Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ІГРОВИХ РУШІЇВ ACTION ІГОР МОБІЛЬНИХ ДОДАТКІВ»

Виконав: студент 2-го курсу, групи 2ПІ-22м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
СамарасінгхеД.С.В.
(прізвище та ініціали)
Керівник: к.т.н., доцент кафедри ПЗ, Рейда О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«12» грудня 2023 р.
Опонент: к.т.н., доц. каф. ПЗ Войцеховська О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«12» грудня 2023 р.

Допущено до захисту
Завідувач кафедрою ПЗ
д.т.н., проф., Романюк О. Н.
«12» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти другий (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«19» вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Самарасінгхе Дмитро Сампат Вишневські

1. Тема роботи – «Дослідження методів оптимізації ігрових рушіїв "Action" ігор мобільних додатків».

Керівник роботи: Рейда Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від « 18 » вересня 2023 р. № 247.

2. Строк подання студентом роботи
____9 грудня 2023 р._____

3. Вихідні дані до роботи: Типобезпека на етапі компіляції, робота пулу з будь-якими класами, полегшення використання, авто-виділення нових об'єктів, потокобезпека, підтримка безлічі екземплярів пулу.

4. Зміст текстової частини: Розробка робочого класу Pool, для виконання поставлених умов, і перевірка коректного виконання робочого процесу, відслідковування витрат продуктивності.

5. Перелік ілюстративного матеріалу: використання створення простих 3D фігур в програмі Unity3D для перевірки корректного виконання поставленої задачі.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О.М., к.т.н., доцент кафедри ПЗ	19.09.2023	19.12.2023
5	Кавецький В. В., доц. каф. ЕПВМ, к.т.н	21.11.2023	25.11.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз продуктивності багатопотоковості додатку в якому часто створюються об'єкти	20.09.23 – 02.10.23	вик
2	Розробка архітектури та алгоритмів програмного продукту	03.10.23 – 23.10.23	вик
3	Аналіз і вибір мови програмування та середовища розробки	16.10.23 – 23.10.23	вик
4	Розробка програмного продукту	24.10.23 – 13.11.23	вик
5	Тестування програми	14.11.23 – 21.11.23	вик
6	Розробка економічної частини	17.11.23 – 25.11.23	вик
7	Оформлення матеріалів до захисту МКР	22.11.23 – 01.12.23	вик

Студент Самарасінгхе Д.С.В.

(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи Рейда О. М.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Самарасінгхе Д. С. В. Дослідження методів оптимізації ігрових рушіїв action ігор мобільних додатків. Магістерська кваліфікаційно робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – 121 Інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 142 с.

На укр. мові. Бібліогр.: 30 назв; рис.: 41; табл. 16.

У магістерській кваліфікаційній роботі модифіковано методи роботи з великою кількості об'єктів шляхом використання пулу для створених об'єктів. Даний підхід вирішує проблему фрагментації пам'яті, яка ускладнює пошук вільних суміжних областей пам'яті а також надмірною витратою тактів процесора на операцій створення та знищення об'єктів. Також проведено роботу з використанням даного методу в робочий проект для підвищення більш ефективної та стабільної роботи додатку без просадки кадрів.

У результаті розроблено універсальний клас для створення великої кількості об'єктів з моментами оптимізації їх створення в пам'яті, супроводом життєдіяльності таких об'єктів. Класи Pool представлені як asset які може бути перенесений та реалізований в інші проекти для використання оптимізаційних процесів по створенню потрібної кількості об'єктів. Основний принцип полягає у тому що створені об'єкти не видаляються, а вимикаються. Нові об'єкти не створюються, активуватимуться об'єкти з pool, щоб не сповільнювати роботу додатку. Створений програмний продукт написаний на мовах програмування C# під проєкти для Unity3D.

Ключові слова: багатопоковість, перевикористовування, пул об'єктів, алокація пам'яті.

ANNOTATION

Samarasinghe D.S.V. Research of methods for optimizing game drivers of Action games for mobile applications. Master's thesis on the specialty 121 - Software engineering, educational program - 121 Software engineering. Vinnytsia: VNTU, 2023. 142 p.

In Ukrainian speech Bibliography: 30 titles; Fig.: 41; table 16.

In the master's qualification work, the creation of a large number of objects with the method of using the pool to create objects was improved. This approach solves the problem of memory fragmentation, which complicates the search for free contiguous areas of memory, as well as the excessive consumption of processor cycles for operations of creating and destroying objects. Also, work was carried out using this method in a working project to increase the more efficient and stable operation of the application without staff shortages.

As a result, a universal class was developed for creating a large number of objects with moments of optimizing their creation in memory. And also by supporting the life activities of these objects. Pool classes are presented as an asset that can be transferred and implemented in other projects to use optimization processes to create the required number of objects. The main principle is that the created objects are not deleted, but turned off. New objects are not created, objects from the pool will be activated so as not to slow down the application. The created software product is written in the C# programming language for Unity3D projects.

Key words: multithreading, reuse, object pool, memory allocation.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Індустрія мобільних ігор є однією з найшвидше зростаючих галузей розваг та навчальних технологій. За останні кілька років ігри на мобільних платформах стали важливою частиною глобальної індустрії розваг, і цей ринок ще має великий потенціал для подальшого зростання та розвитку. Розробка оптимізованих рушіїв для Action ігор може допомогти забезпечити високоякісний геймплей на цій перспективній платформі та зробити мобільні ігри ще більш захоплюючими для гравців.

Спробуючи проникнути в глибину індустрії комп'ютерних ігор, можна відзначити, що сучасні тенденції демонструють зростаючий інтерес гравців до ігор на мобільних платформах. Ця зростаюча популярність можлива завдяки широкому поширенню смартфонів та планшетів, які стали повноцінними ігровими платформами. Цей факт робить тему дослідження доречною і актуальною, адже можна створювати високоякісні і захоплюючі мобільні додатки, які швидко взаємодіятимуть з користувачем.

Однак, існують виклики і проблеми, які потрібно вирішувати для того, щоб забезпечити високу якість на мобільних платформах. Перш за все, ігри жанру Action відзначаються великою кількістю динамічних об'єктів, складними бойовими сценами та високоякісною графікою, що вимагає високої продуктивності. Мобільні пристрої мають обмежені обчислювальні ресурси та графічні можливості, тому оптимізація графічного рушія є критично важливою для забезпечення гладкого геймплею та задоволення гравців.

Інженерія програмного забезпечення завжди була однією з найбільш динамічних та інноваційних галузей інформаційних технологій. З кожним роком спостерігається стрімкий розвиток обчислювальної техніки та зростаючі вимоги користувачів до продуктивності, функціональності та візуальної якості програмного забезпечення. В цьому контексті ігрова індустрія не є винятком, адже вона не тільки визначає нові стандарти для графічної інтенсивності, але також має величезний вплив на розвиток обчислювальних технологій взагалі.

Зокрема, мобільні ігри стали однією з найпопулярніших форм розваг на сучасних смартфонах та планшетах. Вони привертають мільйони користувачів своєю доступністю та різноманітністю жанрів. Саме тут виникає велика потреба в оптимізації ігрових рушіїв для забезпечення плавності та високоякісної геймплею.

Розглядаючи динаміку ринку мобільних ігор, можна відзначити збільшення конкуренції серед розробників та зростання вимог гравців до якості ігор. Гравці більше не задовольняються простими аркадами, вони очікують від ігор на мобільних платформах захоплюючий сюжет, реалістичну графіку та інтенсивний геймплей, подібний до того, що можна знайти на традиційних ігрових консолях та ПК. Це створює виклик для розробників і вимагає вдосконалення ігрових рушіїв.

З іншого боку, мобільні пристрої мають свої обмеження в апаратному та енергоефективному відношенні, що робить оптимізацію ігрових рушіїв ще більш важливою. Ефективне використання обчислювальних ресурсів та графічних можливостей стає ключовим завданням для розробників мобільних ігор.

На сьогоднішній день існуючі ігрові рушії для мобільних ігор, хоч і ефективні, але не завжди в змозі забезпечити оптимальну продуктивність та графічну якість, особливо для реалістичних Action ігор. Отже, існує потреба в подальшій розробці та вдосконаленні методів оптимізації ігрових рушіїв для забезпечення плавності, якості та ефективності геймплею на мобільних пристроях.

Для досягнення успіху в цій галузі, важливо розуміти, що оптимізація ігрових рушіїв не обмежується лише питаннями продуктивності. Вона також впливає на життєвий цикл батареї мобільних пристроїв, що стосується їхньої енергоефективності та тривалості автономної роботи. Розробка ефективних методів оптимізації стає стратегічно важливою для забезпечення комфортного геймплею та задоволення потреб гравців.

Основними завданнями цього дослідження є розробка та впровадження нових методів оптимізації ігрових рушіїв для Action ігор на мобільних пристроях. Це включає в себе розробку алгоритмів оптимізації графічної складової, удосконалення систем рендерингу, оптимізацію обчислювальних процесів та використання передових технологій, таких як шейдери та паралельні обчислення.

Загальний успіх цього дослідження може значно підняти якість мобільних ігор, зробити їх більш доступними та приємними для гравців, а також сприяти подальшому розвитку мобільної ігрової індустрії. Таким чином, обрана тема дослідження є актуальною та важливою в контексті сучасних викликів та можливостей в інженерії програмного забезпечення.

Тому актуальними є питання дослідження та оптимізація ігрових рушіїв Action ігор мобільних додатків, оскільки існуючі методи оптимізації не задовольняють потреби багатьох робочих процесів мобільних додатків. Це передбачає розробку нових методів і засобів оптимізації.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою роботи є проведенн досліджень методів оптимізації ігрових рушіїв Action ігор мобільних додатків для підвищення продуктивності та оптимізація використання ресурсів за рахунок багатопотоковості процесів та використання пулу (ObjectPool).

Основними задачами дослідження є:

– провести аналізи існуючих методів та засобів створення об’єктів для підвищення продуктивності та оптимізації;

– модифікувати метод підвищення продуктивності використання пам’яті пристрою;

– модифікувати метод оптимізації використання ресурсів за рахунок багатопотоковості;

– створити універсальний Pool Manager на основі патерна пул об'єктів;

– розробити програмні компоненти та систему візуалізації на основі запропонованих методів;

– провести експериментальні дослідження розроблених засобів тестування проекту та продемонструвати отримані результати.

Об’єкт дослідження – процес дослідження методів оптимізації ігрових рушіїв Action ігор мобільних додатків.

Предмет дослідження – методи та засоби оптимізації ігрових рушіїв Action ігор мобільних ігор.

Методи дослідження. У процесі досліджень використовувались: теорія чисел та чисельних методів, теорема Куна-Таккер, методи штрафних функцій, теорія диференціально-інтегрального числення, лінійна алгебра, методи аналітичної геометрії для розробки моделей та методів оптимізації використання ресурсів за рахунок багатопотоковості, комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав методу ігрового пулу, на відміну від існуючих, використано статичні класи та патерни, що дозволило зменшити кількість використання пам'яті в процесі роботи програмного застосунку.

2. Подальшого розвитку отримав метод пулу керуючих потоків, що на відміну від існуючих реалізує інноваційну стратегії динамічного розподілу завдань між потоками для автоматичного адаптування кількості робочих потоків до поточного завантаження системи. Такий підхід дозволяє оптимізувати використання процесорного часу та підвищити загальну продуктивність роботи системи.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано методи оптимізації ігрових рушіїв Action ігор мобільних додатків за рахунок багатопотоковості.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: аналіз методів багатопотоковості ігрових рушіїв, програмна реалізація універсального класу Pool для більш раціонального використання пам'яті в період створення великого кількості об'єктів та контролювання її процесів життєдіяльності.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на

Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022) [1], Міжнародна Науково-Практична Інтернет-Конференція «Електронні Інформаційні Ресурси: Створення, Використання, Доступ» [2].

Публікації. Основні результати досліджень опубліковано в 2 наукових працях, у тому числі 2 – у матеріалах конференцій.

1 АНАЛІЗ ПРОДУКТИВНОСТІ БАГАТОПОТОКОВОСТІ ДОДАТКУ В ЯКОМУ ЧАСТО СТВОРЮЮТЬСЯ ОБ'ЄКТИ

1.1 Багатопотоковість ігрових рушіїв

Багатопотоковими називають додатки, які виконують кілька завдань одночасно в окремих потоках. Додатки, що використовують багатопотоковість, більш оперативно реагують на дії користувача, оскільки інтерфейс користувача залишається активним, у той час як завдання, що вимагають інтенсивної роботи процесора, виконуються в інших потоках. Багатопотокові додатки на мові C# при використанні Mono розробляються за допомогою ключових слів: Thread, ThreadPool і асинхронних делегатів.

Було розглянуто багатопотокове застосування на прикладі будівництва. Представимо, що кожен працівник виконує свої обов'язки одночасно з іншими працівниками. Наприклад, один вимиває підлогу, другий вимиває вікна і т.д. (всі дії відбувається одночасно). Це і є наші потоки.

Thread – клас, який дозволяє створювати нові потоки всередині існуючого додатку [3]. Клас Thread визначає ряд методів і властивостей, які дозволяють керувати потоком і отримувати інформацію про нього. Основні властивості класу:

– ExecutionContext: дозволяє отримати контекст, у якому виконується потік;

– IsAlive: вказує, працює чи потік у поточний момент;

– IsBackground: вказує, є чи потік фоновим;

– Name: містить ім'я потоку;

– ManagedThreadId: повертає числовий ідентифікатор поточного потоку.

Пріоритет: сховище пріоритету потоку – значення перечислення ThreadPriority:

– Lowest;

– BelowNormal;

– Normal;

– AboveNormal;

– Highest.

За умовчанням потоку задається значення Normal. Однак можемо змінити пріоритет у процесі роботи програми. Наприклад, підвищити важливість потоку, встановивши пріоритет найвищий. Середа CLR буде розраховувати та аналізувати значення пріоритетів і на їх підставі виділяти даний потік або іншу кількість часу.

ThreadState повертає стан потоку – одно із значень перечислення ThreadState.

Aborted – потік відновлення.

AbortRequested – для потоку викликаний метод Abort, але встановлення потоку ще не виникає:

– Background: потік виконується у фоновому режимі;

– Running: потік розпочатий і працює (не зупиняючись);

– Stopped: потік завершено;

– StopRequested: потік отримав запит на зупинку;

– Suspended: потік зупинений;

– SuspendRequested: потік отримав запит на зупинку;

– Unstarted: потік ще не був запущений;

– WaitSleepJoin: потік заблокований в результаті дії методів Sleep або Join.

У процесі роботи потоку його статус багатократно може змінитися під дію методів. Так, на початку ще до застосування методу Start його статус має значення Unstarted. Запустивши потік, змінюємо його статус на Running. Визвав метод Sleep, статус зміниться на WaitSleepJoin.

Крім цього статичного властивості CurrentThread класу Thread можна отримати поточний потік.

У програмі на C# є мінімум один потік – головний потік, в якому виконується метод Main.

Асинхронні делегати – асинхронний метод виклику за допомогою делегата, який визначається з такою ж сигнатурою, що і викликаний метод. Для асинхронного виклику необхідно використовувати метод «BeginInvoke». При такому підході делегат бере з потоку «pool» і в ньому виконується деякий код.

ThreadPool – реалізація шаблону «pool object». Його сенс в ефективному управлінні потоками: створення, видалення, призначення їм певної роботи. Повертаючи до будівельної аналогії, ThreadPool – це проробка, яка контролює кількість будівельників на будівництві та призначає кожну з них задачу.

Мова C# надає інструменти для синхронізації потоків. Ці інструменти представлені у вигляді замка і монітора. Вони використовуються для того, щоб виконання блоку коду не здійснювалося кількома одночасно потоками. Однією особливістю, є використання цих інструментів що може привести до «deadlock» у (взаємоблокування потоків). Це відбувається так: потік «А» очікує, коли потік «В» поверне управління, а потік «В», у свою чергу, очікує, коли потік «А» виповнить заблокований код. Тому багатопотоковість і синхронізацію потоків необхідно використовувати з обережністю.

Основна проблема, з якою стикаються при розробці однопоточних програм – це UI-фризи, викликані виконанням складних операцій в основному потоці. В Unity є механізм розпалювання завдань, представлений у вигляді допоміжних програм (coroutine), але він працює в одному потоці. Якщо встановлюємо паралельне виконання функцій в основному потоці, то можна використовувати coroutine. Нічого складного в цьому немає, в документації ця тема дуже добре освітлена. Однак, нагадаємо, що coroutine – це iterators, які в працюють таким чином:

– першим ділом іде реєстрація coroutine;

– після кожного виклику Update і перед викликом LateUpdate, Unity запитує всі зареєстровані програми та обробляє код, який описаний всередині методу, що має тип IEnumerator.

Як згадувалося раніше, coroutine працюють в основному потоці. По цій причині отримуємо фризи, запускаючи в них важкі методи.

Ряд цих недоліків легко втрачається за допомогою реактивних розширень, які в подальшому принесуть ще багато різних покращень та полегшують розробку.

1.2 Порівняльний аналіз аналогів

Пули об'єктів у Unity: якщо хочемо додати пул об'єктів у свій проект Unity, у

вас є три варіанти:

– створити власну систему;
– купити сторонню систему pooling;
– імпортувати UnityEngine.Pool.

Спочатку розглянемо їх. Перший спосіб, створення своєї власної системи pooling, не виглядає надто складним, лише реалізувати кілька операцій:

– створити та видалити пул (Create & dispose);
– взяти з пулу (Take);
– повернутись у пул (Return);
– операції скидання (Reset).

Але це часто стає набагато складніше, коли ви починаєте думати про:

– типобезпеки;
– управління пам'яттю та структурами даних;
– алокація користувача/вивільнення об'єктів;
– потокобезпеки.

Оскільки це вже вирішена проблема, використовується те, що працює, щоб була змога зосередитись на проекті. Щоб задовільнити користувачів мобільного додатку.

Другий спосіб, сторонні системи pooling object, тут потрібно лише вибирати одного з таких постачальників, як:

– The Unity Asset Store;
– Github.

Розглянемо кілька прикладів. Інструмент Pooling Toolkit (див. рисунок 1.1) – містить елегантне об’єднання об’єктів, з можливістю створювати пули GameObjects простим перетягуванням. А для типів об’єктів C# потрібен лише один рядок коду [4].

Простий у використанні. Для запиту або відновлення екземпляра будь-якого об’єкта в пулі потрібен лише один рядок коду. Якщо ви неправильно впораєтеся із взаємодією з об’єктами пулу, ви отримаєте попередження, але система докладе всіх зусиль, щоб виправити помилку та не залишити вас на відкритому повітрі.

Автоматичне вивчення розміру пулу з доступним повним ручним режимом – пули можуть автоматично налаштовуватися для майбутніх запусків залежно від використання.

Рисунок 1.1 – Інструмент Pooling Toolkit

13Pixels Pooling (див. рисунок 1.2) – замість створення екземплярів і знищення об’єктів переробляйте їх. Цей пакет використовує GameObject.SetActive за замовчуванням, але ви можете змінити цю поведінку для prefabs. Наприклад, можна перемістити об’єкт за межі видимої області сцени для ще швидшого об’єднання [5].

Особливості використання:

– використовування вікно налагодження для моніторингу активності пулу та налагодження екземплярів пулу;

– попереднє заповнення різними способами, за допомогою коду або використовуючи компонент попереднього заповнення, який дозволяє завантажувати кілька кадрів для підтримки екрана завантаження;

– об’єкти пулу можна вільно використовувати як дочірні об’єкти скільки

завгодно (немає батьківського об’єкта GameObject, специфічного для пулу).

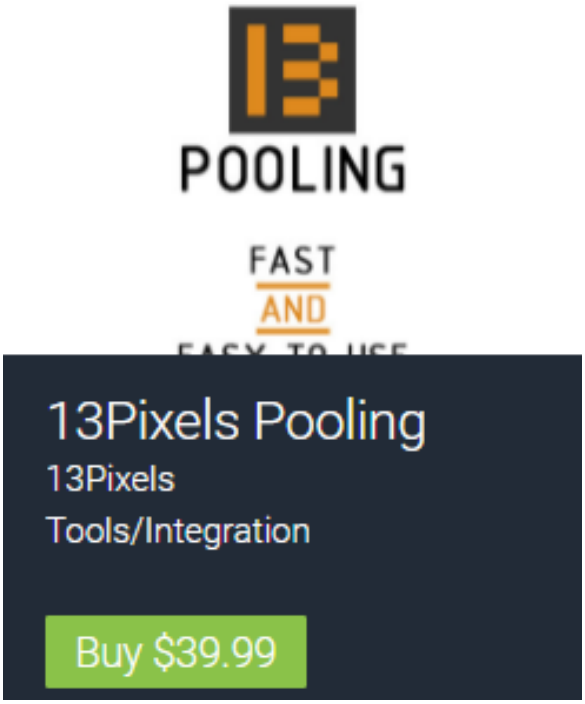

Рисунок 1.2 – Інструмент 13Pixels Pooling

Pure Pool (див. рисунок 1.3) – це професійне та розширене рішення для об’єднання об’єктів, призначене для покращення продуктивності вашої гри [6].

З легкістю замінює виклики Instantiate і Destroy на Acquire і Release і можна створити більш плавну гру.

Об’єднання об’єктів за допомогою Pure Pool дозволяє переробляти та повторно використовувати ваші старі об’єкти, а не постійно їх знищувати та створювати заново.

Замініть ваші виклики Object.Instantiate на виклики Pool Manager.Acquire, а виклики на Object.Destroy із викликами Pool Manager.Release.

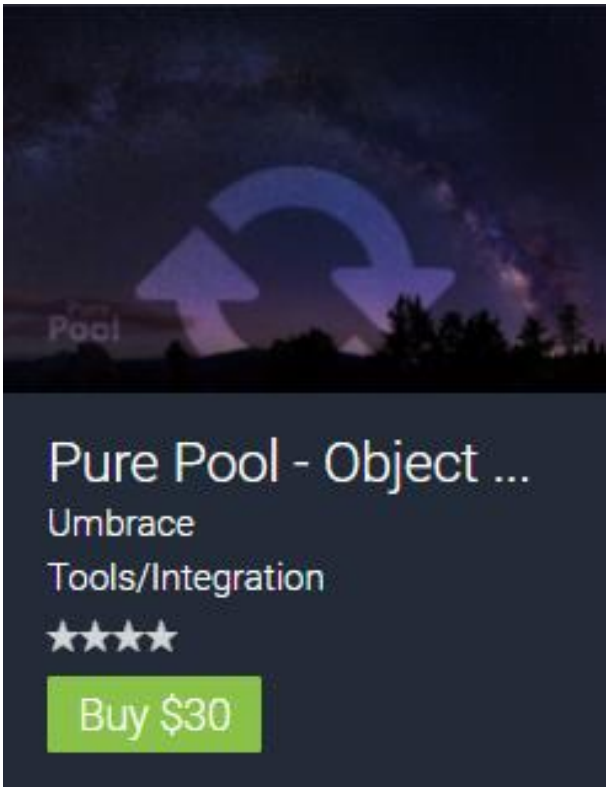

Рисунок 1.3 – Інструмент Pure Pool

Pooling Manager (див. рисунок 1.4) – це конвеєр візуалізації з можливістю сценарію, який можна швидко й легко налаштувати та дає змогу створювати оптимізовану графіку на багатьох платформах [7].

Основні особливості:

– більш гнучкий механізм управління;

– додано власника пулу для централізованого керування всіма об’єктами пулу;

– перемикаючись між сценами, є можливість зберегти або знищити вказаний пул. За замовчуванням ці об’єкти, зібрані під час виконання, буде знищено;

– додано інспектора менеджера пулів. Шлях який має [Window/HHK/Pooling Inspector] у меню;

– Автоматично збільшує розмір пулів.

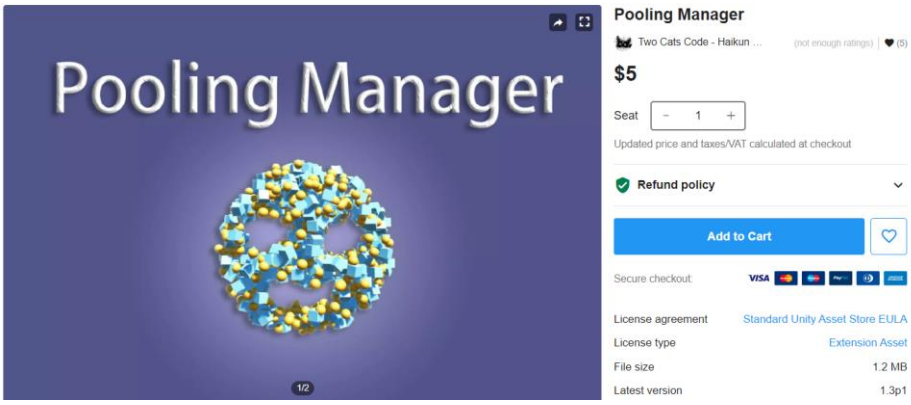

Рисунок 1.4 – Інструмент Pooling Manager

Сторонні інструменти можуть полегшити процес створення і мають багато зручних інструментів. Але вони мають недоліки:

– покладаємося на їх підтримку у виправленні проблем та оновленні пакетів для нових версій редактора;

– якщо немає коду, не маємо змоги виправити проблеми самостійно;

– більше можливостей дорівнює складніший код. Знадобиться час, щоб зрозуміти і використовувати їхню систему;

– вони можуть бути досить дорогими (і за грошима і за часом).

Шкала оцінювання критеріїв Pool Managers:

– не відповідність даному критерію – 0;

– відповідність даному критерію – 1.

Спільний показник критеріїв пул менеджерів (СПКПМ) – це середнє значення всіх необхідних критеріїв пул менеджеру для розробки додатку. СПКПМ обчислюється за формулою 2.1.

$$СПКПМ = (П1+П2+П3+П4)/4 \quad (2.1)$$

де $П_1$ – безкоштовна Ліцензія продукту;

$П_2$ – підтримка версій та оновлень;

$П_3$ – редакція скриптів;

$П_4$ – легкість у використанні.

Створимо порівняльну характеристику Pool Managers у вигляді таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристики Pool Managers

Назва продукту	Безкоштовна Ліцензія продукту	Підтримка версій та оновлень	Редакція скриптів	Легкість у використанні	Сума показників
Pooling Toolkit	0	1	0	1	2
13Pixels Pooling	0	1	0	0	1
Pure Pool	0	0	0	1	1
Pooling Manager	0	0	1	1	2
Властний код pooling	1	1	1	1	4

Результати розрахунку СПКПМ:

– СПКПМ «Pooling Toolkit» дорівнює 1;
– СПКПМ «13Pixels Pooling» дорівнює 0,8;
– СПКПМ «Pure Pool» дорівнює 0,2;
– СПКПМ «Властний код pooling» дорівнює 0,2.

Провівши порівняльний аналіз та між існуючими альтернативами, було вирішено створити свою систему poolings, так як воно має найвищий СПКПМ, та яку можна спокійно оновлювати та не витрачати лишні кошти на їх придбання.

1.3 Ознаки потреби використання пулів

Відомо, що створення, знищення, перемикання між потоками – це затратна операція. Щоб уникнути накладних витрат пов'язаних із цим, основною ідеєю пулу потоків у .NET стало зменшення кількості задіяних потоків і збільшення виконаної ними роботи. Тому в пулі на нас завжди чекає певна відома CLR кількість потоків

готових на виконання завдань. І саме тому майже всі книги про .NET говорять, що для швидкого створення та виконання потоку ThreadPool.QueueUserWorkItem годиться, а метод Start класу Thread немає.

У .NET 3.5 і раніше Thread Pool складається з глобальної черги, в яку потрапляють завдання на виконання з потоку нашої програми та певної кількості робочих потоків – worker threads [8]. Саме робочі потоки розбирають завдання із глобальної черги на виконання, і коли вони виконають завдання – вони звертаються до глобальної черги за новою порцією. Але так як потоки розбирають завдання одночасно, то для того, щоб два робочі не взяли одну і ту ж порцію – існує синхронізація між ними за допомогою блокувань. Блокування можуть стати «пляшковим шийкою» цього механізму і продуктивність через це впаде.

Хороша аналогія для демонстрації того, як працює пул у .net 3.5 – це процес розробки програмного забезпечення (ПЗ). Написати ПЗ у більш короткий термін можна за рахунок збільшення людей на проекті. І якщо це один, два чи три розробники – то на синхронізацію спільних дій у них піде небагато часу [9].

Збільшення кількості ядер у системі аналогічно до збільшення числа учасників, оскільки завдання будуть ділитися на дрібніші шматочки і внаслідок частого звернення до глобальної черги – збільшиться необхідність у синхронізації.

Розробники із Microsoft Reasearch (pfx team) переглянули пул потоків. Адже кількість ядер зростає, і навантаження у майбутньому на глобальну чергу лише зростало б. Тож треба було щось робити. І вони зробили новий пул потоків.

