МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
Т.в.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

-

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»_____,

освітньо-наукової програми «Інформаційні технології проектування»_____

на тему: «**Інформаційна технологія оптимізації комп'ютерної гри на основі Instanced Static Meshes**»

Здобувача групи _ІТм-11н_ __Проценко Максима Олеговича_____
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ ____Максим ПРОЦЕНКО_____
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник ____к.т.н., доцент Наталія ФЕДОТОВА_____ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-наукова програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. зав. кафедри ІТ

_____ С. М. Ващенко
«_____» __ 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу магістра студентові
Проценко Максим Олегович
(прізвище, ім'я, по батькові)

1 Тема проекту *Інформаційна технологія оптимізації комп'ютерної гри на основі Instanced Static Meshes*

затверджена наказом по університету від «05» травня 2023 р. №*0465-VI*
2 Термін здачі студентом закінченого проекту «17» травня 2023 р.
3 Вхідні дані до проекту технічне завдання на розробку рішення для оптимізації на основі Instanced Static Mesh
4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *Аналіз предметної області; Постановка задачі та аналіз методів дослідження; Моделювання та проектування, Розробка плагіну та тестування*, презентація.
5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *Об'єкт, предмет, мета та гіпотеза; Актуальність і передумови досліджень; Вибір методів реалізації; Процес розробки Програмна реалізація інформаційної технології;Тестування, Висновки; Апробація результатів роботи*.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Аналіз предметної області	15.09.22	
2	Аналіз існуючої проблеми	30.09.22	
3	Аналіз аналогів	14.10.22	
4	Визначення мети та задач	25.11.22	
5	Формування вимог	30.12.22	
6	Створення WBS структури проєкту	10.01.23	
7	Створення OBS структури проєкту	17.01.23	
8	Створення календарного плану	31.01.23	
9	Створення скриптів	28.02.23	
10	Тестування скриптів	10.03.23	
11	Виправлення помилок реалізації	31.03.23	
12	Створення тестового проєкту	14.04.23	
13	Інтегрування розробленої системи	21.04.23	
14	Тестування роботи	05.05.23	
15	Презентація	22.05.23	

Магістрант _____ Проценко М.О.

Керівник роботи _____ к.т.н., доц. Федотова Н.А.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Інформаційна технологія оптимізації комп'ютерної гри на основі Instanced Static Meshes».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 21 найменування, додатків. Загальний обсяг роботи − 71 сторінка, у тому числі 42 сторінок основного тексту, 2 сторінки списку використаних джерел, 31 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці рішення для покращення методу оптимізації Instanced Static Mesh (Unreal Engine).

В першому розділі роботи було проведено аналіз проблемної області, виконано порівняння класичного Instanced Static Mesh з рішенням для покращення.

В другому розділі визначено мету, задачі та засоби реалізації для реалізації задумок.

Третій розділ присвячено моделюванню розробленого плагіну.

Результатом роботи є плагін, який являє собою папку з файлами, який може бути легко інтегрований в будь-який проєкт Unreal Engine 5.

Практичне значення полягає у створення плагіну, який покращує використання методу оптимізації Instanced Static Mesh

Ключові слова: Instanced Static Mesh, Unreal Engine, Instanced Static Mesh Holder, плагін, C++, Windows, IOS, Andorid.

ЗМІСТ

ВСТУП

Актуальність. В сучасних іграх кількість різноманітних об’єктів на рівні може досягати десятків тисяч, а кількість полігонів на одній моделі – мільйонів. Ресурси ігрових пристроїв небезкінечні, а тому і виникає потреба зменшення навантаження, які використовує гра. Існують методи оптимізації, які можна застосовувати на різних етапах розробки. При цьому ці методи дуже часто не достатньо оптимізовані і показують не максимально позитивний результат, або дуже складні для реалізації, що звужує сферу його використання.

Комп’ютерна гра – це складна програма, під час роботи якої виконується велика кількість математичних обчислень. Кожен об’єкт у грі має набір своїх унікальних геометричних (форма, кількість полігонів) та фізичних налаштувань (вага, розмір, щільність), оптичних властивостей поверхні об’єкта, які характеризують здатність об’єкта відбивати світло й тінь. Також інколи необхідно враховувати часткове або повне руйнування об’єкта. Кожен із цих аспектів потребує прорахунків хоча б один раз за кожен кадр, які послідовно відображаються на моніторі. Залежно від характеристик гри це може відбуватись 30, 60, 120, й навіть 240 fps (Frame per Second – кадрів за секунду). Чим менше навантаження на систему і чим більше при цьому показники fps – тим сильніше збільшуються шанси гри на фінансовий успіх.

Об’єкт дослідження. Процес оптимізації гри з великою кількістю однакових об'єктів.

Предмет дослідження. Технологія Instanced Static Mesh, як метод оптимізації гри з великою кількістю однакових об’єктів.

Наукова новизна дослідження полягає у покращенні технології Instanced Static Meshes до рівня автоматичної оптимізації та можливості об'єднувати моделі декількох

об'єктів на рівні в один Instanced Static Mesh. Дослідження має на меті знайти ефективний метод оптимізації гри з великою кількістю однакових об'єктів, що впливає на продуктивність комп'ютерних ігор.

Гіпотези дослідження:

1. Використання технології Instanced Static Meshes може покращити продуктивність комп'ютерних ігор з великою кількістю однакових об'єктів.
2. Розроблений метод оптимізації на основі Instanced Static Mesh може знизити кількість об'єктів у грі та покращити її продуктивність.
3. Використання технології Instanced Static Meshes може знизити навантаження на процесор та графічну підсистему гри, що може призвести до збільшення продуктивності гри.
4. Реалізація методу оптимізації на основі Instanced Static Mesh може зменшити час, необхідний для завантаження об'єктів у гру, що може призвести до покращення досвіду користувача та збільшення продуктивності гри.

Мета роботи. Дослідження й покращення Instanced Static Mesh до рівня автоматичної оптимізації та можливості об’єднувати моделі декількох об’єктів на рівні в один Instanced Static Mesh.

Задачі:

– провести аналіз технології Instanced Static Meshes та її впливу на продуктивність комп'ютерних ігор;

– вдосконалити технологію оптимізації на основі Instanced Static Mesh та описати принцип дії створеного рішення;

– реалізувати розроблене рішення в грі та провести порівняльний аналіз продуктивності гри до та після його впровадженн. Зробити висновки про ефективність використання створеного рішення;

Практичне значення. Розроблена інформаційна технологія допоможе розширити зону можливого використання технології Instanced Static Mesh у комп’ютерних іграх та значно спростить процес його інтеграції. Оптимізація відбуватиметься автоматично для обраних класів об’єктів, що кардинально відрізняється від використання класичного Instanced Static Mesh, який можна налаштувати лише окремо для кожного об’єкту на рівні.

Апробація дослідження. Доповідь відбувалася на конференції IMA 2023, за результатами роботи опубліковані тези та подану статтю до розгляду у редакцію журналу IAPGOS (Informatyka, Automatyki, Pomiary w Gospodarce i Ochronie Środowiska).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

На цей час, стосовно Instanced Static Mesh в інтернеті майже відсутні публікації, окрім офіційної документації Epic Games [1], в якій відсутня інформація про те як використовувати дану технологію, і вона містить лише описання методів. Також, існують лише поодинокі публікації на форумах та відео на YouTube, які і являються основним джерелом інформації для людини, яка намагається використати Instanced Static Mesh в своєму проєкті.

Після дослідження більшості наявних публікацій стосовно Instanced Static Mesh (далі – ISM) – стало зрозуміло, що ISM – це компонент актору, який додається на сцену в UE. Він являє собою модель, яка створюється один раз, але відображається на сцені декілька разів (кожен раз зі своєю позицією, поворотом та масштабом). Цим самим досягається величезна економія пам’яті та інших ресурсів ПК, за рахунок одноразового прорахування моделі.

Найбільш явна проблема ISM – це саме те, що він є компонентом актору – тобто, його можна прив’язати лише до якогось конкретного об’єкту. Наприклад, ми можемо створити будинок, ISM якого стануть одинакові цеглини, які просто розміщені в різних місцях і цим самим створюють візуалізацію цього будинку. Але при цьому ми не можемо розмістити одинакові червоні бочки на сцені, які являються різними об’єктами, і при цьому об’єднати їх в один ISM. Саме вирішення цього недоліку і є основною проблемою дослідження.

Через брак інформації про ISM вирішено аналізувати всю область використання методів оптимізації, щоб зрозуміти, які показники методів оптимізації мають найбільшу роль під час їх вибору до застосування у проєкті.

Але вже під час проведення аналізу було визначено, що все залежить від конкретної гри і від результату, якого розробники бажають досягти. Якщо гра дуже маленька – то її взагалі можна не оптимізувати. Якщо обсяг середній – можна використати найпростіші методи оптимізації, такі як Distance culling, Occlusion culling та Frustrum culling [2]. У випадку, якщо проєкт великий, з великою кількістю різноманітних об’єктів та великими мапами – краще використати усі методи оптимізації. При використанні великої кількості однотипних об’єктів на сцені, з високою деталізацією – варто задуматись про використання Nanite System [3]. У деяких випадках лише ця система може збільшити fps в декілька разів.

Якщо звертати увагу на якусь конкретну ситуацію, то Кріс Дікинсон у своїй книзі [4] описує випадок, коли йому з командою довелось зайнятись оптимізацією однією гри. Саме в тому випадку вдалося збільшити середній фреймрейт в грі з 30 до 120 fps (випробування проводились на одній і тій же машині з одними і тими ж налаштуваннями).

В дослідженні Intel [5], яке вони проводили з рушієм Unreal Engine 4, йдеться про приріст fps в 12,2%. Оптимізація проводилась лише шляхом оптимізації моделей та текстур. Такий же принцип оптимізації використовувався у роботі Альберто Альвареза [6], де результатом для простої 2D-гри стали лише 5% покращення fps.

В деяких специфічних іграх дуже часто використовуються такі ж специфічні методи оптимізації, як наприклад описується в статті Hai Xu [7] з описом оптимізації 3D-міста в Unreal Engine 5. Він описує цікаву систему зміну моделей міста на об’ємну мапу для оптимізації, що в рамках його проєкту дало величезний приріст до кількості кадрів (більше ніж на 224%).

1.2 Аналіз програмних продуктів-аналогів

Порівняти розроблену систему для оптимізації з іншими методами оптимізації ігрових додатків неможливо, бо кожен з них використовується в залежності від особливостей гри (положення камери, кількість об’єктів, динамічність, якість моделей і т.д.). Але ми можемо порівняти власну систему з початковою версією Instanced Static Meshes. Порівняльний аналіз наведено в табл.1.1.

Як бачимо, єдиний мінус Instanced Static Meshes Holder (системи, яка розроблюється в рамках даного дослідження) – це те, що він не доступний відразу після створення проєкту. Також, важливим фактом є те, що ні класичний Instanced Static Mesh, ні Instanced Static Meshes Holder не можуть підтримувати фізику. Скоріше за все, підтримку фізики можна реалізувати, але це потребує окремого дослідження.

Таблиця 1.1. - Порівняльний аналіз

Категорії для порівняння / Методи	Стандартний Instanced Static Mesh	Instanced Static Meshes Holder
Автоматичне додавання нових моделей для інстансу	-	+
Можливість об’єднувати різні об’єкти (актори) на сцені в один інстанс	-	+
Можливість додати абсолютно різні моделі	-	+
Можливість руху моделей за таймлайном (вручну)	+	+
Вплив фізики на моделі в інстансі	-	-
Відразу доступний в проєкті UE5	+	-
Автоматичне додавання нових моделей для інстансу	-	+

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Мета роботи є дослідження й покращення Instanced Static Mesh до рівня автоматичної оптимізації та можливості об’єднувати моделі декількох об’єктів на рівні в один Instanced Static Mesh.

Відповідно до поставленої мети необхідно вирішити такі взаємозв’язані задачі:

- провести аналіз технології Instanced Static Meshes та її впливу на продуктивність комп'ютерних ігор;
- покращити метод оптимізації на основі Instanced Static Mesh та описати його принцип дії;
- реалізувати розроблений метод в грі та провести порівняльний аналіз продуктивності гри до та після впровадження методу;
- зробити висновки про ефективність використання технології Instanced Static Meshes у процесі оптимізації комп'ютерних ігор.

Розроблена система для оптимізації має містити такі функції:

- можливість розробнику самостійно визначати список класів, об’єкти яких будуть оптимізуватись при додаванні на рівень;
- можливість додати до одного інстансу моделі різних об’єктів;
- автоматична оптимізації після старту гри;
- можливість додати безкінечну кількість об’єктів класу і безкінечну кількість класів для оптимізації.

Об’єкт дослідження. Процес оптимізації гри з великою кількістю однакових об'єктів.

Предмет дослідження. Технологія Instanced Static Mesh, як метод оптимізації гри з великою кількістю однакових об’єктів.

Гіпотези дослідження:

1. Використання технології Instanced Static Meshes може покращити продуктивність комп'ютерних ігор з великою кількістю однакових об'єктів.
2. Розроблений метод оптимізації на основі Instanced Static Mesh може знизити кількість об'єктів у грі та покращити її продуктивність.
3. Використання технології Instanced Static Meshes може знизити навантаження на процесор та графічну підсистему гри, що може призвести до збільшення продуктивності гри.
4. Реалізація методу оптимізації на основі Instanced Static Mesh може зменшити час, необхідний для завантаження об'єктів у гру, що може призвести до покращення досвіду користувача та збільшення продуктивності гри.

2.2 Методи дослідження.

Для дослідження ефективності використання технології Instanced Static Meshes у процесі оптимізації комп'ютерних ігор були використані наступні методи:

- Експериментальний метод - проведено порівняльний аналіз продуктивності гри до та після впровадження методу. При цьому вимірюється час завантаження гри, кількість кадрів в секунду (fps) та кількість ресурсів, які використовуються для відображення гри.

- Експертний метод було використано як думка експертів у галузі розробки комп'ютерних ігор щодо використання технології Instanced Static Meshes та її впливу на продуктивність гри (проєкт завантажено на GitHub та надано доступ директору однієї з IT-компаній).

2.3 Вибір технологій

Після проведення аналізу актуальності роботи, вивчення найпопулярніших методів оптимізації та визначення методу, який можна оптимізувати, стало зрозуміло і які технології будуть використовуватись.

Так як Instanced Static Mesh – це технологія Unreal Engine, то і робота виконувалась на основі цього двигуна, і була обрана остання версія двигуна на момент початку написання роботи (Unreal Engine 5.0.2) [8].

Для написання логіки покращення методу оптимізації для UE існують 2 варіанти: C++ та Blueprints (візуальна мова програмування, вбудована в Unreal Engine починаючи з 4 версії). Так як Blueprints має дуже багато мінусів (зміни не відображаються у системах контролю версій, відсутній пошук по використаним методам, вище навантаження на систему, в порівнянні з C++), було обрано C++ 11 версії [9]. Саме цей стандарт C++ підтримує UE, додаючи певний функціонал C++.

Середовищем розробки для написання коду було обрано Rider for Unreal Engine [10], так як він є більш зручним, ніж Visual Studio (яка в свою чергу також використовувалась для генерування бінарних файлів для UE).

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

3.1 Структурно-функціональне моделювання процесу

Для виконання етапу моделювання та проектування систему були розроблені діаграми нотації IDEF0, які описують процес використання розробленої системи, та сутності, які впливають на цей процес.

Діаграми нотації IDEF0

IDEF0 – це графічна нотація, яка призначена для опису бізнес-процесів. На першому етапі проєкт представляється у вигляді прямокутника, в середині якого описано одним реченням основний його функціонал. Біля нього малюються стрілки, які являють собою вхідні дані для користування продуктом (зліва), вихід (справа), стрілки управління (зверху) та механізми, за допомогою яких виконується функціонал (знизу).

На рисунку 3.1 зображена контекстна діаграма процесу оптимізації гри для технології оптимізації гри на основі Instanced Static Meshes.

Наступний крок для моделювання проєкту – це створення діаграми декомпозиції першого рівня. Вона відрізняється від попередньої розбиттям головного процесу на блоки і поєднанням стрілок з блоками, до яких вони відносяться.

Діаграма декомпозиції першого рівня була також створена і має наступний вигляд (рис 3.2).

