Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Бакалаврська кваліфікаційна робота

на тему
«Комп'ютерна гра-головоломка на ігровому рушії Unity»
08-54.БКР.002.00.000 ПЗ

Виконав: студент 4 курсу, групи 1КІ-21б
Спеціальності 123 – «Комп`ютерна інженерія»
Освітня програма – «Комп`ютерна інженерія»
Бондар А.М
Керівник phd, ст, виклад, каф. ОТ
Обертюх М.Р
Рецензент к.т.н., доцент, доцент каф.ПЗ
Ракитянська Г.Б

Допущено до захисту
Зав. кафедрою ОТ
д.т.н., проф. Азаров О.Д.

«20» 06 2025 р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо — кваліфікаційний рівень бакалавр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. _____О.Д. Азаров
«21» березня 2025 року

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

Бондар Артур Михайлович

1 Тема роботи: Комп'ютерна гра-головоломка на ігровому рушії Unity, Керівник проєкту Обертюх Максим Романович, викладач кафедри ОТ, затверджені наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Строк подання студентом роботи 13.05.2025

3 Вихідні дані до роботи: Розробка гри на ігровому рушії Unity, створити проект головоломку з унікальними механіками, меню, і відповідними налаштуваннями.

4 Зміст розрахунково-пояснювальної вступ; обґрунтування технічного завдання та постановка задачі дослідження; аналіз засобів проектування програмного продукту; розробка програмних компонентів; тестування програми; висновки; список використаних джерел

5 Перелік графічного матеріалу: UML-діаграма класової структури скриптів; послідовність оновлення положення та орієнтації камери через CameraController; послідовність завантаження та ініціалізації рівня; послідовність відтворення звукового ефекту через SoundManager; діаграма переходів між UI-панелями; послідовність викликів для збереження та завантаження гри; діаграма послідовності обробки TriggerZone; діаграма послідовності обробки Idle.

6 Консультанти розділів роботи наведено в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Обертюх М.Р., викладач кафедри ОТ	21.03.25	13.05.25
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.25	30.05.25

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план приведений в таблиці 2.

Таблиця 2 — Календарний план

Ч.ч.	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз технічного завдання	21.03.25	Виконав
2	Вступ, техніко-економічне обґрунтування технічного завдання	21.03.25— 30.03.25	Виконав
3	Вибір технологій та розробка гри	31.03.25— 10.04.25	Виконав
4	Розробка технічного обґрунтування технічного завдання та розділу висновки	11.04.25— 21.04.25	Виконав
5	Проектна частина	22.04.25— 30.04.25	Виконав
6	Оцінка результатів та перспективи розвитку	31.04.25— 02.05.25	Виконав
7	Оформлення пояснювальної записки та ілюстративного матеріалу	03.05.25— 11.05.25	Виконав
8	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	Виконав

Студент Артур БОНДАР

Керівник роботи Максим ОБЕРТЮХ

АНОТАЦІЯ

УДК 004.3

Бондар А.М. Комп’ютерна гра-головоломка на ігровому рушії Unity. Бакалаврська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна інженерія, освітня програма — Комп’ютерна інженерія. Вінниця: ВНТУ, 2025. 70 с.

На укр. мові. Бібліогр.: 26 назв; рис.: 26; табл.: 3.

У бакалаврській роботі було створено комп’ютерну гру-головоломку на рушії Unity із застосуванням мови C#. Основна ігрова механіка — зміна розмірів об’єктів, що впливає на їхню фізичну поведінку. Це стимулює логічне мислення гравця та забезпечує варіативність рішень. Для покращення графіки використовувалися шейдери, що підвищили реалістичність візуалу та продуктивність гри. Відмова від сторонніх плагінів дозволила досягти високої оптимізації та стабільної роботи навіть на середніх пристроях. Гра має інноваційний підхід до геймплею та може бути використана як для розваги, так і для розвитку когнітивних навичок.

