоретичні основи жанру «hypercasual» та його ринковий контекст. Крім того, аналізуються особливості геймдизайну в цьому жанрі та специфіка аудиторії. У другому розділі розглядаються існуючі методи оптимізації коду, а також вбудовані інструменти в Unity для оптимізації гри. В кінці розділу наведені приклади та порівняння отриманих результатів. У третьому розділі визначається ідея для гри та її проектування з використанням оптимізаційних методів.

**Ключові слова:** оптимізація, код, ігровий застосунок, hypercasual, Unity, геймдизайн.

ABSTRACT

The purpose of this qualification work is to study code optimization methods when creating a game application in the "hypercasual" genre using the tools and approaches provided by Unity. The object of the study is the process of optimizing a game application. The subject of the study is the methods of code optimization in creating a game application on the Unity platform.

The qualification work includes an introduction, the main part, conclusions, a list of used sources.

The first chapter contains the theoretical foundations of the "hypercasual" genre and its market context. In addition, the features of game design in the same genre and the specifics of the audience are described. The second chapter examines existing methods of code optimization, as well as the built-in tools in Unity for optimizing the game. At the end of the chapter, examples and comparison of the obtained results are given. The third chapter defines the idea for the game and its design using optimization methods.

**Keywords:** optimization, code, game application, hypercasual, Unity, game design.

ЗМІСТ

[ВСТУП 6](#_Toc167995016)

[РОЗДІЛ 1 8](#_Toc167995017)

[ТЕОРЕТИЧНІ ОСНОВИ ЖАНРУ «HYPERCASUAL» ТА ЙОГО РИНКОВИЙ КОНТЕКСТ 8](#_Toc167995018)

[1.1 Визначення жанру «Hypercasual» 8](#_Toc167995019)

[1.2 Історія розвитку жанру 9](#_Toc167995020)

[1.3 Аналіз поточного ринку 11](#_Toc167995021)

[1.4 Особливості гейм-дизайну 13](#_Toc167995022)

[1.5 Специфіка аудиторії 15](#_Toc167995023)

[1.6 Вплив гіперказуальних ігор на ігрову індустрію 15](#_Toc167995024)

[1.7 Висновки до розділу 1 15](#_Toc167995025)

[РОЗДІЛ 2 17](#_Toc167995026)

[МЕТОДИ ОПТИМІЗАЦІЇ РОЗРОБКИ З ВИКОРИСТАННЯМ UNITY 17](#_Toc167995027)

[2.1 Огляд інструментів Unity та оптимізації 17](#_Toc167995028)

[2.1.1 Unity Profiler 17](#_Toc167995029)

[2.1.2 Frame Debugger 18](#_Toc167995030)

[2.1.3 Physics Debugger 19](#_Toc167995031)

[2.2 Способи оптимізації проєкту в Unity 20](#_Toc167995032)

[2.2.1 Батчинг виклику графічних об'єктів 20](#_Toc167995033)

[2.2.2 Користувацький інтерфейс Unity UI 21](#_Toc167995034)

[2.2.3 Occlusion Culling та Frustum Culling 22](#_Toc167995035)

[2.3 Оптимізація коду в C# та Unity 23](#_Toc167995036)

[2.3.1 Використання Update методів 23](#_Toc167995037)

[2.3.2 Клас Component та його кешування 24](#_Toc167995038)

[2.3.3 Debug log інструкції 25](#_Toc167995039)

[2.4 Оптимізаційні шаблони проєктування 26](#_Toc167995040)

[2.4.1 Пул об’єктів 26](#_Toc167995041)

[2.4.2 Шаблон проєктування Singleton 29](#_Toc167995042)

[2.5 Методи оптимізації ресурсів проєкту 31](#_Toc167995043)

[2.5.1 Налаштування проєкту 31](#_Toc167995044)

[2.5.2 Стиснення текстур 33](#_Toc167995045)

[2.5.3 Стиснення мешів. 36](#_Toc167995046)

[2.5.4 Addressable Asset System 38](#_Toc167995047)

[2.6 Висновки до розділу 2 40](#_Toc167995048)

[РОЗДІЛ 3 41](#_Toc167995049)

[СТВОРЕННЯ ГРИ З ВИКОРИСТАННЯМ ОПТИМІЗАЦІЙНИХ МЕТОДІВ В UNITY 41](#_Toc167995050)

[3.2 Аналіз проблеми та тестування гри 42](#_Toc167995051)

[3.2.1 Аналіз викликів «Draw calls» 42](#_Toc167995052)

[3.2.2 Аналіз використання Occlusion culling 44](#_Toc167995053)

[3.3 Оптимізація коду 45](#_Toc167995054)

[3.3.2 Оптимізація викликів методу GetComponent<T>() 49](#_Toc167995055)

[3.3 Висновки до розділу 3 53](#_Toc167995056)

[ВИСНОВКИ 54](#_Toc167995057)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55](#_Toc167995058)

ВСТУП

**Актуальність теми.** В останні роки індустрія мобільних ігор значно змінилася, зокрема завдяки зростанню популярності ігор жанру «hypercasual» ‒ ігри, які виділяються простими механіками, коротким часом ігрових сесій і легкою доступністю для широкої аудиторії. Популярність цього жанру зумовлена його здатністю швидко залучати гравців і підтримувати їхній інтерес завдяки простому, але захоплюючому ігровому процесу.

Оптимізація коду є одним із основних етапів під час розробки ігрових застосунків, оскільки вона забезпечує плавність ігрового процесу, зменшення затримок і збоїв, а також ефективне використання ресурсів пристроїв. Unity, як одна з найпопулярніших платформ для розробки ігор, надає широкий спектр інструментів для оптимізації, що дозволяє розробникам створювати ефективні ігрові рішення навіть для обмежених ресурсів мобільних пристроїв.

Актуальність дослідження зумовлена необхідністю створення конкурентоспроможних ігрових продуктів, які б задовольняли вимоги сучасних гравців та забезпечували високу продуктивність на різних типах пристроїв. Враховуючи зростаючий попит на ігри жанру «hypercasual», питання оптимізації коду стає особливо важливим для розробників, які прагнуть зменшити час завантаження гри, покращити її продуктивність і забезпечити безперебійну роботу.

**Об'єктом дослідження** ‒ процес оптимізації ігрового застосунку.

**Предмет дослідження** – методи оптимізації коду при створенні ігрового застосунку на платформі Unity.

**Метою** кваліфікаційної роботи є дослідження методів оптимізації коду при створенні ігрового застосунку в жанрі «hypercasual» за допомогою інструментів наданих ігровим рушієм Unity.

Відповідно до зазначених об'єкта, предмета і для досягнення поставленої мети визначені такі **завдання** роботи:

* проаналізувати технічну та спеціальну літературу відповідно до теми кваліфікаційної роботи;
* вивчити інструменти та методи оптимізації, надані Unity;
* розробити тестовий ігровий застосунок у жанрі «hypercasual»;
* провести тестування та аналіз продуктивності застосунку;
* зробити висновки щодо ефективності використаних методів оптимізації коду.

**Методи дослідження.** У роботі використано аналітичні методи оброблення інформації, методи аналізу та синтезу.

**Наукова новизна отриманих результатів** полягає у виявленні оптимальних методів оптимізації для забезпечення високої продуктивності та плавності ігрового процесу при створені ігор в жанрі «hypercasual».

**Практичне значення** роботи полягає в тому, що результати дослідження можуть бути використані розробниками ігор для створення продуктивних та якісних ігрових застосунків у жанрі «hypercasual», які задовольнятимуть вимоги сучасного ринку мобільних ігор.

**Апробація результатів дослідження.** Результати роботи оприлюднені у доповіді на VII Міжнародній науково-практичній конференції здобувачів вищої освіти і молодих учених «Інформаційні технології: теорія і практика» (Дніпро, Україна, 20-22 березня, 2024 р.) [25].

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ ЖАНРУ «HYPERCASUAL» ТА ЙОГО РИНКОВИЙ КОНТЕКСТ

1.1 Визначення жанру «Hypercasual»

Епоха ігрової індустрії постійно розвивається. Щороку розробники намагаються покращити ігровий досвід для гравців. Постійно експериментуючи, вони поєднують різні жанрові парадигми, створюють нові, та покращують вже наявні. При цьому, ігровий ринок настільки непередбачуваний, що дуже часто вже давно забуті концепції через деякий час стають популярними. Одним із таких жанрів є «hypercasual».

Жанр «hypercasual» походить від об’єднання двох слів «hyper» ‒ висока активність, динамічність, та «casual». ‒ простота та доступність. Гіперказуальна гра – це гра, яка містить в собі, легкі, інтуїтивно зрозумілі механіки для гравця, та швидким перенесенням гравця до геймплею [1].

Крім того, ігри такого жанру мають просте керування, лаконічне візуальне оформлення та постійно повторюваний сценарій – біг, стрибки, збір предметів тощо. На малюнку 1.1 показано приклади гіперказуальних ігор.



Рисунок 1.1 – Приклади Hyper casual ігор [2]

1.2 Історія розвитку жанру

Жанр «hypercasual» розпочав свій розвиток на початку 2016 року [3]. В основному ігри з’являлися на Android та IOS платформах. Це були ігри з простими для розуміння механіками та короткими ігровими сесіями. Варто зауважити, що через наявність простих механік та дизайну жанр має багато спільного з іграми старшого класу 70-х років [1].

З самого початку гіперказуальні ігри не були такими популярними як зараз. Оскільки в ті часи домінували вже популярні на той момент такі жанри як шутери, екшн-рпг, стратегії та інші.

У 2017 році гіперказуальні ігри почали лідирувати в Google Play та App Store, після виходу Flappy Bird (Рисунок 1.2), яка мала понад 50 мільйонів завантажень [5]. Можна побачити, що за статистикою 2019 року (Рисунок 1.3), 16 з 20 найбільш завантажених на той момент ігор ‒ гіперказуальні [4].



Рисунок 1.2 – Flappy Bird (2013 рік)

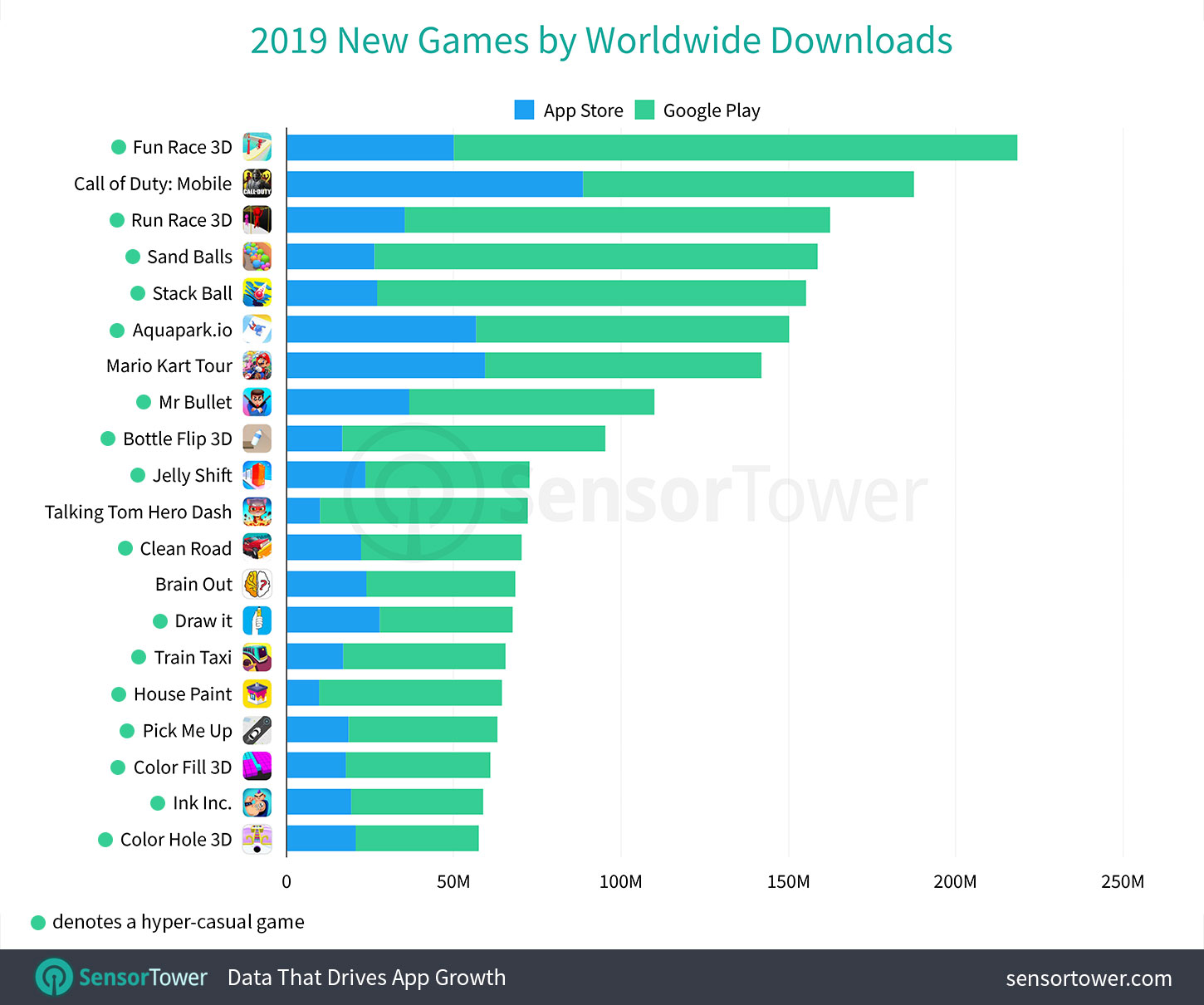


Рисунок 1.3 – Статистика найбільш завантажених ігор за 2019 рік [4]

До того ж в період пандемії 2019-го року кількість завантажень гіперказуальних ігор збільшилась на 103%. При цьому, слідуючи статистиці 2021-2022 років можна побачити (Рисунок 1.4), що гіперказуальні ігри займають 34% завантажень у світі порівняно з іншими жанрами. Слід зазначити, що згідно з дослідженнями, загальний розмір світового ринку гіперказуальних ігор у 2022 році становив 16665.88 мільйонів доларів США [6].