Новий пул потоків працює трохи іншим чином. Для кожного робочого потоку у пулі існує своя локальна черга. Завдання, яке потрапило до локальної черги, може породити дочірню і вони розміщуються в цю ж локальну чергу. Також завдання виконуються у локальній черзі в LIFO (last-in-first-out) порядку, на відміну від пулу в 3.5, де завдання з глобальної черги виконуються робочими потоками у FIFO (first-in-first-out) порядку [10]. Оскільки робочий потік розгрібає свою власну купу, вірніше має доступом до її початку (head), то рівні локальної черги не потрібно ніяких синхронізацій. Тому додавання та вилучення завдання з цієї черги відбуваються дуже швидко. Побічним ефектом такого виконання є те, що черга

відбувається у зворотному порядку. Хоча робити якісь припущення у програмі про порядок виконання завдань у черзі не можна – оскільки пул потоків є своєрідною чорною скринькою.

Коли робочий потік бачить, що робити йому більше нічого, тобто його локальна черга порожня, він намагається поцупити завдання в іншого робочого потоку. Але краде він елементи з хвоста чужої локальної черги. Така дія вже потребує синхронізації. І продуктивність трохи знизиться. Якщо всі локальні черги порожні, то робочий потік спробує забрати завдання з глобальної черги в FIFO порядку. Якщо ж глобальна черга виявиться порожньою, тоді робочий потік «засне», поки не з'явиться робота для нього. Якщо робочий потік проспав занадто багато часу і роботи так і не отримав, він прокинеться і знищиться, звільнивши ресурси, що займаються.

Пул об'єктів – фондовий шаблон проектування, набір ініціалізованих і готових об'єктів для використання. Його використовують для підвищення продуктивності, коли створення нового об’єкта призводить до великих витрат [11]. Важливо знати, що вбудований у .NET збірник сміття чудово справляється з видаленням недовго тривалих об’єкті, тому обмеження застосовності pool дотримуються наступних критеріїв:

– для створення та/або видалення об'єктів (наприкладі: сокети, потоки, некеровані ресурси);

– очищення об'єктів для перевикористання кращого створення (або нічого не варто);

– предмети дуже великого розміру.

Більш детально про останній пункт, якщо ваш об'єкт займає в пам'яті 85 000 байт і більше, він переходить у відкриті більші об'єкти, у другому поколінні збірки мусора, що автоматично робить його «довготривалим» об'єктом. Прибавимо до цієї фрагментованості і виникне потенційна проблема нехватки пам’яті при постійному виділенні/видаленні.

Ідея Pool складається в тому, щоб організувати перевикористання «дорогих» об'єктів, за наступним сценарієм:

Проблеми такого підходу:

– після виконання робіт з об'єктом може знадобитися його скинути в початковий стан, щоб попереднє використання не вплинуло на наступні;

– пул повинен забезпечувати потокобезпечність, адже застосовується він, як правило, в багатопотокових системах;

– пул повинен обробляти ситуацію, коли в ньому не залишилося доступних для видачі об'єктів.

1.4 Постановка задач дослідження

Пул об'єктів в Unity визначено має деякі важливі недоліки, які приносять більше шкоди, ніж користі, тому використовуємо його потрібно з умом.

Варто розглянути можливість використання pooling, коли:

– створюємо і знищуємо ігрові об'єкти дуже швидко, наприклад пулі зброї.

– часто allocate і вивільняємо об'єкти, що зберігаються в кучі (в місці їх повторного використання). Це відноситься і до колекцій C#.

Ці операції викликають багато виділень, внаслідок чого можемо отримати:

– розхід тактів процесора на операції створення та знищення (або new/dispose);

– передчасною збірки сміття, що викликають фризи, які наші гравці не оцінюють;

– фрагмент пам'яті, що ускладнює пошук вільних міжміських областей пам'яті.

Сутність може бути чим завгодно: ігровим об'єктом, instance prefab, словником C# і т.д.

Наприклад, можна взяти багаторазові сумки. Зрештою, потрібні контейнери, щоб донести які продукти додому. Так, ви берете пусті багаторазові сумки, наповнюєте їх продуктами і повертаєтеся додому. Вернувшись додому, звільняємо свої сумки і кладемо їх навпаки в ящик.

Багаторазові сумки – найкраща альтернатива, ніж покупка (розподіл) пластикових пакетів та їх вивільнення кожен раз, коли ви збираєтеся купувати.

Добре, ідемо до свого полю сумок (наприклад, ящик на кухні), беремо кілька, використовуємо їх, виймаємо все з них і, нарешті, повертаємо їх навпаки в пул.

Ось основні деталі використання pooling:

– елементи, для яких ви хочете задіяти пул, наприклад, багаторазові сумки, згенерована пуля;

– глобальна ціль для всіх цих елементів, наприклад, перенесення продуктів, стрільба пулями;

– функції, які ви виконуєте над пулом і його елементами: Take (взяти), Return (вернути), Reset (скинути).

З урахуванням цих деталей були складені вимоги до нового класу:

– типобезпечність pool на етапі компіляції;

– робота pool з будь-якими класами, в тому числі іноземними;

– просте використання в коді;

– автовиділення нових об'єктів при нехватці, їх ініціалізація користувача;

– обмеження кількості загальних виділених об'єктів;

– автоочищення об'єкта при його поверненні в пул;

– потокобезпечність (бажано, з урахуванням витрат на синхронізацію);

– підтримка поширення екземплярів пулу (звідси випливає хоча б найпростіший контроль того, щоб об'єкти поверталися саме в пули).

Після аналізу актуальності стану питання оптимізації ігрових рушіїв Action ігор мобільних додатків та порівняння існуючих аналогів за певним переліком критеріїв було визначено такі завдання:

– створення класу Pool<T>;

– створення статичного класу Pool Manager;

– розробка ігрових рушіїв з використанням Pool;

– розробка Asset Pool Manager;

– створення Unity Package.

1.5 Висновки

У першому розділі було розглянуто багатопотоковість ігрових рушіїв, надано інформацію про важливість використання потоків для оптимізації роботи додатків. Описано ключові елементи, такі як клас Thread, ThreadPool та асинхронні делегати, зазначено їхні основні властивості та використання. Висвітлено синхронізацію потоків, включаючи замки та монітори, та зазначено можливі проблеми, такі як "deadlock". Надано огляд використання асинхронних методів та корутин для оптимізації важких операцій в основному потоці.

Порівняно систем-аналогів, окреслення їх сильних та слабких сторін. Серед аналогів було виділено такі програмні продукти, як Pooling Toolkit, E-13Pixels Pooling, Pure Pool, Pooling Manager, Властний код pooling. Наведено порівняльну таблицю характеристик пул менеджерів, та визначено між існуючими альтернативами було вирішено створити свою систему poolings.

Розглянуто основні проблеми з управлінням потоками в .NET, спроби оптимізації за допомогою пулів та використання пулів об'єктів для зменшення витрат пам'яті та оптимізації ресурсів. Вказано на переваги нового пулу потоків, який використовує локальні черги та дозволяє збільшити продуктивність.

Також згадано про проблеми і обмеження пулів об'єктів, такі як необхідність скидання об'єктів в початковий стан та потреба в потокобезпечності.

З огляду на недоліки аналогів було визначено такі основні задачі для розробки, як розробка методу ігрових рушіїв з використанням Pool, розробка Asset Pool Manager, створення власного Unity Package. З урахуванням цих проблем були складені вимоги до нового класу, таким чином щоб економимо такти процесора, необхідні для створення і знищення цих prefab. Крім того, зменшуємо навантаження на збірник сміття

У результаті порівняння було доведено необхідність розробки магістерської кваліфікаційної роботи та сформульовано задачі дослідження, які необхідно розв’язати для вирішення питання.

2 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ОПТИМІЗАЦІЇ ІГРОВИХ РУШІЇВ

2.1 Обґрунтування вибору створення Unity Package

Unity3D – платформа, яка існує досить давно і постійно розвивається. Однак, працюючи в ньому з кількома проектами одночасно, має складність у використанні загальних вихідних файлів (.cs), бібліотеки (.dll) та інших ресурсів (зображень, звуків, моделей, prefabs). У цьому розділі опишемо про досвід роботи з нативними рішеннями, такі проблеми як для Unity3D.

Методи поширення загальних ресурсів. Існує більше одного способу використовувати загальні ресурси для різних проектів, але у кожного підходу є свої плюси та мінуси.

1. Дублювання – «ручним способом» дублюємо ресурси між проектами.

Переваги:

– підходить для всіх видів ресурсів;

– немає проблем з залежностями;

– немає проблем з активами GUID’ами.

Недоліки:

– гігантські репозиторії;

– немає можливості версії;

– складність відстеження змін в загальних ресурсах;

– складність оновлення загальних ресурсів.

2. Підмодулі Git – поширення загальних ресурсів через зовнішні підмодулі.

Переваги:

– можна працювати з вихідниками;

– можна поширювати активи;

– немає проблем з залежностями.

Недоліки:

– необхідний навик роботи з Git;

– Git не дуже об'єктивно працює з бінарними файлами – приходиться

додатково підключати LFS;

– розмежування доступу для репозиторіїв;

– складності при підвищенні та зниженні версії;

– можливі зіткнення GUID’ів і немає однозначних повідомлень зі сторони Unity для їх дозволів.

3. NuGet – поширення загальних бібліотек через NuGet-пакети.

Переваги:

– зручна робота з проектами, не залежними від Unity;

– версії і дозвіл залежностей.

Недоліки:

– Unity не вміє працювати з NuGet-пакетами «з коробки» (на GitHub можна знайти NuGet Package Manager for Unity, який виправляє це, але є нюанси);

– складності при поширенні інших видів активів.

4. Unity Package Manager – розповсюдження загальних ресурсів через нативне рішення для Unity.

Переваги:

– легко зрозумілий інтерфейс для роботи з пакетами;

– захист від перезапису файлів .meta в пакетах при конфліктах GUID’ов;

– можливість версії;

– можливість поширення всіх видів ресурсів для Unity3D.

Недоліки:

– все ще може бути випадком конфліктів GUID’ів;

– немає документації для реалізації.

Останній спосіб має більше переваг, ніж недоліків. Однак він зараз не дуже популярний із-за відсутності документації, і тому опишемо його більш детально.

Unity Package Manager (далі UPM) – інструмент для управління пакетами. Його додали в Unity 2018.1, і він використовувався лише для пакетів, які розроблялися Unity Technologies. Однак починаючи з версії 2018.3 з'явилася можливість додавання custom packages (див. рисунок 2.1).

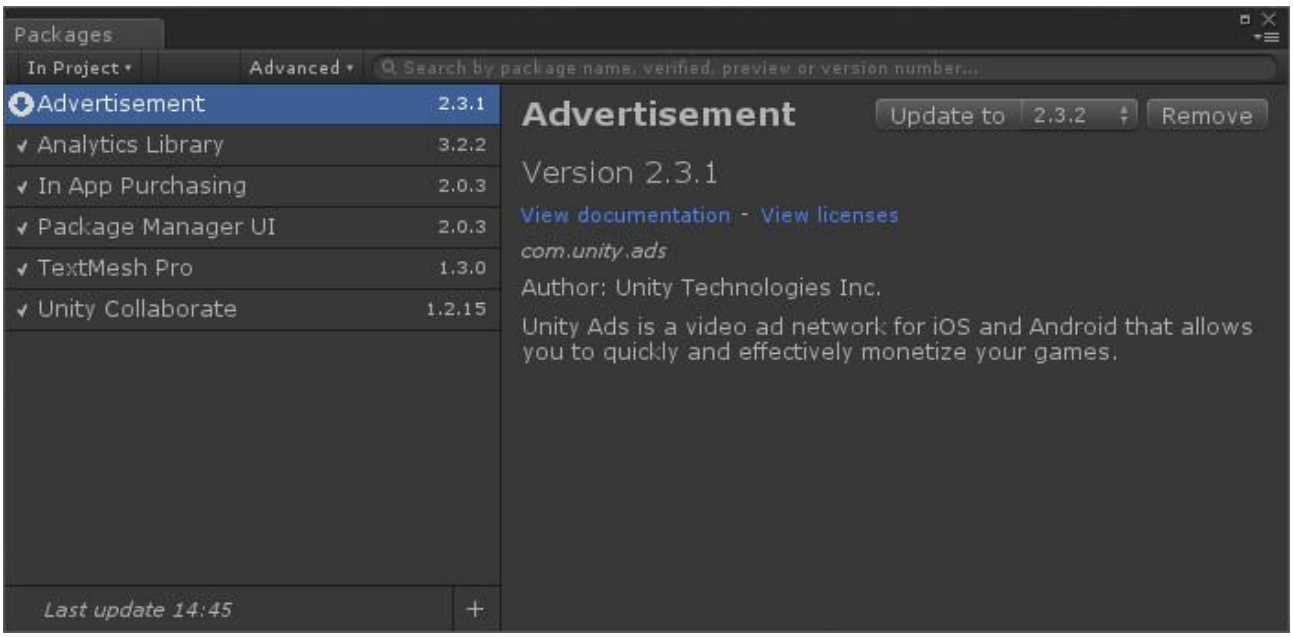

Рисунок 2.1 – Інтерфейс Unity Package Manager

Пакети не потрапляють у вихідники проекту (директорію Assets). Вони знаходяться в окремій директорії %projectFolder%/Library/PackageCache і ніяк на проект не впливають, їх єдине згадування у вихідних документах – у файлі packages/manifest.json (див. рисунок 2.2).

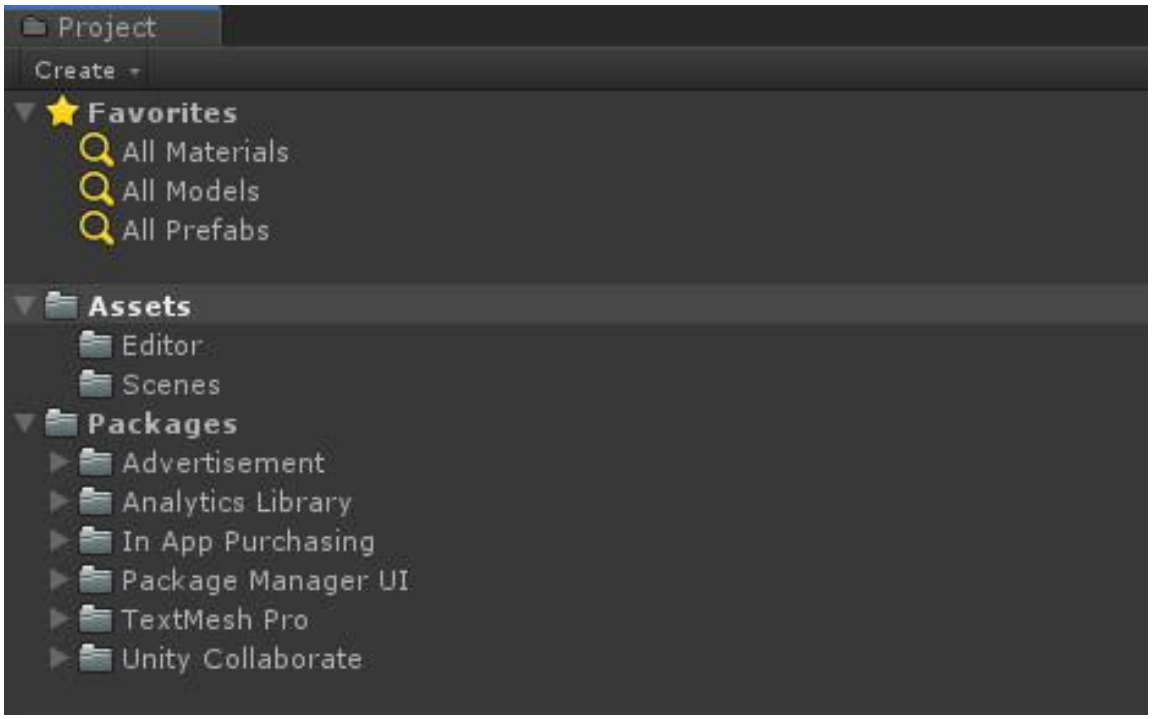

Рисунок 2.2 – Пакети в файловій системі проекту

UPM може використовувати кілька вихідних пакетів:

1. Файлова система.

Переваги:

– швидкість реалізації;

– не вимагає сильних інструментів.

Недоліки:

– складність версії;

– необхідний загальний доступ до файлової системи для всіх, хто працює з проектом.

2. Git-репозиторій.

Переваги:

– потрібен тільки Git-репозиторій.

Недоліки:

– не можна переключатися між версіями через вікно UPM;

– працює не зі всіма Git-репозиторіями.

3. npm-репозиторій.

Переваги:

– повністю підтримує функцію UPM і використовується для розповсюдження офіційних пакетів Unity.

Недоліки:

– На сьогодні ігнорує всі строкові версії пакетів, крім «preview».

Нижче розглядаємо реалізацію UPM + npm. Ця зв'язка зручна, оскільки дозволяє працювати з будь-якими видами ресурсів і керувати версіями пакетів, а також повністю підтримує нативний інтерфейс UPM.

У якості npm-репозиторія можна використовувати Verdaccio. До нього є детальна документація, і для його запуску потрібно буквально пара команди.

Щоб створити пакет, необхідно помістити файл package.json, який буде його описувати, в директорію з вмістом цього пакета. Потрібно зробити наступне:

– перейти в директорію проекту, яку хочу зробити пакетом;

– виконати команду npm init і під час діалогу введіть необхідні значення.

Для імені вказуємо ім'я у форматі зворотного домену, наприклад com.plarium.somepackage;

– для зручного відображення імені пакета – додаємо властивість displayName в package.json і заповнюємо його;

– так як npm js-орієнтований, у файлі є непотрібні властивості main і скрипти, які Unity не використовує. Найкраще їх видалити, щоб не засорити опис пакета;

– відкриваємо Unity та створюємо файл .meta для package.json (Unity не бачить активи без файлів .meta, пакети для Unity відкриваються лише для читання).

Установка та оновлення пакетів через Unity Package Manager. Щоб додати пакет в Unity-проект, потрібно:

– записати у файл manifest.json інформацію про вихідні пакети. Для цього необхідно додати властивість scopedRegistries і вказати «Scope» та адресу джерела, за якою буде відбуватися пошук конкретної «Scope».

– перейти в Unity і відкрийте вікно Package Manager'а (робота з кастомними пакетами не відрізняється від роботи зі вбудованими).

– вибрати всі пакети.

– знайдіть потрібний пакет і додайте його (див. рисунок 2.3).

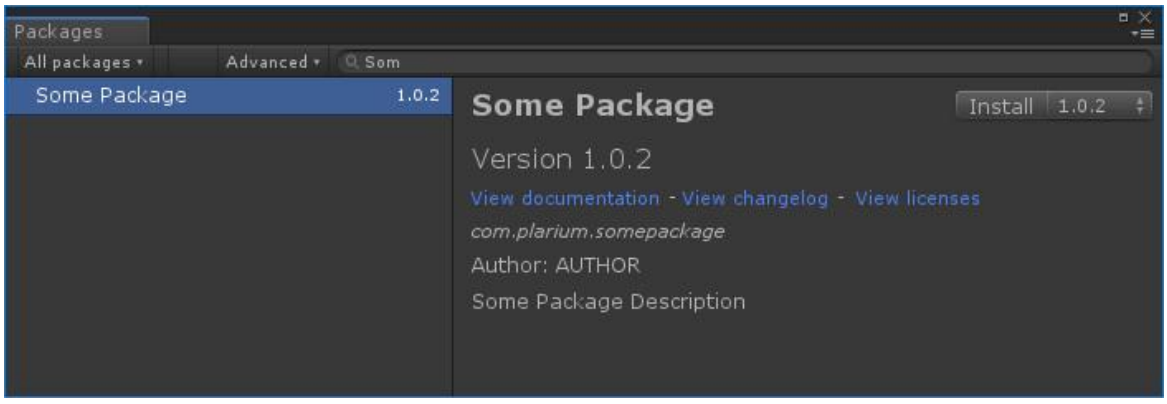

Рисунок 2.3 – Пошук потрібного пакету

Щоб вихідні дані підключилися до проекту, необхідно створити визначення збірки для пакета.

Використання пакетів не обмежує можливості для відладки. Однак при

роботі з пакетами в Unity не можна переходити в IDE після натискання на помилку в консолі, якщо в пакеті виникла помилка. Це пов’язано з тим, що Unity не бачить скрипти як окремі файли, оскільки під час використання визначення Assembly вони збираються в бібліотеці та підключаються до проекту. При роботі з вихідниками з проекту доступний перехід в IDE по кліку.

Додані до проекту пакети Unity відкриті лише для читання, але їх можна редагувати у кеші пакетів. Для цього необхідно:

– перейти до пакету в кеші пакетів (див. рисунок 2.4);
– доповнити потрібні зміни;
– оновити версію у файлі package.json;
– надіслати пакет npm publish --registry «адреса до сховища пакетів»;
– оновити версію пакета перед виправленою через інтерфейс UPM.

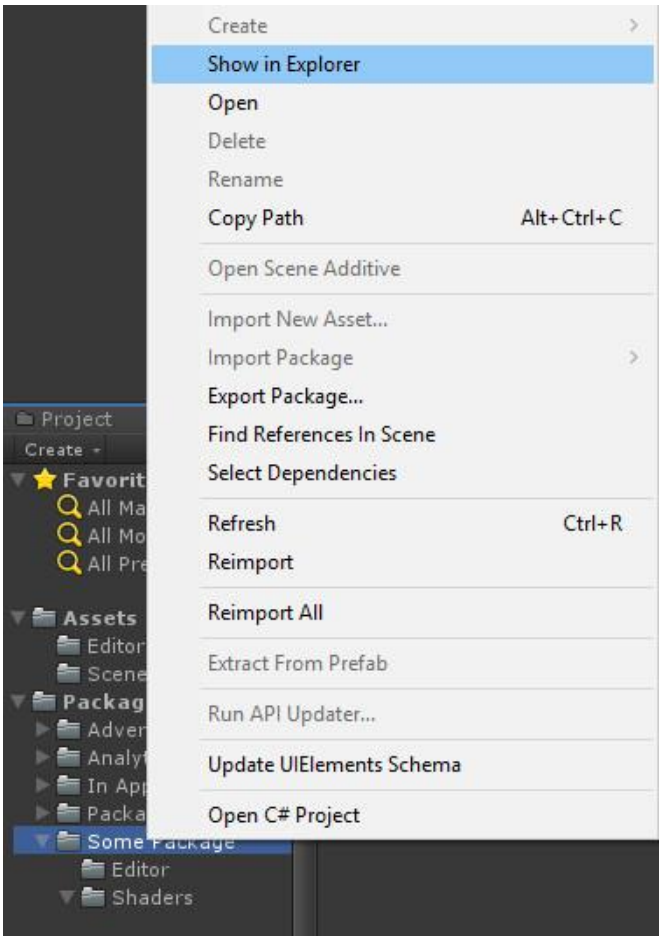

Рисунок 2.4 – Перехід до пакету в кеші пакетів

При імпорті пакетів можуть статися такі конфлікти GUID'ів. Пакет – пакет. Якщо при імпорті пакета виявиться, що в вже доданих пакетах є asset з таким же GUID'ом, assets з GUID'ами, що збіглися, з імпортованого пакета не додаються в проект.

Пакет – проект. Якщо при імпорті пакета виявиться, що в проекті є asset з GUID'ами, що збігаються, то asset з пакета не вносяться в проект. Однак assets, що залежать від них, почнуть використовувати assets із проекту.

Якщо перенести asset з проекту в пакет при відкритій Unity, його функціональність збережеться, а посилання в залежних assets почнуть використовувати asset з пакета.

Важливо: під час копіювання asset з проекту до пакета відбудеться конфлікт «Пакет – проект», описаний у розділі вище.

Перепризначення GUID'ів за власними алгоритмами при імпорті всіх assets, щоб унеможливити колізії.

Додавання всіх assets до одного проекту з наступним поділом на пакети.

Створення бази даних, що містить GUID'и всіх assets, та проведення validation під час відправлення пакетів.

UPM – це нове рішення для поширення спільних ресурсів на Unity3D, яке може стати гідною альтернативою існуючим методам.

2.2 Створення класу Pool<T>

Щоб створити сценарій в Unity3D, ви зазвичай виконуєте такі дії:

– відкриваємо Unity3D: запускаємо редактор Unity3D;

– вибираємо проект: обираємо існуючий проект Unity3D або за потреби створіть новий;

– папка Assets: в інтерфейсі Unity3D знаходимо і вибираємо папку «Assets» на панелі «Project». Тут зберігаємо свої сценарії та інші ресурси.

Створюємо новий сценарій C#:

– натискаємо правою кнопкою миші папку «Активи» або певну папку, куди потрібно розмістити сценарій;

– у контекстному меню вибираємо «Створити», а потім «Сценарій C#»;

– називаємо свій сценарій: даємо назву своєму сценарію. Unity3D автоматично додає розширення файлу «.cs»;

– відредагуємо свій сценарій.

Після створення сценарію під назвою Pool<T> розпочнемо створювати основні поля які знадобяться для нашої задачі. Для початку потрібно створити ще один сценарій який представляє контейнер для об’єкта в пулі. Назвемо його PoolSlot<T> (див. рисунок 2.5).

Визначимо в ньому 2 основних методи:

1. Consume – в ньому будемо описувати споживання об'єкту;
2. Release – описуємо вивільнення об’єкту після його використання.

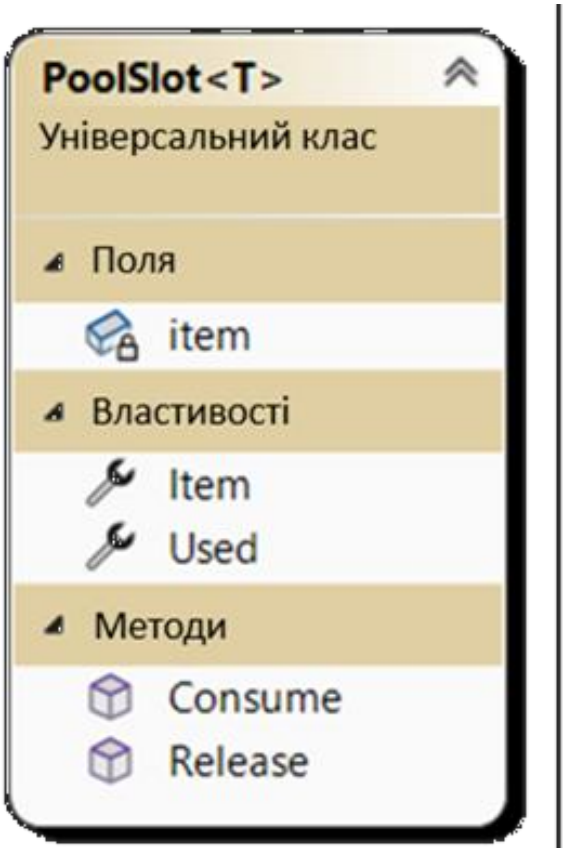

Рисунок 2.5 – Діаграма класу PoolSlot<T>

PoolSlot<T> – це клас, який представляє контейнер для об’єкта в пулі. У ньому є поля для зберігання елемента (Item) і відстеження того, чи він використовується (Used).

Методи Consume та Release використовуються для позначення слота як використаного та вільного відповідно.

Після чого повертаємося до написання нашого основного класу Pool<T>. В нас вже є сформований сценарій нашого контейнера тепер визначимо інші поля (lastIndex, list, lookup). Після чого напишем деякі властивості:

– Count;

– CountUsedItems.

Далі почнемо описувати потрібні методи такі як:

– Warm;

– CreateContainer;

– GetItem;

– ReleaseItem.

Діаграма ілюструє основну структуру класу Pool<T> і пов’язаного з ним класу PoolSlot<T>, а також ключові поля та методи (див. рисунок 2.6).