Ключові слова: Unity, C#, шейдери, головоломка, геймдизайн, інтерфейс користувача, анімація, візуальні ефекти, освітлення, управління сценами.

ABSTRACT

Bondar A. M. Computer Puzzle Game on the Unity Game Engine. Bachelor's Qualification Bachelor's qualification work in specialty 123 — Computer Engineering, educational program — Computer Engineering. Vinnytsia: VNTU, 2025. 70 pp.

In Ukrainian. Bibliography: 26 titles; fig: 26; tab.: 3.

In the bachelor's thesis, a computer puzzle game was created on the Unity engine using the C# language. The main game mechanics are changing the size of objects, which affects their physical behavior. This stimulates the player's logical thinking and provides variability in solutions. Shaders were used to improve the graphics, which increased the realism of the visuals and the performance of the game. The rejection of third-party plugins made it possible to achieve high optimization and stable operation even on medium-sized devices. The game has an innovative approach to gameplay and can be used both for entertainment and for the development of cognitive skills.

Keywords: Unity, C#, shaders, puzzle, game design, user interface, animation, visual effects, lighting, scene management.

ЗМІСТ

ВСТУП

Актуальність теми полягає у необхідності створення інноваційного ігрового продукту, що поєднує у собі нестандартні механіки (зокрема, зміну фізичних властивостей і розмірів об’єктів у реальному часі), захопливий сюжет, якісний дизайн рівнів та орієнтацію на широку геймерську аудиторію. Це відповідає сучасним тенденціям у галузі розробки ігор, а також сприяє підвищенню інтересу до вітчизняного геймдеву.

Метою є — створення інноваційної інді гри нарушії Unity з унікальною ігровою механікою, яка дозволяє змінювати фізичні властивості об’єктів у реальному часі.

Основні завдання:

— опрацювання теоретичних аспектів розробки комп’ютерних ігор;

— проєктування архітектури гри;

— реалізація основних механік за допомогою C# та інструментів Unity;

— оптимізація продуктивності з використанням шейдерів;

— впровадження сюжетної лінії та рівнів;

— тестування гри на різних пристроях.

Індустрія комп’ютерних ігор є однією з найдинамічніших галузей цифрових технологій. Завдяки стрімкому розвитку апаратного забезпечення, програмних засобів та зростаючому попиту з боку користувачів, відеоігри перетворилися з простої форми розваги на потужний інструмент комунікації, навчання, моделювання та творчої самореалізації. Особливої популярності набувають інді-ігри — незалежні проєкти, які часто вирізняються креативністю, інноваційними ідеями та глибоким опрацюванням геймплейних і сюжетних елементів.

Станом на сьогодні ігрова індустрія в Україні перебуває на етапі активного розвитку, попри певне відставання у темпах зростання порівняно з

провідними світовими ринками. Це створює сприятливе середовище для реалізації нових проєктів, що орієнтовані на міжнародну аудиторію. Розробка інді-гри на базі сучасного ігрового рушія Unity з використанням мови програмування C# є важливим кроком у напрямку формування професійних компетентностей розробника, а також у популяризації українських ігрових продуктів.

Об’єкт дослідження — процес розробки комп’ютерних ігор на базі платформи Unity.

Предмет дослідження — методи та засоби реалізації інтерактивної ігрової механіки, побудови ігрових рівнів і структурування сюжетної лінії в межах розробки інді-гри мовою програмування C# на ігровому рушії Unity.

1 ОБҐРУНТУВАННЯ ТЕХНІЧНОГО ЗАВДАННЯ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Огляд напрямку дослідження

Ігрова індустрія є одним із найдинамічніших секторів цифрових технологій, що за останні десятиліття перетворилась із вузькоспеціалізованої ніші для ентузіастів на потужну галузь з мільярдними прибутками, масштабною аудиторією та високими стандартами якості. Відеоігри вже давно не сприймаються лише як розвага. Це інструмент комунікації, навчання, дослідження, соціальної взаємодії та самовираження.