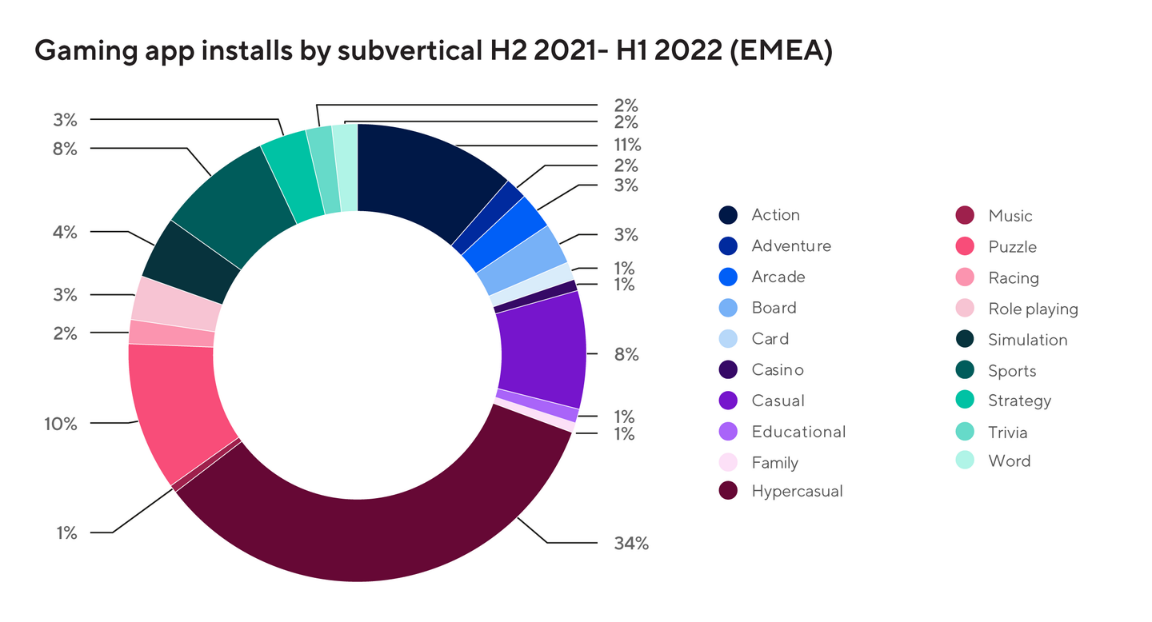


Рисунок 1.4 – Статистика завантаження ігор за жанрами у 2021-2022 роках [6]

Після скасування обмежень в період пандемії на початку 2022 року кількість ігрових сесій знизилась приблизно на 10% у порівнянні із серединою 2021 року. Наразі гіперказуальні ігри досі є актуальними та лідерами мобільного ринку.

1.3 Аналіз поточного ринку

У 2021 році глобальний ринок гіперказуальних ігор становив 15600 мільйонів доларів США. Прогнозується (Рисунок 1.5), що включно до 2032 року ринок буде сягати понад 35225 мільйонів доларів США з приростом в 7.96% за весь період часу [7].

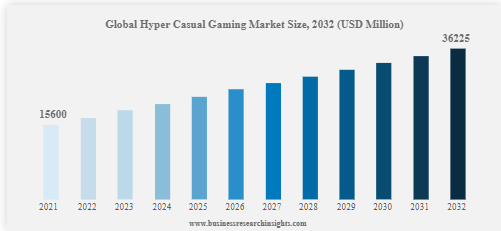


Рисунок 1.5 – Прогноз зростання ринку гіперказуальних ігор до 2032 року [7]

Для складання прогнозу враховувалися основні фактори для таких ігор, а саме мінімалістичний інтерфейс, чудовий та легкий ігровий досвід та популярність серед підлітків.

Поточний ринок ділиться на сегменти двох типів ОС – Android та IOS. Наразі домінує саме Android сегмент. Також, очікується, що протягом всього періоду дослідження у ролі споживачів кількість чоловіків буде більшою, ніж жінок (Рисунок 1.6) [7].

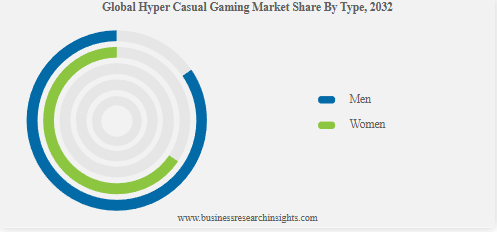


Рисунок 1.6 – Порівняння двох сегментів чоловіків та жінок [7]

При цьому, є декілька факторів, які стимулюють зростання ринку. Найголовнішою перевагою є мінімалістичний дизайн. Оскільки гравець може не втомлюватися навіть при довгих ігрових сесіях. До того ж такі ігри мають велику кількість рівнів різної складності, що надає користувачам цікавий досвід гри.

Інший, не менш важливий фактор ‒ простота розробки самої гри. Оскільки такі ігри мають одну або дві механіки та простий дизайн, звичайну гіперказуальну гру можна створити від декількох днів до тижнів. Постійно експериментуючи розробники можуть дуже швидко створити особливу та цікаву гру.

Слід також зазначити можливість заниження ринку, а саме через велику конкурентність між собою. Досить часто не вчасно випущена гра чи оновлення може призвести до занепаду навіть популярної гри [7].

Найбільшим акціонером ринку є Північна Америка. В її регіонах перебуває велика кількість як розробників, так і споживачів [7].

Основним доходом гіперказуальних ігор на ринку є реклама. Будь-яка гіперказуальна гра може мати декілька варіантів монетизації гри – Reward, банери та Interstitial. Найбільш ефективним вибором для монетизації є Rewarded відео (Рисунок 1.7) – відео, за перегляд якого надається винагорода. Як правило нагороди можуть включати в себе додаткові ігрові валюти, життя, бонуси, тощо. При чому, на відмінну від Interstitial, який з’являється без дозволу гравця, Rewarded відео гравець може переглядати за власним бажанням. Інтеграція рекламних відео приносять дохід розробникам та утримують гравців у грі [1].

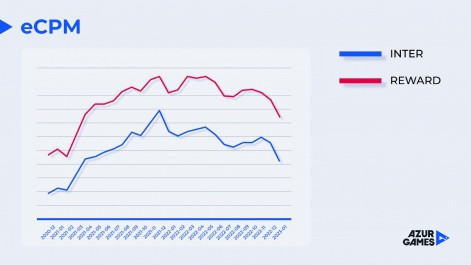


Рисунок 1.7 – Порівняння між Reward та Interstitial [8]

1.4 Особливості гейм-дизайну

Під час розробки гри важливо дотримуватися простих візуальних елементів. Головне завдання полягає в тому, щоб гравець максимально швидко зрозумів як грати у гру.

Слід додати, що кольорова гама повинна бути яскравою та одночасно не відволікати гравця під час гри. Це допомагає забезпечити приємний візуальний досвід, не перенавантажуючи гравця зайвою інформацією. Прикладом такої гри може бути «Helix Jump» (Рисунок 1.8) від паризького видавництва «Voodoo» [9].

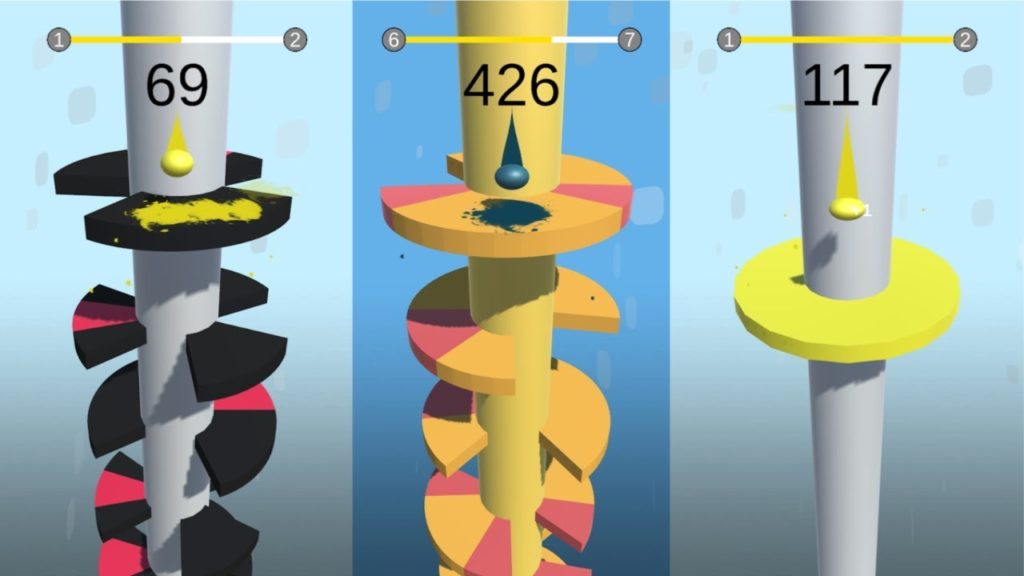


Рисунок 1.8 – Гра Helix Jump

Helix Jump має простий та одночасно захопливий геймплей ‒ гравець керує м’ячем, який постійно рухається вниз.

Мета гри ‒ дійти до кінця рівня, уникаючи зіткнень з платформами.

У грі використовуються базові геометричні фігури та «гра кольорів», які змінюються в залежності від рівня. Також, гра має простий користувацький інтерфейс – лічильник рахунку та прогресу рівня. Не зважаючи на досить просте рішення, гра має понад 100 мільйонів завантажень.

1.5 Специфіка аудиторії

Гіпе рказуальна гра — це тип гри, який підходить для будь-кого. Завдяки своїй простоті гра може забезпечити приємний ігровий досвід як дітям, так і людям похилого віку [10]. До того ж такі ігри ідеально підходять для людей, які рідко або майже ніколи не грають в них. В той час як, проєкти, наприклад, mid-core жанру, націлені на людей які полюбляють змагатися між собою.

За найбільшим проведенням часу у грі відзначаються люди з країн Європи та Тихоокеанського регіону Азії. За даними 2022 року в ігри грають більше жінки, аніж чоловіки. Слід зауважити, що в США 45% споживачів – чоловіки [10].

1.6 Вплив гіперказуальних ігор на ігрову індустрію

Починаючи з 2017 року, гіперкаузальні ігри почали набирати популярність серед інших жанрів завдяки своїй простоті та можливості швидкого занурення гравця безпосередньо до ігрового досвіду [5].

Слід зауважити, що близько 20% споживачів, що грають в ігри казуального або mid-core жанру є гравцями, що були залучені з реклами, яка відображається в гіперказуальних іграх [11]. Можна вважати, що завдяки зростанню гіперказуальних ігор зростає й сам мобільний ігровий ринок.

На ринку з’являються нові гібрид-казуальні ігри – ігри, що походять від гіперказуальних. Такі ігри поєднують в собі механіки як mid-core, так і гіперказуальних [11].

1.7 Висновки до розділу 1

Гіперказуальні ігри – це ігри, які мають одну або декілька механік, мінімалістичний дизайн, велику кількість рівнів з короткими сесіями проходження. Вони надають дивовижний ігровий досвід для споживачів на Android та IOS платформах.

Гіперкаузальні ігри є перспективним лідером на ринку, а їх актуальність прогнозується до 2032 року. За аналізом часу проводення у грі, Європа та Тихоокеанський регіон Азії визначаються як провідні регіони. Основний дохід ігрові студії та компанії отримують від інтегрованої реклами – Rewarded, Interstitial та рекламні банери. Серед перелічених Rewarded реклама являється найпопулярнішим джерелом доходу. Просуваючи рекламу, розробники не тільки отримують дохід, але й рекламують інші ігри з різними концепціями.

Гіперказуальні ігри мають велику конкуренцію. Розробникам доводиться постійно створювати нові механіки, поєднувати старі, придумувати щось унікальне та просте.

РОЗДІЛ 2

МЕТОДИ ОПТИМІЗАЦІЇ РОЗРОБКИ З ВИКОРИСТАННЯМ UNITY

2.1 Огляд інструментів Unity та оптимізації

Unity [23] – це одне з найпопулярніших інтегрованих середовищ розробки (IDE) для створення ігор і візуалізації 3D-моделей. Ігровий рушій надає розробникам широкий спектр інструментів та можливостей для створення ігрових проєктів на різних платформах, включаючи персональні комп’ютери, мобільні пристрої, консолі та віртуальну реальність.

Оптимізація є критично важливим аспектом під час розробки гри. За допомогою оптимізації можна підвищити кількість кадрів за секунду (FPS), зменшити витрати пам'яті та забезпечити ефективну роботу гри на різних пристроях.

Unity має такі вбудовані інструменти для оптимізації проєкту: Profiler, Frame Debugger та Physics Debugger.

2.1.1 Unity Profiler

Unity Profiler [13] – це інтегрований інструмент в Unity (див. рисунок 2.1), який дозволяє аналізувати продуктивність проєкту та виявляти проблеми, пов'язані з часом виконання, використанням пам'яті та відображенням графіки в грі, а також є одним з найважливіших інструментів для оптимізації.



Рисунок 2.1 ‒ Загальний вигляд Unity Profiler

Unity Profiler є потужним інструментом для аналізу продуктивності гри, який надає деталізовану інформацію про те, як гра використовує ресурси, включаючи час центрального процесора (CPU), пам'ять та використання графічного процесора (GPU). За допомогою цього інструменту розробники можуть відстежувати функції та скрипти, які найбільше впливають на процесор, ідентифікувати зміни в споживанні пам'яті та аналізувати роботу рендерингу. Важливою перевагою Unity Profiler є здатність автоматично виявляти гарячі точки (hotspots) та надавати рекомендації щодо оптимізації проєкту.

Profiler є незамінним інструментом для виявлення та розв’язування проблем з продуктивністю та ресурсами в ігровому проєкті. Використовуючи цей інструмент, розробники можуть зробити гру більш ефективною та оптимізованою для запуску на різних платформах, включаючи мобільні пристрої.

2.1.2 Frame Debugger

Frame Debugger [14] – це інструмент, який аналізує як кожен окремий кадр гри відтворюється і відображається на екрані. Використовуючи цей інструмент розробники можуть докладно розглядати та аналізувати рендеринг графіки, а також визначати проблеми, які пов'язані зі швидкодією відображення (див. Рис. 2.2).

