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

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: Програмний додаток – 2D графічний рушій

_____

Здобувача (ки) групи IT-01 Сухенича Максима Миколайовича
(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 СУХЕНИЧ_____
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доцент, к.т.н., доцент Світлана ВАЩЕНКО_____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) _____ (підпис)

Суми – 2024

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

ЗАТВЕРДЖУЮ

В. о. зав. кафедри ІТ

_____Світлана ВАЩЕНКО

«___» _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Сухеничу Максиму Миколайовичу

1 Тема роботи Програмний додаток – 2D графічний рушій

керівник роботи Ващенко Світлана Михайлівна к.т.н., доцент_____,

затверджені наказом по університету від « 29 » 04 2024 р. №0588-VI

2 Строк подання студентом роботи « 30 » 05 2024 р.

3 Вхідні дані до роботи технічне завдання

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити вступ; аналіз предметної області, проектування програмного додатку, розробка програмного додатку, висновки, список використаних джерел.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність роботи, меті та задачі, аналіз аналогів рушіїв, порівняльна таблиця аналізів, вимоги до програмного додатку, Структурно-функціональний аналіз, діаграма декомпозиції, діаграма варіантів використання, засоби реалізації, практична реалізація додатку, практична реалізація головної функції та життєвого циклу додатку, практична реалізація класу Application та Layer, практична реалізація та приклад використання 2D рендерера, Тестування способів

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання 20.12.2023

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	20.12.23 – 27.12.23	
2	Визначення властивостей застосунка	27.12.23 – 08.01.24	
3	Back-end розробка	08.01.24 – 10.07.24	
4	Frond-end розробка	10.07.24 – 24.07.24	
5	Тестування додатку	24.07.24 – 09.08.24	
6	Впровадження в дію	09.08.24 - 15.08.24	

Студент _____ Максим СУХЕНИЧ

(підпис)

Керівник роботи _____ к.т.н., доц. Світлана ВАЩЕНКО

(підпис)

АНОТАЦІЯ

Тема кваліфікаційної роботи бакалавра «Програмний додаток – 2D графічний рушій».

Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел із 25 найменувань, додатків. Загальний обсяг роботи – 106 сторінок, у тому числі 42 сторінок основного тексту, 2 сторінки списку використаних джерел, 60 сторінок додатків.

Актуальність роботи полягає в необхідності створення доступного інструменту для розробки графічних двовимірних ігор, що не вимагає глибоких знань у графічному програмуванні.

Головною метою роботи є розробка додатку, який буде відрізнятися простим для оволодіння інтерфейсом та вільним доступом, а також забезпечувати абсолютний контроль над редагуванням внутрішнього коду.

В першому розділі проведено дослідження предметної області та публікацій, проаналізовано сучасні графічні рушії, їхні функціональні компоненти та основні принципи роботи.

В розділі 2 висвітлено процес проектування програмного додатку: функціональне моделювання у відповідності до стандарту IDEF0, де детально розглядаються функції та їхні взаємозв'язки; розробка діаграми варіантів використання; діаграма класів, що відображає структуру програми та взаємозв'язки між класами для досягнення поставлених цілей.

В розділі 3 наведено процес розробки програмного додатку. Розглядається структура та внутрішній механізм програми, описано основні кроки та етапи розробки програмного забезпечення з використанням необхідних технологій та інструментів. Наприкінці розділу розглянуто процес тестування способів використання додатку.

Ключові слова: графіка, аудіомодуль, рушій рендерингу, модуль анімації, модуль ігрових механік, програмний додаток, 2D-сцени, рендеринг.

ЗМІСТ

ВСТУП

День за днем ігрові інформаційні технології швидко набирають обертів. Ігрова індустрія перетворилася із суто розважального сегменту в повсякденне життя більшості студентів, школярів та навіть дорослих.

Галузь відеоігор значно зросла і перетворилася на суттєву складову сучасного бізнесу. На даний момент, однією з актуальних проблем є доступність певних інструментів для створення ігор без потреби у глибоких знаннях в графічному програмування.

Сьогодні у цій галузі не так багато конкурентів, проте вони відзначаються масштабною потужністю та популярністю. Щоб зберегти конкурентоспроможність, продукт має вражати своєю потужністю, або визначатися спеціалізацією у певному напрямі. Це допоможу йому бути помітним серед лідерів галузі, привертати увагу користувачів та здобувати популярність серед споживачів.

Наявність легкодоступного, безкоштовного, простого у використанні та спрямованого графічного рушія є фундаментальним елементом для ефективного створення відеоігор. Графічний рушій є основним інструментом, який формує візуальні аспекти відеогри, допомагаючи розробникам створювати відеоігри з легкістю та ефективністю, без поглибленого розуміння понять графічного програмування. Це дозволяє розробникам зосередитися на креативних аспектах гри не витрачаючи час на вивчення концепцій графічного програмування. Однак, сьогодні, усі графічні рушії вимагають великих обсягів пам'яті та високої продуктивності комп'ютера через свою багатофункціональність, а також не безкоштовними і це може бути певним обмеженням для певної категорії користувачів.

Тому розробка графічного рушія є актуальною задачею, як і для того, щоб забезпечити користувачів зручним і безплатним додатком, так і для того, щоб забезпечити повний контроль і вільність дій над додатком, не потребуючи при цьому високої продуктивності комп’ютера.

Отже, основною метою даного дослідження є розробка графічного рушія, спрямованого на створення ігор та симуляцій у двовимірному просторі. Головною ціллю є забезпечення користувача усіма необхідними інструментами для цього, надаючи йому повний контроль над технічними аспектами та конструктор для візуального програмування. Також, не менш важливим аспектом буде невелика вага із збереженням високої продуктивності, та усіх переваг гігантів даної індустрії.

Щоб досягнення мети проекту потрібно виконати наступні задачі:

- визначити актуальність роботи, дослідити предметну область та провести аналіз аналогів графічних рушіїв;
- розробити чіткі функціональні вимоги до графічного рушія;
- провести функціональне моделювання роботи графічного рушія;
- розробити та реалізувати структуру та функціонал графічного рушія;
- протестувати роботу розробленого програмного додатку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень

В основі більшості сучасних ігор чи симуляцій лежить графічний рушій – це основний інструмент для створення графічних застосунків [1].

Графічний рушій є абстракцією для взаємодії з графічним обладнанням девайсу. Рушій надає доступ до методів за допомогою яких користувач формує довкілля свого застосунку. Головними функціями будь якого рушія є інструменти для побудови рівнів та взаємодій між об’єктами чи локаціями.

Графічний рушій являє собою потужний інструмент для реалізації сцен та ігор, призначеним для користувачів з лімітованими знаннями у графічному програмуванні [2]. Тому простота та доступність є головними аспектами. Додаток повинен відрізнятися серед інших, але включати головні компоненти, такі як конструктор рівня, підтримку текстур, скриптинг, анімація, аудіо та інші.

Розглядаючи існуючі графічні рушії, можна виділити наступні основні компоненти графічного рушія [3]:

- аудіо модуль (Audio module) – головною метою даного компоненту є адаптація звукових ефектів у простір графічного рушія;
- рушій рендерингу (Rendering Engine) – ціллю даного компонента є перетворення вхідних даних у пікселі на екрані користувача [4];
- модуль Анімації (Animation module) – компонент, який відповідає за анімацію об’єктів у сцені;
- модуль ігрових механік (Game mechanics) – компонент, який керує правилами віртуального світу [4];
- програмні інструменти (Software Tools) – інструменти, які дозволяють оптимізувати роботу додатку і покращати ефективність роботи графічного рушія.

Перед користувачем без досвіду роботи з графічними застосунками, вибір графічного рушія може постати серйозною проблемою. Засилаючись на дослідження, у якому було проведено опитування щодо того який графічний рушій є кращим[5], можна побачити що однозначної відповіді на це запитання не існує. Кожен графічний рушій пропонує свої переваги та недоліки. Виходячи з цього, перш ніж обирати графічний рушій, користувачеві слід покористуватися різними графічними рушіями і обрати той, у якому буде найзручніший інтерфейс та функціонал в відповідності до поставлених користувачем задач [5].

У той час як графічний рушій повинен забезпечити інтерфейс для побудови локацій, важливо, щоб у користувача був повний доступ та контроль над впровадження своїх ідеї, зокрема оптимізація, спосіб рендеренгу чи структури шарів у програмі. Забезпечення змоги імпортувати текстури та аудіо є обов’язковим компонентом для досягнення високого рівня креативності у розробці сцен та ігор [6].

1.2 Аналіз існуючих продуктів-аналогів

На сьогоднішній день існує велика кількість графічних рушіїв, серед яких найвідоміші «Unity», «Unreal Engine», але також існують і менш відомі, такі як «Frostbite», «Source», «Rockstar» «Advanced Game Engine», які, власне кажучи, є закритими системами і недоступними для звичайних користувачів.

Зважаючи на це, критичним фактором конкурентоспроможності є розробка доступного та надійного програмного продукту. Для формування вимог до майбутнього графічного рушія виконано аналіз існуючих аналогів, таких як «Unity», «Unreal Engine» і «Frostbite».

«Unity» є одним з провідних графічних рушіїв у галузі розробки графічних додатків, з неймовірно великою кількістю успішних проектів. Близько 38% усіх розробників ігор використовують «Unity», як свій головний графічний рушій для розробки ігор [7]. Цей рушій характеризується надійністю та

багатофункціональністю. Однією з головних переваг «Unity» є підтримка розробки ігор як у двовимірному так і тривимірному просторах [8].

«Unity» надає велике різноманіття функцій та ресурсів (рис. 1.1) та має зручний інтерфейс.

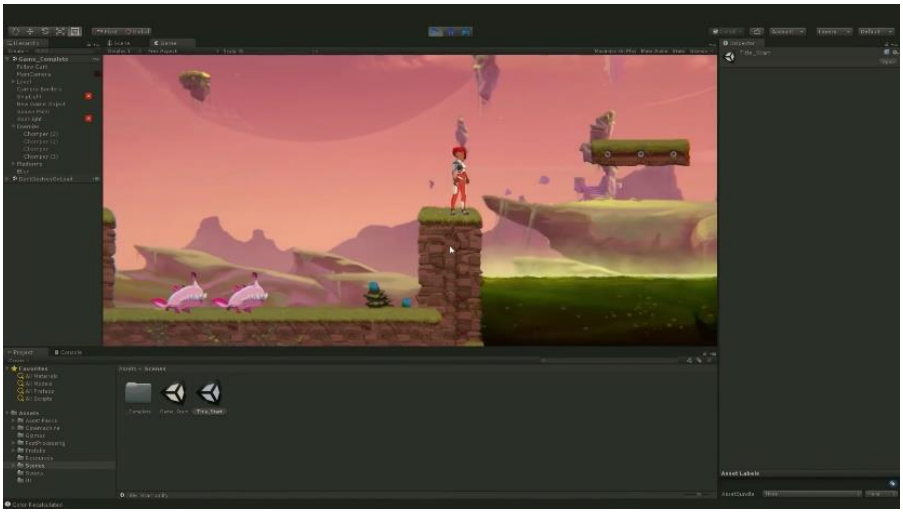


Рисунок 1.1 – Інтерфейс двовимірного робочого простору у «Unity»

Однією з головних функцій «Unity» є можливість розробки мобільних додатків, що визначило його репутацію та популярність (рис. 1.2). Проте варто зазначити, що при побудові значних та графічно складних проектів у рушія можуть виникнути деякі обмеження щодо продуктивності та швидкодії [9].

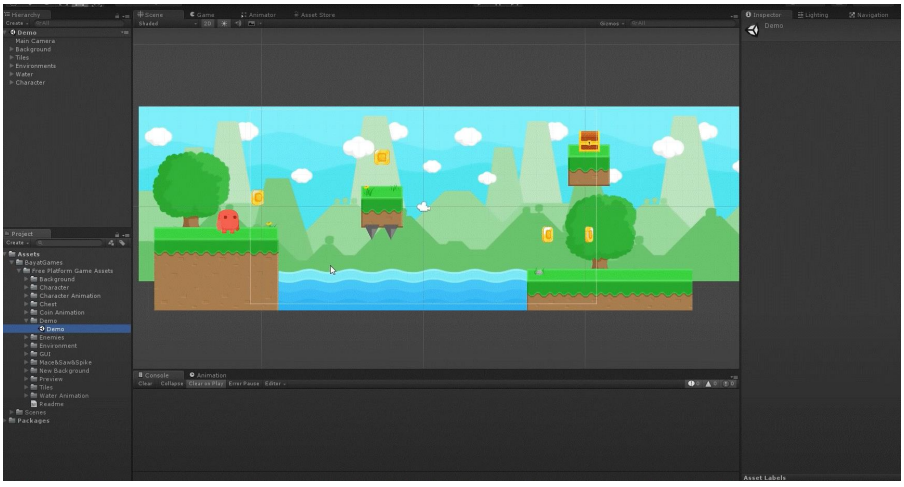


Рисунок 1.2 – Приклад створення сцени у графічному рушію «Unity».

Важливо враховувати, що починаючи з 2024 року графічний рушій «Unity» буде вимагати плату за скачування опублікованого додатка створеного на його основі [10]що може стати проблемою для розробників, які планують створювати безкоштовні ігри. Це обмеження вартості може вплинути на доступність рушія для окремих користувачів та проектів.

«Unreal Engine» є ще одним неймовірно потужним графічним рушієм у галузі розробки відеоігор та графічних додатків. Більшість професіоналів у галузі обирають саме його, що у свою чергу, підтверджує його домінацію на ринку [10] (рис.1.3).

Рисунок 1.3 – Приклад сцени у графічному рушію «Unreal Engine».

Не зважаючи на те що «Unreal Engine» був розроблений для створення реалістичних сцен та реалістичної графіки, в останні версіях додатка була додана можливість розробляти двовимірні ігри (рис. 1.4). Слід відзначити, що порівняно з «Unity», «Unreal Engine» має значно менший набір інструментів та засобів для розробки двовимірних додатків [11].

Рисунок 1.4 – Приклад 2D сцени у графічному рушію «Unreal Engine»

До недоліків «Unreal Engine» слід віднести обмежену кросплатформність. Цей недолік значно зменшує гнучкість додатків на його основі. На додачу до цього, як додаток так і ігри створені на його основі часто є дуже громіздкі [12].

Також важливо відзначити, що «Unreal Engine» не є безкоштовним і знімає певний відсоток від продажу продуктів, створених на його основі [12].

«Frostbite Engine» - це графічний рушій який розроблений відомою компанією «Electronic Arts». Цей додаток доступний лише для внутрішнього використання та для студій які співпрацюють з «Electronic Arts». Рушій відзначається неймовірною якістю графіки та реалістичністю деталей (рис. 1.5) [13].

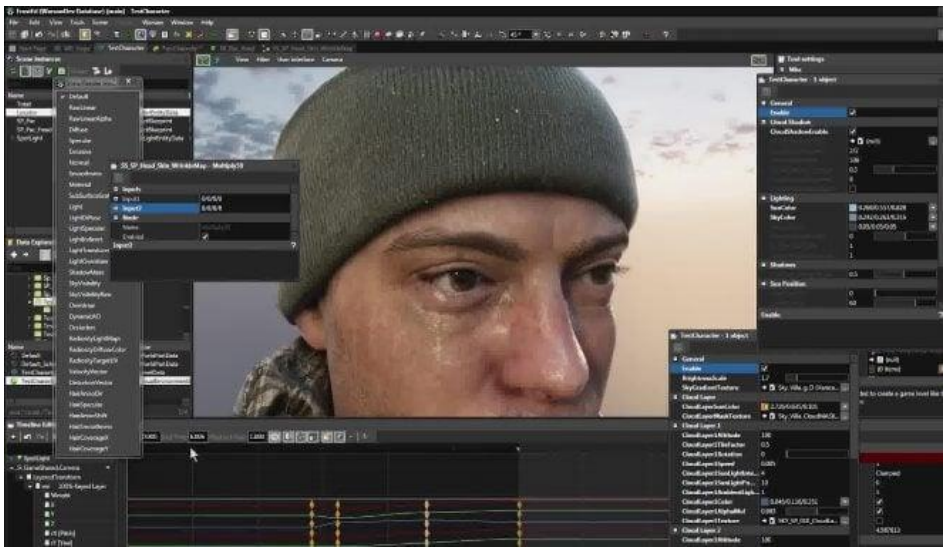


Рисунок 1.5 – Приклад створення реалістичної графіки у рушію «Frostbite».

До переваг графічного рушія «Frostbite» слід віднести функціонал, який дозволяє розробникам контролювати код на найнижчому рівні і забезпечує ефективну адаптацію для створення та оптимізації гри в різних жанрах [14]. Також варто відзначити часткову підтримку 2D графіки та кросплатформність.

Проте важливо враховувати, що доступ до «Frostbite» є обмежений. Додаток доступний лише для працівників компанії «Electronic Arts» та їх партнерів. Звичайні користувачі не можуть використовувати цей рушій для своїх проектів. До недоліків «Frostbite» можна віднести його складність в певних аспектах, таких як фізика чи анімація (рис. 1.6), що у свою чергу ускладнює роботу розробників порівняно з іншими рушіями [15].

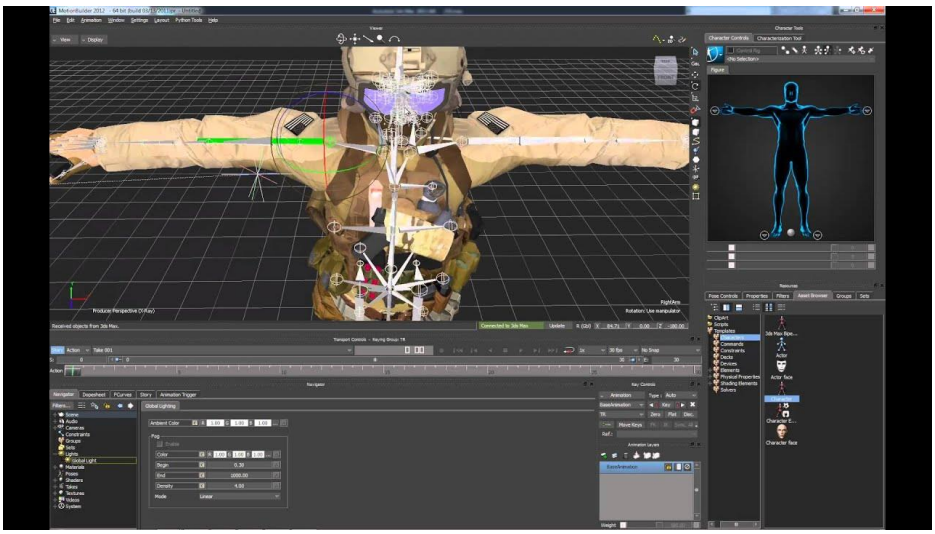


Рисунок 1.6 – Приклад створення анімацій у графічному рушію «Frostbite».

Після ретельного аналізу аналогів графічних рушіїв, було визначено та проаналізовано їх переваги та недоліки. Результати представлені в таблиці 1.1

Таблиця 1.1 – Порівняльна характеристика аналогічних 3D моделей

Критерії порівняння	Графічні рушії		
	«Unreal Engine»	«Unity»	«Frostbite»
Інтерфейс	Потужний інтерфейс, але складний для початківців	Легкий та доступний для користувачів на всіх рівнях	Відомий своєю складністю для освоєння
Кросплатформність	Непогана підтримка різних платформ	Вражаюча підтримка різних платформ, включаючи мобільні	Кросплатформність обмежена

Продовження табл. 1.1.

Доступність	Безкоштовний лише для освітянських та некомерційних проектів	Безкоштовний лише для освітянських та некомерційних проектів	Обмежений доступ, використовується внутрішньо в EA
Контроль над кодом	Забезпечує повний контроль над кодом, можливість програмування на C++	Забезпечує низький рівень контролю, можливість програмування на C# та Javascript	Повний контроль, але із обмеженим доступом до коду
2D розробка	Слабка підтримка 2D лише з акцентом на 3D розробці	Сильна підтримка 2D розробки, придатний для створення ігор різних жанрів	Часткова підтримка 2D, але менш розвинена порівняно з іншими функціями
Інтеграція	Підтримує широкий спектр інструментів та мов програмування	Має низьку сумісність з багатьма іншими інструментами	Рівень інтеграції залежить від конкретних проектів та можливостей інтеграції
Спільнота та Ресурси	Має активну та велику спільноту розробників та широкий вибір ресурсів	Велика та активна спільнота розробників, багато онлайн-ресурсів	Офіційна спільнота може бути обмеженою через обмежений доступ

Дані з таблиці 1.1 надають змогу зробити висновок про те які функціональні доповнення слід використати, і недоліки, яких слід уникати. Даний програмний продукт повинен володіти повною функціональністю для розробки двовимірних додатків, бути дружелюбним для користувачів початківців, а також забезпечувати безкоштовний доступ. Важливо, щоб цей продукт гарантував повний контроль над кодом, підтримував скриптинг і дозволяв ефективно інтегрувати різноманітні інструменти та мови програмування.

1.3 Постановка задачі

Головною метою даного дослідження є розробка графічного рушія, який буде відрізнятися простим для оволодіння інтерфейсом та вільним доступом, в додачу до цього забезпечувати абсолютний контроль над внутрішнім кодом та оптимізацією.

Основні завдання щодо створення програмного продукту є такими:

- розробити дизайн графічного рушія;
- підготувати основу для створення програми;
- впровадити підготовчий код, такий як логування та визначення точки входу (entry point);
- обрати інтерфейс програмування застосунків (API) для розробки графічної частини;
- розробити систему обробки подій, управління шарами та введенням;
- розробити систему інтерфейсу;
- розпочати розробку рендерера;
- реалізувати систему камери;
- впровадити обробку текстур;
- розпочати оптимізацію рендерера;
- розробити сцену для представлення об'єктів;
- розробити користувацький інтерфейс для взаємодії з програмою;

- провести тестування готового продукту.

Більш детальні вимоги до проекту наведено у технічному завданні на розробку проекту (додаток А).

1.4 Вибір засобів реалізації

Для втілення даного додатку були вибрані наступні технології:

- основна мова програмування - «C++», з використанням засобів «STL»;

«C++» є низькорівневою мовою програмування, яка забезпечує необхідні інструменти для побудови оптимізованого та багатофункціонального застосунку.

- графічна бібліотека застосунку - «OpenGL» ;

«OpenGL» пропонує користувачеві легкий та зрозумілий доступ до потужностей графічної карти. На відміну від інших графічних бібліотек, «OpenGL» відзначається простотою реалізації, та достатньою потужністю і оптимізацією для побудови складних графічних додатків [16].

- створення інтерфейсу - бібліотека «imgui»;

«imgui» - це одна з найпопулярніших бібліотек для створення графічного інтерфейсу. Бібліотека представляє набір функцій для побудови тестового або графічного інтерфейсу програми [17].

- математична підтримка - «GLM» ;

«GLM» - Математична бібліотека яка представляє собою великий набір функцій для роботи з векторами, матрицями, та іншими математичними типами даних [18].