Розвиток нових графічних технологій, зростання обчислювальної потужності комп’ютерів і поява зручних інструментів розробки, таких як ігровий рушій Unity, відкрили нові можливості для створення ігор різного рівня складності — від великих AAA-проєктів до незалежних (інді) розробок. Саме в сегменті інді-ігор особливо активно розвиваються нові ідеї, нестандартні ігрові механіки, оригінальний дизайн і глибокий смисловий зміст.

Сьогоднішні гравці очікують від ігор не лише привабливої графіки, а й високого рівня інтерактивності, гнучкості в налаштуванні, логічно побудованої структури рівнів, а також можливості зануритися в унікальний віртуальний світ. Це визначає високі вимоги до процесу розробки ігрового продукту.

У рамках даної бакалаврської дипломної роботи поставлено завдання створити інноваційний ігровий проєкт на платформі Unity, який базується на фізичних взаємодіях об'єктів та зміні їх властивостей — як основі ігрової механіки. Такий підхід не лише додає інтересу до ігрового процесу, а й надає простір для досліджень і навчання. Проєкт передбачає інтеграцію модульних інструментів для тонкого налаштування фізичних параметрів об’єктів, що забезпечить глибоке занурення та індивідуалізацію ігрового процесу.

1.2 Актуальність теми

На сучасному етапі цифрової трансформації розробка ігор стала актуальною не лише з точки зору бізнесу, але й як важливий елемент освітньої, культурної та наукової екосистеми. У глобальному контексті комп'ютерні ігри вже давно є платформою для соціального діалогу, експериментів, мистецтва та глибокої рефлексії. У країнах із розвиненою індустрією геймінгу, такі як США, Японія, Південна Корея та Німеччина, ігри мають підтримку на рівні державної політики, освіти і наукових досліджень.

В Україні, попри наявність талановитих фахівців і студій, геймдев ще перебуває на етапі активного формування. Водночас попит на високоякісний локальний продукт зростає. Це обумовлює необхідність підготовки нової генерації розробників, які володіють сучасними технологіями, інструментами й підходами до створення цифрового контенту.

Створення гри з унікальною фізичною механікою дозволяє поєднати в одному продукті одразу декілька перспективних напрямів: розробку, програмування, 3D-моделювання, інтерактивний дизайн, тестування, геймдизайн, а також елементи симуляції і навчання. Отже, тема дипломної роботи відповідає сучасним тенденціям ринку та є надзвичайно актуальною як в академічному, так і в професійному контексті.

1.3 Мета та завдання дослідження

Розробити інтерактивну гру на рушії Unity з використанням C#, в основі якої лежить зміна фізичних властивостей об'єктів (розмір, маса, взаємодія з середовищем), а також створення логічно побудованих рівнів, що стимулюють гравця до критичного мислення, пошуку рішень і навчання через досвід.

Основні завдання, які необхідно реалізувати в рамках проєкту:

— дослідити основні принципи побудови ігрових механік на основі фізичних взаємодій;

— розробити структуру гри та дизайн рівнів, у яких поступово ускладнюється ігровий процес;

— створити ігрове середовище з урахуванням особливостей візуалізації та оптимізації;

— інтегрувати шейдери та механізми зміни властивостей об’єктів у реальному часі;

— забезпечити інтуїтивно зрозумілий інтерфейс користувача;

— здійснити тестування гри з використанням вбудованих інструментів Unity та сторонніх засобів (наприклад, GPU Visualizer);

— проаналізувати результати тестування та внести необхідні покращення до функціоналу гри.

1.4 Аналіз аналогів та порівняння з іншими іграми

З метою обґрунтування доцільності та унікальності запропонованого ігрового проєкту було проведено аналіз існуючих ігор, які використовують подібні ігрові механіки або мають схожі концептуальні підходи. Такий аналіз дозволяє не лише виявити спільні риси, але й окреслити унікальні особливості розроблюваного продукту, а також врахувати недоліки конкурентів з метою їх подальшого уникнення. Серед подібних ігор можна навести такі приклади.