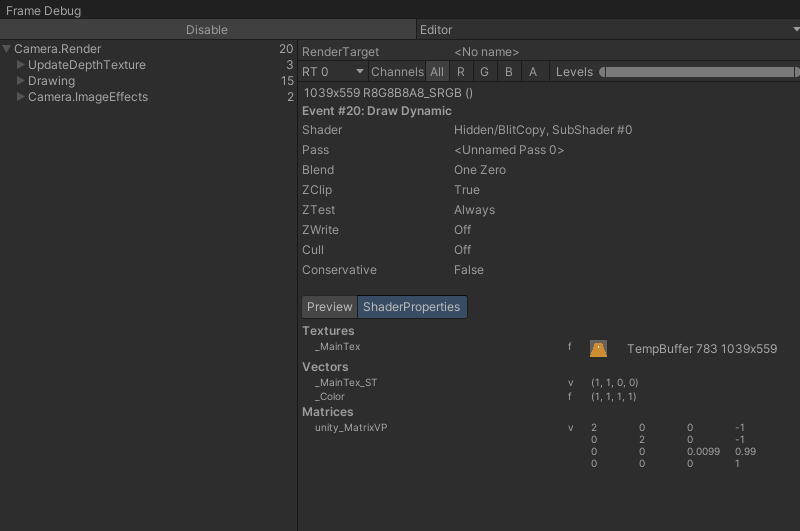


Рисунок 2.2 ‒ Загальний вид Frame Debugger

Важливим аспектом є відстежування кількості викликів draw call, які генеруються кожним об'єктом на кадрі. Зайві виклики draw call можуть призвести до зниження продуктивності гри. Frame Debugger надає інформацію про час, який витрачається на виконання GPU та CPU обчислень. Інструмент розв’язує проблеми зі швидкодією рендерингу, а саме зайве використання ресурсів, неоптимізованих матеріалів або надмірне використання апаратного обладнання.

Frame Debugger дозволяє розробникам вивчати кожен окремий кадр гри та визначати, як він відтворюється на екрані. Це особливо корисно для виявлення та розв’язання проблем з графікою та рендерингом, що можуть вплинути на продуктивність та відображення гри.

2.1.3 Physics Debugger

Physics Debugger [15] – це інструмент, який дозволяє налаштовувати та створювати візуальні представлення тривимірної фізики на ігровій сцені (див. Рис. 2.3).

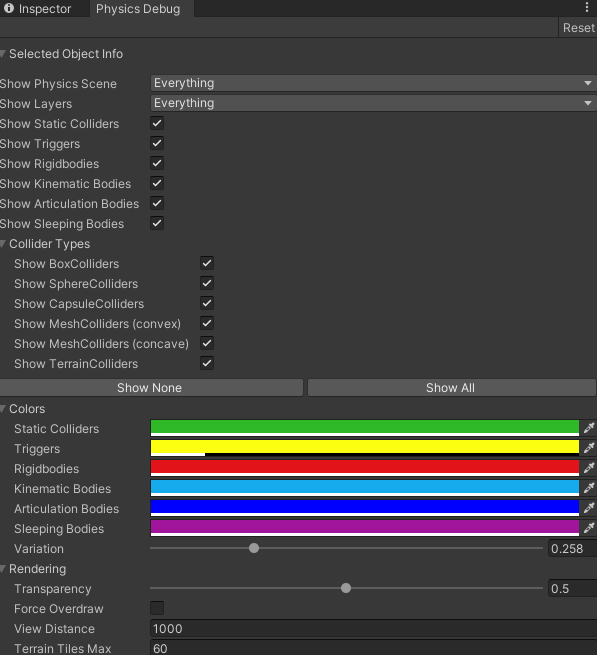


Рисунок 2.3 – Загальний вид Physics Debugger

За допомогою Physics Debugger розробники мають змогу відстежувати об’єкти з фізичним компонентом Rigidbody та Articulation Body, а також центр та візуалізацію тензора інерції, а також самостійно визначати які фізичні об’єкти показувати, а які ні.

2.2 Способи оптимізації проєкту в Unity

Unity відомий своєю функціональністю та можливістю створення дивовижних ігор, проте ця потужність може призвести до високого споживання системних ресурсів і погіршити користувацький досвід. Досить часто виникає потреба у вдосконаленні продуктивності та оптимізації ресурсів.

2.2.1 Батчинг виклику графічних об'єктів

Для того, щоб намалювати об’єкт на екрані ігровий рушій надсилає команду «draw call» графічному API, наприклад, OpenGL або Direct3D. Обробка кожної команди значно впливає на продуктивність центрального процесора. Для розв'язання цієї проблеми Unity використовує два методи, а саме статичний батчинг (англ. Static Batching) – комбінування нерухомих об’єктів у великі меші (англ. Mesh), та динамічний батчинг (англ. Dynamic Batching) – групування маленьких мешів в одне ціле [16].

Динамічний батчинг відбувається автоматично та не потребує жодних дій від розробника. Він застосовується тільки у випадках, коли меші містять у собі менш ніж 900 вершин в сумі.

Статичний батчинг дозволяє ігровому рушію знизити кількість викликів draw call незалежно від розміру геометрії об’єкта, але лише за умови, що об’єкт є нерухомим та використовує спільний матеріал. Цей метод є більш ефективним, оскільки він допомагає оптимізувати навантаження на центральний процесор, але потребує більше пам’яті, ніж попередній.

Нерухомі об’єкти на ігровій сцени можна позначити статичними, встановивши прапорець «Static» в компоненті «Inspector» (див. Рис. 2.4). Таким чином, Unity буде використовувати метод статичного батчингу саме для цих об’єктів.

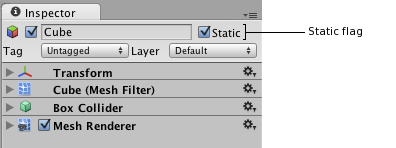


Рисунок 2.4 – Налаштування статичного батчингу в інспекторі об’єкта [15]

2.2.2 Користувацький інтерфейс Unity UI

Unity UI – це набір інструментів для створення користувацьких інтерфейсів. Система ґрунтується на об'єктах GameObject та використовує компоненти разом з вікном гри для розміщення, позиціювання та стилізації інтерфейсів. Важливою особливістю є можливість роботи з інтерфейсами безпосередньо в середовищі редагування Unity Editor. Як правило, всі елементи інтерфейсу розташовані всередині компоненту Canvas. На рисунку 2.5 наведений приклад ігрового інтерфейсу з даним компонентом.

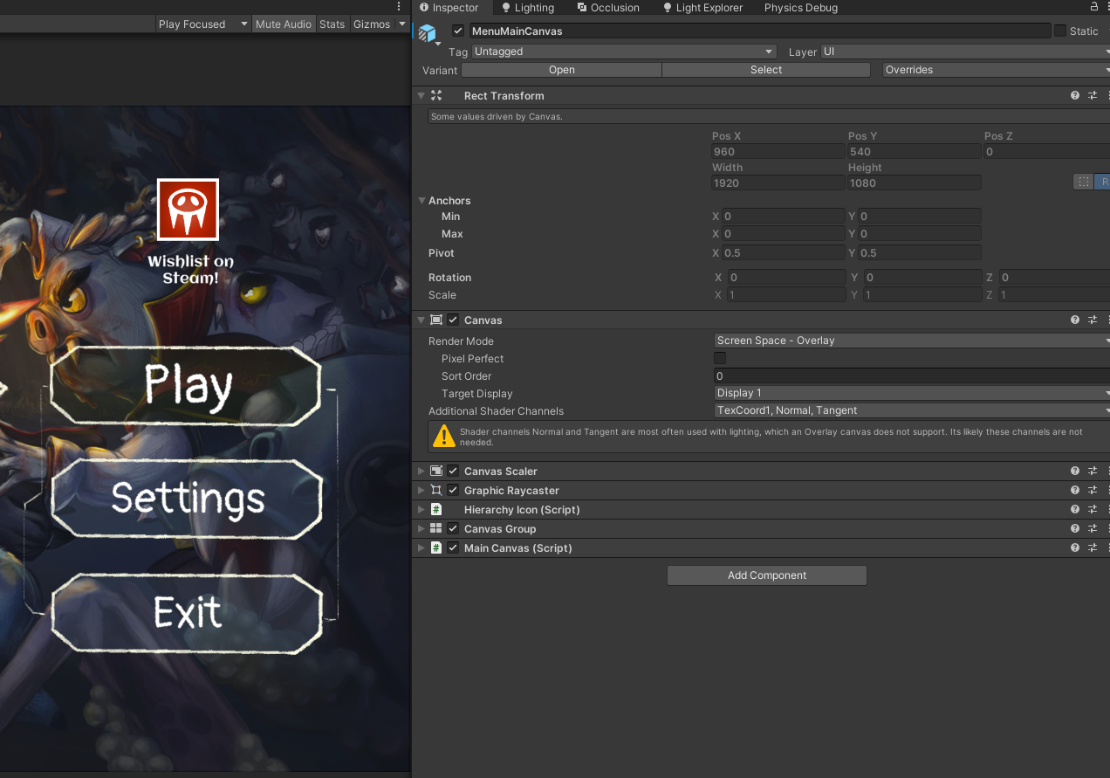


Рисунок 2.5 – Ігровий об’єкт, який містить компонент Canvas

Зазвичай малодосвідчені розробники використовують тільки один Canvas як контейнер для всіх елементів користувацького інтерфейсу у своїх проєктах. Проте результати даного дослідження свідчать, що цей підхід не є оптимальним і, в деяких випадках, може негативно вплинути на продуктивність всього застосунку.

Розглянемо цю ситуацію на прикладі конкретного механізму в грі, а саме «таймера», який постійно оновлюється протягом гри. Кожного разу, коли таймер оновлюється і змінюється вміст тексту, графічні ресурси, включаючи весь Canvas, якому належить таймер, піддаються перемальовуванню. Це означає, що всі інші елементи UI в грі також повторно перемальовуються, навіть якщо вони не підлягають змінам. У випадку мобільних проєктів це може призвести до надмірних ресурсних витрат. Процес перемальовування графічних елементів може бути особливо вимогливим до швидкодії і пам'яті, що впливає на продуктивність та витрати енергії пристроїв.

Щоб уникнути проблеми, яка пов'язана із надмірним перемальовуванням UI елементів, потрібно створювати окремі об’єкти, які містять компонент Canvas для різних груп UI елементів [12]. Цей підхід дозволяє оптимізувати процес перемальовування та підвищити продуктивність гри, особливо на мобільних платформах, а також дає можливість контролювати оновлення кожного компоненту окремо.

Таким чином, коли таймер оновлюється, лише той Canvas, якому він належить, піддається перемальовуванню, і це не впливає на інші елементи інтерфейсу. Це допомагає зменшити надмірне навантаження на графічну підсистему та підвищити продуктивність гри.

2.2.3 Occlusion Culling та Frustum Culling

Occlusion Culling [17] являє собою інтегровану функціональність у комп'ютерній 3D графіці, призначену для автоматизованого управління процесом рендерингу об'єктів у віртуальному просторі. Його основна функція полягає в призупиненні графічного відображення об'єктів, які не перебувають у зоровому полі камери, індукуючи неефективну графічну обробку та споживання ресурсів обчислювальної системи.

Occlusion Culling відтворюється разом з Frustum Culling [17], який обмежує процес рендерингу тих об’єктів, які не попадають у визначену область огляду камери. Occlusion Culling вимикає об'єкти, які знаходяться за іншими об'єктами. Цей процес ретельно враховує геометричні взаємозалежності між об'єктами та дозволяє зменшити «overdraw» – явище надмірного повторного рендерингу областей на екрані, що допомагає підвищити продуктивність і покращити якість візуалізації в 3D графіці.

2.3 Оптимізація коду в C# та Unity

Оптимізація коду – один із найвпливовіших способів покращити продуктивність програми. Арсенал Unity вміщає в собі багато корисних методів для швидкого розв’язання широкого кола задач в залежності від ситуації.

2.3.1 Використання Update методів

Unity має 3 методи подій [18], які викликаються за кожен кадр. З метою оптимізації роботи програми варто уникати використання цих функцій, а саме Update, LateUpdate та FixedUpdate [13]. Замість того, щоб включати логіку в функції, що викликаються кожен кадр, слід розглядати їх використання лише в тих випадках, коли дані дійсно потребують змін кожного кадру (наприклад, переміщення персонажів, стрільба з вогнепальної зброї та інше). Уникання зайвого використання Update методів дозволяє позбутися надмірних витрат ресурсів на виконання логіки, яка не потребує постійного оновлення.

Якщо все ж таки розглядати використання Update методів, варто розглянути можливість використання часового розбиття для кращої оптимізації коду. Наприклад, можна викликати код всередині функції кожен n кадр або час. Подібний підхід забезпечить ефективніше використання ресурсів та підвищить продуктивність програми.

Крім того, якщо задача стоїть у тестуванні певної логіки, то для кращої продуктивності можна використати директиви препроцесора як показано на рисунку 2.6

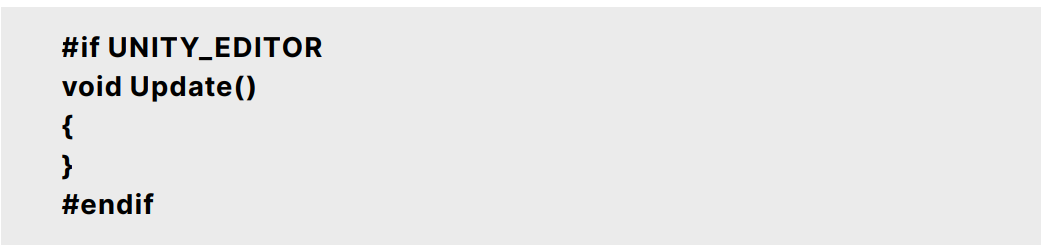


Рисунок 2.6 – Приклад оголошення предпроцесорної директиви [13]

2.3.2 Клас Component та його кешування

Клас Component має велику кількість корисних методів, які часто використовуються при розробці застосунку. Водночас використання методів цього класу є доволі важким процесом. У даному дослідженні хочемо приділити особливу увагу методу AddComponent. Під час виклику цього методу Unity перевіряє наявність дубльованих компонентів для новоствореного об’єкта. Це є важким процесом, який потребує значних ресурсів. Слід дотримуватися альтернативного варіанту, наприклад, замість того, щоб викликати AddComponent метод, можна створювати префаби (анг. prefab), які вже заздалегідь мають налаштовані компоненти [13].

До того ж варто не забувати про метод, який часто може використовуватися у розробці, а саме GetComponent. За допомогою цього методу можна отримати доступ до будь-якого компоненту об’єкта, якщо такий є в ньому. На практиці, постійне використання цього методу може стати причиною неочікуваного зниження продуктивності застосунку. Щоб забезпечити його ефективне використання, необхідно викликати його один раз, у таких методах подій як Start або Awake (див. рис. 2.7). Завдяки цьому можна повторно використовувати отриманий результат в будь-якому місці класу компонента.