- завантаження текстур - бібліотека «stb_image»;

«stb_image» - Проста у використанні бібліотека для завантаження зображень різних форматів.

- сутнісно-компонентна система – «entt»;

«entt» - Бібліотека призначена для реалізації сутнісно-компонентної системи, яка є невід’ємною частиною будь якого графічного рушія. «entt» пропонує зручний та гнучкий спосіб керування сутностями та компонентами у графічній сцені [19].

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

2.1 Функціональне моделювання програмного додатку в IDEF0

IDEF0 – це графічно тестовий метод моделювання. IDEF0 забезпечує високий рівень розуміння, підтримки аналізу, визначення вимог системи. Також даний метод моделювання підтримує моделювання та інтеграцію проекту на системному рівні. IDEF0 призначена для моделювання систем різних типів, таких як програмне чи апаратне забезпечення. Іншою перевагою IDEF0 є можливість використання її для аналізу функцій систем та документування відповідних механізмів [20].

Функціональне моделювання графічного рушія «MyRevoke» в нотації IDEF0 представлена на рисунку 2.1.

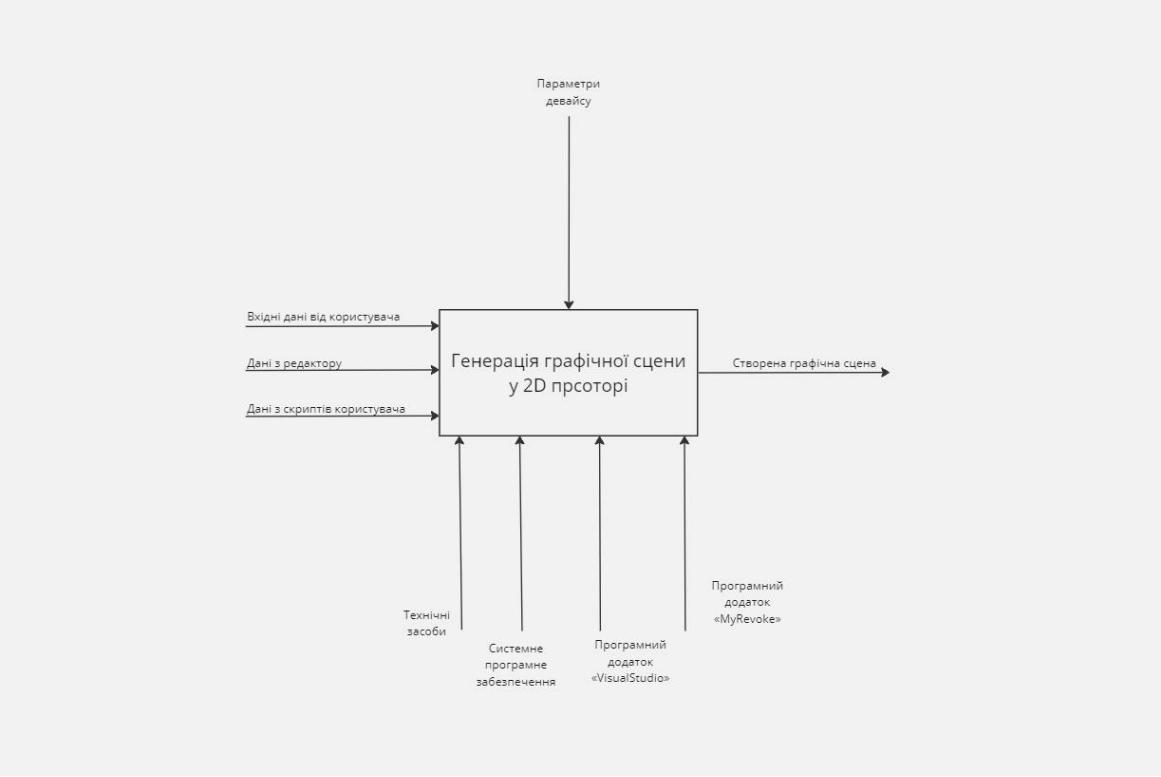


Рисунок 2.1 – Функціональна діаграма IDEF0 для додатку «MyRevoke»

На вхід поступають дані від користувача, такі як налаштування текстур звуків, об’єктів, дані з редактора «RevokeCraft», такі як створені об’єкти,

налаштування сцени, та дані з скриптів користувача, які описують логіку об’єктів у сцені. Реалізація основного процесу відбуватиметься за рахунок використання програмного додатку «MyRevoke». Адаптація під систему девайсу відбувається в залежності від параметрів девайсу. Технічні засоби, системне програмне забезпечення та програмний додаток «MyRevoke» є механізмами реалізації. При виході з додатку, отримуємо Створену графічну сцену.

Наступним етапом розробки моделі IDEF0 є поділення процесу на складові. Цей етап називається декомпозиція контексту. Декомпозиція є основним поняттям стандарту IDEF0. Для побудови успішної моделі декомпозиції необхідно розбити складний процес на його складові [20].

Декомпозиція 1-го рівня моделювання графічного рушія «MyRevoke» в нотації IDEF0 представлена на рисунку 2.2.

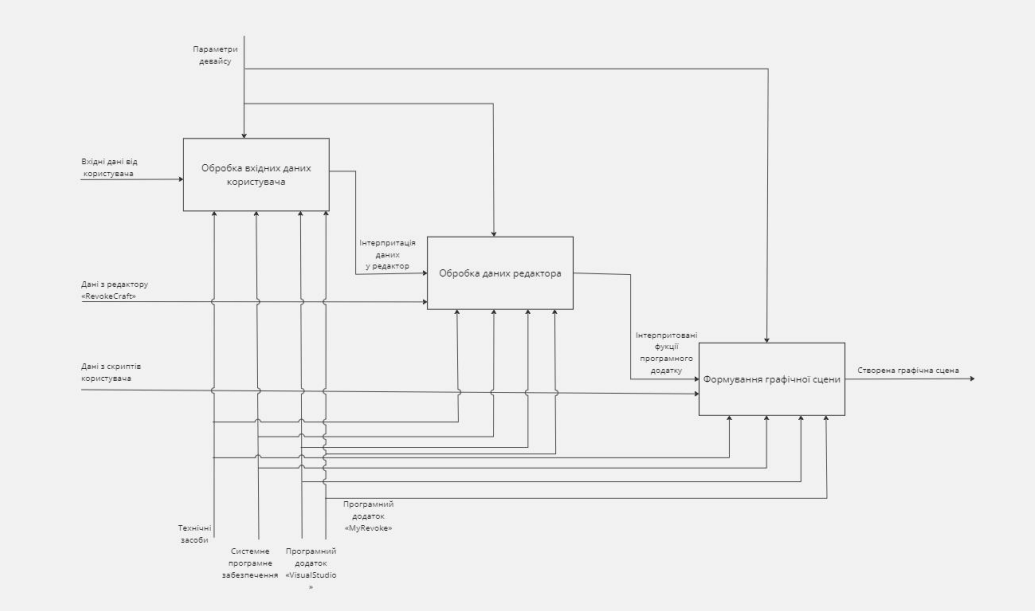


Рисунок 2.2 – Декомпозиція функціональної моделі для додатку «MyRevoke»

Створення графічної сцени складається з таких етапів, як обробка вхідних даних користувача, обробка даних редактора «RevokeCraft», формування графічної сцени. Обробка вхідних даних користувача – це перший етап, який інтерпретує дані які вводить користувач при взаємодії з додатком. Обробка даних редактора

«RevokeCraft» – це другий етап, який полягає у обробці усіх даних введених користувачем і згенерованих редактором. Формування графічної сцени – це четвертий етап, який викликає усі необхідні функції програмного додатку для створення графічної сцени.

2.2 Діаграма варіантів використання

UML – це набір діаграми варіантів використання, який являє собою стандартизовану мову моделювання. Діаграми варіантів використання використовуються для опису функцій та областей застосування систем та відображення взаємодії між акторами та системою. Головною ідеєю діаграми варіантів використання є опис функціональності системи без опису того як саме це виконується [21].

Діаграма варіантів використання в нотації UML для графічного рушія «MyRevoke» представлена на рисунку 2.3.

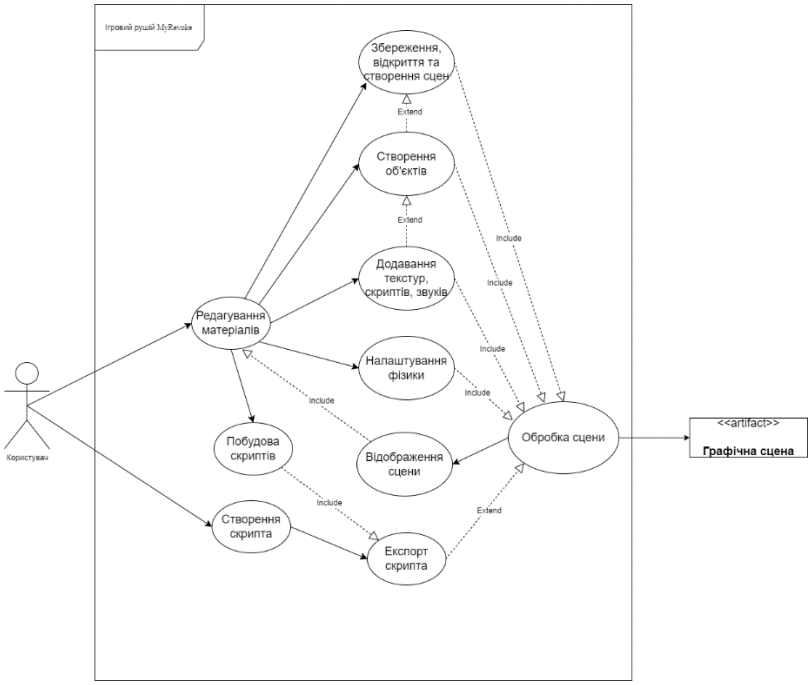


Рисунок 2.3 – Діаграма варіантів використання для графічного рушія «MyRevoke»

У даному додатку є один актор: − Користувач: людина яка завантажила графічний рушій «MyRevoke» і створює графічну сцену за допомогою графічного редактора «RevokeCraft» і скриптів. Людина створює свій скрипт, редактор будує його dll файл та додає до рушія.

2.3 Діаграма класів

Для побудови головної структури додатку буде використано технології об’єктно-орієнтованого програмувння, які дозволяють ефективно та зручно будувати програмні додатки, використовуючи інтерфейсні класи, наслідування, поліморфізм. Тому для відображення структури програмної реалізації графічного рушія «MyRevoke» слід використати діаграму класів. Діаграма класів відображає структуру додатку використовуючи графічне модулювання. Дана діаграма вказує на класи, їх атрибути, методи та взаємозв’язки між ними [22].

Діаграми класів додатку «MyRevoke» зображено на рис. 2.4.

Клас “Application” відповідатиме за життєвий цикл додатку, та усі необхідні функціональні можливості для підтримки його роботи.

Клас “Layer” є класом інтерфейсом, який дозволяє користувачеві створювати свої шари усередині його додатку, також він використовується як батьківський клас для побудови інтерфейсу додатку.

Клас “Camera” відповідає за усю логіку відображення 3D простору на двовимірному екрані.

Клас “Interface” реалізує усі необхідні частини інтерфейса застосунка.

Клас “Scene” Відповідає за все що зв’язане з побудовою, роботою, та збереженням графічних сцен.

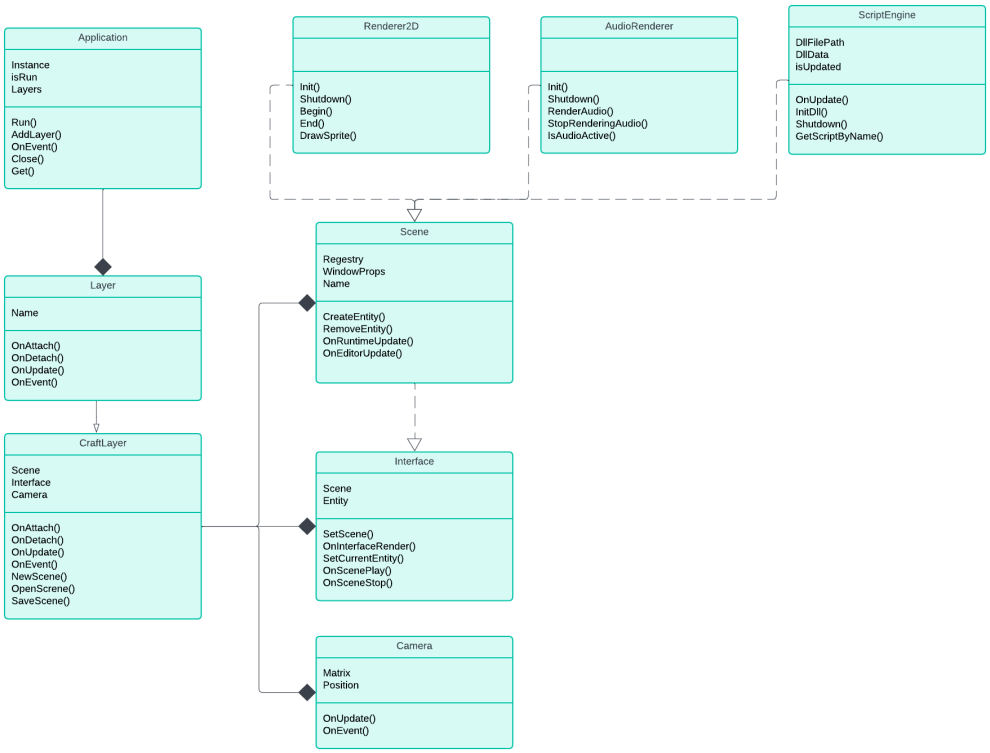


Рисунок 2.4 – Діаграма класів додатку «MyRevoke»

Клас “Renderer2D” тримає у собі усі необхідні функції для відображення графіки на екрані користувача.

Клас “AudioRenderer” та клас “Script Engine” Відповідають за звуки та скрипти відповідно.

3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1 Архітектура додатку

Структура додатку буде складатися з двох головних модулів та двох допоміжних. До головних відноситься модуль «MyRevoke» - ядро додатку, у якому будуть розташовані усі функції рендеренгу і життя програми. Другим буде «RevokeCraft» - модуль користувацького графічного інтерфейсу, у якому користувачеві буде надано графічний доступ до функціональності головного модуля «MyRevoke». «MyRevoke-NativeScriotCore» - допоміжний модуль для побудови скриптів, які у свою чергу забезпечують користувачу вільний доступ до функціональностей модуля «MyRevoke» і свободу створення і імплементації своїх ідей. Модуль «SandBox» - це проект без інтерфейсу націлений на тестування окремих функціональних можливостей додатку.

Модуль «MyRevoke» є ядром програмного додатку і відповідає за основну функціональність та життєвий цикл додатку у цілому. Даний модуль можна розділити на наступні ключові компоненти:

1. Класи та методи рендеренгу – компоненти які відповідають за створення графічних сцен.
2. Класи та методи скриптингу – компоненти які відповідають за зв’язок між модулем «MyRevoke» та скриптами користувача.
3. Класи та методи аудіо менеджеру – компоненти які відповідають за обробку звуку у графічних сценах додатку.
4. Класи та методи системи подій – компоненти які відповідають за події у графічних сценах створених користувачем.
5. Класи та методи системи шарів – компоненти які за допомогою наслідування створюють зв'язок між ядром програми і графічним інтерфейсом.

6. Клас управління додатком – основний клас ядра програми, який відповідає за життєвий цикл програми і управління додатком.

Модуль «RevokeCraft» відповідає за графічний інтерфейс додатку. З головних компонентів модуля слід виділити наступні:

1. Класи та методи, які відповідають за вікно у якому буде відображена графічна сцена.
2. Класи та методи панелі сутностей – область інтерфейсу у якій користувач може створювати, обирати та видаляти сутності.
3. Класи та методи контент браузера – область інтерфейсу у якій користувач може змінювати сцени, додавати скрипти, звуки та текстури до обраної графічної сцени.
4. Класи та методи налаштувань сцени – область у якій користувач може налаштовувати головні аспекти сцени, такі як налаштування фізики чи фон.
5. Клас управління інтерфейсом – основний клас-шар який унаслідує систему шарів головного модуля «MyRevoke» і відповідає за головні аспекти управління інтерфейсом.

Архітектура програмного додатку «MyRevoke» зображено на рис. 3.1

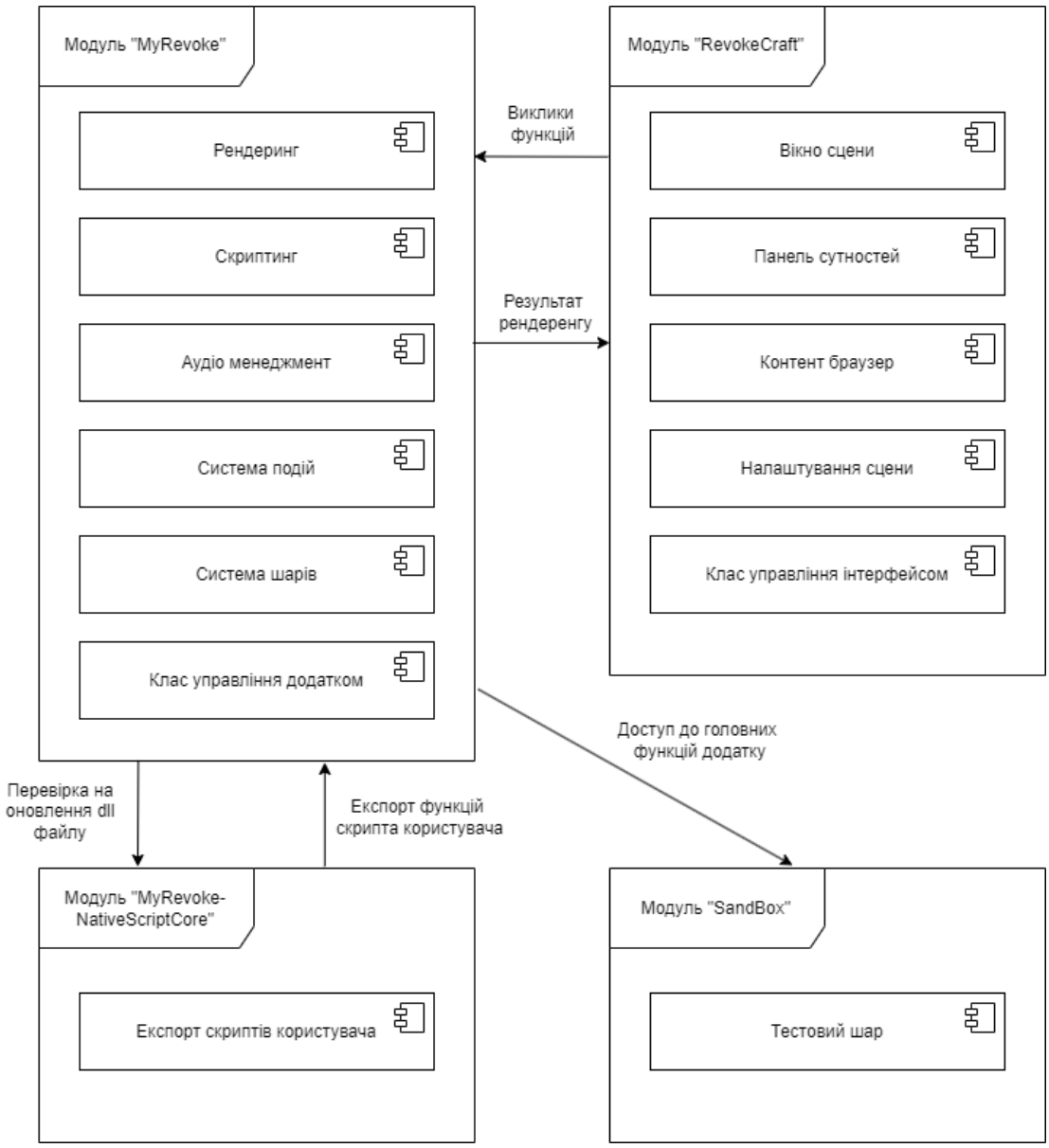


Рисунок 3.1 – Архітектура програмного додатку «MyRevoke»

3.2 Програмна реалізація програмного додатку

Для розробки додатку «MyRevoke» в якості IDE було обрано Visual Studio та premake5, як механізм побудови проекту. Visual Studio забезпечує розробника усіма необхідними інструментами компіляції та дебагінгу коду (рис 3.2). Інструмент premake5 є простішим аналогом відомої системи побудови проектів CMake та пропонує легкий спосіб побудови структури проектів мовою програмування Lua (рис 3.3).

