МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Ігровий рушій для створення інді-ігор з використанням OpenGL**

Виконав: студент 4 курсу, групи 6.1219-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
Д. С. Воробець
(ініціали та прізвище)
Керівник доцент, к.т.н., доцент В. І. Заяц
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т.В. Критська_
"_01_" ___березня__ 2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Воробцю Дмитру Сергійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи Ігровий рушій для створення інді-ігор з використанням OpenGL

керівник роботи ___Заяц Валерій Іванович, доцент, к.т.н._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 29.12.2022 №1893-с

2. Строк подання студентом кваліфікаційної роботи_____14.06.2023_____
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми збору та аналізу даних з веб-сторінок;
- створення програмного продукту та його опис;
- дослідження поставленої проблеми та розробка висновків.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз предметної області	15.03.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	17.03.23	виконано
3	Аналіз існуючих методів рішення	18.03.23	виконано
4	Дослідження засобів реалізації ігрового рушія	19.03-25.03.23	виконано
5	Дослідження засобів реалізації сайту під ігровий рушій	26.03.23	виконано
6	Узгодження подальших дій з науковим керівником	30.03.23	виконано
7	Проектування архітектури ігрового рушія	01.04-15.04.23	виконано
8	Програмна реалізація ігрового рушія	16.04-15.05.23	виконано
9	Проектування архітектури сайту під ігровий рушій	16.05-17.05.23	виконано
10	Програмна реалізація сайту під ігровий рушій	18.05-24.05.23	виконано
11	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	25.05.23	виконано
12	Реалізація користувацького інтерфейсу ігрового рушія	26.05-30.05.23	виконано
13	Реалізація користувацького інтерфейсу сайту під ігровий рушій	31.05-01.06.23	виконано
11	Перевірка роботоздатності проектів	02.06-04.06.23	виконано
12	Оформлення звіту	05.06-12.06.23	виконано
13	Оформлення презентації. Отримання рецензій від опонентів.	13.06-15.06.23	виконано

Студент _____ Воробець Д.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Заяц В.І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 93

Рисунків — 16

Джерел — 15

Воробець Д.С. Ігровий рушій для розробки інді ігор на основі OpenGL: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник В. І. Заяц. Запоріжжя : ЗНУ, 2023.

Мета полягає у створенні зручного та швидкого застосунку, який дозволить розробникам швидко створювати відео-ігри, без необхідності написання всього коду з нуля, на операційній системі Windows.

У процесі розробки була розглянута проблема побудови ігрових додатків, використовуючи зручний інструментарій. У результаті був розроблений застосунок TWE, створений на основі графічного API OpenGL. Даний застосунок містить в собі 2 головних шари (рушій та редактор) та 1 зв’язуючий шар(контроллер). Рушій відповідає за усю логіку малювання, взаємодії сутностей, відігрівання звуків, взаємодії скриптів. Редактор надає зручний інструментарій, у вигляді графічного інтерфейсу, який допомагає швидко та без труднощів створювати ігрові додатки. Окрім цього, був розроблений веб сайт, на основі платформи Node JS, для просування даного продукту у сфері розробці ігор. Використовуючи сайт можна встановити додаток TWE або переглянути новини про нього.

Ключові слова*: ігровий рушій, інді гра, редактор, малювання, взаємодія сутностей, взаємодія скриптів, OpenGL, Node JS.*

ABSTRACT

Pages — 93

Drawings — 16

Sources — 15

Vorobets D.S. Game engine for the development of indie games based on OpenGL: bachelor's thesis in specialty 121 "Software Engineering" / Science. manager V.I. Zayats. Zaporizhzhia: ZNU, 2023.

The goal is to create a convenient and fast application that will allow developers to quickly create video games, without the need to write all the code from scratch, on the Windows operating system.

During the development process, was considered the problem of building gaming applications using convenient tools. As a result, was developed the TWE application based on the OpenGL graphics API. This application contains 2 main layers (engine and editor) and 1 binding layer (controller). The engine is responsible for all the logic of drawing, interaction of entities, sound warming, and script interaction. The editor provides a convenient toolkit in the form of a graphical interface that helps to create game applications quickly and easily. In addition, a website based on the Node JS platform was developed to promote this product in the field of game development. Using the website, you can install the TWE application or view news about it.

Keywords*: game engine, indie game, editor, drawing, entity interaction, script interaction, OpenGL, Node JS.*

ЗМІСТ

ВСТУП

Актуальність теми

Створення власної відео гри є складним завданням, яке вимагає багато часу та зусиль. Однак, завдяки зростанню популярності інді-ігор, все більше розробників намагається створити свою власну гру. Проте, однією з найбільших проблем, з якими стикаються розробники інді-ігор, є відсутність потужного ігрового рушія, який би відповідав їх потребам та вимогам. Більшість готових ігрових рушіїв мають високу вартість або обмеження використання, що робить їх недосяжними для більшості розробників інді-ігор. Актуальність проблеми створення потужного ігрового рушія для інді-ігор полягає в тому, що розробники інді-ігор потребують доступного та ефективного рішення для створення своїх ігор. Ігровий рушій, який би відповідав їх потребам, дозволив би зробити процес розробки ігор більш доступним та простим, що збільшить кількість якісних ігор на ринку та забезпечить конкурентну перевагу для розробників інді-ігор.

Для інді-розробників, які мають обмежений бюджет та команду, створення власного ігрового рушія може стати складним завданням [1]. Однак, на сьогоднішній день, існує багато інструментів та технологій, які можуть допомогти у розробці власного ігрового рушія. Основні проблеми, з якими стикаються інді-розробники при створенні власного ігрового рушія, включають.

1. Високі витрати на розробку. Розробка ігрового рушія може бути дорогим та складним процесом. Для інді-розробників, які працюють з обмеженим бюджетом та ресурсами, створення власного ігрового рушія може бути недосяжним.
2. Відсутність досвіду. Створення власного ігрового рушія вимагає значного досвіду у програмуванні та розробці ігрової механіки. Для розробників, які тільки починають свій шлях у галузі розробки ігор, це може бути викликом.

3. Відсутність знань про архітектуру. Створення власного ігрового рушія вимагає розуміння архітектури гри та принципів її функціонування. Це може бути складним завданням для розробників, які не мають досвіду у цій області.

Створення ігрового рушія для інді ігор є актуальною проблемою в індустрії геймдеву(розробки ігор). Існує багато причин, які підтверджують цю актуальність.

По-перше, зростання популярності інді-ігор в останні роки [2]. Інді-розробники створюють ігри на різноманітні теми та жанри, що відрізняються від традиційних ігор великих видавництв. Інді-ігри можуть бути успішними та навіть вигравати нагороди, які раніше були зарезервовані для великих видавництв.

По-друге, створення власного ігрового рушія дозволяє інді-розробникам мати повний контроль над своїми іграми. Вони можуть внести свої унікальні ідеї та функції у свій рушій, що може допомогти їм вирізнятися на ринку.

По-третє, створення власного ігрового рушія може допомогти інді-розробникам зменшити витрати на розробку ігор. Іноді використання сторонніх рушіїв може бути дорожчим та не давати повного контролю над процесом розробки.

По-четверте, створення власного ігрового рушія може допомогти інді-розробникам зберегти час та зосередитися на важливих аспектах своїх ігор. Вони можуть використовувати свій рушій для створення різних ігор на різні теми та жанри, що дозволяє їм ефективніше використовувати свій час.

Створення ігрового рушія може допомогти інді-розробникам створювати більш якісні та креативні ігри з меншими витратами та більшим контролем. Крім того, розробка власного ігрового рушія може допомогти інді-розробникам відрізнятися від конкурентів на ринку, що може збільшити їхню популярність та дохід. Ще однією причиною, чому створення ігрового рушія є актуальною проблемою для інді-розробників, є те, що багато

сторонніх ігрових рушіїв, таких як Unity, Unreal Engine та CryEngine, можуть бути досить складними для вивчення та використання для початківців. Створення власного рушія може допомогти інді-розробникам зменшити ступінь складності та дозволити їм більш ефективно використовувати свій час.

Загалом, створення ігрового рушія є актуальною проблемою для інді-розробників, які бажають створювати якісні та креативні ігри з меншими витратами та більшим контролем. Інді-розробники можуть використовувати свій рушій для створення різних ігор на різні теми та жанри, що дозволяє їм ефективніше використовувати свій час та ресурси.

Мета дослідження

Створення простого інструментарію, який дозволяє інді-розробникам створювати високоякісні та захоплюючі ігри.

Завдання дослідження

Аналіз існуючих рішень щодо створення ігор. Вивчення архітектури OpenGL та його основних функціональних можливостей з метою розуміння принципів роботи та використання їх у створенні ігрового рушія. Розробка ефективної архітектури ігрового рушія, яка дозволить забезпечити оптимальну продуктивність та гнучкість в розробці різноманітних ігор.

Об’єкт дослідження

Об'єктом дослідження є процес розробки ігрового рушія для інді-ігор на основі OpenGL.

Предмет дослідження

Предметом дослідження є вивчення основних принципів роботи з OpenGL, аналіз сучасних тенденцій у розробці інді-ігор та ігрових рушей, для врахування актуальних вимог до розроблюваного рушія.

Методи дослідження

Проведення детального огляду літературних джерел, наукових публікацій, статей, підручників та інших джерел, що стосуються технологій розробки ігрових рушіїв, особливостей OpenGL; вивчення та аналіз популярних ігрових рушіїв, які використовують OpenGL, з метою виявлення їх особливостей, сильних і слабких сторін, а також накопичення досвіду розробки ігрових рушіїв; апробація результатів.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у наданні розробникам інструментарію, що спростить та прискорить процес розробки ігор. Забезпечення готових модулів для графіки, аудіо та управління ресурсами дозволить зосередитися на самому контенті гри, зменшуючи час та зусилля, потрібні для вирішення технічних аспектів. Результати дослідження можуть сприяти обміну знаннями та досвідом серед інді-ігрової спільноти.

Глосарій

Інді ігри (англ. Indie games) — це невеличкі ігри, які створюються маленькими компаніями (5-10 чоловік) без фінансової підтримки великих видавництв та компаній.

OpenGL — це графічна бібліотека, яка використовується для створення візуалізації 2D та 3D графіки.

ImGUI (Immediate Mode Graphical User Interface) — це підхід до створення графічного інтерфейсу користувача, де кожен кадр зображення обробляється безпосередньо, без необхідності використання структури даних для зберігання інформації про інтерфейс.

CUDA Toolkit (Compute Unified Device Architecture) — це пакет розробки програмного забезпечення (SDK) від компанії NVIDIA, який надає

набір інструментів, бібліотек і ресурсів для розробки програм, які використовують графічний процесор (GPU) NVIDIA з технологією CUDA.

Bullet Physics Library — це відкрите програмне забезпечення, яке надає набір інструментів для симуляції фізики твердих тіл.

CMake — це крос-платформний системний інструмент для автоматизації процесу збирання (build) програмного забезпечення. Він дозволяє розробникам визначати та керувати процесом компіляції, лінкування та упаковки програми незалежно від конкретної платформи або компілятора.

VAO (Vertex Array Object) — це об'єкт у контексті графічного програмування, який використовується для зберігання конфігурації вершинного масиву, таку як вершинні дані, вказівники на атрибути вершин.

VBO (Vertex Buffer Object) — це об'єкт у контексті графічного програмування, який використовується для зберігання вершинних даних, таких як координати вершин, нормалі, текстурні координати.

EBO (Element Buffer Object) — це об'єкт у контексті графічного програмування, який використовується для зберігання індексів вершин, які використовуються для візуалізації графічних об'єктів.

FBO (Frame Buffer Object) — це об'єкт у контексті графічного програмування, який дозволяє створювати та керувати власними буферами кадрів.

Шейдер (англ. Shader) — це програмний код, який використовується для керування відображенням графічних об'єктів.

Mesh — це структура даних у комп'ютерній графіці, що використовується для опису геометричних об'єктів у тривимірному просторі. Він складається з набору вершин і зв'язаних з ними граней, ребер або трікутників.

DLL (Dynamic Link Library) — це тип файлу, який містить код і дані, які використовуються програмами під час виконання.

Ропир тепи (спливаюче меню) — це елемент інтерфейсу користувача, який зазвичай з'являється при натисканні правою кнопкою миші і відображає список опцій або команд.

Endpoint — це позначення кінцевої точки або адреси, за допомогою якої клієнтська програма може взаємодіяти з веб-службою, API або іншою системою.

Pagination — це процес розподілу великого набору даних на окремі сторінки з метою показу їх поетапно або посторінково.

Thunks — це обгортка навколо дії, яка може приймати додаткові параметри та виконувати асинхронні операції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

У світі інді-ігор, де творчість та інновації є основними стовпами розвитку, розробка ефективного ігрового рушія на основі OpenGL стає особливо актуальною. При підготовці дипломної роботи, метою є огляд літературних джерел, що допоможе розширити розуміння цієї теми та виявити розробки, що вже існують у галузі ігрових рушіїв.

Велика увага приділяється збору і аналізу літературних джерел, які становлять фундаментальну основу нашої роботи. Методологія включає використання баз даних, онлайн-бібліотек та академічних журналів. Під час відбору літературних джерел я керуюсь критеріями актуальності, наукового рівня та авторитетності авторів.

Відкриті літературні джерела піддані детальному аналізу, зокрема зосередженням на основних темах та розробках у галузі ігрових рушіїв на основі OpenGL. Різні підходи до розробки ігрових рушіїв розглядаються з урахуванням їхнього впливу на ефективність інді-ігор.

Серед основних тем, виявлених під час аналізу літературних джерел, варто зазначити наступні. По-перше, опис технології OpenGL та її потенціалу для розробки ігрових рушіїв [15]. По-друге, дослідження різних методів оптимізації графічних обчислень для забезпечення плавності та швидкодії ігрового процесу. По-третє, вивчення архітектури рушія та підходів до розробки модульної системи компонентів.

Важливість дослідження та розробки ігрового рушія на основі OpenGL підкреслюється, оскільки він стане основою для створення вражаючих інді-ігор. Додатково, вказується на можливості подальшого дослідження в галузі ігрових рушіїв та їх використання для досягнення нових вершин у сфері інді-ігор.

Створення ігрового рушія для інді-ігор може бути складною задачею, особливо для тих, хто тільки починає свій шлях у галузі GameDev . Але маючи свій власний рушій ви можете мати не тільки більше контролю над розробкою ігор, але і створити свій власний бренд та визначити його унікальність. Тому зараз, та мабуть у найближчому майбутньому, буде актуальною тема створення власного ігрового рушія [14].

Однією з найбільших проблем при створенні ігрового рушія для інді ігор є брак ресурсів та обмежений доступ до інформації про технології, що використовуються в галузі [13]. Інді розробники зазвичай працюють з обмеженим бюджетом та не мають доступу до ресурсів та знань, які є доступні для великих студій.

Розробка ігрового рушія для інді-ігор зазвичай зустрічається з кількома викликами. Перш за все, це складність процесу розробки. Особливо для одиночних розробників або невеликих команд, створення ефективного ігрового рушія потребує значних зусиль, часу, знань і технічної експертизи.

Друга проблема пов'язана з вимогами до продуктивності. Щоб забезпечити плавний геймплей і уникнути затримок, ігровий рушій повинен мати високу продуктивність [14]. Це може вимагати великих інвестицій у комп'ютерну техніку і програмне забезпечення, а також освіти з оптимізації коду.

Третя проблема, з якою стикаються інді-розробники, - недостатня підтримка від виробників ігрових рушіїв. Це може ускладнювати вирішення проблем, що виникають під час розробки, і уповільнювати процес.

Нарешті, нестабільність ринку ігор є ще одним викликом. Розробники ігрових рушіїв повинні забезпечити гнучкість свого продукту, щоб відповідати змінам і потребам ринку, який є непередбачуваним.

Але створення ігрового рушія для інді-ігор надає такі переваги:

1. Зниження витрат на розробку: Інді-розробники можуть скористатися готовим ігровим рушієм, що дозволяє їм зосередитися на створенні

вмісту гри, не витрачаючи час на розробку власного рушія. Це може допомогти знизити загальні витрати на розробку гри.

2. Зменшення часу до випуску.
3. Підтримка розробників: Готові ігрові рушії часто мають активну спільноту розробників, яка може допомогти з вирішенням проблем та наданням підтримки. Розробники можуть знайти відповіді на свої питання або звернутися за допомогою до інших розробників.
4. Можливість прискорити розробку.

1.2 Аналіз програмного забезпечення

1.2.1 Unreal Engine

Unreal Engine — це ігровий рушій, розроблений і підтримуваний компанією Epic Games. Він використовується для створення ігор, а також для розробки віртуальної реальності, архітектурних візуалізацій, відео та інших інтерактивних додатків [8].

Unreal Engine містить широкий набір інструментів для розробки ігор, включаючи графічний рушій, фізичний рушій, систему штучного інтелекту, систему звуку та інші компоненти, що дозволяють розробникам створювати різноманітні ігри з різними механіками та геймплеєм. Рушій має потужну графічну підсистему, що дозволяє створювати візуально захоплюючі ігри з високою якістю графіки та реалістичним освітленням.

Unreal Engine використовує мову програмування C++, а також мову скриптування Blueprint. Мова Blueprint дозволяє розробникам створювати складні ігрові системи та поведінку об'єктів без написання коду, що полегшує процес розробки для новачків. На Рис.*1* 1 зображено графічний інтерфейс ігрового рушія Unreal Engine.

Рис.*1 Графічний інтерфейс ігрового рушія Unreal Engine*

Рушій має вбудовану підтримку для розробки віртуальної реальності та 360-градусних відео, що дозволяє створювати вражаючі візуальні досвіди для VR-гарнітур та інших пристроїв.

Unreal Engine також має потужну систему редактора, що дозволяє розробникам створювати та редагувати ігровий контент без необхідності виходити з середовища розробки. Редактор містить різні інструменти для створення та редагування графіки, анімації, звуку та іншого контенту.

Використовуючи Unreal Engine було створено багато хороших та відомих ігор, як Fortnite, Gears of War, Borderlands, Batman: Arkham Knight та багато інших. Рушій має досить велику спільноту розробників, що забезпечує стабільну підтримку та постійне оновлення інструментів для розробки.

Unreal Engine також має вбудовану систему мультиплатформенності, що дозволяє розробляти ігри для різних платформ, таких як Windows, Mac, Linux, iOS, Android, PlayStation, Xbox та інші. Це дає розробникам можливість створювати одну версію гри та розгортати її на різних платформах без необхідності переписування коду.

Для Unreal Engine доступний великий набір сторонніх бібліотек та плагінів, що дозволяють розширювати функціональність движка та додавати нові можливості до проекту. Наприклад, для Unreal Engine існує велика

кількість плагінів для створення ігор в жанрі RPG, використання системи фізики Havok, підтримки VR-гарнітур та багато інших.

Unreal Engine також має велику спільноту користувачів та розробників, що дозволяє знайти відповіді на будь-які технічні питання та проблеми, які можуть виникнути під час розробки. Спільнота активно підтримує різні проекти, які дають можливість розробникам співпрацювати та обмінюватись знаннями та досвідом.

Узагальнюючи, Unreal Engine є потужним інструментом для розробки ігор та інтерактивних додатків з високою якістю графіки та широким спектром функціональних можливостей. Його велика спільнота розробників та підтримка з боку Epic Games роблять Unreal Engine одним з найпопулярніших ігрових движків на ринку.

1.2.2 Unity

Unity — це популярний ігровий рушій, що використовується для створення ігор та інтерактивних додатків. Цей рушій розробляється компанією Unity Technologies засновником якої є Девід Хелсбі.

Unity надає зручний інтерфейс для розробки, що дозволяє розробникам легко створювати 2D та 3D ігри [9], додавати в них різноманітні об'єкти, ефекти та анімацію, налаштовувати фізику та поведінку об'єктів, розгортати гру на різних платформах та багато іншого.

Особливістю Unity є велика кількість сторонніх ресурсів, таких як моделі персонажів, музика, звуки, графічні ефекти тощо, які можна безкоштовно завантажити з інтернету та використовувати у своїх проектах. Unity також має велику спільноту користувачів та розробників, яка допомагає вирішувати технічні питання та надає поради щодо розробки ігор. На Рис.*2* 2 зображено графічний інтерфейс ігрового рушія Unity.