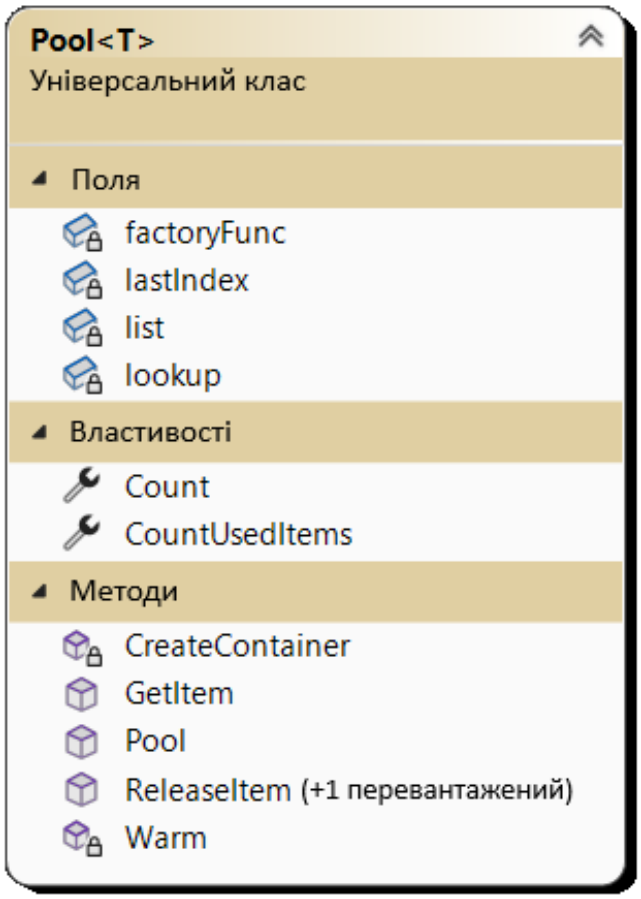

Рисунок 2.6 – Діаграма класу Pool<T>

Pool<T> – це клас, що представляє пул об’єктів. У ньому є поля для керування списком об’єктів (список) і словник (пошук) для відстеження їх використання.

factoryFunc – це делегат, який використовується для створення нових об’єктів для пулу.

lastIndex використовується для відстеження останнього доступного індексу в списку.

Такі методи, як Warm, CreateContainer, GetItem і ReleaseItem, керують об’єктами в пулі.

2.3 Створення статичного класу Pool Manager

Pool Manager – це клас який можемо викликати в любому сценарій, так як він унаслідується від Singleton<T> який створюється при в Awake() і потім живе в DontDestroy().

Основна задача Pool Manager це керування створеними pool object. Він має в собі такі параметри:

– LogStatus – це bool який буде виводити повідомлення у консоль для візуального перевірки праце здатності Pool Manager;

– Root – це варіант об'єкту який потрібно буде створювати в пулі;

– PrefabLookup – являє собою словник (Dictionary «GameObject», Pool «GameObject») в якому створюються початкова кількість об'єктів;

– InstanceLookup – являє собою словник (Dictionary «GameObject», Pool «GameObject») в якому створюються об'єкти кількість якої перевищує початкову тобто додаткове створення об'єктів;

– Dirty – прапор який вмикається для відстеження, створення додаткових об'єктів.

Тепер розглянемо набір методів який потрібен для повноцінного функціювання сценарію, та які данні будемо запитувати для коректної роботи. Так як Pool в нас самописний то є можливість описати всі потрібні методи для подальшої роботи або дописувати в Pool Manager нові методи які знадобляться в

подальшої розробці. Спочатку почнемо описувати наші основні методи які потрібні в сценарії Pool Manager.

– WarmPool – метод для створення пулу з такими параметрами об'єкт який треба буде створити та кількість первинного створення;

– ReleaseObject – метод для звільнення об'єкта з пулу;

– SpawnObject – для включення об'єкта пулю. Якщо кількість використовуваних об'єктів перевищує розмір пулу, будуть створені нові об'єкти.

Реалізована діаграма надає спрощений огляд структури та функціональності вашого класу Pool Manager (див. рисунок 2.7).

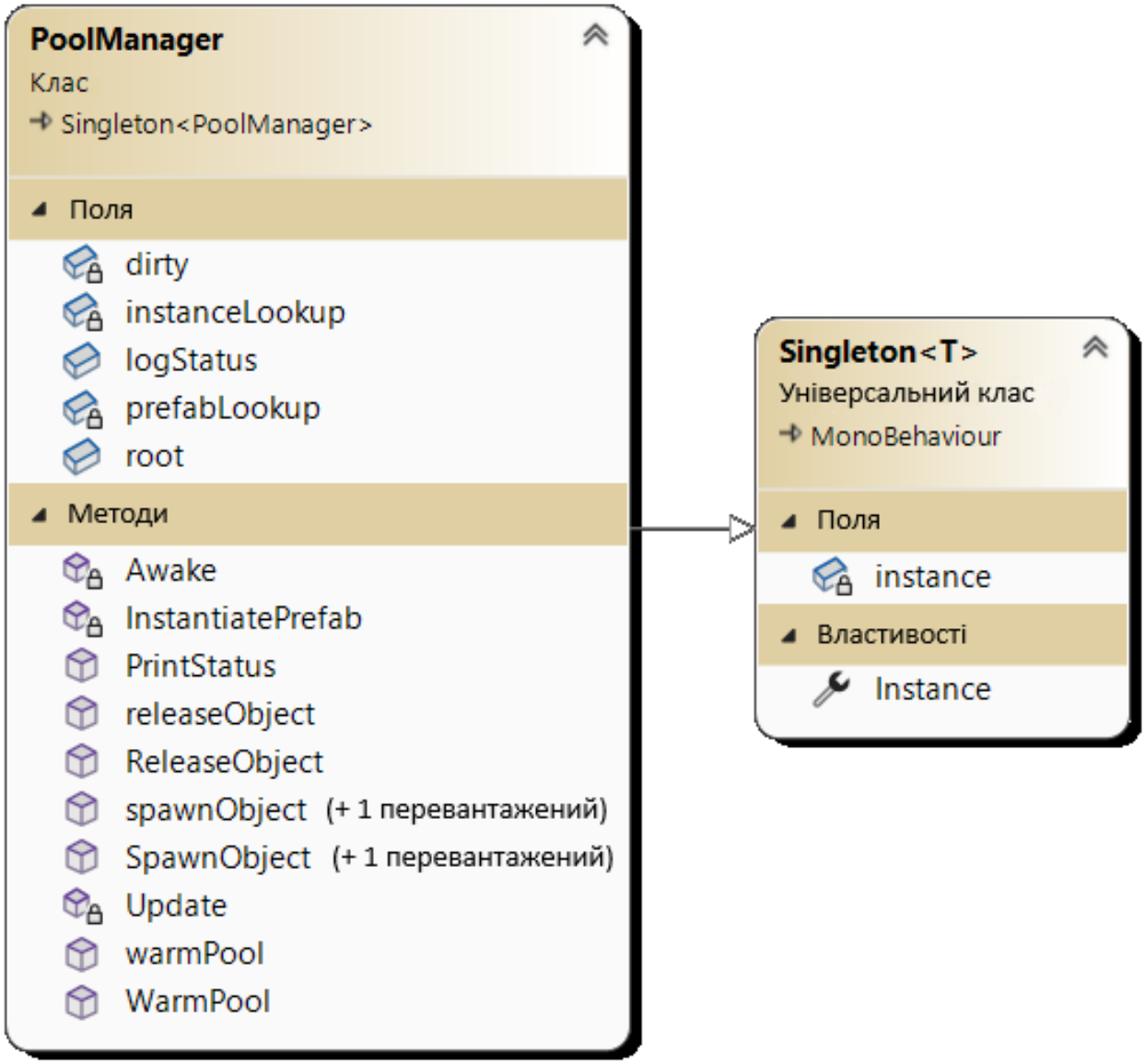

Рисунок 2.7 – Діаграма класу Pool Manager

Pool Manager – це клас MonoBehaviour в Unity. Він має різні поля для конфігурації, такі як logStatus і root. Два словники, prefabLookup та instanceLookup, використовуються для керування pool objects. Прапор dirty використовується для відстеження того, чи потрібно реєструвати статус пулу об’єктів. Клас містить методи для різних завдань керування пулом, включаючи створення та звільнення об’єктів.

Область Static API надає набір статичних методів для доступу до менеджера пулу, що дозволяє взаємодіяти з ним з інших класів за допомогою статичних методів.

Реалізована діаграма надає спрощений огляд структури та функціональності вашого класу Pool Manager (див. рисунок 2.8).

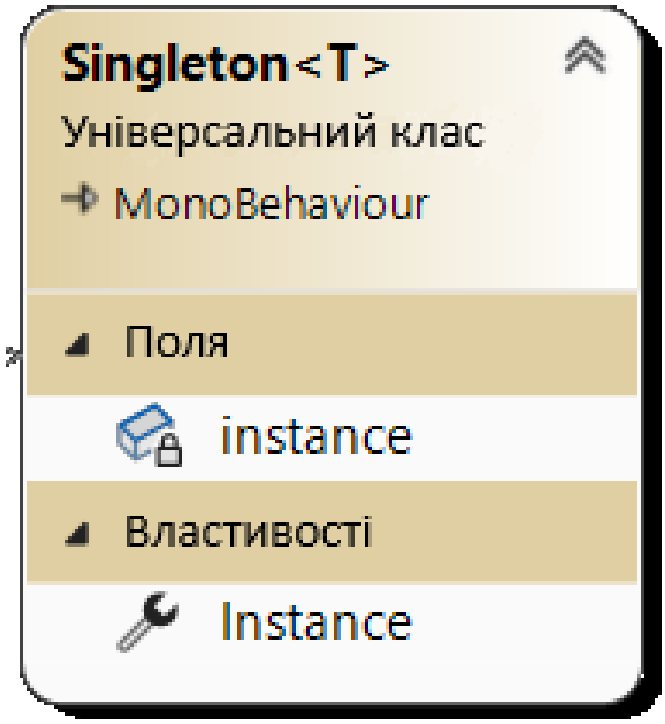

Рисунок 2.8 – Діаграма класу Singleton<T>

Singleton<T> – це загальний клас MonoBehaviour в Unity. Він визначає екземпляр властивості типу T, який дозволяє отримати доступ до єдиного екземпляра класу. Поле екземпляра використовується для зберігання фактичного екземпляра.

У методі Awake (не показано на схемі) клас виконує перевірки та налаштування, щоб переконатися, що існує лише один екземпляр класу. Якщо екземпляр не існує, він створює його.

Цей клас Singleton<T> забезпечує застосування шаблону singleton, гарантуючи, що може існувати лише один екземпляр похідного класу (наприклад, YourSingletonClass). Це загальний шаблон в Unity для створення класів менеджерів або контролерів, які повинні мати лише один екземпляр протягом життєвого циклу гри.

2.4 Розробка ігрових рушіїв з використанням Pool

Однією з ключових вимог до сучасних ігор є висока продуктивність та безперервна ігрова динаміка. Використання Pool є важливим стратегічним рішенням для досягнення цих цілей. Цей pattern дозволяє ефективно управляти об'єктами, зменшуючи навантаження на ресурси системи та роблячи гру більш доступною та оптимізованою Забезпечуючи перерозподіл ресурсів та ефективне управління об'єктами, розробники можуть створити ігральні рушії, які працюють стабільно, навіть при великій кількості взаємодіючих об'єктів. У цьому розділі ми розглянемо, як використання Pool сприяє оптимізації ігральних рушіїв і забезпечує відмінний геймплей.

1. Початок роботи додатку.
2. Ініціалізація роботи сценаріїв Unity відбувається в таких методах як Awake() який викликається один раз при ініціалізації об'єкта перед будь-яким іншим методом.
3. Запит на створення нового Pool за типом об’єктів, відбувається в методі Start() використовується для виконання дій, які повинні статися один раз на початку виконання сценарію, такі як ініціалізація змінних чи налаштування посилання та їхню початкову кількість.
4. Ініціалізація статичного класу Pool Manager, як правило, відбувається автоматично при завантаженні сцени або в момент запуску додатка. Є кілька можливих способів роботи зі статичними методами в Unity.

5. Якщо запит надійшов на створення нового об'єкту за певним типом.

6. Створюємо масив об’єктів за вказаними початковими даними за допомогою методу Instantiate(), який використовується для створення нових екземплярів об'єктів в грі та вимикаємо сам створений об’єкт.

7. Якщо надійшов запит на отримання об’єкту то ми звертаємося до нашого масиву об’єктів та запитуємо у нього потрібний вільний об’єкт, якщо це інший запит то завершуємо роботу.

8. Беремо об’єкт з нашого Pool масиву та віддаємо його до запитуваного місця, де з ним проводяться інші операції.

9. У разі інших запитів завершуємо роботу та переходим у основний потік роботи додатку.

На рисунку 2.9 зображена блок-схема алгоритму методу ігрових рушіїв з використанням Pool.

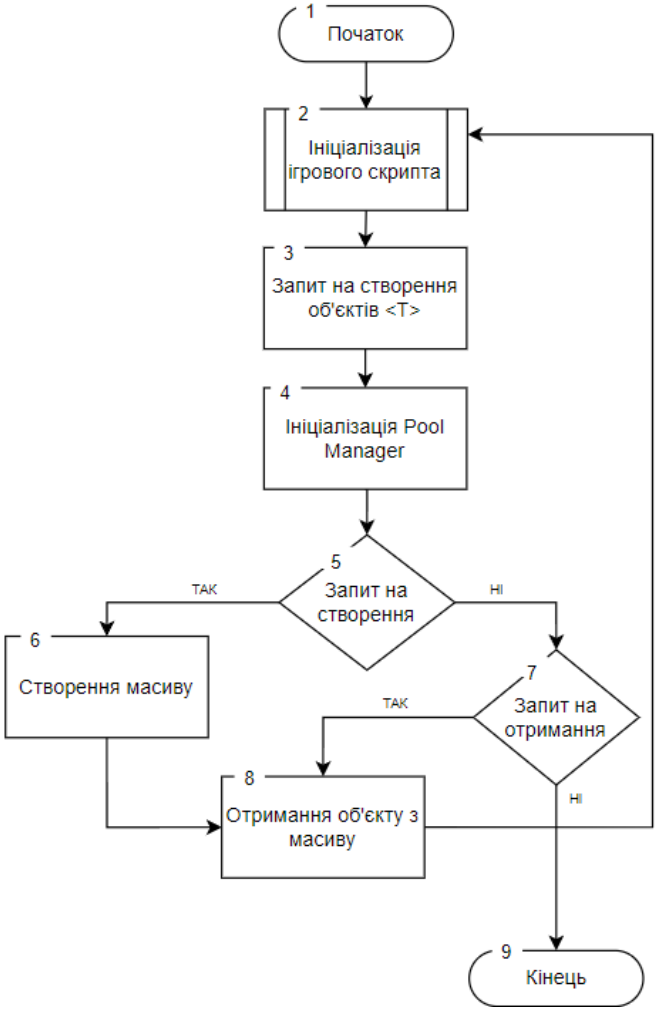

Рисунок 2.9 – Блок-схема алгоритму методу ігрових рушіїв з використанням Pool

Сутність методу полягає в тестуванні гри, яка має на меті переживання ворожих атак і досягнення кінця кожного рівня. Гравець переміщується з одного рівня на інший, де ворожі одиниці стають складнішими з кожним наступним рівнем. Гра складається з одних і тих самих елементів, але розташованих в різних комбінаціях та кількості. Кожен рівень має ігрового персонажа, супротивників, задній фон, корисні предмети для бонусів та точки входу та виходу на новий рівень.

Супротивники у грі є Non-Player Characters (NPC), що означає персонажі, керовані комп'ютером, і вони мають взаємодіяти з гравцем. У рольових іграх термін "NPC" вказує на персонажів, які можуть бути дружніми, нейтральними або ворожими. NPC є важливим елементом створення ігрової атмосфери та мотивації гравців [12].

Рівень додатку, який розробляється, повинен включати наступні елементи:

1. Ігрового персонажа – це основний герой, яким управляє гравець під час гри. Його дії і взаємодія з іншими елементами створюють головну динаміку гри.
2. Супротивників (Non-Player Characters - NPC) – це ворожі персонажі, керовані комп'ютером. Вони можуть бути дружніми, нейтральними або ворожими і додають елемент виклику та стратегії до гри.
3. Задній фон планети – ігровий світ повинен мати атмосферний тло, що створює відчуття місцевості та контексту для подій гри.
4. Корисні предмети – об'єкти, які гравець може зібрати або використовувати для отримання бонусів, покращення чи стрільби. Ці предмети додають різноманітність ігровому процесу.
5. Вхід та вихід на новий рівень – точки, через які гравець може переходити між рівнями гри. Вони визначають, коли завдання на одному рівні виконано та гравець готовий перейти до наступного виклику.

Для поліпшення інтересу гравців до мобільного ігрового додатку використовується метод ігрових рушіїв з використанням Pool. Цей метод включає послідовність дій, які визначають оптимізацію програмного коду гри, що позволяє додавання різних типів рівнів та можливість гравців купувати поліпшені ігрові кораблі для подальшого проходження гри. Можливості ігрового процесу:

1. Гра розпочинається, і для ідеального проходження гравцеві необхідно завершувати рівні з повним запасом життя і захисного поля. Якщо гравець отримав 2 зірки, його життя залишається на рівні 100%, а захисне поле стає менше 0%.

2. Починається ініціалізація ігрових одиниць середнього типу, яка аналізує ігрові навички гравця на перших п'яти рівнях.

3. Якщо гравець легко пройшов перші 5 рівнів, йому випадає можливість виконати наступні 5 рівнів із важкими одиницями, що мають вищі характеристики життя та сили пошкодження. Якщо гравець не подолав всі перші 5 рівнів, йому дається шанс пройти наступні п'ять легших рівнів для покращення навичок.

4. Починається ініціалізація легких рівнів, які мають багато ворожих одиниць, але із меншими характеристиками життя. Рівень пошкодження ігрового корабля зроблений невеликим для сприяння звиканню до гри і накопичення гравцем ігрової валюти для майбутніх покупок вдосконаленого корабля. Після успішного проходження 5 рівнів гравцю надаються рівні із середніми ігровими персонажами.

5. У випадку поразки на будь-якому рівні, гравець отримує можливість відвідати ігровий магазин та купити нові кораблі із покращеними характеристиками. Пройшовши такі рівні, гравцю доступний рівень із босом.

6. Гравець має можливість придбати новий корабель в ігровому магазині, де є шість доступних кораблів із різними характеристиками. Покупка здійснюється за допомогою ігрової валюти, яку гравець збирає, знищуючи ворожі одиниці.

7. Починається ініціалізація важких рівнів, які представляють собою найскладніші ворожі одиниці. Ці рівні можуть бути здолані тільки гравцями з високим рівнем ігрових навичок, які отримали три зірки на кожному з попередніх рівнів.

8. Гравець зустрічає рівень із босом, якщо успішно пройшов важкий шлях і підвищив свої ігрові навички.

9. У випадку, якщо гравець не подолав рівень із босом, він може відвідати ігровий магазин та купити новий корабль. В іншому випадку, гравець продовжує свої пригоди.

10. Якщо гравець успішно пройшов всі додаткові складні рівні, він завершує гру з відчуттям задоволення і отримує відмінні оцінки на всіх рівнях.

11. Починається ініціалізація ігрових одиниць для всіх типів рівнів з додатковими бонусами +50% до характеристик супротивника. Це робить гру важчою з кожним пройденим рівнем, оскільки ворожі одиниці отримують додаткові підсилення.

На рисунку 2.10 зображена блок-схема алгоритму методу ускладнення гри.

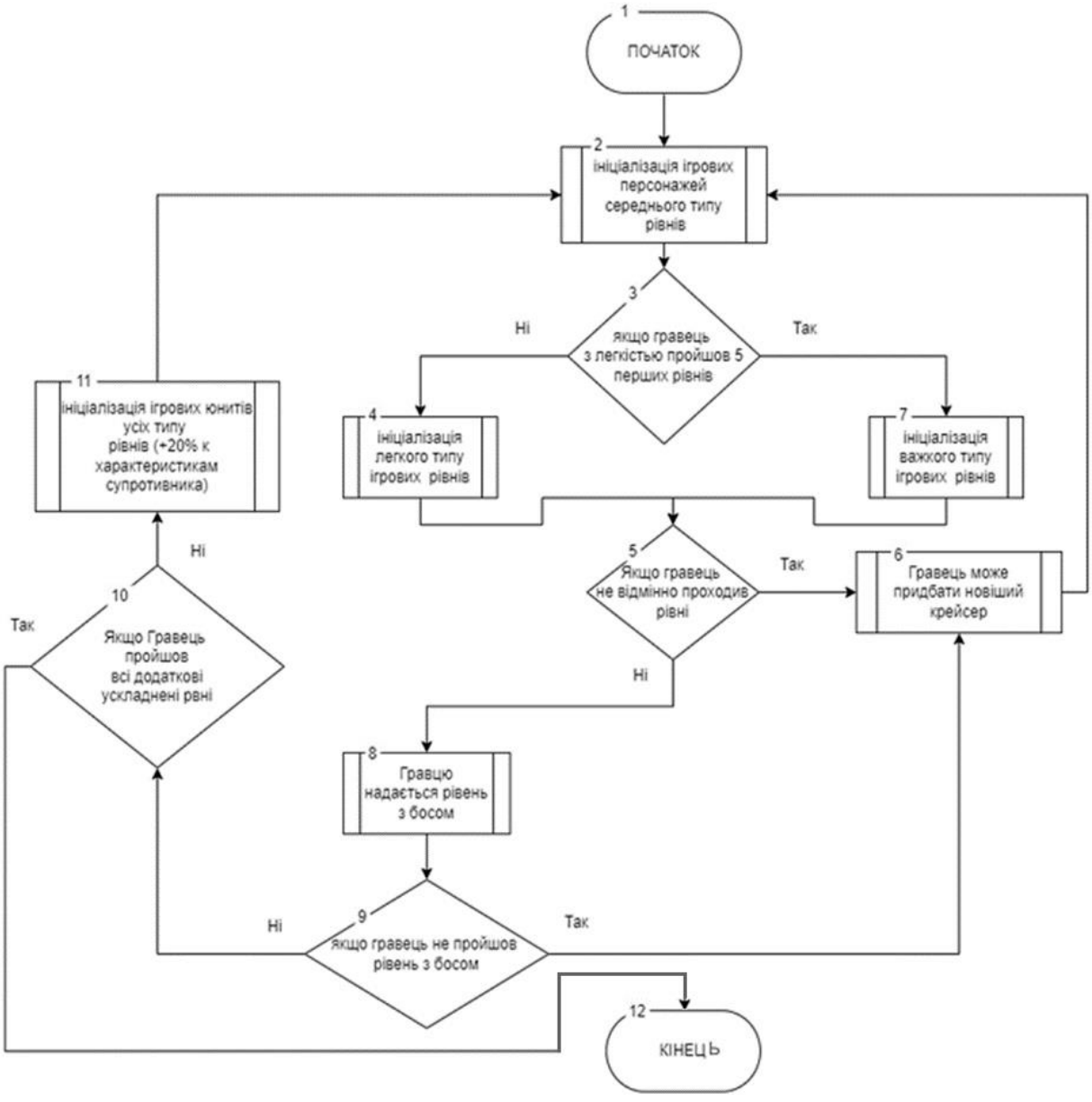

Рисунок 2.10 – Блок-схема алгоритму методу гри

В нашому підході супротивники автоматично нападають на головного героя, коли той опиняється в їхньому радіусі атаки. Гравець має ефективно знищувати

ворожі одиниці для отримання більш великої винагороди в кінці кожного етапу та активно збирати всі бонуси, які полегшують йому пройти гру.

Метод багаторівнева структура гри поданий на рисунку 2.11.

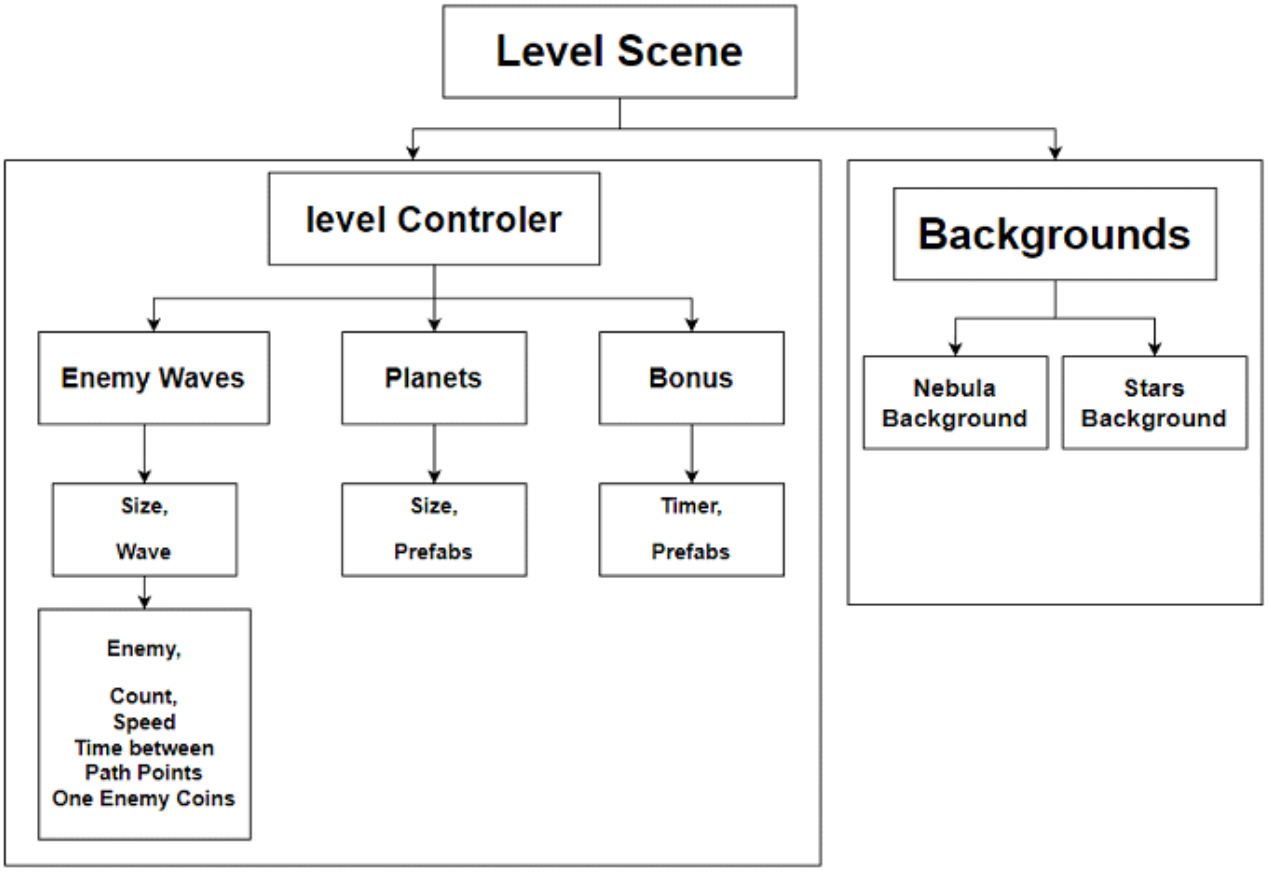

Рисунок 2.11 – Багаторівнева структура гри

Структура гри розкриває ігрову сцену "Level Scen", де контролер рівня "Level Controler" та фон гри "Background" змінюються відповідно до прогресу гравця.

Контролер рівня включає "Enemy Waves", "Planets" та "Bonus", кожен з них має свої власні особливості. "Enemy Waves" - це ворожі хвилі з визначеною кількістю юнітів, які рухаються по певній траєкторії та вступають у бій з гравцем. Ці хвилі поступово з'являються і зникають, якщо гравець не встигає їх знищити. Кожен тип ворожої одиниці має власні характеристики, такі як вигляд, рівень життя, швидкість та винагорода за їх знищення.

"Planets" відповідають за управління планетами, які слугують декоративним елементом фону гри. Це охоплює час їх появи та кількість "prefabs", що послідовно з'являються та зникають під час гри. "Bonus" додають додатковий функціонал для полегшення проходження рівня, дозволяючи космічному корабля підвищити рівень та отримати апгрейд.

"Background" - це візуальний фон гри, що змінюється в залежності від розташування та руху ігрових об'єктів, включаючи "Nebula Background" та "Stars Background".

Завдяки "Level Controler" легко налаштовується різноманіття нових видів ворожих одиниць та реалізуються складніші "Level Scen", надаючи захоплюючий шлях гравця, розширюючи геймплей та створюючи насичену візуальну атмосферу через "Background".

В створеній грі основний акцент робиться на внутрішній ігровій валюті, що використовується виключно в ігровому процесі. Гра не преслідує комерційних цілей і виступає як тренувальний інструмент для гравців різних вікових груп, сприяючи розвитку моторики та стратегічного мислення. У магазині гравцям відкриється можливість придбати покращений космічного корабля за допомогою накопиченої валюти, що забезпечить їм перевагу при зіткненні з модифікованими одиницями та босами..

Модель впровадження мобільної гри в жанрі "Action" відображає підтримку гравцеві космічного корабля через ігровий магазин, доступ до якого забезпечується з головного меню. У магазині кожен гравець отримає можливість придбати новий корабель із покращеннями, які перевершують його попередника. Властивості цих кораблів включають рівень життя, захисний щит, швидкість стрільби, дизайн зброї, вигляд ракет, їх швидкість та візуальний стиль космічного корабля.

На рисунку 2.12 наведено модель реалізації «магазину» гри.