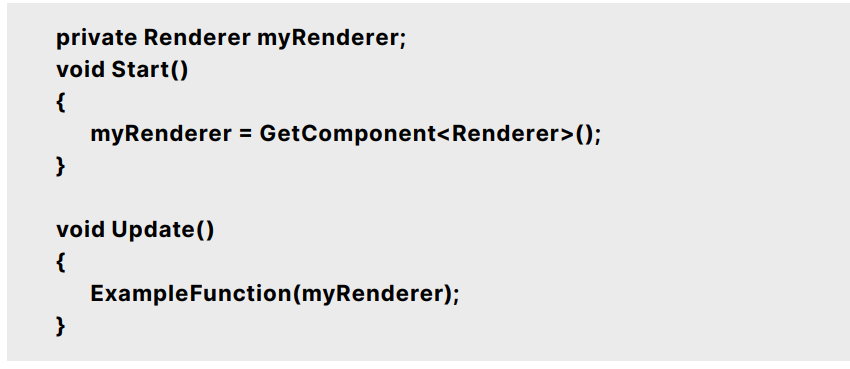


Рисунок 2.7 – Використання кешування в методі Start() [13]

2.3.3 Debug log інструкції

У процесі розробки проєкту розробники завжди використовують Debug Log інструкції [19]. Насамперед функція Log класу Debug – це функція, яка дозволяє вивести будь-яку вставлену всередині себе інформацію безпосередньо на консоль в редакторі Unity. Це дає змогу дуже швидко та зручно тестувати вхід будь-яких даних або помилок, щоб краще зрозуміти як працює та чи інша створена система з середини. Попри свою безмежну користь, Debug log інструкції також можуть сповільнювати продуктивність застосунку, особливо якщо вони використовуються в таких методах як Update, LateUpdate, або FixedUpdate [18].

Unity надає можливість створення власних умовних атрибутів з препроцесорною директивою. Ці директиви можна налаштовувати у налаштуваннях гравця «Player Settings». Прикладом оптимізації Debug log інструкції [19] може бути створення власного класу як показано на рис 2.8:

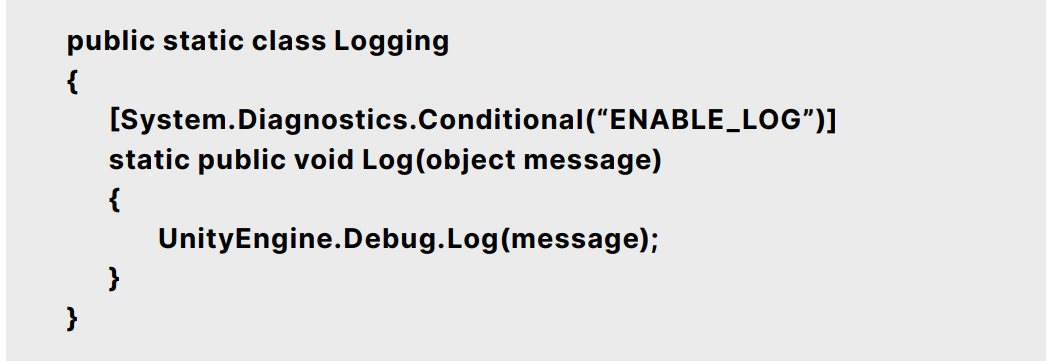


Рисунок 2.8 – Статичний класс Logging [13]

На рис 2.9 показано принцип додавання новоствореної препроцесорної директиви у налаштуваннях гравця «Player Settings».

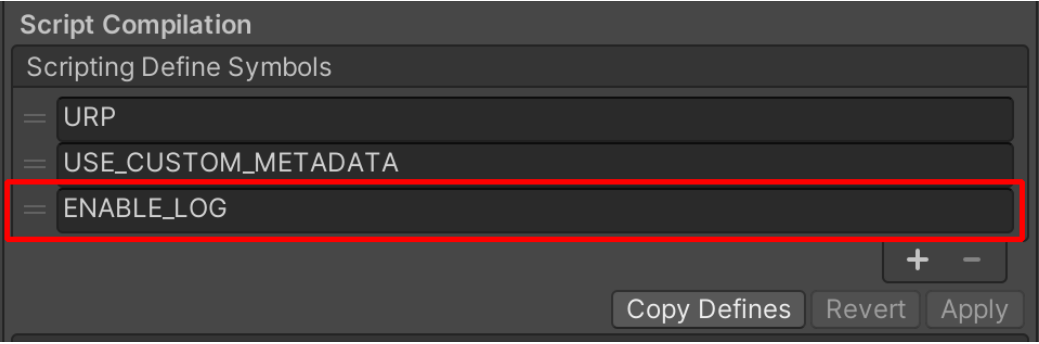


Рисунок 2.9 – Інтерфейс додавання предпроцесорної директиви [13]

Тепер, всі Debug log інструкції, що викликаються новоствореним статичним класом Logging можуть як вмикатися, так і вимикатися одночасно залежно від налаштувань гравця.

2.4 Оптимізаційні шаблони проєктування

Використання шаблонів проєктування під час розробки застосунку надає не лише гарний приріст продуктивності, але й покращує читабельність коду та зменшує його зв’язність.

2.4.1 Пул об’єктів

Object Pool [20] – це шаблон проєктування, який попередньо створює екземпляри всіх об’єктів перед початком гри. Ці об’єкти можуть використовуватися в будь-який момент. Завдяки цьому Object Pool усуває необхідність створювати нові або видаляти старі об’єкти під час гри. За деяких обставин даний шаблон може значно покращити продуктивність гри, наприклад, коли в проєкті постійно створюється та знищується один і той самий об’єкт у швидкій послідовності.

Методи Instantiate (створення об’єкта) та Destroy (знищення) під час свого виклику створюють «сміття», через що викликається «збірник сміття» для видалення зайвої пам’яті у програми. Оскільки, видалення та виділення пам’яті є повільним процесом у грі можуть з’являтися різкі «зависання». Таким чином, шаблон Object Pooling є найкращим вибором до розв'язання подібних проблем під час розробки [13].

Unity має вбудований клас ObjectPool<T0>, конструктор якого декларується наступним чином:

public ObjectPool<T0>(Func<T> createFunc, Action<T> actionOnGet, Action<T> actionOnRelease, Action<T> actionOnDestroy, bool collectionCheck, int defaultCapacity, int maxSize);

Нижче наведено таблицю 2.1 для опису всіх параметрів цього конструктора:

Таблиця 2.1 ‒ Значення параметрів

|  |  |  |
| --- | --- | --- |
| № | Назва параметра | Опис параметра |
| 1. | createFunc | Створює новий екземпляр об'єкта, якщо пул об’єктів порожній. |
| 2. | actionOnGet | Викликається, коли екземпляр об'єкта береться з пулу. |
| 3. | actionOnRelease | Викликається, коли екземпляр об'єкта повертається в пул. |
| 4. | actionOnDestroy | Викликається, коли елемент не може повернутися в пул. Наприклад, досягнено максимальний розмір пулу. |
| 5. | collectionCheck | Перевірка колекції викликається, коли екземпляр об'єкта повертається до пулу. |
|  | defaultCapacity | Розмір пулу за замовчуванням, з якою буде створено стек |
|  | maxSize | Максимальний розмір пулу |

Змоделюємо приклад використання даного шаблону проєктування для ситуації, де постійно потрібно створювати монети для гравця, щоб той їх підбирав у процесі гри.

На рисунку 2.10 наведено клас CustomObjectPool, який використовує вбудований в Unity клас ObjectPool<T0> та містить відповідні функції для своєї ініціалізації, а саме: CreateFuncPoolCoin, GetFuncPool та ReleaseFuncPool. Вони реалізують відповідний функціонал для створення об’єкту монети класу Coin, взяття його з пулу та повернення в пул. В свою чергу, клас Coin містить інтерфейс IObjectPool<Coin> та основні методи для роботи з пулом: SetPool() та ResetCoin(). Функція SetPool() необхідна для того, щоб передати екземпляр класу CustomObjectPool класу Coin при створені монети для того, щоб потім повернути монету в пул (рис 2.11), якщо її підбере гравець.



Рисунок 2.10 – Клас CustomObjectPool



Рисунок 2.11 – Клас Coin

Шаблон проєктування Object Pool завжди показує позитивні результати, якщо у проєкті постійно створюються та знищуються об’єкти. Однак, якщо процес створення або знищення відбувається за великий проміжок часу, то результати будуть майже непомітними.

2.4.2 Шаблон проєктування Singleton

Singleton [21] – це глобально доступний клас, який існує в ігровій сцені, але лише один раз. Загальна концепція шаблону полягає в тому, що будь-який інший клас може отримати доступ до класу Singleton та оперувати усіма публічними методами, змінними або властивостями цього класу. На рисунку 2.12 наведено клас GameController, що імплементує шаблон Singleton:

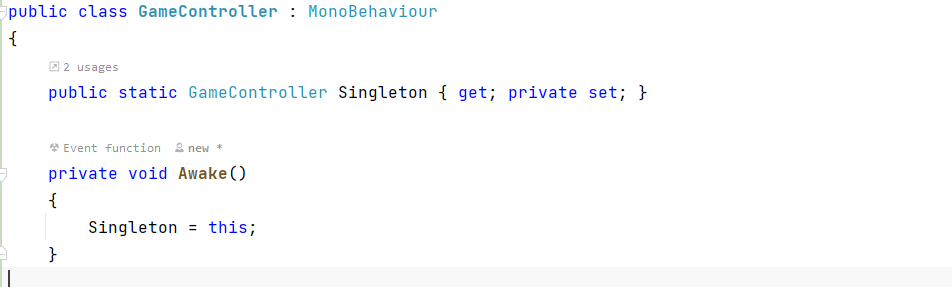


Рисунок 2.12 – Клас GameController, який імплементує шаблон Singleton

Часто, малодосвідчені розробники, які не знають про існування цього шаблону можуть використовувати вбудований метод, який схожий за принципом до Singleton – Object.FindObjectOfType<T>(), де T – це будь-який наявний компонент на сцені. Зазначимо, що навіть в документації Unity вказано, що даний метод є дуже повільним та запропоновано замість нього використовувати клас Singleton. Наведемо цитату самої статті Unity Documentation: «Please note that this function is very slow. It is not recommended to use this function every frame. In most cases you can use the singleton pattern instead.» [24].

Змоделюємо приклад використання самого шаблону. На рисунку 2.13 наведено клас GameController з публічним методом GameLose(). Нижче, на рисунку 2.14 наведено клас Player та його метод Death().

Зазвичай, в іграх коли гравець помирає, повинна з’явитися відповідне повідомлення про цю подію тощо. В даній ситуації клас Player повинен отримати доступ до класу GameController, щоб не використовувати вище зазначений «важкий» метод Object.FindObjectOfType< GameController >(), ми використовуємо шаблон проєктування Singleton.

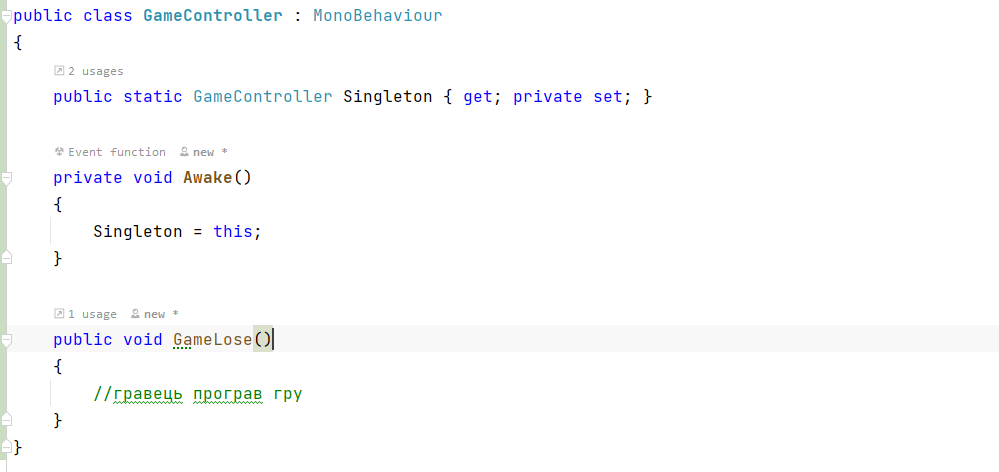


Рис 2.13 – Клас GameController

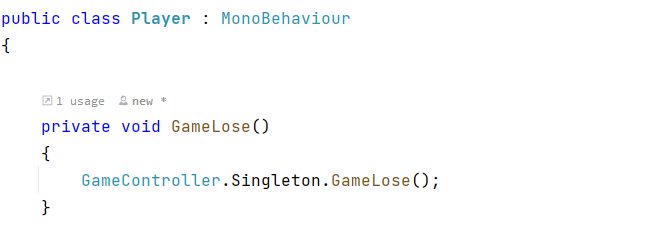


Рис 2.14 – Клас гравця Player

Шаблон проєктування Singleton хоч простий та ефективний, але має певні недоліки з якими можна часто стикатися під час розробки. По-перше, доступ до Singleton неможливо фактично отримати на початку гри. Оскільки клас, що реалізує цей шаблон на початку гри повинен ініціалізуватися. Зазвичай Singleton ініціалізують в Awake [18] методі, оскільки цей метод викликається ще до запуску скріптів. По-друге, використовуючи даний шаблон потрібно пам’ятати, що клас, який має Singleton обов’язково повинен бути на сцені та бути лише одним.

2.5 Методи оптимізації ресурсів проєкту

Оптимізація ресурсів проєкту вирішує проблему зменшення розміру кінцевої версії застосунку, що значно збільшить шанси на його попит.

2.5.1 Налаштування проєкту

Існують декілька налаштувань в Project Settings [13], які можуть суттєво впливати на продуктивність гри.

Перш за все, це налаштування на вкладці Player Settings. Player Setting має багато різних налаштувань, але найголовніше, для покращення оптимізації необхідно вимкнути лише декілька параметрів, а саме вимкнути Auto Graphics API (рис. 2.15), для платформ під яких не буде випускатись сама гра, перемикнути серверний сценарій з Mono на IL2CPP (рис. 2.16).