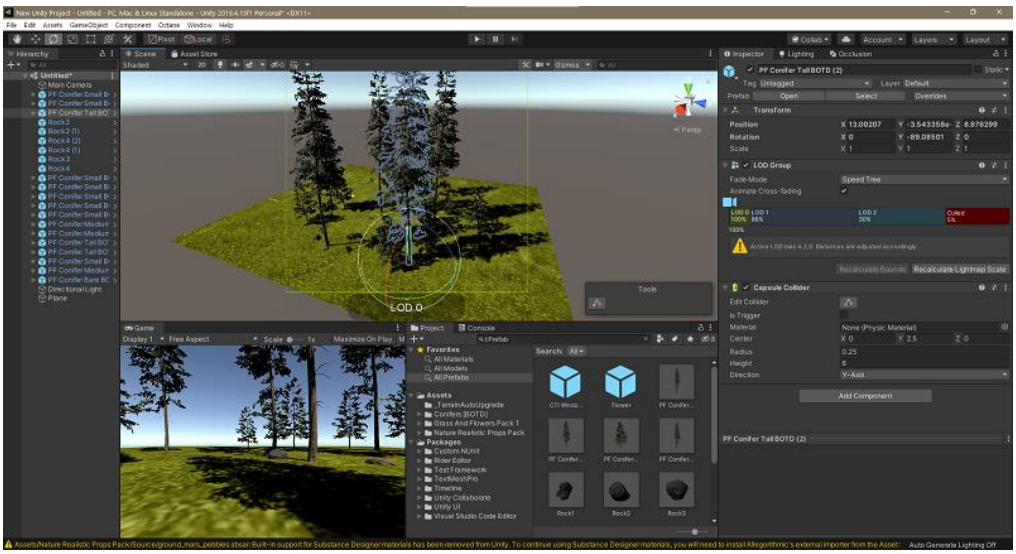

Рис.*2 Графічний інтерфейс ігрового рушія Unity*

Для Unity доступний мовний стек C# та JavaScript, які є популярними у світі програмування. Це дозволяє розробникам з різним рівнем досвіду легко зрозуміти та змінювати код гри.

Unity також надає можливість розробникам робити взаємодію зі сторонніми програмами та бібліотеками, що дозволяє розширювати можливості гри. Крім того, Unity підтримує віртуальну та доповнену реальність, що дозволяє створювати ігри з використанням VR та AR технологій.

Рушій Unity підтримує багато різних платформ та мов програмування, що дозволяє розробникам створювати ігри на різних платформах та використовувати мову програмування, з якою вони знайомі. Крім того, в Unity є велика кількість готових компонентів та ресурсів, що дозволяє розробникам економити час та зусилля на створенні власних елементів гри.

Unity також має багато інструментів для тестування та налагодження гри, що дозволяє розробникам відстежувати та вирішувати проблеми з грою на ранніх стадіях розробки. Крім того, Unity має вбудовані засоби аналізу гри та збору даних, які допомагають розробникам зрозуміти поведінку гравців та покращити гру на основі цих даних.

Однією з головних переваг Unity є те, що він має велику та активну спільноту розробників, яка розробляє різні додаткові інструменти та ресурси для Unity. Це дозволяє розробникам ігор використовувати готові рішення для різних задач, що значно спрощує та прискорює процес розробки ігор.

Unity також має велику кількість вбудованих функцій для роботи з 2D та 3D графікою,звуком, фізикою, штучним інтелектом та іншими аспектами гри. Це дозволяє розробникам не витрачати час на написання власних рішень для цих завдань, а зосередитися на більш складних та цікавих аспектах розробки ігор.

Ще одна важлива перевага Unity полягає в тому, що він підтримує різні платформи, включаючи мобільні пристрої, комп'ютери, консолі та віртуальну реальність. Це дозволяє розробникам легко портувати свої ігри на різні платформи та максимально розширювати свою аудиторію.

Загалом, Unity є потужним та гнучким інструментом для розробки ігор, який дозволяє розробникам швидко та ефективно створювати якісні ігри різних жанрів та для різних платформ.

1.2.3 CryEngine

CryEngine — це ігровий рушій від Crytek GmbH, який використовується для розробки відеоігор на різних платформах, включаючи ПК, консолі та мобільні пристрої.

CryEngine був розроблений спочатку для відеоігор серії Crysis, але згодом став доступним для загального використання. Він володіє великими можливостями в роботі з графікою, фізикою та штучним інтелектом. Рушій використовує технологію рельєфного мапування (height mapping) та технологію затінення (soft shadows), що дозволяє створювати реалістичну та деталізовану графіку з високою деталізацією [10].

Однією з особливостей CryEngine є динамічне середовище змін, яке дозволяє розробникам створювати ігрові світи зі змінними умовами, такими

як денна та нічна зміна, зміна погодних умов, сезонні зміни та інші. На Рис.*3* 3 зображено графічний інтерфейс ігрового рушія CryEngine.

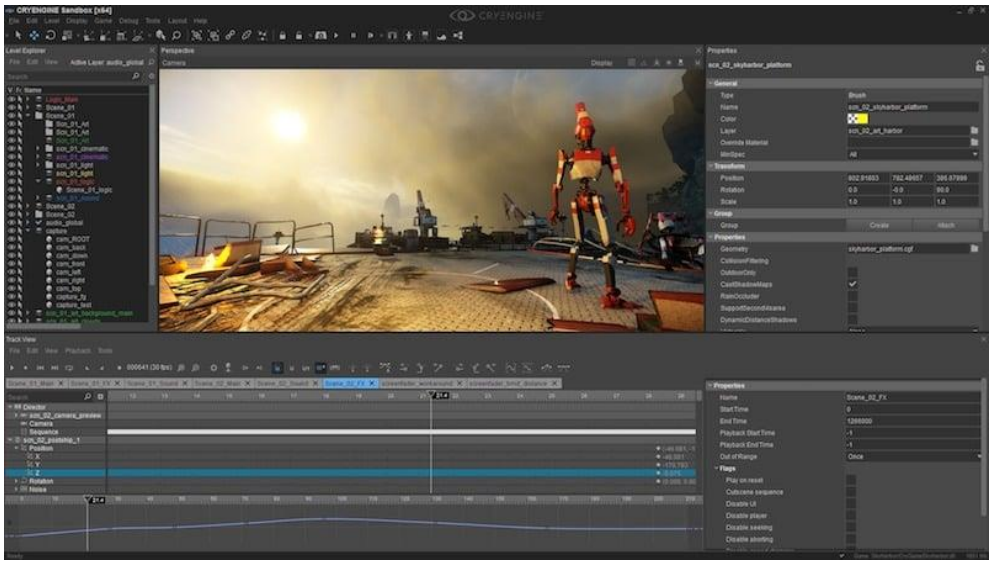

Рис.*3 Графічний інтерфейс ігрового рушія CryEngine*

CryEngine також має вбудовану підтримку віртуальної реальності та може працювати з різними віртуальними пристроями.

CryEngine також має велику спільноту розробників, яка надає підтримку, ресурси та інформацію для розробки ігор. Більшість ресурсів, таких як 3D-моделі, текстури та ефекти, можуть бути взяті з CryEngine Marketplace, що спрощує розробку ігор.

Недоліками CryEngine є складність використання та відносна висока вартість ліцензії порівняно з деякими іншими ігровими рушіями. Також розробка ігор з використанням CryEngine потребує більш високих технічних знань та досвіду, ніж у деяких інших інструментів.

Крім того, CryEngine має такі переваги:

1. Висока якість графіки: CryEngine відомий своєю високоякісною графікою та спеціальними ефектами, такими як затінення, туман, водні ефекти та інші.

2. Можливості модифікації: CryEngine має великі можливості для модифікації ігор, що дозволяє розробникам налаштувати свої проекти настільки, наскільки це потрібно.
3. Підтримка мультиплеєру: CryEngine має вбудовану підтримку мультиплеєру, що дозволяє розробникам створювати ігри з можливістю гри в мережі.

У загальному, CryEngine — це потужний та високоякісний двигун для створення ігор з вражаючою графікою та можливостями модифікації. Однак, він має певні недоліки, такі як високі вимоги до апаратного забезпечення та складність навчання. Цей двигун може бути корисним для розробників, які мають достатній досвід у галузі геймдеву та мають великі амбіції щодо створення ігор високої якості. З урахуванням його переваг та недоліків, розробники можуть прийняти обгрунтоване рішення щодо використання цього двигуна у своєму проекті.

1.3 Постановка завдання

Мета кваліфікаційної роботи — це створення ігрового рушія для створення інді ігор на основі OpenGL. Постановка завдання дослідження включає наступні етапи:

1. Аналіз потреб і вимог: Визначення основних потреб та вимог, які повинен задовольняти розроблений ігровий рушій. Це включає підтримку графіки 2D та 3D, обробку аудіо, фізику, управління ресурсами, оптимізацію продуктивності.
2. Вибір технологій: Вибір технологічного стеку, який найкраще відповідає потребам дослідження. У даному випадку, вибір технології OpenGL як основи для ігрового рушія.
3. Розробка архітектури: Визначення архітектури ігрового рушія, включаючи модулі для графіки, фізики, аудіо, скриптингу, графічного

інтерфейсу. Врахування принципів гнучкості, розширюваності та ефективності рушія.

4. Розробка функціональності: Реалізація основних функціональних можливостей ігрового рушія, таких як обробка графіки, аудіо, взаємодія з користувачем та управління ресурсами. Врахування принципів оптимізації продуктивності.
5. Апробація: Виконання ряду перевірок для переконання в коректності роботи розробленого ігрового рушія. Валідація рушія на основі визначених вимог та порівняння результатів з іншими існуючими ігровими рушіями.

Також для просування ігрового рушія потрібно створити сайт, що допоможе майбутнім розробникам ознайомитись з системою.

1.4 Висновки до розділу 1

1. Було оглянуто головні проблеми створення ігрових рушіїв: складність розробки, вимоги до продуктивності, недостатня підтримка, нестабільність ринку.
2. Проаналізовані конкурентні ігрові рушії: Unreal Engine, Unity, CryEngine. Були визначені їх плюси та мінуси.
3. Визначена постановка завдання дослідження.

2 ВИМОГИ ДО ОТОЧЕННЯ

2.1 Аналіз сучасних технологій для розробки ігрових рушіїв

Ігровий рушій є програмною платформою, що дозволяє розробникам створювати комп'ютерні ігри. Він надає набір інструментів для розробки графіки, звуку, фізики, інтерфейсу користувача та інших компонентів, що використовуються в іграх. Так як з початку розвитку відео-ігрового жанру пройшло вже багато років, за цей час було розроблено багато зручних технологій, які можуть допомогти розробити рушій. Тому за основу свого ігрового рушія було обрано саме такі технології:

Мова програмування C++: забезпечує можливість ефективної роботи зі швидкодіючими алгоритмами та структурами даних, що дозволяє досягти високої продуктивності ігрового рушія.

Графічний рушій (Graphics Engine): Це програмне забезпечення, що відповідає за відображення графіки на екрані комп'ютера або іншого пристрою. Воно здійснює обробку графічної інформації та управління графічними ресурсами, такими як текстури, моделі, освітлення та інше.

Одним з головних принципів роботи графічниї рушіїв є робота з кадрами (frames) [3]. Кожен кадр — це одне повне відтворення гри, яке відбувається протягом певного періоду час.

За основу мого рушія було обрано таку графічну бібліотеку як “OpenGL”, так як вона є стандартом у галузі комп'ютерної графіки та використовується для розробки графічних додатків у багатьох галузях, таких як відеоігри, візуалізація даних, комп'ютерна архітектура та інші. OpenGL забезпечує доступ до низькорівневих графічних функцій та можливостей апаратного прискорення, що дозволяє створювати складні графічні сцени з високою швидкістю рендерингу [4].

Фізичний рушій (Physics Engine): Це програмне забезпечення, що відповідає за зміну положення тіла відносно іншого тіла чи системи тіл в

часі. Це може бути рух вздовж прямої лінії (лінійний рух) або вздовж кривої (криволінійний рух), а також обертальний рух. Рух може бути одновимірним (наприклад, рух по вулиці), двовимірним (рух у площині) або тривимірним (рух у просторі).

Розробка фізичного руху є складною задачею, оскільки вона пов'язана зі складною взаємодією тіл, різними фізичними законами, ефектами, що впливають на рух тіла. Для опису фізичного руху використовуються математичні методи та формули, а також різні види моделей, що дозволяють апроксимувати реальні фізичні процеси.

За основу мого рушія було обрано фізичний рушій “Bullet Physics Library” [5], а точніше “Bullet3”, так як він випущений під ліцензією zlib/libpng, що дозволяє використовувати її у комерційних та не-комерційних проектах безкоштовно. Ще великий плюс у тому, що Bullet Physics Library використовується у багатьох відомих відеоіграх, а також в різних проектах віртуальної реальності та симуляції фізичних процесів. Вона є потужним інструментом для розробки ігор та додатків з фізичними ефектами, які відтворюють реальні фізичні явища. Також для компіляції бібліотеки було використано CUDA Toolkit.

Звуковий рушій (Audio Engine): Це програмне забезпечення, що відповідає за відтворення звукових ефектів та музики в іграх з метою покращення геймплею та забезпечення більш іммерсивного досвіду гравця.

Звуковий рушій для ігор необхідний для того, щоб створити реалістичний атмосферний звуковий фон, який допомагає гравцеві краще зануритися у гру. Звукові ефекти також можуть бути корисними для сповіщення гравця про важливі події в грі, наприклад, про те, що його персонаж отримав ураження або виконав якусь місію. Крім того, звукові рушії можуть допомогти покращити комунікацію між гравцями, що може бути корисно в онлайн-іграх, де гравці з усього світу можуть грати разом.

За основу мого рушія було обрано бібліотеку “irrKlang”, тому що вона забезпечує різноманітні можливості для обробки звукових ефектів, таких як

еквалайзер, підсилювання звуку та інші ефекти [6]. Також, цей звуковий рушій дозволяє здійснювати роботу з 3D-звуком, що дозволяє розміщувати звукові ефекти у віртуальному просторі, що дозволяє створити більш реалістичний звуковий досвід.

Система скриптінгу (Scripts System): це механізм, що дозволяє розробникам створювати скрипти , які керують поведінкою об'єктів в грі. Це важлива частина розробки ігор, оскільки дозволяє гнучко керувати геймплеєм, фізикою та іншими аспектами гри, не змінюючи сам двигун.

Для написання скриптів використовується мова C++. Одна з головних переваг C++ полягає у швидкості виконання. Як низькорівнева мова програмування, C++ набагато швидше виконується порівняно з іншими високорівневими мовами. Це особливо важливо для ігрового рушія, який потребує великої продуктивності для обробки великої кількості даних у реальному часі.

Інтерфейс користувача (User Interface Engine): Це компонент програмного забезпечення, який відповідає за створення та керування користувацьким інтерфейсом в додатку. Основна мета - це забезпечити зручну та ефективну взаємодію користувача з додатком, крім того, дозволяє забезпечити єдність дизайну та стилю інтерфейсу, що допомагає зробити додаток більш професійним та привабливим для користувачів.

За основу мого рушія було обрано бібліотеку “ImGUI”, тому що вона відрізняється від традиційних бібліотек GUI, таких як Qt чи GTK, тим, що вона використовує безпосередній режим рендерингу [7].

У бібліотеці ImGUI відбувається пряма взаємодія між програмою та інтерфейсом користувача. Інтерфейс створюється та відображається протягом одного циклу програми. Якщо користувач змінює стан елементів інтерфейсу, то він безпосередньо впливає на логіку програми, не зберігаючи надмірної кількості інформації в пам'яті. Такий підхід робить бібліотеку ImGUI ефективною для використання в ігровій розробці, де продуктивність є критично важливою.

Так як ігровий рушій потрібно якось поширювати у маси, для нього був створений власний сайт. Для створення сайту було використано такі технології:

TypeScript: дозволяє писати структурований та безпечний код, знижує кількість помилок в процесі розробки та підвищує швидкість розробки. Використовувався для написання серверної та клієнтської частини.

NodeJS: платформа, що забезпечує можливість створення високопродуктивних, масштабованих та ефективних додатків на основі JavaScript та TypeScript. Використовувалась для написання серверної та клієнтської частини.

NestJS: використовується для розробки серверних застосунків на мові програмування TypeScript, зокрема для створення веб-серверів, RESTful API.

ReactJS: використовується для створення клієнстьких застосунків з багатою та інтерактивною взаємодією з користувачем.

PostgreSQL: БД, яка забезпечує широкі можливості для роботи з даними.

PrismaORM: модель інтерфейсу програмування додатків, яка забезпечує легку роботу з базами даних.

2.2 Аналіз вимог до користувачів ігрового рушія та сайту під ігровий рушій

Головною метою цього аналізу було з'ясування потреб та очікувань цільової аудиторії, а також визначення вимог, які повинні відповідати розроблюваному ігровому рушію.

З'ясувалося, що головними користувачами даного ігрового рушія будуть розробники інді-ігор, які обирають OpenGL для створення своїх проектів. Для більш детального розуміння потреб цільової аудиторії, були враховані різні характеристики, такі як рівень досвіду у розробці ігор та інтереси.

Було визначено головні ролі користувачів ігрового рушія та сайту під ігровий рушій. Ігровий рушій “TWE” містить в собі такі ролі:

1. Розробник ігор.
    - Функції: Створення і редагування графічних об'єктів, програмування логіки гри.
    - Можливості системи: Доступ до розширеного набору інструментів для створення графічних об'єктів, взаємодія зі шейдерами, налаштування фізичних параметрів, робота зі звуком, створення скриптів.
    - Вимоги: Розуміння принципів комп'ютерної графіки, досвід програмування (перевага мови C++ або подібних), знання математики для розрахунків фізичних моделей.
2. Гравець.
    - Функції: Відтворення та проходження ігор, взаємодія з клавіатурою та мишкою, налаштування параметрів гри.
    - Можливості системи: Графічне відтворення відповідно до можливостей ігрового рушія, можливість налаштування графічних, звукових та керування грою.
    - Вимоги: Основні навички гри на комп'ютері, розуміння взаємодії з ігровими пристроями, знання основ геймплею та жанрів ігор.

Загальні вимоги до всіх користувачів системи:

- Базове розуміння принципів 3D-графіки та OpenGL.
- Здатність до самостійного вивчення документації та використання ресурсів спільноти розробників для отримання додаткової інформації та рішення проблем.

Для сайту під ігровий рушія були визначені такі користувацькі ролі:

1. Розробники ігор або інші зацікавлені користувачі.
    - Функції: Перегляд новин про ігровий рушій, оцінювання новин.
    - Можливості системи: Доступ до сторінки новин, доступ до оцінювання новин.

- Вимоги: Обліковий запис на сайті.

2. Адміністратор/Модератор.
    - Позначення функцій: Модерація контенту, співпраця з гравцями.
    - Можливості системи: Доступ до модераторської панелі, можливість перегляду та видалення контенту.
    - Вимоги до знань, умінь і навиків: Знання політик та правил платформи, комунікативні навички для взаємодії з користувачами.

2.3 Аналіз вимог до апаратного та програмного забезпечення

Ігрові рушії зазвичай є доволі вимогливими до продуктивності так як вони надають величезний функціонал, включаючи хорошу графіку. У моему випадку я намагався знизити цей поріг, щоб більше користувачів могли використовувати саме мій рушій.

Нижче наведено загальні вимоги до апаратного забезпечення, які можна врахувати при початку розробки(характеристики саме такі, так як на такому апаратному забезпечені були проведені заміри продуктивності):

Рекомендовані характеристики апаратного забезпечення:

1. Процесор: AMD Ryzen 5 2600 3.4Ghz.
2. Графічний процесор: Nvidia RTX 2060 6GB.
3. RAM: 16GB 3200Mhz.
4. Жорсткий диск або SSD: 1GB.
5. Звукова карта: Realtek HD Audio.

Основні вимоги до програмного забезпечення включають наступні пункти:

1. Операційна система Windows 10 x64.
2. Компілятор MSVC 2019.
3. CMake 3.20.