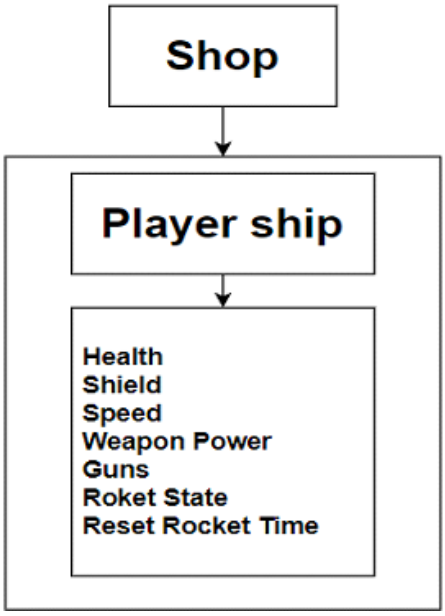

Рисунок 2.12 – Модель реалізації «магазину» гри

Процеси взаємодії у мобільній ігровій аплікації жанру "Action" втілюють алгоритм, що розпочинається зі завантаження головного меню, яке пропонує гравцеві вибрати подальші дії: "Почати гру", "Магазин", "Вихід" чи "Налаштування".

Компонент "Почати гру" ініціює перехід на сцену "Ігрові рівні" та відкриває можливість вибору рівня гри, який автоматично веде до активізації ігрової сцени.

На ігровій сцені відбувається геймплей, де гравець стикається з ворожими юнітами і повинен їх усіх знищити, а також долати всі труднощі для досягнення кінцевої точки. Після завершення сутички відбувається перехід на сцену "Результат гри", де гравець може перейти до наступного рівня чи повернутися до "Меню".

Компонент меню "Налаштування" надає можливість регулювання аудіофону гри, встановлюючи параметри музичного супроводу.

Компонент "Вихід" реалізує завершення функціонування мобільного додатку, що є заключним етапом його використання.

Блок-схема загального алгоритму роботи мобільної «Action» ігри зображена на рисунку 2.13.

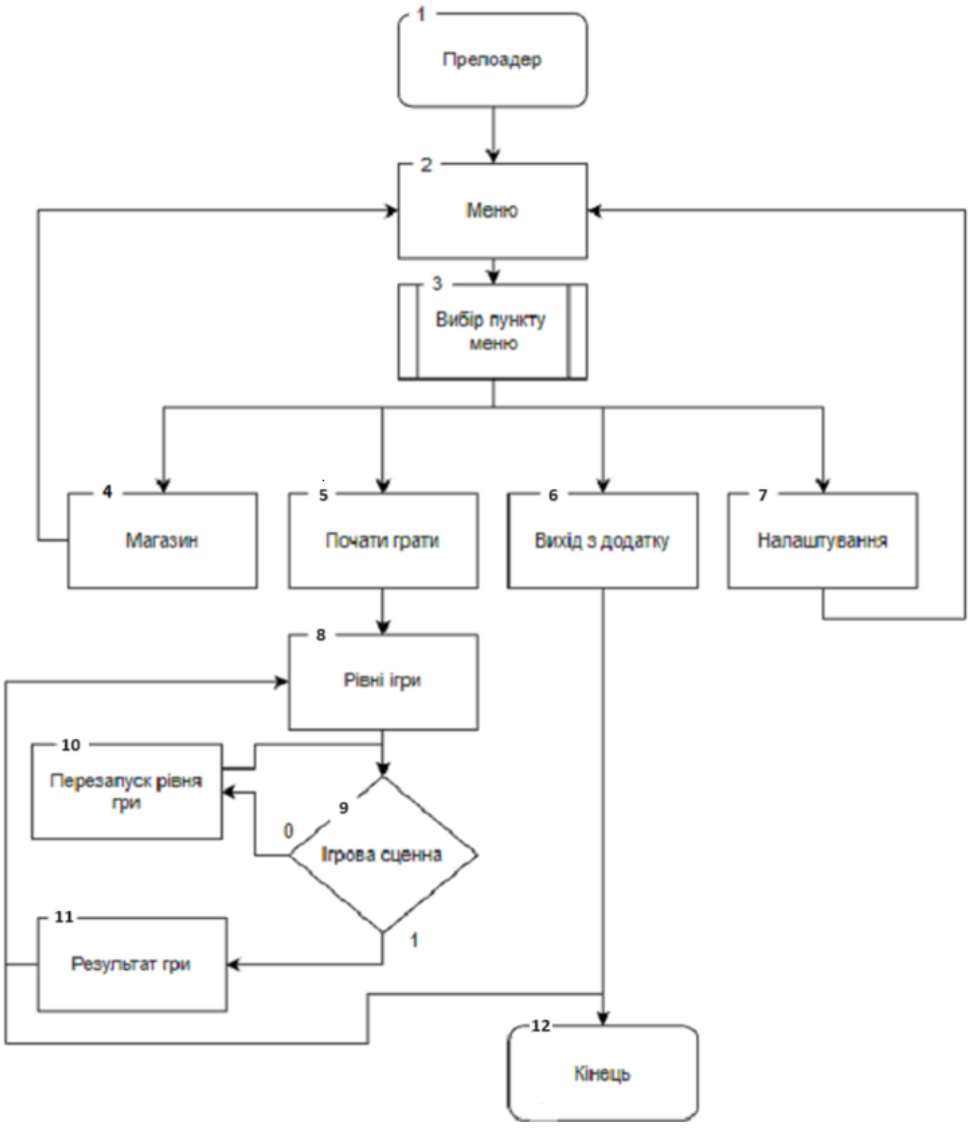

Рисунок 2.13 – Блок-схема загального алгоритму програми

2.5 Розробка Asset Pool Manager

У проектах Unity, які ви створюєте, зазвичай багато GameObjects. У багатьох випадках ваші GameObjects будуть копіями інших – у шутері від першої особи може бути багато копій одного і того ж ворожого персонажа, ті самі валуни, за якими можна ховатися, і ті самі дерева, зброя та предмети, які потрібно збирати.

Під час розробки цих GameObjects ви можете ввести зміни до всіх копій одного елемента. Ви навіть можете змінити всі дерева на кактуси! Якщо дерев багато, то така зміна потребує багато роботи. Однак замість того, щоб керувати багатьма копіями елементів, ви можете впорядкувати свої дубльовані GameObjects за допомогою збірних елементів (prefabs) [13].

Prefab – це asset, який є шаблоном GameObject. З prefabs можна створити кілька копій, які називаються екземплярами. Зміна prefab asset також спричиняє зміни всіх його екземплярів.

Нам для даної гри потрібно створити декілька шаблонів. Види шаблонів які використовуються в даній грі:

– снаряди які випускає наш корабель;

– снаряди які випускають наші вороги;

– створення самих ворогів;

– інтерактивні об’єкти на фоні.

Розберемо спосіб створення Prefab [14]. Для початку в робочий зоні програми Unity3D створимо приклад снаряду який потрібен для нашого корабля. Потім створюємо папку на «Prefabs». Відкриваємо папку так, щоб у правій панелі вікна проекту (у Two Column Layout) відображалася порожня папка У вікні Hierarchy перетягніть Enemy_Straight_Projetile у папку Prefabs. Новим asset у вашій папці Prefabs із синім значком куба є ваш prefab приклад Prefab (див. рисунок 2.14).

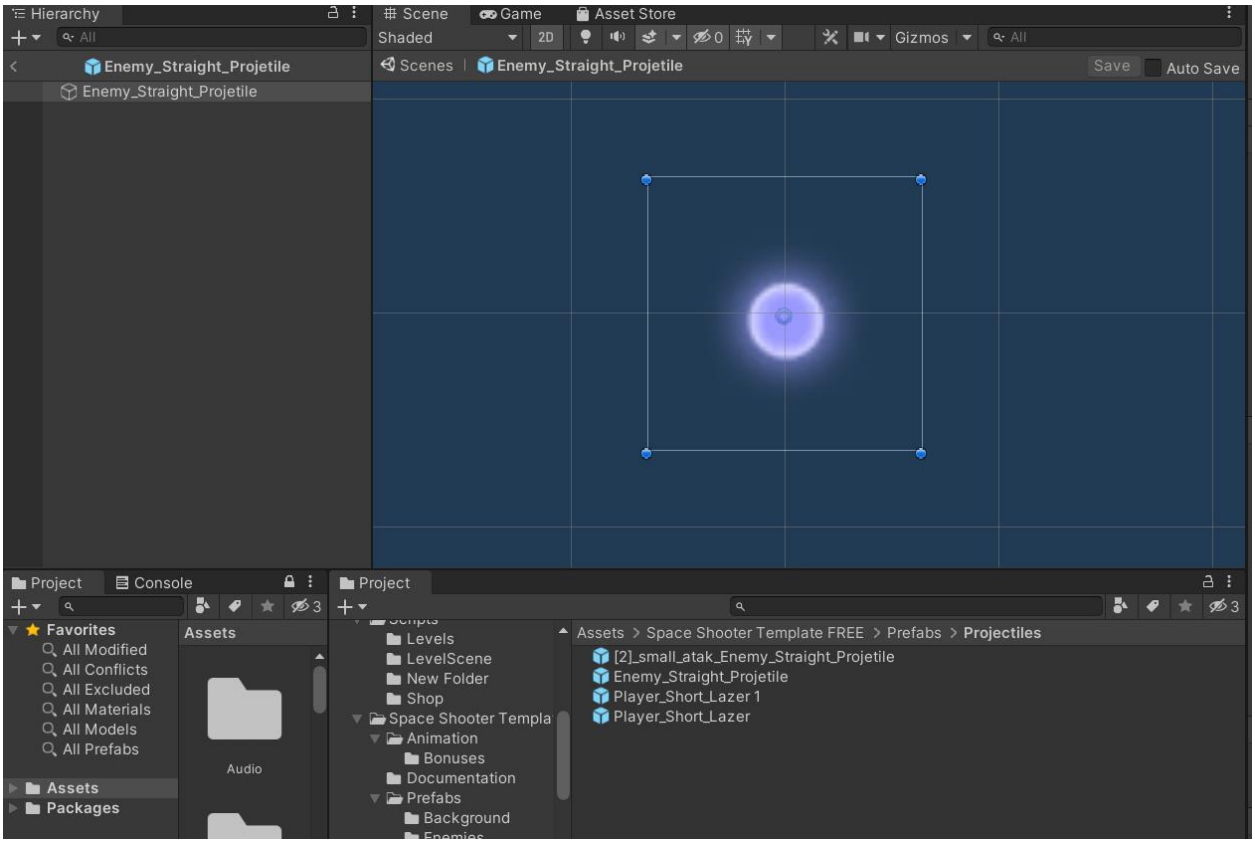

Рисунок 2.14 – Вигляд Prefab снаряду в Unity3D

Таким чином створюємо всі інші шаблони Prefab для наших пере використовуваних об’єктах (див. рисунок 2.15) та (див. рисунок 2.16).

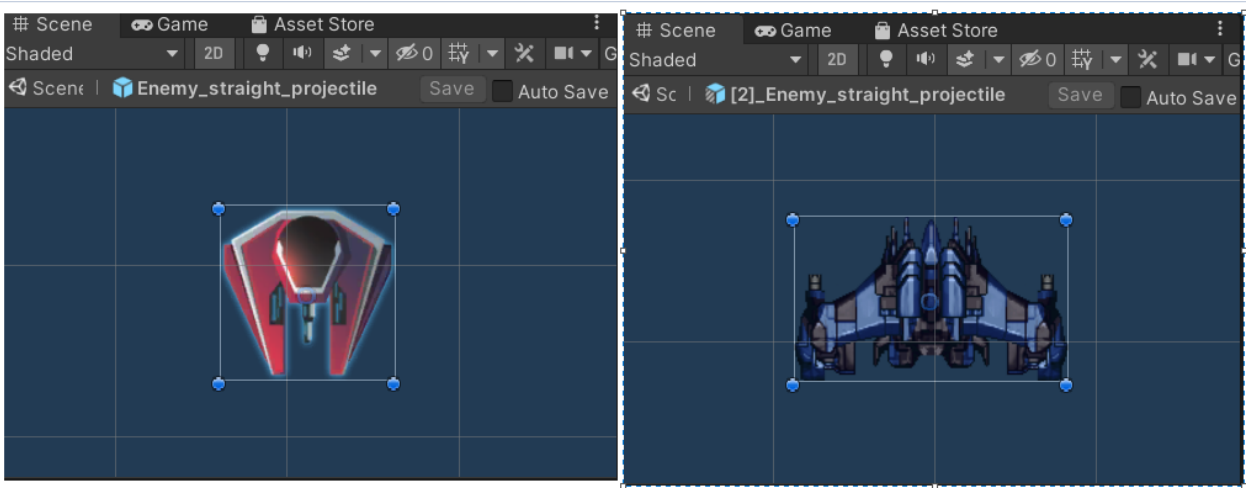

Рисунок 2.15 – Шаблони наших ворогів

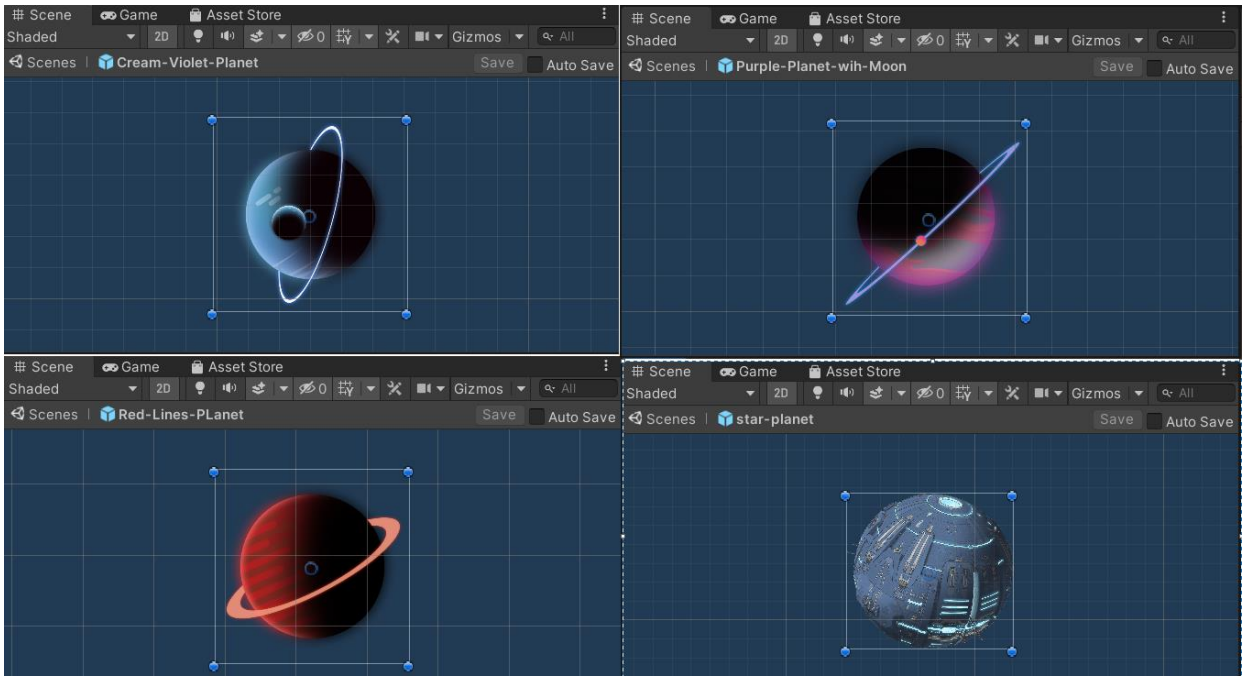

Рисунок 2.16 – Шаблони інтерактивних об’єктів

Тепер всі приготування для створення кодової частини зроблені, та приступаємо до моделювання та написання класів нашого Pool Manager.

Давайте визначимо деякі загальні поняття, які будемо використовувати протягом. Об’єкт, який запитує екземпляри з пулу об’єктів, буде позначено як

«споживач». Кожен екземпляр об’єкта, який можна об’єднати, збережений у Pool об’єктів, буде названо «примірником». Pool об’єктів, що представляє сховище екземплярів, буде позначатися як «пул».

Pool в основному використовуються для підвищення продуктивності. У деяких випадках пули значно покращують продуктивність, але, з іншого боку, це ускладнює тривалість життя екземплярів, оскільки екземпляри, отримані з пулу та повернуті назад, фактично не створюються та не знищуються в цей час, і, отже, пули, як і екземпляри, потребують обережності при реалізації [15].

Використовуємо шаблон пулу об’єктів переважно в ситуаціях, коли потрібно (повторно) породити багато об’єктів одного типу на сцені під час виконання:

– різні ефекти (ефекти частинок, звуки, анімовані об’єкти, кулі тощо);

– об’єкти в сцені використовується процедурним генератором (шаблони макета та їхні відповідні частини, такі як стіни, предмети колекціонування, вороги, перешкоди, пастки тощо).

Потрібно будьте надзвичайно обережними, коли працюєте з екземплярами, і завжди пам’ятайте про скидання стану екземпляра до стандартного, коли повертаєтеся до пулу, щоб уникнути неочікуваної поведінки.

Кожен Pool відрізняється унікальною назвою (рядковим представленням) і містить посилання на Prefab ігрового об’єкта, який використовується для створення екземплярів пулу [16]. Якщо ви не були достатньо уважними під час створення нових пулів у редакторі та використовували те саме ім’я пулу для кількох пулів, ви не можете бути впевнені, що об’єднані екземпляри на сцені будуть відповідного типу. Такі випадки можуть призвести до неочікуваних помилок виконання та порушити стабільність вашої гри. Замість використання рядкового представлення унікального імені пулу ви також можете використовувати змінну InstanceID, яка гарантовано буде унікальною для будь-якого компонента ігрового об’єкта.

Наша реалізація дозволяє встановлювати початкову кількість екземплярів пулу, створених під час ініціалізації, а також вирішувати, чи бути батьківськими для екземплярів під об’єктом гри в пул в ієрархії Unity3D.

Існує кілька можливостей для реалізації пулу об’єктів для зберігання екземплярів, наприклад, ви можете використовувати стек або чергу:

– стек базується на методі «Останній прийшов – першим вийшов» (LIFO), тобто коли споживач запитує примірник, пул повертає посилання на перший екземпляр у стеку, і коли екземпляр повертається назад до пулу, він автоматично повертається назад у першу позицію черги стека;

– заснований на методі «першим увійшов, першим вийшов» (FIFO), отже, коли споживач запитує, наприклад, пул повертає посилання на перший примірник у черзі, і коли insatnce повертається назад до пулу, він зберігається в останньому полі черги.

Якщо після повернення до пулу не потрібно виконувати багаточасові процедури очищення екземплярів, достатньо використовувати стек [17]. В іншому випадку реалізація пулу за допомогою черги є набагато надійнішою, оскільки екземпляри мають більше часу для виконання дій очищення.

Було формалізовані дозволені операції з примірниками, що містяться в різних pooling. Кожен екземпляр у пулі має реалізовувати інтерфейс, щоб гарантувати, що кожен екземпляр пулу буде належним чином ініціалізований, коли вони надсилають запит екземпляру з пулу, і що будь-який екземпляр може мати власну процедуру очищення (за потреби).

Основні функціональні можливості шаблону проектування пулу об’єктів можна підсумувати в наступних кроках і описано на (див. рисунок 2.17):

1. пул створює та ініціалізує всі екземпляри

2. споживач отримує екземпляри з пулу за потреби (якщо немає жодного доступного пулу екземплярів створюється новий)

3. якщо екземпляр більше не потрібен, він повертається назад до пулу (за потреби виконується процедура очищення) [18].

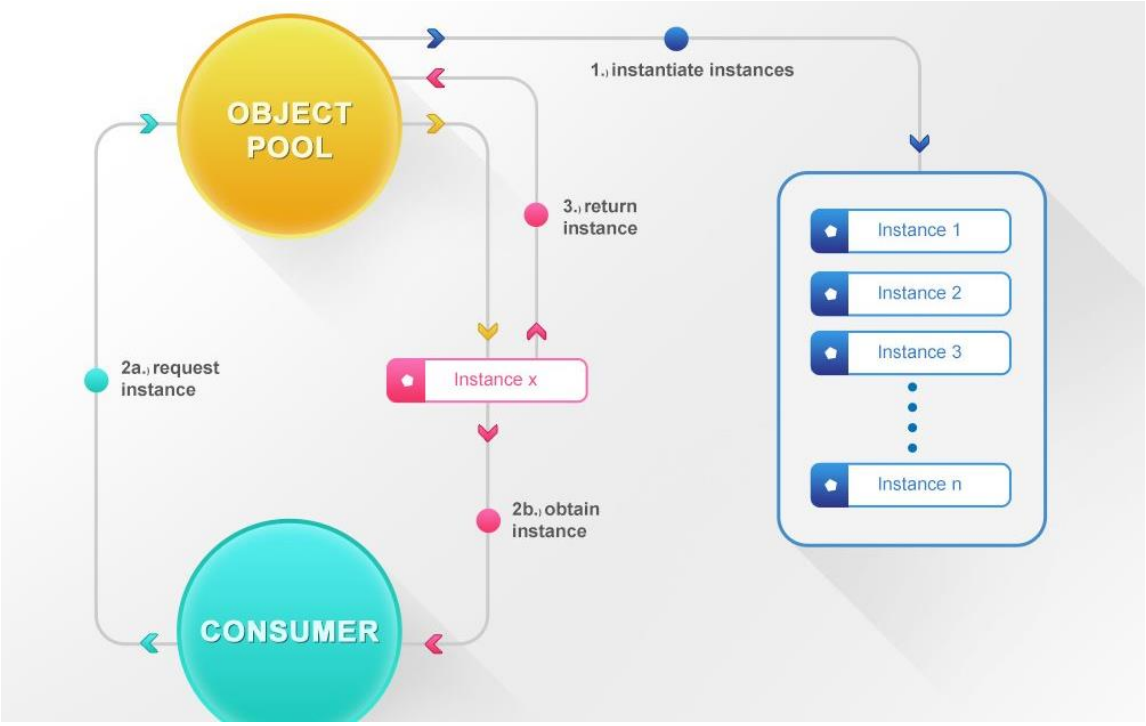

Рисунок 2.17 – Модель роботи Pool Manager

Пул створює попередньо визначену кількість примірників і зберігає посилання на примірники в черзі або стеку. Будь-який конкретний пул може одночасно обслуговувати більше одного споживача. Після ініціалізації пулу споживачі можуть довільно запитувати екземпляри з пулу відповідно до своїх потреб. Якщо пул отримує запит і екземпляри закінчуються, він негайно створює новий екземпляр і повертає його посилання споживачу. Як тільки екземпляр нікуди не потрібен споживачеві, він повертається назад у пул (при необхідності виконується процедура очищення) [19].

Після написання скриптів збираєм через Asset Pool Manager.

2.6 Висновок

У другому розділи було обрано використати принципи Unity Package для створення нашого Assets. Такий спосіб допоможе переносити нашу реалізацію Pool Manager у проект значно швидше. Це свого роду пресет скриптів якій можна перенести у любий проект Unity та почити його використовувати.

Описано процес створення класу Pool<T> для управління пулом об'єктів у середовищі Unity. Пояснено кроки створення нового сценарію, а також наведено

приклад створення класу, який представляє контейнер для об'єкта в пулі. В класі Pool<T> детально розглянуті основні поля та методи, такі як Consume та Release у класі PoolSlot<T>, а також Count, CountUsedItems, Warm, CreateContainer, GetItem та ReleaseItem у класі Pool<T>. Пояснено структуру класів, їхні взаємозв'язки та функціональні можливості.

Представлено клас Pool Manager, який управляє pooling objects в Unity. Він є Singleton<T>, гарантуючи наявність лише одного екземпляра протягом гри. Основні параметри включають LogStatus, Root, PrefabLookup, InstanceLookup та Dirty. Представлені методи включають WarmPool для створення пулу, ReleaseObject для звільнення об'єкта та SpawnObject для включення об'єкта в пул.

Розроблено метод ігрових рушіїв з використанням Pool програмного продукту, який використовує різні шаблони побудовані розробником, та «Level Controler», за допомогою якого створюються нові «Level Scen».

Розроблено модель роботи ігрової системи та модель реалізації ігрового магазину. Розроблено алгоритм роботи мобільного ігрового додатку та описано процес його роботи.

Створено Asset Pool Manager для Unity. Показано важливість використання Prefab для шаблонів об'єктів у грі. Розглянуто реалізацію класів Pool та Pool Manager для керування pooling objects. Привернуто увагу на важливі аспекти, такі як вибір стеку чи черги для пулу, а також правильне очищення та повернення екземплярів. Підкреслено, що використання пулів може покращити продуктивність гри, але вимагає обережності при роботі з екземплярами.

Також у даному розділі описані блок схеми та діаграми класів нашої майбутньої розробки Pool Manager. У цьому розділі було використано принципи до організації коду, як Singleton патерн проектування, але й розглянути найбільш комфортні та правильні підходи до обробки подій та поговорити про зручність коду загалом. Для зручності звертання до класів Pool Manager з любої точки коду.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ІГРОВОГО ДОДАТКУ

3.1 Обґрунтований вибір ігрового рушія

Перед початком нового програмного проекту необхідно вирішити, який ігровий рушій будемо використовувати. Розглянемо три найбільш використовувані ігрові рушій для розробки додатків. Результати порівняння ігрових рушіїв приведені у таблиці 3.1.

Unity – це не лише інтегроване середовище розробки, а й потужний графічний двигун, спрямований на створення ігор та інтерактивних додатків. Оснований на мові програмування C#, Unity приваблює розробників своєю легкістю використання, кросплатформенністю та великим набором розширень. Підтримуючи як 2D, так і 3D графіку, Unity також знаходить застосування в області віртуальної реальності та симуляцій, завдяки активній та розширеній спільноті розробників.

Unreal Engine – це високопотужний графічний двигун і інтегроване середовище розробки, створене компанією Epic Games для розробки ігор та інтерактивних додатків. Цей інструмент відомий своєю захоплюючою графікою та розширеними можливостями. З використанням мови програмування C++, Unreal Engine надає гнучкість у створенні складних ігрових сценаріїв. Зараз він широко використовується в галузі розробки ігор та віртуальної реальності завдяки високоякісній графіці та розширеній функціональності..

MonoGame – інструмент з відкритим вихідним кодом, який надає можливість розробникам створювати захоплюючі ігри з використанням мови програмування C#. Розроблений на основі XNA, MonoGame дозволяє створювати кросплатформенні ігри, які працюють на різних операційних системах, таких як Windows, macOS, Linux, Android, iOS та інші. Зручна бібліотека створена для надання розробникам гнучкості у створенні ігор, дозволяючи їм зосередитися на творчому процесі, уникаючи зайвих труднощів.

Шкала оцінювання критеріїв мов програмування:

– не відповідність даному критерію – 0;

– відповідність даному критерію – 1.

Спільний показник критеріїв середовище розробки ігрових рушіїв (СПКСРІР) – це середнє значення всіх необхідних критеріїв розробки ігрових рушіїв для розробки додатку. СПКСРІР обчислюється за формулою 3.3.

$$СПКСРІР = (П_1+П_2+П_3+П_4+П_5+П6)/6 \quad (3.3)$$

де $П_1$ – простий інтерфейс;

$П_2$ – підтримує написання скриптів на C#;

$П_3$ – масштабний розмір документації;

$П_4$ – магазин готових assets та plugins;

$П_5$ – розробка мобільних ігор;

$П_6$ – кросплатформеність.

Таблиця 3.1– Порівняння мов програмування

Параметри	Unity	Unreal Engine	MonoGame
Простий інтерфейс	1	1	1
Підтримує написання скриптів на C#	1	1	1
Масштабний розмір документації	1	1	0
Магазин готових assets та plugins	1	0	0
Розробка мобільних ігор	1	0	0
Кросплатформеність	1	0	0
Сума показників	6	3	2

Результати розрахунку СПКСРІР:

– СПКСРІР Unity дорівнює 1;

– СПКСРІР Unreal Engine дорівнює 0,5;

– СПКСРІР MonoGame дорівнює 0,33

В ході аналізу середовище розробки ігрових рушіїв було вирішено розробляти додаток у Unity [20], так як даний ігровий рушій має найвищий СПКСРІР.

3.2 Обґрунтований вибір мови програмування

У наш час існує велика кількість мов програмування. Розглянемо три найбільш мови для розробки «desktop» додатків. Порівняння мов програмування приведено у таблиці 3.2.

Мова програмування C# – це продукт компанії Microsoft, і вона вражає своєю об'єктно-орієнтованою структурою та численними корисними властивостями. Простота використання, підтримка об'єктної орієнтації, система типізації, автоматичне вивільнення пам'яті ("збірка сміття"), можливість сумісності версій роблять її привабливим вибором для розробників.

Мова програмування C++ володіє як імперативними, так і об'єктно-орієнтованими функціями, що робить її відомою як мову середнього рівня. Здатність комбінувати об'єктно-орієнтований підхід, універсальність та функціональні можливості із здатністю до низькорівневих маніпуляцій з пам'яттю робить її потужним інструментом для програмістів.