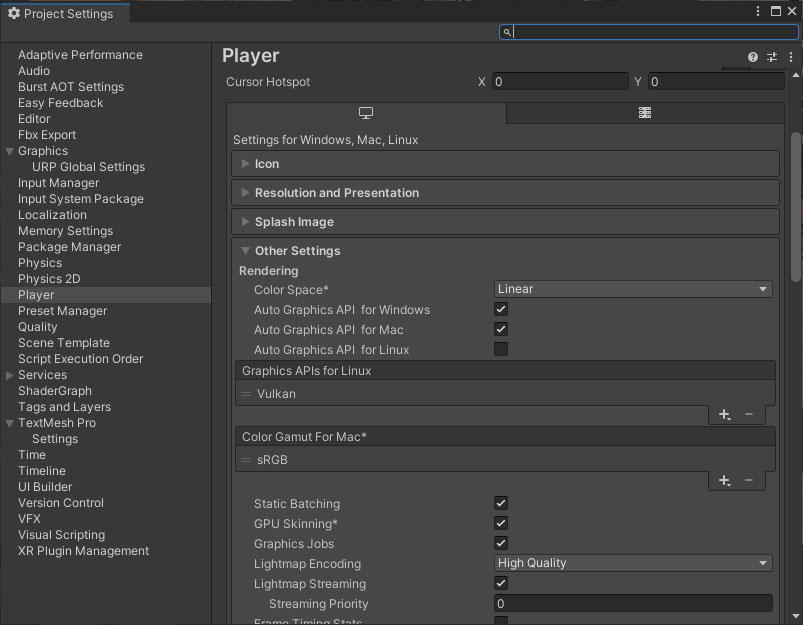


Рис 2.15 – Вигляд налаштування Auto Graphics API

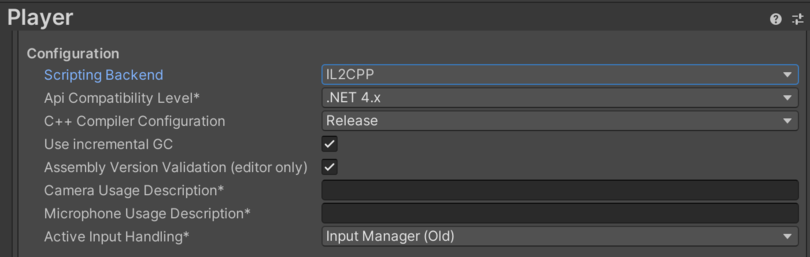


Рис 2.16 – Вигляд налаштування серверного сценарію

Додатково, не варто забувати про налаштування рівня графіки на вкладці Quality Settings (рис. 2.17).



Рис 2.17 – Налаштування рівня графіки

2.5.2 Стиснення текстур

Текстура [12] – це звичайне зображення, яке наноситься поверх мешу. Вона додає до неї колір, рельєф та інші властивості, які роблять об’єкт більш реалістичним та деталізованим. В Unity можна імпортувати текстури з найбільш популярних форматів зображення, наприклад .jpg, .png та інші.

Текстури застосовуються до об’єктів використовуючи матеріали, тобто компонент, який визначає як повинна відображатися поверхня на 3D-об’єкті. Матеріали використовують шейдери, тобто спеціалізовані графічні програми, які можуть відобразити текстури на поверхні 3D-об’єкту.

Оптимізація текстур у Unity є важливим аспектом, який безпосередньо впливає на продуктивність та якість відображення в ігрових проєктах. Після вдалого імпорту будь-яких ресурсів в Unity завжди потрібно їх правильно налаштовувати. Не налаштовані ресурси в проєкті, можуть мати неочікуваний вплив на розмір кінцевої версії гри. Оскільки, дуже часто бувають випадки коли імпортоване зображення може за замовчуванням мати найвищу якість налаштування та при цьому, в самій грі гравець може зовсім не звертати уваги на цей об’єкт.

Unity має декілька зручних параметрів для налаштування текстур. Кожен з них може значно вплинути на їх оптимізацію [13].

Max Size є найвпливовішим параметром. За замовчуванням, будь-яка імпортована текстура має значення 2048ⅹ2048 пікселів та може бути налаштована від 32ⅹ32 до 16384ⅹ16384 пікселів. Потрібно бути уважним підбираючи оптимальне значення для Max Size, щоб випадково не знизити якість відображення об’єктів на сцені, які використовують цю текстуру.

Наступний параметр це формат стиснення Format. Format може мати різні значення залежно від платформи під яку розробляється проєкт. Загалом, потрібно використовувати стиснені формати DXT або PVRTC, там де це можливо.

Останні дві опції ‒ Read/Write Enabled та Mip Maps. Перший параметр, створює копію текстури для обробки як центральним, так і графічним процесором, тим самим подвоюючи обсяг пам’яті самої текстури. У більшості випадків його можна вимкнути, оскільки для створення текстури можна скористатися функцією Texture2D.Apply. Другий параметр не потрібний, якщо ця текстура буде використовуватися при створені користувацького інтерфейсу, оскільки він, як правило, ніяк не змінює свою відстань від камери.

Додатково Unity має можливість створення «атласів текстур» – розміщення кількох текстур в одній. Використання атласів текстур може зменшити кількість викликів «draw calls» та прискорити процес рендерингу. Це покращує продуктивність особливо на мобільних пристроях.

Після успішного інтегрування текстур персонажа у проєкт, було проведено дослідження для порівняння кількості займаної пам’яті до та після їх налаштування. На рисунках 2.19 та 2.20 показані результати до та після налаштування текстури.

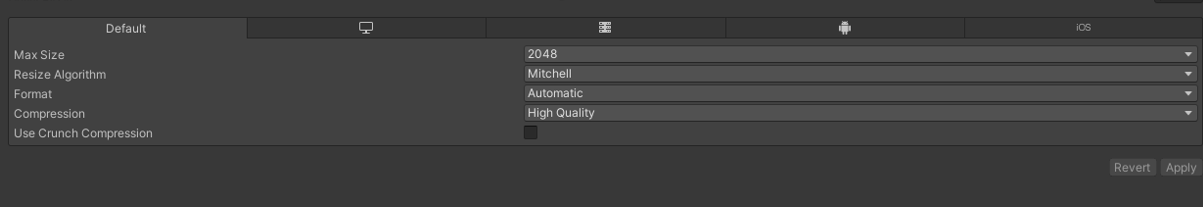
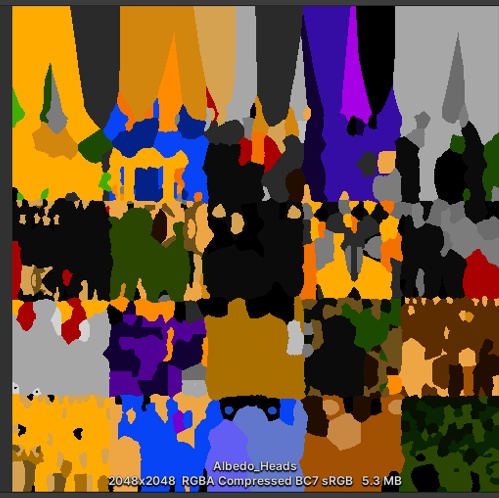


Рис 2.19 – Не налаштована, текстура гравця

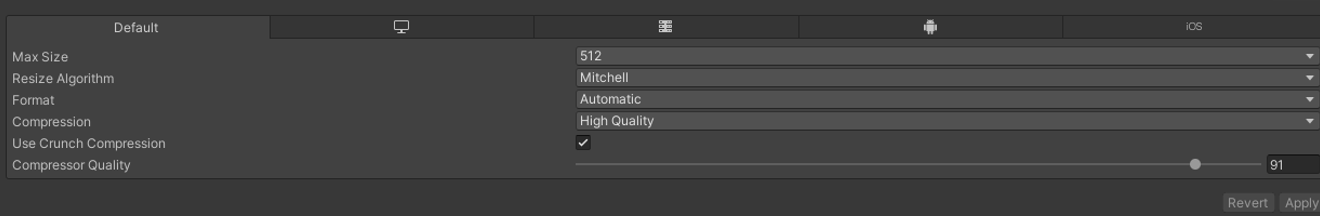


Рис 2.20 – Налаштована текстура гравця

Порівнюючи дані, можна побачити як оптимізована текстура гравця займає приблизно на 5MB менше, при чому, якість зображення у грі ніяк не змінилася.

2.5.3 Стиснення мешів.

Меш (від англ. Mesh) [12] – основний графічний примітив в Unity, який складає значну частини 3D-світів. Unity підтримує трикутні, чотирикутні, багатокутні меші. Зазвичай, Unity створює меш під час імпорту його моделі, але його також можна створювати безпосередньо в Unity.

Оптимізація мешів є важливою частиною розробки графічних ігор і додатків. Так само як і текстури, меші [13] можуть споживати значний обсяг ресурсів. Велика кількість полігонів у меші може призвести до збільшення обсягу пам'яті, необхідного для відтворення об'єкта в грі, що може негативно вплинути на продуктивність програми. Важливо використовувати оптимальні рівні деталізації для кожної моделі та виконувати компресію мешів, дотримуючись оптимальних параметрів.

Меші також можна стискати. На відміну від текстур, вони мають спеціальний параметр Mesh Comprassion. Результат стискання мешу впливає на його відображення в ігровій сцені, тому потрібно експериментувати з рівнями компресії та знайти оптимальний. Read/Write Enabled – виконує такі ж самі функції як і з текстурами.

У вікні налаштування мешу також присутні ще три параметри, на які варто звернути увагу, а саме Rigs, Normals та Tangents. Rigs відповідає за можливість мешу до скелетної або BlendShapes анімації. Normals та Tangents потрібні для правильного відображення та обробки освітлення на об’єкті. Якщо матеріал об’єкта не використовує текстури або освітлення параметри Normals та Tangents можна вимкнути з ціллю покращення оптимізації мешу.

Загалом, це всі перелічені параметри на які варто звертати увагу при імпорті мешу. Слід зазначити, що основний компонент, який відповідає за потенційний обсяг займаної пам’яті мешу – це його роздільна здатність. Варто взяти це до уваги, оскільки, навіть геометрія заднього фону у грі може займати півмільйона полігонів.

Після успішного інтегрування 3D-об’єктів персонажа у проєкт, було проведено дослідження для порівняння кількості займаної пам’яті до та після їх налаштування (див. Рис. 2.21 та 2.22).

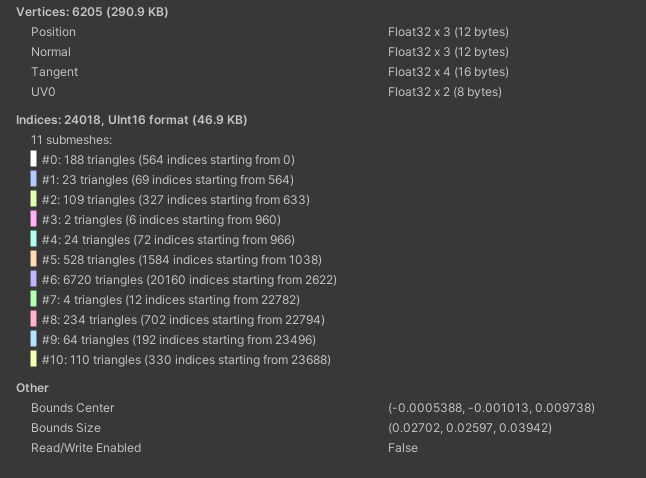


Рис 2.21 – Щойно імпортований меш та його вид на ігровій сцені



Рис 2.22– Щойно налаштований меш та його вид на ігровій сцені

Порівнюючи результати, можна побачити, що після налаштування мешу, його якість у грі ніяк не змінилася. Проте тепер щойно налаштований меш в проєкті займає приблизно на 210KB менше.

2.5.4 Addressable Asset System

Addressable Asset System [22] – це система керування ресурсів проєкту, яка використовує асинхронне завантаження даних з будь-якого місця розташування та взаємодіє з асетами[[1]](#footnote-1). За допомогою Addressables можна значно зменшити обсяг скомпільованого проєкту, та відповідно, скоротити час на його компіляцію.

Імпортувати Addressables можна за допомогою Unity Package Manager як показано на рисунку 2.22. Після успішного імпортування будь-який asset можна буде позначити як «Addressable» (рис. 2.23). Це необхідно для створення адреси, яка потім може бути викликана та повернена системою локально або віддалено.