Human Fall Flat. У цій грі гравець керує гуманоїдом із "розслабленою фізикою" та розв’язує фізичні головоломки у відкритому середовищі. Основною особливістю є непередбачувана фізична модель та орієнтація на експериментальний підхід до розв’язання задач. Обидві гри використовують фізичні рушії як основу геймплею. У нашому проєкті фізика також є ключовим елементом, проте реалізована вона інакше — через зміну властивостей об’єктів, таких як масштаб, щільність і взаємодія з іншими тілами. Це дає змогу застосовувати логічні та раціональні стратегії, на відміну від імпровізацій, що ґрунтуються на хаотичних діях персонажа в аналізованій грі.

Size Matters. У цьому проєкті гравець виступає в ролі науковця, який поступово зменшується й має встигнути знайти ліки до повного зникнення. Механіка зміни розміру безпосередньо впливає на швидкість руху, взаємодію з об’єктами та доступ до них. Це одна з ігор, що найбільше наближена до концепції розроблюваного проєкту. В обох випадках зміна масштабу відіграє центральну роль, однак у нашій грі ця механіка поширюється не лише на головного героя, а й на інші об’єкти, що суттєво розширює можливості побудови рівнів і створює більш різноманітні умови проходження.

Poly Bridge. Ця гра зосереджена на побудові мостів з урахуванням обмеженого бюджету, типу матеріалів та фізичних навантажень. Вона сприяє формуванню інженерного мислення та розумінню базових фізичних принципів шляхом практичного моделювання. Обидва проєкти мають освітній потенціал та базуються на засвоєнні фізичних закономірностей через ігровий досвід. Проте у нашій грі структура задач є менш формалізованою, а спектр взаємодії з об’єктами ширший, що дозволяє моделювати більш комплексні ситуації.

Besiege. Гравець у цій грі конструює бойові машини, використовуючи реалістичну фізику та інженерні принципи для розв’язання поставлених задач. Основну увагу зосереджено на творчості, експериментах із конструкціями та досягненні мети через самостійне проєктування. Хоча наш проєкт не є конструктором, між іграми простежується спільна риса — інтерактивність середовища та реакція об’єктів на дії гравця. Замість створення механізмів, у нашій грі реалізовано можливість змінювати властивості об’єктів, що також спонукає до експериментів і самостійного пошуку ефективних рішень.

2 АНАЛІЗ ЗАСОБІВ ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Ігровий рушій Unity

Unity — потужний та універсальний багатоплатформовий інструмент, призначений для розробки відеоігор та інтерактивних застосунків різного рівня складності. Цей продукт поєднує в собі зручне середовище розробки (IDE) та високопродуктивний ігровий рушій, що забезпечує можливість створення, тестування та запуску проектів у єдиному середовищі. Unity підтримує розробку як двовимірних (2D), так і тривимірних (3D) ігор, а також дозволяє створювати рішення для віртуальної (VR) та доповненої реальності (AR), що відкриває широкий спектр можливостей для креативного застосування технологій.

Платформи, на яких можна запускати продукти, розроблені в Unity, включають різноманітні операційні системи і пристрої: від настільних комп’ютерів на базі Windows, macOS та Linux до мобільних гаджетів на iOS та Android, а також популярних ігрових консолей, таких як PlayStation, Xbox і Nintendo Switch. Крім того, Unity підтримує розгортання вебдодатків через WebGL, що робить ігри та інтерактивні проєкти доступними безпосередньо в браузері, а також сумісний із спеціалізованими VR/AR-платформами, що підсилює інтерактивність та занурення користувача у віртуальні світи.

Технічна основа Unity включає підтримку сучасних технологій рендерингу, серед яких DirectX, OpenGL, Vulkan — дозволяє використовувати максимально ефективні графічні можливості різних пристроїв, забезпечуючи високу якість зображення та продуктивність. Завдяки цьому розробники можуть створювати візуально привабливі ігри з плавною анімацією та реалістичними ефектами, адаптовані під апаратні ресурси конкретної платформи.