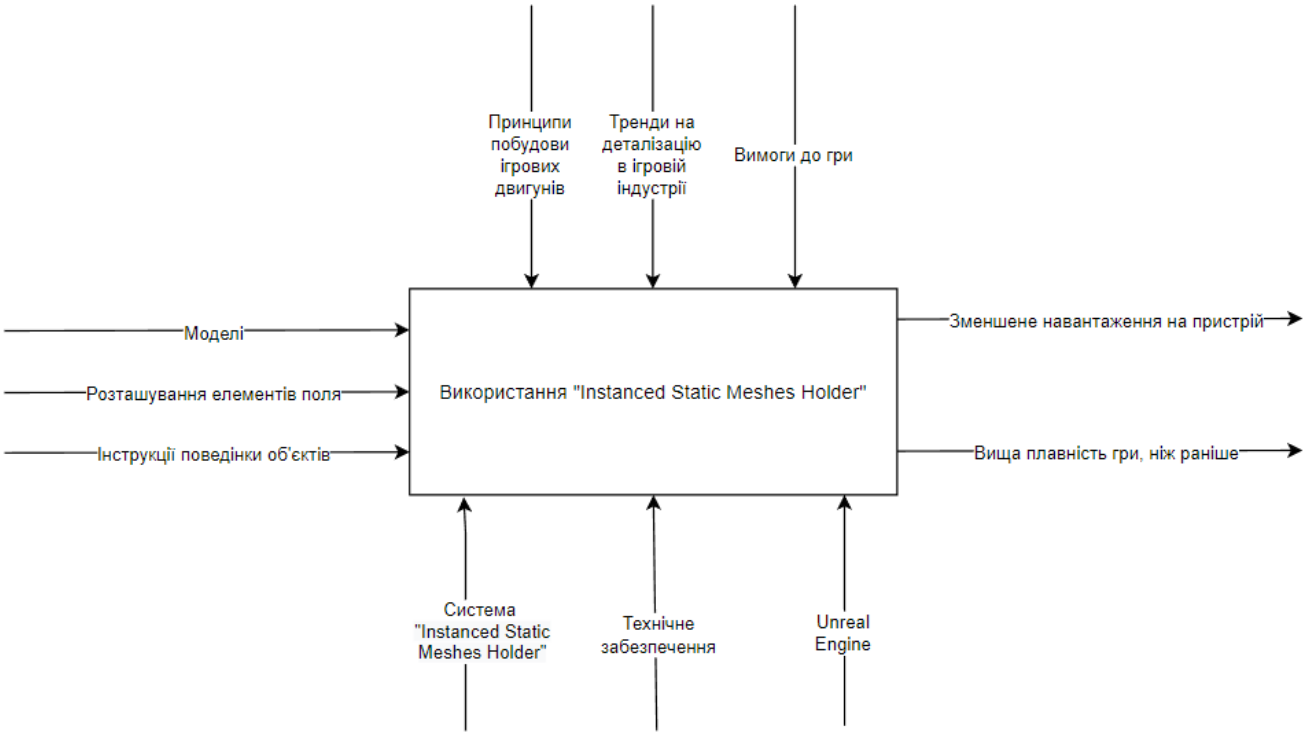


Рисунок 3.1 – Контекстна діаграма IDEF0

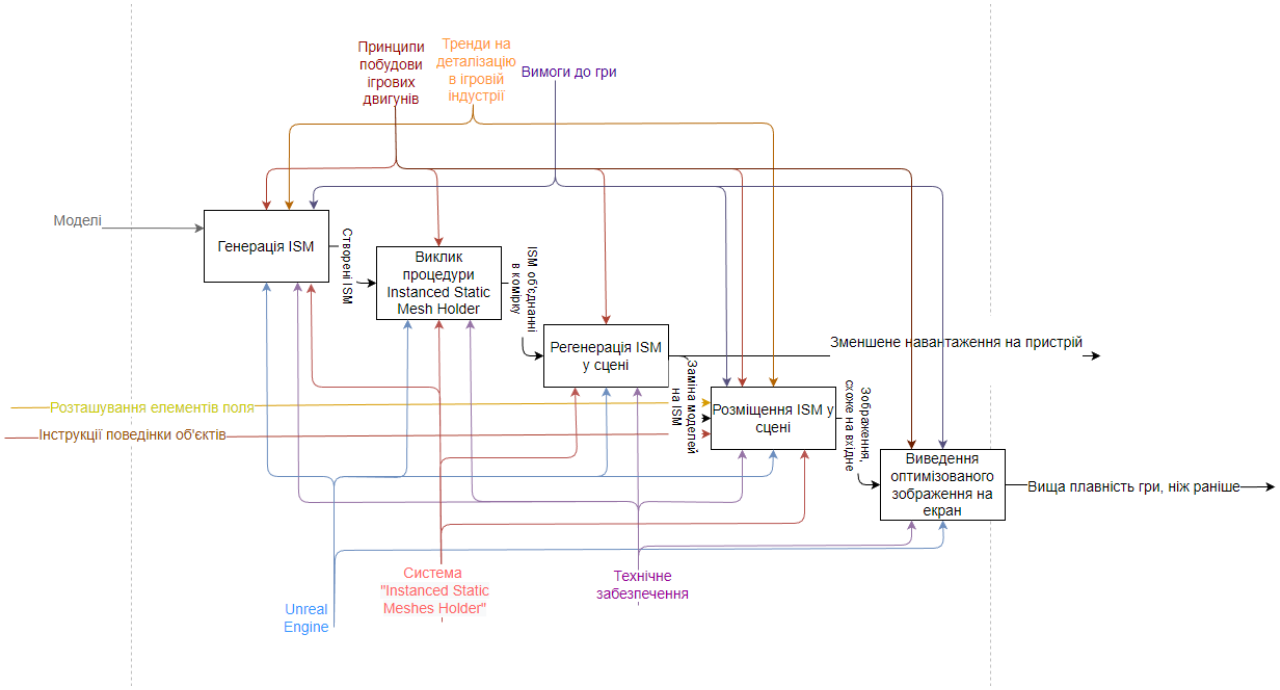


Рисунок 3.2 – Контекстна діаграма декомпозиції першого рівня

3.2 Діаграма варіантів використання

Діаграма варіантів використання відображає взаємодію зовнішніх об’єктів з розробленим продуктом. В нашому випадку було визначено 3 актора:

1. Користувач.
2. Instanced Static Mesh Holder.
3. Unreal Engine

Також, було визначено 6 сценаріїв використання продукту:

1. Створення вихідної сцени.
2. Генерація ISM.
3. Виклик логіки класів.
4. Регенерація моделей в ISM.
5. Розміщення ISM на сцені.
6. Виведення оптимізованого зображення на екран.

Діаграма варіантів використання наведена нижче (рис.3.3):

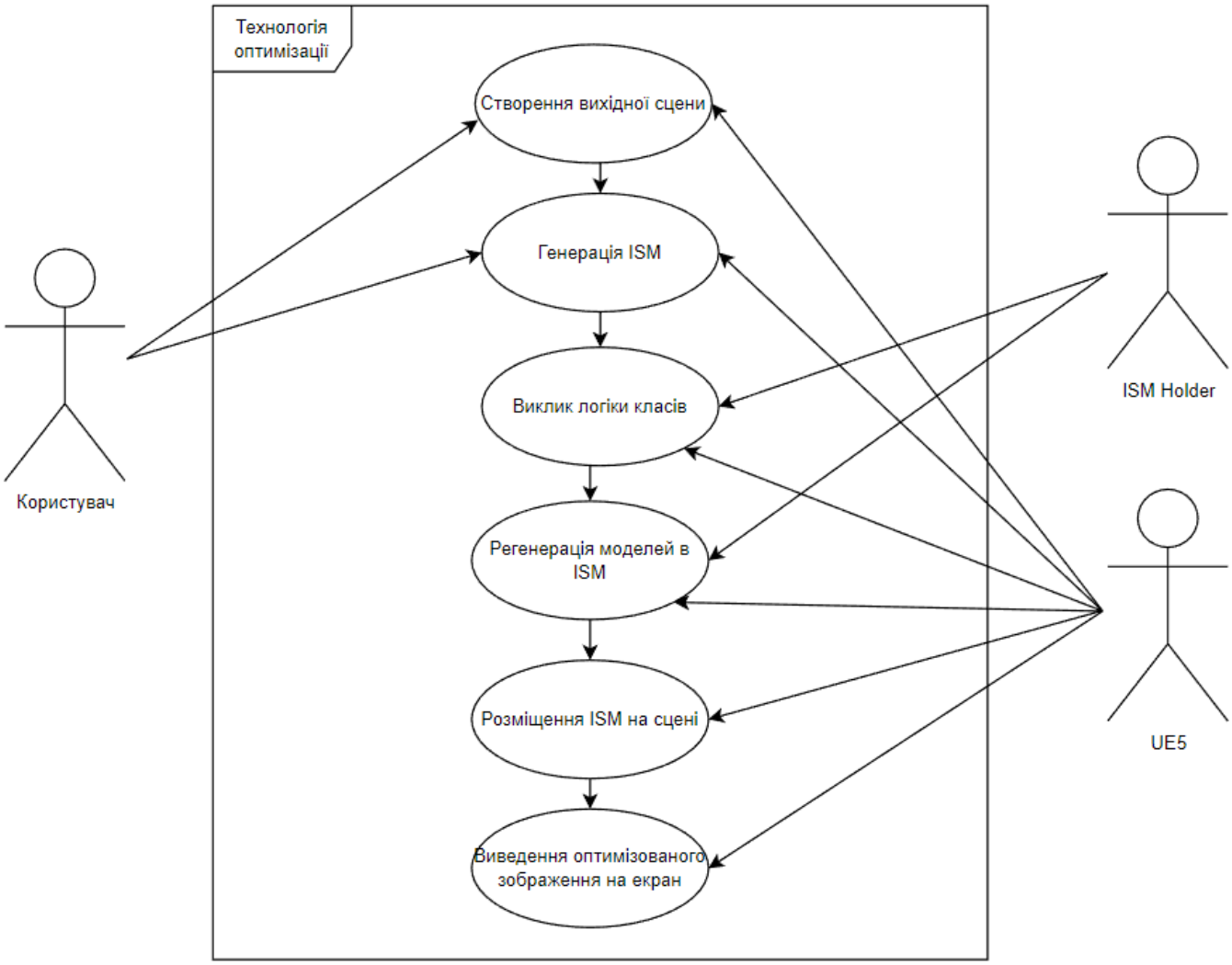


Рисунок 3.3 – Діаграма варіантів використання

4 ПРАКТИЧНА РОЗРОБКА РІШЕННЯ

4.1 Розробка плагіну Instanced Static Mesh

Процес оптимізації можна описати за допомогою блок-схеми (рис.4.1).

Плагін спочатку створює контейнер ISM. Потім він створює ISM для кожного типу моделі, яку потрібно оптимізувати і додає ці ISM до контейнера. Після додавання кожного ISM, плагін отримує підтвердження про успішне додавання ISM до контейнера.

Далі, починається гра і плагін передає контейнеру ISM інформацію про моделі, які були додані до сцени та можуть бути оптимізовані. Контейнер ISM виконує оптимізацію і повідомляє плагін про успішну заміну ISM.

Контейнер ISM повертає зворотній конвертований Mesh плагіну,та зберігає додаткові дані для подальшої зворотної конвертації ISM в Mesh за потреби.

Під час гри клієнт може викликати функцію для видалення ISM з контейнера. Плагін передає цей запит контейнеру ISM, який видаляє ISM з контейнера і повідомляє плагін про успішне видалення ISM.

Instanced Static Mesh - це компонент в Unreal Engine, який дозволяє відображати багато копій однієї моделі на сцені, використовуючи лише один екземпляр самої моделі в пам'яті. Це дає можливість ефективно відображати велику кількість однакових об'єктів на сцені без зайвого навантаження на систему. Крім того, Instanced Static Mesh можна маніпулювати в реальному часі, змінюючи позиції та параметри окремих інстансів. Instanced Static Mesh дозволяє підвищити продуктивність гри та зменшити час розробки завдяки ефективному використанню пам'яті та ресурсів.

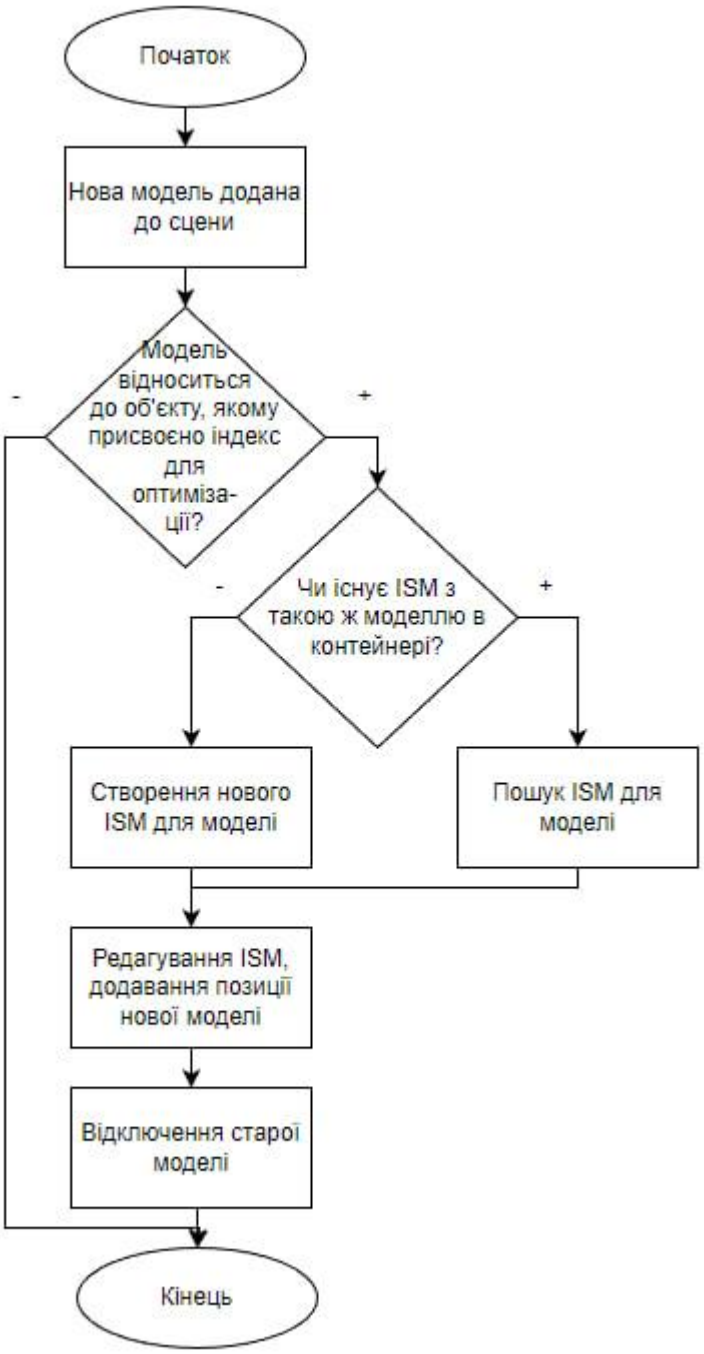


Рисунок 4.1 – Процес оптимізації

Для швидкої і простої інтеграції покращеного Instanced Static Mesh краще помістити його в плагін, який можна просто буде скопіювати в папку з плагінами в будь-якому проєкті.

Для того, щоб створити новий плагін для Unreal Engine необхідно спочатку створити пустий проєкт (рис.4.2), та вже на його основі виконувати необхідні дії [11].

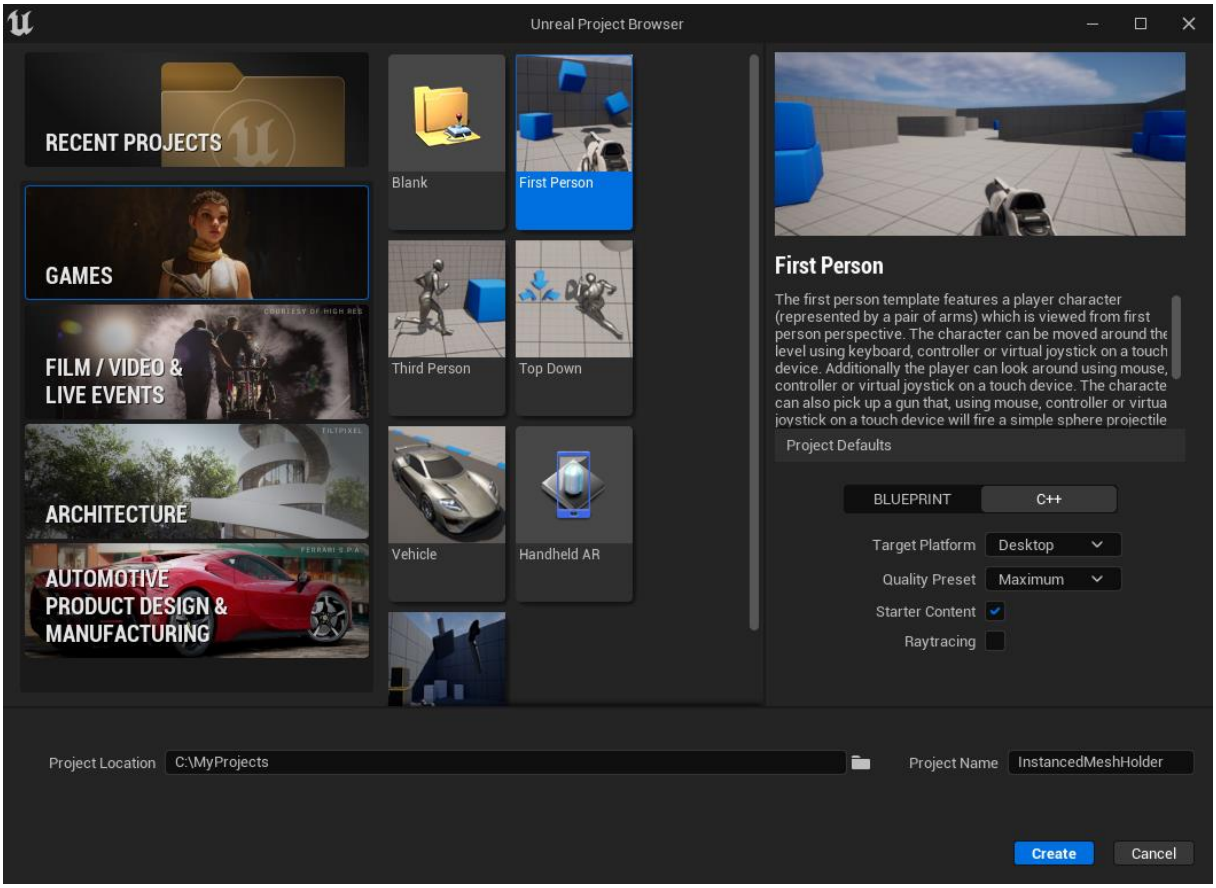


Рисунок 4.2 – Створення проєкту

Коли проєкт створився, можемо перейти до створення нового плагіну (рис.4.3).

Після створення плагіну, можемо відкрити програмний код проєкту. У папці з плагінами створилася папка нашого плагіну зі стандартною структурою папок Unreal

Engine (Private/Public секції). В середині цих папок створимо необхідні класи, які будуть архівовані в плагін далі [12]. Створені класи наведено на рисунку 4.3.