Також для роботи сайту потрібне таке програмне забезпечення:

1. NodeJS 16.13.0.
2. Web-Browser з Chrome V8.
3. Усі завантажені модулі проекту.
4. TypeScript Compiler: 5.0.4.

2.4 Висновки до розділу 2

1. Розглянуті загальні вимоги до апаратного та програмного забезпечення для ігрового рушія та сайту під ігровий рушій.
2. Вказані рекомендовані параметри, що є достатніми для забезпечення високої продуктивності та графічної якості в ігрових додатках, що використовують цей рушій.
3. Визначені вимоги для кожної із ролей.
4. Проведено аналіз технологій, що використовувались при розробці ігрового рушія: C++, графічний рушій, фізичний рушій, звуковий рушій, система скриптінгу, інтерфейс користувача; та при розробці сайту під ігровий рушій: TypeScript, NodeJS, NestJS, ReactJS, PostgreSQL, PrismaORM.

3 РОЗРОБКА ІГРОВОГО РУШІЯ ДЛЯ ІНДІ ІГОР ТА САЙТУ ПІД НЬОГО

3.1 Опис предметної області

Предметна область цієї роботи включає в себе розробку та використання ігрового рушія для створення незалежних ігор, які націлені на інді-розробників. Дана предметна область поєднує знання з комп'ютерної графіки, програмування та ігрової розробки.

Ігровий рушій є основним компонентом в процесі розробки комп'ютерних ігор, який забезпечує виконання різних завдань, таких як рендеринг графіки, обробка фізики, управління взаємодією гравця з оточенням і багато іншого. Використання ігрового рушія спрощує процес розробки, дозволяючи розробникам інді-ігор сконцентруватися на творчому процесі створення гри, замість вирішення технічних проблем.

Метою створеного сайту під ігровий рушій є надання можливості завантаження самого ігрового рушія, перегляду новин про рушій та оцінка цих новин.

Дипломна робота зосереджується на розробці ігрового рушія на основі OpenGL - відкритої графічної бібліотеки, яка надає розробникам доступ до низькорівневих функцій графічного прискорювача. OpenGL забезпечує широкі можливості для створення візуально привабливих ігор з використанням різноманітних ефектів, таких як освітлення, тіні, текстури та шейдери.

Окрім цього, у роботі будуть розглянуті загальні принципи розробки ігрових рушіїв, включаючи архітектуру та функціональність.

Ця дипломна робота спрямована на розширення знань у галузі комп'ютерної графіки та ігрової розробки, а також на створення корисного інструменту для інді-розробників, які мають бажання створювати свої власні незалежні ігри з використанням OpenGL.

3.2 Архітектура системи

3.2.1 Архітектура ігрового рушію для інді ігор

Архітектура є важливим аспектом при створенні ігрового рушія, оскільки вона визначає структуру та організацію коду, що використовується для розробки відеоігор. Для реалізації рушія було створено два головних шари додатку та один зв’язуючий [11]. На Рис.*4* 4 зображено зв’язок шарів додатку ігрового рушія.

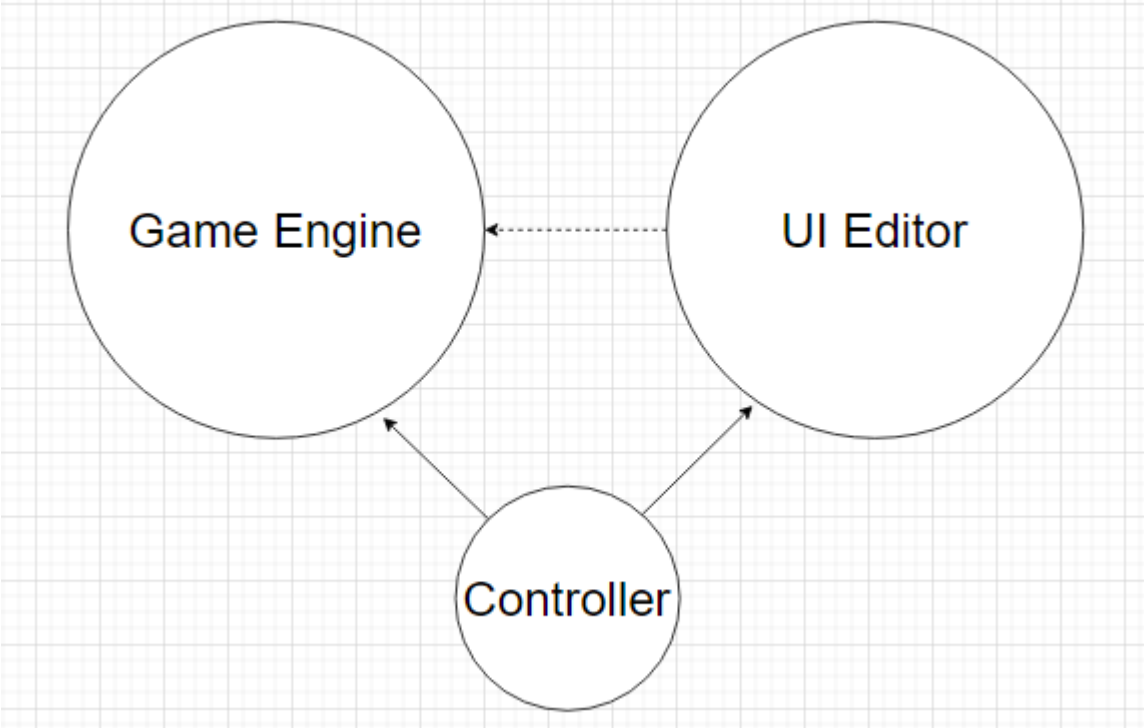

Рис.*4 Шари додатку “Ігровий рушій”*

Зв'язуючий шар (Controller) є важливою складовою цієї системи, оскільки він відповідає за забезпечення взаємодії між різними компонентами рушія під час запуску додатку. Для реалізації цього контролеру був створений клас “Engine”.

Зв’язуючий шар пов’язує роботу двох головних шарів між собою: Ігровий рушій (Game Engine) та Графічний редактор (UI Editor).

3.2.1.1 Шар “Ігровий рушій”

Перший шар програми це сам ігровий рушій. **Ігровий рушій** має складну архітектуру, яка включає в себе багато модулей:

Одним із таких модулей є **рендерер** (клас “Renderer”) — модуль, який відповідає за візуалізацію графіки на екрані. Він отримує дані з ігрової сцени, обробляє їх і створює візуальний результат, який гравець бачить на своєму екрані. На лістингу 1 зображено структуру модулю рендерера.

Лістинг *1 Модуль “Renderer”*

```
class Renderer {
    public:
        static void init(int lightShaderNamesSize = 30);
        static void render2D(const RendererSpecification& rendererSpec);
        static void render3D(const RendererSpecification& rendererSpec, const
glm::vec3& cameraPosition, const glm::mat4& cameraView, const glm::mat4&
cameraProjection,
            const glm::mat4& cameraProjectionView, int lightsCount);
        static void renderScene(IScene* scene);
        static void cleanScreen(const glm::vec4& color);
        static void cleanDepth();
        static void setViewport(int startX, int startY, int endX, int endY);
        static void setLight(MeshRendererComponent& meshRendererComponent,
const LightComponent& light, TransformComponent& transform, int lightIndex);
        static void setShadows(MeshRendererComponent& meshRendererComponent,
const glm::mat4& lightSpaceMat, int lightIndex);
        static void generateDepthMap(LightComponent& lightComponent, const
TransformComponent& transformComponent, const glm::mat4& lightProjection,
                                     const glm::mat4& lightView, const
glm::mat4& projectionView, IScene* scene);
    private:
        static void renderShadowMap(IScene* scene, const glm::vec3& position,
const glm::mat4& view, const glm::mat4& projection, const glm::mat4&
projectionView);
        static std::vector<LightShaderNamesSpecification> lightShaderNames;
};
```

1. **init**() — Цей статичний метод ініціалізує рендерер.
2. **render2D**() та **render3D()** — Статичні методи для візуалізації 2D та 3D зображення.
3. **renderScene**() — Статичний метод для візуалізації сцени.

4. **cleanScreen**() — Статичний метод для очищення екрану з заданим кольором.
5. **cleanDepth**() — Статичний метод для очищення буфера глибини.

Інший важливий модуль є **фізичний рушій** (інтерфейс “IScenePhysics”) — модуль, який відповідає за обробку фізики в грі. Фізичний двигун розраховує фізичні взаємодії між об'єктами, такі як зіткнення та гравітація. На лістингу 2 зображено структуру модулю фізичного рушія.

Лістинг *2 Модуль “IScenePhysics”*

```
class IScenePhysics {
    public:
        IScenePhysics() = default;
        virtual void reset(entt::registry* registry) {}
        virtual void updateWorldSimulation(entt::registry* registry, float
deltaTime) {}
        virtual void debugDrawWorld() {}
        virtual void checkCollisionsDetection() {}
        virtual void cleanCollisionDetection() {}
        [[nodiscard]] virtual std::vector<const btCollisionObject*>
getCollisionDetection(const btCollisionObject* obj) { return {}; }
        [[nodiscard]] virtual btDynamicsWorld* getDynamicWorld() const
noexcept { return nullptr; }
        [[nodiscard]] virtual SceneBulletDebugDrawer* getDebugDrawer() const
noexcept { return nullptr; }
};
```

1. **reset**() — метод для скидання стану фізичної сцени або симуляції.
2. **updateWorldSimulation**() — метод для оновлення симуляції світу.
3. **debugDrawWorld**() — метод для відображення візуального представлення світу. Він може відображати фізичні об'єкти, колізії або інші візуальні елементи для налагоджування і тестування.
4. **checkCollisionsDetection**() — метод для виявлення колізій між об'єктами у фізичній сцені або симуляції.
5. **getCollisionDetection**() — метод, який повертає вектор константних вказівників btCollisionObject, що представляють об'єкти, які зіштовхнулися з заданим об'єктом obj.

Третім важливим модулем є **система скриптів** (інтерфейс “ISceneScripts”) — модуль, який відповідає за взаємодію сутностей на сцені за допомогою створених, розробником гри, скриптів, які надають реалізований, в цих скриптах, функціонал даній сутності. На лістингу 3 зображено структуру модулю системи скриптів.

Лістинг *3 Модуль “ISceneScripts”*

```
class ISceneScripts {
    public:
        ISceneScripts() = default;
        virtual void reset(entt::registry* registry) {}
        virtual void update(entt::registry* registry, IScene* scene) {}
        virtual void bindScript(DLLLoadData* dllData, Entity& entity) {}
        virtual void bindScript(DLLLoadData* dllData, std::vector<Entity>&
entities) {}
        virtual void validateScript(const std::string scriptName, IScene*
scene) {}
        virtual void validateScripts(IScene* scene) {}
        virtual std::vector<Entity> unbindScript(entt::registry* registry,
DLLLoadData* dllData, IScene* scene) { return {}; }
};
```

1. **reset**() — цей метод викликається, коли сцена скидається або перезавантажується. Він використовується для очищення даних, пов'язаних зі скриптами, які збережені в реєстрі сутностей.
2. **update**() — цей метод викликається кожен кадр гри. Він використовується для оновлення стану скрипту відповідно до змін, які відбуваються в грі.
3. **bindScript**() — цей метод викликається при зв'язуванні скрипту з конкретною сутністю або групою сутностей.
4. **validateScript**() та **validateScripts()** — цей метод викликається для перевірки валідності скрипту/скриптів перед зв'язуванням з сутністю.
5. **unbindScript**() — цей метод викликається для від'єднання скрипту від сутності чи групи сутностей.

Останнім, та не менше важливим, модулем є **аудіо система** (інтерфейс “ISceneAudio”) — модуль, який відповідає за відтворення звукових ефектів та

музики в іграх з метою покращення геймплею та забезпечення більш іммерсивного досвіду гравця. На лістингу 4 зображено структуру модулю аудіо системи.

Лістинг *4 Модуль "ISceneAudio"*

```
class ISceneAudio {
    public:
        ISceneAudio() = default;
        virtual void reset() {}
        virtual void startAudioOnRun(entt::registry* registry) {}
        virtual void setAudioPauseState(entt::registry* registry, bool
paused) {}
        virtual bool updateAudioListenerPosition(const
SceneCameraSpecification& cameraSpec) { return false; }
        virtual irrklang::ISoundEngine* getSoundEngine() { return nullptr; }
};
```

1. **reset**() — метод, який використовується для скидання поточного стану об'єкта;
2. **startAudioOnRun**() — метод, який служить для запуску звукових ефектів в ігровій сцені;
3. **setAudioPauseState**() — метод, який дозволяє призупинити або продовжити відтворення звуків в ігровій сцені;
4. **updateAudioListenerPosition**() — метод, який оновлює позицію слухача звуків в ігровій сцені;
5. **getSoundEngine**() — метод, який повертає об'єкт звукового двигуна для взаємодії зі звуками в ігровій сцені.

Для створення взаємодії між всіма, переліченими зверху (окрім модуля "Рендерер"), модулями використовується система під назвою **"сцена" (Scene).** Сцена містить в собі ці модулі, так як вона успадковується від інтерфейсу "IScene", та використовує цей функціонал, кожен кадр, для забезпечення так званого "руху гри", таким чином генерується картинка. На Рис.5 5 зображено діаграму класів поверхневих модулів рушія.

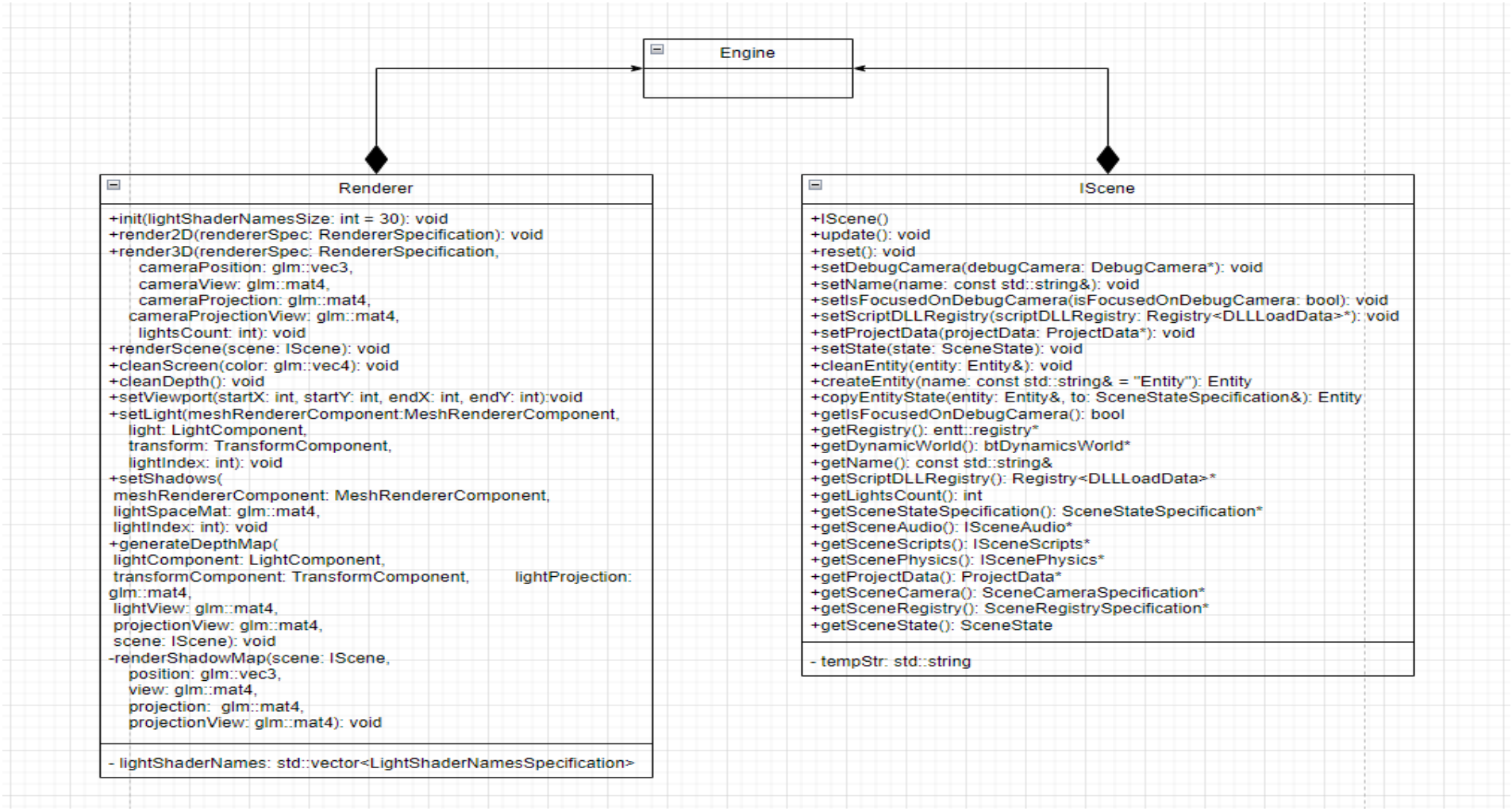

Рис.5 *Діаграма класів поверхневих модулів рушія*

Для реалізації інтерфейсу сцени був створений клас “Scene”, який реалізовує не тільки успадковані методи, а ще містить в собі такі дані, як реєстр сутностей сцени. Щоб створювати взаємодію сутностей зі сценою, для них можна додавати компоненти. Кожна сутність може містити в собі декілька, заздалегідь створених, компонентів, кожен з яких містить в собі свій функціонал:

1. AudioComponent: Цей клас надає можливість додавати, видаляти та отримувати інформацію про джерела звуку в компоненті аудіо. Він також містить посилання на двигун звуку, який можна використовувати для відтворення звукових ефектів. Клас AudioComponent дозволяє створювати багато джерел звуку та управляти ними для створення різних аудіофункцій в програмі.
    a. clean() — виконує очищення компонента аудіо, включаючи видалення всіх джерел звуку;
    b. addSoundSource() — додає нове джерело звуку до компонента аудіо;
    c. removeSoundSource() — видаляє вказане джерело звуку з компонента аудіо;
    d. getSoundSources() — повертає вектор вказівників на всі джерела звуку, що містяться в компоненті аудіо;
    e. getPathsOfSoundSources() — повертає вектор рядків зі шляхами до всіх джерел звуку, що містяться в компоненті аудіо.
2. CameraComponent: містить методи для встановлення джерела та фокусу камери, а також методи для отримання інформації про фокус та джерело камери.
    a. setSource() — встановлює джерело камери;
    b. setFocuse() — встановлює фокус на камеру;
    c. isFocusedOn() — повертає логічне значення, яке показує, чи є фокус налаштованим на цю камеру;

   d. getSource() — повертає посилання на об'єкт типу Camera, який є джерелом камери для даного компоненту.

3. CreationTypeComponent: містить дані про тип сутності.
   a. setType() — встановлює тип сутності;
   b. getType() — повертає тип сутності.
4. IDComponent: містить дані для ідентифікування сутності.
5. LightComponent: Цей клас представляє компоненту світла, яка містить параметри та налаштування світла, такі як колір, радіуси, тип, коефіцієнти затухання та інші. Він також надає методи для отримання цих налаштувань і взаємодії з об'єктом FBO для використання світла.
   a. setType() — встановлює тип освітлення;
   b. setInnerRadius(), setOuterRadius() — встановлює внутрішній/ зовнішній радіус освітлення;
   c. setConstant(), setLinear(), setQuadratic() — встановлює константний, лінійний/квадратичний/коефіцієнт освітлення;
   d. setColor() - встановлює колір освітлення;
   e. getInnerRadius(), getOuterRadius() — повертає внутрішній/ зовнішній радіус освітлення;
   f. getConstant(), getLinear(), getQuadratic() — повертає константний, лінійний/квадратичний/коефіцієнт освітлення;
   g. getType() — повертає тип освітлення;
   h. getColor() — повертає колір освітлення.
6. MeshComponent: представляє компонент мешу в контексті комп'ютерної графіки. Він зберігає дані про вершини, індекси, VAO, VBO, EBO та текстуру, пов'язані з мешем. Клас надає різні конструктори для створення об'єкта MeshComponent з різними комбінаціями параметрів. Методи дозволяють встановлювати та отримувати значення для різних властивостей компонента мешу, таких як VAO, VBO, EBO, текстура, ідентифікатор реєстрації та специфікація моделі. Приватні поля зберігають внутрішні дані компонента мешу, а

приватний метод create використовується для створення VAO, VBO та EBO на основі переданих масивів вершин та індексів.

   a. setMesh() — встановлює нові значення для VAO, VBO, EBO, ідентифікатора реєстрації та специфікації моделі;
   b. setTexture() — встановлює нову текстуру;
   c. getVAO(), getVBO(), getEBO() — повертає спільний вказівник на VAO/ VBO/ EBO;
   d. getTexture() — повертає спільний вказівник на текстуру;
   e. create() — створює VAO, VBO та EBO на основі переданих масивів вершин та індексів.