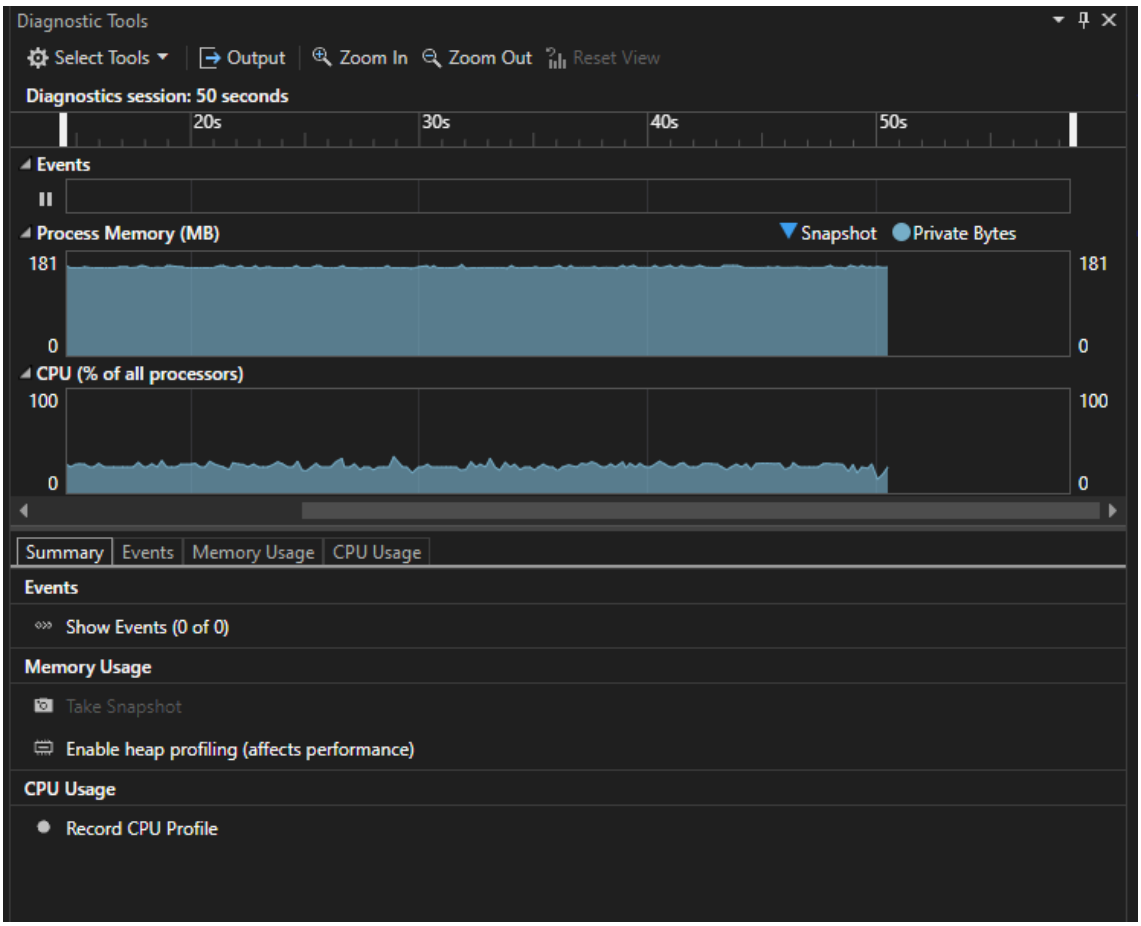
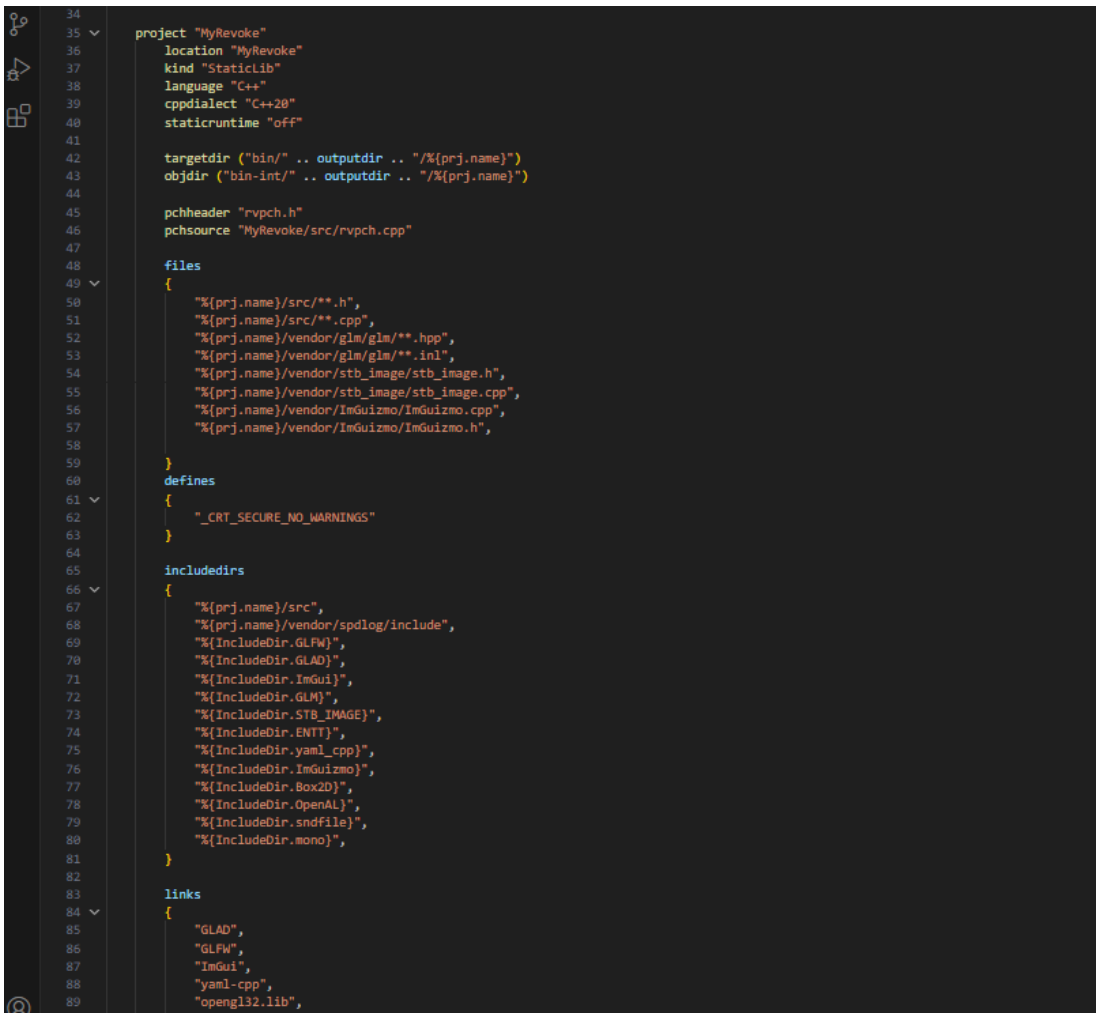


Рисунок 3.2 – Приклад засобів дебагінгу в Visual Studio



```
project "MyRevoke"
    location "MyRevoke"
    kind "StaticLib"
    language "C++"
    cppdialect "C++20"
    staticruntime "off"

    targetdir ("bin/" .. outputdir .. "/%{prj.name}")
    objdir ("bin-int/" .. outputdir .. "/%{prj.name}")

    pchheader "rvpch.h"
    pchsource "MyRevoke/src/rvpch.cpp"

    files
    {
        "%{prj.name}/src/**.h",
        "%{prj.name}/src/**.cpp",
        "%{prj.name}/vendor/glm/glm/**.hpp",
        "%{prj.name}/vendor/glm/glm/**.inl",
        "%{prj.name}/vendor/stb_image/stb_image.h",
        "%{prj.name}/vendor/stb_image/stb_image.cpp",
        "%{prj.name}/vendor/ImGuizmo/ImGuizmo.cpp",
        "%{prj.name}/vendor/ImGuizmo/ImGuizmo.h",

    }
    defines
    {
        "_CRT_SECURE_NO_WARNINGS"
    }

    includedirs
    {
        "%{prj.name}/src",
        "%{prj.name}/vendor/spdlog/include",
        "%{IncludeDir.GLFW}",
        "%{IncludeDir.GLAD}",
        "%{IncludeDir.ImGui}",
        "%{IncludeDir.GLM}",
        "%{IncludeDir.STB_IMAGE}",
        "%{IncludeDir.ENTT}",
        "%{IncludeDir.yaml_cpp}",
        "%{IncludeDir.ImGuizmo}",
        "%{IncludeDir.Box2D}",
        "%{IncludeDir.OpenAL}",
        "%{IncludeDir.sndfile}",
        "%{IncludeDir.mono}",
    }

    links
    {
        "GLAD",
        "GLFW",
        "ImGui",
        "yaml-cpp",
        "opengl32.lib",
```

Рисунок 3.3 – Приклад побудови модуля «MyRevoke» засобами premake5

Розробка починається з реалізації модуля «MyRevoke». Головними функціям даного модуля повинні бути рендеринг, життєвий цикл додатку, менеджмент шарів, подій, і точки входу.

Для реалізації головної функції програми було використано популярну технологію, у якій користувач створює визначення функції у редакторі, яка викликається з функції main в ядрі додатку, і починається життєвий цикл додатку [23] (рис. 3.4).

Реалізація діаграми функції входу зображена на рисунку 3.5.

```
extern Revoke::Application* Revoke::CreateApplication();

int main()
{
    Revoke::Log::Init();
    Revoke::Application* application = Revoke::CreateApplication();
    application->Run();
    delete application;
}
```

Рисунок 3.4 – Головна функція додатку «MyRevoke»

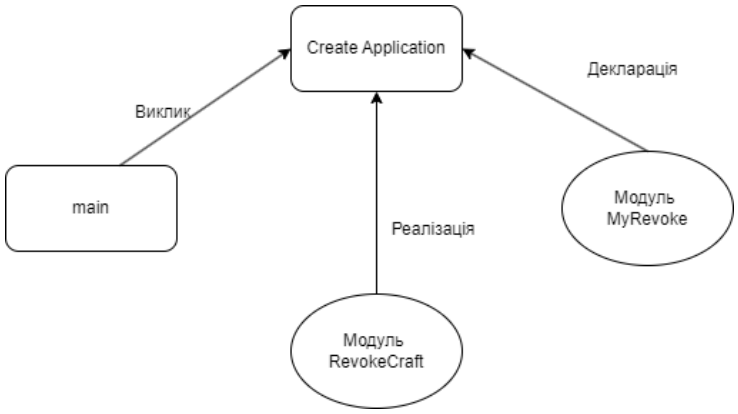


Рисунок 3.5 – Діаграма головна функція додатку «MyRevoke»

Наступним кроком є створення класу “Application”, який буде відповідати за життєвий цикл додатку, контролювати потік застосунку та ініціалізувати необхідні класи. Клас створено технологією “Singleton”, яка являє собою клас, який дозволяє створити лише один екземпляр цього класу, тримаючи у собі статично єдиний екземпляр цього класу.

Реалізація класу входу “Application” зображена на рисунку 3.6.

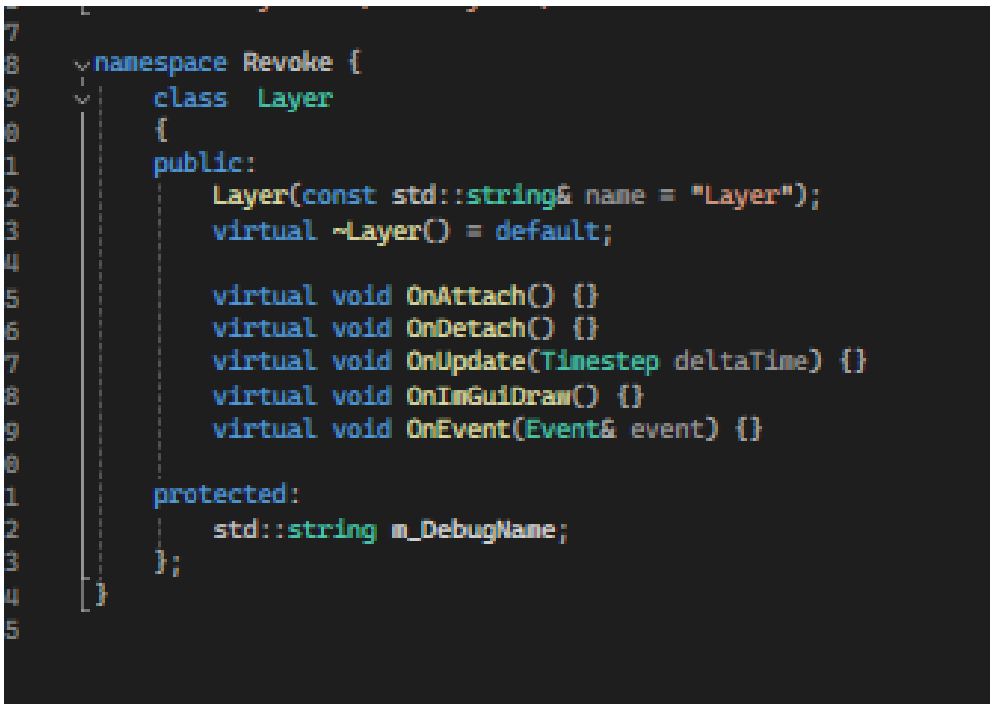
```
namespace Revoke
{
    class  Application
    {
    public:
        Application();
        virtual ~Application();

        void Run();

        void OnEvent(Event& e);

        void PushLayer(Layer* layer);
        void PushOverlay(Layer* layer);

        void Close();

        ImGuiLayer* GetImGuiLayer() { return m_ImGuiLayer; }

        static Application& Get() { return *s_Instance; }
        Window& GetWindow() { return *m_Window; }

    private:
        bool OnWindowClose(WindowsCloseEvent e);
        bool OnWindowResize(WindowResizeEvent e);
    private:
        Shared<Window> m_Window;

        LayerStack m_LayerStack;
        ImGuiLayer* m_ImGuiLayer;

        float m_LastFrame = 0;

        bool m_Run = true;

    private:
        static Application* s_Instance;
    };
```

Рисунок 3.6 – Головна клас додатку «MyRevoke»

Щоб уникнути проблеми конфліктів вікна GLFW та графічного інтерфейсу DearImGui необхідно розробити систему шарів. Клас “Layer” буде батьківським класом для усіх шарів. У ньому буде задекларовано інтерфейс функцій, у яких відбуватиметься більшість логіки додатку (рис. 3.7). Дані функції будуть викликатися з головного циклу у класі “Application” (рис. 3.8).

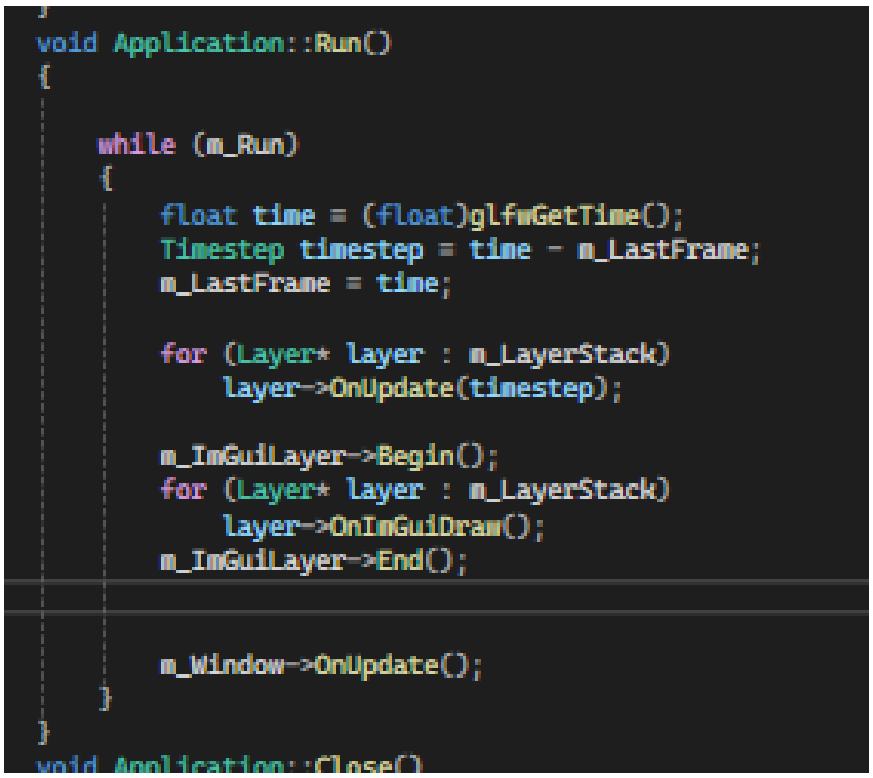
Клас “ImGuiLayer” наслідуватиме функції “Layer” і виконуватиме логіку графічного інтерфейсу.

На рисунку 3.9 зображена діаграма роботи життєвого циклу застосунку.



```
namespace Revoke {
    class  Layer
    {
    public:
        Layer(const std::string& name = "Layer");
        virtual ~Layer() = default;

        virtual void OnAttach() {}
        virtual void OnDetach() {}
        virtual void OnUpdate(Timestep deltaTime) {}
        virtual void OnImGuiDraw() {}
        virtual void OnEvent(Event& event) {}

    protected:
        std::string m_DebugName;
    };
}
```

Рисунок 3.7 – Клас-інтерфейс “Layer”



```
void Application::Run()
{

    while (m_Run)
    {
        float time = (float)glfwGetTime();
        Timestep timestep = time - m_LastFrame;
        m_LastFrame = time;

        for (Layer* layer : m_LayerStack)
            layer->OnUpdate(timestep);

        m_ImGuiLayer->Begin();
        for (Layer* layer : m_LayerStack)
            layer->OnImGuiDraw();
        m_ImGuiLayer->End();


        m_Window->OnUpdate();
    }
}
void Application::Close()
```

Рисунок 3.8 – Головний цикл додатку «MyRevoke»

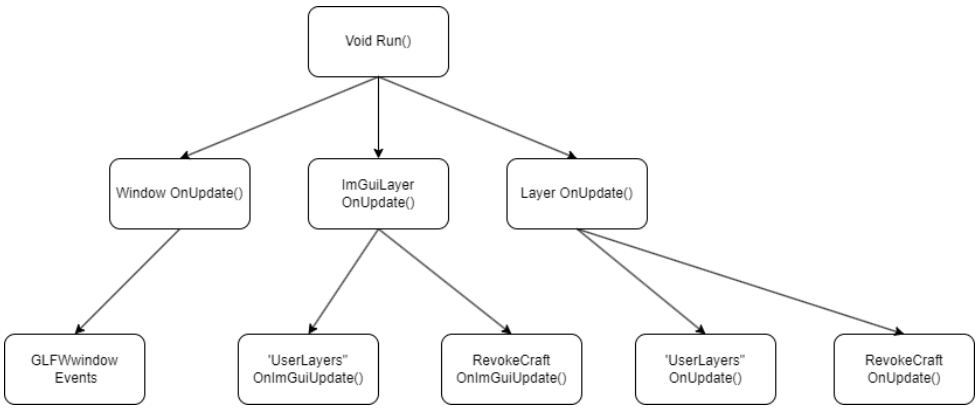


Рисунок 3.9 – Діаграма життєвого циклу додатку «MyRevoke»

Важливим етапом розробки є створення системи подій вікна GLFW. Важливо щоб дана система працювала швидко та точно. Для її реалізації було створено класи для визначення типів подій (рис. 3.10), та клас “EventDispatcher”, який викликає та повертає статус виконання функції при певній події (рис. 3.11). Самі події надходять з класу “Window” (рис. 3.12), який в свою чергу відповідає за ініціалізацію GLFW вікна, подій та інших необхідних функцій. Після ініціалізації вікна в класі “Application”, функція “OnEvent” встановлюється для зворотного зв’язку з подіями [24].

Реалізація класу GLFW вікна зображена на рисунку 3.13.

```
class  KeyPressedEvent : public KeyEvent
{
public:
    KeyPressedEvent(int keycode, int repeatCount)
        : KeyEvent(keycode), m_RepeatCount(repeatCount) {}
    inline int GetRepeatCount() const { return m_RepeatCount; }

    std::string ToString() const override
    {
        std::stringstream ss;
        ss << "KeyPresedEvent: " << m_KeyCode << " (" << m_RepeatCount << " repeats)";
        return ss.str();
    }

    static EventType GetStaticType() { return EventType::KeyPressed; } // used staic `cou
    virtual EventType GetEventType() const override { return GetStaticType(); }
    virtual const char* GetName() const override { return "KeyPressed"; }

private:
    int m_RepeatCount;
};
class  KeyTypedEvent : public KeyEvent
```

Рисунок 3.10 – Приклад одного з класів системи подій

```cpp
class EventDispatcher
{
public:
    EventDispatcher(Event& event)
        :m_Event(event) {}

    template<typename T>
    bool Dispatch(std::function <bool(T&)> func)
    {
        if (m_Event.GetEventType() == T::GetStaticType())
        {
            m_Event.Handled = func(*(T*)&m_Event);
            return true;
        }
        return false;
    }

private:
    Event& m_Event;
};
```

Рисунок 3.11 – Клас для виклику функцій при відповідних подіях

```cpp
glfwSetMouseButtonCallback(m_Window, [](GLFWwindow* window, int button, int action, int mods)
    {
        WindowData& data = *(WindowData*)glfwGetWindowUserPointer(window);

        switch (action)
        {
        case GLFW_PRESS:
        {
            MouseButtonPressedEvent event(button);
            data.EventCallback(event);
            break;
        }
        case GLFW_RELEASE:
        {
            MouseButtonReleasedEvent event(button);
            data.EventCallback(event);
            break;
        }
        }
    });
```

Рисунок 3.12 – Приклад встановлення зворотного виклику функцій “OnEvent” з вікна GLFW

```
class Window
{
public:
    Window(const WindowSettings& settings = WindowSettings());

    ~Window();

    void OnUpdate();

    unsigned int GetWidth() const { return m_Data.Width; }
    unsigned int GetHeight() const { return m_Data.Height; }

    inline void SetEventCallback(const std::function<void(Event&)>& callback) { m_Data.EventCallback = callback; }
    void SetVSync(bool enable);
    bool IsVSync() const;

    inline void* GetCoreWindow() const { return m_Window; }
private:
    void Init(const WindowSettings& settings);
    void Shutdown();
private:
    GLFWwindow* m_Window;
    RenderContex* m_Context;

    struct WindowData
    {
        std::string Title = "";
        unsigned int Width = 0, Height = 0;
        bool VSync = false;
        std::function<void(Event&)> EventCallback;
    };
    WindowData m_Data;

};
```

Рисунок 3.13 – Клас вікна GLFW

Головною частиною будь якого графічного рушія є рендерер. Важливо, щоб дана система працювала швидко та ефективно тому для оптимізації було вирішено реалізувати “батчинг” багатьох об’єктів у один виклик команди “glDrawElements” [25]. Для реалізації було створено структури для зберігання даних про вершини та текстури, а також клас “Renderer2D”, який відповідає за ініціалізацію та керування рендерингом 2D-об'єктів (рис. 3.14). У функції “DrawQuad” об’єкти додаються до поточного “батчу”. Коли поточний “бачт” переповнений, або коли викликається функція кінця поточного створення графіки, відбувається рендеринг. (рис. 3.15). Оскільки у двовимірному просторі не потрібно рендерити складні, багато вершинні об’єкти, усі інші геометричні фігури будуть відображені як текстури з прозорим фоном. Також, оскільки рушій розрахований на двовимірну графіку, потреби у створенні окремих шейдерів на кожен об’єкт немає, тому увесь рендеринг відбувається з одного головного шейдера (рис 3.16).

```
class Renderer2D
{
public:

    static void Init();
    static void Shutdown();

    static void Begin(const Camera& camera, const glm::mat4 transform);
    static void Begin(const EditorCamera& camera);
    static void End();



    static void DrawQuad(const glm::vec3& position, const glm::vec2 size, const glm::vec4& color, int entityID);
    static void DrawQuad(const glm::vec2& position, const glm::vec2 size, const glm::vec4& color, int entityID);
    static void DrawQuad(const glm::vec3& position, const glm::vec2 size, const Shared<Texture>& texture, int entityID);
    static void DrawQuad(const glm::vec2& position, const glm::vec2 size, const Shared<Texture>& texture, int entityID);

    static void DrawQuad(const glm::mat4& transform, const glm::vec4& color, int entityID);
    static void DrawQuad(const glm::mat4& transform, const Shared<Texture>& texture, int entityID);

    static void DrawSprite(const glm::mat4& transform, SpriteRendererComponent& sprite, int entityID);

    static void QuadInit();
private:
    static void NewBatch();

};
```

Рисунок 3.14 – Структура класу “Renderer2D”

```
if (mainCamera)
{
    Renderer2D::Begin(*mainCamera, cameraTransform);

    auto group = m_Registry.group<TransformComponent>(entt::get<SpriteRendererComponent>);
    for (auto entity : group)
    {
        auto [transform, sprite] = group.get<TransformComponent, SpriteRendererComponent>(entity);
        Renderer2D::DrawSprite(transform.GetTransform(), sprite, (int)entity);

    }
    Renderer2D::End();
}
```

Рисунок 3.15 – Приклад використання “Renderer2D”