Python – це високорівнева мова програмування загального призначення, створена для підвищення продуктивності розробників та полегшення читання коду. Мова відзначається динамічною типізацією, автоматичним управлінням пам'яттю та використанням зручних високорівневих структур даних, таких як словники, списки та кортежі.

Шкала оцінювання критеріїв мов програмування:

- не відповідність даному критерію – 0;
- відповідність даному критерію – 1.

Спільний показник критеріїв мов програмування (СПКМП) – це середнє значення всіх необхідних критеріїв мов програмування для розробки додатку. СПКМП обчислюється за формулою 3.2.

$$СПКМП=(П_1+П_2+П_3+П_4+П_5+П6)/6 \quad (3.2)$$

де $П_1$ – наявність бібліотек для роботи з графікою;

$П_2$ – можливість роботи з динамічною пам'яттю;

$П_3$ – наявність збірника;

$П_4$ – можливість об'єктно-орієнтованого програмування;

$П_5$ – наявність бібліотек для роботи з операційними системами Windows;

$П_6$ – більш ознайомлений з мовою програмування.

Таблиця 3.2– Порівняння мов програмування

Параметри	C#	C++	Python
Наявність бібліотек для роботи з графікою	1	1	1
Можливість роботи з динамічною пам'яттю	1	1	1
Наявність збірника	1	0	1
Можливість об'єктно-орієнтованого програмування	1	1	1
Наявність бібліотек для роботи з операційними системами Windows	1	1	1
Більш ознайомлений з мовою програмування	1	1	0
Сума показників	6	5	5

Результати розрахунку СПКМП мов програмування:

– СПКМП мови програмування C# дорівнює 1;

– СПКМП мови програмування C++ дорівнює 0,8;

– СПКМП мови програмування Python дорівнює 0,8

В ході аналізу мов програмування було вирішено розробляти додаток на мові C# [21], так як дана мова має найвищий СПКМП.

3.3 Обґрунтований вибір середовища розробки

Інтегроване середовище розробки (IDE) – це програмний набір, який об'єднує основні інструменти, користувач записує та редагує вихідний код у редакторі коду. Компілятор переводить вихідний код на читабельну мову, яку можна виконати для комп'ютера. І debugger тестує програмне забезпечення для вирішення будь-яких проблем або помилок. Порівняння IDE наведено в таблиці 3.3.

Microsoft Visual Studio 2019 – інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів. Даний продукт дозволяє розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows.

MonoDevelop – це інтегроване середовище розробки з відкритим вихідним кодом для Linux, macOS, і Windows. MonoDevelop об'єднує функції, аналогічні NetBeans і Microsoft Visual Studio, такі як автоматичне завершення коду, управління вихідним кодом, графічний інтерфейс користувача (GUI) і веб-дизайнер.

Sublime Text – це умовно-безкоштовний крос платформний редактор вихідного коду з інтерфейсом прикладного програмування (API) Python. Він спочатку підтримує безліч мов програмування і мов розмітки, і функції можуть бути додані користувачами з плагінами.

Шкала оцінювання критеріїв IDE:

- не відповідність даному критерію – 0;
- відповідність даному критерію – 1.

Спільний показник критеріїв інтегрованого середовища розробки (СПКІСР) – це середнє значення всіх необхідних критеріїв IDE для розробки додатку. СПКІСР обчислюється за формулою 3.1.

$$СПКІСР = (П1+П2+П3+П4+П5+П6)/6 \quad (3.1)$$

де $П_1$ – безкоштовна ліцензія;

$П_2$ – можливість розробки GUI;

$П_3$ – наявність компілятора;

$П_4$ – створення крос-платформових додатків.

$П_5$ – наявність засобів для роботи з графікою;

$П_6$ – більш ознайомлений з середовищем розробки.

Таблиця 3.3– Порівняння IDE

Параметри	Microsoft Visual Studio Community 2019	MonoDevelop	Sublime Text
Безкоштовна ліцензія	1	1	1
Можливість розробки GUI	1	1	0
Наявність компілятора	1	1	0
Створення кросплатформиних додатків	1	1	0
Наявність засобів для роботи з графікою	1	0	0
Більш ознайомлений з середовищем розробки	1	0	0
Сума показників	6	4	1

Результати розрахунку СПКІСР інтегрованих середовищ розробки:

– СПКІСР інтегрованого середовища розробки «Microsoft Visual Studio Community 2019» дорівнює 1;

– СПКІСР інтегрованого середовища розробки «MonoDevelop» дорівнює 0,8;

– СПКІСР інтегрованого середовища розробки «Sublime Text» дорівнює 0,2.

В ході аналізу IDE було вирішено розробляти додаток в Microsoft Visual Studio Community 2019 [22], так як воно має найвищий СПКІСР.

3.4 Вибір використовуваних API

API – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено – це набір чітко визначених методів для взаємодії різних компонентів [23]. API надає розробнику засоби для швидкої розробки програмного забезпечення.

Для реалізації необхідної функціональності для розробки гри було сформовано список використаних API:

– Scripting;
– Profiler;
– Analytics Library;
– In App Purchasing;
– TextMeshPro;
– 2D Sprite.

3.5 Вибір використовуваних бібліотек

Unity як бібліотека призначена для користувачів-спеціалістів, які використовують власні технології платформи, такі як Java/Android, Objective C/iOS або Windows Win32/UWP, і хочуть включити функції на базі Unity у свої ігри чи програми.

Для початку передбачається, маємо досвід розробки для технологій рідної платформи, таких як Java/Android, Objective C/iOS або Windows Win32/UWP, і знайомі зі структурою проекту, мовними функціями та конкретними параметрами конфігурації платформи (наприклад дозволи користувача) [24].

Використовувати Unity як бібліотеку в інших програмах, інтегрувавши свій вміст і компоненти середовища виконання Unity у проект рідної платформи. Це дає

змогу вставляти вміст, який використовує 3D або 2D візуалізацію в реальному часі.

Досвід, взаємодія з 3D-моделями та 2D-міні-іграми. Бібліотека середовища виконання Unity розкриває способи керування завантаженням, активацією та вивантаженням у рідній програмі [25].

Для реалізації необхідної функціональності для розробки ігрових рушіїв було сформовано список модулів [26], які будуть містити структури та функції для:

– System.Collections (Містить інтерфейси та класи, які визначають різноманітні колекції об’єктів, такі як списки, черги, бітові масиви, хеш-таблиці та словники.);

– System.Collections.Generic (Містить інтерфейси та класи, що визначають універсальні колекції, які дозволяють користувачам створювати строго типізовані колекції, що забезпечують підвищену продуктивність та безпеку типів у порівнянні з неуніверсальними строго типізованими колекціями.);

– UnityEngine.Profiling (Містить класи тестування у свої сценарії за допомогою Profiler.BeginSample і Profiler.EndSample);

– UnityEngine.UI (Містить всі елементи для графічного відображення на холсті.);

– DG.Tweening (Містить інтерфейси та класи для руху чи анімації об’єктів);

– UnityEngine.Audio (Містить класи роботи з звуками)

3.6 Висновок

У третьому розділі було проведено аналіз різних мов програмування для розробки методів оптимізації ігрових рушіїв Action ігор мобільних додатків, і середовищ програмування. У якості мови програмування було обрано мову C# для розробки головних модулів програмного додатку. Ця мова є об’єктно-орієнтована, забезпечують достатньо високу швидкодію програмного продукту та дозволяють напряму взаємодіяти з системними функціями. Дозволяє не втрачати продуктивність роботи програми.

Також було прийнято рішення використовувати Microsoft Visual Studio

Community 2019 в якості IDE, адже дане середовище розробки відповідає всім необхідним вимогам, а головне дозволяє писати код одночасно на мові C#. Було сформовано список використаних API. Реалізовано необхідні функціональності для розробки ігрових рушіїв було сформовано список модулів, які будуть містити структури та функції.

4 ТЕСТУВАННЯ ТА АНАЛІЗ ASSET POOL MANAGER

4.1 Тестування Pool Manager за допомогою Unity Profiler

Unity Profiler – це інструмент, який можна використовувати для отримання інформації про продуктивність вашої програми. Можемо підключити його до пристроїв у вашій мережі або пристроїв, підключених до вашого комп'ютера, щоб перевірити, як ваша програма працює на запланованій платформі випуску. Також можем запустити його в редакторі, щоб отримати огляд розподілу ресурсів під час розробки програми.

Profiler збирає та відображає дані про продуктивність вашої програми в таких областях, як процесор, пам'ять, рендерер і аудіо. Це корисний інструмент для визначення областей для покращення продуктивності вашої програми та повторення цих областей. Ви можете точно визначити, як ваш код, активи, сцена налаштування, камера [27].

Налаштування візуалізації та збірки впливають на продуктивність програми. Він відображає результати в серії діаграм, щоб ви могли візуалізувати, де відбуваються сплески продуктивності вашої програми.

Окрім використання вбудованого Unity Profiler, ви можете використовувати низькорівневий плагін Profiler API для експорту даних профілювання до сторонніх інструментів профілювання, а також пакет Profiling Core для налаштування аналізу профілювання. Ви також можете додати до свого проекту потужні інструменти профілювання, такі як Memory Profiler і Profile Analyzer , щоб детальніше аналізувати дані про продуктивність.

Щоб отримати доступ до вікна Profiler, перейдіть до меню: Window > Analysis > Profiler. Щоб отримати докладний огляд вікна, перегляньте документацію вікна Profiler .

Щоб порівняти зміни треба спочатку поставити мітки, щоб вони відобразились у вікні Profiler.

Profiler відображає зразок у поданнях ієрархії та часової шкали. Зразок вкладено під події або функціональні виклики, які призводять до виконання

зразкового коду. Наприклад, зразок, розміщений в Оновленні, відображається під Update.ScriptRunBehaviourUpdate ієрархією Profiler та часовою шкалою. Якщо ви надаєте targetObject, ви можете клацнути зразок на часовій шкалі Profiler, щоб вибрати цей об’єкт у редакторі (під час профілювання в режимі відтворення редактора).

Profiler.BeginSample умовно компілюється за допомогою ConditionalAttribute. Таким чином, він матиме нульові накладні витрати, коли його розгортають у нероздробній збірці. Алокація пам’яті до використання Pool (див. рисунок 4.1) [28].

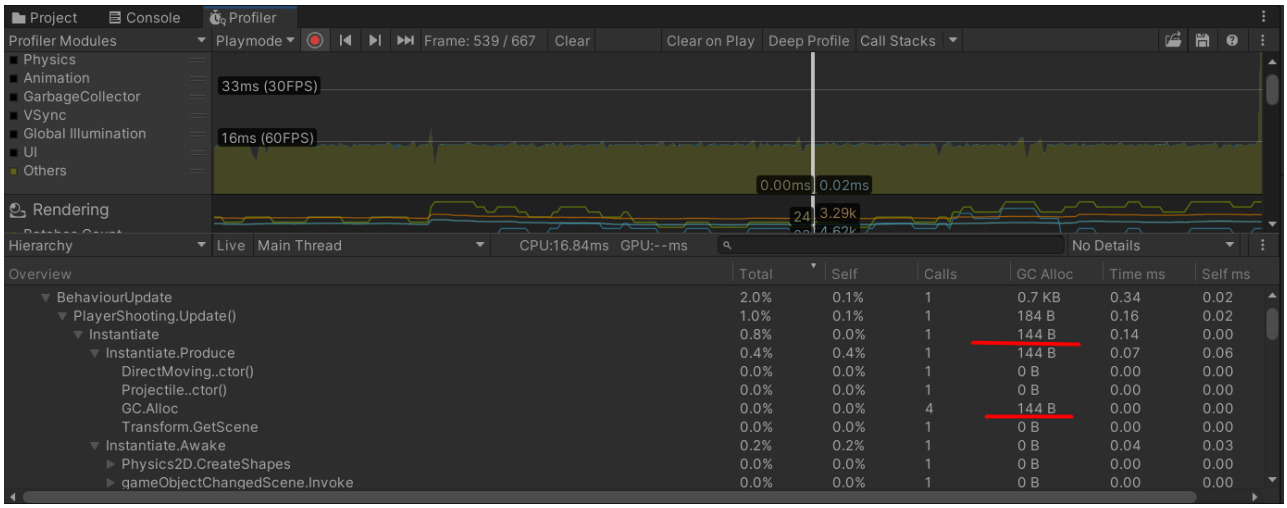

Рисунок 4.1 – Алокація пам’яті до використання Pool

Алокація пам’яті при використанні Pool Manager (див. рисунок 4.2).

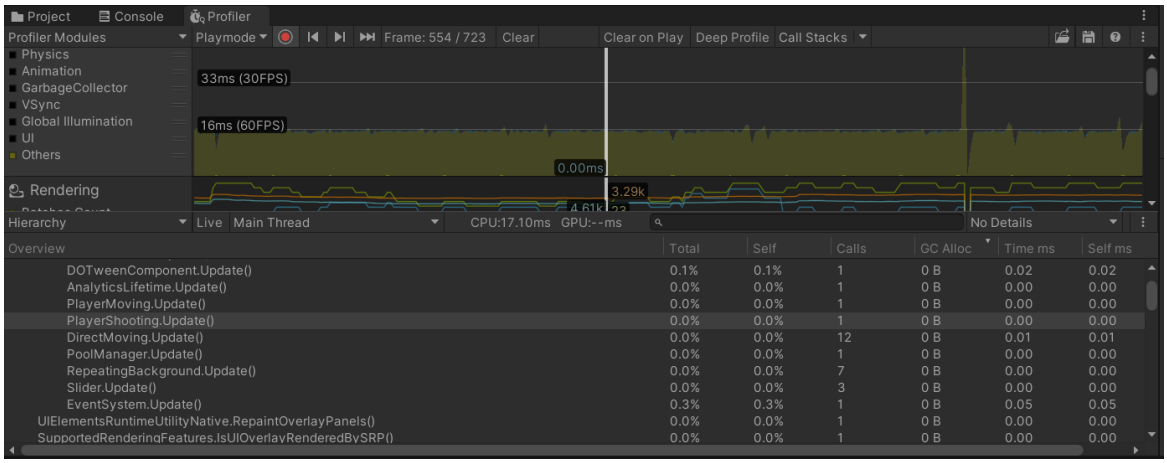

Рисунок 4.2 – Алокація пам’яті при використанні Pool Manager

4.2 Внесення реалізацій пулу у ігровий додаток

Для того щоб працювати з Pool Manager потрібно для початку імпортувати його в наш проект [29]. Є декілька варіантів внесення файлу форматом .unitypackage у проект:

1. Відкрити потрібний проект та перенести файл до робочого простору Project.
2. Відкрити проект потім двічи натиснути на ЛКМ по файлу.
3. Відкрити проект та вибрати потрібний файл за такими шляхом Assets/ImportPackage/CustomPackage.

Після того як виконали один з варіантів у нас з’явиться таке вікно (див. рисунок 4.3). В якому спостерігаємо які файли та данні додаються у наш проект.

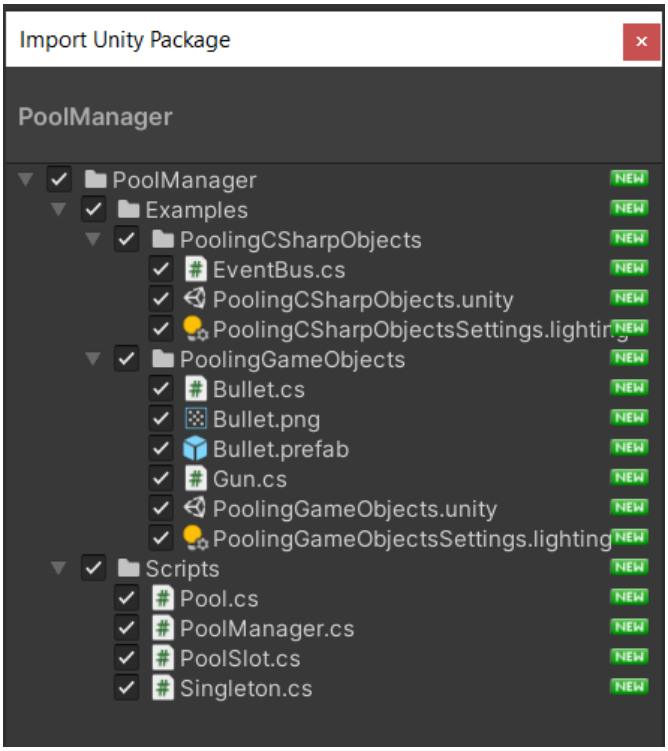

Рисунок 4.3 – Вікно Import Unity Package

Після експорту Asset Pool Manager потрібно визначити місця де потрібно використовувати в нашому проекті. Визначимо ці місця:

– стрільба нашого гравця клас PlayerShooting;
– створення ворогів клас Wave;

– стрільба ворогів клас Enemy та BossShooting.

Пишемо декілька нових метів в PlayerShooting в якому будемо описувати нові методи спавну снарядів для пушки:

SpawnPoolObjects – в цьому методі створюємо набір ініціалізованих об’єктів для цього використовуємо метод Pool Manager.WarmPool(parent, 50), де parent – це наш об’єкт а 50 – це кількість клонів.

Метод Pool Manager.WarmPool(GameObject, int) відповідає за ініціалізацію об’єктів та приймає в себе такі параметри як GameObject – це підготовлений нами Prefab тобто об’єкт клони яких потрібно створити. Другий параметр має тип integer і використовується для кількості створення початкових клонів.

SpawnShot – де беремо об’єкт по типу(T) з Pool Manager. Використовуємо метод Pool Manager.SpawnObject(obj), де obj – це parent.

Метод Pool Manager.SpawnObject(obj) відповідає за отримання об’єкту з Pool .якщо в Pool Manager немає об’єкту потрібного типу то він створюється і віддається.

В методі CreateLazerShot() визиваємо метод SpawnShot та передаєм потрібні данні по розташуванню та повороту об’єкта.

Повернення снаряду знову у Pool описуємо вже в самому Prefab снаряді.

В класі Wave в методі Start() викликаємо наш метод створення ворогів, передаєм потрібний Prefab в метод Pool Manager.WarmPool(enemy, int) та кількість 50. В старті запускається також Corutine в якій через певні проміжки часу які описані в налаштуваннях Wave визиваємо потрібних ворогів викликавши Pool Manager.SpawnObject(enemy, enemy.transform.position,Quaternion.identity);
Робимо такі операцій і в інших класах Enemy та BossShooting.

4.3 Тестування ігрових рушіїв додатку

Для демонстрації високих можливостей нашого Pool Manager, розробили та тестуватимемо творчого процесу гру у жанрі космічних пригод. Для даного ініціативного проекту, найкращою опцією для набору текстур виявилася безкоштовна версія, що доступна у Asset Store.

Кожен етап має свою вагу і важливість, вдавалися до вибору текстур з великою відповідальністю, забезпечуючи гру красивим та насиченим візуальним зображенням. Вибір безкоштовних текстур з Asset Store виявився оптимальним рішенням, дозволяючи зекономити ресурси для інших складових гри.

Створені різноманітні сцени, такі як ««Головне Меню» (MainMenu), «Сцена Рівня» (LevelScene) і «Ігрова Сцена» (GameScene), функціонально відповідають своїм призначенням.

При завантаженні через інтерфейс рівнів важкості «LevelScene» зображений на (див. рисунок 4.4).

Рисунок 4.4 – Інтерфейс рівнів важкості «LevelScene»

При натисканні на основну ігрову сцену, гравець миттєво поринає у світ «GameScene», пов'язаний із вибраним рівнем.

Процес ініціації нової хвилі ворожих сил, як і відтворення наступної хвилі через певний час, автоматично налагоджує гравець, причому попередню хвилю анігілює для зменшення навантаження на геймплейний пристрій.

Цей сценарій відповідає за генерацію ворожих хвиль, де кожна хвиля має унікальні характеристики, такі як кількість ворогів, їх швидкість руху, і часовий інтервал між їхніми появами. Крім того, встановлюється їхній шлях руху та режим стрільби з метою зробити гру більш високоінтенсивною.

З метою полегшення прогресу в грі, введено додаткові об'єкти, які мають функцію підняти рівень космічного корабля, що відзначається покращенням його озброєння та активує бонус «levelUp».

Розглядаючи на безмежні простори космосу, виявилось недостатньо простої фонової картини зірок. Тому було реалізовано динамічний фон, який плавно змінюється при русі вгору, створюючи враження переміщення гравця у просторі. Така локація космосу також доповнена візуальною привабливістю за рахунок присутності планет. Ігровий процес сцени «GameScene» показано на (див. рисунок 4.5).

Рисунок 4.5 – Ігровий процес

На рисунку 4.6 продемонстровано процес появи нової ворожої хвилі «Enemy Waves» в ігровій сцені.

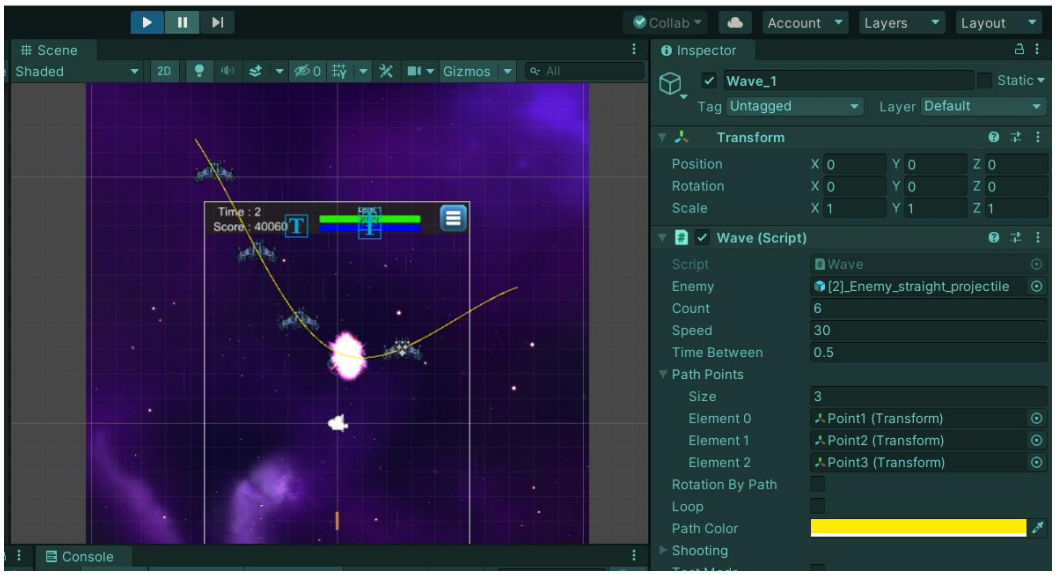

Рисунок 4.6 – Процес появи нової ворожої хвилі «Enemy Waves»

На рисунку 4.7, представлено геймплейний елемент - бонус "levelUp" та його вплив на ігрового персонажа. Спостерігатимемо що кількість випущених лазерних променів перевищує одиничний потік. Це вказує на те, що на етапі початку гри гравець має лише один лазерний промін, але після отримання бонусу "levelUp" ця можливість значно розширюється. Це дозволяє гравцеві ефективніше подолати ворожі сили та спростити процес проходження гри.

Рисунок 4.7 – Представлено ігровий об’єкт - бонус «levelUp»

На рисунку 4.8 передбачено дві з чотирьох планет, які відіграють ключову роль у візуальному представленні руху гравця вгору. Екзопланети активно взаємодіють із гравітаційним полем, з'являючись та спускаючись вниз, створюючи ілюзію руху в обраному напрямку. Усі об'єкти, які виходять за межі камери, автоматично вилучаються для оптимізації роботи пристрою, на якому відбувається ігровий процес.

Рисунок 4.8 – Представлено ігрові об’єкти планети

Для додаткового різнобарв’я геймплею було створено магазин. Реалізуючи цю концепцію в програмному рушії Unity, була розроблена графічна сцена під назвою "ShopScene", де програмний код розподілений між такими ключовими компонентами, як "CoinManager", "DataInfo", "SelectButton" і "ShopManager".

Кожен з цих компонентів відповідає за власний функціонал і реалізацію внутрішньо-гравецького магазину.

"CoinManager" бере на себе управління грошовою валютою гри, яку можна отримати, знищуючи ворожих одиниць. Кожен вид одиниць приносить певну кількість монет гравцеві. В ігровій сцені "ShopScene" гравець може витрачати цю валюту на придбання вдосконалених космічних кораблів для подальшого проходження високих рівнів та боротьби з босами. Специфікації кожного корабля включають в себе підвищений рівень життя, збільшений захисний щит, вищу швидкість стрільби, новий вигляд зброї, оновлений дизайн ракет і покращену швидкість цих ракет. Даний скрипт забезпечує вірне відстеження кількості монет гравця під час купівлі нових космічних кораблів.

"DataInfo" є визначальним скриптом, відповідальним за інформацію, що дозволяє створювати різноманітні ігрові космічні кораблі з різними характеристиками, забезпечуючи більш цікавий геймплей для гравця. Характеристики кожного корабля включають такі параметри:

- унікальний ідентифікатор корабля;
- унікальне зображення корабля;
- статус придбання корабля гравцем.
- назва корабля;
- кількість захисного шару корабля;
- кількість життя;
- рівень атаки;
- рівень швидкості вистрілів;
- ціна корабля.

"SelectButton" відповідає за перевірку та зміну тексту на кнопці для придбання, вибору та обрання конкретного космічного корабля. Кнопка "Придбати" активна тільки у випадку, якщо гравець має достатню кількість ігрової валюти для придбання корабля. У разі придбання, текст кнопки змінюється на "Обрати". Кнопка "Обрати" стає доступною після покупки корабля, і після обрання

гравець може керувати ним в ігровій сцені "GameScene". Текст кнопки змінюється на "Обрано".

"ShopManager" грає вирішальну роль у візуалізації внутрішнього ігрового магазину та взаємодії з попередніми скриптами, щоб втілити їхній функціонал. Графічний інтерфейс внутрішнього ігрового магазину побудований із використанням слайдера, який сприяє зручному огляду різних кораблів в обмеженому просторі, а також відображенню характеристик та кількості грошової валюти, яку гравець має на момент прогресу у грі.

4.4 Тестування розробленого додатку

Використанням управління пулом може бути об'єднано використання Pool Manager. Початком тестування мобільного ігрового додатку є його запуск. Після запуску додатку очікується побачити завантажене головне меню. Результат запуску додатку наведено на рисунку 4.9.

Рисунок 4.9 – Головне меню гри

Услід за успішним відкриттям програми розпочинається етап випробування функціоналу додатку. Для аналізу методів оптимізацції ігрового рушія «Action» ігри мобільного додатку, першим етапом тестування є запуск першого рівня. Для здійснення цієї дії потрібно натискати відповідну кнопку на головному меню та переходити на екран «рівня 1». Результат натискання кнопки наведено на рисунку 4.10.

Рисунок 4.10 – Результат натискання кнопки «рівня 1»

Після успішного запуску екрану «рівня 1», відкривається відтворення гри, де гравець повинен ліквідувати ворожі космічні одиниці, щоб накопичити ігрову

валюту. Це дає можливість придбати нові та удосконалені кораблі для подальшого подолання складніших рівнів. Під час прогресу в грі гравець може використовувати підняття рівня, що активує додатковий вогонь космічним бластером, сприяючи подальшому подоланню рівня. Результат покрокового проходження «рівня 1» зображено на рисунку 4.11.

Рисунок 4.11 – Результат покрокового проходження «рівня 1»

У випадку, якщо значення бар’єра гравця буде 0, відображений у верхній частині екрану синькою полосою, а також втратить всю зелену полосу життя, рівень вважатиметься непройденим. Після такого результату гравець має можливість спробувати рівень знову або повернутися в меню рівнів для вибору іншого виклику. Результат програшу гравця зображений на рисунку 4.12.

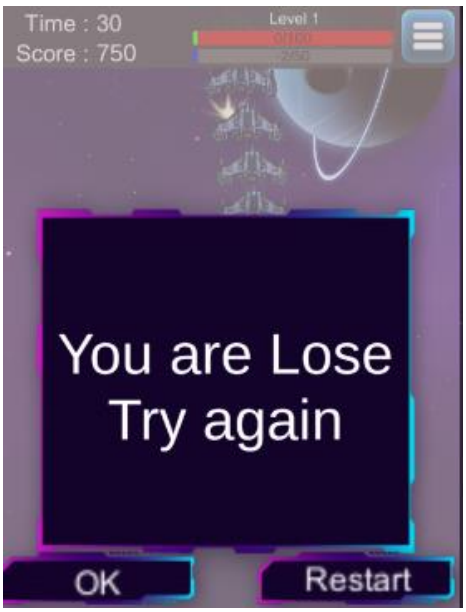

Рисунок 4.12 – Результат програшу гравця

У даного ігрового геймплею є два відмінних способи досягнення перемоги гравцем на різних рівнях гри. В першому випадку гравець успішно завершує весь ігровий рівень, зберігаючи свій космічний корабель, незалежно від отриманих ушкоджень під час подорожі; цей тип рівня не містить босів. Другий варіант передбачає наявність боса на рівні, і тут успіх гравця вимагає не лише виживання, але і обов'язкового знищення ворожого боса. Результат перемоги гравця зображений на рисунку 4.13.