7. MeshRendererComponent: реалізує функціональність візуалізації мешів за допомогою шейдерів та матеріалів. Він забезпечує зручний інтерфейс для встановлення та отримання різних властивостей компонента, таких як шейдер, матеріал, ідентифікатор сутності та ідентифікатор реєстру.
   a. updateMaterialUniform() — оновлює глобальні змінні матеріалу у шейдері;
   b. updateMatsUniform() — оновлює глобальні змінні матриць моделі, виду та проекції;
   c. setShader() — встановлює шейдер;
   d. setMaterial() — встановлює матеріал;
   e. getMaterial() — повертає посилання на об'єкт Material;
   f. getShader() — повертає об'єкт std::shared_ptr<Shader>, який представляє шейдер компонента.
8. NameComponent: містить дані про назву сутності.
   a. setName() — метод, що встановлює значення імені;
   b. getName() — метод, який повертає значення імені.
9. ParentChildsComponent: містить дані про батьківського та дочірніх сутностей.

10. PhysicsComponent: використовується для керування фізичною поведінкою об'єктів у грі. Він має різні методи для встановлення параметрів фізичних властивостей об'єкта, таких як маса, позиція, обертання, розмір і тип колайдера. Клас також має методи для отримання цих параметрів. Він також надає статичні методи для створення форми колізії відповідно до вказаного типу колайдера.
    a. setMass() — встановлює масу об'єкта;
    b. setPosition(), setRotation(), setSize() — встановлює позицію/ обертання/розмір об'єкта;
    c. setColliderType() — встановлює тип колайдера;
    d. setIsTrigger() — встановлює прапорець, що вказує, чи є об'єкт тригером;
    e. getMass() — повертає масу об'єкта;
    f. getColliderType() — повертає тип колайдера;
    g. getIsTrigger() — повертає прапорець, що вказує, чи є об'єкт тригером;
    h. getPosition(), getRotation(), getSize() — повертає позицію/ обертання/розмір об'єкта;
    i. createShape() — створює форму колізії з вказаними розмірами для вказаного типу колайдера.
11. ScriptComponent: надає можливість додавати, видаляти та отримувати інформацію про скрипти, пов'язані з певними поведінками.
    a. clean() — видаляє всі скрипти з компоненту;
    b. bind() — зв'язує новий скрипт з компонентом;
    c. unbind() — від'єднує скрипт, пов'язаний з поведінкою за заданим ім'ям класу поведінки;
    d. getScript(), getScripts() — повертає вказівник/вказівники на скрипт/ скрипти;
    e. getScriptsBehaviorName() — повертає вектор імен поведінок, зв'язаних з усіма скриптами.

12. TransformComponent: відповідає за обробку і зберігання трансформаційних даних об'єкта в тривимірному просторі.
    a. rotateAroundOrigin() — поворот об'єкта навколо вказаної точки по трьом осях;
    b. rotate() — поворот об'єкта навколо заданої вісі на певний кут;
    c. move() — переміщення об'єкта до вказаної позиції;
    d. scale() — зміна розмірів об'єкта;
    e. getPosition(), getRotation(), getSize() — повертає позицію/ обертання/розмір об'єкта;
    f. getForward(), getRight(), getUp() — отримання вектора, що вказує у напрямку вперед/праворуч/вгору від об'єкта;
    g. getZeroRotationAroundPos() — отримання вектора, що вказує на позицію об'єкта після повороту на 0 градусів навколо вказаної точки.

Щоб користувач міг взаємодіяти зі сценою, сутностями та їх компонентами під час запущеної гри, потрібно скористатися системою скриптів. Для цього потрібно створити один з них. Для того, щоб скрипт був валідний, він повинен успадковуватися від класу “Behavior”, що надає інтерфейс методів, які потрібно реалізувати для надання певного функціоналу. Скрипти пишуться на мові “C++”. На лістингу 5 зображено клас Behavior.

Лістинг *5 Клас “Behavior”*

```
class Behavior {
    public:
        virtual void start() {}
        virtual void update(float deltaTime) {}
        virtual void collisionDetection(Entity collidedEntity, const
btCollisionObject* collisionObj) {}
    protected:
        template<typename T>
        bool hasComponent();
        template<typename T>
        T* getComponent();
        template<typename T, typename ...Args>
```

```
        T* addComponent(Args&&... args);
        template<typename T>
        void removeComponent();
        template<typename T = void>
        void destroy();
        void loadScene(const std::filesystem::path& scenePath);
        InputSpecification* input;
        ImGuiContext* imguiContext;
        Entity gameObject;
        std::filesystem::path rootPath;
        ShapeSpecification* shapeSpec;
    private:
        bool needLoadScene = false;
        std::filesystem::path loadScenePath;
        void setInput(InputSpecification* input);
        void setImguiContext(ImGuiContext* context);
        void setRootPath(const std::filesystem::path& rootPath);
        void setShapeSpecification(ShapeSpecification* shapeSpec);
        friend class ScriptSpecification;
        friend class SceneScripts;
};
```

Після створення скрипта, він компілюється у **DLL** файл, який потім підключається до ігрового рушія.

3.2.1.2 Шар “Графічний редактор”

Другий шар додатку має назву — **Графічний редактор**. Він дозволяє користувачам редагувати вміст та параметри ігрових об'єктів за допомогою зручного інтерфейсу. На Рис.*6* 6 зображено діаграму класів вікон графічного редактору.

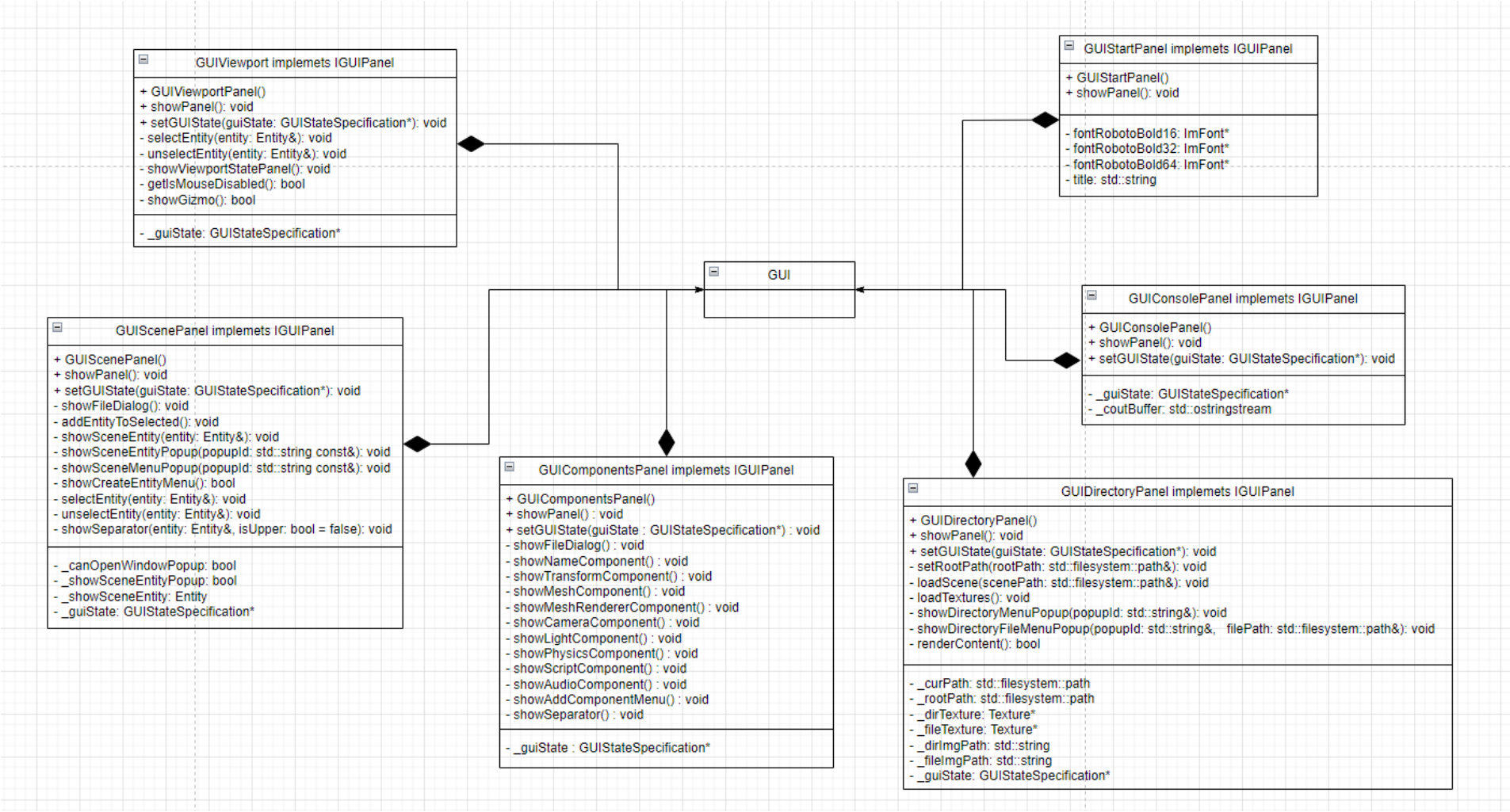

Рис.6 *Діаграма класів вікон графічного редактору*

Головним класом цього шару є “**GUI**”. Цей клас містить в собі функціонал для реалізації відображення графічного редактора.

1) **begin**() та **end**() — Виконують початок та завершення процесу відображення графічного інтерфейсу користувача;
2) **getHasBGFuncs**(), **getIsMouseOnViewport**(), **getIsFocusedOnViewport**(), **getIsMouseDisabled**() та **getIsURExecuted**() — Повертають стан певних атрибутів графічного інтерфейсу;
3) **showDockSpace**() — Відображає док-простір;
4) **showFileDialog**() — Відображає діалогове вікно файлу;
5) **showGizmo**() — Відображає гізмо (інструменти переміщення/повороту/масштабування);
6) **showMenuBar**() — Відображає меню панелі;
7) **processInput**() — Обробляє введення користувача;
8) **processBGFuncs**() — Обробляє функції фонового графічного інтерфейсу;
9) **processUR**() — Обробляє команди скасування/повторення.

Загальна структура класу GUI полягає в тому, що він виконує ініціалізацію та управління різними компонентами графічного інтерфейсу, відображає різні панелі, обробляє введення користувача та зберігає стан інтерфейсу. Клас також надає методи для установки сцени, вікна та даних проекту для взаємодії з інтерфейсом. Ця архітектура дозволяє зручно організувати роботу з графічним інтерфейсом у контексті класу GUI та його методів. За допомогою вказаних методів та приватних функцій, клас GUI може ефективно відображати різні елементи інтерфейсу, обробляти введення користувача та зберігати стан інтерфейсу для подальшого використання.

Для відображення редактора були створені окремі вікна, що реалізовують свій функціонал. Для зберігання цих вікон у одному контейнері, вони успадковуються від інтерфейсу “IGUIPanel”. Цей інтерфейс містить метод для відображення самого вікна та, метод для встановлення

стану вікна. Стан вікна — це клас “GUIStateSpecification”, що містить в себе такі дані:

1) посилання на інтерфейс сцени;
2) обрана сутність;
3) дані про поточний проект;
4) посилання на вікно;
5) масив фонових функцій;
6) інформація про обраний піксель на панелі “Viewport”;
7) інформація про те, чи курсор миші на панелі “Viewport”;
8) інформація про те, чи фокус на панелі “Viewport”;
9) поточна операція над сутністю;
10) дані про undo/redo.

Графічний редактор містить в собі декілька вікон:

a) Вікно зі сценою (Scene): містить дані про реєстр сутностей та надає змогу його змінювати.
   - **a)** showPanel() — відображає панель сцени;
   - **b)** showFileDialog() — відображає діалогове вікно файлу;
   - **c)** addEntityToSelected() — додає сутність до вибраних елементів;
   - **d)** showSceneEntity() — відображає сутність сцени;
   - **e)** showCreateEntityMenu() — відображає меню створення сутності.

b) Вікно з відображенням результуючої картинки (Viewport): містить дані про зображення гри та надає змогу легко взаємодіяти з сутностями.
   - **a)** showPanel() — відображає панель;
   - **b)** showViewportStatePanel() — відображає панель стану вікна перегляду;
   - **c)** showGizmo() — метод, який відображає гізмо (інструмент для трансформації об'єктів).

c) Вікно з компонентами сутності (Components): містить дані про компоненти обраної сутності та надає змогу змінювати їх параметри.
   - **a)** showPanel() — відображає панель;

b) showFileDialog() — відображає діалогове вікно для вибору файлу;

c) showNameComponent(), showTransformComponent(), showMeshComponent(), showMeshRendererComponent(), showCameraComponent(), showLightComponent(), showPhysicsComponent(), showScriptComponent(), showAudioComponent(), showAddComponentMenu() — відображає компонент.

d) Вікно директорії проекту (Directory): містить дані про поточну директорію та надає змогу змінювати її вміст.

a) showPanel() — відображає панель;

b) loadScene() — завантажує сцену за заданим шляхом;

c) loadTextures() — завантажує текстури;

d) renderContent() — рендерить вміст панелі;

e) Вікно з консоллю (Console): містить дані про повідомлення рушія та відображає їх: showPanel() — відображає панель.

Також редактор містить в собі реалізацію менюбара (menubar) та гарячих клавіш (hot keys), що дозволяє зручно та ефективно керувати процесом розробки ігор. Менюбар містить набір пунктів меню, які групують основні функції та опції редактора. Завдяки менюбару розробник може швидко здійснювати такі дії, як відкриття та збереження проекту, налаштування параметрів робочого середовища, керування об'єктами та ресурсами гри, а також виконання інших необхідних операцій. Гарячі клавіші надають можливість швидкого доступу до основних функцій редактора без потреби використовувати мишу або навігаційні панелі.

3.2.2 Архітектура сайту під ігровий рушій

Сайт було створено з використанням клієнт-серверної архітектуру [12], так як вона є стандартним підходом до розробки веб-додатків, який передбачає розділення функцій на дві основні частини: клієнтську (Front-end)

та серверну (Back-end) частини, з використанням бази даних (DB) для зберігання і керування даними.

3.2.2.1 DB частина

Частина бази даних (DB) в клієнт-серверній архітектурі відповідає за зберігання, організацію та керування даними, які використовуються в веб-додатку. База даних є центральним елементом системи, яка використовується для довготривалого зберігання даних та може бути доступна для використання серверної частини. Для побудови було обрано реляційну базу даних **PostgreSQL**, що надає можливість виконувати складні операції з даними, забезпечуючи цілісність даних, масштабованість, стабільність.

База даних містить в собі такі сутності(Табл.1 — 11):

Таблиця *1*

Таблиця “Token”

Token
userId: унікальний рядок, що містить ідентифікатор користувача
accessToken: унікальний рядок, що містить токен доступу
isActive: булеве значення, яке вказує, чи активний цей токен
lastAuthorization: поле дати/часу, яке містить дату останньої авторизації, зі значенням за замовчуванням - поточний час

Таблиця *2*

Таблиця "Role"

Role
id: унікальний рядок, що містить ідентифікатор ролі
value: рядок, який містить назву ролі

Таблиця *3*

Таблиця "ActivationLink"

ActivationLink
userId: унікальний рядок, що містить ідентифікатор користувача
link: унікальний рядок, що містить посилання на активацію аккаунту
isActivated: булеве значення, яке вказує, чи активоване це посилання на активацію

Таблиця *4*

Таблиця "ResetPassword"

ResetPassword
userId: унікальний рядок, що містить ідентифікатор користувача
token: унікальний рядок, що містить токен відновлення
isApproved: булеве значення, що вказує на підтвердження зміни пароля

Таблиця *5*

Таблиця "User"

User
id: унікальний рядок, що містить ідентифікатор користувача
email: рядок, що містить електронний адрес користувача
username: рядок, що містить ім'я користувача
password: рядок, що містить пароль користувача
roleId: рядок, що містить ідентифікатор ролі користувача
createdAt: дата, яка використовується як дата створення запису, значення за замовчуванням - поточний час

Таблиця *6*

Таблиця "NewsCard"

NewsCard
id: унікальний рядок, що містить ідентифікатор новинної картки
imgPath: рядок, що містить шлях до зображення новини
createdAt дата, яка використовується як дата створення запису, значення за замовчуванням - поточний час
createdById: рядок, що містить ідентифікатор користувача, який створив новину

Таблиця *7*

Таблиця "NewsBlock"

NewsBlock
id: унікальний рядок, що містить ідентифікатор новинного блоку
imgPath: рядок, що містить шлях до зображення блоку
createdAt: дата, що містить дату і час створення блоку
newsCardId: рядок, що містить зовнішній ключ на поле id таблиці NewsCard
createdById: рядок, що містить ідентифікатор користувача, який створив новину

Таблиця *8*

Таблиця "NewsText"

NewsText
id: унікальний рядок, що містить ідентифікатор тексту новини
title: рядок, що містить заголовок новини
body: рядок, що містить вміст новини
globalRegionId: рядок, що містить зовнішній ключ на поле id таблиці GlobalRegion
newsCardId: рядок, що містить зовнішній ключ на поле id таблиці NewsCard
newsBlockId: рядок, що містить зовнішній ключ на поле id таблиці NewsBlock

Таблиця *9*

Таблиця "GlobalRegion"

GlobalRegion
id: унікальний рядок, що містить ідентифікатор глобального регіону
name: рядок, що містить назву регіону

Таблиця *10*

Таблиця "NewsBlockRate"

NewsBlockRate
id: унікальний рядок, що містить ідентифікатор оцінки новинного блоку
likes: ціле число, що містить кількість лайків
dislikes: ціле число, що містить кількість дизлайків
newsBlockId: рядок, що містить зовнішній ключ на поле id таблиці NewsBlock

Таблиця *11*

Таблиця "UserBlockRate"

UserBlockRate
newsBlockRateId: рядок, що містить зовнішній ключ на поле id таблиці NewsBlockRate
userId: унікальний рядок, що містить ідентифікатор користувача
isLike: булеве значення, що позначає, чи оцінений цей рейтинг лайком чи ні.

Було створено таку ER схему бази даних даної системи. На Рис.*7* 7 зображено ER діаграму бази даних.

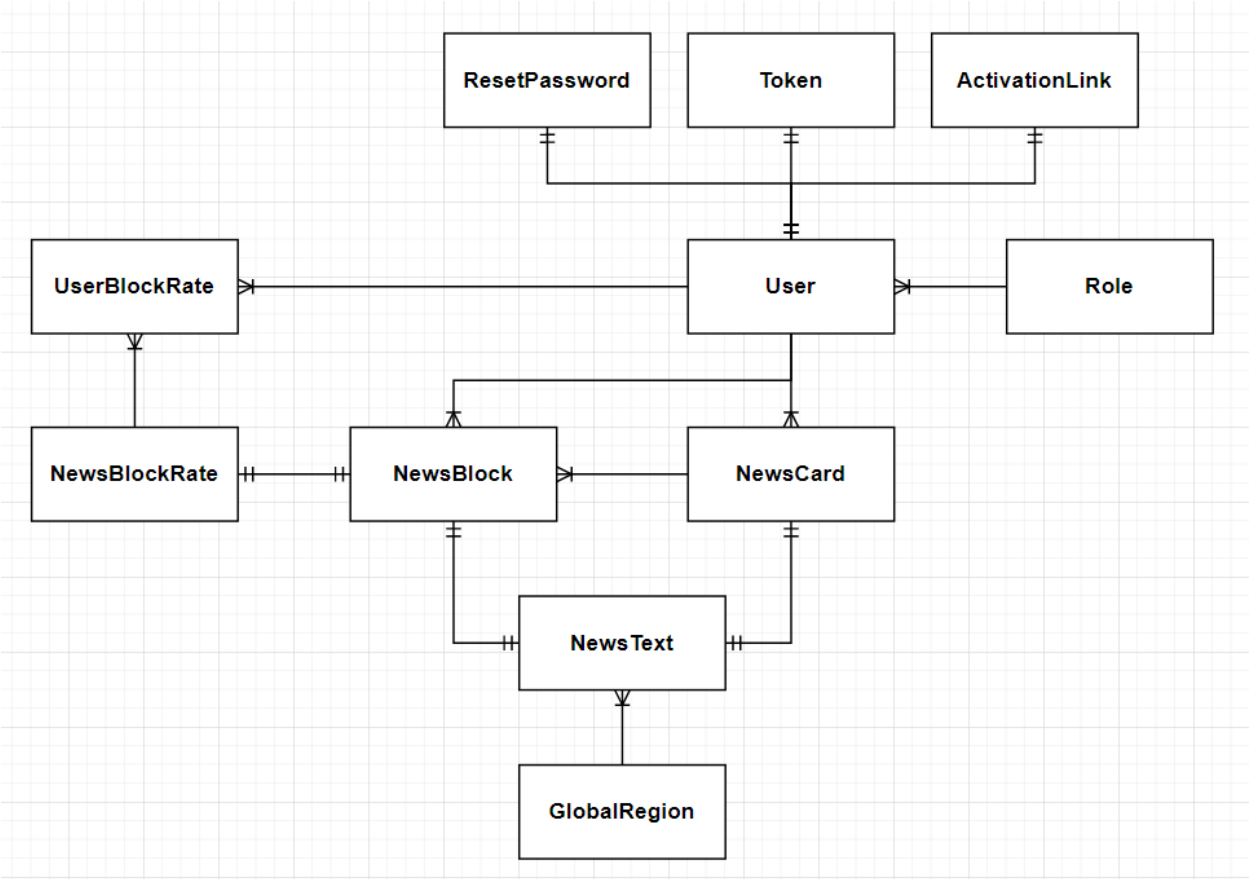

Рис.*7 ER діаграма бази даних*

3.2.2.2 Back-end частина

Back-end частина в клієнт-серверній архітектурі відповідає за обробку запитів, логіку, бізнес-логіки, доступ до баз даних, автентифікацію та авторизацію користувачів, а також за надання даних та ресурсів клієнтській частині додатка веб-сайту.