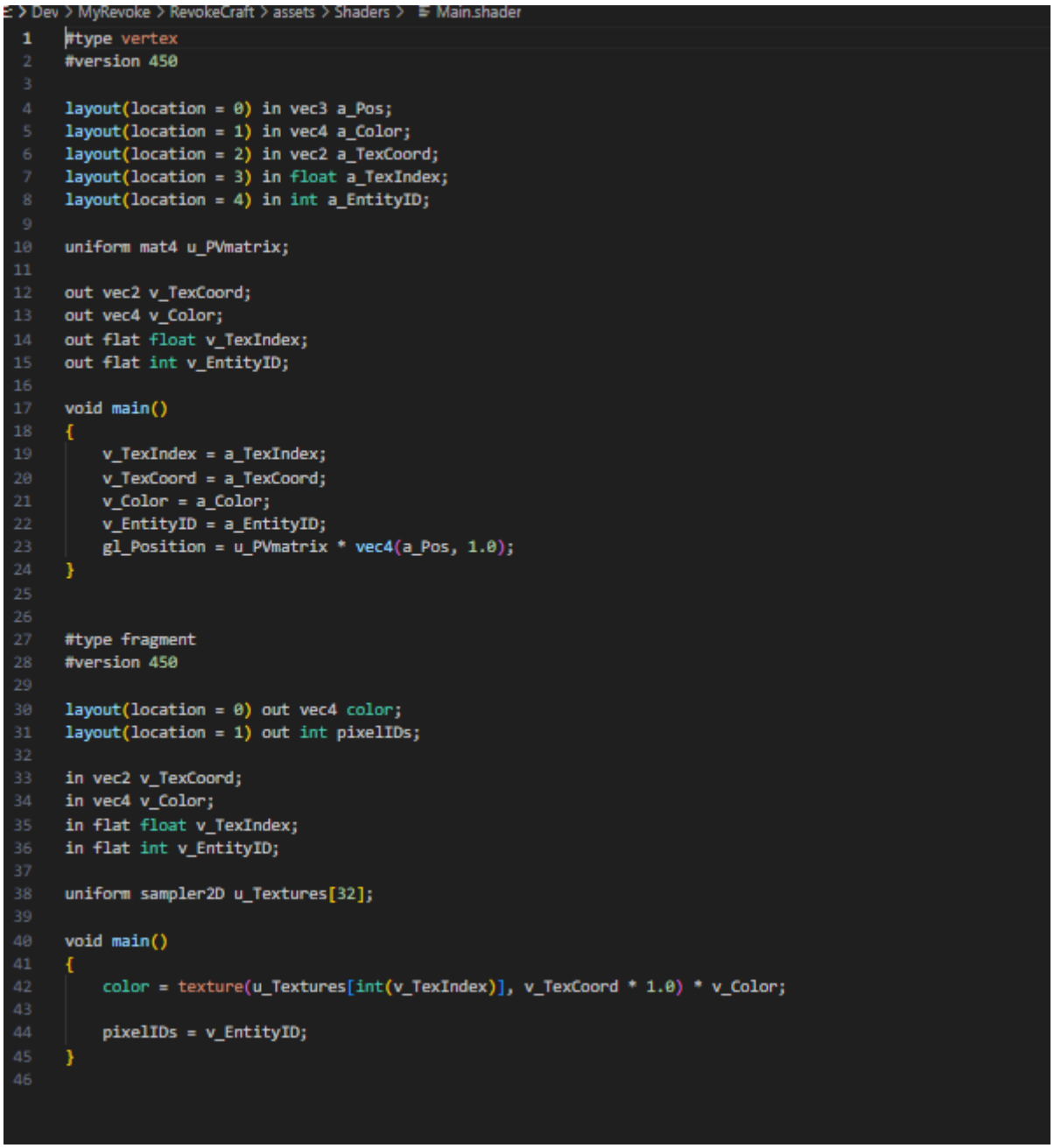


Рисунок 3.16 – Головний шейдер додатку «MyRevoke»

Дотримуючись даного стилю ООП та зв’язків у програмі було побудовані інші компоненти додатку, такі як бібліотека шейдерів, система камер, текстури, аудіо підтримка, скриптування.

Для розробки інтерфейсу використано бібліотеку DearImGUI. Було створено клас, який наслідує головний клас “Application” і реалізовано визначення функції для створення застосунку. Далі у цьому класі в конструкторі додано шар

“CraftLayer” що дозволить додатку викликати функції даного класу за допомогою поліморфізму (рис. 3.17).

```
namespace Revoke {

    class RevokeCraft : public Revoke::Application
    {
    public:
        RevokeCraft()
        {
            PushLayer(new CraftLayer());
        }
        ~RevokeCraft()
        {

        }

    private:

    };

    Revoke::Application* Revoke::CreateApplication()
    {
        return new RevokeCraft();
    }
};
```

Рисунок 3.17 – Головний клас інтерфейсу додатку

Інтерфейс розділений на декілька класів, які представляють собою різні частини інтерфейсу. Наприклад клас “ContentBrowser” відповідає за частину інтерфейсу у якій користувач можу переміщатися через папки усередині проекту, та перетягувати текстури, скрипти, звуки та сцени (рис 3.18).

Реалізація даного класу представлена на рисунку 3.19.

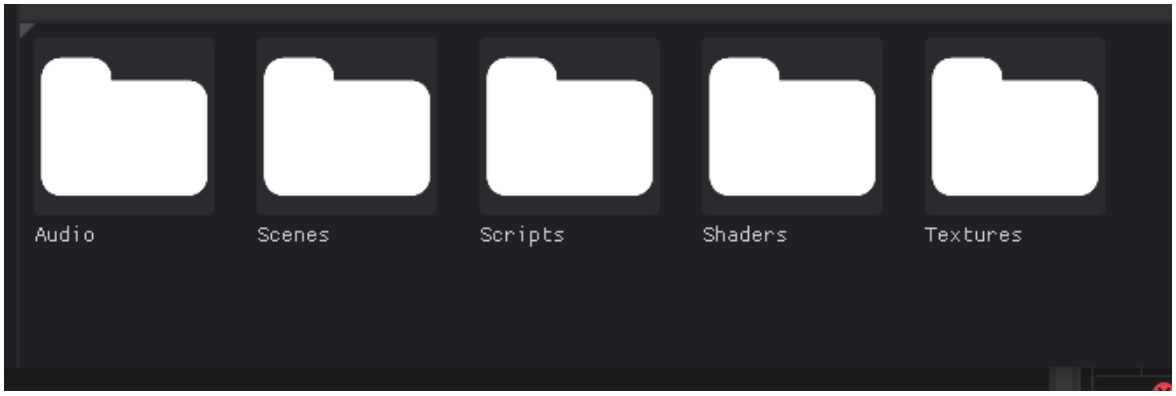
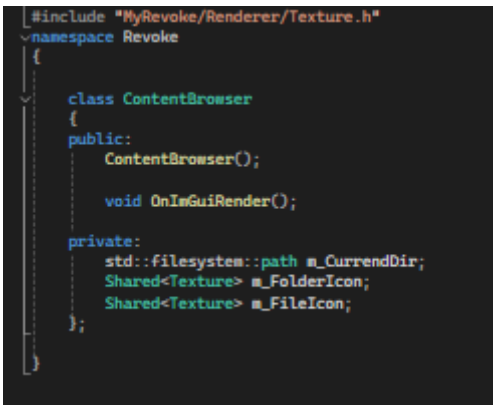


Рисунок 3.18– Приклад вигляду інтерфейсу “ContentBrowser”



```
#include "MyRevoke/Renderer/Texture.h"
namespace Revoke
{

    class ContentBrowser
    {
    public:
        ContentBrowser();

        void OnImGuiRender();

    private:
        std::filesystem::path m_CurrendDir;
        Shared<Texture> m_FolderIcon;
        Shared<Texture> m_FileIcon;
    };

}
```

Рисунок 3.19 – Приклад структури класу “ContentBrowser”

3.3 Тестування способів використання додатку

Оскільки, головною задачею графічного рушія є створення ігор, для успішного проведення тестування доцільно створити примітивну двовимірну гру.

Для цього відкриваємо додаток і створюємо нову сцену (рис. 3.20). Додаємо гравця та рівень, призначаємо їм відповідні текстури (рис 3.21). Для створення нових об’єктів потрібно натиснути праву кнопку миші у зоні “Scene Hierarchy” і обрати опцію “Create Entity”. Після цього даний об’єкт можна обрати і додати до нього компоненти текстур, фізики, звуків, камери, скиптів.

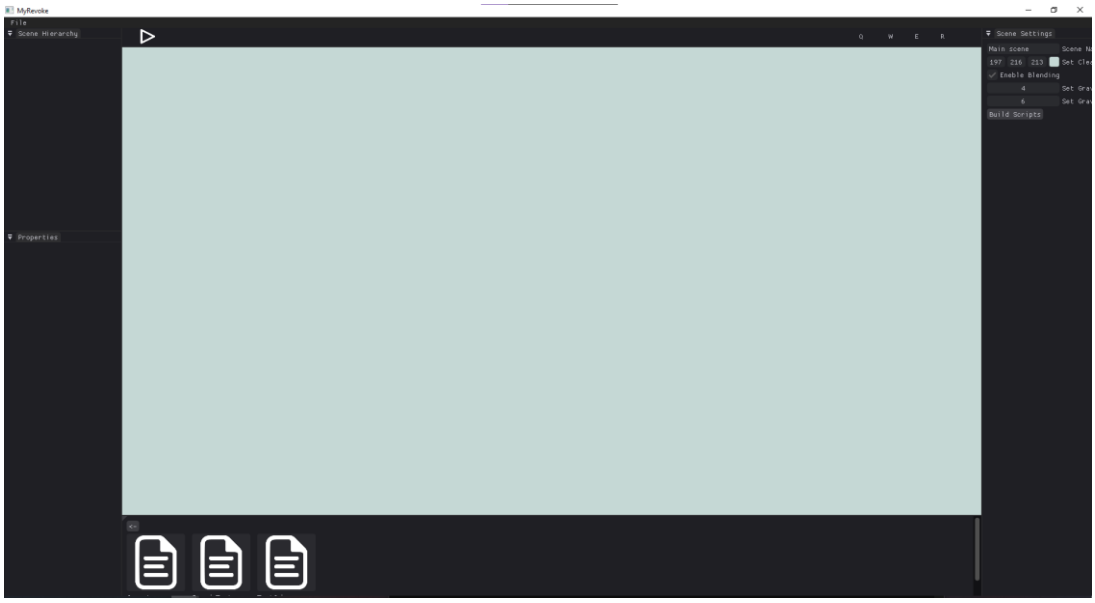


Рисунок 3.20 – Інтерфейс додатку «MyRevoke»

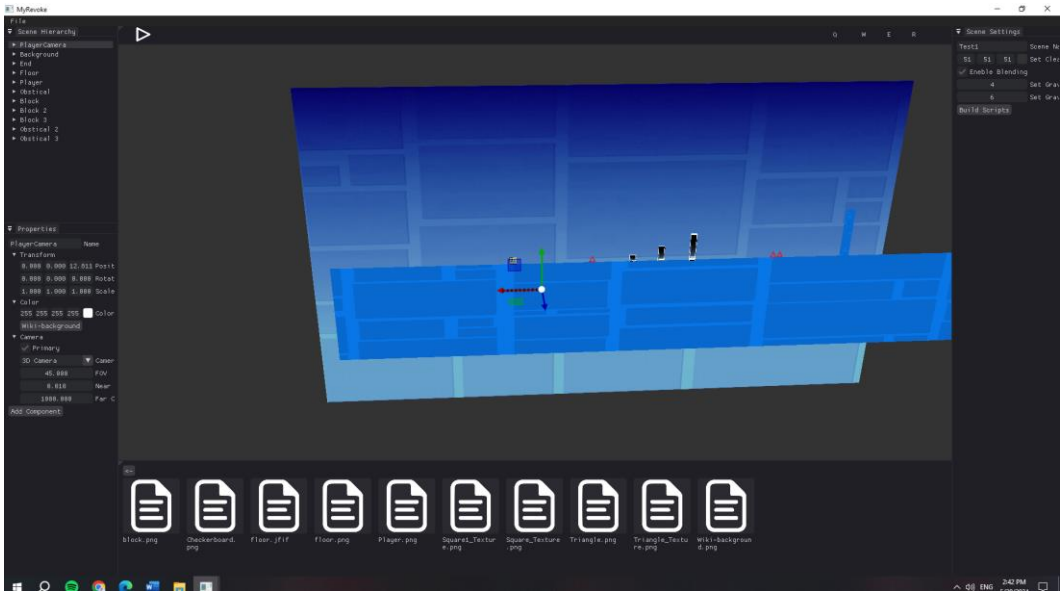
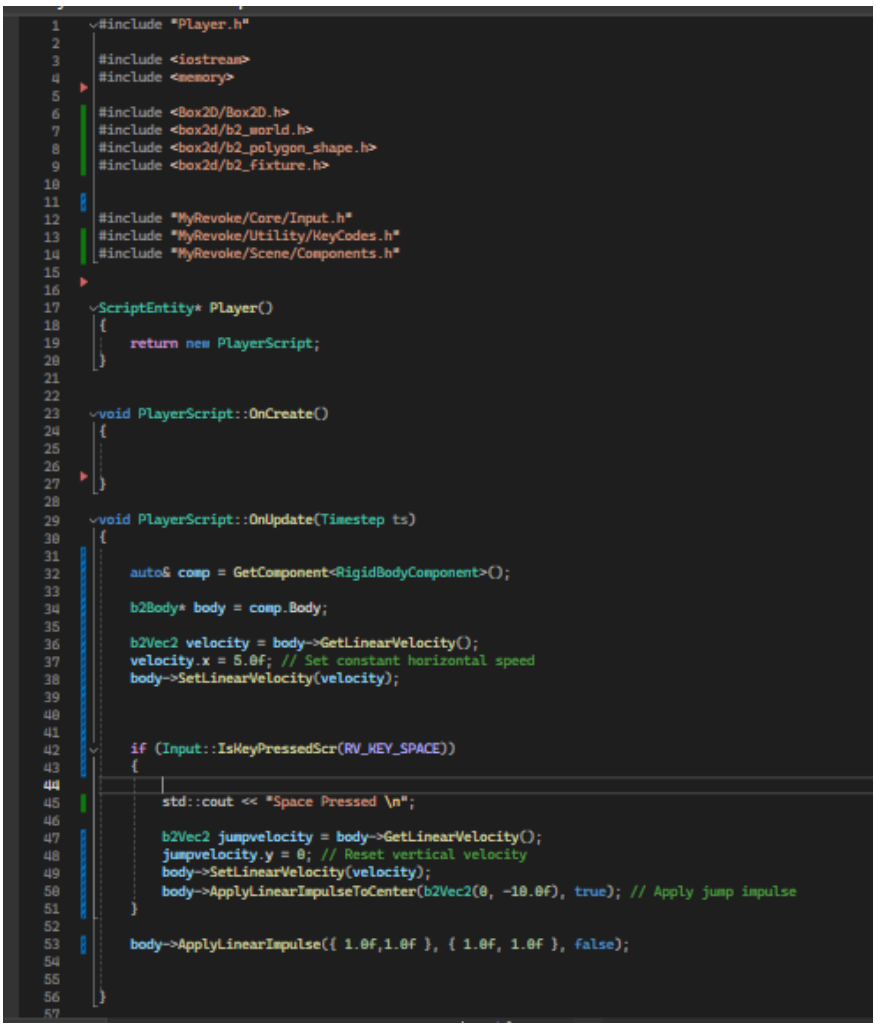


Рисунок 3.21 – Приклад створення сцени у додатку «MyRevoke»

Після створення тестової сцени, необхідно написати скрипт для руху гравця. Для цього необхідно створити і відкрити C++ клас по зразку “ScriptTamplate” (рис. 3.22). Створений скрипт необхідно перетягнути у компонент обраного об’єкта (рис. 3.23).



```cpp
#include "Player.h"

#include <iostream>
#include <memory>

#include <Box2D/Box2D.h>
#include <box2d/b2_world.h>
#include <box2d/b2_polygon_shape.h>
#include <box2d/b2_fixture.h>


#include "MyRevoke/Core/Input.h"
#include "MyRevoke/Utility/KeyCodes.h"
#include "MyRevoke/Scene/Components.h"

ScriptEntity* Player()
{
    return new PlayerScript;
}


void PlayerScript::OnCreate()
{


}

void PlayerScript::OnUpdate(Timestep ts)
{

    auto& comp = GetComponent<RigidBodyComponent>();

    b2Body* body = comp.Body;

    b2Vec2 velocity = body->GetLinearVelocity();
    velocity.x = 5.0f; // Set constant horizontal speed
    body->SetLinearVelocity(velocity);



    if (Input::IsKeyPressedScr(RV_KEY_SPACE))
    {
        std::cout << "Space Pressed \n";

        b2Vec2 jumpvelocity = body->GetLinearVelocity();
        jumpvelocity.y = 0; // Reset vertical velocity
        body->SetLinearVelocity(velocity);
        body->ApplyLinearImpulseToCenter(b2Vec2(0, -10.0f), true); // Apply jump impulse
    }

    body->ApplyLinearImpulse({ 1.0f,1.0f }, { 1.0f, 1.0f }, false);


}
```

Рисунок 3.22 – Приклад створення скрипта у додатку «MyRevoke»

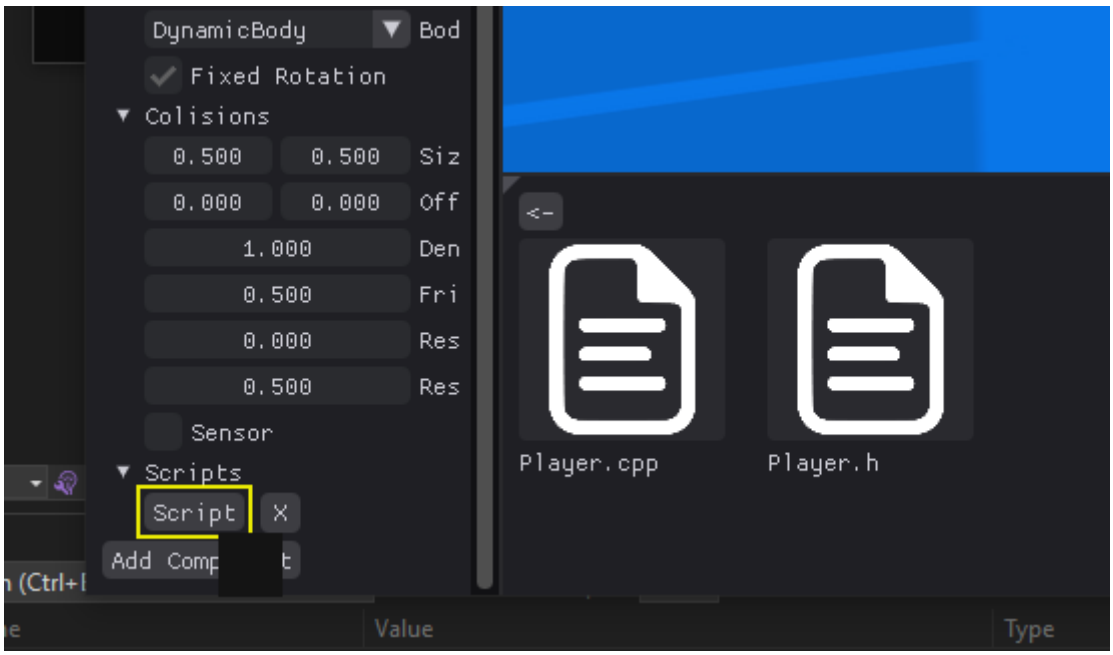


Рисунок 3.23 – Приклад додавання скрипта у додатку «MyRevoke»

Наступним кроком стане додавання звуку при стрибку. Для цього необхідно додати компонент звуку (рис. 3.24) до об’єкта “Player”. Наступним кроком потрібно перетягнути обраний звук у компонент та у скрипті додати до події стрибка відтворення звуку. Також необхідно додати логіку для камери.

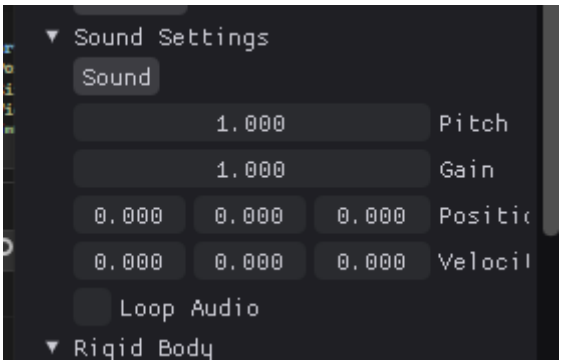


Рисунок 3.24 – Приклад додавання компоненту звуку в додатку «MyRevoke»

В результаті, за декілька хвилин було створено примітивну двовимірну сцену-гру.

Приклад роботи гри можна переглянути на рисунку 3.25.

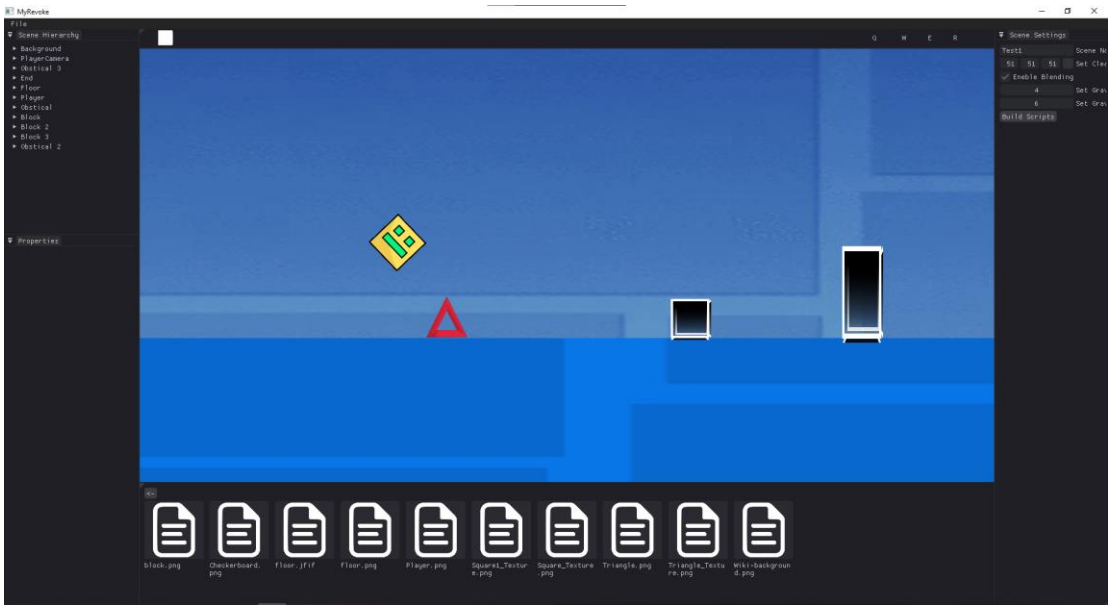


Рисунок 3.25 – Приклад відтворення сцени у додатку «MyRevoke»

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра було проведено аналіз предметної області, дослідження актуальності розробки за тематикою ігрових локацій та аналогічні проекти. В результаті підтверджена актуальність розробки, сформульовані вимоги до моделі.