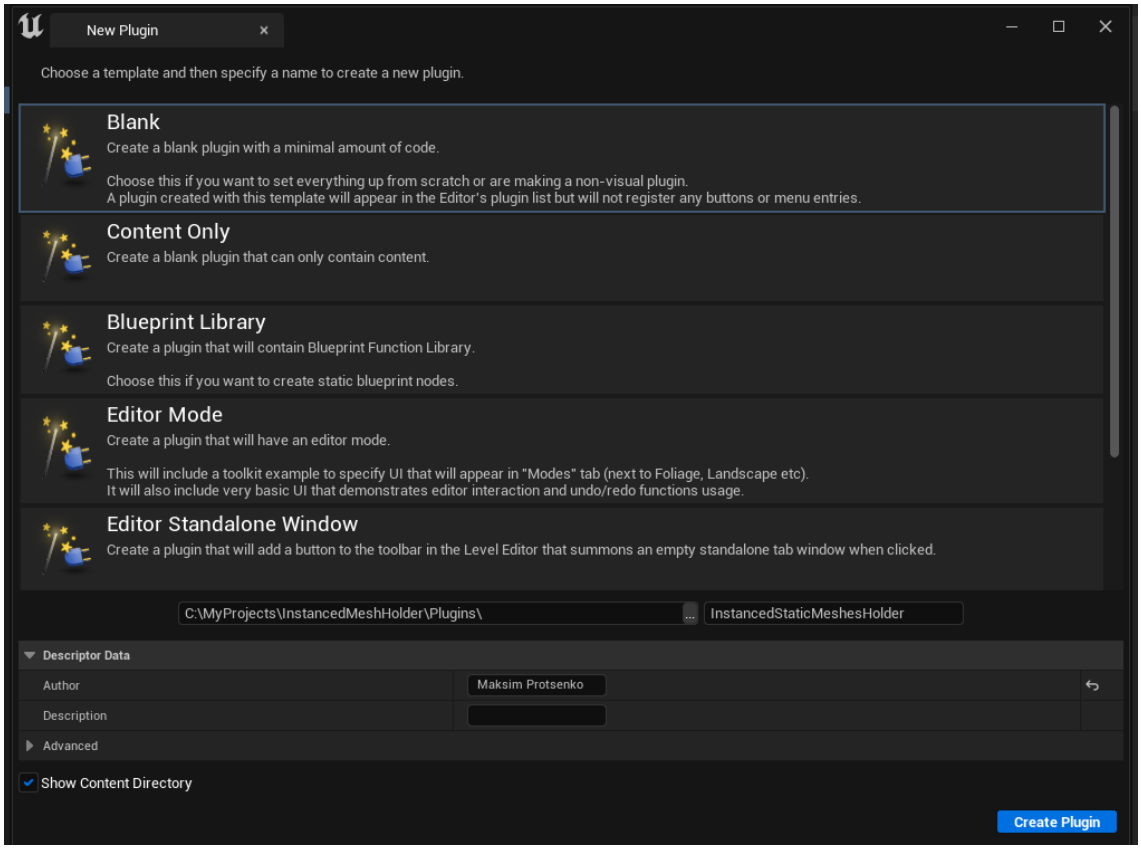


Рисунок 4.3 – Створення плагіну

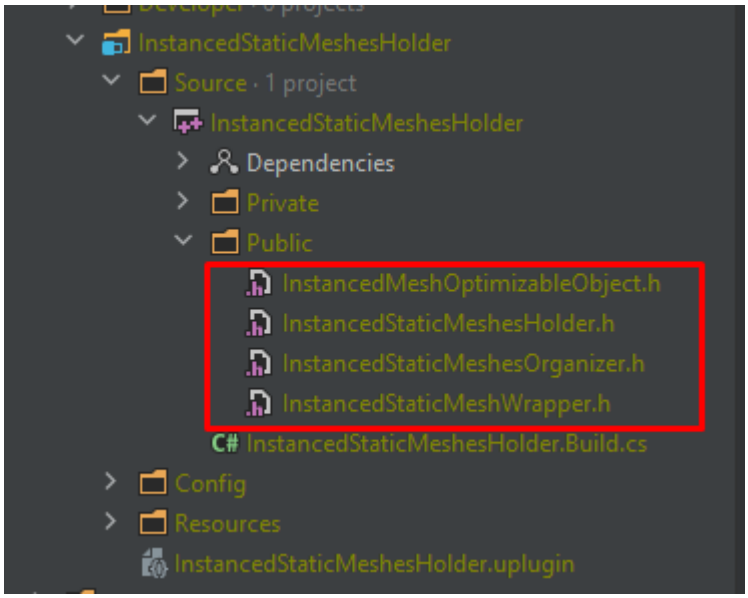


Рисунок 4.4 – Демонстрація створених класів в середині плагіну

Перейдемо до заповнення цих класів. AInstancedMeshOptimizableObject – це об’єкт, який буде оптимізуватись. Декларуємо йому те, що він обов’язково повинен мати Static Mesh Component, бо це саме те, що оптимізується за допомогою Instanced Static Mesh. Також, він повинен мати індекс – це ідентифікатор об’єкта в середині Instanced Static Mesh Holder. За допомогою його можна звернутись до інстансу і перемістити відображення конкретного об’єкта, якому відповідає цей індекс. Також, додамо bool змінну для швидкого відключення оптимізації [13] (рис. 4.5).

Для того, щоб зручно було використовувати Instanced Static Mesh – довелось створити клас-обкладинку із розширеним функціоналом для нього. Функціонал цього класу передбачає методи з додавання та віднімання конкретних об’єктів з інстансів, та включення/відключення оптимізації для всіх об’єктів, які були оптимізовані за допомогою Instanced Static Mesh Component. Також, він зберігає інформацію про всі оптимізовані об’єкти та їх індекси для доступу до їх зображення в будь-який час [14] (рис.4.6-4.7).

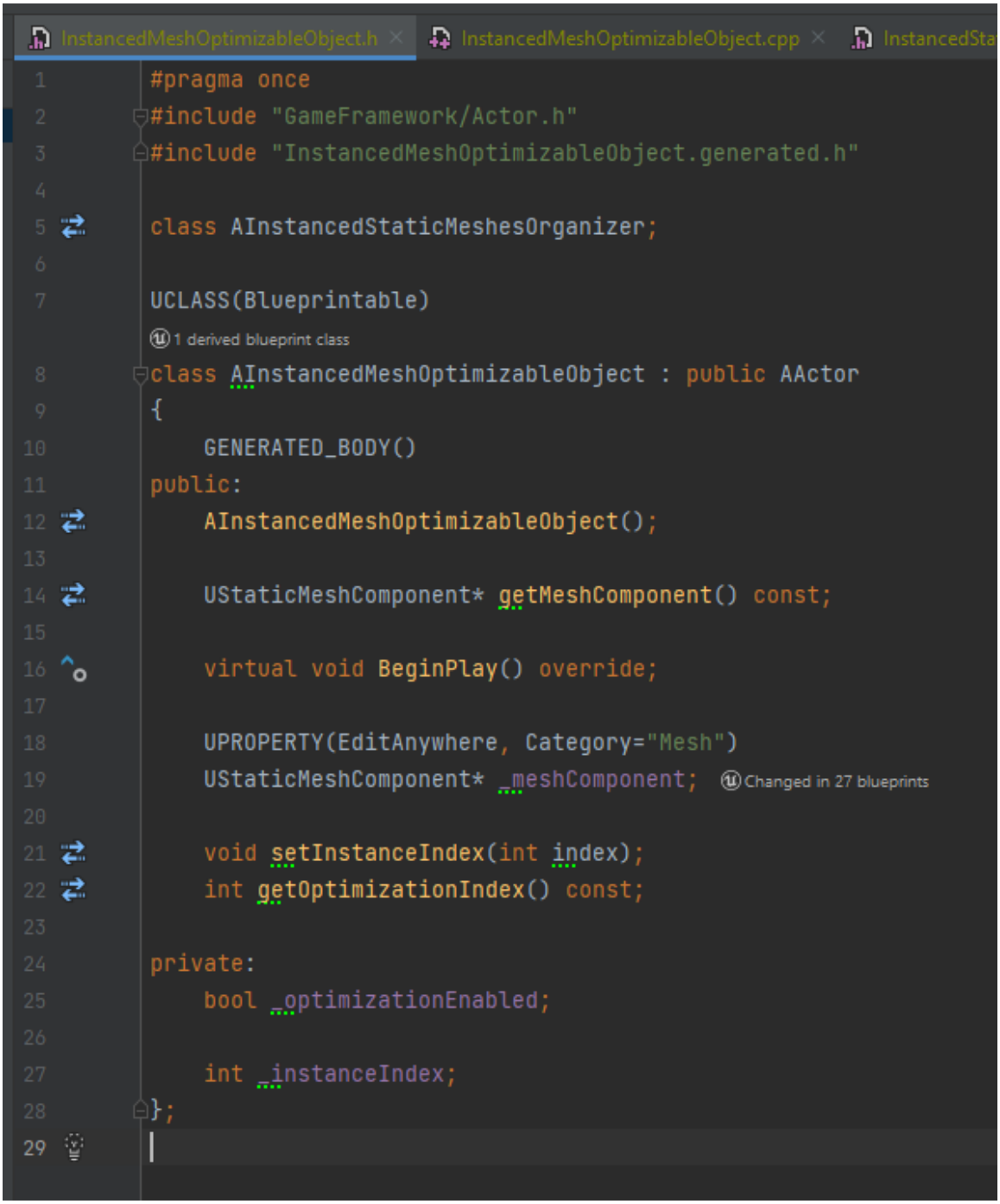


Рисунок 4.5 – Декларація класу InstancedMeshOptimizableObject

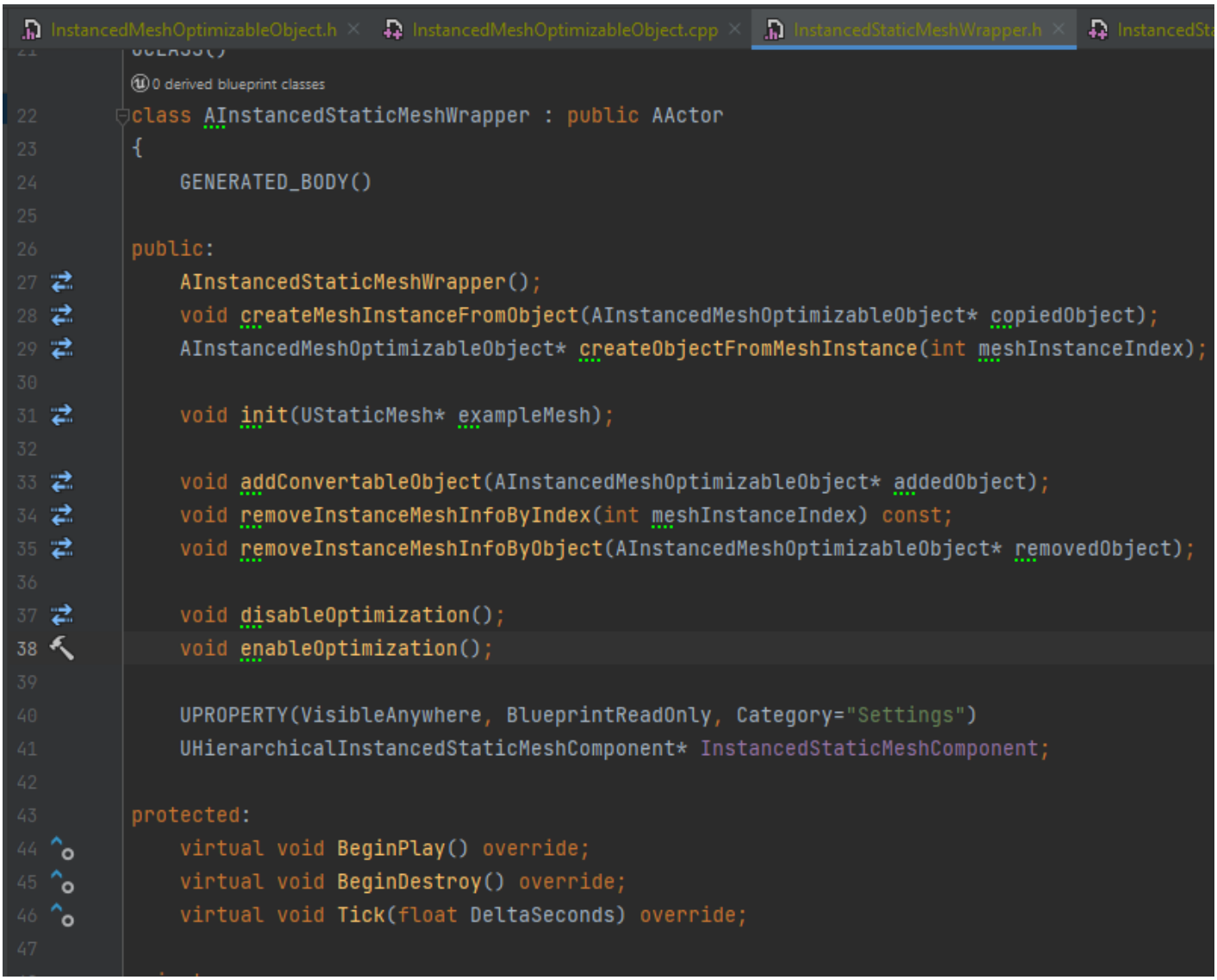
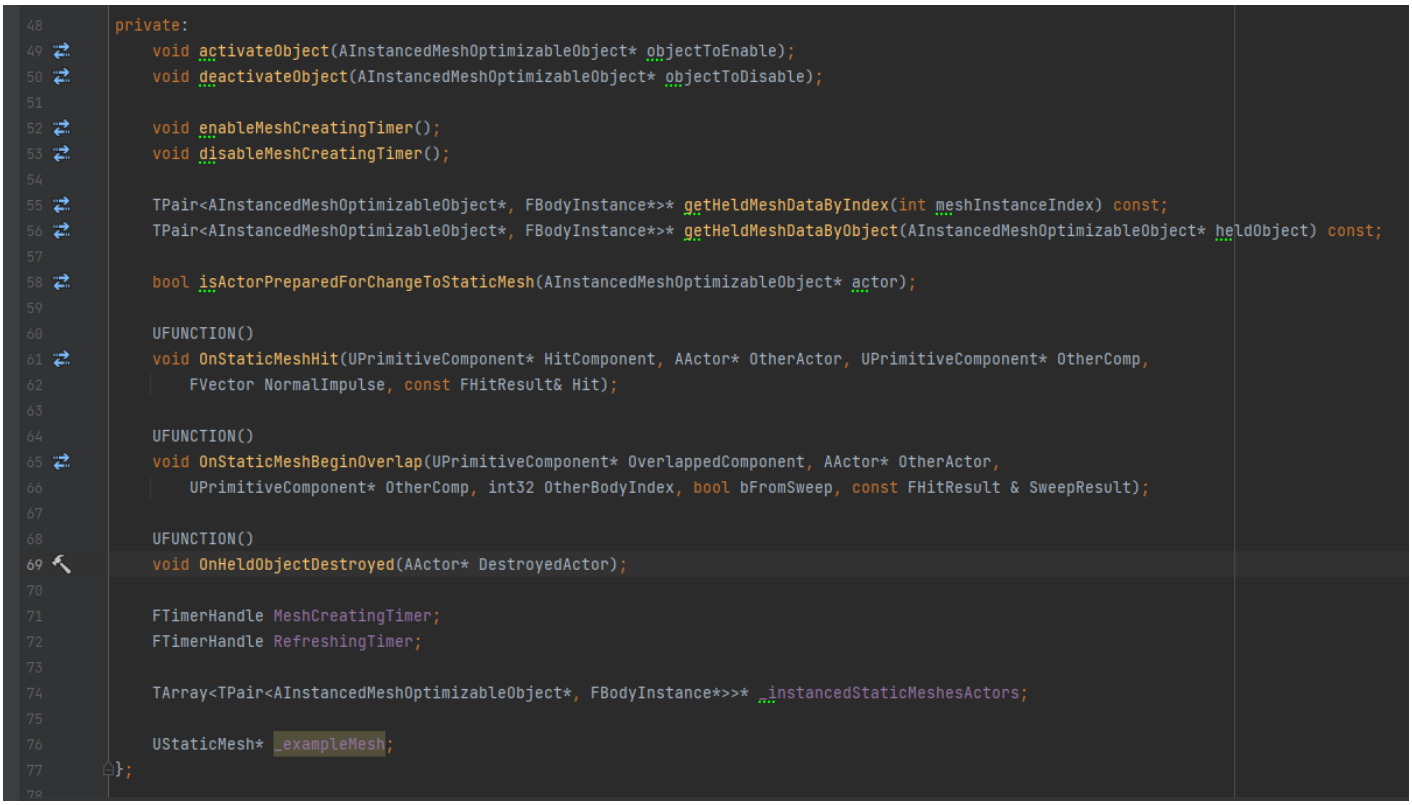


Рисунок 4.6 – Декларація класу InstancedStaticMeshWrapper



```
private:
    void activateObject(AInstancedMeshOptimizableObject* objectToEnable);
    void deactivateObject(AInstancedMeshOptimizableObject* objectToDisable);

    void enableMeshCreatingTimer();
    void disableMeshCreatingTimer();

    TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>* getHeldMeshDataByIndex(int meshInstanceIndex) const;
    TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>* getHeldMeshDataByObject(AInstancedMeshOptimizableObject* heldObject) const;

    bool isActorPreparedForChangeToStaticMesh(AInstancedMeshOptimizableObject* actor);

    UFUNCTION()
    void OnStaticMeshHit(UPrimitiveComponent* HitComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp,
        FVector NormalImpulse, const FHitResult& Hit);

    UFUNCTION()
    void OnStaticMeshBeginOverlap(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor,
        UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult & SweepResult);

    UFUNCTION()
    void OnHeldObjectDestroyed(AActor* DestroyedActor);

    FTimerHandle MeshCreatingTimer;
    FTimerHandle RefreshingTimer;

    TArray<TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>>* _instancedStaticMeshesActors;

    UStaticMesh* _exampleMesh;
};
```

Рисунок 4.7 – Продовження декларації класу InstancedStaticMeshWrapper

Останній клас – це AInstancedStaticMeshOrganizer. Цей клас керує всіма інстансами, які були створені для оптимізації об’єктів. Для оптимізації кожної нової моделі створюється новий Instanced Static Mesh Wrapper, який може зберігати в собі безкінечну кількість об’єктів з такою ж моделлю (для кожної нової моделі яка оптимізується створюється власний Instanced Static Mesh Wrapper).