Back-end частина була створена на платформі **NodeJS** використовуючи фреймворк **NestJS**, так як він пропонує модульну структуру, яка спрощує організацію коду, підтримує паттерн архітектури MVC та має вбудовану підтримку для розробки масштабованих і ефективних додатків. Також це дозволить створити **REST API** додаток, який базується на **HTTP** протоколі.

Побудувавши схему бази даних, у попередньому пункті, можна виділити 2 головних домени додатку: User та NewsCard.

User — домен, який відображає обліковий запис користувача та містить в собі такі сутності, як:

1. **ActivationLink** — сутність, яка надає посилання для створення облікового запису користувача.
2. **ResetPassword** — сутність, яка надає можливість відновити пароль користувача.
3. **Token** — сутність, що автентифікує та надає доступ користувачеві до головної частини додатку.
4. **Role** — сутність, що яка надає користувачеві певний функціонал.

NewsCard — домен, який відображає новинну картку та містить в собі такі сутності, як:

1. **NewsBlock** — сутність, яка містить в собі дані про новину.
2. **NewsText** — сутність, яка містить в собі тексти новинного блоку або новинної картка, який залежить в регіону.
3. **GlobalRegion** — сутність, що відображає регіон.
4. **NewsBlockRate** — сутність, яка містить в собі оцінки користувачів певного новинного блоку.
5. **UserBlockRate** — сутність, що пов’язує дві сутності між собою: User та NewsBlockRate.

Для полегшення роботи з БД було використано **Prisma ORM**. За допомогою цього ORM можна швидко побудувати усі моделі, зробити до них міграції (migrations) та зробити сіди (seeds) до них. Клас, який виконує підключення для БД називається “TWEPostgreSQLMainPrismaService”. Цей клас надає доступ до усіх моделей, за допомогою яких можна виконувати запити до БД. Усі модулі, яким потрібен доступ до БД, містять в собі цей клас. Для додання цього класу у інші модулі, використовується така техніка як **Dependency Injection**.

Для реалізації енд поінтив (end-points) для модулів були створені свої конкретні контролери (controllers), які відловлюють запити користувачів та повертають відповідь з певним результатом обробки запиту. Самі контроллери в свою чергу реалізовують функціонал обробки конкретних

запитів. Також контролери включають в себе ще такі сутності, як: репозиторії(repositories) та сервіси(services):

1. Репозиторії використовуються для розділення бізнес-логіки від логіки доступу до даних та надають змогу взаємодіяти з даними за певним шаблоном.
2. Сервіси використовуються для виконання бізнес-логіки та обробки бізнес-операцій в окремих компонентах додатку.

3.2.2.3 Front-end частина

В клієнт-серверній архітектурі, front-end відповідає за ту частину додатка, яка взаємодіє безпосередньо з користувачем. Front-end здійснює візуальне представлення інформації, обробку користувацьких дій та взаємодію з сервером для отримання та відправлення даних.

Front-end частина, так як і серверна, побудована на платформі **NodeJS**. За основу було обрано фреймворк **ReactJS**, так як він дозволяє створювати масштабовані, ефективні та динамічні користувацькі інтерфейси за допомогою компонентної архітектури, яка спрощує розподіл веб-додатку на окремі реактивні компоненти, що легко керувати та підтримувати. Крім того, ReactJS забезпечує високу продуктивність, завдяки використанню **віртуального DOM**, який дозволяє ефективно оновлювати відображення інтерфейсу при змінах даних.

Усі сторінки діляться по окремим файлам. Кожна сторінка містить в собі окремі компоненти, які реалізовують свій функціонал. Для універсальності цих компонентів вони мають свої певні атрибути (props), за допомогою яких можна контролювати їх станом.

Для того, щоб клієнтська частина розуміла, яку сторінку потрібну відобразити, була реалізована роут (route) частина, що регулює за цим за допомогою перевірки **URL шляху** користувача. Доступ до всіх роутів регулюється автентифікацією користувача. На лістингу 6 зображено компонент регулювання доступів користувача до сторінок.

Лістинг 6 *Головний компонент клієнтської частини, що регулює шлях користувача*

```
<div className={styles.app}>
       <div className={styles.app_container}>
       <NavbarComponent ready={ready}/>
        <Routes>
             <Route path={'/main'} element={<MainPage/>}/>
             <Route path={'/news'} element={<NewsPage/>}/>
             <Route path={'/news/:newsCardId'} element={<NewsCardPage/>}/>
             <Route path={'/forgot-password'} element={<ForgotPasswordPage
/>}/>
             <Route path={'/reset-password/:userId/:token'}
element={<ResetPasswordPage />}/>
             {!isAuth && <Route path={'/login'} element={<LoginPage/>}/>}
             {!isAuth && <Route path={'/registration'}
element={<RegistrationPage/>}/>}
             {!isAuth && <Route path={'/registration/accept/:userId/:link'}
element={<UserActivationPage/>}/>}
             <Route
              path="*"
              element={<Navigate to="/main" replace />}
             />
        </Routes>
        <FooterComponent />
       </div>
</div>
```

Для відправки запитів на сервер реалізована окрема директорія “actions”, що містить в собі файли з реалізацією. Після відправки клієнтська частина отримує відповідь на запит з певними даними. Ці дані потрібно десь зберігати тому я використав **Redux**, який дає змогу управляти станом, за допомогою окремих редюсерів (reducers), додатку та зберігати там отримані дані. Було створено 3 редюсери:

1. **userReducer**: зберігає дані про поточного користувача;
2. **newsReducer**: зберігає дані про поточний список новин та поточну обрану новину;
3. **globalRegionReducer**: зберігає дані про поточний регіон.

Кожен з цих редюсерів використовується у сторінках та компонентах, що допомагає у відображені конкретного стану клієнтської частини.

3.3 Функціональні вимоги системи

3.3.1 Загальні вимоги

3.3.1.1 Ігровий рушій “TWE”

1. Система повинна забезпечувати реалізацію графічного руху та візуалізацію ігрового світу за допомогою технології OpenGL.
2. Система повинна надавати можливості для взаємодії з ігровим світом, зокрема:
    - **a.** Створювати та завантажувати об'єкти гри (моделі персонажів, об'єкти оточення, текстури тощо).
    - **b.** Обробляти вхідні дані від користувача (клавіатура, миша) для керування грою.
    - **c.** Реалізовувати фізику руху об'єктів та взаємодію між ними (колізії, гравітація тощо).
    - **d.** Забезпечувати відображення графічних ефектів (освітлення, тіні, частинки тощо).
    - **e.** Обробляти події та стан гри (завантаження рівнів, збереження прогресу, обробка втрати життів тощо).
3. Система повинна бути модульною і розділити функціональність на підсистеми згідно з наступними вимогами:
    - **a.** Графічна підсистема.
        1. Забезпечити створення та управління вікном гри.
        2. Реалізувати систему камери для відображення ігрового світу.
        3. Забезпечити відображення 2D та 3D графіки.
        4. Реалізувати механізм обробки текстур та шейдерів.
    - **b.** Фізична підсистема: Забезпечити симуляцію фізики об'єктів у грі (рух, колізії, гравітація тощо).
    - **c.** Управління введенням: Реалізувати обробку вхідних даних від користувача (клавіатура, миша).

d. Звукова підсистема.

1. Забезпечити відтворення звукових ефектів та музики у грі.
2. Реалізувати систему обробки звукових подій та мікшування звукових доріжок.

e. Підсистема скриптів.

1. Забезпечити створення скриптів.
2. Забезпечити обробку сутностей гри та їх компонентів за допомогою скриптів.
3. Система повинна мати підтримку операційної системи Windows.
4. Система повинна забезпечувати можливість збереження та завантаження стану проектів рушія та його сцен.
5. Система повинна забезпечувати оптимізацію швидкодії та продуктивності, використовуючи оптимальні алгоритми та структури даних.
6. Система повинна надавати зручні графічні інструменти для створення та редагування різноманітних ресурсів гри, таких як моделі, текстури тощо.
7. Система повинна надавати можливість створювати білд гри.

3.3.1.2 Сайт під ігровий рушій

1. Система повинна мати сторінку зі списком новин, які відображаються в хронологічному порядку.
2. Система повинна забезпечувати можливість перегляду повної версії кожної новини окремо.
3. Система повинна дозволяти користувачам реєструватися та автентифікуватися на сайті.
4. Система повинна забезпечувати різні рівні доступу для користувачів(адміністратори та звичайні користувачі).

5. Система повинна мати можливість завантаження ігрового рушія з сайту.
6. Система повинна забезпечувати можливість зміни пароля та відновлення забутого пароля користувачами.
7. Система повинна зберігати інформацію про користувачів, включаючи їхні облікові дані та персональну інформацію.
8. Система повинна забезпечувати захист від несанкціонованого доступу до особистої інформації користувачів.
9. Система повинна відображати повідомлення про помилки.

3.3.2 Бібліотека підпрограм (класів)

3.3.2.1 Бібліотеки ігрового рушія “TWE”

1. assimp: бібліотека використовується для імпорту та обробки 3D-моделей у різних форматах. Вона дозволяє користувачеві завантажувати моделі, отримувати доступ до їх геометрії, матеріалів та анімації.
2. bullet3: Bullet Physics Engine є високоефективним двигуном фізики, який надає реалістичне моделювання фізики об'єктів у грі. Він дозволяє користувачеві створювати тверді тіла, встановлювати фізичні властивості, обчислювати зіткнення та реагувати на них.
3. entt: бібліотека, що надає систему управління компонентами для реалізації екземплярно-орієнтованих систем у грі. Вона дозволяє користувачеві визначати компоненти, системи та системи обробки подій для ефективного управління сутностями гри.
4. glad: бібліотека генерує код для завантаження точок входу OpenGL в залежності від версії OpenGL, яка використовується. Вона допомагає користувачам отримати доступ до функцій OpenGL та розширень.
5. GLFW: бібліотека надає переносимий інтерфейс для роботи з вікнами, контекстами OpenGL, введенням подій та обробкою клавіш. Вона

дозволяє користувачам створювати вікна, обробляти введення, керувати контекстами OpenGL та обробляти події вікна.

6. glm: бібліотека надає математичні функції та класи для роботи з геометрією і матрицями у 3D-графіці. Вона дозволяє користувачам виконувати операції з векторами, матрицями, кутами повороту, проекціями та інші математичні операції, необхідні для обробки 3D-графіки.
7. imgui: бібліотека (ImGui, що означає "Immediate Mode GUI") надає інструменти для швидкої та простої реалізації графічного інтерфейсу користувача (GUI). Вона дозволяє розробникам створювати кнопки, полоси прокрутки, вікна та інші елементи інтерфейсу для взаємодії з користувачем.
8. imgui-filedialog: бібліотека розширює функціональність ImGui, дозволяючи користувачеві відкривати та зберігати файли через діалогові вікна. Вона надає можливість вибору файлів та каталогів зручним способом у вашій ігровій програмі.
9. ImGuizmo: бібліотека є розширенням ImGui і додає інструменти для маніпулювання об'єктами у 3D-просторі. Вона дозволяє користувачам переміщати, обертати та масштабувати об'єкти у візуалізації 3D-сцени.
10. irrKlang: бібліотека є потужним аудіо-двигуном для відтворення звуків у вашій ігровій програмі. Вона надає можливості програвання звуків різної формату, зміни гучності, позиції звукових джерел та інші функції аудіо-управління.
11. nlohmann json: бібліотека дозволяє роботу з форматом JSON у вашій програмі. Вона дозволяє користувачам зручно читати та записувати дані в форматі JSON, обробляти їх, виконувати пошук, зчитувати та записувати значення за ключами та інші операції для роботи зі структурованими даними.
12. KHR: бібліотека надає доступ до розширень OpenGL, включаючи нові можливості та функціональність. Вона дозволяє користувачам

використовувати останні розширення OpenGL, які підтримуються їхнім апаратним забезпеченням, для поліпшення графічного візуалу та продуктивності.

13. stb: бібліотека містить набір одноразових файлів заголовків для завантаження зображень, шрифтів та інших ресурсів. Вона надає користувачам прості та легкі у використанні інструменти для завантаження та роботи з різними типами файлів зображень та інших ресурсів.

3.3.2.2 Бібліотеки сайту під ігровий рушій

1. Express: бібліотека дозволяє створювати веб-додатки на базі Node.js, надаючи розширену функціональність для обробки HTTP-запитів, роутінгу та управління сесіями та куками.
2. NestJS: бібліотека є фреймворком для створення ефективних та масштабованих серверних додатків на базі Node.js. Вона надає модульну структуру, вбудовану обробку HTTP-запитів та підтримку веб-сокетів.
3. Uuid: бібліотека дозволяє генерувати унікальні ідентифікатори (UUID) для об'єктів. Вона корисна для створення унікальних ідентифікаторів для об'єктів, наприклад, користувачів або ігрових елементів.
4. Bcryptjs: бібліотека надає функції для хешування та перевірки паролів. Вона використовує алгоритм bcrypt, що забезпечує безпеку паролів шляхом хешування з сolью та повільним обчисленням хешу.
5. Prisma: ORM (Object-Relational Mapping) для Node.js, яке дозволяє легко взаємодіяти з базою даних. Воно надає можливість визначати моделі даних та виконувати операції з базою даних, такі як створення, читання, оновлення та видалення даних.
6. React та React-DOM: бібліотеки дозволяють розробляти користувацький інтерфейс за допомогою компонентної моделі. React

надає реактивні компоненти та react-dom забезпечує можливість відображати ці компоненти в браузері.

7. React-Router-DOM: бібліотека дозволяє реалізувати маршрутизацію на стороні клієнта в React-додатку. Вона надає можливість визначати шляхи та відповідні компоненти, які мають бути відображені при відповідному шляху.
8. Node-Sass: бібліотека дозволяє використовувати препроцесор Sass для написання стилів у проекті. Вона надає можливість використовувати змінні, міксини та інші функції Sass для більш зручного та організованого написання CSS.
9. Redux та React-Redux: ці бібліотеки дозволяють реалізувати управління станом додатку в React-додатку за допомогою патерну Flux. Redux надає централізоване сховище стану, а react-redux забезпечує зв'язок між React-компонентами та Redux-сховищем.
10. Redux-Thunk: бібліотека дозволяє використовувати асинхронні дії (thunks) у Redux-додатку. Вона надає можливість виконувати асинхронні запити, обробляти побічні ефекти та диспетчеризувати дії в Redux.

3.4 Функціонал системи

3.4.1 Функціонал ігрового рушія “TWE”

Так як ігровий рушій має надавати як можна більше функціоналу користувачеві, було реалізовано значну кількість різноманітних функцій, щоб полегшити розробку користувачам, реалізовуючи примітивні речі, щоб вони не гаяли час при розробці та надати більший фокус розробці конткретних функцій гри. Так як усіляких функцій багато було створено буде показано лише поверхневіші, що допоможе скласти уявлення про те, як воно взагалі працює та які алгоритми використовувались.

3.4.1.1 Функціонал шару “Ігровий рушій”

Ігровий рушій містить в собі чимало заздалегідь створених функцій, які не потребують повторного створення при розробці декількох ігор, що є важливим компонентом для економії часу та пришвидшення розробки.

Рендер сцени

Так як рендерер має містити функціонал для генерування зображення, було створено певний функціонал. Для рендеру сцени гри використовуються метод “renderScene” класу “Renderer”. На лістингу 7 зображено метод рендеру сцени.

Лістинг *7 Метод рендеру сцени “renderScene”*

```
void Renderer::renderScene(IScene* scene) {
        SceneCameraSpecification* camera = scene->getSceneCamera();
        if(!camera->camera)
            return;
        int lightsCount = scene->getLightsCount();
        bool isFocusedOnDebugCamera = scene->getIsFocusedOnDebugCamera();
        static std::vector<RendererSpecification> _2DEntities;
        scene->getRegistry()->view<MeshComponent, MeshRendererComponent,
TransformComponent>()
            .each([&](entt::entity entity, MeshComponent& meshComponent,
MeshRendererComponent& meshRendererComponent, TransformComponent&
transformComponent){
                if(!meshRendererComponent.getIs3D()) {
                    if(!isFocusedOnDebugCamera)
                        _2DEntities.push_back({&meshComponent,
&meshRendererComponent, &transformComponent});
                    return;
                }
                render3D({&meshComponent, &meshRendererComponent,
&transformComponent}, camera->position, camera->view, camera->projection,
camera->projectionView, lightsCount);
            });
        if(isFocusedOnDebugCamera) {
            #ifndef TWE_BUILD
            scene->getSceneStateSpecification()->physics->getDebugDrawer()-
>setMats(camera->view, camera->projection);
            scene->getSceneStateSpecification()->physics->debugDrawWorld();
            #endif
            return;
        }
        cleanDepth();
        for(auto& rendererSpec : _2DEntities)
```

```
            render2D(rendererSpec);
        _2DEntities.clear();
}
```

У метод “renderScene” передається вказівник на інтерфейс сцени “IScene”. Далі отримується вказівник на об'єкт камери з інтерфейсу “IScene”, який містить інформацію про камеру, таку як позиція, проекційна матриця і т. д. Потім перевіряється, чи існує камера (перевірка на нулевий вказівник), і якщо вона не існує, то рендеринг сцени припиняється і функція повертається. Отримуємо кількість джерел освітлення в сцені, значення якого буде передано у шейдери (shaders) сутностей. Отримує стан фокусу на камеру для дебагу (debug), що вказує, чи наразі фокус знаходиться на камері дебагу. Оголошується статичний вектор типу RendererSpecification, який використовується для зберігання відомостей про об'єкти, які будуть рендеритися в 2D. Виконується ітерація по всіх об'єктах сцени, які мають компоненти MeshComponent, MeshRendererComponent і TransformComponent, використовуючи функцію each(). Для кожного об'єкта виконується перевірка, чи це 2D-об'єкт, і якщо так, то відповідна інформація про цей об'єкт додається до вектора типу RendererSpecification для подальшого рендерингу. Якщо об'єкт є 3D-об'єктом, то викликається функція render3D(), яка приймає відомості про цей об'єкт, камеру та кількість джерел освітлення. Ця функція відповідає за рендеринг 3D-об'єкта на екрані, використовуючи вказані параметри камери та світлові моделі. Якщо фокус знаходиться на дебаг камері, то виконується відповідний рендеринг для коллайдерів(colliders), що візуалізують фізичні об'єкти сцени. Потім виконується очищення буфера глибини перед рендерингом 2D-об'єктів. Виконується рендеринг 2D-об'єктів на екрані, використовуючи відомості про ці об'єкти, які були збережені в векторі RendererSpecification. Та на у кінці очищується вектор типу RendererSpecification після рендерингу 2D-об'єктів, щоб підготувати його до наступного кадру.