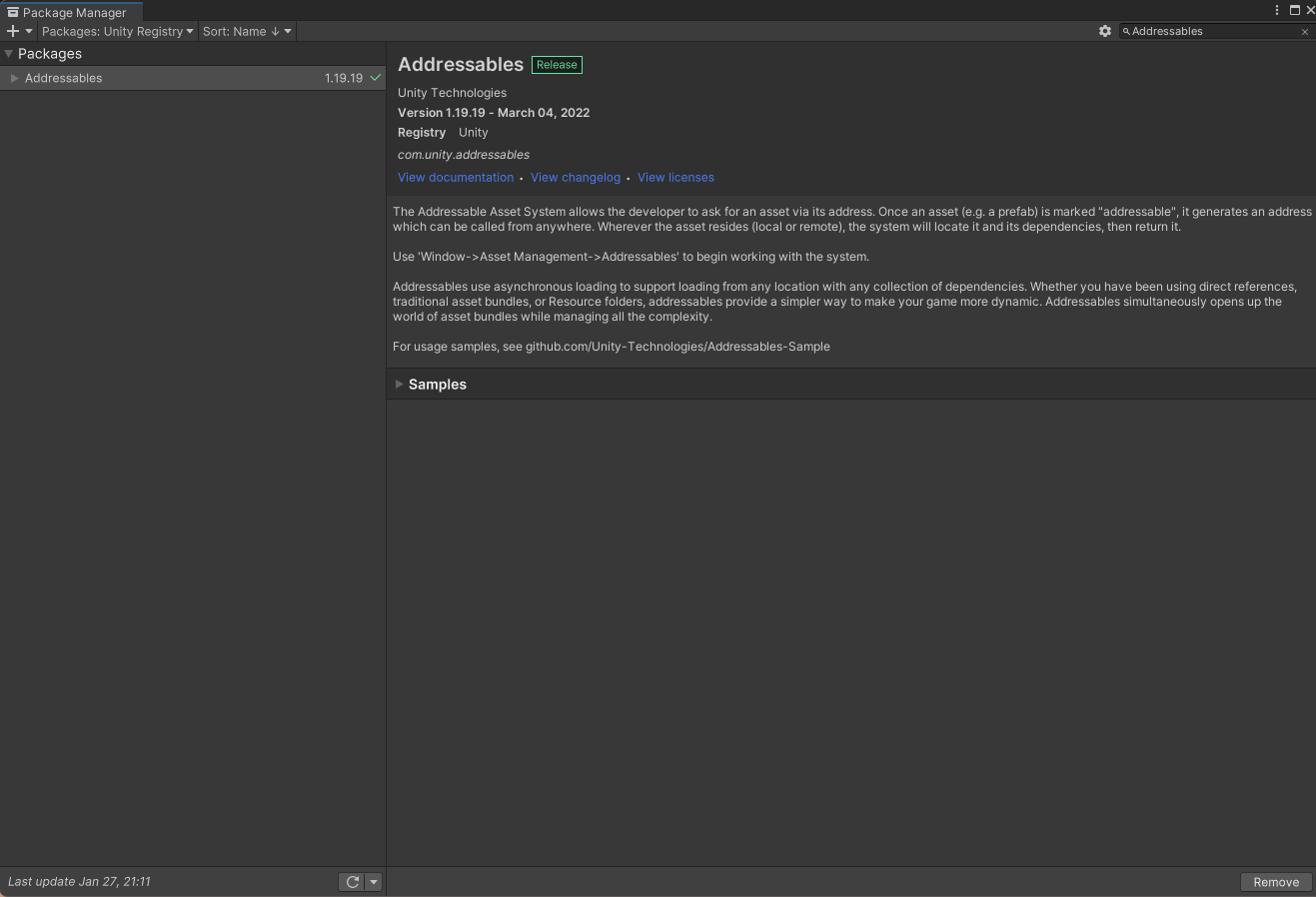
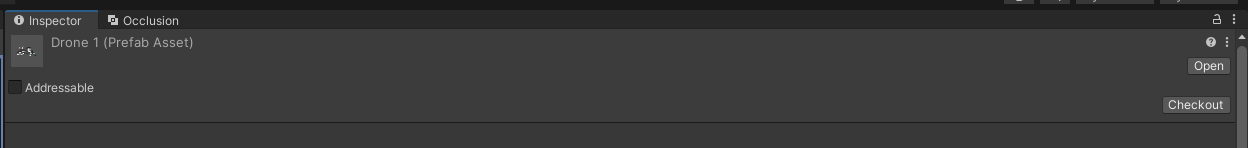


Рисунок 2.23 – Імпорт Addressables з Unity Package Manager

Рисунок 2.24 – можливість позначити префаб як «Addressable»

У процесі використання Addressables (див. Рис. 2.25) перший етап передбачає проходження запиту через систему каталогу. Ця система розкодовує адресу, перетворюючи її в місцезнаходження. Місцезнаходження містить в собі дані, які були згенеровані під час створення версії застосунку. Ці дані містять в собі інформацію про ресурси, їх залежності та способи зберігання (локально або віддалено).

Другий етап – потрапляння запиту до постачальника. Постачальників може бути декілька. Вони використовують декодовані дані, які були згенеровані на попередньому етапі, щоб знайти контент та повернути його до пристрою.

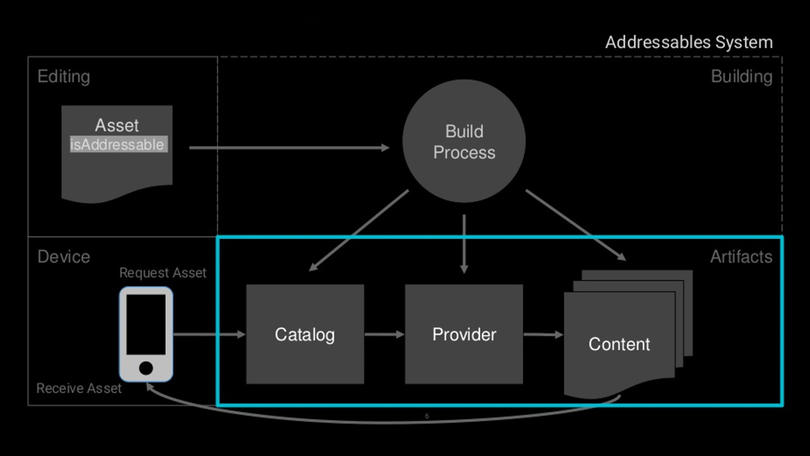


Рисунок 2.25 – Робочий процес Addressables [22]

2.6 Висновки до розділу 2

У даному розділі кваліфікаційної роботи було розглянуто різноманітні аспекти оптимізації в Unity, які стосуються як програмування, так і самого ігрового рушія та його ресурсів.

Було досліджено оптимізаційні шаблони проєктування такі як Pool Object та Singleton, їх переваги та недоліки, а також приклади до їхнього застосування. Крім того, було розглянуто основні принципи оптимізації коду в С# та Unity: використання функцій подій Update, FixedUpdate, LateUpdate, приклади використання кешування, огляд класу Component та ефективне використання Debug log інструкцій.

Велика увага була приділена оптимізації проєкту та його ресурсів. Крім того, було наведено порівняльну характеристику налаштування текстур та мешів для зменшення споживання пам’яті з використанням методів оптимізації. Розглянуто можливість використання системи «Addressable Asset System», що здатна завантажувати дані з будь-якого місця розташування

Оглянуто основні інструменти в Unity, які допомагають аналізувати проблеми оптимізації, а також наступні методи оптимізації проєкту в Unity:

* Методи використання користувацького інтерфейсу.
* Батчинг викликів малювання.
* Occlusion Culling та Frustum Culling.
* Використання ієрархії об’єктів.

РОЗДІЛ 3

СТВОРЕННЯ ГРИ З ВИКОРИСТАННЯМ ОПТИМІЗАЦІЙНИХ МЕТОДІВ В UNITY

3.1 Визначення ідеї гри

Головний герой веде боротьбу на відкритій локації проти великої кількості ворогів, які постійно надходять хвилями. Гравець має змогу удосконалювати або купувати нові здібності, витрачаючи монети, які випадають після смерті ворога. З самого початку та весь цикл гри гравець використовує вогнепальну зброю. На рисунку 3.1 показано концепт ігрової сцени зверху.

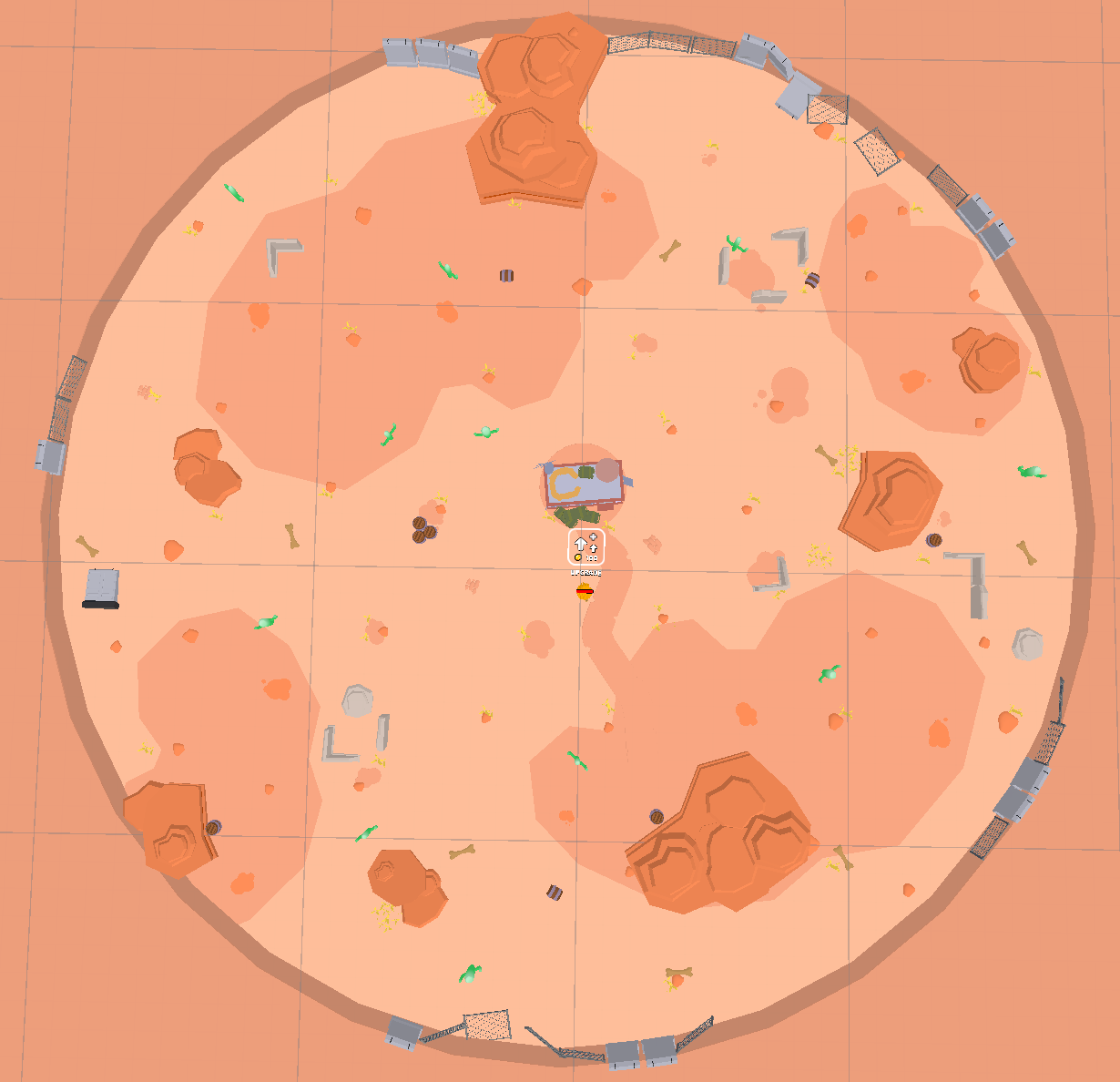


Рисунок 3.1 – Вигляд ігрового поля зверху

3.2 Аналіз проблеми та тестування гри

Перед початком проведення оптимізації необхідно проаналізувати поточний стан гри, щоб визначитися які методи краще використати для покращення її продуктивності.

3.2.1 Аналіз викликів «Draw calls»

На рисунку 3.1 наглядно видно, що поле гри має багато декоративних об’єктів: кактуси, скелі, паркани та інші. До того ж, при вбивстві ворога випадає монета, яка також є статичним об’єктом.

Після тестування гри на рисунку 3.2 та 3.3 можна побачити загальний аналіз на 626-му кадрі в Unity Profiler та детальну інформацію в «Rendering».

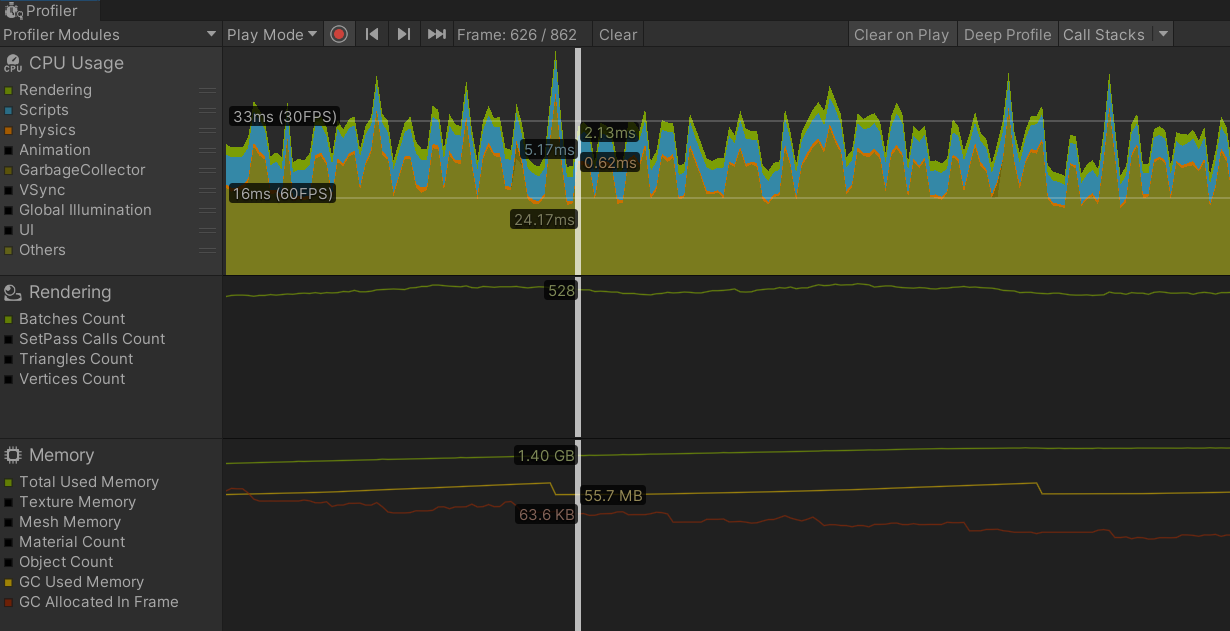


Рисунок 3.2 – Загальний аналіз гри в Unity Profiler

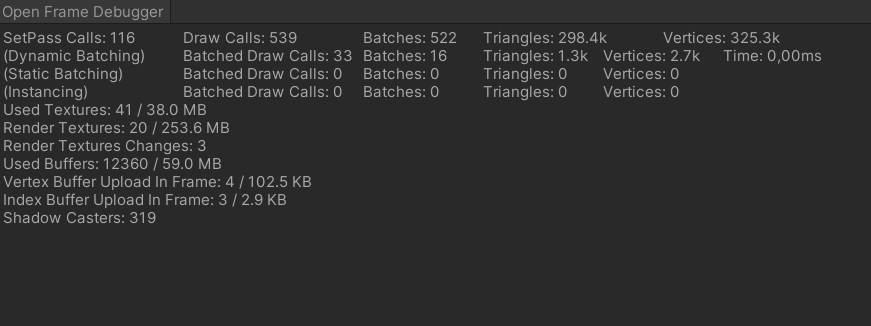


Рисунок 3.3 Детальна інформація секції «Rendering»

На рисунку 3.3 можна побачити, що кількість викликів «draw calls» сягає 539. В свою чергу, кількість батчів (Batches) – 522. Крім того, середній час затримки для центрального процесора 33-41 мілісекунда.