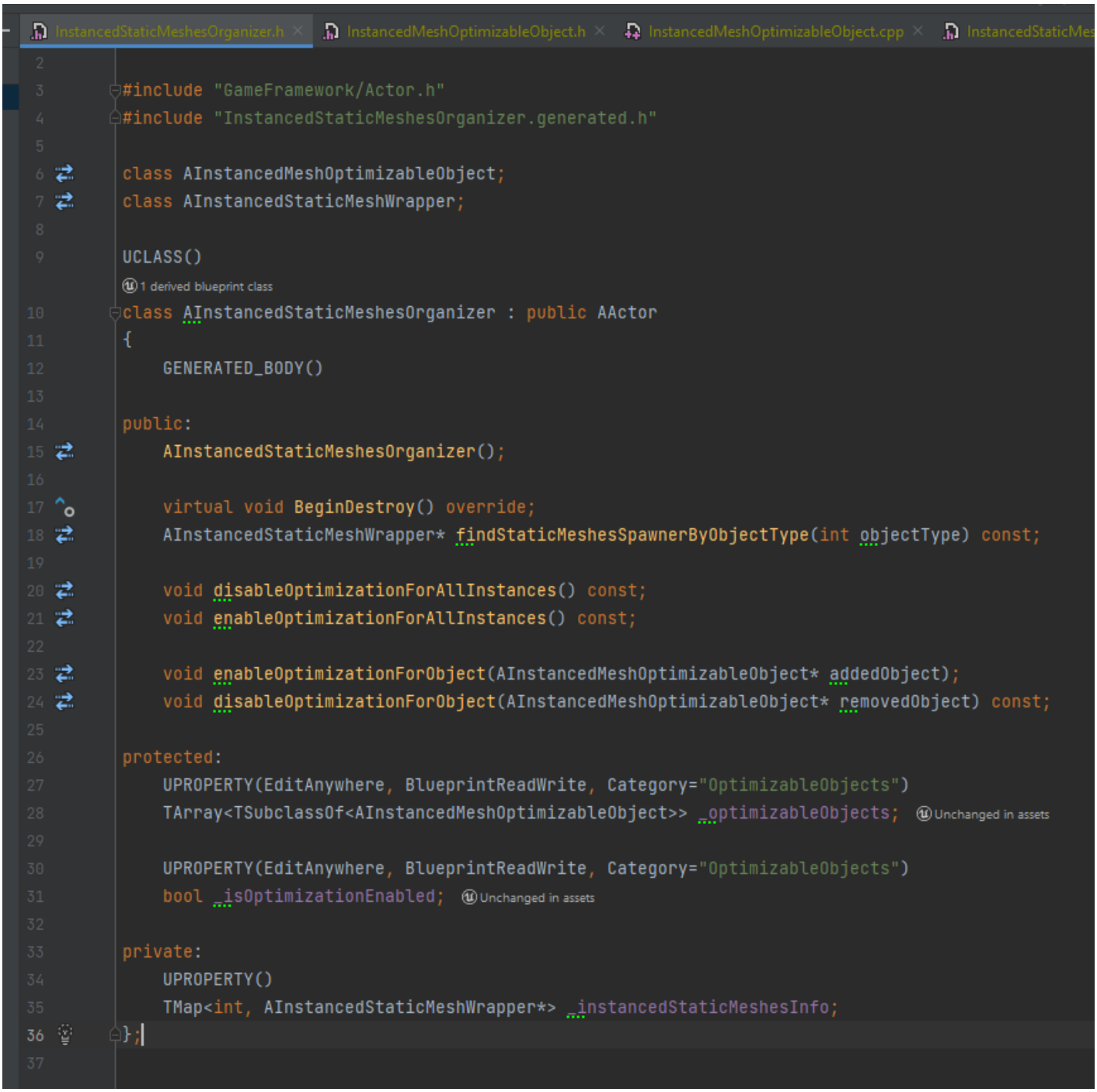


Рисунок 4.8 – Декларація класу InstancedStaticMeshesOrganizer

Для того, щоб тримач інстансів можна було розмістити на рівні, необхідно створити блюпринт-класс, з наслідуванням від Instanced Static Mesh Organizer. При інтеграції в інший проєкт, цій сутності необхідно задавати список класів, які будуть оптимізуватись після старту гри [15]. В проєкті цей клас має такий вигляд:

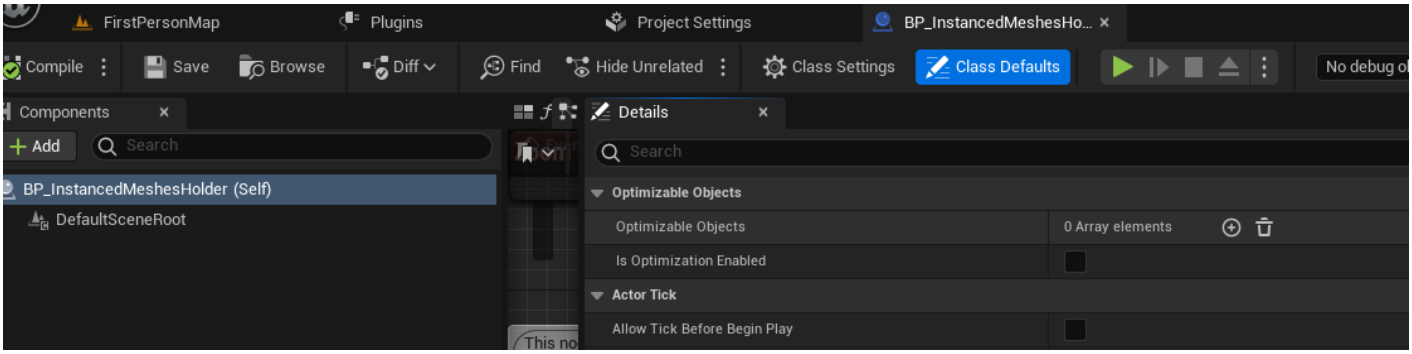


Рисунок 4.9 – Вигляд BP_InstancedMeshesHolder

Програмний код проєкту готовий. Залишилось запакувати плагін для інтеграції і тестування його в іншому проєкті.

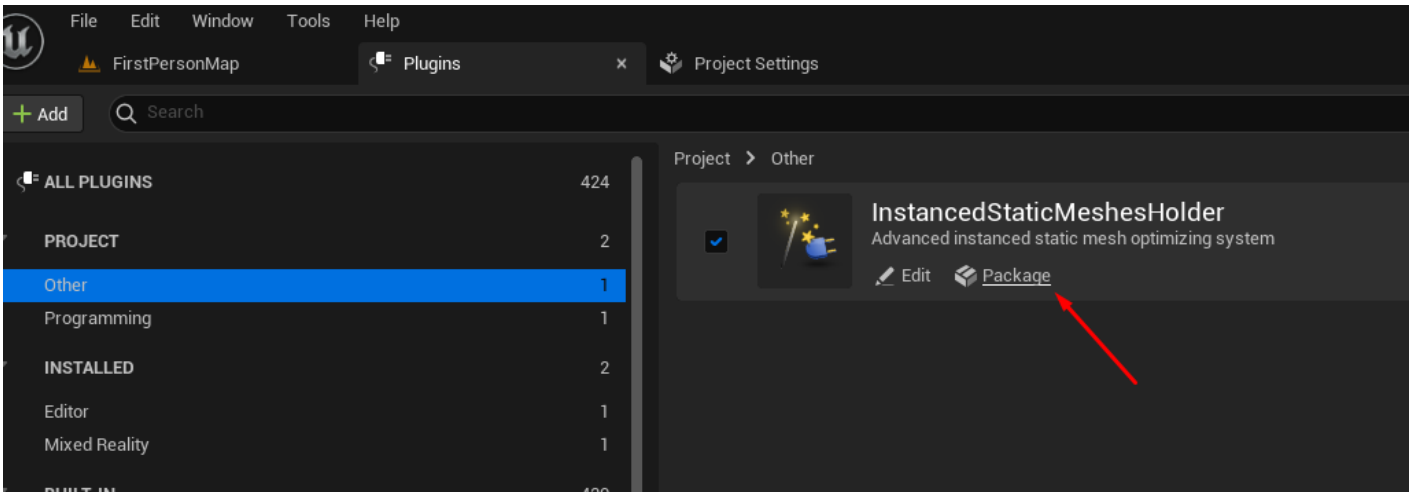


Рисунок 4.10 – Пакування плагіну

При спробі запакувати плагіна з’являлось багато помилок через те, що особливі файли плагіну потребують більше заголовних файлів та відмовляються працювати якщо в *.h-файлах присутні зайві “include”, що викликає проблему циклічного включення. Також, плагін створюється не тільки під платформу Windows, а ще й для IOS та Andorid [16, 17, 18], що викликало свої специфічні помилки. Необхідно додатково встановити необхідні фреймворки для вирішення помилок. В результаті, було отримано готовий плагін (рис.4.11):

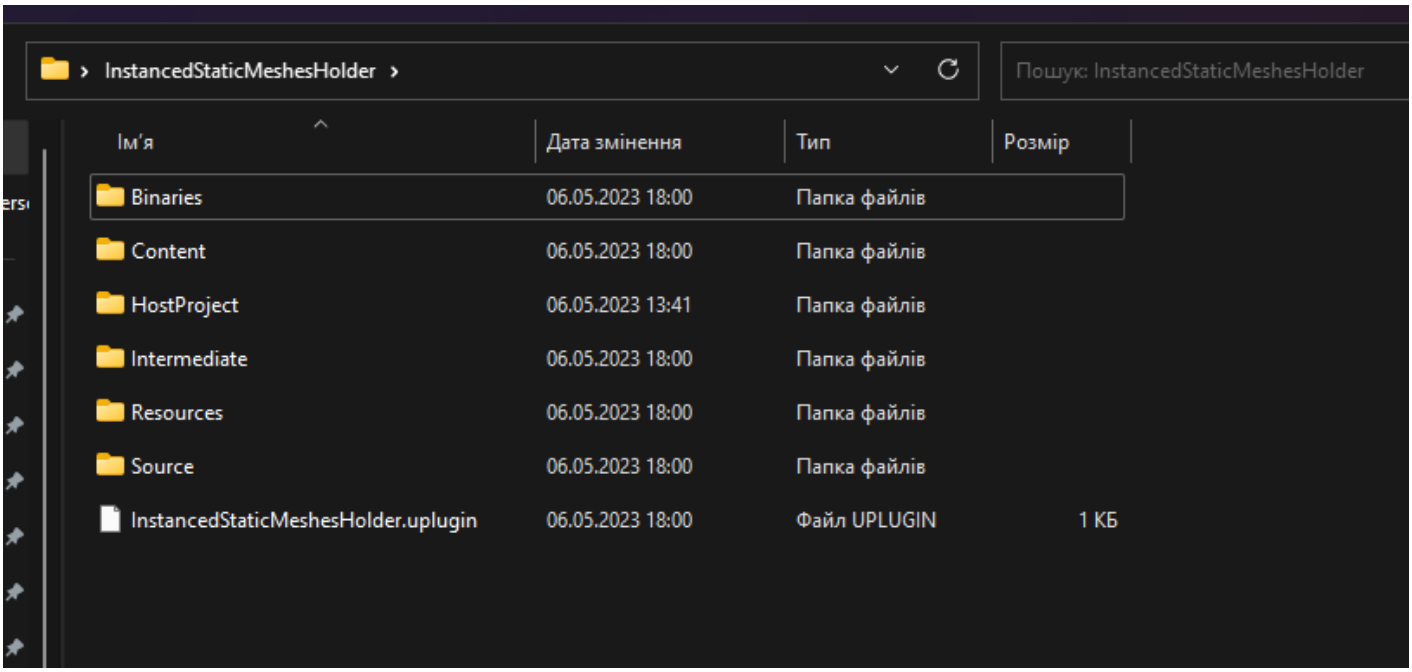


Рисунок 4.11 – Структура папки запакованого плагіну

4.2 Інтегрування плагіну проєкт

Для виконання цього етапу було створено новий проєкт. В папку з плагінами додано папку плагіну:

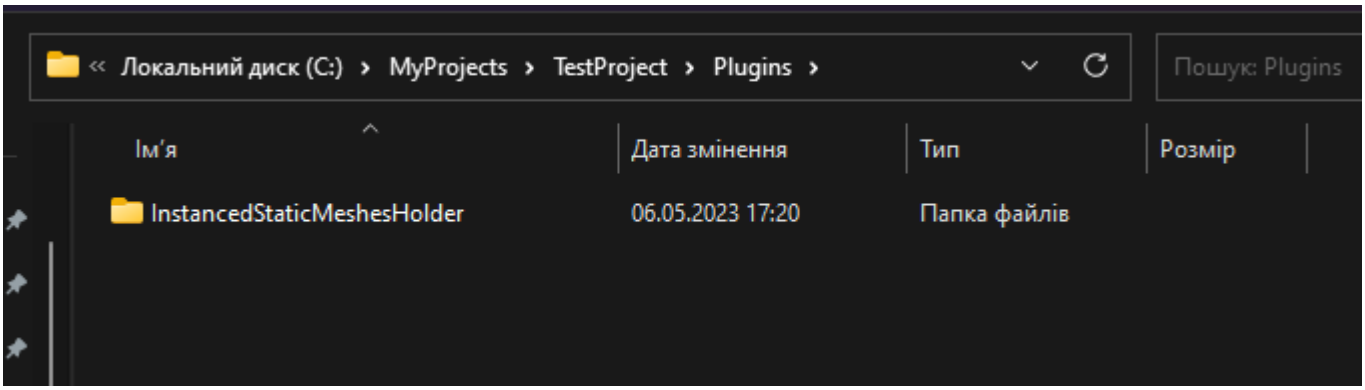


Рисунок 4.12 – Результат інтегрування плагіну в проєкт

Запускаємо проєкт, відкриваємо контент браузер та бачимо папку з плагіном. В середині є всі необхідні C++ та блюпрінт-класи (рис.4.13).

Створимо об’єкти, які будуть оптимізуватись. Вони обов’язково повинні бути нащадком класу InstancedMeshOptimizableObject (рис.4.14):

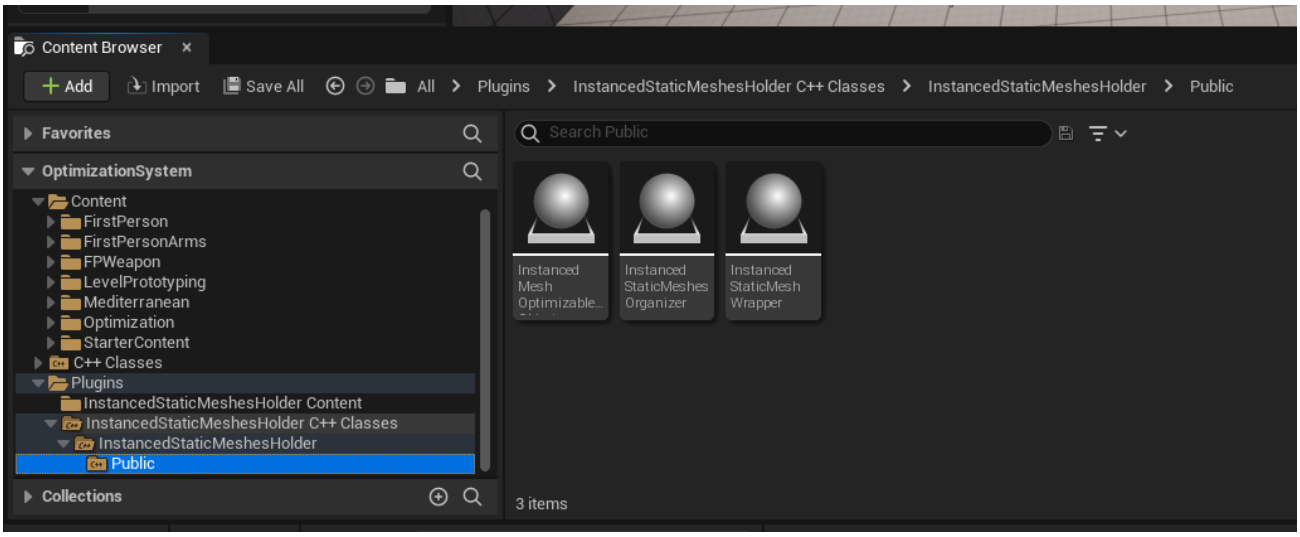


Рисунок 4.13 – Вміст плагіну

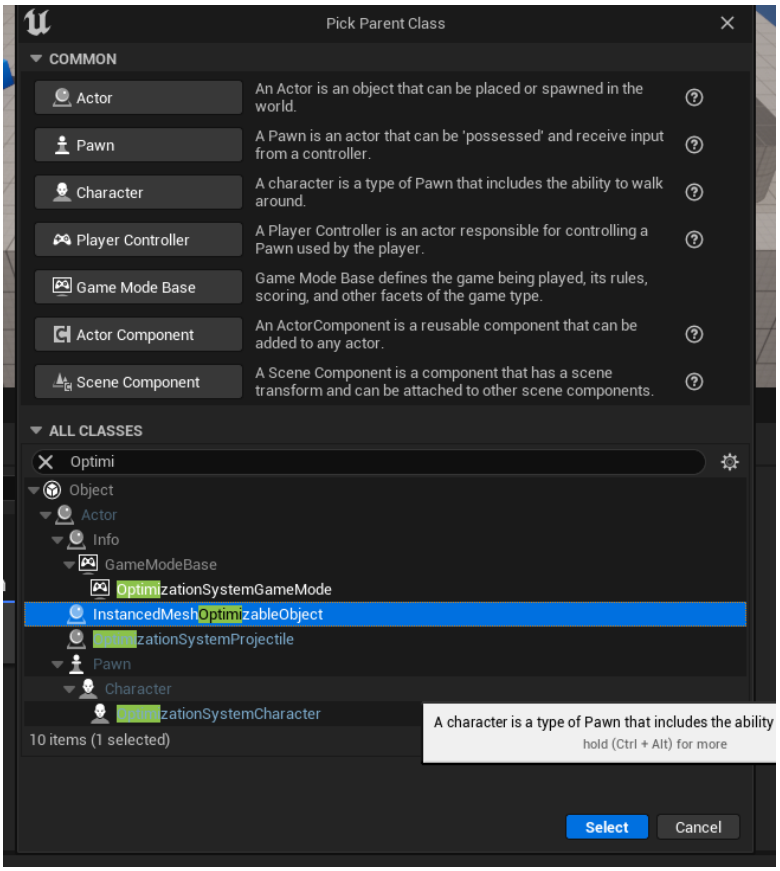


Рисунок 4.14 – Створення об’єкту для оптимізації

Задля тестування було використано безкоштовні моделі та матеріали з маркетплейсу. Цей крок зроблено щоб уникнути сторонніх помилок або додаткового навантаження на систему під час тестування.