Оновлення положення фізичних об’єктів

Для оновлення положення фізичних об’єктів використовується метод “updateWorldSimulation” класу “ScenePhysics”. На лістингу 8 зображено метод оновлення положення фізичних об’єктів сцени.

Лістинг *8 Метод оновлення положення фізичних об’єктів сцени “updateWorldSimulation”*

```
void ScenePhysics::updateWorldSimulation(entt::registry* registry, float
deltaTime) {
        registry->view<PhysicsComponent,
TransformComponent>().each([](entt::entity entity, PhysicsComponent&
physicsComponent,
        TransformComponent& transformComponent){
            btRigidBody* rigidBody = physicsComponent.getRigidBody();
            rigidBody->activate();
            if(physicsComponent.getMass() == 0.f)
                return;
            btTransform worldTransform;
            rigidBody->getMotionState()->getWorldTransform(worldTransform);
            btVector3 pos = worldTransform.getOrigin();
            btQuaternion quat = worldTransform.getRotation();
            transformComponent.setPosition(glm::vec3(pos.getX(), pos.getY(),
pos.getZ()));
            glm::vec3 rot;
            quat.getEulerZYX(rot.z, rot.y, rot.x);
            transformComponent.setRotation(rot);
            physicsComponent.setNeedUpdate(false);
        });
        _world->stepSimulation(deltaTime);
        checkCollisionsDetection();
}
```

Метод updateWorldSimulation має два параметри: вказівник на реєстр registry типу entt::registry, який містить сутності з компонентами PhysicsComponent та TransformComponent, і значення deltaTime типу float, яке відповідає за часовий крок симуляції. Використовується метод each реєстру registry для ітерації через кожної сутності, яка має компоненти PhysicsComponent та TransformComponent. Всередині кожного циклу функція виконує наступні дії:

1. Активує тіло rigidBody, яке зберігається в об'єкті PhysicsComponent, за допомогою методу activate() з бібліотеки Bullet Physics.

2. Перевіряє, чи маса фізичного об'єкту, збереженого в physicsComponent, дорівнює 0. Якщо так, то функція пропускає цю ітерацію за допомогою оператора return, інакше вона продовжує виконання наступних дій.
3. Отримує трансформацію фізичного тіла rigidBody за допомогою методів getWorldTransform() та getOrigin() та getRotation() відповідно.
4. Оновлює позицію та орієнтацію трансформації об'єкту, яка зберігається в transformComponent, використовуючи методи setPosition() та setRotation() і метод getEulerZYX().
5. Встановлює прапор needUpdate в false в physicsComponent, щоб відзначити, що фізичний компонент більше не потребує оновлення.

Після ітерації через всі сутності, функція викликає метод _world->stepSimulation(deltaTime) для виконання кроку симуляції фізичного світу. Після виконання симуляції, функція викликає метод checkCollisionsDetection(), який, перевіряє виявлення колізій між об'єктами в фізичному світі.

Оновлення стану скриптів

Для оновлення стану скриптів використовується метод “update” класу “SceneScripts”. На лістингу 9 зображено метод оновлення стану скриптів.

Лістинг 9 *Метод оновлення стану скриптів “update”*

```
void SceneScripts::update(entt::registry* registry, IScene* scene) {
        static std::filesystem::path loadScenePath;
        bool needLoadScene = false;
        registry->view<ScriptComponent>().each([&](entt::entity entity,
ScriptComponent& scriptComponent){
            auto& scripts = scriptComponent.getScripts();
            for(auto& script : scripts) {
                if(!script.isEnabled)
                    return;
                try {
                    if(!script.isInitialized) {
                        script.initialize(entity, scene, _projectData-
>rootPath, Shape::shapeSpec);
                        script.instance->start();
                    }
                    if(script.instance-
>gameObject.hasComponent<PhysicsComponent>()) {
```

```
                        auto& physicsComponent = script.instance-
>gameObject.getComponent<PhysicsComponent>();
                        auto collisions = scene-
>getSceneStateSpecification()->physics-
>getCollisionDetection(physicsComponent.getRigidBody());
                        for(auto collisionObj : collisions){
                            auto userPointer =
(PhysicsUserPointer*)collisionObj->getUserPointer();
                            script.instance->collisionDetection(Entity{
userPointer->entity, scene }, collisionObj);
                        }
                    }
                    script.instance->update(Time::getDeltaTime());
                    if(script.instance->needLoadScene) {
                        needLoadScene = true;
                        loadScenePath = script.instance->loadScenePath;
                        script.instance->needLoadScene = false;
                    }
                } catch(const std::exception& error) {
                    std::cout << error.what() << "\n\n";
                    script.isEnabled = false;
                }
            }
        });
        if(needLoadScene) {
            if(SceneSerializer::deserialize(scene, loadScenePath.string(),
_projectData)) {
                std::filesystem::path scenePath =
std::filesystem::relative(loadScenePath, _projectData->rootPath);
                if(!scenePath.empty())
                    _projectData->lastScenePath = scenePath;
                else
                    _projectData->lastScenePath = loadScenePath;
                scene->setState(SceneState::Run);
            }
        }
}
```

Він приймає два аргументи: entt::registry, який зберігає дані про сутності в сцені, і IScene, який представляє собою сцену.Метод містить цикл, який проходиться по кожній сутності, яка має компонент ScriptComponent. Для кожної такої сутності, метод отримує список скриптів, які були додані до сутності за допомогою ScriptComponent, і проходиться по кожному скрипту в списку.Для кожного скрипту метод перевіряє, чи він включений (isEnabled) і спробує його ініціалізувати, якщо він ще не був ініціалізований. Якщо скрипт містить PhysicsComponent, метод отримує стан колізії і здійснює обробку

подій зіткнення. Потім метод оновлює кожен скрипт та перевіряє, чи є потреба завантажити нову сцену. Якщо так, то він використовує SceneSerializer для десеріалізації нової сцени, зберігає шлях до неї та встановлює стан сцени на Run.

Завантаження сцени

Для загрузки сцени гри використовується метод “deserialize” класу “SceneSerializer”. Метод виконує десеріалізацію сцени з JSON-файлу і заповнює об'єкт scene відповідними даними про імена сцени, сутності та компоненти, зчитуючи їх з вхідного файлу JSON. На лістингу 10 зображено метод завантаження сцени.

Лістинг *10 Метод завантаження сцени “deserialize”*

```
bool SceneSerializer::deserialize(IScene* scene, const std::string& path,
ProjectData* projectData) {
        std::string jsonBodyStr = File::getBody(path.c_str());
        nlohmann::json jsonMain = nlohmann::json::parse(jsonBodyStr);
        if(!jsonMain.contains("Scene"))
            return false;
        scene->reset();
        scene->setName(jsonMain["Scene"]);
        auto& entities = jsonMain["Entities"].items();
        for(auto& [index, components] : entities) {
            Entity instance = deserializeCreationTypeComponent(scene,
components, projectData->rootPath);
            if(instance.getSource() != entt::null)
                deserializeEntity(scene, instance, components, projectData-
>rootPath);
        }
        revalidateParentChildsComponent(scene);
        return true;
}
```

Приймає параметри: scene: Вказівник на об'єкт типу IScene, який представляє сцену, в яку будуть десеріалізовані дані; path: Рядок, що містить шлях до файлу, з якого будуть зчитуватись дані для десеріалізації сцени; projectData: Вказівник на об'єкт типу ProjectData, який містить додаткові дані про проект. Зчитує вміст файлу, вказаного шляхом path, за допомогою функції File::getBody() і отримує рядок jsonBodyStr, що містить цей вміст.

Парсить рядок jsonBodyStr в об'єкт типу nlohmann::json за допомогою функції nlohmann::json::parse() і отримує об'єкт jsonMain, який представляє основний об'єкт JSON. Перевіряє наявність ключа "Scene" в об'єкті jsonMain. Якщо цей ключ відсутній, метод повертає false, що може свідчити про помилку десеріалізації. Викликає метод reset() на об'єкті scene, щоб очистити його поточний стан. Встановлює ім'я сцени з значення, що міститься у ключі "Scene" об'єкта jsonMain. Перебирає кожен елемент "Entities" у jsonMain. Для кожного елемента викликає функцію deserializeCreationTypeComponent(), передаючи йому об'єкт scene, компоненти елемента і шлях кореневої папки проекту з projectData. Результат цієї функції зберігається в змінній instance. Перевіряє, чи instance має джерело, використовуючи метод getSource(). Якщо так, то викликає функцію deserializeEntity(), передаючи їй об'єкт scene, instance, компоненти елемента і шлях кореневої папки проекту з projectData. Ця функція відповідає за десеріалізацію сутності. Після завершення ітерації по всіх елементах "Entities" викликається метод revalidateParentChildsComponent(), який перевіряє та оновлює зв'язки батьківської та дочірньої сутностей в сцені. Метод повертає значення true для позначення успішного виконання десеріалізації.

Створеня білд версії гри

Для створення білд версії гри використовується метод “create” класу “BuildCreator”. На лістингу 11 зображено метод створення білд версії гри.

Лістинг *11 Метод створення білд версії гри “create”*

```
bool BuildCreator::create(ProjectData* projectData, const
std::filesystem::path& startScenePath) {
        if(!projectData)
            return false;
        if(projectData->rootPath.empty())
            return false;
        if(!std::filesystem::exists(projectData->rootPath.string() + '/' +
startScenePath.string()))
            return false;
        std::string buildDirPath = projectData->rootPath.string() + "/build";
        if(!std::filesystem::exists(buildDirPath))
            std::filesystem::create_directory(buildDirPath);
```

```
        std::string debugPath = buildDirPath + "/Release";
        if(!std::filesystem::exists(debugPath))
            std::filesystem::create_directory(debugPath);
        std::string buildFilePath = debugPath + '/' + projectData-
>projectName + ".build";
        createBuildFile(projectData, buildFilePath, startScenePath);
        createCMakeFile(projectData, buildDirPath);
        copyRootPathFiles(projectData, buildDirPath);
        std::string generateCommand = "cmake -DCMAKE_BUILD_TYPE=Release -S "
+ buildDirPath + " -B " + buildDirPath;
        if(system(generateCommand.c_str()) != 0) {
            std::cout << "Failed to generate project build files\n";
            return false;
        }
        std::string buildCommand = "cmake --build " + buildDirPath + " --
config Release";
        if(system(buildCommand.c_str()) != 0) {
            std::cout << "Failed to build project\n";
            return false;
        }
        return true;
}
```

Приймає параметри: projectData: вказівник типу ProjectData*, який містить в собі дані про поточний проект; startScenePath: шлях початкової сцени гри. Перевіряється, чи передано дійсний вказівник projectData. Якщо вказівник недійсний (nullptr), повертається значення false і робота методу припиняється. Перевіряється, чи rootPath в projectData не є порожнім рядком. Якщо rootPath порожній, повертається значення false і робота методу припиняється. Перевіряється, чи існує файл, що відповідає шляху rootPath + '/' + startScenePath. Якщо файл не існує, повертається значення false і робота методу припиняється. Формується шлях до директорії buildDirPath, який складається з rootPath і "/build". Якщо buildDirPath не існує, створюється ця директорія. Формується шлях до директорії debugPath, який складається з buildDirPath і "/Release". Якщо debugPath не існує, створюється ця директорія. Формується шлях до файлу збірки buildFilePath, який складається з debugPath, "/", і імені проекту з projectData і розширенням ".build". Викликається функція createBuildFile для створення файлу збірки за допомогою projectData, buildFilePath і startScenePath. Викликається функція

createCMakeFile для створення файлу CMake за допомогою projectData і buildDirPath. Викликається функція copyRootPathFiles для копіювання файлів з rootPath в buildDirPath. Формується команда generateCommand для запуску команди CMake для генерації файлів проекту зі зборки. Команда використовує шлях buildDirPath і передає параметр -DCMAKE_BUILD_TYPE=Release для встановлення режиму збірки на "Release". Викликається функція system, яка виконує команду generateCommand у командному рядку. Якщо результат виконання команди не дорівнює 0, виводиться повідомлення про невдачу генерації файлів проекту, повертається значення false і робота методу припиняється. Формується команда buildCommand для запуску команди CMake для збірки проекту. Команда використовує шлях buildDirPath і передає параметр --config Release для вказання конфігурації збірки "Release". Викликається функція system, яка виконує команду buildCommand у командному рядку. Якщо результат виконання команди не дорівнює 0, виводиться повідомлення про невдачу збірки проекту, повертається значення false і робота методу припиняється. Якщо всі кроки успішно виконані, метод повертає значення true, що вказує на успішне створення і збірку проекту.

3.4.1.2 Функціонал шару “Графічний редактор”

Так як графічний інструментарій є одним із найголовнішим аспектом при виборі ігрового рушія для розробки, було розроблено максимально простий, для користувачів, функціональний інтерфейс. На Рис.*8* 8 зображено графічний редактор рушія “TWE”.

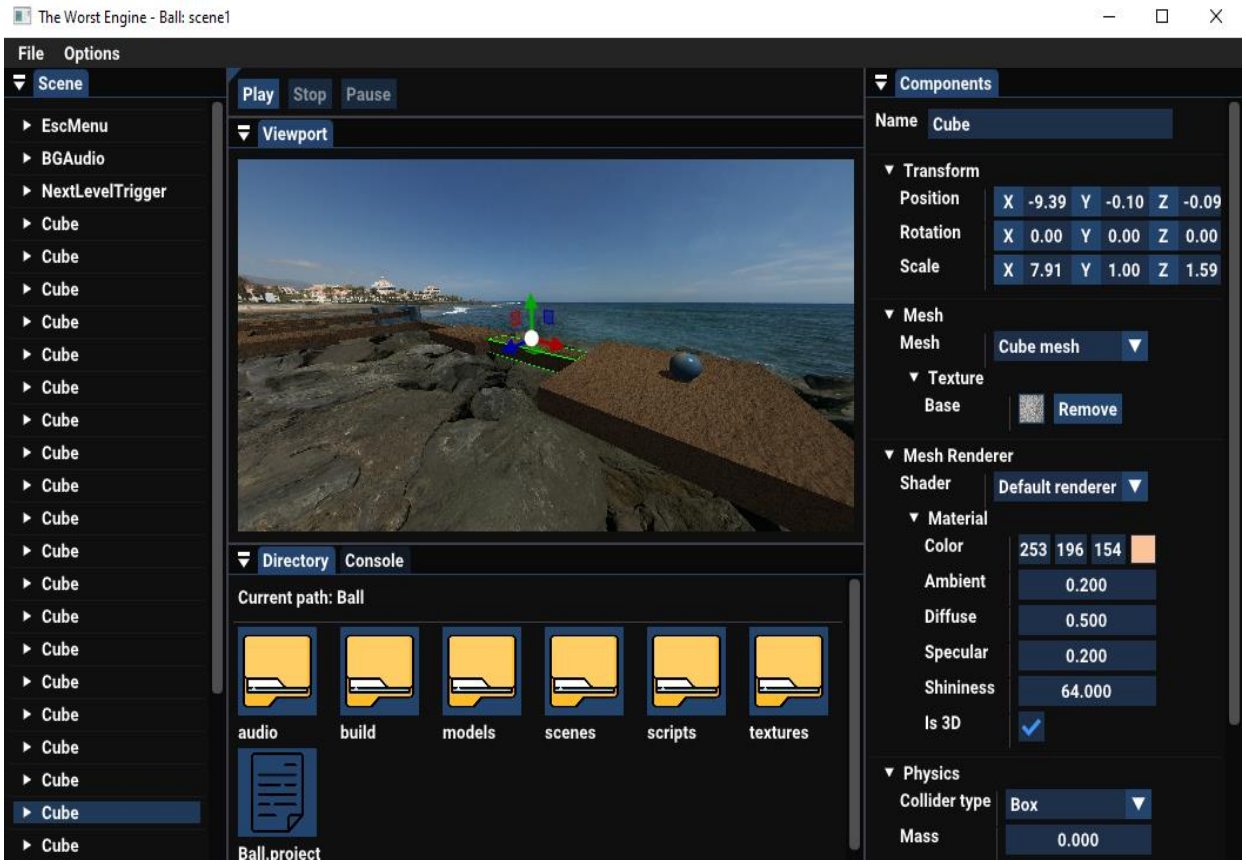

Рис.*8 Графічний редактор рушія*

Графічний редактор надає функціонал, який допомагає, без написання коду, додавати, створювати новий функціонал гри, використовуючи при цьому зручний та компактний графічний додаток. На Рис.*9* 9 зображено діаграму прецедентів графічного редактору рушія.

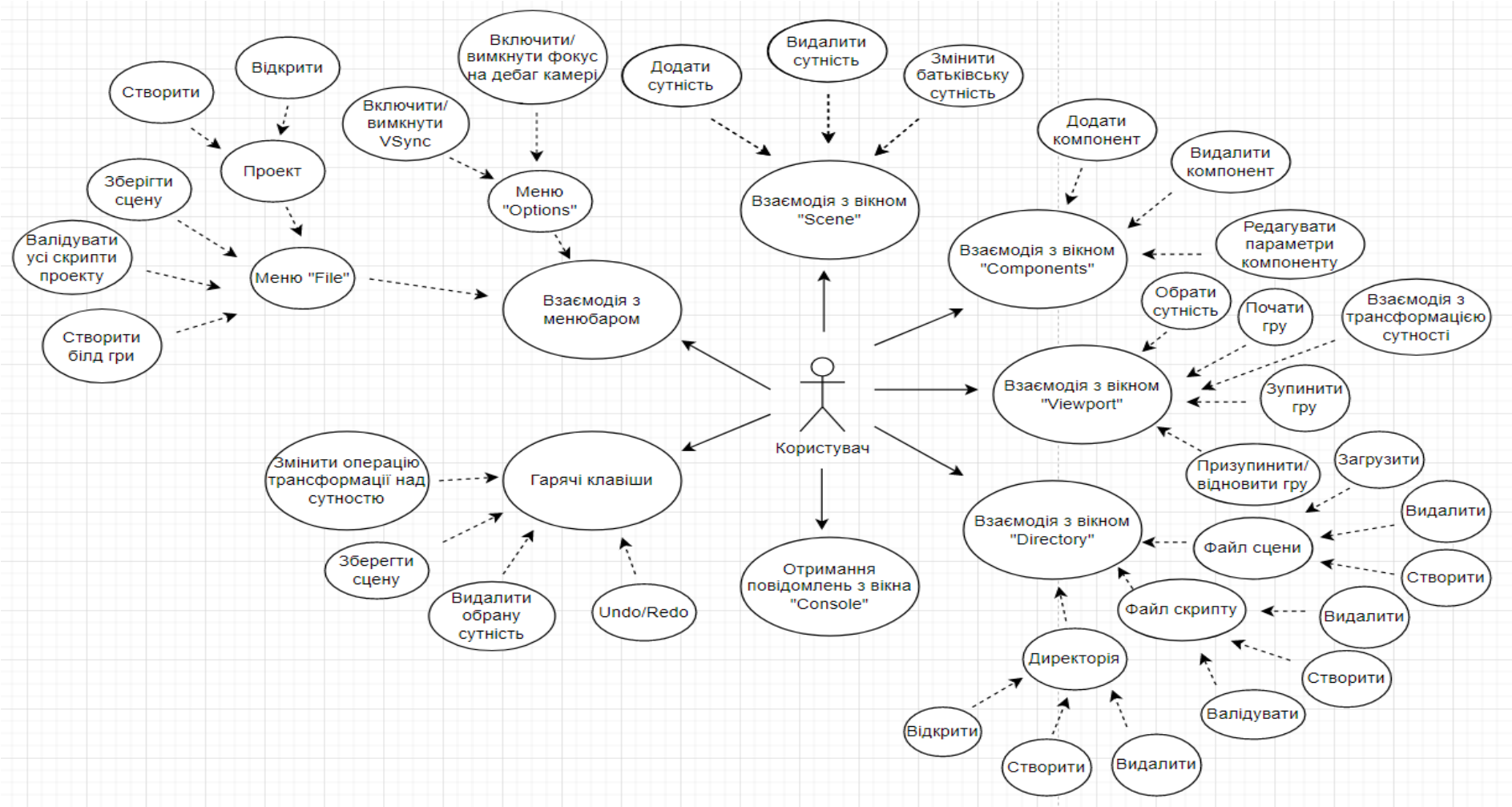

Рис.9 *Use case діаграма графічного редактору рушія*

Створення нового проекту

Графічний редактор надає можливість створити новий проект за допомогою класу “ProjectCreator”. Користувач може створити проект з початкового вікна редактору або вже при відчиненому іншому проекті за допомогою меню-бару. На Рис.*10* 10 зображено діаграму послідовності по створенню нового проекту.