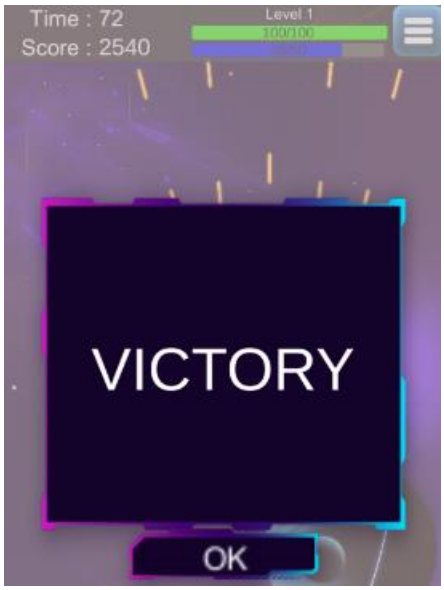

Рисунок 4.13 – Результат перемоги гравця

Після успішного завершення інтерфейсу меню рівнів гри з’являються зірки над вже успішно пройденими рівнями, які показують рівень виконання всіх необхідних умов для подальшого прогресу гри. Одну зірку можна отримати при пройденні рівня, коли захисний бар’єр досягає 0, а полоса життя залишається менше 50%. Дві зірки отримуєте за успішне подолання рівня, коли захисний бар’єр досягає 0, при цьому полоса життя залишається більше 50%. Три зірки нараховуються, якщо рівень життя залишається на рівні 100%, а захисний бар’єр має значення більше 0. Результат прогресу проходження гравця зображений на рисунку 4.14.

Рисунок 4.14 – Результату прогресу проходження гравця

У ігровій області, де пройшов другий рівень, відбулося експериментальне тестування для оцінки роботи боса. Результати тестування підтвердили успішність функціоналу, визначивши, що бос відрізняється від стандартних ворожих одиниць як за кількістю життя, так і за тактикою бою. Бос залишається на місці, не

пересуваючись вздовж бойового простору, і виконує певні траєкторії руху, що робить його подолання можливим лише через його повне знищення. Результат проходження гравцем рівень з босом зображений на рисунку 4.15.

Рисунок 4.15 – Результату проходження гравцем рівень з босом

В ході ігрової сцени рівня гри можливість встановлення гри на паузу врахована для забезпечення гравцеві зручності в разі його відволікання та з метою уникнення марної втрати часу, пов'язаної з прерванням прогресу гри. Ця опція викликає відображення спеціального меню, де гравець може налаштовувати звуковий супровід гри та користуватися кнопками для продовження гри, перезапуску поточного рівня чи повернення до головного меню. Результат паузи гри зображений на рисунку 4.16.

Рисунок 4.16 – Результату паузи гри

В ході цікаво проведеного часу та накопиченої грошової валюти випадає можливість здійснити придбання нового корабля з оновленими характеристиками, які є більш вдосконаленими, ніж у попередній версії корабля. Відповідно до параметрів кораблів визначається їхня вартість, яка з кожним наступним кораблем зростає, оскільки його характеристики перевищують ті, що були у попередніх. З метою тестування було здійснено придбання корабля з назвою "Riker", що в магазині представлений як другий за порядком, та має вартість 5000 ігрової валюти. Результат купівлі в магазині гравцем корабля зображений на рисунку4.17.

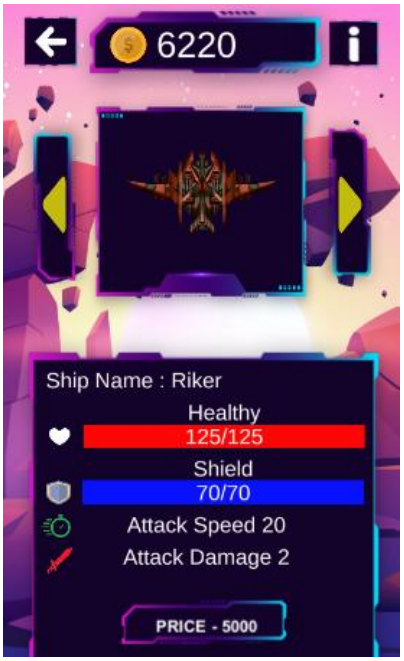

Рисунок 4.17 – Результату купівлі в магазині гравцем корабля

Після того, як гравець закупив собі новий вид корабля, йому відкривається можливість випробувати її в бойових випробуваннях на ігрових рівнях. Це можна зробити, натискавши кнопку «SELECT», і після цього на екрані з'явиться напис «SELECTED». Результат вибору гравцем корабля в магазині зображений на рисунку 4.18.

Рисунок 4.18 – Результату вибору гравцем корабля в магазині

На ігровому полі рівня проведено випробування допоміжного буста підняття рівня, який дарує кораблю гравця перевагу над опонентами у вигляді додаткових потужних вистрілів бластера. Цей буст має декілька рівнів, від 1 до 4. Рівні корабля застосовуються лише на протязі одного ігрового раунду, оскільки при наступному раунді вони повертаються до початкового рівня. На початковому рівні корабель може здійснювати лише один потужний вистріл бластера. При досягненні другого рівня кількість вистрілів збільшується вдвічі, на третьому рівні – три вистріли, а на кінцевому рівні – п'ять потужних вистрілів. Результат отримання підняття рівня корабля гравцем зображено на рисунку 4.19.

Рисунок 4.19 – Результату отримань підняття рівня корабля гравцем

Завершенням етапу тестування виявився музичний супровід в ігрових сценах. Кожній сцені призначалася окрема композиція, яка гармонійно впліталася у ігровий інтерфейс та функціонал. Гучність звуку у головному меню можна легко налаштувати, щоб забезпечити комфорт користувача. Ці параметри синхронізовані із всіма ігровими сценами та доступні в меню паузи під час гри. Результат регулювання звуку гравцем зображений на рисунку 4.20.

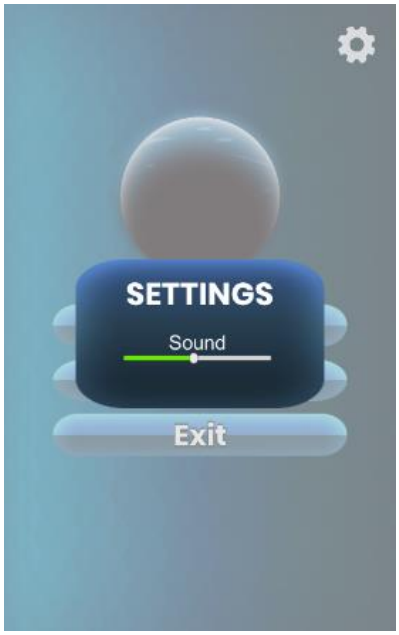

Рисунок 4.20 – Результату регулювання звуку гравцем

Інформація про мінімальні та рекомендовані технічні характеристики можна знайти в таблицях 4.3 та 4.4.

Таблиця 4.3 – Мінімальна конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 1.8 ГГц
Об’єм оперативної пам’яті	700 МБ для 32-розрядної системи і 2 ГБ для 64-разрядної системи
Місце на жорсткому диску	43 166КБ
Операційна система	Від Android 4.4 та вище

Таблиця 4.4 – Рекомендована конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 2.4 ГГц
Об’єм оперативної пам’яті	1 ГБ для 32-розрядної системи і 4 ГБ для 64-разрядної системи
Розмір жорсткого диску	43 166КБ
Операційна система	версія Android 13.0

4.5 Висновок

Провели тестування за допомогою Unity Profiler та провели порівняльні тести програми до оптимізації і після неї. Продуктивність використання Pool при великих кількостях об’єктів виправдали очікування. На початку при створенні об’єктів нашим пулом є allocation пам’яті але вже через кілька хвилин гри вона вже майже нульова, у порівняні з створенням об’єкту в разі виклику як було перед оптимізацією без використання Pool Manager і allocation кожен виклик. Данна оптимізація значно підвищила стабільність роботи нашої гри в довготривалій перспективі.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення наукового аудиту науково-дослідної роботи

Для наукових і пошукових науково-дослідних робіт зазвичай здійснюють оцінювання наукового ефекту. Основними ознаками наукового ефекту науково-дослідної роботи є новизна роботи, рівень її теоретичного опрацювання, перспективність, рівень розповсюдження результатів, можливість реалізації. Науковий ефект НДР можна охарактеризувати двома показниками: ступенем наукової новизни та рівнем теоретичного опрацювання. Значення показників ступеня новизни і рівня теоретичного опрацювання науково-дослідної роботи в балах наведено в табл. 5.1 та 5.2.

Таблиця 5.1 - Показники ступеня новизни науково-дослідної роботи

Ступінь новизни	Характеристика ступеня новизни	Значення показника ступеня новизни, бали
Принципово нова	ота якісно нова за постановкою задачі і ґрунтується на застосуванні оригінальних методів дослідження. Результати дослідження відкривають новий напрям в цій галузі науки і техніки. Отримано принципово нові факти, закономірності; розроблено нову теорію. Створено принципово новийпристрій, спосіб, метод	.100
Нова	Отримано нову інформацію, яка суттєво зменшує невизначеність наявних значень (по-новому або вперше пояснено відомі факти, закономірності, впроваджено нові поняття, розкрито структуру змісту). Проведено суттєве вдосконалення, доповнення і уточнення раніше досягнутих результатів	.60
Відносно нова	Робота має елементи новизни в постановці задачі і методах дослідження. Результати дослідження систематизують і узагальнюють наявну інформацію, визначають шляхи подальших досліджень; вперше знайдено зв’язок (або знайдено новий зв’язок) між явищами. В принципі, відомі положення поширено на велику кількість об’єктів, в результаті чого знайдено ефективне рішення.	.40

Продовження таблиці 5.1

Ступінь новизни	Характеристика ступеня новизни	Значення показника ступеня новизни, бали
Традиційна	Робота виконана за традиційною методикою. Результати дослідження мають інформаційний характер. Підтверджено або поставлено під сумнів відомі факти та твердження, які потребують перевірки. Знайдено новий варіант рішення, який не дає суттєвих переваг порівняно з існуючим	10
Не нова	Отримано результат, який раніше зафіксований в інформаційному полі та не був відомий авторам	2

Таблиця 5.2 - Показники рівня теоретичного опрацювання

актеристика рівня теоретичного опрацювання	Значення показника рівня теоретичного опрацювання, бали
криття закону, розробка теорії	.100
Глибоке опрацювання проблеми: багатоаспектний аналіз зв'язків, взаємозалежності між фактами з наявністю пояснень, наукової систематизації з побудовою евристичної моделі або комплексного прогнозу	.80
Розробка способу (алгоритму, програми), пристрою, отримання нової речовини	.60
Елементарний аналіз зв'язків між фактами та наявною гіпотезою, класифікація, практичні рекомендації для окремого випадку тощо	20
Опис окремих елементарних фактів, викладення досвіду, результатів спостережень, вимірювань тощо	5

Показник, який характреризує науковий ефект, визначається за формулою:

$$\mathrm{E}_{\text{нау}} = 0.6 * k_{\text{нов}} + \ 0.4 * k_{\text{теор}} \qquad (5.1)$$

$$k_{\text{нов}} = 45;\ k_{\text{теор}} = 50;$$

$$\mathrm{E}_{\text{нау}} = 0.6 * 45 + 50 * 0.4 = 47$$

Розробка має достатній рівень наукового ефекту за рахунок суттєвого покращення методів багатокритеріальної оцінки житлової нерухомості. Отримані методи покращають точність і швидкість. оцінки житлової нерухомості, створює більші можливості для побудови нових оцінок.

5.2 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу методів створення об'єктів за допомогою Pool Manager, що дозволить більш раціонально використовувати память пристрою на якому використовується додаток.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету кафедри програмного забезпечення: Войтко Вікторія Володимирівна, Майданюк Володимир Павлович, Ракитянська Ганна Борисівна.

Для проведення технологічного аудиту було використано таблицю 5.3 [30] в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 5.3 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження табл. 5.3

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовую-ться у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.4 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 5.5 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.5 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Войтко В. В.	Майданюк В. П.	Ракитянська Г. Б.
	Бали, виставлені експертами:		
1	3	2	2
2	2	3	3
3	3	2	4
4	4	4	3
5	3	3	3
6	3	3	4
7	2	3	3
8	2	1	3
9	3	4	3
10	4	3	2
11	4	2	3
12	4	3	4
Сума балів	$СБ_1$=37	$СБ_2$=33	$СБ_3$=37
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{37 + 33 + 37}{3} = 35.6$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 35.6 бали, що згідно таблиці 5.5 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Нова розробка буде корисною для розробників та проектів де потрібно створювати велику кількість однотипних об’єктів. Так як створення об'єктів в Unity3D визиває за собою аллокацію пам'яті та при знищенні об'єкта викликається GC(менеджер пам'яті). Менеджер пам'яті відстежує зони в купі, які визначені як невикористовувані. При запиті нового блоку пам'яті (припустимо, при створенні екземпляра об'єкта), менеджер вибирає зону, що не використовується, з якої слід виділити блок, і потім видаляє виділену пам'ять із зони відомого невикористовуваного простору. Для оптимізації цих процесів використовуємо Pool Manager який буде звертатися не до необхідного обьєкту в купі.

Так як GC видаляє, блоки купи які більше не використовуються, менеджер пам'яті переглядає всі активні змінні посилання і відзначає блоки, до яких вони посилаються як "live" (використовуються). Pool Manager не видаляє а вимикає єлементі які вже не використовуються, а потім перевикористовує їх за допомогою вмикання та переналаштування на нові параметри.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом. В таблиці 5.6 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 5.6 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Використання ОЗУ, ГБ	1,5	3	1,3	30%
Використання ЦП, %	23	15	1,4	30%
Середня кількість кадрів за секунду	60	80	1,3	30%
Середня швидкість компілювання додатку, с.	120	80	1,5	10%

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.2) та (5.3) і занесемо їх у відповідну колонку табл. 5.6.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{5.2}$$

або

$$q_i = \frac{P_{Бi}}{P_{Hi}} \tag{5.3}$$

де P_{Hi}, $P_{Бi}$ – числові значення i-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{3}{1,5} = 2;$$

$$q_2 = \frac{23}{15} = 1,5;$$

$$q_3 = \frac{80}{60} = 1,3;$$

$$q_4 = \frac{120}{80} = 1,5.$$

Відносний рівень якості нової розробки визначаємо за формулою (5.4):

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i, \tag{5.4}$$

$$К_{я.в.} = 2 \cdot 0,3 + 1,5 \cdot 0,3 + 1,3 \cdot 0,3 + 1,5 \cdot 0,1 = 1,59$$

Відносний коефіцієнт показника якості нової розробки більший одиниці, отже нова розробка якісніший базового товару-конкурента.

Наступним кроком є визначення конкурентоспроможності товару. Конкурентоспроможність товару є головною умовою конкурентоспроможності підприємства на ринку і важливою основою прибутковості його діяльності.

Однією із умов вибору товару споживачем є збіг основних ринкових характеристик виробу з умовними характеристиками конкретної потреби покупця. Такими характеристиками найчастіше вважають нормативні та технічні параметри, а також ціну придбання та вартість споживання товару.

В табл. 5.7 наведено технічні та економічні показники для розрахунку конкурентоспроможності нової розробки відносно товару-аналога, технічні дані взяті з попередніх розрахунків.

Таблиця 5.7 – Нормативні, технічні та економічні параметри нової розробки і товару-виробника

Показники	Варіанти	
	Базовий (товар-конкурент)	Новий (інноваційне рішення)
1	2	3
1. Нормативно-технічні показники		
Використання ОЗУ, ГБ	3	1,5
Використання ЦП, %	23	15
Середня кількість кадрів за секунду	80	60
Середня швидкість компілювання додатку, с.	120	80
2. Економічні показники		
Ціна придбання, грн.	500	300

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою (5.5):

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.5}$$

де $I_{m.n.}$ – індекс технічних параметрів; $I_{e.n.}$ – індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.6)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \tag{5.6}$$

де $P_{Нei}$, $P_{Бei}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$I_{е.п.} = \frac{300}{500} = 0{,}6;$$

$$К = \frac{1{,}59}{0{,}6} = 2{,}65.$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде конкурентоспроможне, ніж конкурентний товар.

5.3 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою (5.7):

$$З_О = \frac{М}{Т_р} * t \text{ (грн)} \qquad (5.7)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

T_P – число робочих днів в місяці; приблизно $T_P \approx 21...23$ дні;

t – число робочих днів роботи дослідника.

Для розробки програмні засоби для реалізації користувацьких інтерфейсів для їх використання в комп’ютерних іграх необхідно залучити програміста з посадовим окладом 28000 грн, та керівника з посадовим окладом 40000 грн. Кількість робочих днів у місяці складає 22, а кількість робочих днів програміста складає 35. Зведемо сумарні розрахунки до таблиця 5.8.

Таблиця 5.8 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Кількість днів роботи	Витрати на заробітну плату грн.
Керівник	40000	1818	35	63630
Програміст	28000	1278	35	44730
Всього				108360

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 – 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{доп}}{100\%} \quad (5.8)$$

$$З_д = 0{,}11 * 108360 = 11919 \text{ (грн)}$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.9):

$$З_н = (З_о + З_д) * \frac{Н_{ЗП}}{100} \text{ (грн)} \quad (5.9)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

$Н_{ЗП}$ – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$З_н = (108360 + 11919\ ) * \frac{22}{100} = 26461 \text{ (грн)}$$

4. Витрати на комплектуючі вироби, які використовують при виготовленні одиниці продукції, розраховуються, згідно їх номенклатури, за формулою:

$$К=\sum_{i=1}^{n} Н_i \cdot Ц_i \cdot К_i \tag{5.10}$$

де Ні – кількість комплектуючих і-го виду, шт.;

Ці – покупна ціна комплектуючих і-го найменування, грн.;

Кі – коефіцієнт транспортних витрат (1,1…1,15).

Потрібно закладати витрати на доставку у вигляді коефіцієнту транспортних витрат – 1.1. Інформацію про використані матеріали та комплектуючі подаємо у вигляді табл. 5.9.

Таблиця 5.9 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір	125	1	125
Ручка	11	1	11
CD-диск	12	1	12
Флешка	135	1	135
Всього			283
З врахуванням коефіцієнта транспортування			311,3

5. Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення необхідного для проведення дослідження. В роботі використовувалось безкоштовне програмне забезпечення тому витрати на нього не враховуються.

6. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$А_{обл} = \frac{Ц_б}{Т_в} * \frac{t_{вик}}{12} \tag{5.11}$$

де $Ц_б$ – балансова вартість даного виду обладнання (приміщень), грн.;

$t_{вик}$ – час користування;

$Т_в$ – термін використання обладнання (приміщень), цілі місяці.

Для розробки продукту використовувався персональний комп'ютер вартістю 27500 грн

Таблиця 6.10 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук (2)	55000	3	3	4583
Офісне приміщення	1 000 000	15	3	16666
Всього				21249

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_і} \qquad (5.12)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_і$ – коефіцієнт корисної дії обладнання, $\eta_і < 1$.

При використані 2 комп'ютерів з сумарною потужністю 1.5 кВт витрачалася електроенергія з ціною 7.6 грн за кВт в 2023 році.

$$B_e = \frac{1.5 \cdot 320 \cdot 7.6 \cdot 0{,}35}{0{,}81} = 1576$$

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = \left(З_о + З_р\right) \cdot \frac{Н_{св}}{100}, \quad (5.13)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$B_{нзв} = 108360 \cdot 0.22 = 23839 \text{ грн}$$

До статті «Витрати на роботи, які виконують сторонні підприємства, установи і організації» належать витрати на проведення досліджень, що не можуть бути виконані штатними працівниками або наявним обладнанням організації, а виконуються на договірній основі іншими підприємствами, установами і організаціями незалежно від форм власності та позаштатними працівниками.

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = \left(З_о + З_р\right) \cdot \frac{Н_{сп}}{100} \quad (5.14)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$B_{сп} = 108360 * 0.33 = 35758 \text{ грн}$$

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) * \frac{Нів}{100\%} \quad 5.15$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$І_в = 108360 * 0.7 = 75852 \text{ грн}$$

Накладні (загальновиробничі) витрати $В_{нзв}$ охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як 110% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_о + З_р) * \frac{Ннзв}{100\%} \quad 5.16$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$В_{нзв} = 108360 * 1.1 = 119196 \text{ грн}$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв} \quad 5.17$$

$$В = 108360 + 11919 + 26461 + 1576 + 23839 + 35758 + 75852 + 119196 = 402961 \text{ грн}$$

Загальні витрати ЗВ на завершення науково-дослідної (науковотехнічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{Взаг}{\eta} \quad 5.18$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії дослідного зразка, то коефіцієнт $\beta = 0.5$.

Звідси:

$$ЗВ = \frac{402961}{0.5} = 805922 \text{ грн.}$$

5.4 Оцінювання важливості та наукової значимості науково-дослідної роботи фундаментального чи пошукового характеру

Для обґрунтування доцільності виконання науково-дослідної роботи використовується спеціальний комплексний показник, що враховує важливість, результативність роботи, можливість впровадження її результатів у виробництво, величину витрат на роботу Комплексний показник рівня науково-дослідної роботи може бути розрахований за формулою:

$$К_р = \frac{I^n * T_C * R}{B * t}, \quad (5.19)$$

де I – коефіцієнт важливості роботи, I = 2...5;

n – коефіцієнт використання результатів роботи;

n = 0, коли результати роботи не будуть використовуватись;

n =1, коли результати роботи будуть використовуватись частково;

n = 2, коли результати роботи будуть використовуватись в дослідно-конструкторських розробках;

n = 3 , коли результати можуть використовуватись навіть без проведення дослідно конструкторських розробок;

T_c – коефіцієнт складності роботи, $T_c = 1 \dots 3$;

R – коефіцієнт результативності роботи; якщо результати роботи плануються вище відомих, то R = 4 ; якщо результати роботи відповідають відомому рівню, то R = 3; якщо нижче відомих результатів, то R =1;

B – вартість науково-дослідної роботи, тис. грн; t – час проведення дослідження, років. Визначення показників I, n, T_c, R, B, t здійснюється експертним шляхом або на основі нормативів.

Якщо $К_р > 1$, то науково-дослідну роботу можна вважати ефективною з високим науковим, технічним і економічним рівнями.

$$К_р = \frac{4^2 * 3 * 4}{806 * 0.25} = \frac{192}{201{,}5} = 0.97$$

Науково-дослідна робота є достатньо коштовною з економічної точки зору.

5.5 Висновки до економічного розділу

Було проведено оцінку комерційного потенціалу створення об'єктів за допомогою Pool Manager, що дозволить більш раціонально використовувати пам’ять пристрою на якому використовується додаток. При порівнянні нової розробки з аналогом виявлено, що вона є якіснішою і конкурентоспроможнішою відносно аналога, а також краще по технічним і економічним показникам.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 805 922 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 805 922 грн.

Оцінювання важливості та наукової значимості науково-дослідної роботи фундаментального характеру при прогнозі 0.97 за три місяці є достатньо коштовною з економічної точки зору.

ВИСНОВОК

У магістерської кваліфікаційної роботі дослідження методів оптимізації ігрових рушіїв action ігор мобільних додатків. Розроблено пакет Pool Manager для оптимізації ігрових рушіїв Action ігор мобільних додатків. Такий пакет сценаріїв можна використовувати у проектах Unity3D в яких необхідно вирішити проблему фрагментації пам’яті при створенні великої кількості однотипних елементів чи об’єктів. В основі розробки лежить розробка Pool з повторним використанням об’єктів після виконання поставлених цілей.

У першому розділі проведено аналыз багатопотоковості ігрових рушіїв, їх раціональне використання пам’яті при роботи програми з великою кількістю створюваних об’єктів. Розглянуто аналоги вже існуючих пул менеджерів, визначено їх особливості та недоліки, та між існуючими альтернативами, було вирішено створити свою систему Pool Manager, за рахунок чого визначено актуальність і задачі розробки. Розглянуто можливісті використання pooling, після аналізу було визначено такі завдання: створення класу Pool<T>, створення статичного класу Pool Manager, розробка ігрових рушіїв з використанням Pool, розробка Asset Pool Manager, створення Unity Package.

Було використано принципи Unity Package для створення Assets, що значно полегшило перенос реалізації Pool Manager між проектами. Описано розробку класу Pool<T> для управління пулом об'єктів у середовищі Unity, розглянуті його основні методи та взаємозв'язки. Також представлено клас Pool Manager як Singleton, що керує pooling objects у грі. Використано його для створення і управління пулами об'єктів. Детально розглянуто створення ігрових рушіїв з використанням Pool, "Level Controller" та модель роботи ігрової системи з ігровим магазином. Описано процес створення Asset Pool Manager для Unity, визначено важливі аспекти, такі як використання Prefab для шаблонів та правильне управління пулами для покращення продуктивності гри.

Проведено аналіз різних мов програмування де на основі кінцевого результату було обрано мову C#. Для вибору засобів реалізації пакету був вибраний

Unity Package в ігровому рушії Unity3D, що підтримує мову програмування C#, та середовище розробки Microsoft Visual Studio Community 2019 в якості IDE

У роботі розв’язано такі задачі: розроблено пакет Pool Manager вирішує проблему фрагментації пам'яті, яка ускладнює пошук вільних суміжних областей пам'яті а також надмірною витратою тактів процесора на операцій створення та знищення об'єктів, розроблено універсальний сценарії для створення великої кількості об'єктів з моментами оптимізації їх створення в пам'яті, супровід життєдіяльності створених об'єктів, швидке внесення сценаріїв пакету у любих додаток Unity3D, зручний та зрозумілий код використання сценаріїв.

Для тестування було обрано систему тестування Unity Profiler інструмент, який використовується для отримання інформації про продуктивність роботи програми у реальному часі. Проведено тестування пакету вже в існуючої програми та приведені приклади тестування та внесення змін для покращення її роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Войтко В.В. Особливості розробки мобільної шутер гри «SPACE CONFRONTATION» / В.В. Войтко, А. В. Денисюк, О. В. Гаврилюк, Н. Є. Барчук, Д. С. В. Самарасінгхе // Матеріали Всеукраїнської науково-практичної інтернет-конференції "Молодь в науці: дослідження, проблеми, перспективи - 2022", Секція - Інформаційні технології та комп‘ютерна інженерія. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/viewFile/16192/13632

2. Самарасінгхе Д.С.В. Дослідження методів оптимізації ігрових рушіїв ACTION ігор мобільних додатків / Д.С.В. Самарасінгхе, О.М. Рейда // Міжнародна Науково-Практична Інтернет-Конференція «Електронні Інформаційні Ресурси: Створення, Використання, Доступ» [Електронний ресурс] – Режим доступу до ресурсу: https://drive.google.com/file/d/1oVmxS3W_sEQPjes9S9AzWDaJxDxi6I0X/view

3. Metanit.com [Електронний ресурс] – Режим доступу: Metanit.com – https://metanit.com/sharp/tutorial/11.1.php

4. Pooling Toolkit [Електронний ресурс] – Режим доступу: Pooling Toolkit – https://assetstore.unity.com/packages/tools/pooling-toolkit-23177

5. 13Pixels Pooling [Електронний ресурс] – Режим доступу: 13Pixels Pooling – https://assetstore.unity.com/packages/tools/integration/13pixels-pooling-133317

6. Pure Pool [Електронний ресурс] – Режим доступу: Pure Pool https://thegamedev.guru/unity-cpu-performance/object-pooling/

7. Pooling Manager [Електронний ресурс] – Режим доступу: Pooling Manager – https://assetstore.unity.com/packages/tools/utilities/pooling-manager-195534

8. Msugvnua000 [Електронний ресурс] – Режим доступу: Msugvnua000 – http://msugvnua000.web710.discountasp.net/Posts/Details/3581

9. C# та Платформа .NET 3.0: Підручник / ЭндрюТроелсен: Видавець Apress, 2008. – 1239с.

10. Geeksforgeeks [Електронний ресурс] – Режим доступу: Geeksforgeeks https://www.geeksforgeeks.org/lifo-last-in-first-out-approach-in-programming/

11. Unity Game Development Cookbook: Підручник / В. Паріс, М. Джон: Видавець O'Reilly Media, 2019 – 391с.

12. Non-player character [Електронний ресурс] – Режим доступу: Non-player character: https://en.wikipedia.org/wiki/Non-player_character

13. Pro C# 9 with .NET 5 Основні принципи та практики програмування: Підручник / Філіп Япіксе: Видавець Apress, 2021 – 1411с.

14. Разработка игр на Unity: Підручник / Гейг Майк: Видавець Бомбора 2020 – 464с.

15. Unity Game Development Cookbook: Essentials for Every Game 1st Edition: Підручник / Jon Manning, Tim Nugent, Paris Buttfield-Addison: Видавець O'Reilly, 2019. – 406с.