Проаналізовано найбільш поширені графічні рушії для розробки ігор, локацій, та за результатами порівняння обрано напрям розробки та цілі по реалізації проекту «MyRevoke». Також сформовано постановку задачі та технічне завдання на виконання проекту.

Виконано планування робіт та проаналізовано можливі ризики при виконання проекту. Це допомогло запобігти проблемам, які могли б виникнути при програмній реалізації додатку.

Було проведено моделювання принципу роботи програмного додатку для створення графічних сцен у двовимірному просторі за допомогою моделі в нотації IDEF0. За допомогою нотації UML виконане графічне представлення функціонування програмного додатку - створена діаграма варіантів використання. Розроблено діаграму класів, яка відображає головні взаємозв’язки елементів програмної реалізації. Розроблено діаграму, яка відображає архітектуру програмного додатку.

Згідно технічного завдання та розроблених моделей виконано програмну реалізацію додатку. Додаток створений для побудови двовимірних ігор та графічних сцен. Продемонстровані головні моменти програмної реалізації додатку. Проведено тестування функціональних можливостей додатку, створюючи примітивну двовимірну гру.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. J. Ward, What is a Game Engine?, URL: http://www.gamecareerguide.com/features/529/what_is_a_game.php. (дата звернення: 20.03.2024).

2. D. H. Eberly, 3D Game Engine Architecture: книга 3-тє вид. 1200с.

3. A. Barczak , H. Woźniak, Comparative Study on Game Engines,. Studia Informatica, no. 23, 2020, doi: 10.34739/si.2019.23.01.

4. Jason Gregory (Author) , Richard Lemarchand (Foreword), Game Engine Architecture. книга 2-ге вид.. 2014. 900.

5. A. Andrade, Game engines: a survey, EAI Endorsed Transactions on Game-Based Learning, частина 2, ноиер 6, 2015, doi: 10.4108/eai.5-11-2015.150615.

6. Elements of a game engine | Outsource Accelerator. URL: https://www.outsourceaccelerator.com/articles/game-engine/ (дата звернення: 26.04.2024).

7. Did you know that 60% of game developers use game engines? Accessed: Apr. 29, 2024. . URL: https://www.slashdata.co/post/did-you-know-that-60-of-game-developers-use-game-engines (дата звернення: 27.04.2024).

8. Top 15 Reasons to Use Unity 3D for Game Development | LinkedIn. URL: https://www.linkedin.com/pulse/top-15-reasons-use-unity-3d-game-development-jklucent-cvhjc/ (дата звернення: 28.04.2024).

9. The Pros And Cons Of Mobile Game Development With Unity 3D - VARTEQ Inc. URL: https://varteq.com/the-pros-and-cons-of-mobile-game-development-with-unity-3d/ (дата звернення: 28.04.2024).

10. Unity plan pricing and packaging updates. Unity Technologies. URL: https://blog.unity.com/news/plan-pricing-and-packaging-updateshttps://blog.unity.com/news/plan-pricing-and-packaging-updates (дата звернення: 28.04.2024).

11. Is Unreal Engine Good for 2D Games? A Detailed Review with Examples. DANG PHUC. URL: https://animost.com/ideas-inspirations/is-unreal-engine-good-for-2d-games/ (дата звернення: 29.04.2024).

12. T. K. Mohd, F. Bravo-Garcia, L. Love, M. Gujadhur, and J. Nyadu. Analyzing Strengths and Weaknesses of Modern Game Engines. International Journal of Computer Theory and Engineering, частина 15, ноиер 1, 2023, doi: 10.7763/IJCTE.2023.V15.1330.

13. What is the Frostbite Engine? URL: https://www.techcareer.net/en/blog/frostbite-motoru-nedir (дата звернення: 29.04.2024).

14. Frostbite: Powering world-class gaming experiences and opportunities.” URL: https://www.ea.com/ea-studios/ea-romania/news/frostbite-powering-world-class-gaming-experiences-and-opportunities (дата звернення: 30.04.2024).

15. Bryan Wirtz, The Frostbite Game Engine: Overview & Analysis. URL: https://www.gamedesigning.org/engines/frostbite/ (дата звернення: 30.04.2024).

16. OpenGL Wiki. URL: https://www.khronos.org/opengl/wiki/ (дата звернення: 30.04.2024).

17. Dear ImGui: Bloat-free Graphical User interface for C++ with minimal dependencies. URL: https://github.com/ocornut/imgui (дата звернення: 30.04.2024).

18. OpenGL Mathematics (GLM). URL: https://github.com/g-truc/glm (дата звернення: 30.04.2024).

19. Gaming meets modern C++ - a fast and reliable entity component system (ECS) and much more. URL: https://github.com/skypjack/entt (дата звернення: 30.04.2024).

20. Q. Li, Y.-L. Chen, IDEF0 Function Modeling. Modeling and Analysis of Enterprise and Information Systems. 2009. doi: 10.1007/978-3-540-89556-5_5.

21. M. J. Chonoles. What is UML?. OCUP Certification Guide: UML 2.5 Foundational Exam. 2018. doi: 10.1016/b978-0-12-809640-6.00003-9.

22. H. Osman, A. Van Zadelhoff, D. R. Stikkolorum, M. R. V. Chaudron, UML class diagram simplification: What is in the developer’s mind?, in Proceedings of the 2nd International Workshop on Experiences and Empirical Studies in Software Modelling, EESSMod 2012, 2012. doi: 10.1145/2424563.2424570.

23. Yan Chernikov, “Entry Point For a Game Engine,” Entry Point | Game Engine Series. URL: https://www.youtube.com/watch?v=meARMOmTLgE&t=3s (дата звернення: 03.05.2024).

24. Yan Chernikov, “Event System in a Game Engine,” Event System | Game Engine series. URL: https://www.youtube.com/watch?v=5mlziHwq90k (дата звернення: 03.05.2024).

25. Saurav Sikarwar, “Batching in OpenGL,” Batching in OpenGL. URL: https://www.youtube.com/watch?v=SUMUtevioe4&t=62s (дата звернення: 10.05.2024).

26. Setting Goals and Developing Specific, Measurable, Achievable, Relevant, and Time-bound Objectives. Samhsa Native Connections2018. С. 1-4.

27. Christine Organ, Cassie Bottorff, Work Breakdown Structure (WBS) In Project Management – Forbes Advisor. URL: https://www.forbes.com/advisor/business/what-is-work-breakdown-structure/ (дата звернення: 10.02.2024).

28. Shweta and Cassie Bottorff, What Is A Gantt Chart? The Ultimate Guide – Forbes Advisor. URL: https://www.forbes.com/advisor/business/software/what-is-a-gantt-chart/ (дата звернення: 12.02.2024).

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ
на розробку інформаційної системи
«Програмний додаток – 2D графічний рушій»

ПОГОДЖЕНО:

Доцент кафедри інформаційних технологій

_____ Ващенко С.М.

Студент групи ІТ-01

_____ Сухенич М.М.

Суми 2024

1 Призначення й мета створення додатку

1.1 Призначення додатку

Додаток призначений для створення ігор та графічної візуалізації сюжетів у двовимірному просторі.

1.2 Мета створення додатку

Головною метою даного дослідження є розробка додатку, який буде відрізнятися простим для оволодіння інтерфейсом та вільним доступом, в додачу до цього забезпечувати абсолютний контроль над редагуванням внутрішнього коду.

1.3 Цільова аудиторія

До цільової аудиторії даного додатку відносяться користувачі, які зацікавлені створенням графічних програм, дизайном, сюжетними іграми та інтерактивними медіа.

2 Вимоги до додатку

2.1 Вимоги до додатку в цілому

Додаток повинен забезпечувати встановлений ряд функціональних можливостей та бути у вільному доступі.

2.2 Вимоги до структури й функціонування додатку

Додаток «MyRevoke» має бути реалізований за допомогою мови програмування C++ й додаткових бібліотек та надавати доступ до набору визначених функціональних можливостей.

Фінальний продукт повинен бути представлений комп’ютерним додатком, який забезпечує обширний функціонал та зручний інтерфейс.

Функціонал рушія «MyRevoke»:

- можливість створення та редагування сцен використовуючи лише інтерфейс додатку;
- можливість інтегрування власних текстур та аудіо файлів;
- надання необмеженого контролю над структурою рендерера та модулів зв’язаних з створенням графіки;
- висока оптимізація при рендеренгу великої кількості об’єктів;
- низький рівень вимог до характеристик пристрою користувача;

2.3 Вимоги до програмного та апаратного забезпечення

Оптимальні технічні характеристики пристрою для надійного функціонування програми включають графічний адаптер Nvidia GeForce 760 або новіший, центральний процесор Intel Core i3 3.0 GHz або швидший, 4 ГБ оперативної пам’яті та 8 ГБ вільного дискового простору.

2.4 Вимоги до збереження інформації

Усі файли як додатку «MyRevoke», так і створених сцен, текстур чи аудіо, повинні зберігатися локально на пристрої користувача.

3 Структура додатку

3.1 Загальна інформація про структуру додатку «MyRevoke»

До структури додатку входять усі внутрішні модулі, які є лімітовано доступні. Модулі для розробки, тестування та основний модуль є загальнодоступними, в той час як вторинні модулі, які не потребують редагування, недоступні.

Перелік модулів у додатку «MyRevoke»:

- Основний модуль «MyRevoke», який містить ядро програми. Даний модуль забезпечує роботу додатку і виконує такі завдання, як рендеринг, система подій, система шарів, абстракція додаткових модулів.
- Модуль для тестування - «Sandbox». Змодельований для тестування функціоналу застосунку без користувацького інтерфейсу та додаткових обмежень модуля редактора.
- «RevokeCraft» - модуль у якому реалізовано користувацький інтерфейс для створення сцен у двовимірному просторі.

3.2 Дизайн додатку «MyRevoke»

Дизайн додатку повинен бути реалізований у мінімалістичному та сучасному стилі. Користувацький інтерфейс повинен бути розроблений з використанням бібліотеки «imgui». Палітра кольорів – у сучасному стилі. Шрифти повинні бути приємними для читання. Інтерфейс повинен бути гнучким і зрозумілим з можливістю внесення змін користувачем. Шаблон користувацького інтерфейсу майбутнього додатку «MyRevoke» зображено на рисунку А.1.

Рисунок А.1 – Схема головного інтерфейсу

Карта додатку зображена на рисунку А.2.

Рисунок А.2 – Система навігації

4. Склад і зміст робіт зі створення додатку «MyRevoke»

Докладний опис етапів роботи зі створення графічного рушія наведено в таблиці А.2.

Таблиця А.2 – Етапи створення двовимірного графічного рушія

№	Склад і зміст робіт	Строк розробки
1	Аналіз предметної області	5 днів
2	Визначення властивостей додатку	5 днів
3	Розробка шаблону додатку	5 днів
4	Підготовка структури застосунка	14 днів
5	Розробка системи подій	7 днів
6	Розробка системи шарів	7 днів
7	Розробка базових шейдерів	7 днів
7	Розробка абстракції API бібліотеки	15 днів
8	Розробка базового 2D рендерера	25 днів
9	Розробка системи камер	14 днів
10	Розробка підтримки текстур	7 днів
11	Розробка підтримки аудіо	7 днів
12	Розробка підтримки пост-ефектів	7 днів
13	Розробка підтримки освітлення	10 днів
14	Розробка підтримки скриптування	7 днів
15	Розробка інтерфейсу застосунка	10 днів
16	Написання супровідної документації	1 день
17	Beta - тестування	7 днів
18	Alpha - тестування	5 днів
19	Реліз застосунку	1 день
	Загальна тривалість робіт	164 дні

майбутнього додатку «MyRevoke» зображено на рисунку А.1.

Рисунок А.1 – Схема головного інтерфейсу

Карта додатку зображена на рисунку А.2.

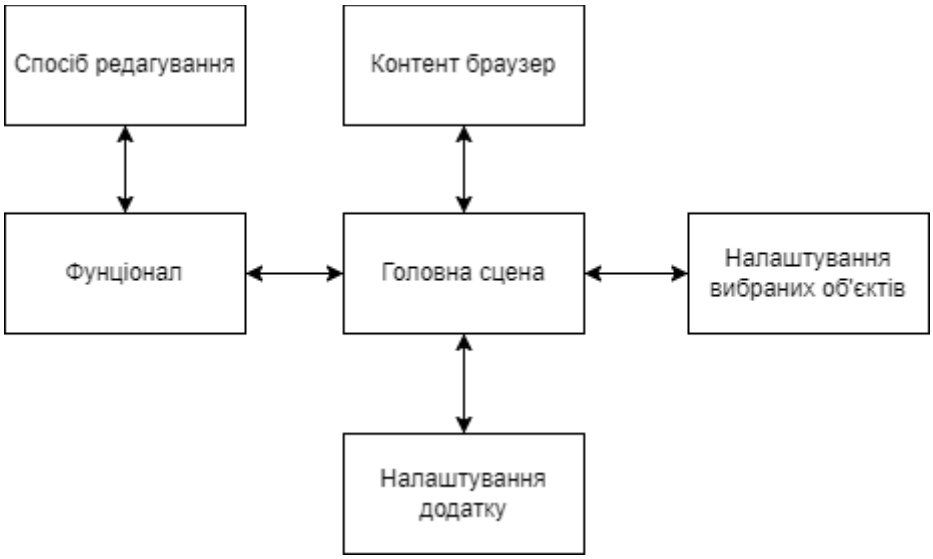


Рисунок А.2 – Система навігації

4. Склад і зміст робіт зі створення додатку «MyRevoke»

Докладний опис етапів роботи зі створення графічного рушія наведено в таблиці А.2.

Таблиця А.2 – Етапи створення двовимірного графічного рушія

№	Склад і зміст робіт	Строк розробки
1	Аналіз предметної області	5 днів
2	Визначення властивостей додатку	5 днів
3	Розробка шаблону додатку	5 днів
4	Підготовка структури застосунка	14 днів
5	Розробка системи подій	7 днів
6	Розробка системи шарів	7 днів
7	Розробка базових шейдерів	7 днів
7	Розробка абстракції API бібліотеки	15 днів
8	Розробка базового 2D рендерера	25 днів
9	Розробка системи камер	14 днів
10	Розробка підтримки текстур	7 днів
11	Розробка підтримки аудіо	7 днів
12	Розробка підтримки пост-ефектів	7 днів
13	Розробка підтримки освітлення	10 днів
14	Розробка підтримки скриптування	7 днів
15	Розробка інтерфейсу застосунка	10 днів
16	Написання супровідної документації	1 день
17	Beta - тестування	7 днів
18	Alpha - тестування	5 днів
19	Реліз застосунку	1 день
	Загальна тривалість робіт	164 дні

ДОДАТОК Б
ПЛАНУВАННЯ РОБІТ

Головною метою даного проекту є розробка додатку, який буде відрізнятися простим для оволодіння інтерфейсом та вільним доступом, в додачу до цього забезпечувати абсолютний контроль над внутрішнім кодом та оптимізацією.

Б.1 Деталізація мети проекту методом SMART

Для актуальності та конкурентоспроможності додатку потрібно на концептуальному етапі правильно визначити його мету за допомогою SMART-методу [26] “Розробка графічного двовимірного рушія «MyRevoke» на основі затвердженого технічного завдання, розробки ігор і візуалізації сценаріїв у двовимірному просторі, до 01 червня 2024 року”.

Отже, Результати деталізації мети нашого проекту за цими п’ятьма факторами наведені у таблиці Б.1.

Таблиця Б.1 – Формалізація мети за технологією SMART

Specific	Створення графічного рушія для двовимірних додатків, який буде відзначатися легким для освоєння інтерфейсом та безплатним доступом, а також забезпечувати повний контроль над кодом та оптимізацією.
Measurable	Створений зручний та простий спосіб розробки сцен та рівнів у двовимірному просторі.
Achievable	Мета досяжна. Затверджене технічне завдання від замовника та реалізований прототип рушія «MyRevoke».
Revelant	Скорочення часу на створювання ігри чи стени у двовимірному просторі і простий інтерфейс для людей без глибоких знань програмування.

Б.2 Планування змісту робіт та структури виконавців

WBS (Work Breakdown Structure – ієрархічна структура робіт) – це елементи проекту, графічно представлені однією згрупованою ієрархією у єдиній системі з продуктом проекту. Структура декомпозиції робіт спрямована на виконання частин проекту по невеликим частинам і сама по собі є невід'ємною частиною, ціль якої є організація роботи у команді [27].

На першому (найвищому) рівні знаходиться продукт проекту. Дії та заходи для реалізації мети проекту знаходяться на другому рівні декомпозиції. Декомпозиція робіт проводиться до моменту, коли вони перетворюються на елементарні (прості).

Елементарні роботи – це дії, що призводять до однозначного результату за виконання яких відповідає одна людина. Таким чином легко обчислити витрати праці та довжину виконання цих дій. На рисунку Б.1 представлено WBS з розробки графічного двовимірного рушія «MyRevoke».

Після завершення етапу декомпозиції процесів, необхідно розпочати розробку організаційної структури виконавців або OBS. Це графічна структура, яка відображає робітників або відповідальних осіб за відповідні елементарні роботи.

Організаційна структура планування проекту відображена на рисунку Б.2. У таблиці Б.2 викладено список виконавців проекту.

Таблиця Б.2. - Список виконавців, що функціонують в проекті.

Роль	ПІД	Проектна роль
Розробник	Сухенич М.М.	Виконує «front-end» та «back-end» розробку
Проектувальник	Сухенич М.М.	Виконує проектування структури застосунку.

Продовження таблиці Б.2.

Роль	**ПІД**	**Проектна роль**
Тестувальник	Сухенич М.М.	Відповідає за тестування функціоналу та дизайну застосунку.
Керівник проекту	Ващенко С.М.	Формує завдання на розробку проекту.
Менеджер проекту	Сухенич М.М.	Відповідає за виконання термінів, розподіл ресурсів та завдань між учасниками. Виконує збір та аналіз даних.

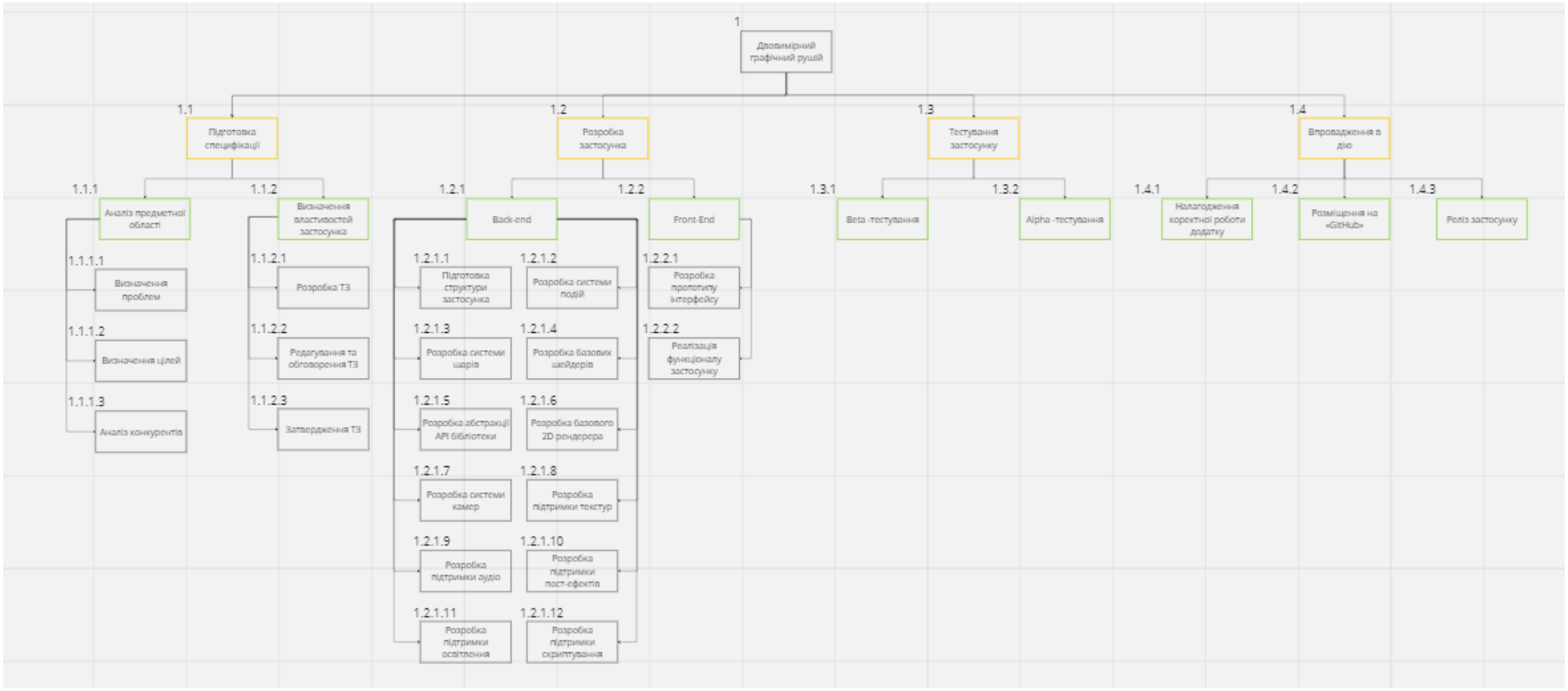


Рисунок Б.1 – WBS-структура робіт проекту

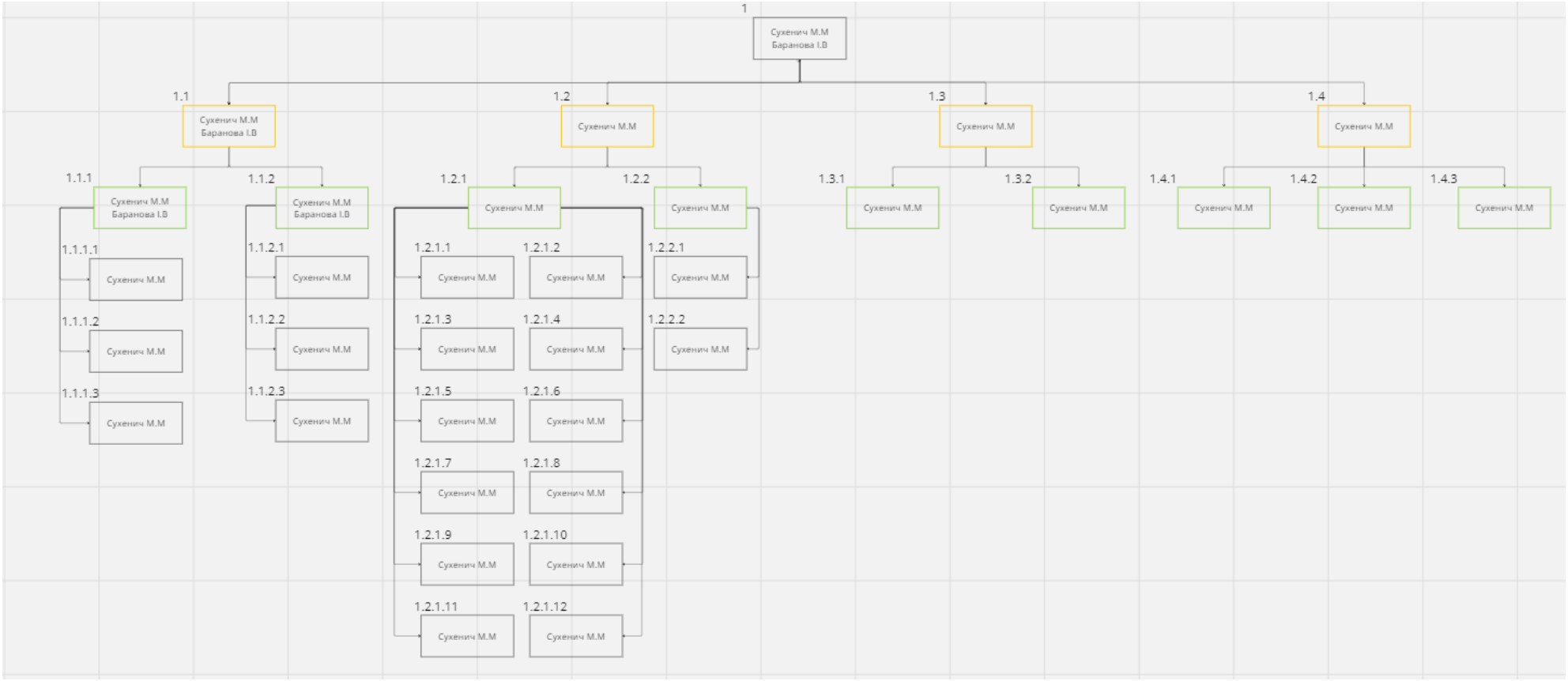


Рисунок Б.2 – OBS-структура робіт проекту

Б.3 Діаграма Ганта

Не менш важливим етапом планування є побудова календарного графіку (діаграми Ганта). Діаграма Ганта це розклад виконання робіт проекту з розподілом дат. За допомогою графіку створеного у діаграмі Ганта можна визначити тривалість процесів з обмеженнями в ресурсах, при цьому враховуючи вихідні та святкові дні [28]. Календарний графік проекту представлено на рисунках Б.3-Б.4.