Для того щоб зменшити кількість батчів та «draw calls» необхідно на ігровій сцені в Unity для всіх статичних об’єктів (декоративні об’єкті, монети) в компонентів Inspector позначити Static (див рис 3.4 ).

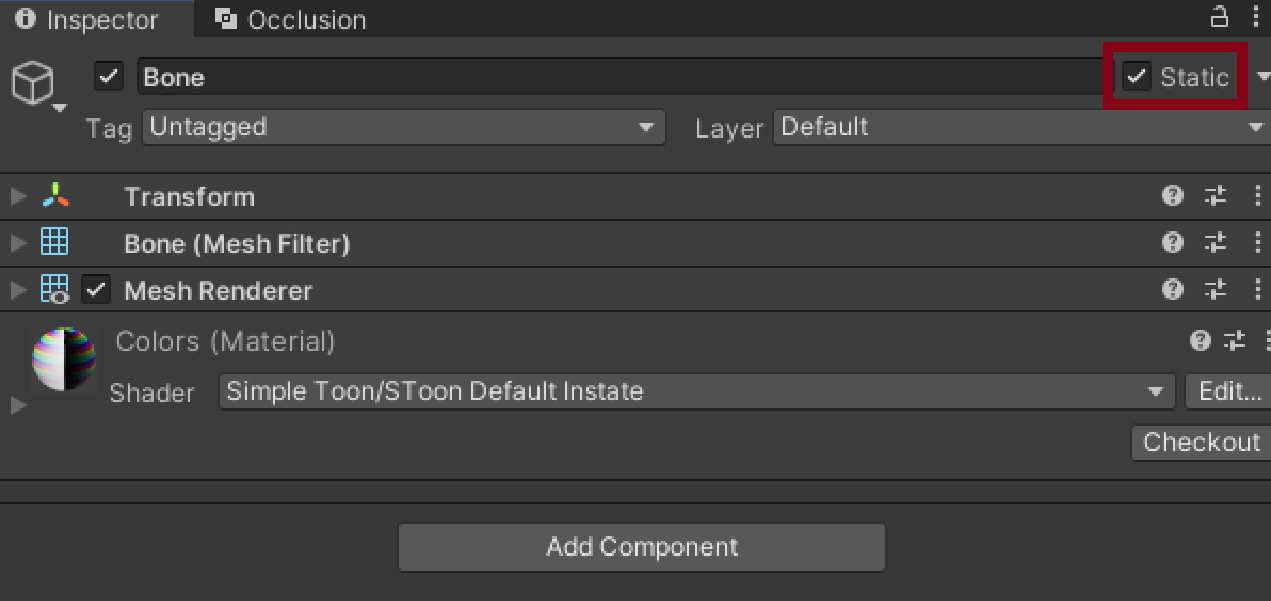


Рисунок 3.4 – Приклад позначення Static об’єкта

Після проведення повторного тестування гри на рисунку 3.5 можна побачити новий аналіз від Unity Profiler та на рисунку 3.6 детальний вигляд секції Rendering на 626-му кадрі.



Рисунок 3.5 – Повторний аналіз гри після проведення оптимізації для зменшення викликів «draw calls»

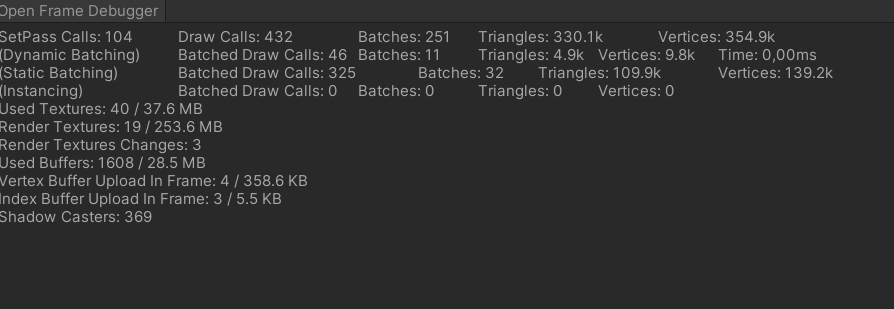


Рисунок 3.6 – Детальна інформація секції Rendering після проведення оптимізації

Порівнюючи результати з рисунків 3.3 та 3.6 можна побачити, що після проведення оптимізації кількість викликів «draw calls» зменшилася майже на 100. До того ж, кількість самих батчів зменшилося приблизно у два рази ‒ 251. Час затримки центрального процесора зменшився приблизно з 33-41 до 15-25 мілісекунд.

3.2.2 Аналіз використання Occlusion culling

Додатковим кроком для гарної оптимізації гри є налаштування Occlusion culling. За допомогою потужного інструмента Occlusion culling, який має досить просте налаштування та допомогає значно знизити споживання ресурсів телефону під час рендерингу гри, так як відображатися буде лише та частина ігрової сцени, яка входить в екран телефону під час ігрової сесії. Завдяки цьому значно зменшується кількість об'єктів, які необхідно рендерити, що, в свою чергу, покращує продуктивність гри та знижує навантаження на графічний процесор.

На рисунку 3.7 можна побачити результат тестування гри за даними Unity Profiler після налаштування Occlusion Culling та на рисунку 3.8 наведена детальна інформація секції «Rendering».

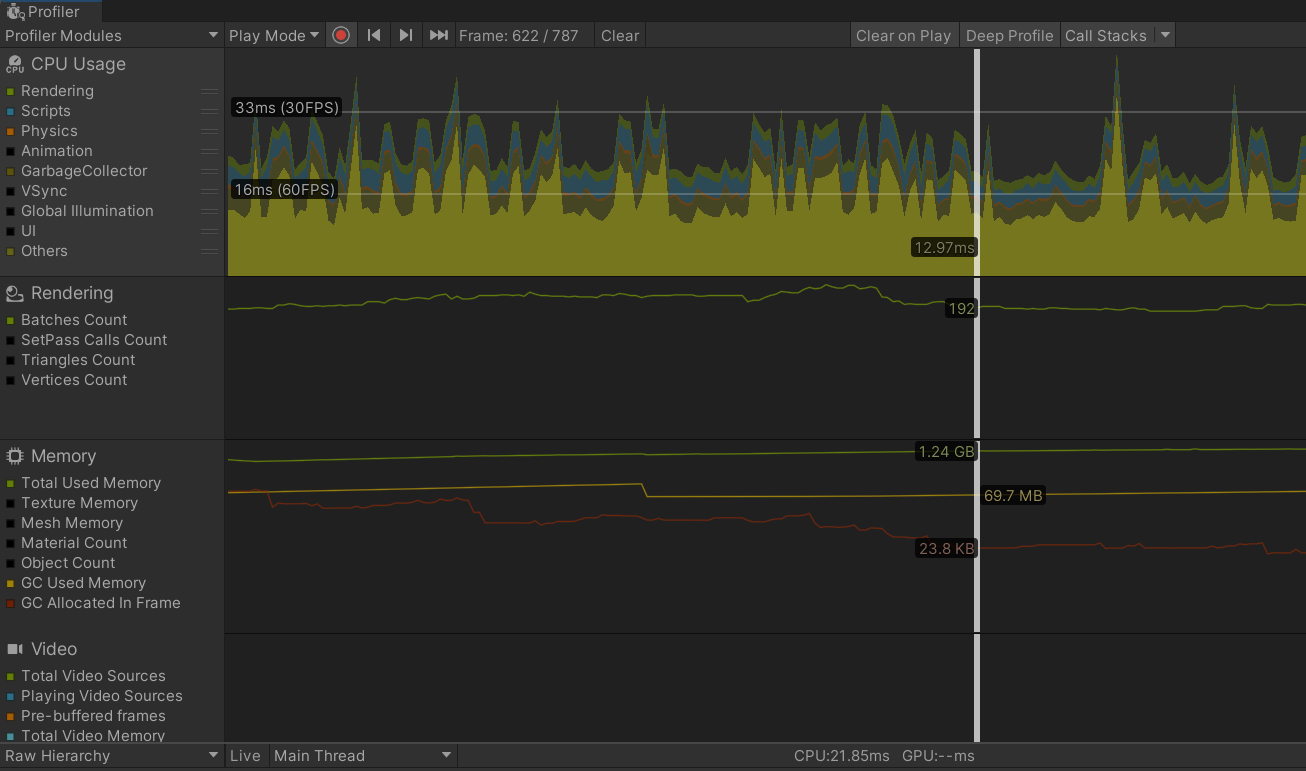


Рисунок 3.7 – Результат аналізу Unity Profiler після налаштування Occlusion culling.

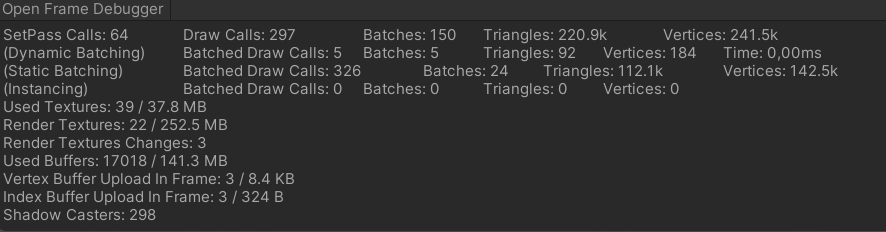


Рисунок 3.8 – Детальний аналіз секції Rendering після налаштування Occlusion culling

Проаналізувавши результати застосування Occlusion culling до проекту можна побачити, що після тестування кількість викликів «Draw calls» значно зменшилася з 432 (рис. 3.6) до 297. Кількість батчів теж знизилася з 250 (рис. 3.6) до 150. Варто звернути додаткову увагу на те, що споживання пам’яті збільшилося приблизно на 280 МБ.

Отже, необхідно мати на увазі, що використання Occlusion culling може зменшити навантаження на рендеринг в графічному процесорі, але також збільшити споживання оперативної пам’яті.

3.3 Оптимізація коду

Оптимізація коду є також важливою частиною для покращення продуктивності гри. В цьому підрозділі розглядатимуться наступні методи оптимізації коду ‒ використання Pool Object та виклики методу GetComponent<T>().

3.3.1 Реалізація шаблону Pool Object для оптимізації пам’яті

Для більш точного тестування було збільшено кількість ворогів у рівні, щоб наглядно побачити різницю використання шаблону Pool Object. В процесі дослідження було проведено звичайний аналіз та тестування після імплементації шаблону для монет та ворогів (див. рисунки 3.9 та 3.10). На рисунку 3.11 можна побачити результат тестування без використання шаблону. На рисунку 3.12 наведено результати після реалізації Object Pool.



Рисунок 3.9 ‒ Реаліція шаблону Pool Object



Рисунок 3.10 ‒ Використання шаблону для створення об’єктів

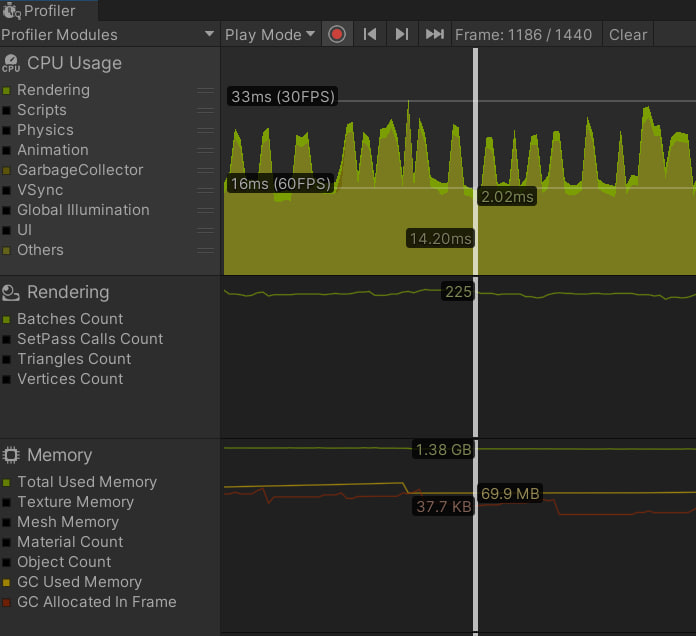


Рисунок 3.11 – Результат аналізу без використання шаблону Pool Object

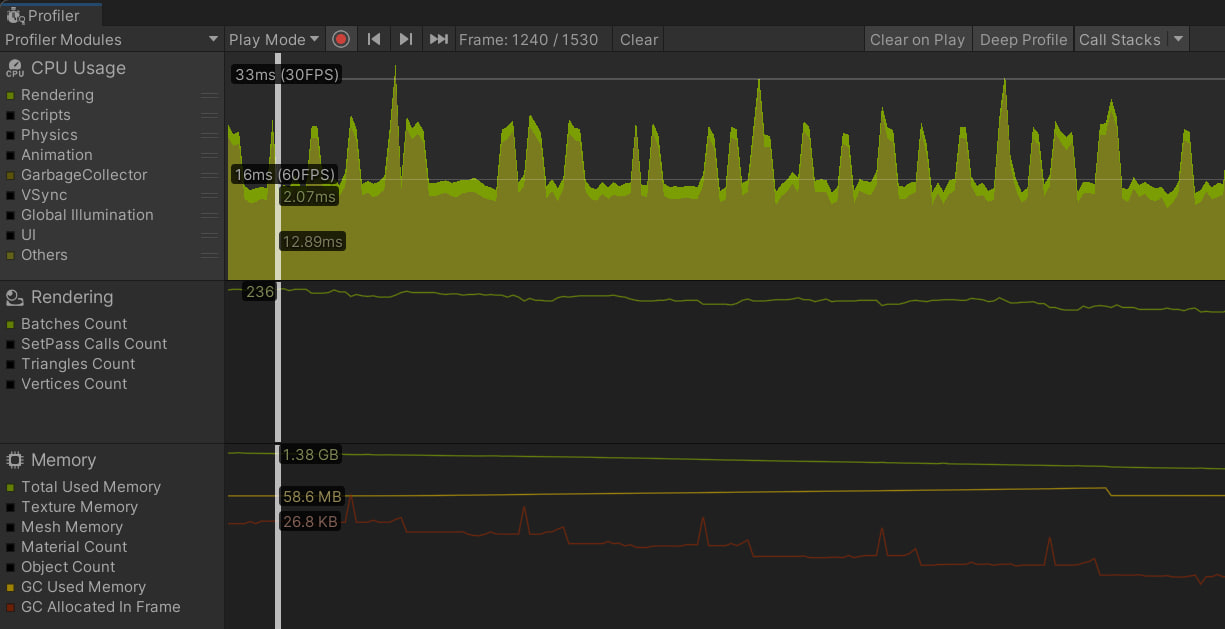


Рисунок 3.12 – Результат тестування після реалізації шаблону Pool Object.