Для тестування обрано першу модель - стілець та проведено необхідне налаштування моделі:

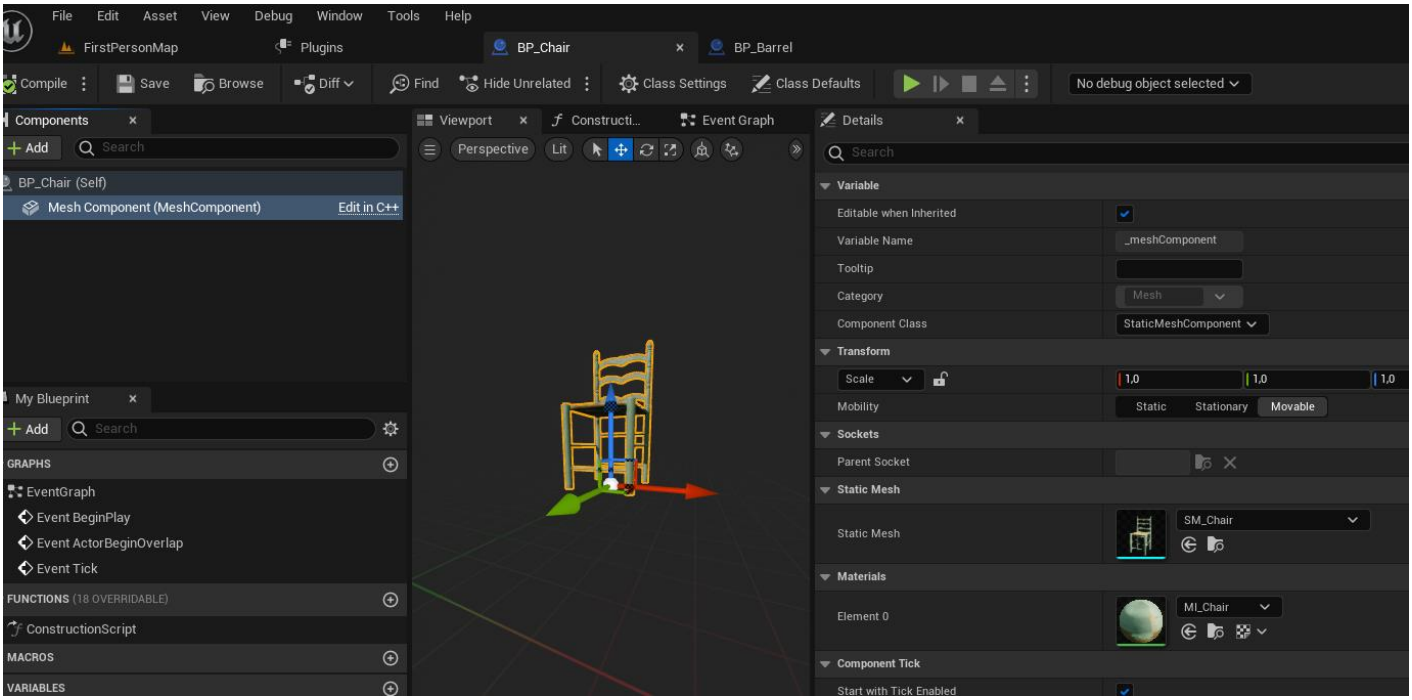


Рисунок 4.15 – Готовий об’єкт для оптимізації в Instanced Static Mesh

Далі для тестування плагіну створено бочку. Ми отримуємо ось такі актори, які можна розмістити на сцені в довільній послідовності (рис.4.16) Аналогічно було створено інші актори для тестування.

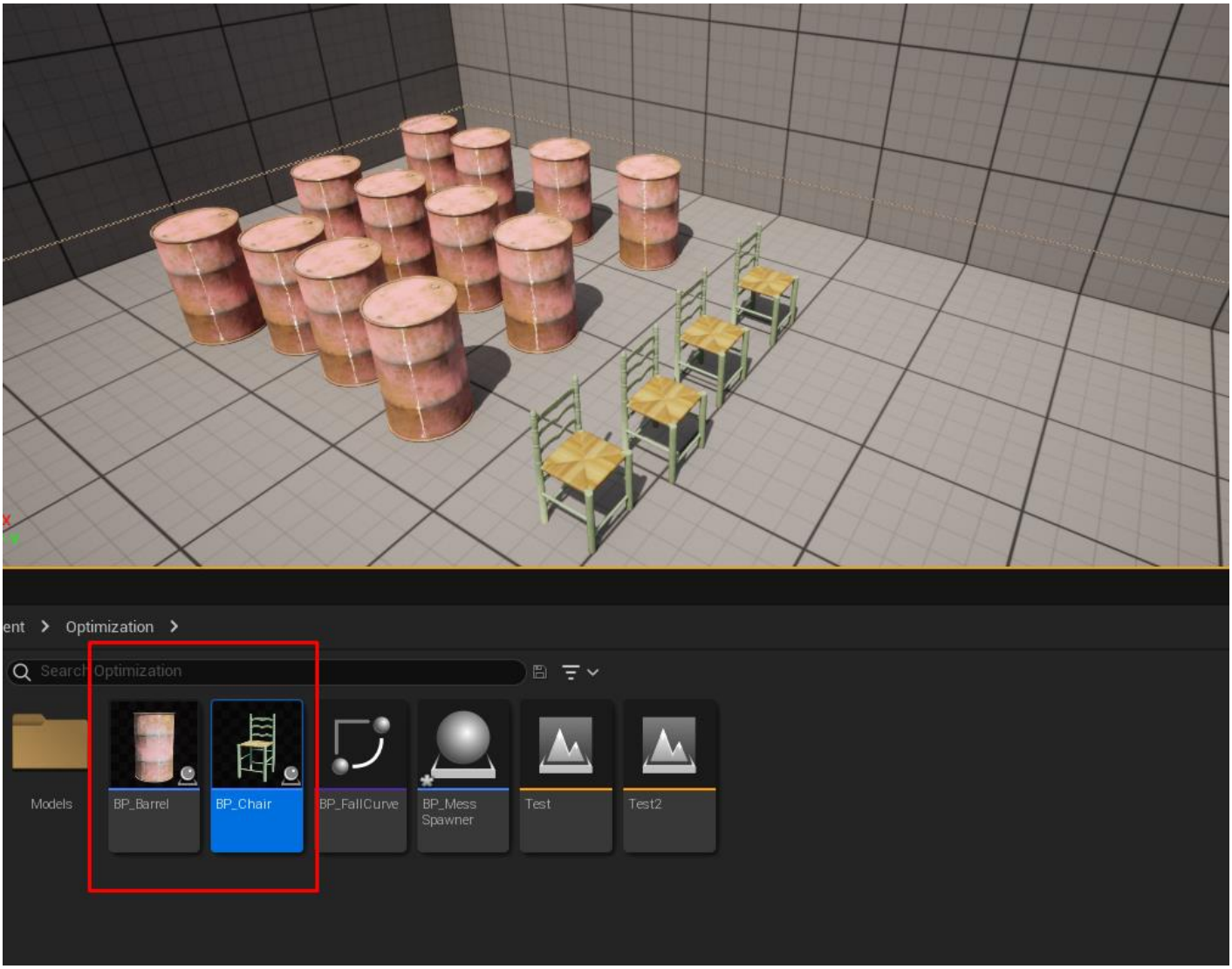


Рисунок 4.16 – Об’єкти для оптимізації

Наступним кроком треба перенести з контент браузеру на рівень BP_InstancedMeshesHolder, за допомогою чого в Unreal Engine 5 створюється екземпляр цього класу. Він може бути розташований в будь-якій локації на рівні, і це не буде заважати процесу оптимізації [19].

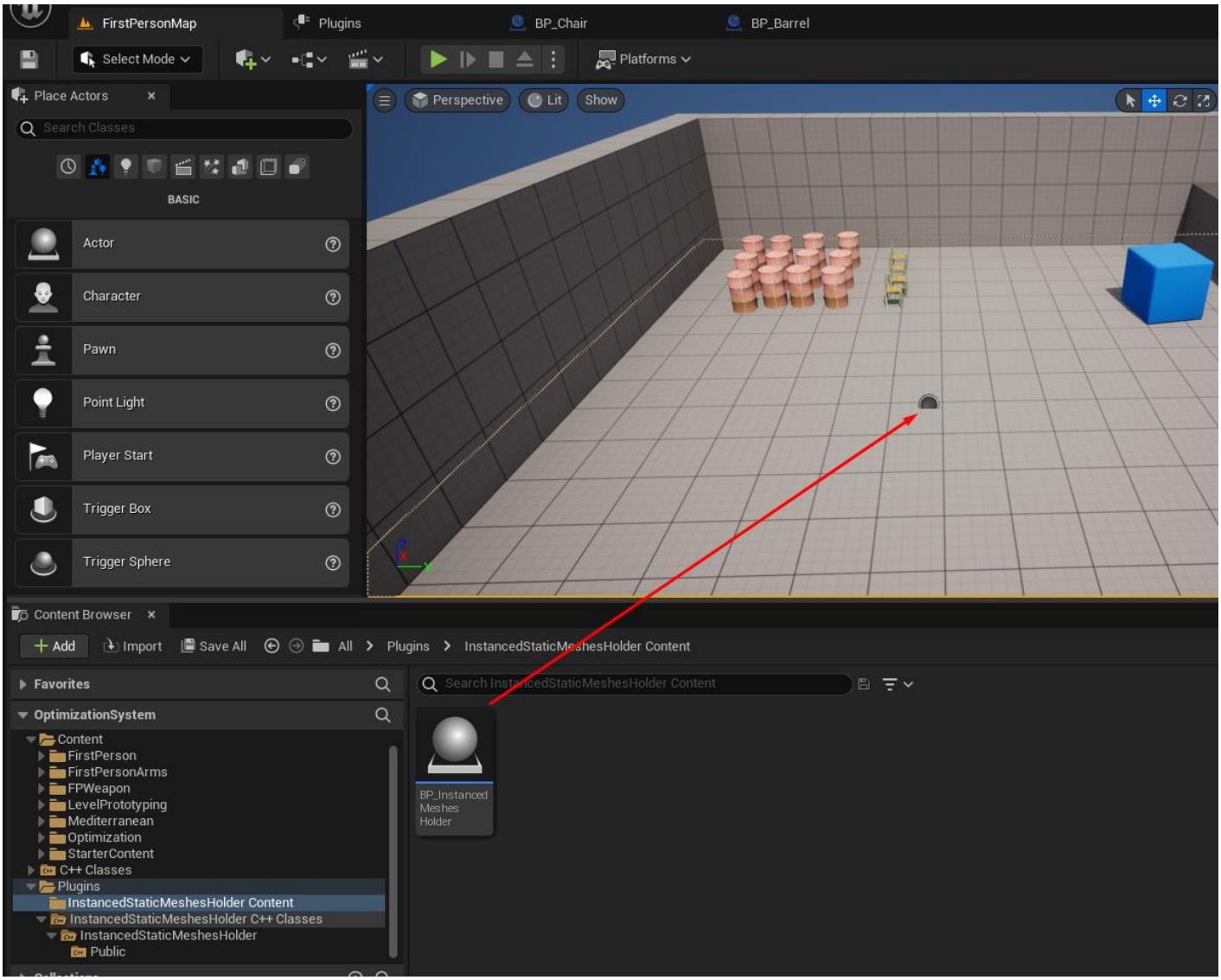


Рисунок 4.17 – Додавання тримача інстансів

Тепер потрібно встановити унікальний індекс для об’єктів які будуть оптимізуватись. Для бочки залишимо 0. Змінимо індекс для стільця:

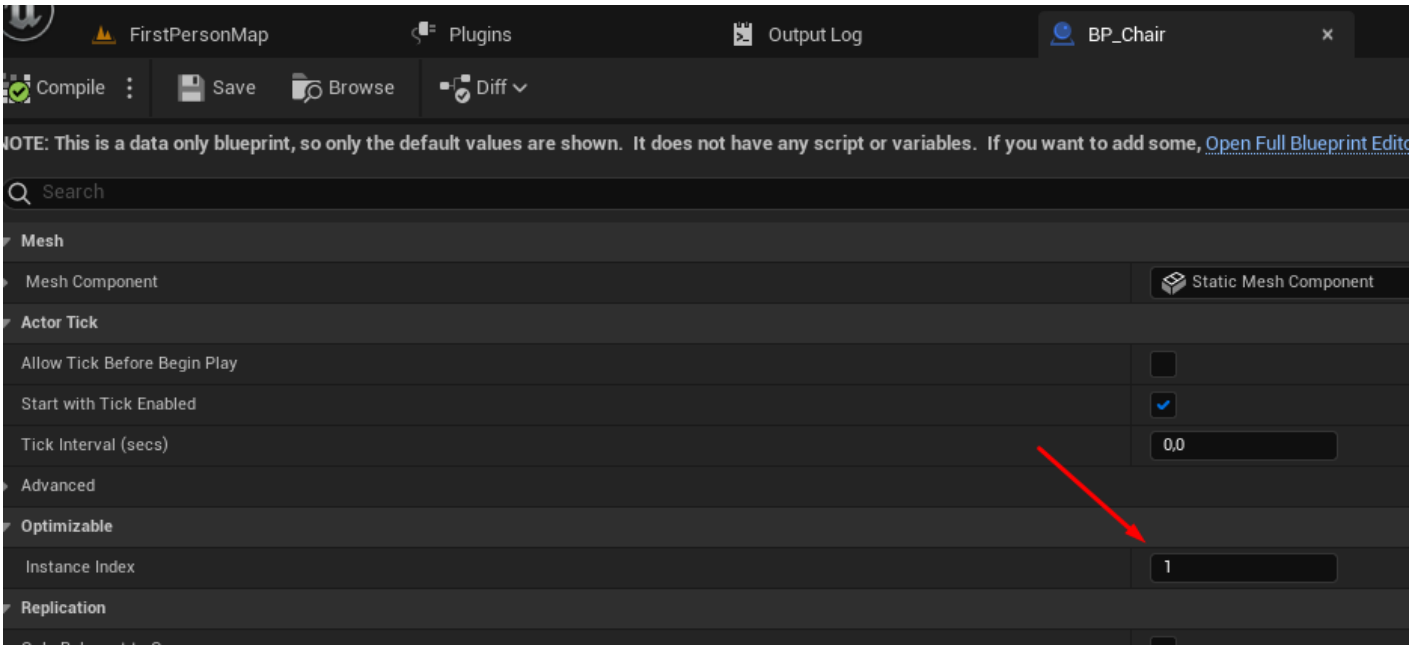


Рисунок 4.18 – Налаштування тримача інстансів

З цього моменту оптимізації працює і виконується при старті симуляції (після старту гри).

Як можна зрозуміти з блок-схеми – створення ISM та додавання до нього моделей відбувається під час гри, що робить покращений ISM набагато гнучкішим, ніж стандартний. Нам не потрібно передбачувати скільки різноманітних моделей буде додано на сцену – це все генерується в процесі гри.

З результатами оптимізації можна ознайомитись в наступному пункті.

4.3 Тестування

Характеристика комп’ютера, на якому тестувались методи оптимізації, наведена в таблиці 4.1:

Таблиця 4.1. Апаратні характеристики

CPU	Ryzen r7 5800x
GPU	GTX 1660 super
RAM	32 Gb DDR4 3600 MHz
SSD	MSI M390 – 1Tb
Monitor	Full HD (1920x1080)

Порівняння результатів проводилось за допомогою програми для моніторингу навантаження на ПК MSI Afterburner [20]. Вона дає змогу вивести статистику на будь-яке місце на екрані і дозволяє обрати, яку саме статистику відображати.

Тестування проводилось двічі: в першому випадку з використанням деталізованої моделі столу з лози (485 столів по 815 полігонів кожен) та 20 стільців, друге тестування – з використанням 2000 однакових куль (200 полігонів кожна). З результатами тестування можна ознайомитись в таблиці 4.2.

Таблиця 4.2. Результати тестування

	Навантаження на GPU	Навантаження на CPU	Використання ОЗП (mb)	Кількість кадрів в секунду
Тест 1	53%	4%	16286	21
Тест 1 + оптимізація	72%	4%	16315	31
Тест 2	32%	5%	18766	11
Тест 2 + оптимізація	97%	2%	18679	93

Далі наведено описання результатів тестування. Результати першого тестування представлені на рисунку 4.19 та 4.20. Приріст кількості кадрів в секунду було отримано, але бачимо, що навантаження на відеокарту також виросло. Це викликано тим, що з вимкненою оптимізацією, можливості ПК обмежуються пропускною здатністю оперативної пам’яті та процесора. Коли оптимізація вмикається – навантаження на оперативну пам’ять знижується і система може навантажити відеокарту більше і отримати вищу кількість кадрів. Взагалі за нормальних умов, відеокарта має бути навантажена на 100%, але такого результату в першому випробуванні досягти не вдалось через дуже чітку деталізацію моделі [21].