		1	**Двовимірний графічний рушій**	**164 days**	**Wed 12/20/23**	**Thu 8/15/24**		
		1.1	**Підготовка специфікації**	**11 days**	**Wed 12/20/23**	**Mon 1/8/24**		
		1.1.1	**Аналіз предметної області**	**5 days**	**Wed 12/20/23**	**Wed 12/27/23**		
		1.1.1.1	Визначення проблем	1 day	**Wed 12/20/23**	Thu 12/21/23		Сухенич М.М
		1.1.1.2	Визначення цілей	2 days	Thu 12/21/23	Mon 12/25/23	4	Сухенич М.М
		1.1.1.3	Аналіз конкурентів	2 days	Mon 12/25/23	Wed 12/27/23	5	Сухенич М.М
		1.1.2	**Визначення властивостей застосунка**	**6 days**	**Wed 12/27/23**	**Mon 1/8/24**	**3**	
		1.1.2.1	Розробка ТЗ	3 days	Wed 12/27/23	Wed 1/3/24	6	Сухенич М.М
		1.1.2.2	Редагування та обговорення ТЗ	2 days	Wed 1/3/24	Fri 1/5/24	8	Сухенич М.М
		1.1.2.3	Затвердження ТЗ	1 day	Fri 1/5/24	Mon 1/8/24	9	Сухенич М.М
		1.2	**Розробка застосунка**	**137 days**	Mon 1/8/24	**Wed 7/24/24**	**2**	
		1.2.1	**Back-end**	**127 days**	**Mon 1/8/24**	**Wed 7/10/24**	**7**	
		1.2.1.1	Підготовка структури застосунка	14 days	Mon 1/8/24	Fri 2/2/24	10	Сухенич М.М
		1.2.1.2	Розробка системи подій	7 days	Fri 2/2/24	Tue 2/13/24	13	Сухенич М.М
		1.2.1.3	Розробка системи шарів	7 days	Tue 2/13/24	Thu 2/22/24	14	Сухенич М.М
		1.2.1.4	Розробка базових шейдерів	7 days	Thu 2/22/24	Mon 3/4/24	15	Сухенич М.М
		1.2.1.5	Розробка абстракції API бібліотеки	15 days	Mon 3/4/24	Mon 3/25/24	16	Сухенич М.М
		1.2.1.6	Розробка базового 2D рендерера	25 days	Mon 3/25/24	Mon 4/29/24	17	Сухенич М.М
		1.2.1.7	Розробка системи камер	14 days	Mon 4/29/24	Fri 5/17/24	18	Сухенич М.М
		1.2.1.8	Розробка підтримки текстур	7 days	Fri 5/17/24	Tue 5/28/24	19	Сухенич М.М
		1.2.1.9	Розробка підтримки аудіо	7 days	Tue 5/28/24	Thu 6/6/24	20	Сухенич М.М
		1.2.1.10	Розробка підтримки пост-ефектів	7 days	Thu 6/6/24	Mon 6/17/24	21	Сухенич М.М
		1.2.1.11	Розробка підтримки освітлення	10 days	Mon 6/17/24	Mon 7/1/24	22	Сухенич М.М
		1.2.1.12	Розробка підтримки скриптування	7 days	Mon 7/1/24	Wed 7/10/24	23	Сухенич М.М
		1.2.2	**Front-End**	**10 days**	Wed 7/10/24	**Wed 7/24/24**	**12**	
		1.2.2.1	Розробка прототипу інтерфейсу	2 days	Wed 7/10/24	Fri 7/12/24	24	Сухенич М.М
		1.2.2.2	Реалізація функціоналу застосунку	8 days	Fri 7/12/24	Wed 7/24/24	26	Сухенич М.М
		1.3	**Тестування застосунку**	**12 days**	Wed 7/24/24	**Fri 8/9/24**	**11**	
		1.3.1	Beta -тестування	5 days	Wed 7/24/24	Wed 7/31/24	27	Сухенич М.М
		1.3.2	Alpha -тестування	7 days	Wed 7/31/24	Fri 8/9/24	29	Сухенич М.М
		1.4	**Впровадження в дію**	**4 days**	Fri 8/9/24	**Thu 8/15/24**	**28**	
		1.4.1	Налагодження коректної роботи додатку	2 days	Fri 8/9/24	Tue 8/13/24	30	Сухенич М.М
		1.4.2	Розміщення на «GitHub»	1 day	Tue 8/13/24	Wed 8/14/24	32	Сухенич М.М
		1.4.3	Реліз застосунку	1 day	Wed 8/14/24	Thu 8/15/24	33	Сухенич М.М

Рисунок Б.3 – Календарний графік проекту

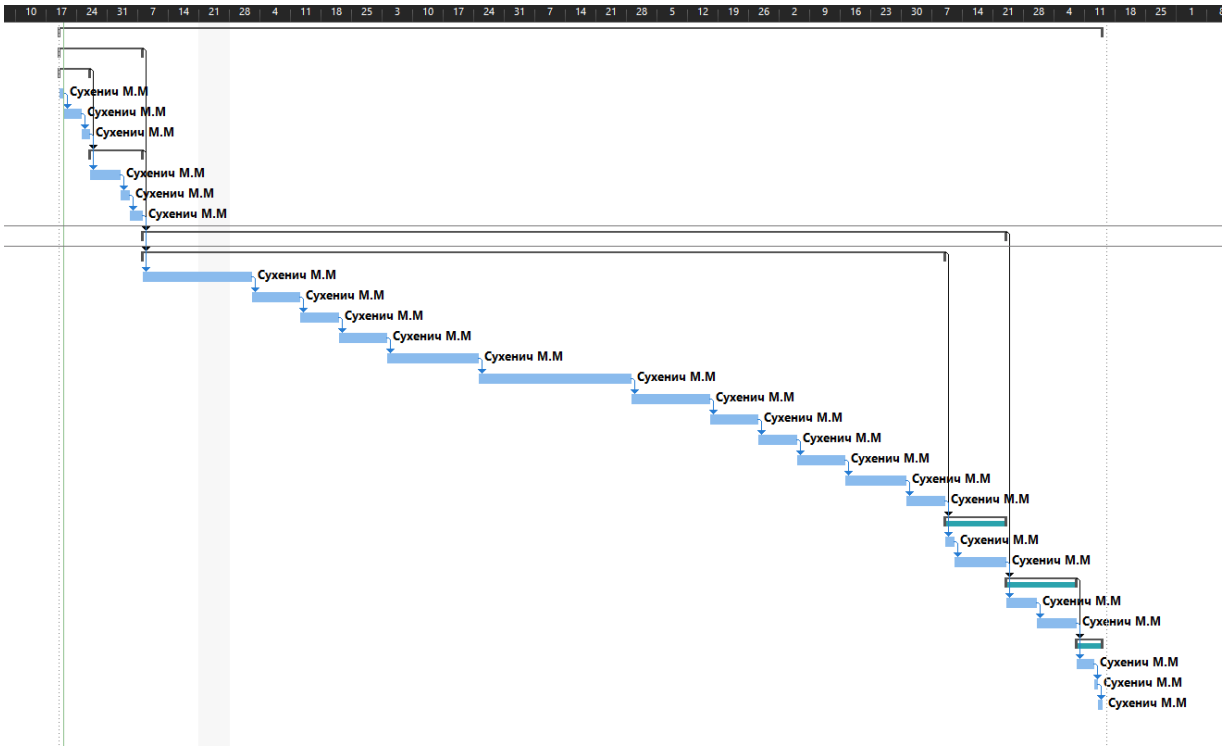


Рисунок Б.4 – Продовження календарного графіку проекту

Б.4 Управління ризиками проекту

Для проведення якісної оцінки ризиків, перш за все потрібно визначити ризики, які слід усунути якнайшвидше. Зважаючи на ступень важливості ризику – реагування буде відповідне. Наступним кроком є реалізація кількісного оцінювання ризиків. В залежності від забезпечення проекту, кількісне оцінювання та якісне оцінювання можуть виконуватися як разом так і окремо. Перелік ризиків даного проекту можна переглянути у таблиці Б. У таблиці Б.4 наведено результати оцінки ризиків. Таблиця Б.5 вміщує критерії для класифікації ризиків по величині впливу на проект та можливості їх появи.

Таблиця Б.3 – Ризики проекту

№ ризику	**Назва (опис) ризику**
R1	Проблема зі здоров'ям розробника
R2	Відключення електроживлення
R3	Нестабільне Інтернет з’єднання
R4	Проблеми з апаратним забезпеченням
R5	Повітряні тривоги
R6	Недоліки у тестуванні
R7	Неочікувані зміни в технологіях
R8	Зміна потреб користувача
R9	Недостатня документація
R10	Невідповідність вимогам обладнання

Таблиця Б.4 – Результати визначення ймовірності, впливу та рангу ризиків проекту

№ ризику	**Назва (опис) ризику**	**Ймовірність (0,1-0,9)**	**Вплив (0,05-0,8)**	**Ранг**
R1	Проблема зі здоров'ям розробника	0.4	0.3	0.12
R2	Відключення електроживлення	0.8	0.1	0.08
R3	Нестабільне Інтернет з’єднання	0.2	0.2	0.04
R4	Проблеми з апаратним забезпеченням	0.1	0.8	0.08
R5	Повітряні тривоги	0.3	0.3	0.09
R6	Недоліки у тестуванні	0.4	0.2	0.08
R7	Неочікувані зміни в технологіях	0.1	0.2	0.02
R8	Зміна потреб користувача	0.5	0.1	0.05
R9	Недостатня документація	0.3	0.2	0.06
R10	Невідповідність вимогам обладнання	0.3	0.1	0.03

Таблиця Б.5 – Шкала оцінювання ризиків за ймовірністю виникнення та величиною впливу

Оцінка	**Ймовірність виникнення**	**Вплив ризику**	**Тип ризику**
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

Для зниження негативного впливу ризиків на проект необхідно розробити планування реагування. До реагувань можна віднести оцінку наслідків впливу на проект та розроблення належних заходів. Аналіз здійснюється за показниками, наведеними у таблиці Б.4.Після проведення планування заходів реагування на ризики, зіставлено матрицю ймовірності появи та впливу ризиків (таблиця Б.6). Зеленим кольором на матриці відображені прийнятні ризики, жовтим – виправдані, а червоним – недопустимі.

Таблиця Б.5. – Матриця ймовірності та впливу

Ймовірність ризику	**Вплив загрози (ризику)**				
	Дуже малий	**Малий**	**Середній**	**Великий**	**Дуже великий**
	0,1	0,2	0,3	0,4	0,8
0,8	0.08(R2, R4)	0.16	0.24	0.32	0.64 (R2)
0,5	0.05 (R8)	0.1	0.15	0.2	0.4
0,4	0.04	0.08(R6)	0.12 (R1)	0.16 (R6)	0.32
0,3	0.03 (R10)	0.06 (R9)	0.09 (R5)	0.12	0.24
0,2	0,01	0.04(R3)	0,06	0,08	0,16
0,1	0.01	0.02 (R7)	0.03	0.04	0.08 (R4)

У таблиці Б.6 наведена класифікація ризиків за рівнем, відповідно до отриманого значення індексу.

Таблиця Б.6 – Шкала оцінювання ризику за рівнем

№	**Назва**	**Межі**	**Ризики, які входять**
1	Прийняті	$0,005 \leq R \leq 0,05$	R3, R7, R8, R10
2	Виправдані	$0,05 < R \leq 0,14$	R1, R2, R4, R5, R6, R9
3	Недопустимі	$0,14 < R \leq 0,72$	

Посилаючись до попередньо представлених даних, було розроблено заходи реагування на виявлені прийнятні та виправдані ризики проекту. Результати подані у формі таблиці Б.7.

Таблиця Б.8. – Ризики проекту та стратегії реагування

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	Тип стратегії реагування	План А	План Б
1	Виправданий	Проблема зі здоров'ям розробника	0.4	0.3	0.12	Зменшення	Регулярні перерви та тренування	Медична допомога та консультація.
2	Виправданий	Відключення електроживлення	0.8	0.1	0.08	Зменшення	Наявність резервного живлення у вигляді генератора	Швидка зміна джерела електроживлення
3	Прийнятний	Нестабільне Інтернет з’єднання	0.2	0.2	0.04	Зменшення	Використання різних постачальників Інтернет-зв'язку	Швидка зміна джерела Інтернету
4	Виправданий	Проблеми з апаратним забезпеченням	0.1	0.8	0.08	Зменшення	Регулярна підтримка та технічне обслуговування обладнання	Швидка заміна несправного обладнання

Продовження таблиці Б.10

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	Тип стратегії реагування	План А	План Б
5	Виправданий	Повітряні тривоги	0.3	0.3	0.09	Зменшення	Регулярний моніторинг повітряної тривоги	Швидка евакуація персоналу
6	Виправданий	Недоліки у тестуванні	0.4	0.2	0.08	Зменшення	Посилене тестування перед випуском	Швидке виявлення та виправлення помилок
7	Прийнятний	Неочікувані зміни в технологіях	0.1	0.2	0.02	Зменшення	Проведення постійного моніторингу технологічних тенденцій	Швидка адаптація та впровадження нових технологій у проект

Продовження таблиці Б.10

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	Тип стратегії реагування	План А	План Б
8	Прийнятний	Зміна потреб користувача	0.5	0.1	0.05	Зменшення	Ретельний аналіз потреб користувача	Швидке виправлення недоліків та пристосування продукту до потреб замовника
9	Виправданий	Недостатня документація	0.3	0.2	0.06	Зменшення	Консультація людей з досвідом у даній галузі	Швидке виявлення проблемних моментів у роботі та залучення додаткових експертів
10	Прийнятний	Невідповідність вимогам обладнання	0.3	0.1	0.03	Зменшення	Ретельний аналіз технічних потужностей користувачів	Швидке виявлення проблем та заміна не кросплапформних ділянок коду

ДОДАТОК В
ЛІСНИНГ ПРОГРАМНОГО КОДУ ОСНОВНИХ КОМПОНЕНТІВ ДОДАТКУ

Application.h

```
#pragma once

#include "Core.h"
#include "Time.h"

#include "Window.h"
#include "LayerStack.h"

#include "MyRevoke/EventSystem/Event.h"
#include "MyRevoke/EventSystem/AppEvent.h"
#include "MyRevoke/EventSystem/MouseEvent.h"
#include "MyRevoke/EventSystem/KeyEvent.h"

#include "MyRevoke/ImGui/ImGuiLayer.h"
#include "MyRevoke/Renderer/Shader.h"
#include "MyRevoke/Renderer/BuffersAPI.h"
#include "MyRevoke/Renderer/Cmaera.h"


namespace Revoke
{

	class  Application
	{
	public:
		Application();
		virtual ~Application();

		void Run();

		void OnEvent(Event& e);

		void PushLayer(Layer* layer);
		void PushOverlay(Layer* layer);

		void Close();

		ImGuiLayer* GetImGuiLayer() { return m_ImGuiLayer; }

		static Application& Get() {

			return *s_Instance;
```

```
        }

        static Application* GetPointer() {

            return s_Instance;
        }

        Window& GetWindow() { return *m_Window; }
        Window* GetWindowPointer() { return m_Window.get(); }

        Shared<Window> GetWindowScr() { return m_Window; }

    private:
        bool OnWindowClose(WindowsCloseEvent e);
        bool OnWindowResize(WindowResizeEvent e);

    private:

        Shared<Window> m_Window;
        LayerStack m_LayerStack;
        ImGuiLayer* m_ImGuiLayer;

        float m_LastFrame = 0;

        bool m_Run = true;

    private:
        static Application* s_Instance;
    };


    Application* CreateApplication();
}
```

Application.cpp

```
#include "rvpch.h"
#include "Application.h"
#include "Log.h"
#include "Input.h"
#include "Core.h"

#include "MyRevoke/Renderer/Renderer2D.h"
#include "MyRevoke/AudioManager/AudioRenderer.h"
#include "MyRevoke/Scripting/NativeScript.h"

#include "GLFW/glfw3.h"

namespace Revoke
{

    Application* Application::s_Instance = nullptr;
```

```
Application::Application()

{
	ALuint test = 2;
	RV_CORE_ASSERT(!s_Instance, "Application already exist");
	s_Instance = this;

	m_Window = std::make_shared<Window>();
	m_Window-
>SetEventCallback(RV_BIND_EVENT_FUNK(Application::OnEvent));


	RendererAPI::Init();
	Renderer2D::Init();
	AudioRenderer::Init();
	ScriptEngine::InitDll();


	m_ImGuiLayer = new ImGuiLayer();
	PushOverlay(m_ImGuiLayer);


}
Application::~Application()
{
	AudioRenderer::Shutdown();
	Renderer2D::Shutdown();
}
void Application::Run()
{

	while (m_Run)
	{
		float time = (float)glfwGetTime();
		Timestep timestep = time - m_LastFrame;
		m_LastFrame = time;

		for (Layer* layer : m_LayerStack)
			layer->OnUpdate(timestep);

		m_ImGuiLayer->Begin();
		for (Layer* layer : m_LayerStack)
			layer->OnImGuiDraw();
		m_ImGuiLayer->End();


		m_Window->OnUpdate();
	}
}
void Application::Close()
{
```

```
            m_Run = false;
      }
      void Application::OnEvent(Event& e)
      {

            EventDispatcher eDispatcher(e);


      eDispatcher.Dispatch<WindowsCloseEvent>(RV_BIND_EVENT_FUNK(Appl
ication::OnWindowClose));

      eDispatcher.Dispatch<WindowResizeEvent>(RV_BIND_EVENT_FUNK(Appl
ication::OnWindowResize));

            for (auto it = m_LayerStack.rbegin(); it !=
m_LayerStack.rend();++it)
            {
                  if (e.Handled)
                        break;
                  (*it)->OnEvent(e);
            }

      }
      void Application::PushLayer(Layer* layer)
      {
            m_LayerStack.PushLayer(layer);
            layer->OnAttach();
      }
      void Application::PushOverlay(Layer* layer)
      {
            m_LayerStack.PushOverlay(layer);
            layer->OnAttach();
      }
      bool Application::OnWindowClose(WindowsCloseEvent e)
      {
            m_Run = false;
            return true;
      }
      bool Application::OnWindowResize(WindowResizeEvent e)
      {
            RendererAPI::WindowResize((float)e.GetWidth(),
(float)e.GetHeight());
            return true;
      }
}
```

Renderer2D.h

```
#pragma once

#include "RendererAPI.h"
```

```
#include "Shader.h"
#include "Cmaera.h"
#include "Texture.h"
#include "EditorCamera.h"
#include "MyRevoke/Scene/Components.h"

#include "glm/glm.hpp"

namespace Revoke
{

	class Renderer2D
	{
	public:

		static void Init();
		static void Shutdown();

		static void Begin(const Camera& camera, const glm::mat4
transform);
		static void Begin(const EditorCamera& camera);
		static void End();



		static void DrawQuad(const glm::vec3& position, const
glm::vec2 size, const glm::vec4& color, int entityID);
		static void DrawQuad(const glm::vec2& position, const
glm::vec2 size, const glm::vec4& color, int entityID);
		static void DrawQuad(const glm::vec3& position, const
glm::vec2 size, const Shared<Texture>& texture, int entityID);
		static void DrawQuad(const glm::vec2& position, const
glm::vec2 size, const Shared<Texture>& texture, int entityID);

		static void DrawQuad(const glm::mat4& transform, const
glm::vec4& color, int entityID);
		static void DrawQuad(const glm::mat4& transform, const
Shared<Texture>& texture, int entityID);

		static void DrawSprite(const glm::mat4& transform,
SpriteRendererComponent& sprite, int entityID);

		static void QuadInit();
	private:
		static void NewBatch();
	public:

		// Stats
		struct Stats
		{
			uint32_t DrawCalls = 0;
			uint32_t QuadCount = 0;
```

```
            uint32_t GetTotalVertexCount() { return QuadCount * 4;
}
            uint32_t GetTotalIndexCount() { return QuadCount * 6;
}
        };

        static Stats GetStats();
        static void ResetStatistics();

    };

    }
```