16. Unity та C#. Геймдев від ідеї до реалізації: Підручник / Бонд Джеремі: Видавець Osmo, 2020 – 928с.

17. Конкурентність в C#. Асинхронное, паралельне і багатопотокове програмування: Підручник / Стивен Клірі: Видавець O'Reilly, 2020 – 304с.

18. C# для UNITY-розробників. Практичний акаунт у соціальних мережах: Підручник / Ларкович С. Н., Євдокимов Петр Іванович: Видавець O'Reilly, 2023 – 368с.

19. C# 10 in a Nutshell. The Definitive Reference: Підручник / Joseph Albahari: Видавець O'Reilly, 2022 – 1058с.

20. Unity [Електронний ресурс] – Режим доступу: Unity – https://unity.com/learn.

21. C# programming guide [Електронний ресурс] – Режим доступу: Microsoft – https://learn.microsoft.com/uk-ua/dotnet/csharp/programming-guide.

22. Microsoft Visual Studio Community 2019 [Електронний ресурс] – Режим доступу: Microsoft – https://apps.microsoft.com/detail/XP8CDJNZKFM06W?hl=uk-ua&gl=US.

23. Паттерни проектування для платформ C# і .NET Core: Підручник / Гаурав Арораа Джеффрі Жилберто: Видавець O'Reilly, 2021 – 352с.

24. Unity User Manual 2022.3 (LTS) [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/Manual/Prefabs.html

25. Use object pooling to boost performance of c# scripts in unity [Електронний ресурс] – Режим доступу: https://unity.com/how-to/use-object-pooling-boost-performance-c-scripts-unity

26. Object Pool Design Pattern [Електронний ресурс] – Режим доступу: https://lazyscripting.blogspot.com/2019/12/object-pool-design-pattern.html

27. Blazor дії [Електронний ресурс] – Режим доступу: Підручник / Крис Сэйнти: Видавець Print2print, 2023 – 380с.

28. Мистецтво тестування програм: Підручник / Гленфорд МайерсТом БаджеттКори Сандлер: Видавець O'Reilly, 2020 – 272с.

29. Основи юзабіліті-тестування: Підручник / Барнум К. М.: Видавництво ДМК Пресс, 2022 – 408с.

30. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

Додаток А

(обов'язковий)

Міністерство освіти і науки України

Вінницький національний технічний університет

факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою ПЗ

д.т.н., проф.

_____ О. Н. Романюк

"20" вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Дослідження методів оптимізації ігрових рушіїв Action ігор мобільних додатків» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доцент кафедри ПЗ, Рейда О.М.

"20" вересня 2023 р.

Виконав:

_____ студент гр. 2ПІ-22м Д. С. В. Самарасінгхе

"20" вересня 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Дослідження методів оптимізації ігрових рушіїв Action ігор мобільних додатків».

Галузь застосування – оптимізація процесу роботи пристроїв, менше навантаження на технічні пристрої.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на БДР та наказ № 247 від «18» вересня 2023 р. ректора ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи досліджені методів оптимізації ігрових рушіїв Action ігор мобільних додатків для підвищення продуктивності та оптимізація використання ресурсів за рахунок багатопотоковості процесів та використання пулу (ObjectPool).

Основними задачами дослідження є:

– провести аналізи існуючих методів та засобів створення об’єктів для підвищення продуктивності та оптимізації;

– модифікувати метод підвищення продуктивності використання пам’яті пристрою;

– модифікувати метод оптимізації використання ресурсів за рахунок багатопотоковості;

– створити універсальний Pool Manager на основі патерна пул об'єктів;

– розробити програмні компоненти та систему візуалізації на основі запропонованих методів;

– провести експериментальні дослідження розроблених засобів тестування проекту та продемонструвати отримані результати.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР:

1. Pooling Manager [Електронний ресурс] – Режим доступу: Pooling Manager – https://assetstore.unity.com/packages/tools/utilities/pooling-manager-195534

2. Unity Game Development Cookbook: Essentials for Every Game 1st Edition: Підручник / Jon Manning, Tim Nugent, Paris Buttfield-Addison: Видавець O'Reilly, 2019. – 406с.

3. Unity та C#. Геймдев від ідеї до реалізації: Підручник / Бонд Джеремі: Видавець Osmo, 2020 – 928с.

4. C# для UNITY-розробників. Практичний акаунт у соціальних мережах: Підручник / Ларкович С. Н., Євдокимов Петр Іванович: Видавець O'Reilly, 2023 – 368с.

5. 23. Паттерни проектування для платформ C# і .NET Core: Підручник / Гаурав Арораа Джеффрі Жилберто: Видавець O'Reilly, 2021 – 352с

6. Object Pool Design Pattern [Електронний ресурс] – Режим доступу: https://lazyscripting.blogspot.com/2019/12/object-pool-design-pattern.html

7. Мистецтво тестування програм: Підручник / Гленфорд МайерсТом БаджеттКори Сандлер: Видавець O'Reilly, 2020 – 272с

8. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

5. Технічні вимоги

- середовище ігрового рушія: Unity;
- середовище розробки – Microsoft Visual Studio;
- мова програмування – C#;
- метод ігрового пулу – використання Pool Manager;

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

1. Пояснювальна записка до МКР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз продуктивності багатопотоковості додатку в якому часто створюються об'єкти	20.09.23 – 02.10.23
2	Розробка архітектури та алгоритмів програмного продукту	03.10.23 – 23.10.23
3	Аналіз і вибір мови програмування та середовища розробки	16.10.23 – 23.10.23
4	Розробка програмного продукту	24.10.23 – 13.11.23
5	Тестування програми	14.11.23 – 21.11.23
6	Розробка економічної частини	17.11.23 –25.11.23
7	Оформлення матеріалів до захисту МКР	22.11.23 – 01.12.23

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

Додаток Б

(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Дослідження методів оптимізації ігрових рушіїв «Action» ігор мобільних додатків.

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Рейда О. М.

Оригінальність	94 %
Схожість	6 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Самарасінгхе Д. С. В.

Керівник роботи _____ Рейда О. М.

Додаток В

(довідковий)
Лістинг програмного коду оптимізації ігрових рушіїв Action ігор

```
//Модуль Pool
using System;
using System.Collections.Generic;
using UnityEngine;

namespace MonsterLove.Collections
{
	public class Pool<T>
	{
		private List<PoolSlot<T>> list;
		private Dictionary<T, PoolSlot<T>> lookup;
		private Func<T> factoryFunc;
		private int lastIndex = 0;

		public Pool(Func<T> factoryFunc, int initialSize)
		{
			this.factoryFunc = factoryFunc;

			list = new List<PoolSlot<T>>(initialSize);
			lookup = new Dictionary<T, PoolSlot<T>>(initialSize);

			Warm(initialSize);
		}

		private void Warm(int capacity)
		{
			for (int i = 0; i < capacity; i++)
			{
				CreateContainer();
			}
		}

		private PoolSlot<T> CreateContainer()
		{
			var container = new PoolSlot<T>();
			container.Item = factoryFunc();
			list.Add(container);
			return container;
		}

		public T GetItem()
		{
			PoolSlot<T> container = null;
			for (int i = 0; i < list.Count; i++)
			{
				lastIndex++;
				if (lastIndex > list.Count - 1) lastIndex = 0;

				if (list[lastIndex].Used)
				{
					continue;
				}
```

```
                        else
                        {
                                container = list[lastIndex];
                                break;
                        }
                }

                if (container == null)
                {
                        container = CreateContainer();
                }

                container.Consume();
                lookup.Add(container.Item, container);
                return container.Item;
        }

        public void ReleaseItem(object item)
        {
                ReleaseItem((T) item);
        }

        public void ReleaseItem(T item)
        {
                if (lookup.ContainsKey(item))
                {
                        var container = lookup[item];
                        container.Release();
                        lookup.Remove(item);
                }
                else
                {
                        Debug.LogWarning("This object pool does not contain the item provided: " +
item);
                }
        }

        public int Count
        {
                get { return list.Count; }
        }

        public int CountUsedItems
        {
                get { return lookup.Count; }
        }
    }
}

//Модуль Pool Manager
using System;
using System.Collections.Generic;
using MonsterLove.Collections;
using UnityEngine;

public class Pool Manager : Singleton<Pool Manager>
{
```

```
public bool logStatus;
public Transform root;

private Dictionary<GameObject, Pool<GameObject>> prefabLookup;
private Dictionary<GameObject, Pool<GameObject>> instanceLookup;

private bool dirty = false;

void Awake ()
{
	prefabLookup = new Dictionary<GameObject, Pool<GameObject>>();
	instanceLookup = new Dictionary<GameObject, Pool<GameObject>>();
}

void Update()
{
	if(logStatus && dirty)
	{
		PrintStatus();
		dirty = false;
	}
}

public void warmPool(GameObject prefab, int size)
{
	if(prefabLookup.ContainsKey(prefab))
	{
		throw new Exception("Pool for prefab " + prefab.name + " has already been created");
	}
	var pool = new Pool<GameObject>(() => { return InstantiatePrefab(prefab); }, size);
	prefabLookup[prefab] = pool;

	dirty = true;
}

public GameObject spawnObject(GameObject prefab)
{
	return spawnObject(prefab, Vector3.zero, Quaternion.identity);
}

public GameObject spawnObject(GameObject prefab, Vector3 position, Quaternion rotation)
{
	if (!prefabLookup.ContainsKey(prefab))
	{
		WarmPool(prefab, 1);
	}

	var pool = prefabLookup[prefab];

	var clone = pool.GetItem();
	clone.transform.SetPositionAndRotation(position, rotation);
	clone.SetActive(true);

	instanceLookup.Add(clone, pool);
	dirty = true;
	return clone;
}
```

```
	public void releaseObject(GameObject clone)
	{
		clone.SetActive(false);

		if(instanceLookup.ContainsKey(clone))
		{
			instanceLookup[clone].ReleaseItem(clone);
			instanceLookup.Remove(clone);
			dirty = true;
		}
		else
		{
			Debug.LogWarning("No pool contains the object: " + clone.name);
		}
	}


	private GameObject InstantiatePrefab(GameObject prefab)
	{
		var go = Instantiate(prefab) as GameObject;
		if (root != null) go.transform.parent = root;
		return go;
	}

	public void PrintStatus()
	{
		foreach (KeyValuePair<GameObject, Pool<GameObject>> keyVal in prefabLookup)
		{
			Debug.Log(string.Format("Object Pool for Prefab: {0} In Use: {1} Total {2}",
keyVal.Key.name, keyVal.Value.CountUsedItems, keyVal.Value.Count));
		}
	}

	#region Static API

	public static void WarmPool(GameObject prefab, int size)
	{
		Instance.warmPool(prefab, size);
	}

	public static GameObject SpawnObject(GameObject prefab)
	{
		return Instance.spawnObject(prefab);
	}

	public static GameObject SpawnObject(GameObject prefab, Vector3 position, Quaternion rotation)
	{
		return Instance.spawnObject(prefab, position, rotation);
	}

	public static void ReleaseObject(GameObject clone)
	{
		Instance.releaseObject(clone);
	}

	#endregion
```

```
}

//Модуль  PoolSlot

namespace MonsterLove.Collections
{
	public class PoolSlot<T>
	{
		private T item;

		public bool Used { get; private set; }

		public void Consume()
		{
			Used = true;
		}

		public T Item
		{
			get
			{
				return item;
			}
			set
			{
				item = value;
			}
		}

		public void Release()
		{
			Used = false;
		}
	}
}

//Модуль Singleton
using UnityEngine;

public class Singleton<T> : MonoBehaviour where T : Singleton<T>
{
	public static T Instance
	{
		get
		{
			if (instance == null)
			{
				T[] managers = Object.FindObjectsOfType(typeof(T)) as T[];
				if (managers.Length != 0)
				{
					if (managers.Length == 1)
					{
						instance = managers[0];
						instance.gameObject.name = typeof(T).Name;
						return instance;
					}
					else
```

```
                                        {
                                                Debug.LogError("Class " + typeof(T).Name + " exists multiple times in violation of singleton pattern. Destroying all copies");
                                                foreach (T manager in managers)
                                                {
                                                        Destroy(manager.gameObject);
                                                }
                                        }
                                }
                                var go = new GameObject(typeof(T).Name, typeof(T));
                                instance = go.GetComponent<T>();
                                DontDestroyOnLoad(go);
                        }
                        return instance;
                }
                set
                {
                        instance = value as T;
                }
        }
        private static T instance;
}


//Модуль Bullet
using UnityEngine;
using System.Collections;

public class Bullet : MonoBehaviour
{
        public float accel;
        private float velocity;

        void OnEnable()
        {
                velocity = 0;
        }

        void Update()
        {
                velocity += accel;

                transform.Translate(0, velocity, 0);

                if(transform.position.y > 10)
                {
                        Finish();
                }
        }

        void Finish()
        {
                PoolManager.ReleaseObject(this.gameObject);
        }
}

//Модуль DirectMoving
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/// <summary>
/// This script moves the attached object along the Y-axis with the defined speed
/// </summary>
public class DirectMoving : MonoBehaviour {

    [Tooltip("Moving speed on Y axis in local space")]
    public float speed;

    //moving the object with the defined speed
    private void Update()
    {
        transform.Translate(Vector3.up * speed * Time.deltaTime);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Border"))
        {
            PoolManager.ReleaseObject(this.gameObject);
            //Destroy(this.gameObject);
        }
    }
}

//Модуль Enemy
using System.Collections;
using System.Collections.Generic;
using UnityEngine;



/// <summary>
/// This script defines 'Enemy's' health and behavior.
/// </summary>
public class Enemy : MonoBehaviour {

    public static event System.Action<int> OnHitEmemy;

    #region FIELDS
    [Tooltip("Health points in integer")]
    public int health;

    [Tooltip("Enemy's projectile prefab")]
    public GameObject Projectile;

    [Tooltip("VFX prefab generating after destruction")]
    public GameObject destructionVFX;
    public GameObject hitEffect;

    [SerializeField] private int Coins;

    [HideInInspector] public int shotChance; //probability of 'Enemy's' shooting during tha path
    [HideInInspector] public float shotTimeMin, shotTimeMax; //max and min time for shooting from the
```

```
beginning of the path
    #endregion

    private void Start()
    {
        PoolManager.WarmPool(Projectile, 50);
        Invoke("ActivateShooting", Random.Range(shotTimeMin, shotTimeMax));
    }

    //coroutine making a shot
    void ActivateShooting()
    {
        if (Random.value < (float)shotChance / 100)                    //if random value less than shot probability,
making a shot
        {
            GameObject enemyShoot =  PoolManager.SpawnObject(Projectile);
            enemyShoot.transform.position = gameObject.transform.position;
            enemyShoot.transform.rotation = Quaternion.identity;

                                //Instantiate(Projectile,  gameObject.transform.position, Quaternion.identity);
        }
    }

    //method of getting damage for the 'Enemy'
    public void GetDamage(int damage)
    {
        health -= damage;          //reducing health for damage value, if health is less than 0, starting destruction
procedure
        if (health <= 0)
            Destruction();
        else
            Instantiate(hitEffect,transform.position,Quaternion.identity,transform);
    }

    //if 'Enemy' collides 'Player', 'Player' gets the damage equal to projectile's damage value
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            if (Projectile.GetComponent<Projectile>() != null)
                Player.instance.GetDamage(Projectile.GetComponent<Projectile>().damage);
            else
                Player.instance.GetDamage(1);
        }
    }

    //method of destroying the 'Enemy'
    void Destruction()
    {
        Instantiate(destructionVFX, transform.position, Quaternion.identity);
        OnHitEmemy?.Invoke(Coins);
                    Destroy(gameObject);
    }
}

//Модуль Gun
using UnityEngine;
```

```
using System.Collections;

public class Gun : MonoBehaviour
{
	public GameObject bulletPrefab;

	//Optional: Warm the pool and preallocate memory
	void Start()
	{
		PoolManager.WarmPool(bulletPrefab, 20);

		//Notes
		// Make sure the prefab is inactive, or else it will run update before first use
	}

	void Update()
	{
		if(Input.GetButton("Fire1"))
		{
			Vector3 pos = new Vector3(Input.mousePosition.x, Input.mousePosition.y, 10f);
			FireBullet(Camera.main.ScreenToWorldPoint(pos), Quaternion.identity);
		}
	}

	//Spawn pooled objects
	void FireBullet(Vector3 position, Quaternion rotation)
	{
		var bullet = PoolManager.SpawnObject(bulletPrefab, position,
rotation).GetComponent<Bullet>();

		//Notes:
		// bullet.gameObject.SetActive(true) is automatically called on spawn
		// When done with the instance, you MUST release it!
		// If the number of objects in use exceeds the pool size, new objects will be created
	}

}

//Модуль PlayerShooting
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using DG.Tweening;


//guns objects in 'Player's' hierarchy
[System.Serializable]
public class Guns
{
   public GameObject rightGun, leftGun, centralGun;
   [HideInInspector] public ParticleSystem leftGunVFX, rightGunVFX, centralGunVFX;
}

[System.Serializable]
public class Rockets
{
```

```
    public GameObject leftRocket, rightRocket;
    [HideInInspector] public ParticleSystem leftRocketVFX, rightRocketVFX;
}

public class PlayerShooting : MonoBehaviour {

    [Tooltip("shooting frequency. the higher the more frequent")]
    public float fireRate;

    [Tooltip("projectile prefab")]
    public GameObject projectileObject;
    [Tooltip("current Lazer power")]
    [Range(1, 4)]       //change it if you wish
    public int weaponPower = 1;
    [Range(0.2f, 1f)]
    private float resetLazerTime;
    public Guns guns;
    [Space]
    public GameObject projectileObjectRocket;
    [Tooltip("current Rocket power")]
    [Range(1, 2)]
    public int RocketState = 1;
    [SerializeField, Range(0.2f, 1f)]
    private float resetRocketTime;
    private float time;
    private bool canRocketShooting = true;
    public Rockets rockets;
    //time for a new shot
    [HideInInspector] public float nextFire;

    [HideInInspector] public int AttackSpeed;
    [HideInInspector] public int AttackDamage;

    [SerializeField] bool AutoShooting = true;

    [HideInInspector] public int maxweaponPower = 4;
    public static PlayerShooting instance;

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

    private void Start()
    {
        SpawnPoolObjects();

                    guns.leftGunVFX = guns.leftGun.GetComponent<ParticleSystem>();
        guns.rightGunVFX = guns.rightGun.GetComponent<ParticleSystem>();
        guns.centralGunVFX = guns.centralGun.GetComponent<ParticleSystem>();

        ChangeRocketSate(RocketState);
    }

    private void SpawnPoolObjects()
    {
```

```
                PoolManager.WarmPool(projectileObject, 50);
                PoolManager.WarmPool(projectileObjectRocket, 50);
        }

    private void Update()
    {
        if (AutoShooting)
        {
            if (Time.time > nextFire)
            {
                MakeAShot();
                nextFire = Time.time + 1 / fireRate;
            }
        }
        else
        {
                        if (Input.GetMouseButton(0))
                            MakeAShot();
                }

        MakeAShotRocket();
    }

    //method for a shot
    public void MakeAShot()
    {
        switch (weaponPower) // according to weapon power 'pooling' the defined anount of projectiles, on the defined position, in the defined rotation
        {
            case 1:
                CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
                guns.centralGunVFX.Play();
                break;
            case 2:
                CreateLazerShot(projectileObject, guns.rightGun.transform.position, Vector3.zero);
                guns.leftGunVFX.Play();
                CreateLazerShot(projectileObject, guns.leftGun.transform.position, Vector3.zero);
                guns.rightGunVFX.Play();
                break;
            case 3:
                CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
                CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -5));
                guns.leftGunVFX.Play();
                CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 5));
                guns.rightGunVFX.Play();
                break;
            case 4:
                CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
                CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -5));
                guns.leftGunVFX.Play();
                CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 5));
                guns.rightGunVFX.Play();
                CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 15));
                CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -15));
                break;
        }
    }
```

```
    public void MakeAShotRocket()
    {
        if (!canRocketShooting)
            return;

        switch (RocketState)
        {
            case 1:
                return;
            case 2:
                CreateRocketShot(projectileObjectRocket, rockets.rightRocket.transform.position, Vector3.zero);
                CreateRocketShot(projectileObjectRocket, rockets.leftRocket.transform.position, Vector3.zero);
                break;
        }

        time = resetRocketTime;
        canRocketShooting = false;
    }


    public void ChangeRocketSate(int i)
    {
        if(i == 1)
        {
            rockets.leftRocket.SetActive(false);
            rockets.rightRocket.SetActive(false);
            return;
        }

        if (i > 2)
            i = 2;

        rockets.leftRocket.SetActive(!rockets.leftRocket.activeSelf);
        rockets.rightRocket.SetActive(!rockets.rightRocket.activeSelf);

        RocketState = i;
    }

    void CreateLazerShot(GameObject lazer, Vector3 pos, Vector3 rot) //translating 'pooled' lazer shot to the
defined position in the defined rotation
    {
        GameObject Lazer = SpawnShot(lazer);
                    Lazer.transform.position = pos;
        Lazer.transform.rotation = Quaternion.Euler(rot);
                    //Instantiate(lazer, pos, ); //; //PoolManager.SpawnObject(lazer, pos, Quaternion.Euler(rot))
        SetParams(Lazer);
    }

    void CreateRocketShot(GameObject rocket, Vector3 pos, Vector3 rot) //translating 'pooled' lazer shot to the
defined position in the defined rotation
    {
        GameObject Rocket = SpawnShot(rocket);
                    Rocket.transform.position = pos;
                    Rocket.transform.rotation = Quaternion.Euler(rot);
                    //Instantiate(rocket, pos, Quaternion.Euler(rot));
                    SetParams(Rocket);
```

```
    }

    void SetParams(GameObject obj)
    {
        DirectMoving directMoving = obj.GetComponent<DirectMoving>();
        Projectile projectile = obj.GetComponent<Projectile>();

        if(directMoving != null)
        {
            directMoving.speed = AttackSpeed;
        }
        if(projectile != null)
        {
            projectile.damage = AttackDamage;
        }
    }

    private GameObject SpawnShot(GameObject obj)
    {
        return PoolManager.SpawnObject(obj);
    }

}

//Модуль Wave
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

/// <summary>
/// This script generates an enemy wave. It defines how many enemies will be emerging, their speed and
emerging interval.
/// It also defines their shooting mode. It defines their moving path.
/// </summary>
[System.Serializable]
public class Shooting
{
    [Range(0,100)]
    [Tooltip("probability with which the ship of this wave will make a shot")]
    public int shotChance;

    [Tooltip("min and max time from the beginning of the path when the enemy can make a shot")]
    public float shotTimeMin, shotTimeMax;
}

public class Wave : MonoBehaviour {

    #region FIELDS
    [Tooltip("Enemy's prefab")]
    public GameObject enemy;

    [Tooltip("a number of enemies in the wave")]
    public int count;

    [Tooltip("path passage speed")]
    public float speed;
```

```
    [Tooltip("time between emerging of the enemies in the wave")]
    public float timeBetween;

    [Tooltip("points of the path. delete or add elements to the list if you want to change the number of the points")]
    public Transform[] pathPoints;

    [Tooltip("whether 'Enemy' rotates in path passage direction")]
    public bool rotationByPath;

    [Tooltip("if loop is activated, after completing the path 'Enemy' will return to the starting point")]
    public bool Loop;

    [Tooltip("color of the path in the Editor")]
    public Color pathColor = Color.yellow;
    public Shooting shooting;

    [Tooltip("if testMode is marked the wave will be re-generated after 3 sec")]
    public bool testMode;

    public int OneEnemyCoins = 10;
    #endregion

    private void Start()
    {
        PoolManager.WarmPool(enemy, 50);
        StartCoroutine(CreateEnemyWave());
    }

    IEnumerator CreateEnemyWave() //depending on chosed parameters generating enemies and defining their parameters
    {
        for (int i = 0; i < count; i++)
        {
            GameObject newEnemy;
            newEnemy = PoolManager.SpawnObject(enemy, enemy.transform.position, Quaternion.identity);
            FollowThePath followComponent = newEnemy.GetComponent<FollowThePath>();
            followComponent.path = pathPoints;
            followComponent.speed = speed;
            followComponent.rotationByPath = rotationByPath;
            followComponent.loop = Loop;
            followComponent.SetPath();
            Enemy enemyComponent = newEnemy.GetComponent<Enemy>();
            enemyComponent.shotChance = shooting.shotChance;
            enemyComponent.shotTimeMin = shooting.shotTimeMin;
            enemyComponent.shotTimeMax = shooting.shotTimeMax;
            newEnemy.SetActive(true);
            yield return new WaitForSeconds(timeBetween);
        }
        if (testMode)       //if testMode is activated, waiting for 3 sec and re-generating the wave
        {
            yield return new WaitForSeconds(3);
            StartCoroutine(CreateEnemyWave());
        }
        else if (!Loop)
            Destroy(gameObject);
```

```
    }

    void OnDrawGizmos()
    {
        DrawPath(pathPoints);
    }

    void DrawPath(Transform[] path) //drawing the path in the Editor
    {
        Vector3[] pathPositions = new Vector3[path.Length];
        for (int i = 0; i < path.Length; i++)
        {
            pathPositions[i] = path[i].position;
        }
        Vector3[] newPathPositions = CreatePoints(pathPositions);
        Vector3 previosPositions = Interpolate(newPathPositions, 0);
        Gizmos.color = pathColor;
        int SmoothAmount = path.Length * 20;
        for (int i = 1; i <= SmoothAmount; i++)
        {
            float t = (float)i / SmoothAmount;
            Vector3 currentPositions = Interpolate(newPathPositions, t);
            Gizmos.DrawLine(currentPositions, previosPositions);
            previosPositions = currentPositions;
        }
    }

    Vector3 Interpolate(Vector3[] path, float t)
    {
        int numSections = path.Length - 3;
        int currPt = Mathf.Min(Mathf.FloorToInt(t * numSections), numSections - 1);
        float u = t * numSections - currPt;
        Vector3 a = path[currPt];
        Vector3 b = path[currPt + 1];
        Vector3 c = path[currPt + 2];
        Vector3 d = path[currPt + 3];
        return 0.5f * ((-a + 3f * b - 3f * c + d) * (u * u * u) + (2f * a - 5f * b + 4f * c - d) * (u * u) + (-a + c) * u +
2f * b);
    }

    Vector3[] CreatePoints(Vector3[] path)  //using interpolation method calculating the path along the path
points
    {
        Vector3[] pathPositions;
        Vector3[] newPathPos;
        int dist = 2;
        pathPositions = path;
        newPathPos = new Vector3[pathPositions.Length + dist];
        Array.Copy(pathPositions, 0, newPathPos, 1, pathPositions.Length);
        newPathPos[0] = newPathPos[1] + (newPathPos[1] - newPathPos[2]);
        newPathPos[newPathPos.Length - 1] = newPathPos[newPathPos.Length - 2] +
(newPathPos[newPathPos.Length - 2] - newPathPos[newPathPos.Length - 3]);
        if (newPathPos[1] == newPathPos[newPathPos.Length - 2])
        {
            Vector3[] LoopSpline = new Vector3[newPathPos.Length];
            Array.Copy(newPathPos, LoopSpline, newPathPos.Length);
            LoopSpline[0] = LoopSpline[LoopSpline.Length - 3];
```

```
            LoopSpline[LoopSpline.Length - 1] = LoopSpline[2];
            newPathPos = new Vector3[LoopSpline.Length];
            Array.Copy(LoopSpline, newPathPos, LoopSpline.Length);
        }
        return (newPathPos);
    }
}
```