Також, за результатами випробування бачимо, що трішки змінилась тінь на об’єктах, але це викликано зміною часу на рівні і поворотом сонця.

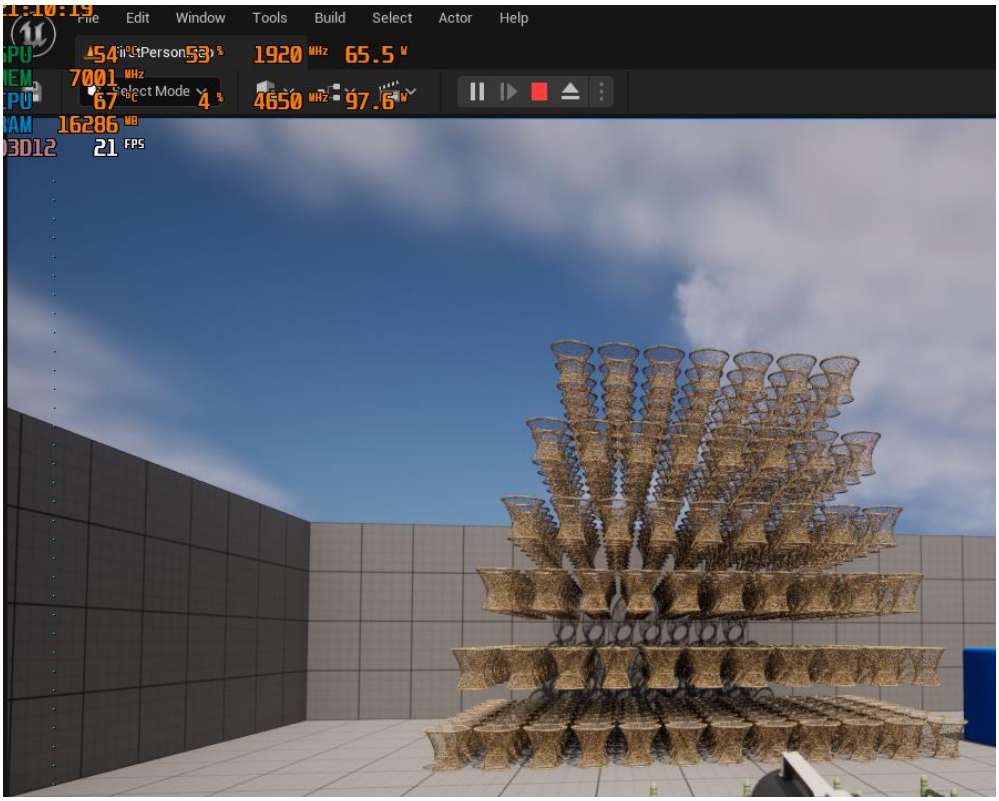


Рисунок 4.19 – Перший тест без оптимізації

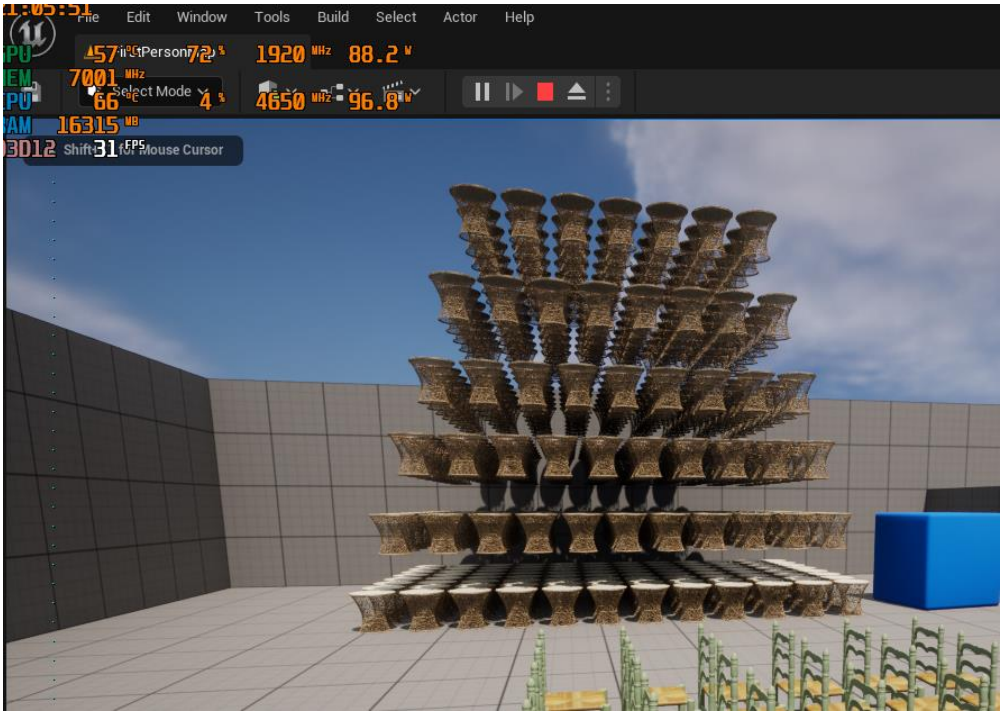


Рисунок 4.20 – Перший тест з оптимізацією

Друге тестування є менш реалістичним і така ситуація в іграх не виникає часто, але цілком може зустрітись в масштабних проєктах. Вона максимально розкриває потенція Instanced Static Mesh. Результати зображені на рисунках 4.21 та 4.22.

Як бачимо, за такої кількості об’єктів, використовувати Instanced Static Mesh критично важливо.

Рисунок 4.21 – Другий тест без оптимізації

Рисунок 4.22 – Другий тест з оптимізацією

ВИСНОВКИ

Оптимізація комп’ютерних ігор – дуже важливий етап в ході їх розробки, який необхідний для покращення показників завантаженості ПК та кількісті кадрів в секунду. В процесі виконання магістерської роботи було обрано один із найбільш проблемних методів оптимізації, який реалізовано в Unreal Engine 5 – Instanced Static Mesh. Було проведено його аналіз та висунуто гіпотези, які було підтверджено під час дослідження. Отримані результати дослідження свідчать про те, що використання технології Instanced Static Meshes може позитивно вплинути на продуктивність комп'ютерних ігор, які містять значну кількість однакових об'єктів. Однак, для досягнення даного ефекту необхідно використовувати покращений метод оптимізації, який базується на виконанні пакування наявних інстансів в контейнер. Для реалізації даного методу було створено спеціальний плагін, який дозволяє автоматизувати зазначений процес. Цей підхід дозволяє скоротити час, необхідний для завантаження об'єктів у гру, що збільшує продуктивність гри та покращує досвід користувача.

Отже, в результаті виконання даної роботи було виконано всі поставлені задачі, тобто:

- проведено аналіз технології Instanced Static Meshes та її впливу на продуктивність комп'ютерних ігор;
- покращено метод оптимізації на основі Instanced Static Mesh та описано принцип його принцип дії;
- реалізовано розроблений метод в грі та проведено порівняльний аналіз продуктивності гри до та після впровадження методу;

- зроблено висновки про ефективність використання технології Instanced Static Meshes у процесі оптимізації комп'ютерних ігор.

Наукова новизна дослідження полягає в покращенні технології Instanced Static Meshes до рівня автоматичної оптимізації за рахунок використання розробленого плагіну, який дозволяє об'єднання моделі декількох об'єктів на рівні в один Instanced Static Mesh.

Основні теоретичні, методологічні та практичні результати проведеного дослідження, були подані у вигляді доповіді на конференції «IMA-2023», головні результати котрої опубліковано у збірці матеріалів конференції та подані до друку у вигляді статті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Instanced Static Mesh [Електронний ресурс] – режим доступу: https://docs.unrealengine.com/4.27/en-US/BlueprintAPI/Components/InstancedStaticMesh/

2. Analyzing Performance Issues of Virtual Reality Applications / Jason Hogan, Aaron Salo, Dhia Elhaq Rzig, Foyzul Hassan, Bruce Maxim – University of Michigan – Dearborn, 2017y. – 11p.

3. Behind the scenes of The Cavern UE5 Cinematic Visual Tech Test / Colin Penty – Association for Computing Machinery, 2022y. – 13p.

4. Оптимізація ігор на Unity 5 / Кріс Дікинсон . – ДМК-пресс, 2016р. – 306 с.

5. Unreal Engine 4 Optimization Tutorial by Intel [Електронний ресурс] – режим доступу: https://www.intel.com/content/www/us/en/developer/articles/training/unreal-engine-4-optimization-tutorial-part-1.html

6. Exploring Game Design Through Human-AI Collaboration / Alberto Alvarez, 2022 – Malmo University, 2022y. – 381p.

7. Efficient visualization of 3D city scenes by integrating GIS and Unreal Engine / Hai Xu, Biao He, Ze Yu Li, HaoJia Lin, AoWei Tang – Socity of Photo-Optical Instrumentation Engineers, 2022y. – 15p.

8. Unreal Engine 5 [Електронний ресурс] – режим доступу: https://www.unrealengine.com/en-US

9. Visual Studio [Електронний ресурс] – режим доступу: https://visualstudio.microsoft.com/

10. Rider for Unreal Engine [Електронний ресурс] – режим доступу: https://www.jetbrains.com/lp/rider-unreal/

11. Creating a new project [Електронний ресурс] – режим доступу: https://docs.unrealengine.com/5.0/en-US/creating-a-new-project-in-unreal-engine/

12. Building plugins [Електронний ресурс] – режим доступу: https://dev.epicgames.com/community/learning/tutorials/qz93/unreal-engine-building-plugins

13. Properties [Електронний ресурс] – режим доступу: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/GameplayArchitecture/Properties/

14. Properties [Електронний ресурс] – режим доступу: Unreal Engine C++ Complete Guide https://www.tomlooman.com/unreal-engine-cpp-guide/

15. Introduction to blueprints [Електронний ресурс] – режим доступу: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/

16. Plugins [Електронний ресурс] – режим доступу: https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/Plugins/ Plugins, content cooking

17. Content cooking [Електронний ресурс] – режим доступу: https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/Deployment/Cooking/

18. Cooking game with custom editor plugins [Електронний ресурс] – режим доступу: https://michaeljcole.github.io/wiki.unrealengine.com/Plugins__Cooking_Game_with_Custom_Editor_plugins/

19. UObject instance cration [Електронний ресурс] – режим доступу: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Objects/Creation/]

20. MSI Afterburner [Електронний ресурс] – режим доступу: https://ua.msi.com/Landing/afterburner/graphics-cards

21. What is memory Bandwidth [Електронний ресурс] – режим доступу: https://www.easytechjunkie.com/what-is-memory-bandwidth.htm

Додаток А
Планування робіт

Ідентифікація мети. Результатом виконання дипломної роботи є система на основі Instanced Static Mesh, яка зможе автоматично оптимізувати об’єкти заданих класів.

Далі наведена таблиця деталізації мети методом SMART:

Таблиця А.1 − Деталізація мети методом SMART

Specific (конкретна)	Створена система – це надбудова над Instanced Static Mesh, яка автоматично створює інстанси для моделей та додає об’єкти з однаковими моделями до конкретного інстансу, створюючи при цьому по одному Instanced Static Mesh для кожного типу моделі.
Measurable (вимірювана)	Середній результати в рості fps після включення Instanced Static Mesh Holder в проєкті має бути позитивним та реалістичним (приблизно на рівні Instanced Static Mesh, але зі значно ширшою сферою застосування).
Achievable (досяжна)	Мета роботи розбита на чіткі задачі та має план реалізації. Для досягнення мети буде використовуватись Rider for Unreal Engine, Visual Studio, C++ та Unreal Engine 5.0.2.
Relevant (реалістична)	Позитивний вплив на fps та навантаження на систему має бути позитивним та схожим зі застосуванням класичного Instanced Static Mesh (бо алгоритм не змінюється, покращується лише процес використання та простота реалізації).
Time-framed (обмежена у часі)	Ціль має бути досягнута в термін виконання дипломної роботи. Проєкт має виконуватись в терміни, визначені з календарним графіком.

Планування змісту структури робіт. Для планування змісту структури роботи було обрано представлення WBS – це візуальне розбиття всієї роботи на менші та більш зрозумілі для виконавця завдання (така собі декомпозиція мети роботи).

В нашому випадку, кожен з етапів вибудовано в порядку їх виконання від самого початку зародження ідеї проєкту. Вийшло 5 етапів:

1. Ознайомлення з предметної областю. Тестування найпопулярніших методів оптимізації і визначення проблемних з подальшим визначенням проблемного методу з перспективою покращення.
2. Аналіз технології Instanced Static Meshes та її впливу на продуктивність комп'ютерних ігор
3. Покращити методу оптимізації на основі Instanced Static Mesh та описання його принцип дії
4. Реалізація розробленого методу в грі та проведення порівняльного аналізу продуктивності гри до та після впровадження методу
5. Написання необхідної документації та формування звітів

Кожен з цих етапів був розбитий підзадачі. З повною схемою можна ознайомитись на рисунку А.1:

На рисунку А.1 приведена WBS-структура даного проекту:

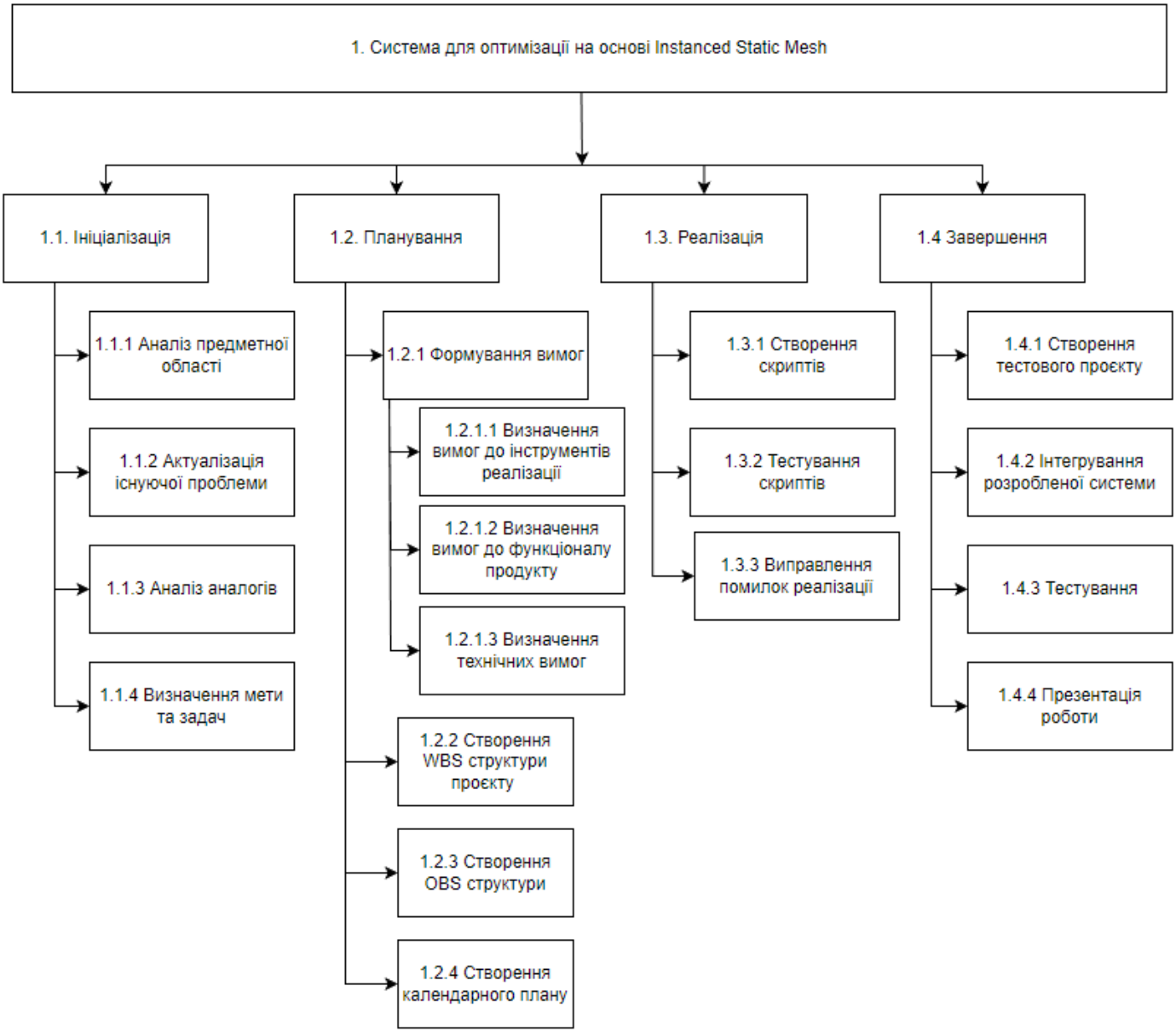


Рисунок А.1 – WBS-структура проекту

Планування структури організації, для впровадження готового проекту (OBS). Для планування структури організації чудово підходить OBS. Він має таку ж структуру як і OBS, але замість задач відображає особу, яка виконує цю задачу.