Однією з найбільш вагомих переваг Unity є його доступність для

новачків. Інтуїтивно зрозумілий інтерфейс і багатофункціональні інструменти дозволяють починати роботу навіть користувачам без глибоких знань у галузі програмування або комп'ютерної графіки. Архітектура Unity базується на компонентно-орієнтованому підході, що дає можливість створювати ігрові об'єкти як набір окремих модулів — компонентів, які додають необхідний функціонал через просте додавання та налаштування. Ця гнучкість сприяє швидкому прототипуванню, ефективній організації коду та легкому масштабуванню проекту.

Завдяки можливості швидкого тестування ігрових механік прямо в середовищі розробки, а також оперативного внесення змін, Unity дозволяє оптимізувати процес створення ігор, скорочуючи час від задуму до реалізації. В результаті розробники отримують потужний інструмент, що підтримує як індивідуальні творчі проекти, так і масштабні комерційні продукти, здатні працювати на багатьох платформах з високою продуктивністю та якістю.

Ще однією значною перевагою є велика екосистема: Unity має власний маркетплейс — «Asset Store», де доступні тисячі безкоштовних і платних ресурсів — моделей, текстур, анімацій, ефектів, плагінів та інструментів, що значно пришвидшує розробку ігор. Також Unity підтримує інтеграцію зі сторонніми сервісами для аналітики, монетизації, мережевих функцій тощо.

Кросплатформеність — ще один важливий плюс Unity. Розробники можуть створювати один і той самий проєкт і публікувати його на десятках платформ без потреби в серйозних модифікаціях, що значно спрощує вихід продукту на ринок.

Проте, у Unity є і певні обмеження. Під час створення масштабних або технічно складних проєктів виникає необхідність у залученні досвідчених програмістів, зокрема з хорошим знанням мови C#. Саме на цій мові пишуться скрипти, що керують логікою гри, взаємодією об'єктів, а також оптимізацією продуктивності.

У процесі створення ігор важливу роль відіграє комп’ютерна графіка. Існує кілька її основних типів:

— растрова графіка використовується найчастіше для зображень, що складаються з пікселів. Вона широко застосовується у розробці мультимедійного контенту, в тому числі для текстур у іграх, і поліграфії;

— векторна графіка базується на математичних формулах і використовується для створення масштабованих елементів інтерфейсу або 2D-анімацій;

— фрактальна графіка менш поширена в ігровій індустрії, однак використовується у генерації природних об’єктів, ландшафтів або візуалізації складних математичних структур.

Отже, Unity 3D (рис. 2.1) — універсальне середовище, яке підходить як для аматорських проєктів, так і для професійної комерційної розробки, з широкими можливостями для творчості, експериментів та реалізації інноваційних ігрових ідей [1].

На відміну від растрової графіки, де зображення формується шляхом розміщення окремих кольорових пікселів у вигляді сітки, векторна графіка будується з геометричних фігур — так званих **графічних примітивів**. До них належать прямі лінії, криві, дуги, кола, еліпси, багатокутники, прямокутники тощо. Розміщення, розміри та форма цих елементів задаються у координатній системі, що зазвичай прив’язана до екрану або робочої області. Завдяки математичній природі опису, векторна графіка легко масштабується без втрати якості, що робить її ідеальною для створення логотипів, іконок, схем та UI-елементів у програмуванні та дизайні.

Інтерфейс Unity 3D зображено на (рис. 2.1), де показано основні елементи середовища розробки.

Фрактальна графіка, як і векторна, базується на математичних розрахунках, однак має зовсім інший принцип побудови. Вона створюється на

основі математичних формул, які описують повторювані, самоподібні структури — фрактали.