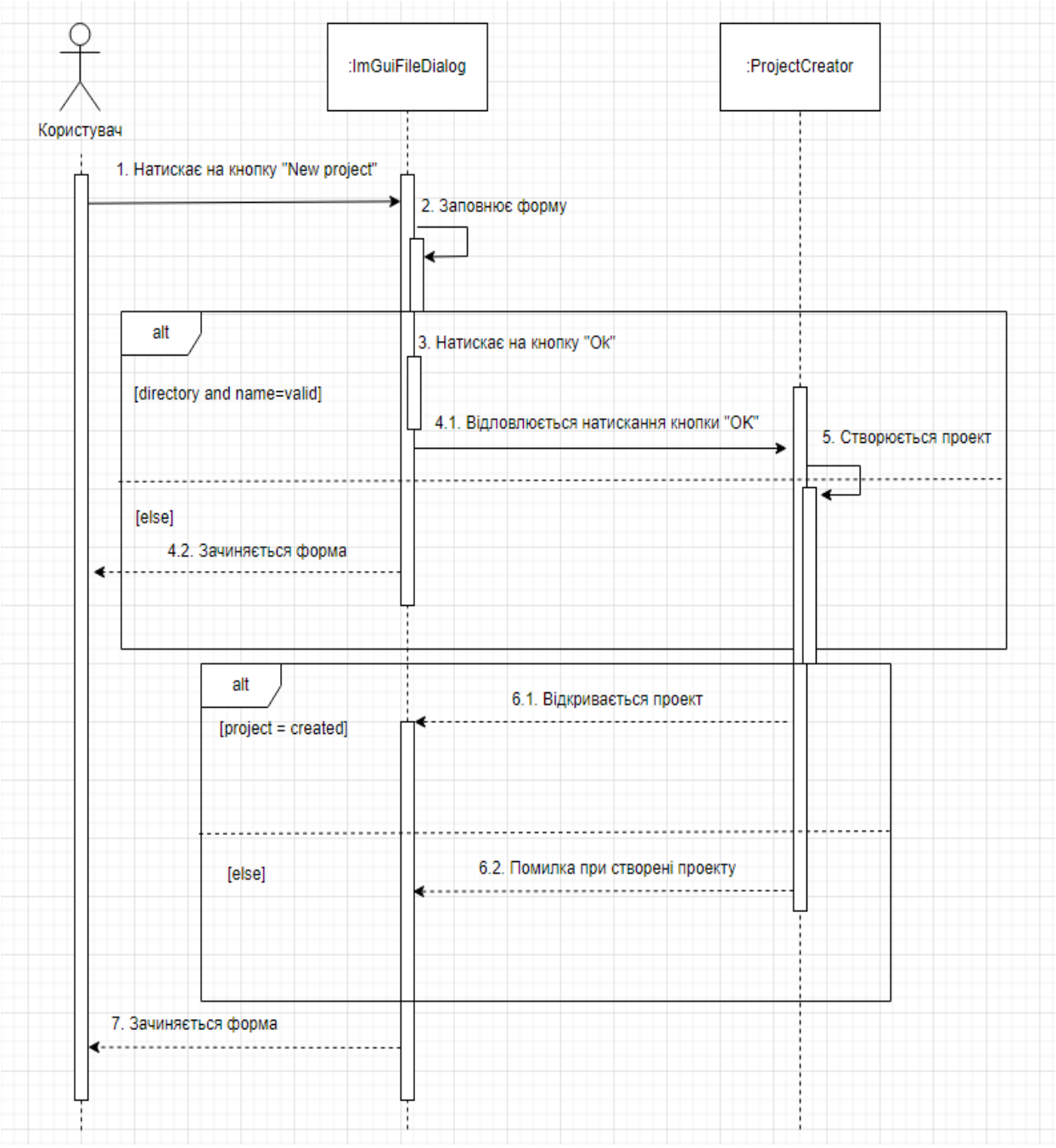

Рис.*10 Sequence діаграма “Створення нового проекту”*

Додання сутності до сцени

Графічний редактор підтримує створення **popup** меню, що допомагає створювати більш гнучкий інтерфейс додатку. За допомогою popup меню можна додавати нові сутності до сцени гри. На Рис.*11* 11 зображено діаграму послідовності по доданню нової сутності.

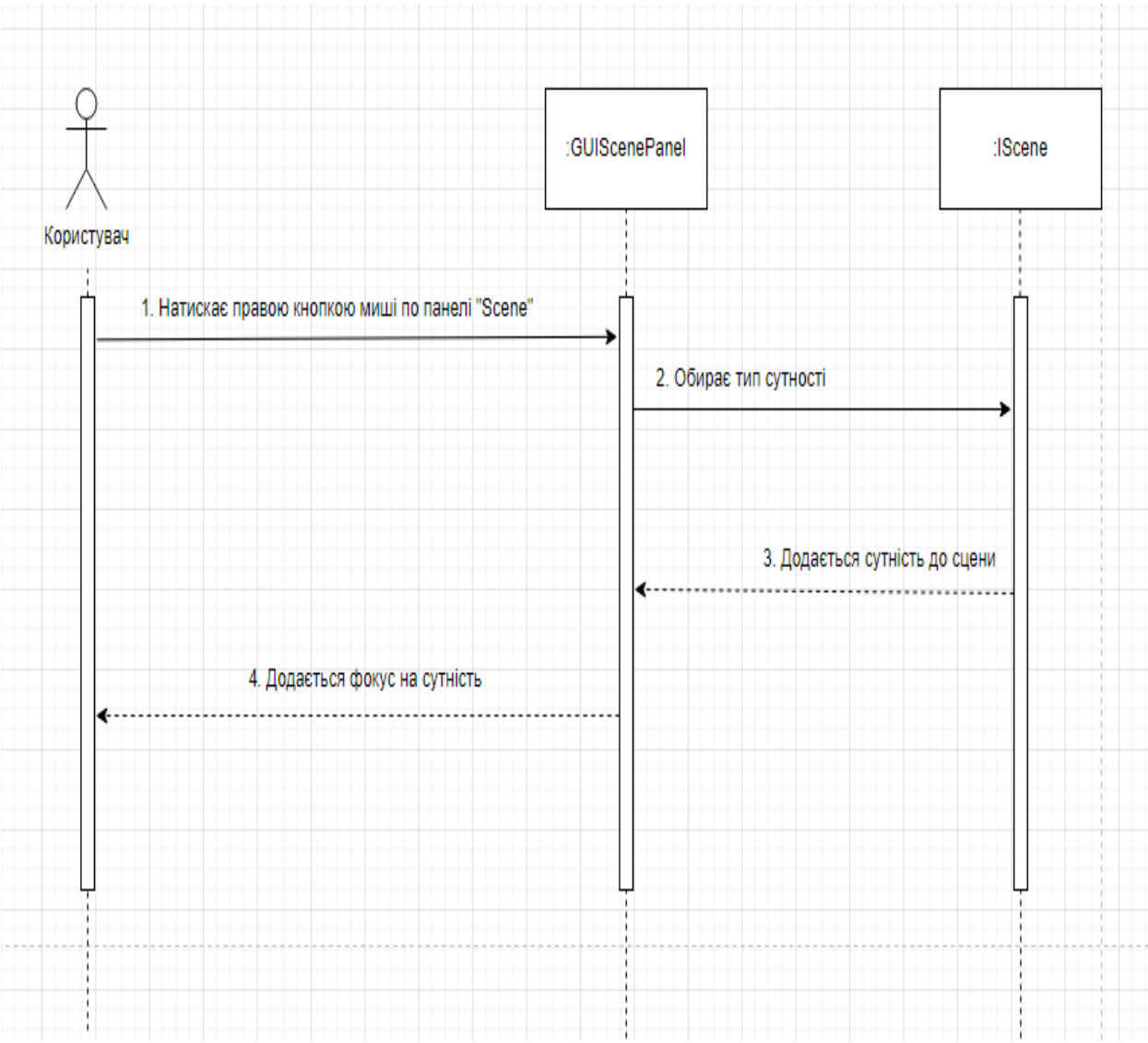

Рис.*11 Sequence діаграма "Додання нової сутності"*

Валідація скрипту

Використовується для того, щоб після написання скрипта його новий функціонал можна було використовувати. За допомогою редактору можна швидко валідувати скрипт. Після валідації скрипт оновлюється в усіх

сутностях, додаючи або прибираючи певний функціонал, який був змінений у скрипті. На Рис.*12* 12 зображено діаграму послідовності по валідації скрипту.

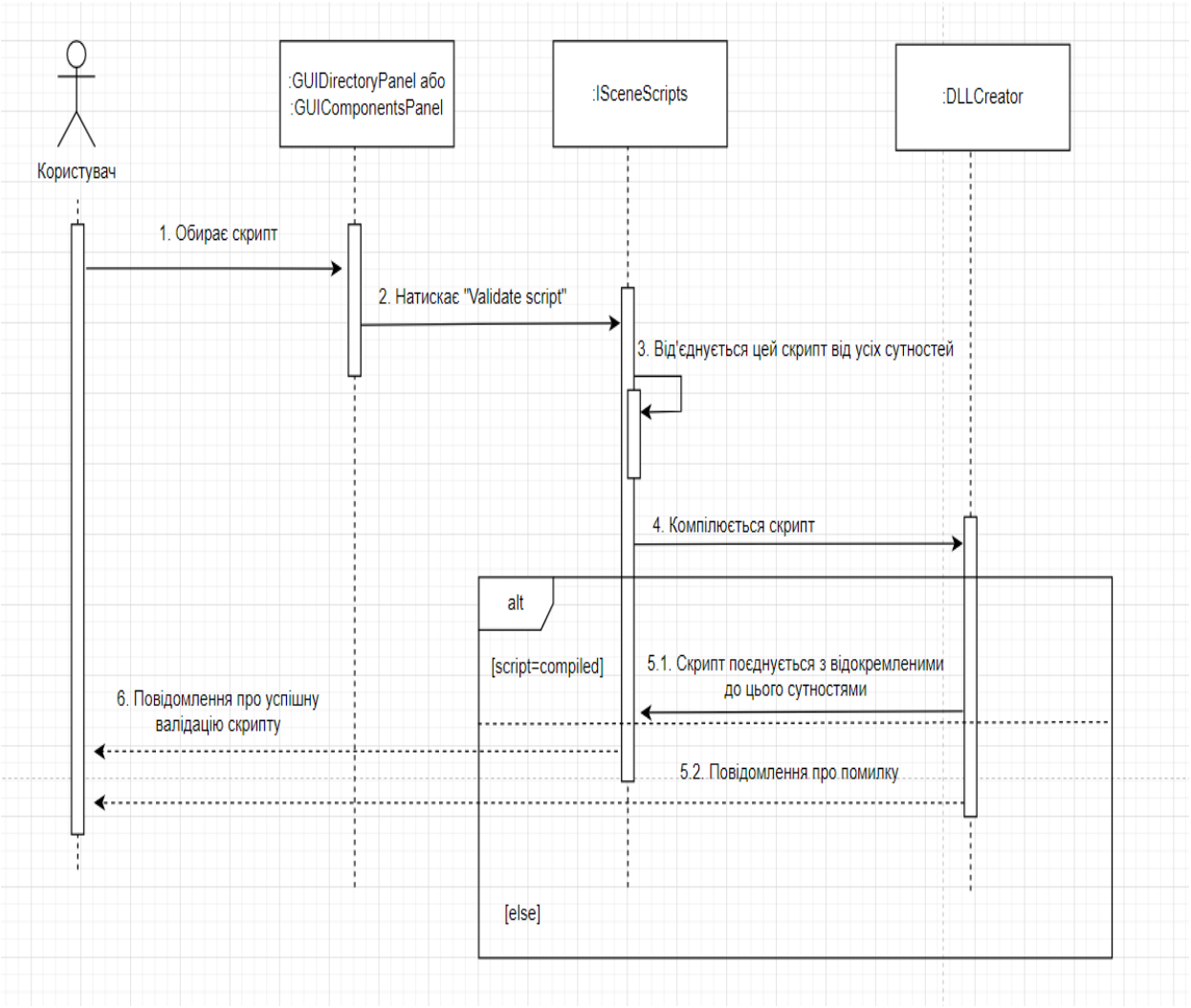

Рис.*12 Sequence діаграма "Валідація скрипту"*

3.4.2 Функціонал сайту під ігровий рушій

Сайт містить в собі невеличкий функціонал, який допомагає отримати сам ігровий рушій та новини про нього, що допомагають більше ознайомитись з розробкою самого рушія.

Завантаження ігрового рушія “TWE”

Завантаження рушія — це найпростіша дія, яку може виконати користувач. Для виконання цієї дії, користувачеві потрібно лише перейти до головної сторінки сайту та натиснути на кнопку ”Завантажити”.

Не менш простим є реалізація цього функціоналу. На backend частині є окремий endpoint, який надає можливість отримати файл з серверу. На лістингу 12 зображено метод завантаження ігрового рушія “TWE” з серверу.

Лістинг *12 Метод завантаження ігрового рушія “TWE” з серверу*

```
@Get('/twe/download')
async tweDownload() {
    if(!fs.existsSync(TWE_BUILD_PATH))
        throw new HttpException('TWE build was not found.', 404);
    const file = fs.createReadStream(TWE_BUILD_PATH);
    return new StreamableFile(file);
}
```

На початку виконується перевірки, чи існує сам додаток на сервері, якщо ні, то користувач отримує повідомлення про це. Якщо додаток є, то сервер відправляє цей файл користувачеві, як відповідь на його запит.

На frontend частині після натиснення кнопки ”Завантажити” користувач відправляє “GET” запит до серверу. Після перевіряє, чи успішною є відповідь. Після перевірки, у випадку успішного запиту, до браузеру додається прив’язка файлу з посиланням, з допомогою якого браузер автоматично завантажує цей файл до девайсу користувача. На лістингу 13 зображено метод завантаження ігрового рушія “TWE” з клієнтської частини.

Лістинг *13 Метод завантаження ігрового рушія “TWE” з клієнтської частини*

```
const tweDownload = async () => {
        try {
            const response = await fetch(`${BACK_URL}/file/twe/download`);
            if(response.status !== 200)
                throw response;
            const blob = await response.blob();
```

```
            const downloadLink = window.URL.createObjectURL(blob);
            const link = document.createElement('a');
            link.href = downloadLink;
            link.download = "TWE.rar";
            document.body.appendChild(link);
            link.click();
            link.remove();
            return response;
        } catch (error) {
            alert(error);
        }
}
```

Логін

На backend частині логін відбувається таким чином. На лістингу 14 зображено метод логіну на серверній частині.

Лістинг *14 Метод логіну на серверній частині*

```
@Post('/login')
    async login(@Body() dto: LoginDto){
        dto = loginMapper.fromControllerToService(dto);
        const user = await this.userRepository.getOneByEmail(dto.email,
SelectUser);
        if(!user)
            throw new HttpException('Incorrect data.', 400);
        const activationLink = await
this.activationLinkRepository.getOneByUserId(user.id);
        if(!activationLink || !activationLink.isActivated)
            throw new HttpException('Incorrect data.', 400);
        const comparePasswords = await bcrypt.compare(dto.password,
user.password);
        if(!comparePasswords)
            throw new HttpException('Incorrect data.', 400);
        const token = await this.tokenService.generateToken(user);
        if(!token)
            throw new HttpException('Error creating a token.', 400);
        await this.tokenService.saveToken(user.id, token);
        delete user.password;
        return { user, token };
}
```

Він приймає POST-запит на шлях '/login' і обробляє його. Параметр dto приймає дані запиту, які передаються в тілі запиту. Ці дані повинні бути у форматі LoginDto. loginMapper.fromControllerToService(dto) перетворює

отримані дані в формат, зручний для подальшої обробки сервісним рівнем додатка. Ймовірно, відбувається нормалізація або валідація даних. this.userRepository.getOneByEmail(dto.email, SelectUser) отримує користувача з бази даних за його електронною адресою. Параметр SelectUser вказує, які поля користувача слід включити до результату. Якщо користувача не знайдено (!user), генерується виняткова ситуація HttpException з повідомленням "Incorrect data." і статусом 400 (Bad Request). Це означає, що дані, надані користувачем, не є правильними. this.activationLinkRepository.getOneByUserId(user.id) отримує посилання активації користувача з бази даних за його ідентифікатором. Якщо посилання активації не знайдено або воно не активоване (!activationLink || !activationLink.isActivated), генерується виняткова ситуація HttpException з повідомленням "Incorrect data." і статусом 400. Це означає, що користувач не активував свій обліковий запис. bcrypt.compare(dto.password, user.password) порівнює введений користувачем пароль з захешованим паролем користувача. Якщо паролі не збігаються (!comparePasswords), генерується виняткова ситуація HttpException з повідомленням "Incorrect data." і статусом 400. Це означає, що користувач надав неправильний пароль. this.tokenService.generateToken(user) генерує токен доступу для користувача. Цей токен може використовуватися для аутентифікації та авторизації користувача на сервері. Якщо токен не вдалося згенерувати (!token), генерується виняткова ситуація HttpException з повідомленням "Error creating a token." і статусом 400. Це означає, що сталася помилка при створенні токену. this.tokenService.saveToken(user.id, token) зберігає токен у базі даних для майбутнього використання. delete user.password видаляє поле "password" з об'єкту користувача перед його поверненням у відповідь. Це зроблено для безпеки, щоб не відправляти пароль у відкритому вигляді на клієнтську сторону. На останок, метод повертає об'єкт, що містить користувача (без поля "password") та згенерований токен. Це відповідь на успішний вхід в систему.

На frontend частині перед відправкою запиту до серверу, користувач заповнює форму, у якій вводить email та пароль. Після цього відбуваються наступні дії. На лістингу 15 зображено метод логіну на клієнтській частині.

Лістинг *15 Метод логіну на клієнтській частині*

```
const login = (body: LoginDto) => {
  return async (dispatch: any) => {
      try {
          const response = await request(`${BACK_URL}/auth/login`,
'POST', body);
          if(response instanceof Error)
              throw response;
          if(response.message)
              alert(response.message);
          dispatch(setUserAction(response.user));
          localStorage.setItem('token', response.token);
          return response;
      } catch (error) {
          alert(error);
      }
  }
}
```

Відправляється “Post” запит до серверу. Після відбувається перевірки, чи успішний була запит. Якщо ні, то користувач отримує alert повідомлення про це. Якщо запит був успішний, то до клієнтського кешу додається інформація про користувача dispatch(setUserAction(response.user)) та до кешу браузера додається токен доступу localStorage.setItem('token', response.token). Після цього клієнтска частина ідентифікує цього користувача та надає більший функціонал.

Отримання новин

На backend частині отримання новин відбувається з використанням пагінації, що суттєво знижує навантаження серверу при роботі з великими об’ємами даних. На лістингу 16 зображено метод отримання новин на серверній частині.