Renderer2D.cpp

```
#include "rvpch.h"
#include "Renderer2D.h"

#include "imgui.h"
#include "glm/glm.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include <glm/gtc/type_ptr.hpp>

namespace Revoke
{
    struct QuadVertex
    {
        glm::vec3 Position;
        glm::vec4 Color;
        glm::vec2 TexCoord;
        float TexIndex;


        int EntityId = 0;
    };

    struct Data2D
    {
        static const uint32_t MaxEllements = 10000;
        static const uint32_t MaxVertices = MaxEllements * 4;
        static const uint32_t MaxIndices = MaxEllements * 6;
        static const uint32_t MaxTextures = 32;

        Shared <VertexArray> QuadVA;
        Shared <VertexBuffer> QuadVB;


        Shared <Shader> Shader;
        Shared <Texture> WhiteTexture;

        uint32_t QuadIndexCount = 0;

        QuadVertex* QuadVertexBufferBase = nullptr;
```

```
            QuadVertex* QuadVertexBufferPointer = nullptr;

            std::array<Shared <Texture>, MaxTextures> Textures;
            uint32_t TextureIndex = 1;

            glm::vec4 QuadVertexPositions[4];

            Renderer2D::Stats Statistic;
      };

      static Data2D* s_Data;

      static const glm::vec4 whiteColor = { 1.0f, 1.0f, 1.0f, 1.0f };

      void Renderer2D::Init()
      {
            s_Data = new Data2D();

            QuadInit();

            s_Data->WhiteTexture = std::make_shared<Texture>(1, 1);
            uint32_t whiteTextureData = 0xffffffff;
            s_Data->WhiteTexture->SetData(&whiteTextureData,
sizeof(uint32_t));

            int32_t samplers[s_Data->MaxTextures];
            for (uint32_t i = 0; i < s_Data->MaxTextures; i++)
            {
                  samplers[i] = i;
            }

            s_Data->Shader =
std::make_shared<Shader>("assets/Shaders/Main.shader");
            s_Data->Shader->Bind();
            s_Data->Shader->SetUniformIntArr("u_Textures", samplers,
s_Data->MaxTextures);

            s_Data->Textures[0] = s_Data->WhiteTexture;

            s_Data->QuadVertexPositions[0] = { -0.5f, -0.5f, 0.0f, 1.0f
};
            s_Data->QuadVertexPositions[1] = { 0.5f, -0.5f, 0.0f, 1.0f
};
            s_Data->QuadVertexPositions[2] = { 0.5f,  0.5f, 0.0f, 1.0f
};
            s_Data->QuadVertexPositions[3] = { -0.5f,  0.5f, 0.0f, 1.0f
};
      }

      void Renderer2D::Shutdown()
      {
            delete[] s_Data->QuadVertexBufferBase;
      }
```

```
	void Renderer2D::Begin(const Camera& camera, const glm::mat4
transform)
	{

		glm::mat4 viewProj = camera.GetProjectionMatrix() *
glm::inverse(transform);

		s_Data->Shader->Bind();
		s_Data->Shader->SetUniformMat4("u_PVmatrix", viewProj);

		s_Data->QuadIndexCount = 0;
		s_Data->QuadVertexBufferPointer = s_Data-
>QuadVertexBufferBase;
		s_Data->TextureIndex = 1;
	}

	void Renderer2D::Begin(const EditorCamera& camera)
	{

		glm::mat4 viewProj = camera.GetViewProjection();

		s_Data->Shader->Bind();
		s_Data->Shader->SetUniformMat4("u_PVmatrix", viewProj);

		s_Data->QuadIndexCount = 0;
		s_Data->QuadVertexBufferPointer = s_Data-
>QuadVertexBufferBase;
		s_Data->TextureIndex = 1;
	}

	void Renderer2D::End()
	{
		//Drawing

		uint32_t dataSize = (uint32_t)((uint8_t*)s_Data-
>QuadVertexBufferPointer - (uint8_t*)s_Data->QuadVertexBufferBase);
		s_Data->QuadVB->InitData(s_Data->QuadVertexBufferBase,
dataSize);

		for (uint32_t i = 0; i < s_Data->TextureIndex; i++)
		{
			s_Data->Textures[i]->Bind(i);
		}
		s_Data->Shader->Bind();
		RendererAPI::DrawElements(s_Data->QuadVA, s_Data-
>QuadIndexCount);
	}

	//Color Draw
	void Renderer2D::DrawQuad(const glm::vec3& position, const
glm::vec2 size, const glm::vec4& color, int entityID)
	{
```

```
          glm::mat4 transform = glm::translate(glm::mat4(1.0f),
position)
               * glm::scale(glm::mat4(1.0f), { size.x, size.y, 1.0f
});

          DrawQuad(transform, color, entityID);

          if (s_Data->QuadIndexCount >= Data2D::MaxIndices)
               NewBatch();

     }

     void Renderer2D::DrawQuad(const glm::vec2& position, const
glm::vec2 size, const glm::vec4& color, int entityID)
     {
          DrawQuad({ position.x, position.y, 0.0f }, size, color,
entityID);
     }

     //Texture Draw
     void Renderer2D::DrawQuad(const glm::vec3& position, const
glm::vec2 size, const Shared<Texture>& texture, int entityID)
     {

          glm::mat4 transform = glm::translate(glm::mat4(1.0f),
position)
               * glm::scale(glm::mat4(1.0f), { size.x, size.y, 1.0f
});

          DrawQuad(transform, texture, entityID);

     }

     void Renderer2D::DrawQuad(const glm::vec2& position, const
glm::vec2 size, const Shared<Texture>& texture, int entityID)
     {
          DrawQuad({ position.x, position.y, 0.0f }, size, texture,
entityID);
     }

     void Renderer2D::DrawQuad(const glm::mat4& transform, const
glm::vec4& color, int entityID)
     {

          constexpr size_t quadVertexCount = 4;
          const float textureIndex = 0.0f; // White Texture
          constexpr glm::vec2 textureCoords[] = { { 0.0f, 0.0f }, {
1.0f, 0.0f }, { 1.0f, 1.0f }, { 0.0f, 1.0f } };

          if (s_Data->QuadIndexCount >= Data2D::MaxIndices)
               NewBatch();
```

```
          for (size_t i = 0; i < quadVertexCount; i++)
          {
               s_Data->QuadVertexBufferPointer->Position = transform
* s_Data->QuadVertexPositions[i];
               s_Data->QuadVertexBufferPointer->Color = color;
               s_Data->QuadVertexBufferPointer->TexCoord =
textureCoords[i];
               s_Data->QuadVertexBufferPointer->TexIndex =
textureIndex;
               s_Data->QuadVertexBufferPointer->EntityId = entityID;
               s_Data->QuadVertexBufferPointer++;
          }

          s_Data->QuadIndexCount += 6;

          s_Data->Statistic.QuadCount++;
     }

     void Renderer2D::DrawQuad(const glm::mat4& transform, const
Shared<Texture>& texture, int entityID)
     {
          constexpr size_t quadVertexCount = 4;
          constexpr glm::vec2 textureCoords[] = { { 0.0f, 0.0f }, {
1.0f, 0.0f }, { 1.0f, 1.0f }, { 0.0f, 1.0f } };

          if (s_Data->QuadIndexCount >= Data2D::MaxIndices)
               NewBatch();

          float textureIndex = 0.0f;

          for (uint32_t i = 1; i < s_Data->TextureIndex; i++)
          {
               if (s_Data->Textures[i]->GetID() == texture->GetID())
               {
                    textureIndex = (float)i;
                    break;
               }
          }

          if (textureIndex == 0.0f)
          {
               if (s_Data->TextureIndex >= Data2D::MaxTextures)
                    NewBatch();
               textureIndex = (float)s_Data->TextureIndex;
               s_Data->Textures[s_Data->TextureIndex] = texture;
               s_Data->TextureIndex++;
          }

          for (size_t i = 0; i < quadVertexCount; i++)
          {
               s_Data->QuadVertexBufferPointer->Position = transform
* s_Data->QuadVertexPositions[i];
               s_Data->QuadVertexBufferPointer->Color = whiteColor;
```

```
                s_Data->QuadVertexBufferPointer->TexCoord =
textureCoords[i];
                s_Data->QuadVertexBufferPointer->TexIndex =
textureIndex;
                s_Data->QuadVertexBufferPointer->EntityId = entityID;
                s_Data->QuadVertexBufferPointer++;
          }

          s_Data->QuadIndexCount += 6;

          s_Data->Statistic.QuadCount++;
     }


     void Renderer2D::DrawSprite(const glm::mat4& transform,
SpriteRendererComponent& sprite, int entityID)
     {
          if (!sprite.Texture2D.empty())
          {
               Shared<Texture> texture;
               texture = std::make_shared<Texture>(sprite.Texture2D);
               DrawQuad(transform, texture, entityID);
          }
          else
          {
               DrawQuad(transform, sprite.Color, entityID);
          }
     }

     void Renderer2D::QuadInit()
     {
          s_Data->QuadVA = std::make_shared<VertexArray>();

          s_Data->QuadVB = std::make_shared<VertexBuffer>(s_Data-
>MaxVertices * sizeof(QuadVertex));

          BufferLayout quadLayout = {
               {ShaderDataTypes::Float3, "a_Position", false},
               {ShaderDataTypes::Float4, "a_Color", false},
               {ShaderDataTypes::Float2, "a_TexCoord", false},
               {ShaderDataTypes::Float,  "a_TexIndex", false},
               {ShaderDataTypes::Int,  "a_EntityID", false}
          };
          s_Data->QuadVB->SetLayout(quadLayout);

          s_Data->QuadVA->AddVertexBuffer(s_Data->QuadVB);

          s_Data->QuadVertexBufferBase = new QuadVertex[s_Data-
>MaxVertices];

          uint32_t* quadIndices = new uint32_t[s_Data->MaxIndices];
```

```
            uint32_t offset = 0;
            for (uint32_t i = 0; i < s_Data->MaxIndices; i += 6)
            {
                  quadIndices[i] = offset ;
                  quadIndices[i + 1] = offset + 1;
                  quadIndices[i + 2] = offset + 2;

                  quadIndices[i + 3] = offset + 2;
                  quadIndices[i + 4] = offset + 3;
                  quadIndices[i + 5] = offset;

                  offset += 4;
            }

            Shared<IndexBuffer> QuadIndexBuffer;
            QuadIndexBuffer = std::make_shared<
IndexBuffer>(quadIndices, s_Data->MaxIndices);

            s_Data->QuadVA->SetIndexBuffer(QuadIndexBuffer);

            delete[] quadIndices;
      }


      void Renderer2D::NewBatch()
      {
            End();
            s_Data->QuadIndexCount = 0;
            s_Data->QuadVertexBufferPointer = s_Data-
>QuadVertexBufferBase;
            s_Data->TextureIndex = 1;
      }

      Renderer2D::Stats Renderer2D::GetStats()
      {
            return s_Data->Statistic;
      }

      void Renderer2D::ResetStatistics()
      {
            memset(&s_Data->Statistic, 0, sizeof(Stats));
      }

      }
```

SceneSettingsPannel.h

```
#pragma once

#include <glm/glm.hpp>
```

```
#include "MyRevoke/Scene/Scene.h"

namespace Revoke
{

	class SceneSettingsPannel
	{
	public:
		SceneSettingsPannel();
		~SceneSettingsPannel();

		void OnImGuiRender();

		void SetScene(Shared<Scene> currentScene);

	private:
		int m_GravityPositionIteration = 4;
		int GravityVelocityIteration = 6;

		Shared<Scene> m_CurrentScene;
		glm::vec3 m_ClearColor = { 0.2, 0.2, 0.2 };
		bool m_EnebleBlending = true;
	};
}
```

SceneSettingPannel.cpp

```
#include "rvpch.h"
#include "SceneSettingsPannel.h"

#include <imgui.h>
#include <glm/gtc/type_ptr.hpp>

#include "MyRevoke/Renderer/RendererAPI.h"


namespace Revoke
{
	static char* sceneName;

	SceneSettingsPannel::SceneSettingsPannel()
	{

	}
	SceneSettingsPannel::~SceneSettingsPannel()
	{
		delete[] sceneName;
	}

	void SceneSettingsPannel::OnImGuiRender()
	{
		ImGui::Begin("Scene Settings");
```

```
            if (ImGui::InputText("Scene Name", sceneName,
sizeof(sceneName)))
            {
                  std::string str(sceneName);
                  m_CurrentScene->SetName(str);
            }

            if (ImGui::ColorEdit3("Set Clear Color",
glm::value_ptr(m_ClearColor)));
                  RendererAPI::SetClearColor({ m_ClearColor.r,
m_ClearColor.g, m_ClearColor.b, 1.0f });

            if(ImGui::Checkbox("Eneble Blending", &m_EnebleBlending))
                  RendererAPI::EnableBlending();

            if(ImGui::DragInt("Set Gravity Position Iteration",
&m_GravityPositionIteration, 0, 100))
                  m_CurrentScene-
>SetGravityPositionIteration(m_GravityPositionIteration);

            if (ImGui::DragInt("Set Gravity Velocity Iteration",
&GravityVelocityIteration, 0, 100))
                  m_CurrentScene-
>SetGravityVelocityIteration(GravityVelocityIteration);

            if (ImGui::Button("Build Scripts"))
            {
#ifdef RV_DEBUG
                  system("msbuild ../MyRevoke-NativeScriptCore/MyRevoke-
NativeScriptCore.vcxproj /p:Configuration=Debug /p:Platform=x64");
#else
                  system("msbuild ../MyRevoke-NativeScriptCore/MyRevoke-
NativeScriptCore.vcxproj /p:Configuration=Release /p:Platform=x64");
#endif

            }

            ImGui::End();
      }

      void SceneSettingsPannel::SetScene(Shared<Scene> currentScene)
      {
            m_CurrentScene = currentScene;
            const std::string& tempName = currentScene->GetName();
            sceneName = new char[tempName.size() + 1];
            strcpy(sceneName, tempName.c_str());
      }

      }
```

ObjectPannel.cpp

```
#include "ObjectsPannel.h"

#include "imgui.h"

#include <glm/gtc/type_ptr.hpp>

namespace Revoke
{

	extern const std::filesystem::path g_AssetsDirectory;

	struct TempData
	{
		std::string CurrentScriptName = "Script";
		std::string CurrentTextureName = "Texture";
		std::string CurrentSoundName = "Sound";
	};

	static TempData s_TempData;

	ObjectsPannel::ObjectsPannel(Shared<Scene> currentScene)
		:m_CurrentScene(currentScene)
	{
	}

	void ObjectsPannel::SetScene(Shared<Scene> currentScene)
	{
		m_CurrentScene = currentScene;
		m_SelectedEntity = {};
	}

	void ObjectsPannel::OnImGuiRender()
	{
		ImGui::Begin("Scene Hierarchy");
		if (m_CurrentScene)
		{
			m_CurrentScene->m_Registry.each([&](auto
entityID)
				{
					Entity	entity{	entityID	,
m_CurrentScene.get() };
					SceneHierarchyWindow(entity);
				});
			if		(ImGui::IsMouseDown(0)		&&
ImGui::IsWindowHovered())
				m_SelectedEntity = {};

			if	(ImGui::BeginPopupContextWindow(0,	1	|
ImGuiPopupFlags_NoOpenOverItems))
			{
				if (ImGui::MenuItem("Create Entity"))
				{
```

```
                    m_CurrentScene->CreateEntity("Empty
Entity");
                }
                ImGui::EndPopup();
            }

        }

        ImGui::End();

        ImGui::Begin("Properties");
        PropertiesWindow();
        ImGui::End();
    }
    void ObjectsPannel::SceneHierarchyWindow(Entity entity)
    {
                bool entityExist = true;
                auto&               entityName               =
entity.GetComponent<NameComponent>().Name;
                ImGuiTreeNodeFlags flags = ((m_SelectedEntity
==    entity)    ?    ImGuiTreeNodeFlags_Selected    :    0)    |
ImGuiTreeNodeFlags_OpenOnArrow;
                flags |= ImGuiTreeNodeFlags_SpanAvailWidth;

                bool                opened                =
ImGui::TreeNodeEx((void*)(uint64_t)(uint32_t)entity,      flags,
entityName.c_str());
                if (ImGui::IsItemClicked())
                {
                    m_SelectedEntity = entity;
                }

                if (ImGui::BeginPopupContextItem())
                {
                    if (ImGui::MenuItem("Delete Entity"))
                    {
                        entityExist = false;
                    }
                    ImGui::EndPopup();
                }

                if (opened)
                {
                    ImGuiTreeNodeFlags        flags        =
ImGuiTreeNodeFlags_OpenOnArrow                              |
ImGuiTreeNodeFlags_SpanAvailWidth;
                    bool              opened              =
ImGui::TreeNodeEx((void*)9817239, flags, entityName.c_str());
                    if (opened)
                        ImGui::TreePop();
                    ImGui::TreePop();
                }
                if (!entityExist)
```

```
                    {
                         m_CurrentScene->RemoveEntity(entity);
                         if (m_SelectedEntity == entity)
                              m_SelectedEntity = {};
                    }

     }
     void ObjectsPannel::PropertiesWindow()
     {
          if (m_SelectedEntity)
          {
               if
(m_SelectedEntity.HasComponent<NameComponent>())
               {
                    auto&                    name                    =
m_SelectedEntity.GetComponent<NameComponent>().Name;

                    char buffer[256];
                    memset(buffer, 0, sizeof(buffer));
                    strcpy_s(buffer, 256, name.c_str());
                    if     (ImGui::InputText("Name",     buffer,
sizeof(buffer)))
                    {
                         name = std::string(buffer);
                    }
               }



               if
(m_SelectedEntity.HasComponent<TransformComponent>())
               {
                    if
(ImGui::TreeNodeEx((void*)typeid(TransformComponent).hash_code(
), ImGuiTreeNodeFlags_DefaultOpen, "Transform"))
                    {
                         auto&        transformComponent        =
m_SelectedEntity.GetComponent<TransformComponent>();

                         ImGui::DragFloat3("Position",
glm::value_ptr(transformComponent.Position), 0.1f);
                         ImGui::DragFloat3("Rotation",
glm::value_ptr(transformComponent.Rotation), 0.1f);
                         ImGui::DragFloat3("Scale",
glm::value_ptr(transformComponent.Scale), 0.1f);

                         ImGui::TreePop();
                    }
               }
               if
(m_SelectedEntity.HasComponent<SpriteRendererComponent>())
               {
```

```
                    if
(ImGui::TreeNodeEx((void*)typeid(SpriteRendererComponent).hash_
code(), ImGuiTreeNodeFlags_DefaultOpen, "Color"))
                    {
                        auto&          spriteComponent          =
m_SelectedEntity.GetComponent<SpriteRendererComponent>();
                        ImGui::ColorEdit4("Color",
glm::value_ptr(spriteComponent.Color));


     ImGui::Button(s_TempData.CurrentTextureName.c_str());

                        if (ImGui::BeginDragDropTarget())
                        {
                            if (const ImGuiPayload* payload =
ImGui::AcceptDragDropPayload("CONTENT_BROWSER_PAYLOAD"))
                            {
                                const wchar_t* path = (const
wchar_t*)payload->Data;
                                std::filesystem::path
texturePath = std::filesystem::path(g_AssetsDirectory) / path;

                                s_TempData.CurrentTextureName
= texturePath.stem().string();
                                spriteComponent.Texture2D    =
texturePath.string();
                            }

                            ImGui::EndDragDropTarget();
                        }

                        ImGui::TreePop();
                    }
                }
                if
(m_SelectedEntity.HasComponent<SoundComponent>())
                {
                    if
(ImGui::TreeNodeEx((void*)typeid(SoundComponent).hash_code(),
ImGuiTreeNodeFlags_DefaultOpen, "Sound Settings"))
                    {
                        auto&          soundComponent          =
m_SelectedEntity.GetComponent<SoundComponent>();


     ImGui::Button(s_TempData.CurrentSoundName.c_str());

                        if (ImGui::BeginDragDropTarget())
                        {
                            if (const ImGuiPayload* payload =
ImGui::AcceptDragDropPayload("CONTENT_BROWSER_PAYLOAD"))
                            {
```

```
                                        const wchar_t* path = (const
wchar_t*)payload->Data;
                                        std::filesystem::path
soundPath = std::filesystem::path(g_AssetsDirectory) / path;
                                        s_TempData.CurrentSoundName =
soundPath.stem().string();


     soundComponent.SetPath(soundPath.string());

                                   }
                                   ImGui::EndDragDropTarget();
                              }
                              ImGui::DragFloat("Pitch",
&soundComponent.Pitch, 0.1f);
                              ImGui::DragFloat("Gain",
&soundComponent.Gain, 0.1f);
                              ImGui::DragFloat3("Position",
glm::value_ptr(soundComponent.Position), 0.1f);
                              ImGui::DragFloat3("Velocity",
glm::value_ptr(soundComponent.Velocity), 0.1f);

                              //TODO: Maybe add on propertiesUpdate

                              ImGui::Checkbox("Loop            Audio",
&soundComponent.LoopSound);


                              ImGui::TreePop();
                         }
                    }
                    if
(m_SelectedEntity.HasComponent<RigidBodyComponent>())
                    {
                         if
(ImGui::TreeNodeEx((void*)typeid(RigidBodyComponent).hash_code(
), ImGuiTreeNodeFlags_DefaultOpen, "Rigid Body"))
                         {
                              auto&          bodyComponent          =
m_SelectedEntity.GetComponent<RigidBodyComponent>();

                              const char* bodyType[] = { "StaticBody",
"KinematicBody", "DynamicBody" };
                              const    char*    currentBodyType    =
bodyType[(int)bodyComponent.Type];

                              if    (ImGui::BeginCombo("Body    Type",
currentBodyType))
                              {
                                   for (int i = 0; i < 3; i++)
                                   {
                                        bool       isSelected       =
currentBodyType == bodyType[i];
```

```
                                        if
(ImGui::Selectable(bodyType[i], isSelected))
                                        {
                                             currentBodyType         =
bodyType[i];
                                             bodyComponent.Type      =
(RigidBodyComponent::BodyType)i;
                                        }
                                        if (isSelected)

     ImGui::SetItemDefaultFocus();
                                   }

                                   ImGui::EndCombo();
                              }
                              ImGui::Checkbox("Fixed       Rotation",
&bodyComponent.IsRotating);

                              ImGui::TreePop();
                         }
                    }

                    if
(m_SelectedEntity.HasComponent<BoxColisionComponent>())
                    {
                         if
(ImGui::TreeNodeEx((void*)typeid(BoxColisionComponent).hash_cod
e(), ImGuiTreeNodeFlags_DefaultOpen, "Colisions"))
                         {
                              auto&      boxCollisionComponent      =
m_SelectedEntity.GetComponent<BoxColisionComponent>();

                              ImGui::DragFloat2("Size",
glm::value_ptr(boxCollisionComponent.Size), 0.1f);
                              ImGui::DragFloat2("Offset",
glm::value_ptr(boxCollisionComponent.Offset), 0.1f);

                              ImGui::DragFloat("Density",
&boxCollisionComponent.Density, 0.01f, 0.0f, 1.0f);
                              ImGui::DragFloat("Friction",
&boxCollisionComponent.Friction, 0.01f, 0.0f, 1.0f);
                              ImGui::DragFloat("Restriction",
&boxCollisionComponent.Restriction, 0.01f, 0.0f, 1.0f);
                              ImGui::DragFloat("Resitution
Treshhold",  &boxCollisionComponent.ResitutionTreshhold,  0.01f,
0.0f);

                              ImGui::Checkbox("Sensor",
&boxCollisionComponent.isSensor);

                              ImGui::TreePop();
                         }
                    }
```

```
            if
(m_SelectedEntity.HasComponent<NativeScriptComponent>())
            {
                if
(ImGui::TreeNodeEx((void*)typeid(NativeScriptComponent).hash_co
de(), ImGuiTreeNodeFlags_DefaultOpen, "Scripts"))
                {
                    auto&      nativeScriptComponent      =
m_SelectedEntity.GetComponent<NativeScriptComponent>();


    ImGui::Button(s_TempData.CurrentScriptName.c_str());

                    if (ImGui::BeginDragDropTarget())
                    {
                        if (const ImGuiPayload* payload =
ImGui::AcceptDragDropPayload("CONTENT_BROWSER_PAYLOAD"))
                        {
                            const wchar_t* path = (const
wchar_t*)payload->Data;
                            std::filesystem::path
scriptName = path;
                            s_TempData.CurrentScriptName
= scriptName.stem().string();

    nativeScriptComponent.scriptClassName                    =
s_TempData.CurrentScriptName;

    nativeScriptComponent.Instance = nullptr;
                        }

                        ImGui::EndDragDropTarget();
                    }

                    ImGui::SameLine();
                    if (ImGui::Button("X", ImVec2(20, 20)))
{
                        s_TempData.CurrentScriptName     =
"Script";

    nativeScriptComponent.scriptClassName.clear();
                        nativeScriptComponent.Instance   =
nullptr;
                    }

                    ImGui::TreePop();
                }
            }

            if
(m_SelectedEntity.HasComponent<CameraComponent>())
            {
```

```
                    if
(ImGui::TreeNodeEx((void*)typeid(CameraComponent).hash_code(),
ImGuiTreeNodeFlags_DefaultOpen, "Camera"))
                    {
                         auto&          cameraComponent          =
m_SelectedEntity.GetComponent<CameraComponent>();

                         ImGui::Checkbox("Primary",
&cameraComponent.isMain);


                         const  char*  projectionType[]  =  {  "3D
Camera", "2D Camera" };
                         const   char*   currentProjectionType   =
projectionType[(int)cameraComponent.Camera.GetProjectionType()]
;

                         if   (ImGui::BeginCombo("Camera   Type",
currentProjectionType))
                         {
                              for (int i = 0; i < 2; i++)
                              {
                                   bool       isSelected       =
currentProjectionType == projectionType[i];
                                   if
(ImGui::Selectable(projectionType[i], isSelected))
                                   {
                                        currentProjectionType   =
projectionType[i];

     cameraComponent.Camera.SetProjectionType((SceneCamera::Proj
ection)i);
                                   }
                                   if (isSelected)

     ImGui::SetItemDefaultFocus();
                              }

                              ImGui::EndCombo();
                         }

                         if
(cameraComponent.Camera.GetProjectionType()                    ==
SceneCamera::Projection::Perspective)
                         {
                              float             FOV             =
glm::degrees(cameraComponent.Camera.GetPerspFOV());
                              if (ImGui::DragFloat("FOV", &FOV))

     cameraComponent.Camera.SetPerspFOV(glm::radians(FOV));

                              float          nearClip          =
cameraComponent.Camera.GetPerspNearClip();
```

```
                              if  (ImGui::DragFloat("Near  Clip",
&nearClip))

     cameraComponent.Camera.SetPerspNearClip(nearClip);

                              float           farClip           =
cameraComponent.Camera.GetPerspFarClip();
                              if   (ImGui::DragFloat("Far  Clip",
&farClip))

     cameraComponent.Camera.SetPerspFarClipl(farClip);
                         }
                         if
(cameraComponent.Camera.GetProjectionType()                    ==
SceneCamera::Projection::Orthographic)
                         {
                              float             size            =
cameraComponent.Camera.GetOrthoSize();
                              if        (ImGui::DragFloat("Size",
&size))

     cameraComponent.Camera.SetOrthoSize(size);

                              float           nearClip          =
cameraComponent.Camera.GetOrthoNearClip();
                              if  (ImGui::DragFloat("Near  Clip",
&nearClip))

     cameraComponent.Camera.SetOrthoNearClip(nearClip);

                              float           farClip           =
cameraComponent.Camera.GetOrthoFarClip();
                              if   (ImGui::DragFloat("Far  Clip",
&farClip))

     cameraComponent.Camera.SetOrthoFarClipl(farClip);

                         }
                         ImGui::TreePop();
                    }
               }


               if (ImGui::Button("Add Component"))
                    ImGui::OpenPopup("Add Component");

               //TODO add all components

               if (ImGui::BeginPopup("Add Component"))
               {
                    if (ImGui::MenuItem("Sprite Renderer"))
                    {
```

```
            m_SelectedEntity.AddComponent<SpriteRendererComponent>();
                                ImGui::CloseCurrentPopup();
                            }
                            if (ImGui::MenuItem("Rigid Body"))
                            {

            m_SelectedEntity.AddComponent<RigidBodyComponent>();
                                ImGui::CloseCurrentPopup();
                            }
                            if (ImGui::MenuItem("Box Colidor"))
                            {

            m_SelectedEntity.AddComponent<BoxColisionComponent>();
                                ImGui::CloseCurrentPopup();
                            }
                            if (ImGui::MenuItem("Sound Component"))
                            {

            m_SelectedEntity.AddComponent<SoundComponent>();
                                ImGui::CloseCurrentPopup();
                            }

                            if (ImGui::MenuItem("Camera"))
                            {

            m_SelectedEntity.AddComponent<CameraComponent>();
                                ImGui::CloseCurrentPopup();
                            }
                            if      (ImGui::MenuItem("Native      Script
Component"))
                            {

            m_SelectedEntity.AddComponent<NativeScriptComponent>();
                                ImGui::CloseCurrentPopup();
                            }
                            ImGui::EndPopup();
                        }
                    }
            }
            void ObjectsPannel::SetSelectedEntity(Entity entity)
            {
                m_SelectedEntity = entity;
            }
}
```