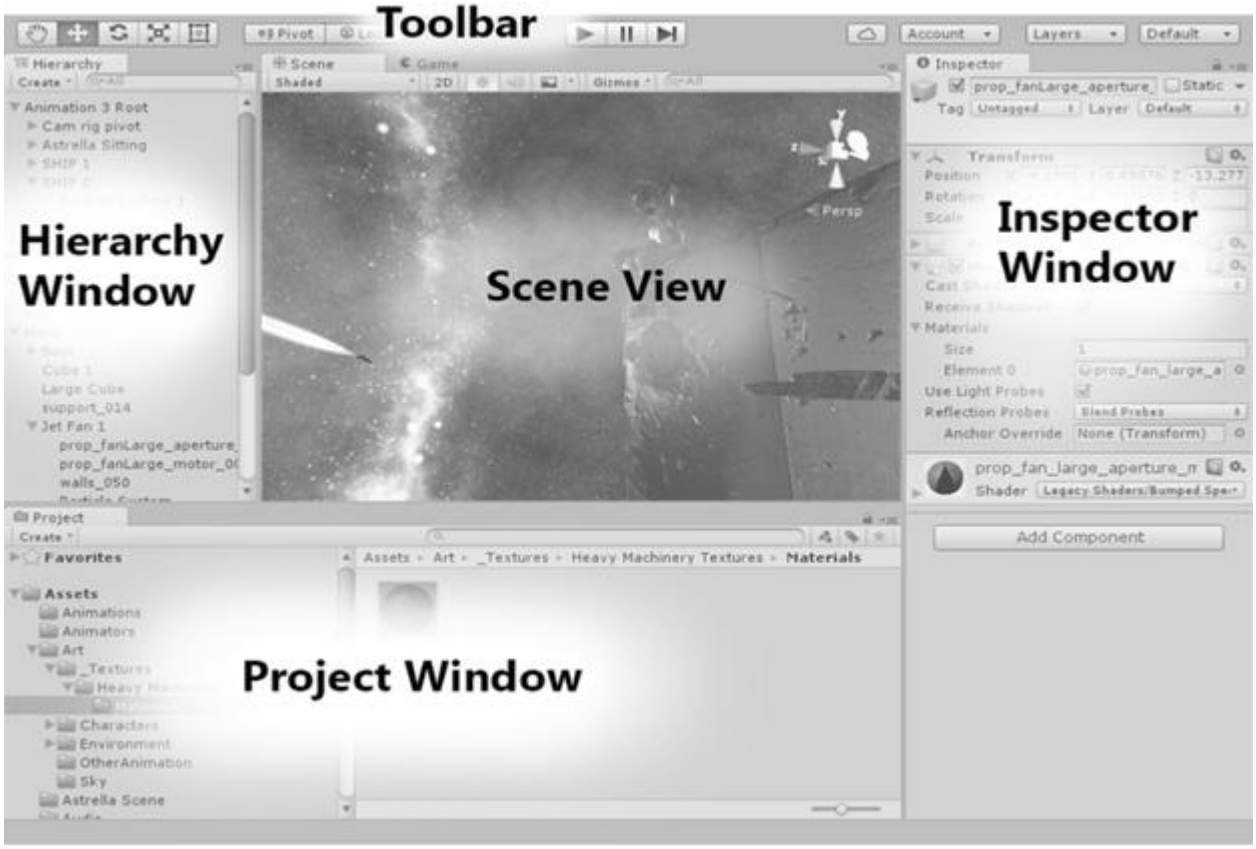

Рисунок 2.1 – Інтерфейс програми Unity3D

Фрактальна графіка, як і векторна, базується на математичних розрахунках, однак має зовсім інший принцип побудови. Вона створюється на основі математичних формул, які описують повторювані, самоподібні структури — фрактали. На відміну від векторної та растрової графіки, у фрактальній графіці в пам’яті не зберігається саме зображення чи об’єкти — зберігається лише формула і набір параметрів. При зміні цих коефіцієнтів в рівняннях можна отримати нові, часто візуально складні й унікальні структури, які раніше не були збережені або створені вручну.