Порівнюючи цю схему з WBS можна чітко зрозуміти які задачі існують в проєкті і хто їх виконує. Організаційна структура зображена на рисунку А.2. Таблиця з ролями проєкту має такий вигляд:

Таблиця А.2 – Виконавці проекту

Роль	Ім’я	Проектна роль
Розробник	Проценко М.О.	Виконує дослідження та покращення методу оптимізації Instanced Static Mesh.
Менеджер проекту	Федотова Н.А.	Допомагає з питаннями аналізу та реалізації. Відповідає за дотримання термінів та вимог виконання роботи
Тестувальник	Проценко М.О.	Тестує систему на продуктивність та відсутність критичних помилок

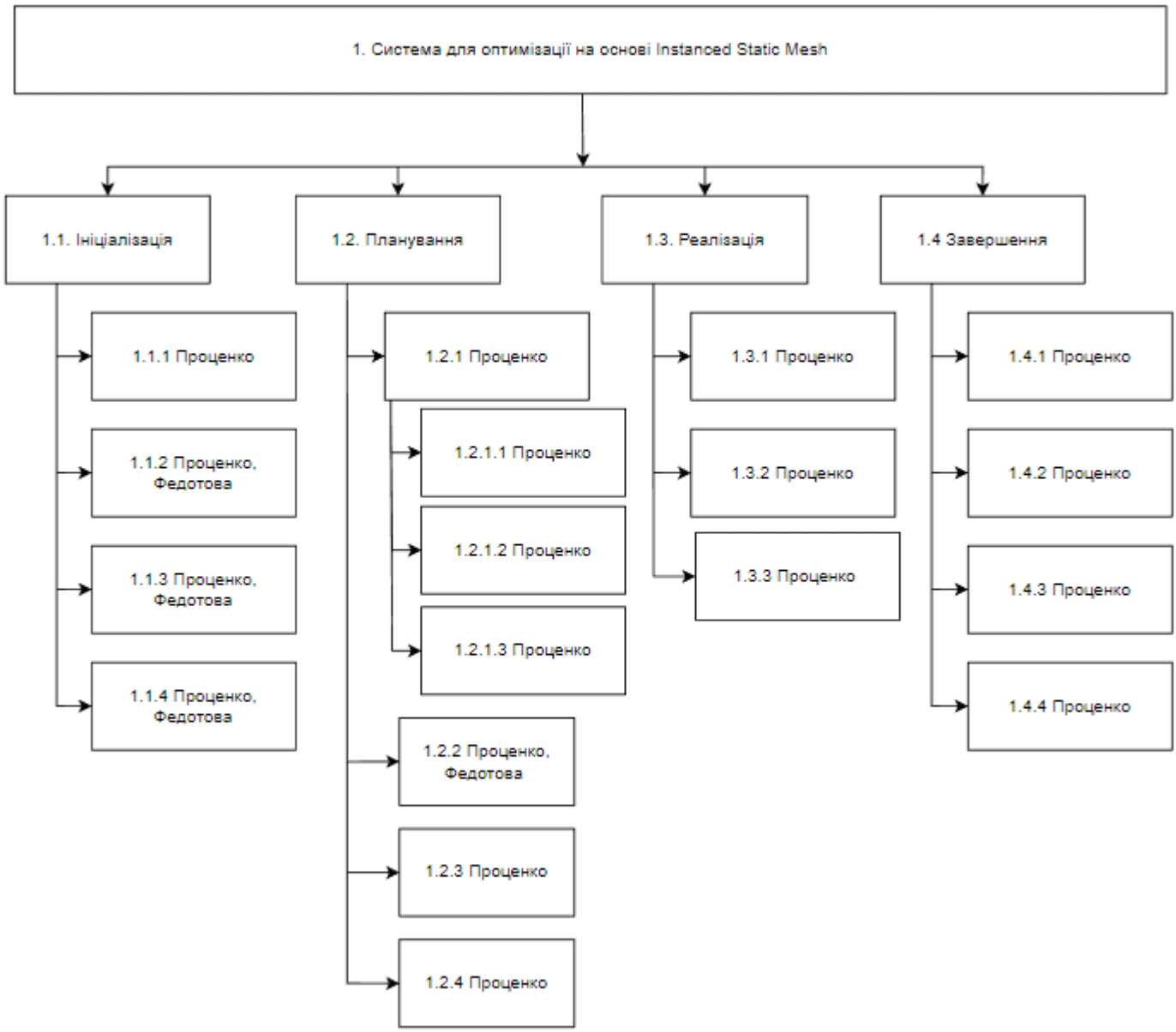


Рисунок А.2 – OBS-структура проекту

Побудова календарного графіку виконання IT - проекту. Для створення календарного графіку проєкту частіше за все використовується діаграма Ганта. Ця діаграма побудована на основі гістограм і будується на основі списку етапів виконання

проєкту з назвою етапу, його тривалість, початковою кінцевою датою. Для великих проєктів діаграма Ганта критично важлива, так як через величезну кількість задач дуже складно слідкувати за їх статусом і порядком.

Діаграма Ганта для системи для оптимізації на основі Instanced Static Meshes наведено на рисунку А.3:

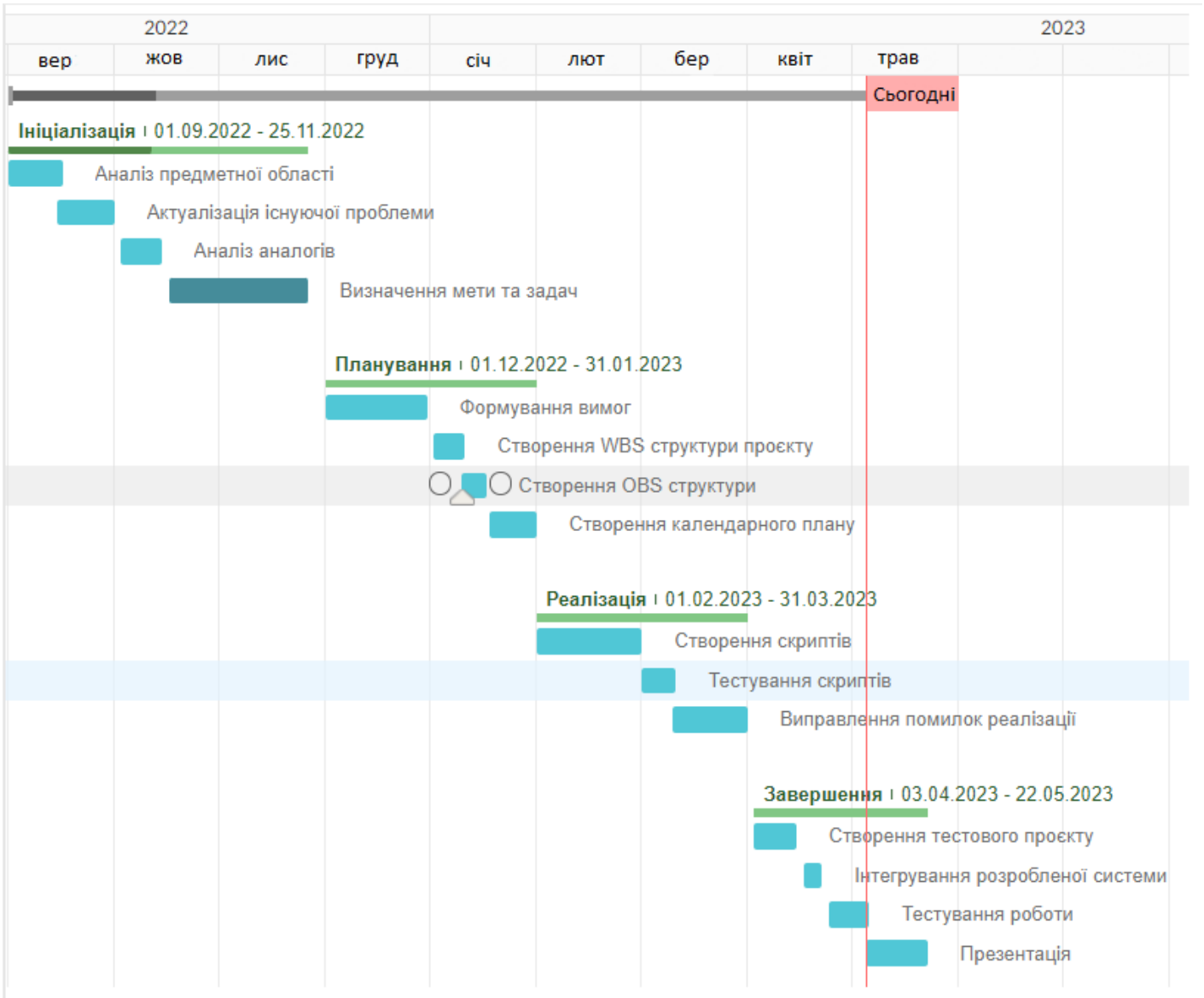


Рисунок А.3 – Діаграма Ганта проекту

	Задача	Початок	Завершення	+
		01.09.2022	22.05.2023	···
1	**Ініціалізація**	**01.09.2022**	25.11.2022	···
1.1	Аналіз предметної області	01.09.2022	15.09.2022	···
1.2	Актуалізація існуючої проблеми	15.09.2022	30.09.2022	···
1.3	Аналіз аналогів	03.10.2022	14.10.2022	···
1.4	Визначення мети та задач	17.10.2022	25.11.2022	···
2	**Планування**	**01.12.2022**	31.01.2023	···
2.1	Формування вимог	01.12.2022	30.12.2022	···
2.2	Створення WBS структури проєкту	02.01.2023	10.01.2023	···
2.3	Створення OBS структури	10.01.2023	17.01.2023	···
2.4	Створення календарного плану	18.01.2023	31.01.2023	···
3	**Реалізація**	**01.02.2023**	31.03.2023	···
3.1	Створення скриптів	01.02.2023	28.02.2023	···
3.2	Тестування скриптів	01.03.2023	10.03.2023	···
3.3	Виправлення помилок реалізації	10.03.2023	31.03.2023	···
4	**Завершення**	**03.04.2023**	22.05.2023	···
4.1	Створення тестового проєкту	03.04.2023	14.04.2023	···
4.2	Інтегрування розробленої системи	17.04.2023	21.04.2023	···
4.3	Тестування роботи	24.04.2023	05.05.2023	···
4.4	Презентація	05.05.2023	22.05.2023	···

Рисунок А.4 – Список робіт по проєкту

Аналіз ризиків. Аналіз ризиків – це один із найважливіших етапів, який має виконуватись ще до старту проєкту. Чим більше ризиків і чим більше їх можливий

негативний вплив на проєкт – тим більше шанс того, що проєкт провалиться на якомусь етапі роботи або не буде фінансово успішним.

Звичайно, передбачити всі ризики проєкту неможливо, але сам процес аналізу ризиків базується на персональному досвіді персони, яка проводить аналізи. Якщо ця людина достатньо компетентна – то майже завжди більшість ризиків вдається передбачити і бути готовими до них.

На основі проведеного аналізу було сформовано матрицю Risk Breakdown Structure:

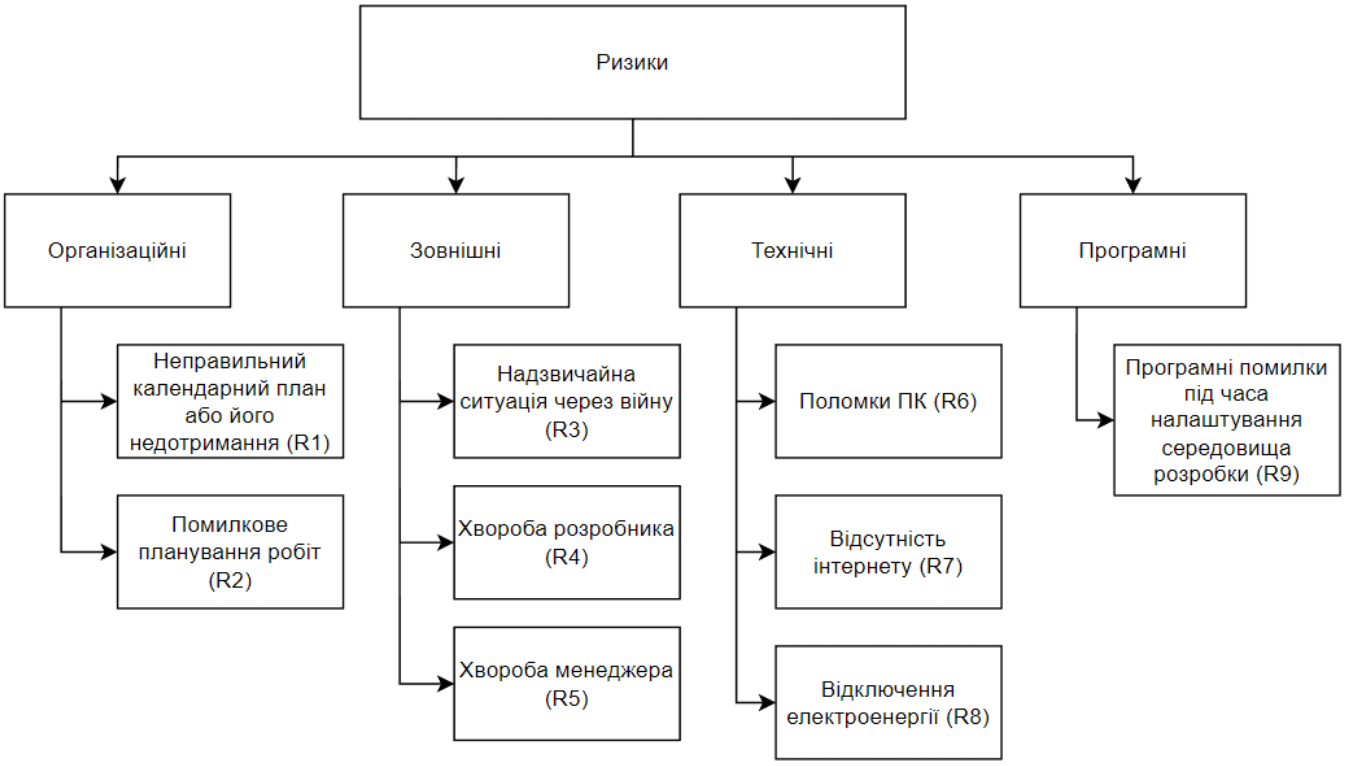


Рисунок А.5 – RBS-матриця

Далі було створено таблиці ймовірності виникнення (А3), відповідальності рівня ризику (А.4), відповідальності ступеня впливу (А.5), відповідальності ступеня

витрат (А.6):

Таблиця А.3 – Матриця відповідальності ймовірності виникнення

Ймовірніть виникнення		R1	R2	R3	R4	R5	R6	R7	R8	R9
1	Мінімальна									
2	Низька									
3	Середня									
4	Висока									
5	Максимальна									

Таблиця А.4 – Матриця відповідальності рівня ризику

Рівень ризику		R1	R2	R3	R4	R5	R6	R7	R8	R9
1	Прийнятний									
2	Виправданий									
3	Недопустимий									

Таблиця А.5 – Матриця відповідальності ступеня впливу

Ступінь впливу		R1	R2	R3	R4	R5	R6	R7	R8	R9
1	Мінімальний									
2	Незначний									
3	Допустимий									
4	Значний									
5	Максимальний									

Таблиця А.6 – Матриця відповідальності ступеня втрат

Ступінь втрат		R1	R2	R3	R4	R5	R6	R7	R8	R9
1	Мінімальні									
2	Незначні									
3	Допустимі									
4	Значні									
5	Максимальні									

Додаток Б.

Програмний код

InstancedMeshOptimizableObject.h:

```
#pragma once
#include "GameFramework/Actor.h"
#include "InstancedMeshOptimizableObject.generated.h"

class AInstancedStaticMeshesOrganizer;

UCLASS(Blueprintable)
class AInstancedMeshOptimizableObject : public AActor
{
   GENERATED_BODY()
public:
   AInstancedMeshOptimizableObject();

   UStaticMeshComponent* getMeshComponent() const;

   virtual void BeginPlay() override;

   UPROPERTY(EditAnywhere, Category="Mesh")
   UStaticMeshComponent* _meshComponent;

   void setInstanceIndex(int index);
   int getOptimizationIndex() const;

private:
   bool _optimizationEnabled;

   int _instanceIndex;
};
```

InstancedMeshOptimizableObject.cpp:

```
#include "InstancedMeshOptimizableObject.h"

#include "InstancedStaticMeshesOrganizer.h"
#include "Components/StaticMeshComponent.h"
#include "Kismet/GameplayStatics.h"
#include "Engine/World.h"

AInstancedMeshOptimizableObject::AInstancedMeshOptimizableObject() :
_optimizationEnabled(true)
{
   _meshComponent =
CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MeshComponent"));
   _meshComponent->SetupAttachment(RootComponent);
}

UStaticMeshComponent* AInstancedMeshOptimizableObject::getMeshComponent()
const
{
   return _meshComponent;
}

void AInstancedMeshOptimizableObject::BeginPlay()
{
   Super::BeginPlay();

   if (_optimizationEnabled)
   {
      TArray<AActor*> holders;
      UGameplayStatics::GetAllActorsOfClass(GetWorld(),
AInstancedStaticMeshesOrganizer::StaticClass(),
         holders);
       Cast<AInstancedStaticMeshesOrganizer>(holders[0])-
>enableOptimizationForObject(this);
   }
}

void AInstancedMeshOptimizableObject::setInstanceIndex(int index)
{
   _instanceIndex = index;
}

int AInstancedMeshOptimizableObject::getOptimizationIndex() const
{
   return _instanceIndex;
}
```