Лістинг *16 Метод отримання новин на серверній частині*

```
@Get()
```

```
    async getAll(@Query('gr') globalRegion: string = GlobalRegionTypes.US,
@Query('limit', ToNumberPipe) limitQ: number = 9, @Query('offset',
ToNumberPipe) offsetQ: number = 0) {
        const { limit, offset } =
getNewsCardsMapper.fromControllerToService(limitQ, offsetQ);
        let newsCards = await
this.newsCardRepository.getManyFront(globalRegion, limit, offset);
        newsCards = newsCards.filter((newsCard) => {
            return newsCard.newsTexts.length > 0;
        });
        const total = await this.newsCardRepository.getTotal(globalRegion);
        return {
            newsCards,
            total
        }
}
```

У даному методі використовується декоратор @Get() для вказівки, що цей метод буде відповідати на HTTP-запити типу GET. Це означає, що коли веб-клієнт або інша програма надсилає GET-запит до відповідного URL-шляху, цей метод буде виконуватись. У параметрах методу використовуються декоратори @Query(), щоб отримати значення параметрів запиту. Наприклад, @Query('gr') globalRegion: string означає, що параметр globalRegion буде отримано з запиту як значення параметра gr. За замовчуванням, якщо значення параметра не вказано в запиті, globalRegion буде мати значення GlobalRegionTypes.US. Аналогічно, для параметрів limitQ і offsetQ використовується @Query('limit', ToNumberPipe) і @Query('offset', ToNumberPipe) відповідно. Ці декоратори також вказують, що значення параметрів повинні бути перетворені в числовий тип за допомогою ToNumberPipe. Усередині методу виконується логіка для отримання новинних карток з бази даних. Спочатку викликається метод getNewsCardsMapper.fromControllerToService(limitQ, offsetQ), який виконує маппінг значень limitQ і offsetQ до внутрішнього формату, необхідного для обробки запиту. Отримані значення використовуються для отримання новинних карток з бази даних за допомогою методу getManyFront об'єкта newsCardRepository. Отримані картки потім фільтруються, залишаючи лише ті, у яких кількість текстів новин більше нуля. Далі, за допомогою методу

getTotal об'єкта newsCardRepository, отримується загальна кількість новинних карток у базі даних для вказаного globalRegion. Нарешті, метод повертає об'єкт з двома властивостями: newsCards, яке містить отримані новинні картки після фільтрації, та total, яке містить загальну кількість новинних карток. Цей метод призначений для обробки GET-запитів до відповідного URL-шляху і повертає список новинних карток разом із загальною кількістю таких карток у базі даних для певного регіону.

На frontend частині для отримання новин достатньо перейти до сторінок “Новини”. Це можна зробити за допомогою навігаційної панелі зверху сайта.

Після переходу до сторінки клієнстька частині відправляє “GET” запит до серверу з певними параметрами пагінації (limit та offset). Якщо запит не був успішний, то користувач отримує alert повідомлення про це. Якщо запит був успішний, то до кешу клієнтської частини записується отриані новини dispatch(setNewsCardsAction(response)). Далі ці новини відображаються на сторінці користувача. На лістингу 17 зображено метод отримання новин на клієнтській частині.

Лістинг *17 Метод отримання новин на клієнтській частині*

```
const getNewsCards = useCallback((globalRegion: GlobalRegionTypes, limit:
number, offset: number) => {
        return async (dispatch: any) => {
            try {
                setReady(false);
                const response = await request(`${BACK_URL}/news-
card?gr=${globalRegion}&limit=${limit}&offset=${offset}`, 'GET');
                if(response instanceof Error)
                    throw response;
                dispatch(setNewsCardsAction(response));
                return response;
            } catch (error) {
                alert(error);
            } finally {
                setReady(true);
            }
        }
    }, [request]);
```

3.5 Вимоги до інтерфейсу

3.5.1 Вимоги до ігрового рушія “TWE”

Вимоги до загального вигляду і платформи інтерфейсу:

1. Інтерфейс повинен бути реалізований на платформі Windows.
2. Вигляд інтерфейсу повинен відповідати стандартам інтерфейсу Windows для забезпечення зручного та зрозумілого користувацького досвіду.
3. Інтерфейс повинен бути реалізований з використанням OpenGL для графічного відображення елементів.

Спеціальні вимоги до інтерфейсу:

1. Інтерфейс повинен бути інтуїтивно зрозумілим та легким у використанні.
2. Інтерфейс повинен бути адаптований для використання на різних роздільних здатностях екрану, забезпечуючи зручну навігацію та читабельність.
3. Інтерфейс повинен мати можливість налаштування графічних параметрів для підтримки різних типів обладнання та упевненості в його плавному відтворенні.

Структура інтерфейсу:

1. Головне меню: Містить початкові команди, щодо створення або відкриття проекту.
2. Панель графічного виходу: Відображає графічний вміст гри, включаючи об'єкти та оточуючий світ.
3. Панель сцени: Містить усі об'єкти, що зараз знаходяться на сцені.
4. Панель компонентів: Містить інформацію про усі компоненти обраної сутності.
5. Панель директорії: Відображає директорії поточного проекту.
6. Панель консолі: Відображаю повідомлення рушія про виконання якоїсь дії.

7. Меню бар: Містить команди для управління проектами.

3.5.2 Вимоги до сайту під ігровий рушій

Вигляд і платформа інтерфейсу: Веб-інтерфейс.

Функціональність доступна через веб-інтерфейс:

1. Головна сторінка: Завантаження ігрового рушія.
2. Сторінка новин: Перегляд новин про сайт.
3. Форма аутентифікації користувача: Вхід до системи або реєстрація нового користувача.
4. Сторінка конкретної новини: Перегляд блоків інформації про конкретну новину.

Спеціальні вимоги до інтерфейсу:

1. Інтерфейс повинен бути добре зрозумілим та інтуїтивно зрозумілим для користувачів.
2. Інтерфейс повинен бути адаптивним і реагувати на різні розміри екрану, щоб забезпечити коректне відображення на різних пристроях, включаючи комп'ютери, планшети та мобільні телефони.
3. З урахуванням цільової аудиторії, інтерфейс повинен бути зручним для використання геймерами та людьми, зацікавленими в інді-іграх.
4. Забезпечити можливість взаємодії з інтерфейсом за допомогою миші, клавіатури та сенсорних пристроїв.

Структура інтерфейсу:

1. Головна сторінка.
    - **a)** Заголовок сайту.
    - **b)** Розділ для завантаження ігрового рушія.
2. Сторінка новин.
    - **a)** Список новин з коротким описом.
    - **b)** Посилання на кожну окрему новину для переходу на сторінку конкретної новини.
3. Форма аутентифікації.

a) Поля для введення ідентифікаційних даних (логін, пароль).

b) Кнопка для входу до системи або реєстрації нового користувача.

4. Сторінка конкретної новини.
   a) Заголовок та зміст новини.
   b) Блоки інформації про новину.

3.6 Реалізація і тестування

3.6.1 Реалізація

Фізичні характеристики поточної версії системи:

Об'єм коду:

1. C++ — 812 кілобайт, ~15 300 рядків.
2. GLSL — 12 кілобайт, 266 рядків.
3. CMake — 8 кілобайт, 85 рядків.
4. TypeScript — 1150 кілобайт, ~29500 рядків.
5. SCSS — 47 кілобайт, ~1500 рядків.
6. Prisma — 4 кілобайти, 115 рядків.

Кількість модулів: 16.

Кількість і об'єм, в кілобайтах, програмних компонент:

1. Додаток “TWE” — 88400 кілобайт, 13 компонентів.
2. Сайт — 1198000 кілобайт, 12 компонентів.

Фактична швидкодія:

1. Додаток “TWE” — Так як швидкодія додатку залежить від поточної гри, запущеної у редакторі, мною було проведено заміри використовуючи вже створений проект у цьому додатку платформер “Ball”.

Таблиця *12*

Таблиця швидкодії додатку “TWE”

Стан додатку	CPU, %	RAM, MB	GPU, %	FPS
У редакторі при зупиненій грі.	~ 10.7%	~ 94 MB	~ 42%	~1000 FPS
У редакторі при запущеній грі.	~ 11%	~ 100.4 MB	~ 44%	~900 FPS
У білді гри	~ 12%	~ 83 MB	~ 55%	~1600 FPS

2. Сайт під ігровий рушій — CPU: <1%; RAM: 700 MB; GPU ~0%;

3.6.2 Апробація

Для перевірки додатку “TWE” на те, що він відповідає своїм вимогам і насправді є ігровим рушієм, мною було створено 2 гри різних жанрів за допомогою нього.

Перша гра жанру “платформер”. У цій грі ви повинні керувати м'ячем та долати перепони. Щоб завершити гру потрібно пройти усі рівні доходячи до фінішу кожного рівня. Ця гра добре відображає взаємодію фізики та як вона впливає на м'яч при переміщені по об’єктах, та при падінні з них. Також у цій грі можна побачити взаємодію джерел світла на м'яч та як відображаються тіні на об’єктах. На Рис.*13* 13 зображено гру жанру платформер “Ball”.

Рис.*13 Гра жанру платформер "Ball" (Перша локація)*

Для покращення навколишнього середовища було додано фон та звуки навколишнього середовища. Для швидкого завершення гри можна просто відкрити меню гри, натиснувши на клавішу "Tab", та натиснути на кнопку "Exit". На Рис.*14* 14 можна побачити вже іншу локацію гри жанру платформер "Ball".

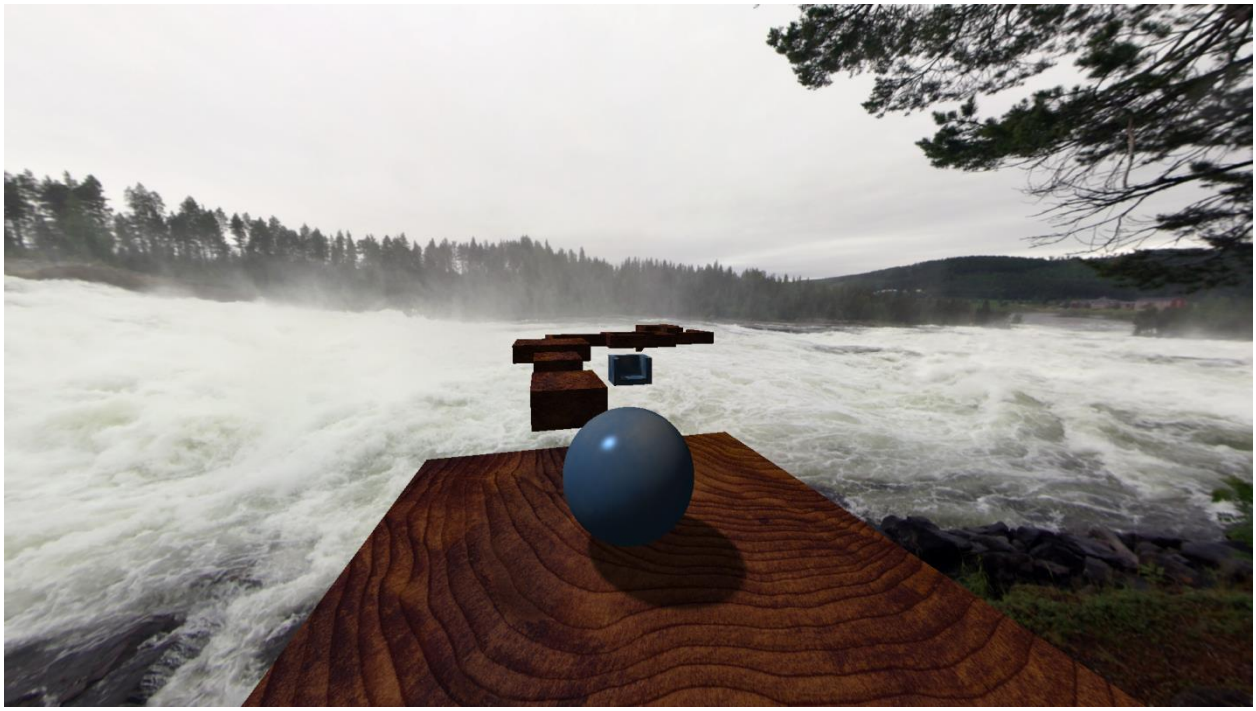

Рис.*14 Гра жанру платформер "Ball" (Друга локація)*

Друга гра жанру "Aim Trainer". У цій грі ви знаходитесь у тиру з повітряними кульками, які ви повинні лопати за допомогою іграшкового пістолету. На Рис.*15* 15 зображено початкове меню гри жанру aim trainer "Shooting Range".

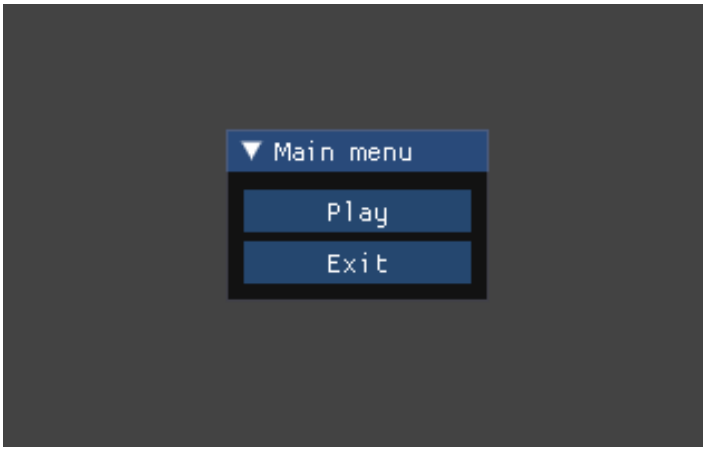

Рис.*15 Меню гри жанру aim trainer "Shooting Range"*

Гра не має завершення, зате вона допомагає покращити реакцію гравця та його володіння управління мишою. У цій грі також можна побачити взаємодію фізики на кульки, так як лопання кульок реалізовона за допомогою

кидання променів, які перетинають кульки та лопають їх. При лопані кульок вони видають звук лопання. Гра містить музику на фоні, що покращує атмосферу гри. Також можна побачити роботу з импортованими моделями (іграшковий пістолет, кульки, колесо огляду). Ця гра також містить в собі меню, яке надає змогу покинути гру. На Рис.*16* 16 зображено гру жанру aim trainer “Shooting Range”.

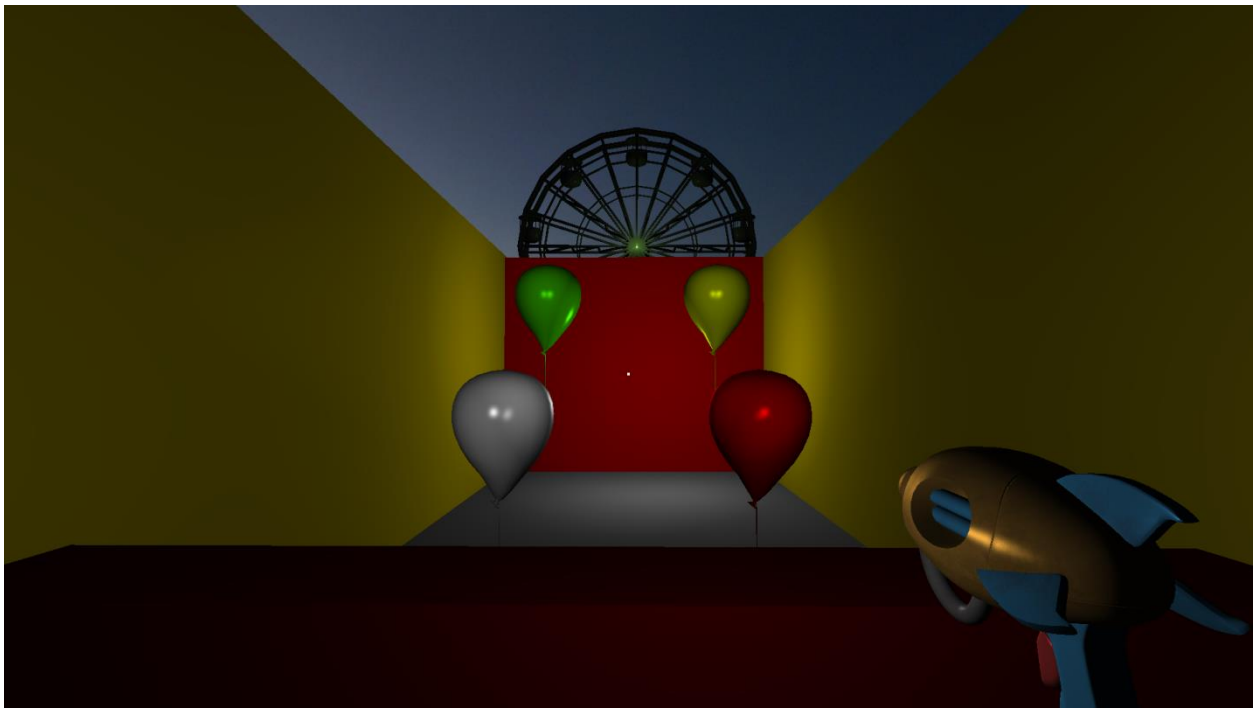

Рис.*16 Гра жанру aim trainer “Shooting Range”*

3.7 Висновки до розділу 3

1. Архітектура ігрового рушія “TWE” складається з двох головних шарів - рушій та редактор та зв’язуючого шару - контроллер. Сайту під ігровий рушій має клієнт-серверну архітектура.
2. Були визначені загальні вимоги, щодо реалізації ігрового рушія “TWE” та сайту під ігровий рушій. Були використані бібліотеки підпрограм, що використовувались при розробці додатку “TWE” та сайту під ігровий рушій.

3. Був розроблений функціонал додатку “TWE” та сайту під ігровий рушій.
4. Визначені основні вимоги до інтерфейсу ігрового рушія “TWE” та сайту під ігровий рушій. Було створено структуру інтерфейсу ігрового рушія TWE” та сайту під ігровий рушій.
5. Розглянута реалізація ігрового рушія “TWE” та сайту під ігровий рушій. Створено 2 гри різних жанрів: платформер та aim trainer, для перевірки відповідності вимогам ігрового рушія.

ВИСНОВКИ

1. Результати аналізу доступних інформаційних матеріалів та наукових досліджень свідчать про те, що велика увага приділяється створенню доступних, потужних і легко використовуваних рушіїв, що дозволяють незалежним розробникам створювати високоякісні ігри.
2. Проведено порівняльний аналіз існуючих ігрових рушіїв на ринку: Unreal Engine, Unity, CryEngine. Визначено їх плюси та мінуси, вивчено їх особливості. Було опрацьовано літературні джерела, що стосуються розробки ігрових рушіїв.
3. Було розроблено архітектуру системи, визначено основні компоненти рушія та їх взаємодію, а також визначено основні функціональні вимоги до рушія.
4. В результаті виконання кваліфікаційної роботи було створено ігровий рушій для інді-ігор. Двигун був розроблений з використанням технологій індустрії геймдеву, зокрема використана популярна інструментальна платформа OpenGL та мова програмування C++.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Making Your Own Video Game Engine: The Beginners Guide. URL: https://www.gamedesigning.org/learn/make-a-game-engine/#:~:text=Making%20a%20game%20engine%20isn,into%20your%20lap%20more%20easily. (дата звернення: 07.05.2023)
2. What is an Indie Game and Why is It So Popular. URL: https://blog.getsocial.im/what-is-an-indie-game-and-why-is-it-so-popular/ (дата звернення: 07.05.2023)
3. Real-Time Rendering Resources. URL: https://www.realtimerendering.com/#intro (дата звернення: 07.05.2023)
4. Learn OpenGL, Learn modern OpenGL graphics programming in a step-by-step fashion by Joey de Vries, Kendall & Welling, June 17, 2020. 254 с.
5. Learning Game Physics with Bullet Physics and OpenGL by Chris Dickinson, Packt Publishing, October 25, 2013. 81 с.
6. irrKlang, high level audio engine. URL: https://www.ambiera.com/irrklang/index.html (дата звернення: 14.05.2023)
7. dear imgui. URL: https://imgui-test.readthedocs.io/en/latest/#how-it-works (дата звернення: 14.05.2023)
8. Unreal Engine: what beginners need to know about software, on which masterpieces are created. URL: https://vokigames.com/unreal-engine-what-beginners-need-to-know-about-software-on-which-masterpieces-are-created/ (дата звернення: 07.05.2023)
9. Unity - What makes it the best game engine? URL: https://kevurugames.com/blog/unity-what-makes-it-the-best-game-engine/ (дата звернення: 07.05.2023)
10. What is the best game engine: is CryEngine right for you? URL: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-cryengine-the-right-game-engine-for-you (дата звернення: 07.05.2023)

11. Game Engine Architecture, Third Edition by Jason Gregory, A K Peters/CRC Press, August 17, 2018. 537 с.
12. Client/Server Architecture (J. Ranade Series on Computer Communications) by Alex Berson, Mcgraw-Hill (Tx), January 1, 1992. 20 с.
13. Challenges faced by indie game making. URL: https://marbleit.rs/blog/challenges-of-indie-game-making/ (дата звернення: 20.05.2023)
14. How to make your own game engine (and why). URL: https://medium.com/geekculture/how-to-make-your-own-game-engine-and-why-ddf0acbc5f3 (дата звернення: 03.06.2023)
15. OpenGL-Performance and Bottlenecks, by Matthias Trapp. URL: https://www.researchgate.net/publication/255651009_OpenGL-Performance_and_Bottlenecks (дата звернення: 03.06.2023)