Додаток Д

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ІГРОВИХ РУШІЇВ ACTION ІГОР МОБІЛЬНИХ ДОДАТКІВ

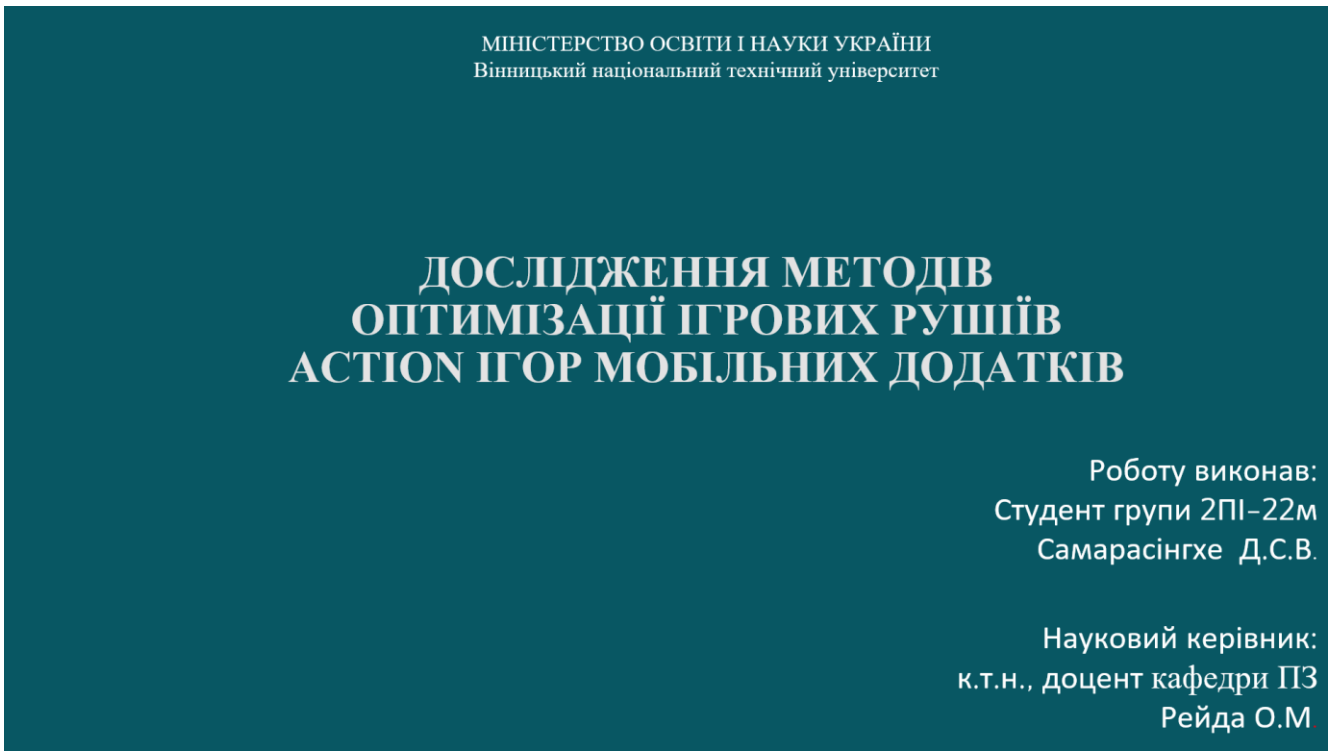

Рисунок Д.1 – Титульний слайд

Рисунок Д.2 – Мета, об’єкт і предмет роботи

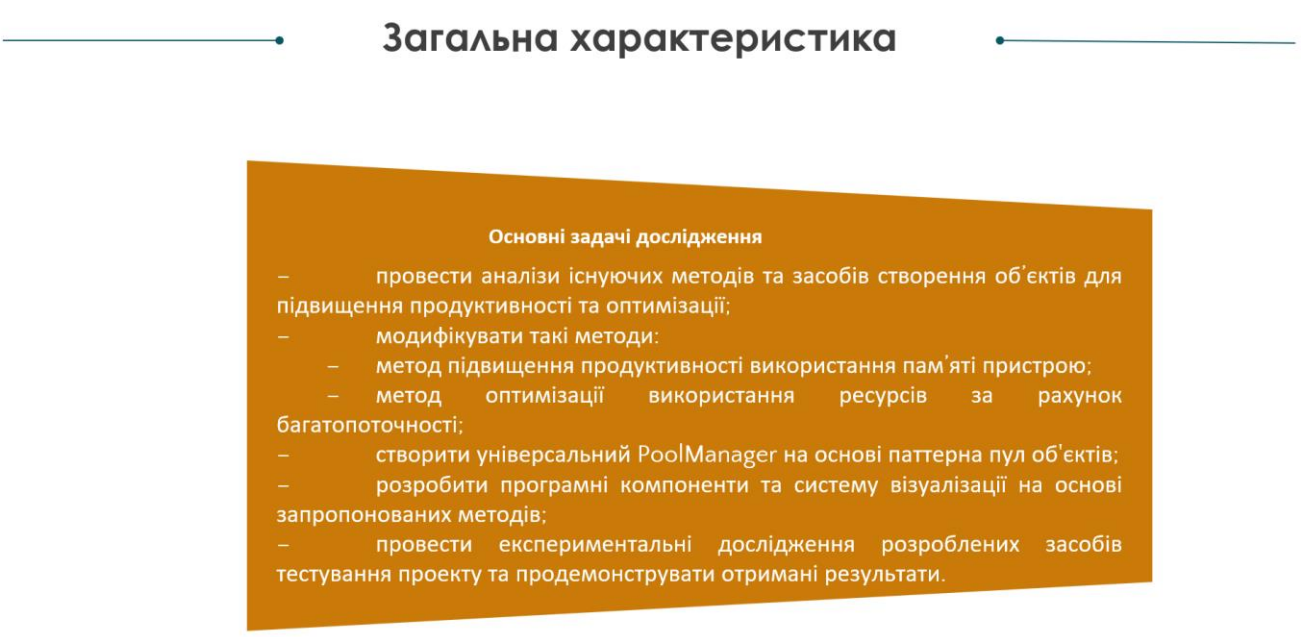

Рисунок Д.3 – Основні задачі дослідження

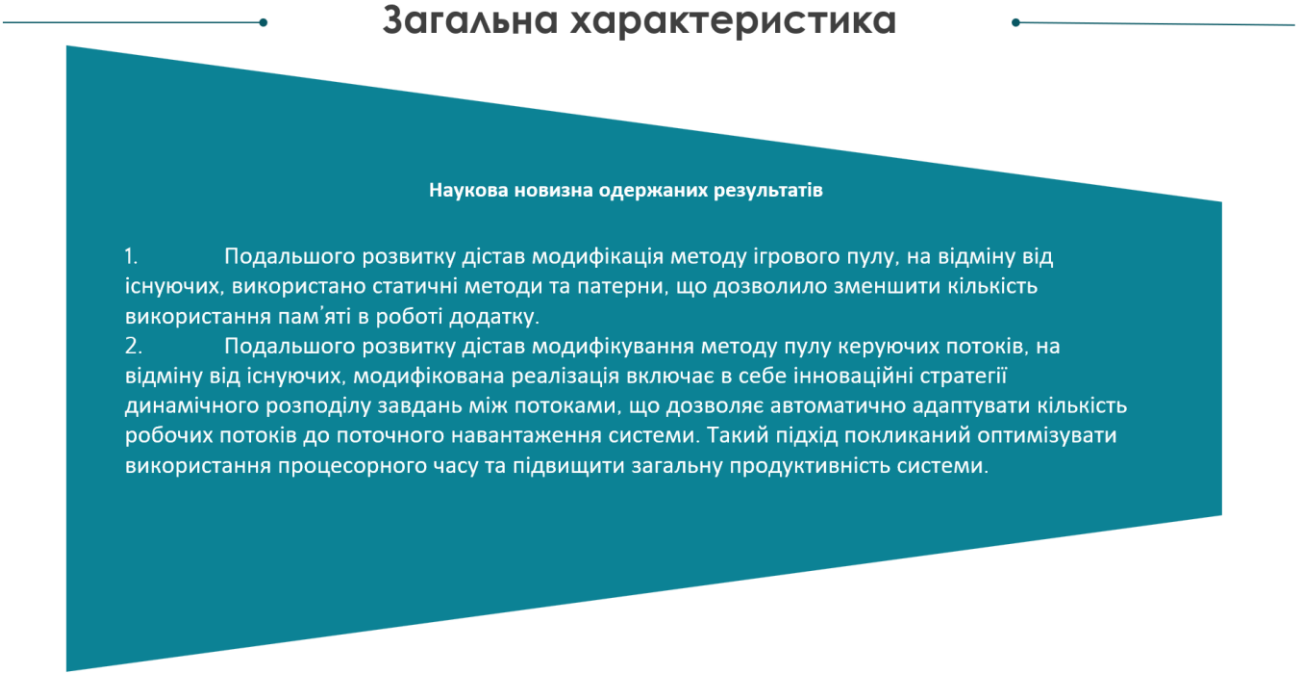

Рисунок Д.4 – Наукова новизна одержаних результатів

Загальна характеристика

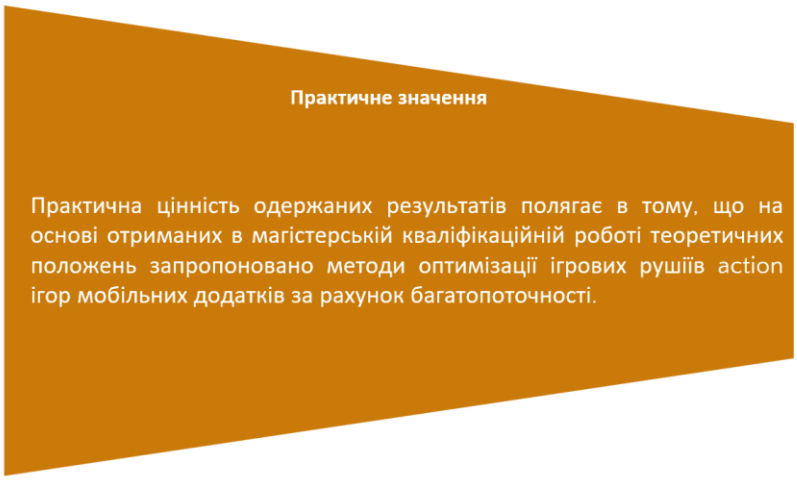

Рисунок Д.5 – Практичне значення

Існуючі інструменти реалізації Pool

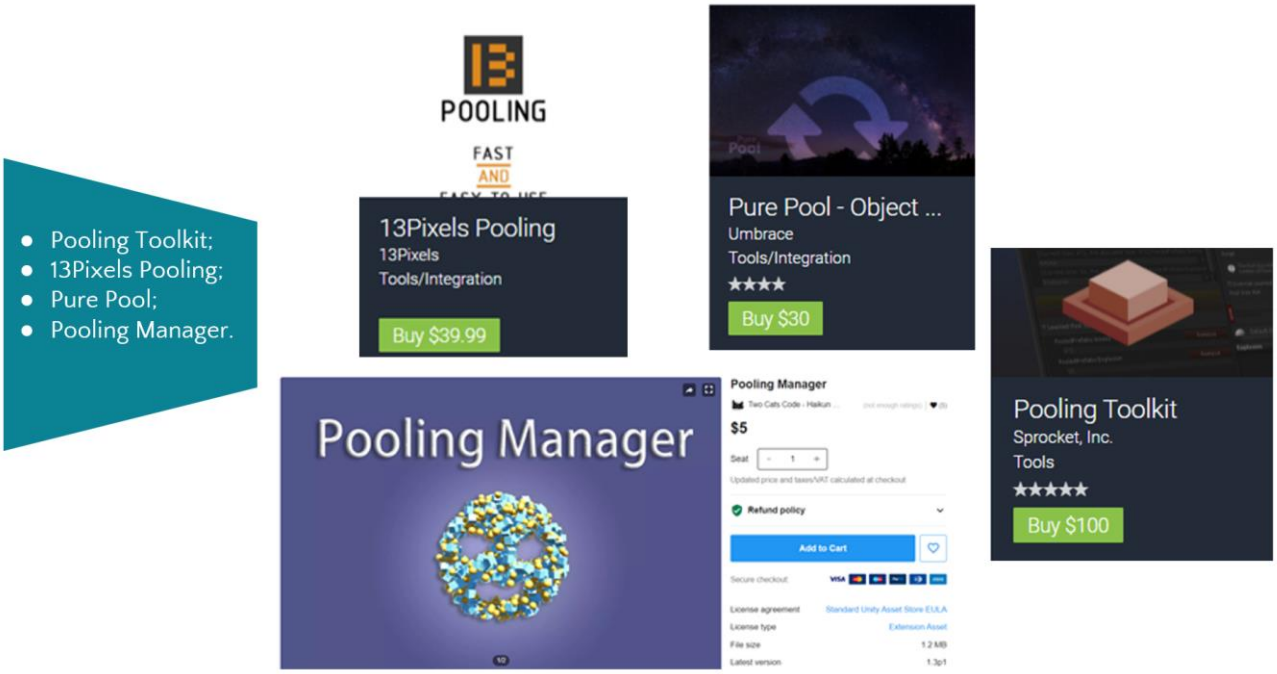

Рисунок Д.6 – Існуючі інструменти реалізації Pool

Задачі дослідження

– типобезпечність пула на етапі компіляції;
– робота пула з будь-якими класами, в тому числі іноземними;
– просте використання в коді;
– автоматичне створення нових об'єктів при нехватці, їх ініціалізація користувача;
– обмеження кількості загальних виділених об'єктів;
– автоочищення об'єкта при його поверненні в пул;
– потокобезпечність (бажано, з урахуванням витрат на синхронізацію);
– підтримка поширення екземплярів пулу (звідси випливає хоча б найпростіший контроль того, щоб об'єкти повертался саме в пули).

Рисунок Д.7 – Задачі дослідження

Етапи розробки

Після аналізу актуальності стану питання оптимізації ігрових рушіїв action ігор мобільних додатків та порівняння існуючих аналогів за певним переліком критеріїв було визначено такі завдання:
– створення класу Pool<T>;
– створення статичного класу PoolManager;
– розробка ігрових рушіїв з використанням Pool;
– розробка ассету PoolManager;
– створення Unity Package.

Рисунок Д.8 – Етапи розробки

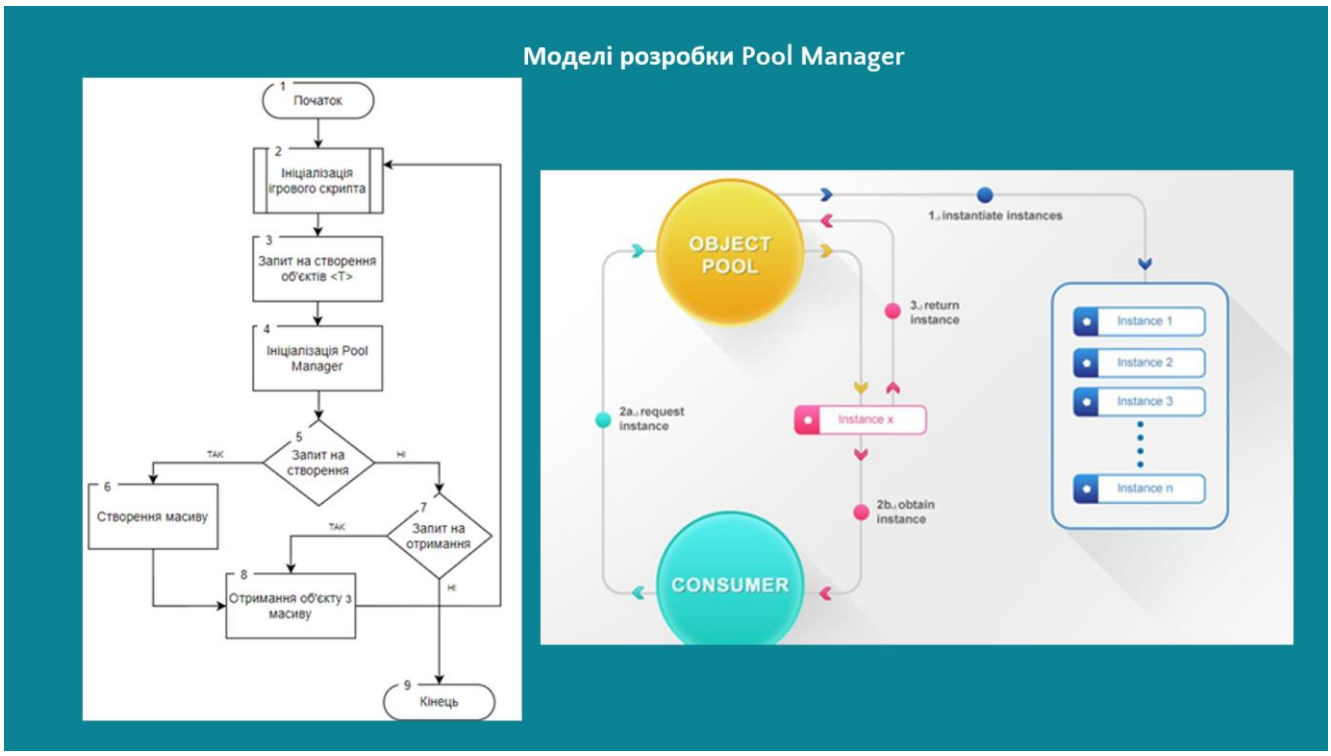

Рисунок Д.9 – Моделі розробки Pool Manager

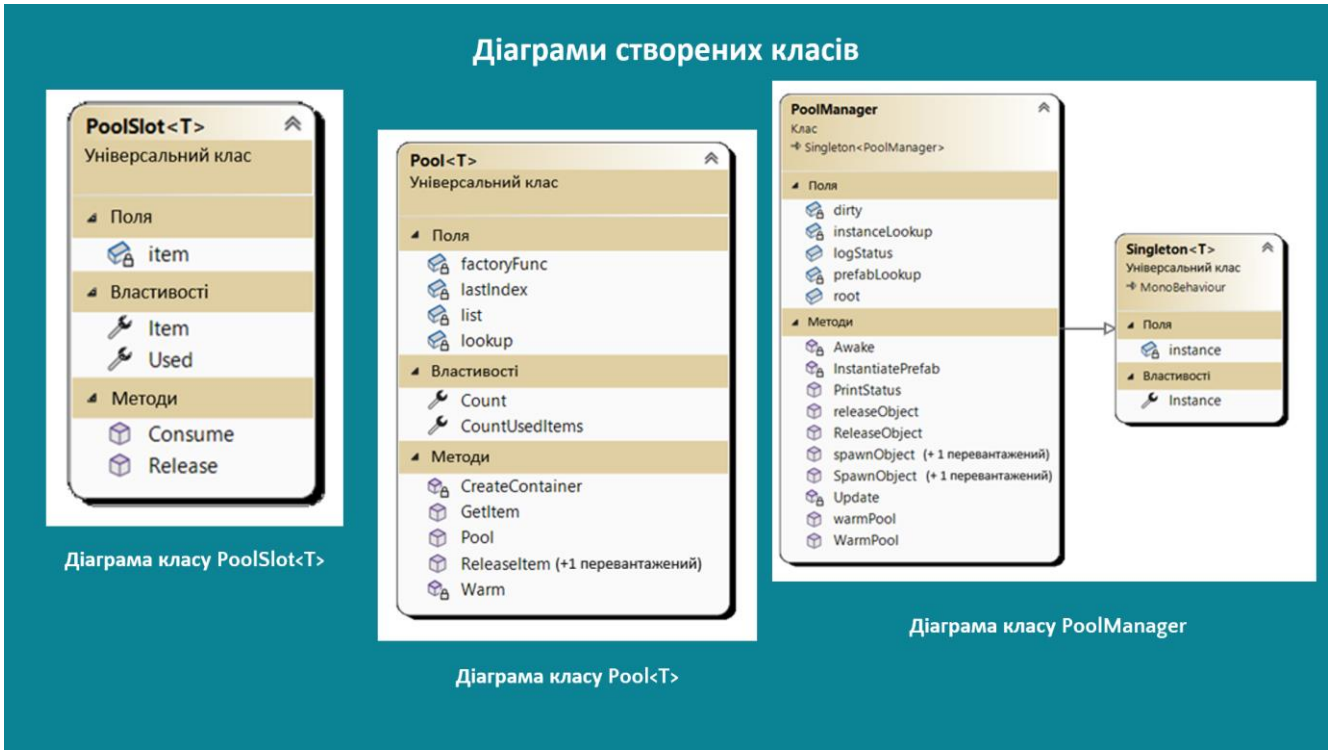

Рисунок Д.10 – Діаграми створених класів

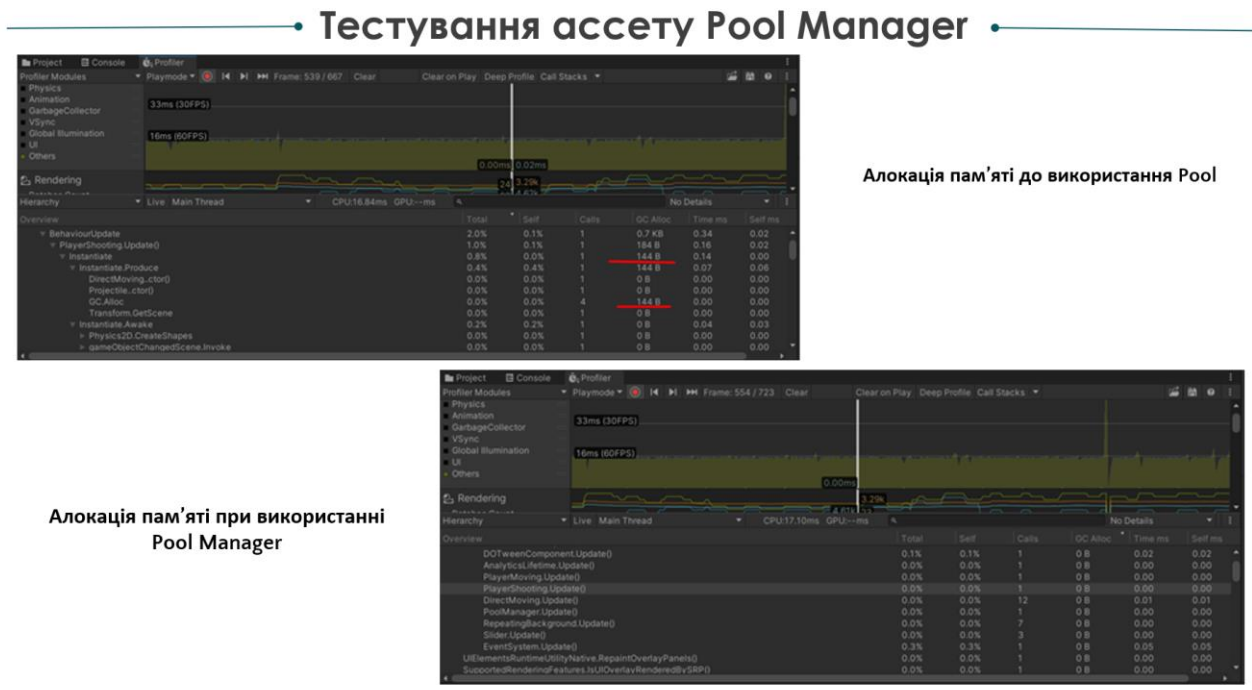

Рисунок Д.11 – Тестування ассету Pool Manager

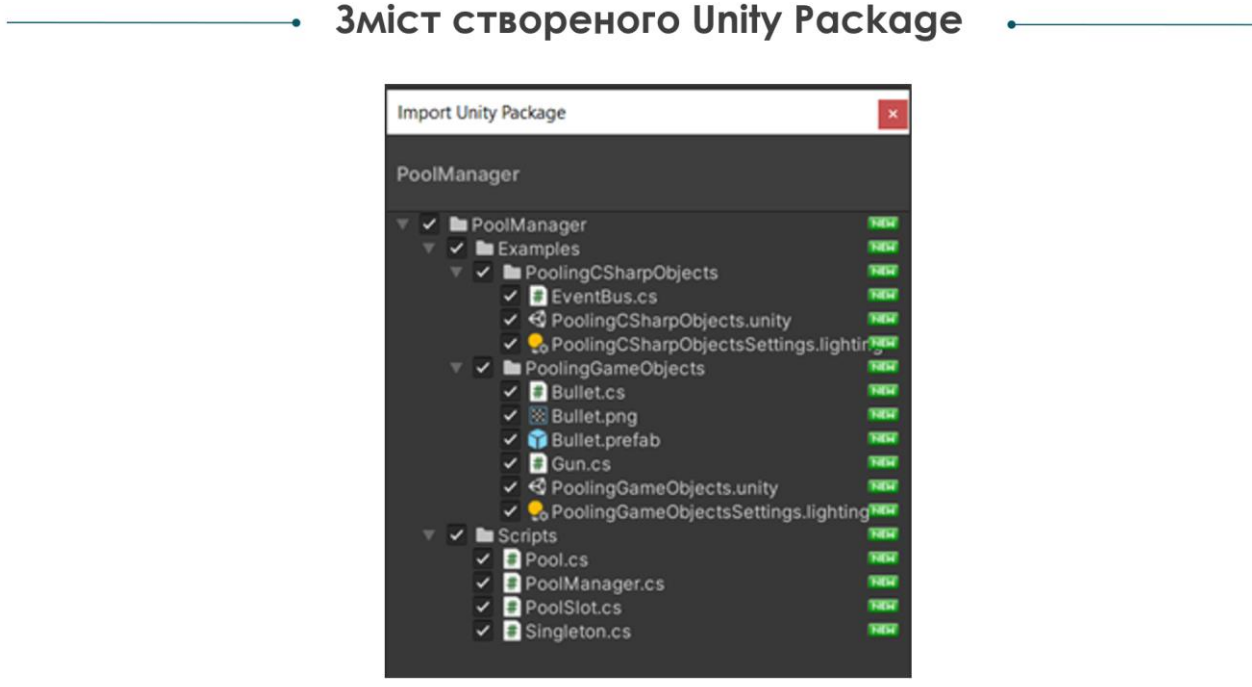

Рисунок Д.12 – Зміст створеного Unity Package

- Проаналізовано існуючі методів оптимізації ігрових рушіїв action ігор мобільних додатків.
- Розглянуто існуючі інструменти реалізації Pool, виявлено їх недоліки та переваги.
- Проаналізовано питання про раціональне використання пам'яті при роботи програми з великою кількістю створюваних об'єктів.
- Складено вимоги до основної функціональності платформи для проведення експериментів.
- Запроектовано архітектуру додатку.
- Розглянуто мови програмування та основні платформи.
- Виконано розробку ассету PoolManager.
- Внесення реалізацій пулу у ігровий додаток.
- Здійснено автоматизоване та ручне тестування ассету.
- Виконано економічний аналіз та розрахунки, результати яких підтвердили доцільність розробки.

Рисунок Д.13 – Результати роботи

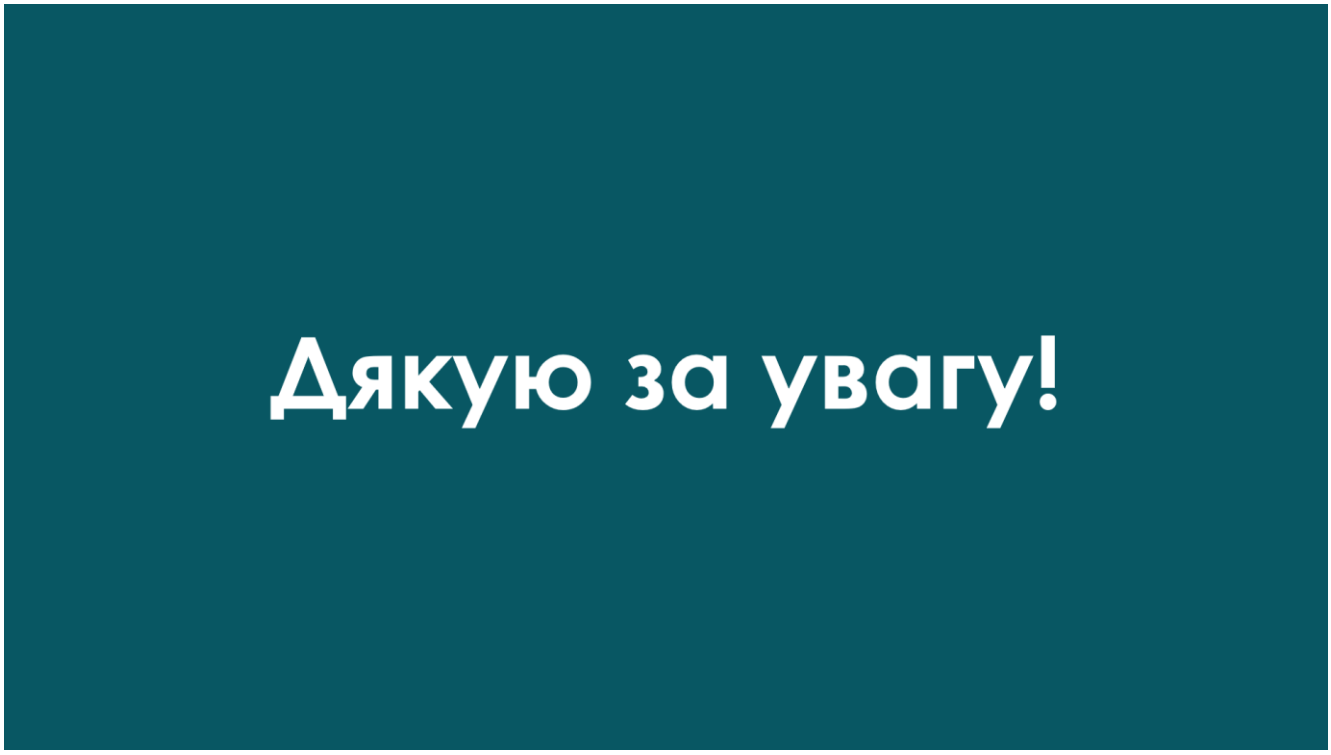

Рисунок Д.14 – Фінальний слайд