InstancedStaticMeshesOrganizer.h:

```
#pragma once

#include "GameFramework/Actor.h"
#include "InstancedStaticMeshesOrganizer.generated.h"

class AInstancedMeshOptimizableObject;
class AInstancedStaticMeshWrapper;

UCLASS()
class AInstancedStaticMeshesOrganizer : public AActor
{
   GENERATED_BODY()

public:
   AInstancedStaticMeshesOrganizer();

   virtual void BeginDestroy() override;
   AInstancedStaticMeshWrapper* findStaticMeshesSpawnerByObjectType(int
objectType) const;

   void disableOptimizationForAllInstances() const;
   void enableOptimizationForAllInstances() const;

   void enableOptimizationForObject(AInstancedMeshOptimizableObject*
addedObject);
   void disableOptimizationForObject(AInstancedMeshOptimizableObject*
removedObject) const;

protected:
   UPROPERTY(EditAnywhere, BlueprintReadWrite,
Category="OptimizableObjects")
   TArray<TSubclassOf<AInstancedMeshOptimizableObject>>
_optimizableObjects;

   UPROPERTY(EditAnywhere, BlueprintReadWrite,
Category="OptimizableObjects")
   bool _isOptimizationEnabled;

private:
   UPROPERTY()
   TMap<int, AInstancedStaticMeshWrapper*> _instancedStaticMeshesInfo;
};
```

InstancedStaticMeshesOrganizer.cpp:

```
#include "InstancedStaticMeshesOrganizer.h"

#include "InstancedMeshOptimizableObject.h"
#include "InstancedStaticMeshWrapper.h"
#include "Kismet/GameplayStatics.h"

AInstancedStaticMeshesOrganizer::AInstancedStaticMeshesOrganizer()
{
}

void
AInstancedStaticMeshesOrganizer::enableOptimizationForObject(AInstancedMe
shOptimizableObject* addedObject)
{
    if (!_isOptimizationEnabled){
        return;
    }

    AInstancedStaticMeshWrapper* wrapperForIndex =
findStaticMeshesSpawnerByObjectType(addedObject->getOptimizationIndex());
    wrapperForIndex->addConvertableObject(addedObject);

    if (wrapperForIndex == nullptr)
    {
        wrapperForIndex = GetWorld()-
>SpawnActorDeferred<AInstancedStaticMeshWrapper>(AInstancedStaticMeshWrap
per::StaticClass(), FTransform());
        wrapperForIndex->init(addedObject->getMeshComponent()-
>GetStaticMesh());
        UGameplayStatics::FinishSpawningActor(wrapperForIndex,
FTransform());
        _instancedStaticMeshesInfo.Add(addedObject-
>getOptimizationIndex(), wrapperForIndex);
    }
}

void
AInstancedStaticMeshesOrganizer::disableOptimizationForObject(AInstancedM
eshOptimizableObject* removedObject) const
{
    AInstancedStaticMeshWrapper* wrapperForIndex =
findStaticMeshesSpawnerByObjectType(removedObject-
```

```
>getOptimizationIndex());

    if (wrapperForIndex != nullptr){
        wrapperForIndex->removeInstanceMeshInfoByObject(removedObject);
    }
}

void AInstancedStaticMeshesOrganizer::BeginDestroy()
{
    Super::BeginDestroy();

    TArray<UActorComponent*> holdingComponents = GetInstanceComponents();
    for (UActorComponent* component : holdingComponents)
    {
        component->DestroyComponent();
    }
}

AInstancedStaticMeshWrapper*
AInstancedStaticMeshesOrganizer::findStaticMeshesSpawnerByObjectType(int
objectType) const
{
    if (_instancedStaticMeshesInfo.Num() > 0)
    {
        return _instancedStaticMeshesInfo.FindRef(objectType);
    }

    return nullptr;
}

void
AInstancedStaticMeshesOrganizer::disableOptimizationForAllInstances()
const
{
    TArray<AInstancedStaticMeshWrapper*> wrappers;
    _instancedStaticMeshesInfo.GenerateValueArray(wrappers);
    for (AInstancedStaticMeshWrapper* wrapper : wrappers)
    {
        wrapper->disableOptimization();
    }
}

void AInstancedStaticMeshesOrganizer::enableOptimizationForAllInstances()
const
{
```

```
    TArray<AInstancedStaticMeshWrapper*> wrappers;
    _instancedStaticMeshesInfo.GenerateValueArray(wrappers);
    for (AInstancedStaticMeshWrapper* wrapper : wrappers)
    {
        wrapper->enableOptimization();
    }
}
```

InstancedStaticMeshWrapper.h:

```
#pragma once

#include "CoreMinimal.h"
#include "InstancedStaticMeshWrapper.generated.h"

class AInstancedMeshOptimizableObject;
class UHierarchicalInstancedStaticMeshComponent;
class AInstancedMeshOptimizableObject;
struct FBodyInstance;
struct FTimerHandle;

UCLASS()
class AInstancedStaticMeshWrapper : public AActor
{
   GENERATED_BODY()

public:
   AInstancedStaticMeshWrapper();
   void createMeshInstanceFromObject(AInstancedMeshOptimizableObject*
copiedObject);
   AInstancedMeshOptimizableObject* createObjectFromMeshInstance(int
meshInstanceIndex);

   void init(UStaticMesh* exampleMesh);

   void addConvertableObject(AInstancedMeshOptimizableObject*
addedObject);
   void removeInstanceMeshInfoByIndex(int meshInstanceIndex) const;
   void removeInstanceMeshInfoByObject(AInstancedMeshOptimizableObject*
removedObject);

   void disableOptimization();
   void enableOptimization();

   UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="Settings")
```

```
   UHierarchicalInstancedStaticMeshComponent*
InstancedStaticMeshComponent;

protected:
   virtual void BeginPlay() override;
   virtual void BeginDestroy() override;
   virtual void Tick(float DeltaSeconds) override;

private:
   void activateObject(AInstancedMeshOptimizableObject* objectToEnable);
   void deactivateObject(AInstancedMeshOptimizableObject*
objectToDisable);

   void enableMeshCreatingTimer();
   void disableMeshCreatingTimer();

   TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>*
getHeldMeshDataByIndex(int meshInstanceIndex) const;
   TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>*
getHeldMeshDataByObject(AInstancedMeshOptimizableObject* heldObject)
const;

   bool
isActorPreparedForChangeToStaticMesh(AInstancedMeshOptimizableObject*
actor);

   UFUNCTION()
   void OnStaticMeshHit(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp,
      FVector NormalImpulse, const FHitResult& Hit);

   UFUNCTION()
   void OnStaticMeshBeginOverlap(UPrimitiveComponent*
OverlappedComponent, AActor* OtherActor,
      UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool
bFromSweep, const FHitResult & SweepResult);

   UFUNCTION()
   void OnHeldObjectDestroyed(AActor* DestroyedActor);

   FTimerHandle MeshCreatingTimer;
   FTimerHandle RefreshingTimer;

   TArray<TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>>*
_instancedStaticMeshesActors;
```

```
   UStaticMesh* _exampleMesh;
};
```

InstancedStaticMeshWrapper.cpp:

```
#include "InstancedStaticMeshWrapper.h"

#include "InstancedMeshOptimizableObject.h"
#include "Components/SphereComponent.h"
#include "Kismet/GameplayStatics.h"
#include "Kismet/KismetSystemLibrary.h"
#include "Components/HierarchicalInstancedStaticMeshComponent.h"
#include "Engine/StaticMesh.h"
#include "TimerManager.h"
#include "PhysicsEngine/BodyInstance.h"
#include "GameFramework/Actor.h"
#include "CoreTypes.h"

AInstancedStaticMeshWrapper::AInstancedStaticMeshWrapper()
{
    InstancedStaticMeshComponent =
CreateDefaultSubobject<UHierarchicalInstancedStaticMeshComponent>(
        FName("HierarchicalInstancedStaticMeshComponent"));
    InstancedStaticMeshComponent->bMultiBodyOverlap = true;
    InstancedStaticMeshComponent->bHasPerInstanceHitProxies = true;
    InstancedStaticMeshComponent->NumCustomDataFloats = 3;
    InstancedStaticMeshComponent->SetNotifyRigidBodyCollision(true);
    InstancedStaticMeshComponent->SetSimulatePhysics(true);

    SetRootComponent(InstancedStaticMeshComponent);
    _instancedStaticMeshesActors = new
TArray<TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>>();
}

void
AInstancedStaticMeshWrapper::createMeshInstanceFromObject(AInstancedMeshO
ptimizableObject* copiedObject)
{

deactivateObject(Cast<AInstancedMeshOptimizableObject>(copiedObject));

    const FTransform& objectTransform = copiedObject-
>GetActorTransform();
    const int32 spawnedInstanceIndex = InstancedStaticMeshComponent-
```

```
>AddInstanceWorldSpace(objectTransform);
    InstancedStaticMeshComponent->SetMaterial(spawnedInstanceIndex,
copiedObject->getMeshComponent()->GetMaterial(0));

    Cast<AInstancedMeshOptimizableObject>(copiedObject)-
>setInstanceIndex(spawnedInstanceIndex);

    TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>* foundData =
        _instancedStaticMeshesActors-
>FindByPredicate([copiedObject](TPair<AActor*, FBodyInstance*> data)
    {
        return copiedObject == data.Key;
    });

    if (foundData != nullptr)
    {
        foundData->Value = InstancedStaticMeshComponent-
>InstanceBodies[spawnedInstanceIndex];
    }
    else
    {
        const TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>
newPair(copiedObject,
            InstancedStaticMeshComponent-
>InstanceBodies[spawnedInstanceIndex]);
        _instancedStaticMeshesActors->Add(newPair);
    }

    copiedObject->OnDestroyed.AddUniqueDynamic(this,
&AInstancedStaticMeshWrapper::OnHeldObjectDestroyed);
}

AInstancedMeshOptimizableObject*
AInstancedStaticMeshWrapper::createObjectFromMeshInstance(int
meshInstanceIndex)
{
    FTransform meshTransform;
    const bool removedInstance = InstancedStaticMeshComponent-
>GetInstanceTransform(meshInstanceIndex, meshTransform, true);
    if (removedInstance)
    {
        TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>*
heldObjectInfo = getHeldMeshDataByIndex(meshInstanceIndex);

        if (heldObjectInfo != nullptr)
```

```
        {
            heldObjectInfo->Value = nullptr;
            InstancedStaticMeshComponent-
>RemoveInstance(meshInstanceIndex);
            AInstancedMeshOptimizableObject* keyGameObject =
Cast<AInstancedMeshOptimizableObject>(heldObjectInfo->Key);
            activateObject(keyGameObject);
            return keyGameObject;
        }
    }

    return nullptr;
}

void AInstancedStaticMeshWrapper::init(UStaticMesh* exampleMesh)
{
    _exampleMesh = exampleMesh;
}

void
AInstancedStaticMeshWrapper::addConvertableObject(AInstancedMeshOptimizab
leObject* addedObject)
{
    const TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>*
heldObjectInfo = getHeldMeshDataByObject(addedObject);

    if (heldObjectInfo == nullptr)
    {
        const TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>
newPair(addedObject, nullptr);
        _instancedStaticMeshesActors->Add(newPair);
    }

    createMeshInstanceFromObject(addedObject);
}

void
AInstancedStaticMeshWrapper::removeInstanceMeshInfoByObject(AInstancedMes
hOptimizableObject* removedObject)
{
    const int removedInstanceIndex = _instancedStaticMeshesActors-
>FindLastByPredicate([removedObject](TPair<AActor*, FBodyInstance*> data)
    {
        return removedObject == data.Key;
    });
```

```
    removeInstanceMeshInfoByIndex(removedInstanceIndex);
}

void AInstancedStaticMeshWrapper::removeInstanceMeshInfoByIndex(int
meshInstanceIndex) const
{
    if (meshInstanceIndex != INDEX_NONE)
    {
        _instancedStaticMeshesActors->RemoveAt(meshInstanceIndex);
        InstancedStaticMeshComponent->RemoveInstance(meshInstanceIndex);
    }
}

void AInstancedStaticMeshWrapper::BeginPlay()
{
    Super::BeginPlay();

    _instancedStaticMeshesActors->Empty();

    InstancedStaticMeshComponent->SetStaticMesh(_exampleMesh);
    InstancedStaticMeshComponent->SetMaterial(0, _exampleMesh-
>GetMaterial(0));

    InstancedStaticMeshComponent-
>SetCollisionProfileName(UCollisionProfile::PhysicsActor_ProfileName);
    InstancedStaticMeshComponent-
>OnComponentBeginOverlap.AddDynamic(this,
&AInstancedStaticMeshWrapper::OnStaticMeshBeginOverlap);
    InstancedStaticMeshComponent->OnComponentHit.AddDynamic(this,
&AInstancedStaticMeshWrapper::OnStaticMeshHit);

    //enableMeshCreatingTimer();
}

void AInstancedStaticMeshWrapper::BeginDestroy()
{
    delete _instancedStaticMeshesActors;
    _instancedStaticMeshesActors = nullptr;

    Super::BeginDestroy();
}

void AInstancedStaticMeshWrapper::Tick(float DeltaSeconds)
{
```

```
    Super::Tick(DeltaSeconds);

    //changeAllActorsToMeshes();
}

void
AInstancedStaticMeshWrapper::activateObject(AInstancedMeshOptimizableObje
ct* objectToEnable)
{
    objectToEnable->SetActorTickEnabled(true);
    objectToEnable->SetActorHiddenInGame(false);
    objectToEnable->SetActorEnableCollision(true);

    UMeshComponent* meshComponent = objectToEnable->getMeshComponent();
    meshComponent->SetSimulatePhysics(true);
    meshComponent->SetNotifyRigidBodyCollision(true);
}

void
AInstancedStaticMeshWrapper::deactivateObject(AInstancedMeshOptimizableOb
ject* objectToDisable)
{
    objectToDisable->SetActorTickEnabled(false);
    objectToDisable->SetActorHiddenInGame(true);
    objectToDisable->SetActorEnableCollision(false);
    objectToDisable->getMeshComponent()->SetSimulatePhysics(false);
}

void AInstancedStaticMeshWrapper::enableMeshCreatingTimer()
{
    if (!MeshCreatingTimer.IsValid())
    {
        GetWorldTimerManager().SetTimer(MeshCreatingTimer, this,
&AInstancedStaticMeshWrapper::enableOptimization,
            3.f, true);
    }
}

void AInstancedStaticMeshWrapper::disableMeshCreatingTimer()
{
    GetWorldTimerManager().SetTimer(RefreshingTimer, this,
&AInstancedStaticMeshWrapper::enableMeshCreatingTimer,
        3.f, false);
    if (MeshCreatingTimer.IsValid())
    {
```

```
        GetWorldTimerManager().ClearTimer(MeshCreatingTimer);
        disableOptimization();
    }
}

void AInstancedStaticMeshWrapper::disableOptimization()
{
    InstancedStaticMeshComponent-
>SetCollisionProfileName(UCollisionProfile::PhysicsActor_ProfileName);
    for (const TPair<AActor*, FBodyInstance*>& pair :
*_instancedStaticMeshesActors)
    {
        if (pair.Key != nullptr && pair.Key->IsHidden())
        {
            pair.Key->SetActorTickEnabled(true);
            pair.Key->SetActorHiddenInGame(false);
            pair.Key->SetActorEnableCollision(true);
        }
    }

    InstancedStaticMeshComponent->ClearInstances();
    InstancedStaticMeshComponent-
>SetCollisionProfileName("Interactable");

    for (TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>& pair :
*_instancedStaticMeshesActors)
    {
        UMeshComponent* meshComponent =
Cast<AInstancedMeshOptimizableObject>(pair.Key)->getMeshComponent();
        if (!meshComponent->IsSimulatingPhysics())
        {
            meshComponent->SetSimulatePhysics(true);
            meshComponent->SetNotifyRigidBodyCollision(true);
        }
    }
}

void AInstancedStaticMeshWrapper::enableOptimization()
{
    for (TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>& pair :
*_instancedStaticMeshesActors)
    {
        if
(isActorPreparedForChangeToStaticMesh(Cast<AInstancedMeshOptimizableObjec
t>(pair.Key)))
```

```
        {
            createMeshInstanceFromObject(pair.Key);
        }
    }
}

TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>*
AInstancedStaticMeshWrapper::getHeldMeshDataByIndex(int
meshInstanceIndex) const
{
    const FBodyInstance* bodyInstance = InstancedStaticMeshComponent-
>InstanceBodies[meshInstanceIndex];
    TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>* foundData =
        _instancedStaticMeshesActors-
>FindByPredicate([bodyInstance](TPair<AActor*, FBodyInstance*> data)
    {
        return bodyInstance == data.Value;
    });

    return foundData;
}

TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>*
AInstancedStaticMeshWrapper::getHeldMeshDataByObject(AInstancedMeshOptimi
zableObject* heldObject) const
{
    TPair<AInstancedMeshOptimizableObject*, FBodyInstance*>* foundData =
        _instancedStaticMeshesActors-
>FindByPredicate([heldObject](TPair<AActor*, FBodyInstance*> data)
    {
        return heldObject == data.Key;
    });

    return foundData;
}

bool
AInstancedStaticMeshWrapper::isActorPreparedForChangeToStaticMesh(AInstan
cedMeshOptimizableObject* actor)
{
    return !actor->IsHidden()
            && actor->GetAttachParentActor() == nullptr
            && actor->GetVelocity().Size() < 1.f
            && actor->getMeshComponent()->GetCollisionObjectType() ==
ECC_GameTraceChannel1;
```

```
}

void AInstancedStaticMeshWrapper::OnStaticMeshHit(UPrimitiveComponent*
HitComponent, AActor* OtherActor,
    UPrimitiveComponent* OtherComp, FVector NormalImpulse, const
FHitResult& Hit)
{
    //disableMeshCreatingTimer();
}

void
AInstancedStaticMeshWrapper::OnStaticMeshBeginOverlap(UPrimitiveComponent
* OverlappedComponent, AActor* OtherActor,
    UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool
bFromSweep, const FHitResult& SweepResult)
{
    if (SweepResult.bBlockingHit || OtherComp->GetCollisionObjectType()
== ECC_GameTraceChannel7)
    {
        return;
    }
    if (OtherBodyIndex == INDEX_NONE)
    {
        createObjectFromMeshInstance(SweepResult.Item);
    }
}

void AInstancedStaticMeshWrapper::OnHeldObjectDestroyed(AActor*
DestroyedActor)
{

removeInstanceMeshInfoByObject(Cast<AInstancedMeshOptimizableObject>(Dest
royedActor));
}
```