CraftLayer.h

```
#include "MyRevoke.h"

#include <imgui.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
```

```
#include <glm/gtc/type_ptr.hpp>

#include <filesystem>

#include "ObjectsPannel.h"
#include "ContentBrowser.h"
#include "ToolBar.h"
#include "SceneSettingsPannel.h"

namespace Revoke
{

	class CraftLayer : public Revoke::Layer
	{
	public:
		CraftLayer();
		~CraftLayer();

		void OnAttach() override;
		void OnDetach() override;

		void OnUpdate(Timestep deltaTime) override;
		void OnImGuiDraw() override;
		void OnEvent(Event& e) override;

		bool OnMouseBtnPressed(MouseButtonPressedEvent& e);
		bool OnKeyPressed(KeyPressedEvent& e);

		void NewScene();
		void OpenScene();
		void OpenScene(const std::filesystem::path& path);
		void SaveAs();
		void Save();

		Shared<Scene> GetCurrentScene() const { return m_Scene; }

	private:

		Shared<FrameBuffers> m_FrameBuffer;

		Shared<Scene> m_Scene;

		Entity m_CameraEntity;
		Entity m_SecondCamera;

		Entity m_HoveredEntity;

		EditorCamera m_EditorCamera;

		ObjectsPannel m_ObjPannel;
		ContentBrowser m_ContentBrowserPanel;
		ToolBar m_ToolBar;
		SceneSettingsPannel m_ProjectSettingsPanel;
```

```
        bool m_PrimaryCamera = true;
        glm::vec2 m_ViewportSize = { 0.0f, 0.0f };
        bool m_ViewportFocused = false;
        bool m_ViewportHovered = false;

        glm::vec2 m_ViewportBounds[2];

        int* m_GizmoType;

        struct ProfileResult
        {
            const char* Name;
            float Time;
        };

};
}
```

CraftLayer.cpp

```
#include "CraftLayer.h"

#include <chrono>
#include <string>

#include "MyRevoke/Utility/FileExplorer.h"
#include "MyRevoke/Math/Math.h"
#include "MyRevoke/AudioManager/AudioRenderer.h"
#include "MyRevoke/Scripting/NativeScript.h"
#include "MyRevoke/Core/Input.h"
#include <ImGuizmo.h>

namespace Revoke
{
    extern const std::filesystem::path g_AssetsDirectory =
"assets";

    CraftLayer::CraftLayer()
        :Layer("CraftLayer")
    {

    }
    CraftLayer::~CraftLayer()
    {
        m_Scene->OnSceneClose();
    }
    void CraftLayer::OnAttach()
    {

        FrameBufferStats frameBufferStats;
```

```
          frameBufferStats.Attachments            =            {
FramebufferTextureFormat::RGBA8,FramebufferTextureFormat::RED_I
NTEGER, FramebufferTextureFormat::DEPTH24STENCIL8 };
          frameBufferStats.Width = 1280;
          frameBufferStats.Height = 720;
          m_FrameBuffer                                        =
std::make_shared<FrameBuffers>(frameBufferStats);
          m_Scene = std::make_shared<Scene>("Main scene");

          m_EditorCamera  =  EditorCamera(30.0f,  1.778f,  0.1f,
1000.0f);

          m_ObjPannel.SetScene(m_Scene);
          m_ToolBar.SetScene(m_Scene);
          m_ProjectSettingsPanel.SetScene(m_Scene);

          m_GizmoType = new int(-1);

          m_ToolBar.SetGuizmo(m_GizmoType);

          RendererAPI::SetClearColor({ 0.2f, 0.2f, 0.2f, 1.0f });
          RendererAPI::EnableBlending();


     }
     void CraftLayer::OnDetach()
     {
          delete m_GizmoType;
     }
     void CraftLayer::OnUpdate(Timestep deltaTime)
     {
          // Resize
          if    (FrameBufferStats    spec    =    m_FrameBuffer-
>GetSpecification();
               m_ViewportSize.x > 0.0f && m_ViewportSize.y > 0.0f
&&
               (spec.Width != m_ViewportSize.x || spec.Height !=
m_ViewportSize.y))
          {
               m_FrameBuffer->Resize((uint32_t)m_ViewportSize.x,
(uint32_t)m_ViewportSize.y);
               m_EditorCamera.SetViewportSize(m_ViewportSize.x,
m_ViewportSize.y);

               m_Scene-
>OnViewportResize((uint32_t)m_ViewportSize.x,
(uint32_t)m_ViewportSize.y);
          }


          m_FrameBuffer->Bind();

          RendererAPI::Clear();
```

```
          m_FrameBuffer->ClearColorTextureAttachment(1, -1);

          ScriptEngine::OnUpdate();

          //TODO: Fix the picking in a play mode!!!
          switch (m_ToolBar.GetSceneState())
          {
          case SceneState::Editor:
          {
               if (m_ViewportFocused)
               {
                    m_EditorCamera.OnUpdate(deltaTime);
               }

               m_Scene->OnEditorUpdate(deltaTime,
m_EditorCamera);
               break;
          }
          case SceneState::Runtime:
          {
               m_Scene->OnRuntimeUpdate(deltaTime);
               break;
          }
          }

          auto [mx, my] = ImGui::GetMousePos();
          mx -= m_ViewportBounds[0].x;
          my -= m_ViewportBounds[0].y;
          glm::vec2   viewportSize   =   m_ViewportBounds[1]   -
m_ViewportBounds[0];
          my = viewportSize.y - my;
          int mouseX = (int)mx;
          int mouseY = (int)my;
          if   (mouseX   >=   0   &&   mouseY   >=   0   &&   mouseX   <
(int)viewportSize.x && mouseY < (int)viewportSize.y)
          {
               int   pixelData   =   m_FrameBuffer->ReadPixel(1,
mouseX, mouseY);
               m_HoveredEntity = pixelData == -1 ? Entity() :
Entity((entt::entity)pixelData, m_Scene.get());
          }

          m_FrameBuffer->UnBind();

     }

     void CraftLayer::OnImGuiDraw()
     {
          static bool dockspaceOpen = true;
          static bool opt_fullscreen_persistant = true;
          bool opt_fullscreen = opt_fullscreen_persistant;
```

```
          static     ImGuiDockNodeFlags     dockspace_flags     =
ImGuiDockNodeFlags_None;

          ImGuiWindowFlags             window_flags             =
ImGuiWindowFlags_MenuBar | ImGuiWindowFlags_NoDocking;
          if (opt_fullscreen)
          {
               ImGuiViewport*             viewport             =
ImGui::GetMainViewport();
               ImGui::SetNextWindowPos(viewport->Pos);
               ImGui::SetNextWindowSize(viewport->Size);
               ImGui::SetNextWindowViewport(viewport->ID);
               ImGui::PushStyleVar(ImGuiStyleVar_WindowRounding,
0.0f);

     ImGui::PushStyleVar(ImGuiStyleVar_WindowBorderSize, 0.0f);
               window_flags   |=   ImGuiWindowFlags_NoTitleBar   |
ImGuiWindowFlags_NoCollapse   |   ImGuiWindowFlags_NoResize   |
ImGuiWindowFlags_NoMove;
               window_flags                                      |=
ImGuiWindowFlags_NoBringToFrontOnFocus                            |
ImGuiWindowFlags_NoNavFocus;
          }
          if                  (dockspace_flags                  &
ImGuiDockNodeFlags_PassthruCentralNode)
               window_flags |= ImGuiWindowFlags_NoBackground;

          ImGui::PushStyleVar(ImGuiStyleVar_WindowPadding,
ImVec2(0.0f, 0.0f));
          ImGui::Begin("DockSpace     Demo",     &dockspaceOpen,
window_flags);
          ImGui::PopStyleVar();

          if (opt_fullscreen)
               ImGui::PopStyleVar(2);

          // DockSpace
          ImGuiIO& io = ImGui::GetIO();
          if (io.ConfigFlags & ImGuiConfigFlags_DockingEnable)
          {
               ImGuiID               dockspace_id               =
ImGui::GetID("MyDockSpace");
               ImGui::DockSpace(dockspace_id,       ImVec2(0.0f,
0.0f), dockspace_flags);
          }

          if (ImGui::BeginMenuBar())
          {
               if (ImGui::BeginMenu("File"))
               {
                    if                 (ImGui::MenuItem("Exit"))
Application::Get().Close();
```

```
                    if (ImGui::MenuItem("New Scene", "Ctrl+N"))
NewScene();
                    if    (ImGui::MenuItem("Open",    "Ctrl+O"))
OpenScene();
                    if    (ImGui::MenuItem("Save",    "Ctrl+S"))
Save();
                    if        (ImGui::MenuItem("Save        as",
"Ctrl+Shift+S")) SaveAs();

                    ImGui::EndMenu();
                }


                ImGui::EndMenuBar();
            }

            //-----------Settings----------------------------------
----------------

            //-----------------------------------------------------
----------------
            m_ObjPannel.OnImGuiRender();
            m_ContentBrowserPanel.OnImGuiRender();
            m_ProjectSettingsPanel.OnImGuiRender();
            //-----------ViewPort----------------------------------
----------------
            ImGui::PushStyleVar(ImGuiStyleVar_WindowPadding,
ImVec2{ 0, 0 });
            ImGui::Begin("Viewport");

            auto                viewportMinRegion                =
ImGui::GetWindowContentRegionMin();
            auto                viewportMaxRegion                =
ImGui::GetWindowContentRegionMax();
            auto viewportOffset = ImGui::GetWindowPos();
            m_ViewportBounds[0]   =   {   viewportMinRegion.x   +
viewportOffset.x, viewportMinRegion.y + viewportOffset.y };
            m_ViewportBounds[1]   =   {   viewportMaxRegion.x   +
viewportOffset.x, viewportMaxRegion.y + viewportOffset.y };

            m_ViewportFocused = ImGui::IsWindowFocused();
            m_ViewportHovered = ImGui::IsWindowHovered();
            Application::Get().GetImGuiLayer()-
>BlockEvents(!m_ViewportFocused && !m_ViewportHovered);

            ImVec2               viewportPanelSize               =
ImGui::GetContentRegionAvail();
            m_ViewportSize      =      {      viewportPanelSize.x,
viewportPanelSize.y };
            uint32_t       textureID       =       m_FrameBuffer-
>GetColorAttachmentRendererID();
```

```
          Entity                 selectedEntity                 =
m_ObjPannel.GetSelectedEntity();

          ImGui::Image((void*)textureID,                  ImVec2{
m_ViewportSize.x, m_ViewportSize.y }, ImVec2{ 0, 1 }, ImVec2{ 1,
0 });

          if (ImGui::BeginDragDropTarget())
          {
               if     (const     ImGuiPayload*     payload     =
ImGui::AcceptDragDropPayload("CONTENT_BROWSER_PAYLOAD"))
               {
                    const     wchar_t*     path     =     (const
wchar_t*)payload->Data;

     OpenScene(std::filesystem::path(g_AssetsDirectory) / path);
               }

               ImGui::EndDragDropTarget();
          }


          //-----------Guizmo-----------------------------------
----------------
          if (selectedEntity && *m_GizmoType != -1)
          {
               ImGuizmo::SetOrthographic(false);
               ImGuizmo::SetDrawlist();

               ImGuizmo::SetRect(m_ViewportBounds[0].x,
m_ViewportBounds[0].y,          m_ViewportBounds[1].x          -
m_ViewportBounds[0].x,          m_ViewportBounds[1].y          -
m_ViewportBounds[0].y);


               auto mainCameraEntt = m_EditorCamera;
               const     glm::mat4&     cameraProjection     =
m_EditorCamera.GetProjectionMatrix();
               glm::mat4              cameraView              =
m_EditorCamera.GetViewMatrix();

               auto&         entityTransformsComponent         =
selectedEntity.GetComponent<TransformComponent>();
               glm::mat4               tranforms                =
entityTransformsComponent.GetTransform();

               bool                    snap                     =
Input::IsKeyPressed(RV_KEY_LEFT_CONTROL);
               float snapValue = 0.5f;
               if (*m_GizmoType == ImGuizmo::OPERATION::ROTATE)
                    snapValue = 45.0f;
```

```
                float snapValues[3] = { snapValue, snapValue,
snapValue };


                ImGuizmo::Manipulate(glm::value_ptr(cameraView),
glm::value_ptr(cameraProjection),
(ImGuizmo::OPERATION)*m_GizmoType,              ImGuizmo::LOCAL,
glm::value_ptr(tranforms), nullptr, snap ? snapValues : nullptr);

                if             (ImGuizmo::IsUsing()             &&
!Input::IsKeyPressed(RV_KEY_LEFT_ALT))
                {
                     glm::vec3 translation, rotation, scale;
                     DecomposeTransform(tranforms,   translation,
rotation, scale);

                     glm::vec3   deltaRotation   =   rotation   -
entityTransformsComponent.Rotation;
                     entityTransformsComponent.Position         =
translation;
                     entityTransformsComponent.Rotation        +=
deltaRotation;
                     entityTransformsComponent.Scale = scale;
                }
          }


          m_ToolBar.OnImGuiRender();

          ImGui::End();
          ImGui::PopStyleVar();

          ImGui::End();
     }

     void CraftLayer::OnEvent(Revoke::Event& e)
     {
          m_EditorCamera.OnEvent(e);

          EventDispatcher dispatcher(e);

     dispatcher.Dispatch<KeyPressedEvent>(RV_BIND_EVENT_FUNK(Cra
ftLayer::OnKeyPressed));

     dispatcher.Dispatch<MouseButtonPressedEvent>(RV_BIND_EVENT_
FUNK(CraftLayer::OnMouseBtnPressed));
     }
     bool CraftLayer::OnMouseBtnPressed(MouseButtonPressedEvent&
e)
     {
          if   (e.GetMouseButton()   ==   RV_MOUSE_BUTTON_1   &&
!Input::IsKeyPressed(RV_KEY_LEFT_ALT))
          {
```

```
                if (m_ViewportHovered && !ImGuizmo::IsOver())
                {

    m_ObjPannel.SetSelectedEntity(m_HoveredEntity);
                }
                return true;
           }
           return false;
     }
     bool CraftLayer::OnKeyPressed(KeyPressedEvent& e)
     {
           switch (e.GetKeyCode())
           {
           case RV_KEY_S:
           {
                if  (Input::IsKeyPressed(RV_KEY_LEFT_CONTROL)  &&
Input::IsKeyPressed(RV_KEY_LEFT_SHIFT))
                {
                     SaveAs();
                }
                else                                          if
(Input::IsKeyPressed(RV_KEY_LEFT_CONTROL))
                {
                     Save();
                }
                break;
           }
           case RV_KEY_N:
           {
                if (Input::IsKeyPressed(RV_KEY_LEFT_CONTROL))
                {
                     NewScene();
                }
                break;
           }
           case RV_KEY_O:
           {
                if (Input::IsKeyPressed(RV_KEY_LEFT_CONTROL))
                {
                     OpenScene();
                }
                break;
           }
           case RV_KEY_Q:
           {
                *m_GizmoType = -1;
                break;
           }
           case RV_KEY_W:
           {
                *m_GizmoType = ImGuizmo::OPERATION::TRANSLATE;
                break;
           }
```

```
            case RV_KEY_E:
            {
                *m_GizmoType = ImGuizmo::OPERATION::ROTATE;
                break;
            }
            case RV_KEY_R:
            {
                *m_GizmoType = ImGuizmo::OPERATION::SCALE;
                break;
            }
            }
            return false;
        }

        //-----------------------------------------------------------
---------------
        void CraftLayer::NewScene()
        {
            m_Scene = std::make_shared<Scene>("New scene");
            m_Scene->OnSceneClose();
            m_Scene->OnViewportResize((uint32_t)m_ViewportSize.x,
(uint32_t)m_ViewportSize.y);
            m_ObjPannel.SetScene(m_Scene);
            m_ToolBar.SetScene(m_Scene);
            m_ProjectSettingsPanel.SetScene(m_Scene);
        }

        void CraftLayer::OpenScene()
        {
            std::string path = FileExplorer::OpenFile("MyRevoke
Scene (*.myrevoke)\0*.myrevoke\0");

            if (!path.empty())
            {
                OpenScene(path);
            }
        }
        void  CraftLayer::OpenScene(const  std::filesystem::path&
path)
        {
            if (m_ToolBar.GetSceneState() == SceneState::Runtime)
            {
                m_ToolBar.OnSceneStop();
            }
            m_Scene = std::make_shared<Scene>();
            m_Scene->OnSceneClose();
            m_Scene->OnViewportResize((uint32_t)m_ViewportSize.x,
(uint32_t)m_ViewportSize.y);

            Serealizer sceneSerealizer(m_Scene);
            sceneSerealizer.DeSerealize(path.string());

            m_ObjPannel.SetScene(m_Scene);
```

```
            m_ToolBar.SetScene(m_Scene);
            m_ProjectSettingsPanel.SetScene(m_Scene);

      }

      void CraftLayer::SaveAs()
      {
            std::string path = FileExplorer::SaveFile("Hazel Scene
(*.myrevoke)\0*.myrevoke\0");

            if (!path.empty())
            {
                  Serealizer sceneSerealizer(m_Scene);
                  sceneSerealizer.Serealize(path+".myrevoke");
            }
      }

      void CraftLayer::Save()
      {
            Serealizer sceneSerealizer(m_Scene);
            sceneSerealizer.Serealize("assets/Scenes/"  +  m_Scene-
>GetName() + ".myrevoke");
      }

}
```