На рисунку 3.11 середня затримка центрального процесора сягає приблизно 15-25 мс, а збірник сміття виділяє 37 кВ пам’яті кожного кадру та загалом використовує 68 МБ.

Після звичайного тестування було виявлено, що система створення ворогів та монет мала значний вплив на продуктивність гри. Для вирішення цієї проблеми було впроваджено шаблон Pool Object. Тепер після смерті вороги не видаляються з гри, а лише вимикаються як об’єкти та додаються до списку. Якщо необхідно створити ворога, то спочатку перевіряється список вже вимкнутих об’єктів. Якщо такі існують, то об’єкт ворога дістається зі списку та вмикається на сцені.

Таким чином, не потрібно виділяти пам’ять для створення або видалення об’єкта на ігровій сцені. Результати наступного тестування на рисунку 3.12 – затримка центрального процесора приблизно сягає 12-23 мс, що є меншим значенням ніж до використання Pool Object. Значні відмінності можна спостерігати в секції «Memory», де Garbage Collector виділяє 26 кВ пам’яті, при чому, можна зауважити, що графік споживання поступово знижується. Також Garbage Collector використовує 58 МБ, що є на 10 МБ менше ніж до реалізації шаблону.

3.3.2 Оптимізація викликів методу GetComponent<T>()

Перш за все необхідно звернути увагу на виклики таких функцій як «GetComponent», «GetComponentInChildren», та «GetComponentInParent». Ці функції можуть бути досить витратними з точки зору продуктивності, особливо якщо вони викликаються часто, наприклад, у методах Update або FixedUpdate.

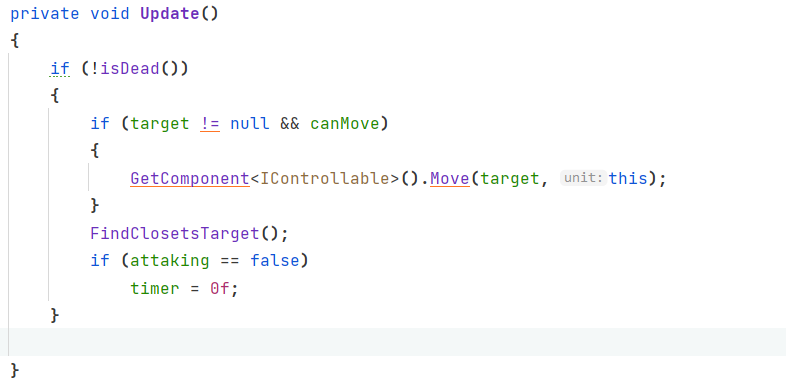
Слід уникати повторних викликів цих функцій та кешувати результати, коли це можливо. У випадку цього проєкту розглянемо та проведемо оптимізацію для двох систем, які викликають функцію GetComponent у методі Update.

Перша система ‒ клас гравця «Player», в якому використовуються системи знаходження ворога та стрільби. На рисунку 3.13 можна побачити як в методі Update звернення до класу Weapon викликається саме через виклик «GetComponent<Weapon>()».



Рисунок 3.13 – Метод Update класу «Player»

Друга система ‒ клас ворога «Unit», який відповідає за повний функціонал ворога. На рисунку 3.14 показано метод Update цього класу, в якому також постійно йде звернення до системи пересування саме через метод «GetComponent».

  
Рисунок 3.14 ‒ Метод Update класу ворога «Unit»

Для оптимізації логіки, що викликається у методі Update слід в Start або Awake зробити кешування цих систем у змінну та надалі використовувати їх у всьому класі. На рисунках 3.15 та 3.16 наведено процес кешування класів у змінну в методі Awake, а на рисунках 3.17 та 3.18 оновлений вигляд методу Update класів «Player» та «Unit».

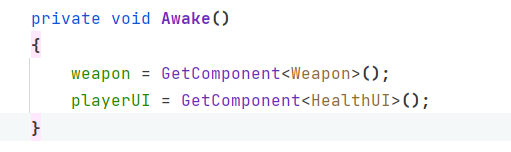


Рисунок 3.15 ‒ Кешування компонентів у змінні в класі гравця «Player»

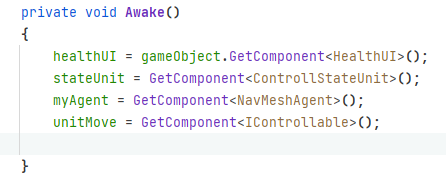


Рисунок 3.16 ‒ Кешування компонентів у змінні в класі ворога «Unit»

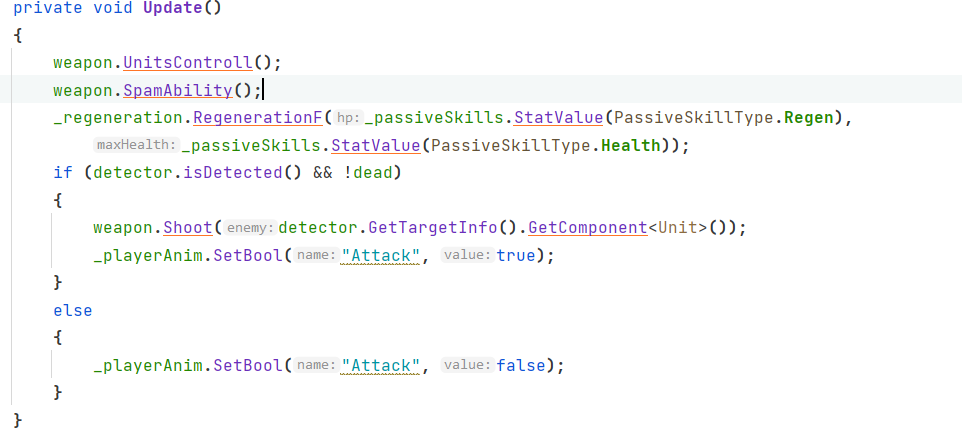


Рисунок 3.17 ‒ Метод Update класу «Player»



Рисунок 3.18 ‒ Метод Update класу «Unit»

Також, було проведено тестування перед і після застосування методу кешування. На рисунках 3.18 та 3.19 наведені результати аналізу тестування до та після оптимізації коду в класах «Player» та «Unit» відповідно.



Рисунок 3.18 ‒ Початковий результат аналізу до оптимізації коду.



Рисунок 3.19 ‒ Результат аналізу після оптимізації коду.

Аналізуючи результати, можна зробити висновок, що після оптимізації коду затримка центрального процесора тепер становить приблизно на 1 мс менше, ніж до неї. Хоча це невеликій приріст продуктивності, у грі було оптимізовано лише дві сутності. У великому проєкті кількість таких сутностей може бути значно більшою.

3.3 Висновки до розділу 3

У даному розділі описано розроблену гру та методи її оптимізації. Було використано 4 різні методи ‒ використання батчингу для статичних об’єктів, Occlusion Culling, шаблон проєктування Pool Object, та кешування даних. Відповідно, проведено тестування для кожного з порівняннями до та після їх реалізації.

За допомогою першого методу вдалося знизити кількість викликів «Draw calls» та затрати ресурсів центрального процесора.

Другий метод значно оптимізував процес рендерингу за допомогою технології Occlusion Culling, яка налаштовує камеру гри так, щоб та відмальовує лише ті об’єкти, які знаходяться в її полі зору, а не всю ігрову сцену.

Проведено оптимізацію коду шляхом використання методу кешування компонентів у змінні та впровадження шаблону проектування Pool Object. Метод кешування був застосований для сутностей гравця та ворога, що дозволило зберігати компоненти у змінних для подальшого використання. Шаблон Pool Object був застосований для ворогів та монет, для яких операції створення на видалення замінились на вимкнення та увімкнення. Обидва методи були протестовані з порівнянням результатів перед та після їх застосування.

ВИСНОВКИ

У даній кваліфікаційній роботі було проведено аналіз ринку, досліджено історію та цільову аудиторію гіперказуальних ігор, а також методи для їх оптимізації.

В першому розділі було досліджено теоретичні основи жанру «hypercasual» та його ринкового контексту. Було виявлено, що гіперказуальні ігри мають прості механіки, мінімалістичний дизайн та короткі ігрові сесії. Це робить їх надзвичайно популярними серед гравців на Android та iOS платформах. Водночас, велика конкуренція вимагає від розробників постійного створення нових і унікальних механік.

У другому розділі було розглянуто різноманітні аспекти оптимізації в Unity. Дослідження включало аналіз оптимізаційних шаблонів проектування, таких як Pool Object та Singleton, а також принципів оптимізації коду в C# та Unity. Було детально розглянуто методи оптимізації проєкту та його ресурсів, включаючи налаштування текстур та мешів для зменшення споживання пам’яті, використання системи Addressable Asset System, та інструментів Unity для тестування та аналізу оптимізаційних проблем.

В третьому розділі було створено гру та представлено практичне застосування оптимізаційних методів, що дозволило значно покращити продуктивність проєкту. Було показано, що застосування технології Occlusion Culling знижує навантаження на процес рендерингу, тоді як кешування компонентів і використання шаблону Pool Object підвищують ефективність управління пам’ятю.

У результаті отриманих даних з кваліфікаційної роботи, можна заявити, що ігри будуть постійно та швидко розвиватися на рівні інформаційних технологій, а процес оптимізації ставати все більш актуальнішим під час розробки гри. З удосконаленням ігрових рушіїв з’являтиметься все більше ефективних методів для покращення продуктивності ігор. Розробники матимуть більші можливість для створення оптимізованої гри високої якості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Requel Korman Hyper-casual games: What are they & how do you monetize them? URL: <https://www.is.com/community/blog/what-are-hyper-casual-games-and-how-do-you-monetize-them/> (дата звернення 01.12.23).

2. Казуальні ігри: що це, якими вони бувають і як розвивається жанр. URL : <https://vokigames.com/ua/kazualni-igri-shho-cze-yakimi-voni-buvayut-i-yak-rozvivayetsya-zhanr/> (дата звернення 02.12.23).

3. Igal Frid How hyper-casual apps are changing the way games monetize. URL : <https://www.appsflyer.com/blog/mobile-marketing/hyper-casual-gaming-ad-monetization/> (дата звернення 03.12.23).

4. Katie Williams Hyper-Casual Grew Share of Top New Game Downloads to 78% in 2019. URL: <https://sensortower.com/blog/hyper-casual-games-downloads-2019> (дата звернення 07.12.23).

5. The History of Hyper-Casual Gaming. URL: https://www.linkedin.com/pulse/history-hyper-casual-gaming-gamerpro/

(дата звернення 12.12.23).

6. Mobile gaming levels up in EMEA: Installs, sessions, and user behavior. URL: <https://www.adjust.com/blog/mobile-app-gaming-emea-2022/> (дата звернення 02.01.24).

7. Hyper Casual Gaming Market Size, Share, Growth and Idustry Analysus by Type (Android, And, IOS) By Application (Men and Women), and Regional Forecast To 2032. URL: https://www.businessresearchinsights.com/market-reports/hyper-casual-gaming-market-103060 (дата звернення 06.01.24).

8. Trends of hyper-casual games at the beginning of 2023 — analysis from Azur Games. URL: <https://app2top.com/analytics/trends-of-hyper-casual-games-at-the-beginning-of-2023-analysis-from-azur-games-226922.html> (дата звернення 14.01.24)

9. How to Make a Hyper-Casual Game: Design Tips & Costs. URL: <https://themindstudios.com/post/how-to-create-hyper-casual-game/> (дата звернення 17.01.24).

10. Hyper-Casual Games Report: Monetization & Advertising. URL: <https://www.blog.udonis.co/mobile-marketing/mobile-games/hyper-casual-games-report> (дата звернення 21.01.24)

11. The development of the hyper-casual game industry URL: <https://hc.games/the-development-of-the-hyper-casual-game-industry/> (дата звернення 24.01.24)

12. Joe Hocking Unity in Action, Third Edition: Multiplatform game development in C#: 3rd ed. Edition: 2015. 361 c.

13. Unity team: unity-e-book-optimize-your-mobile-game-performance. 2023. 52 с.

14. Unity Manual, Frame Debugger window reference. URL: <https://docs.unity.cn/Manual/frame-debugger-window.html> (дата звернення 26.01.24)

15. Unity Technologies Using the Physics Debugger - 2019.3. URL : [https://learn.unity.com/tutorial/using-the-physics-debugger-2019-3#](https://learn.unity.com/tutorial/using-the-physics-debugger-2019-3) (дата звернення 04.02.24)

16. Unity Manual, Draw call batching. URL: <https://docs.unity.cn/550/Documentation/Manual/DrawCallBatching.html> (дата звернення 08.02.24)

17. Understanding Frustum and Occlusion Culling in Unity3D. URL: <https://jaredamlin.medium.com/understanding-frustum-and-occlusion-culling-in-unity3d-9b15cc2b078a> (дата звернення 11.02.24)

18. Unity Manual, Order of execution for event functions. URL: <https://docs.unity3d.com/Manual/ExecutionOrder.html> (дата звернення 15.02.24)

19. How to use Debug Log in Unity (without affecting performance). URL: <https://gamedevbeginner.com/how-to-use-debug-log-in-unity-without-affecting-performance/> (дата звернення 17.02.24)

20. Robert Nystrom, Game Programming Patterns: 2011. 389 с.

21. Singletons in Unity (done right). URL: <https://gamedevbeginner.com/singletons-in-unity-the-right-way/> (дата звернення 18.02.24)

22. Simplify your content management with Addressables. URL: <https://unity.com/ru/how-to/simplify-your-content-management-addressables> (дата звернення 20.02.24)

23. Unity. URL: <https://unity.com/> (дата звернення 14.03.24)

24. Scripting API: Objec.FindObjectOfType. URL: <https://docs.unity3d.com/ScriptReference/Object.FindObjectOfType.html> (дата звернення 29.03.24)

25. Довда Н.О., Лисенко І.М. Оптимізація ресурсів проєкту в Unity // Матеріали Ⅶ Міжнародної науково-практичної конференції здобувачів вищої освіти і молодих учених «Інформаційні технології: теорія і практика». 2024. С. 261‒264.

1. Асет (від англ. Asset) ‒ контент (префаби, текстури, матеріали, тощо), який використовується для створення ігрового застосунку в Unity. [↑](#footnote-ref-1)