Фрактальна графіка використовується в комп’ютерному моделюванні для генерації природних пейзажів, хмар, дерев, гір, берегових ліній, візерунків та навіть у науковій візуалізації. Її головна перевага — можливість створювати

складні, деталізовані зображення з мінімальними ресурсними затратами, використовуючи лише математичну формулу та параметри її побудови.

У розробці ігор та мультимедійних застосунків часто використовують комбінацію цих типів графіки для досягнення оптимальної якості, продуктивності та візуального стилю [1].

2.2 Головні елементи управління Unity

Панель інструментів в Unity надає швидкий доступ до основних функцій управління редактором. Зліва розташовані ключові інструменти для навігації по сцені та взаємодії з об'єктами типу GameObject. У центральній частині знаходяться елементи керування запуском гри: кнопки «Відтворити», «Пауза» та «Покадрове відтворення», які дозволяють тестувати поведінку гри в реальному часі. Праворуч розташовані кнопки доступу до Unity Collaborate, хмарних сервісів Unity Cloud Services, особистого облікового запису користувача, а також меню налаштувань видимості шарів (Layers) і вибору макету редактора. Останнє дозволяє перемикатися між заздалегідь налаштованими схемами розміщення вікон або створювати та зберігати власні конфігурації інтерфейсу.

Вікно Ієрархії (Hierarchy) відображає у вигляді списку всі об'єкти, присутні в поточній сцені. Воно демонструє структуру підпорядкування між GameObjects, дозволяючи зрозуміти, які об'єкти є батьківськими, а які - дочірніми. Вікно ієрархії тісно пов'язане з переглядом сцени: будь-який об'єкт, що існує в сцені, автоматично відображається в цьому списку.

Вікно Game View (Перегляд гри) моделює зовнішній вигляд фінальної версії гри з точки зору камери. Тут ви бачите, як гра виглядатиме для користувача. При натисканні кнопки «Play» запускається симуляція, під час якої відтворюється ігрова логіка в реальному часі.

Scene View (Перегляд сцени) — основне середовище для редагування та

створення сцен. Воно дозволяє вільно переміщуватися в просторі та взаємодіяти з об'єктами, редагуючи їхню позицію, масштаб, обертання тощо. Залежно від типу проєкту, вигляд сцени може переключатися між 2D та 3D режимами.

Inspector (Інспектор) — вікно, яке показує властивості вибраного об'єкта. Саме тут можна змінювати параметри компонентів, додавати нові або видаляти непотрібні. Зміст інспектора змінюється динамічно в залежності від типу виділеного об'єкта, що дозволяє гнучко керувати налаштуваннями кожного GameObject у сцені.

Ці інструменти в сукупності створюють зручне середовище для розробки ігор, дозволяючи контролювати як візуальну частину, так і логіку проєкту на всіх етапах створення [1].

2.3 Ігрові об’єкти і компоненти

Ігрові об’єкти (GameObjects) є базовими елементами, з яких будується будь-який проєкт в Unity. Практично все, що ми бачимо у грі — персонажі, декорації, ефекти — створюється на основі GameObject. Проте сам по собі GameObject не виконує жодної функції — лише основа, яка набуває функціональності завдяки доданим до неї компонентам (Components).

GameObject можна уявити як порожній контейнер, який наповнюється різними складовими залежно від мети. Наприклад, цей контейнер може стати як інтерактивним об’єктом, так і просто частиною декорації, якщо додати відповідні компоненти. Тип об’єкта, який ми хочемо отримати, визначає набір компонентів, що додаються: фізика, рендеринг, колізії, анімація тощо.

Кожен GameObject обов’язково має компонент Transform, який визначає його положення, обертання та масштаб у тривимірному (або двовимірному) просторі сцени. Без Transform об’єкт не матиме координат у просторі, а отже — не існуватиме як візуальний чи фізичний елемент у грі. Transform є

фундаментальним компонентом, без якого неможливе функціонування GameObject в Unity.

Розробка у Unity передбачає зручне маніпулювання GameObject прямо в сцені: можна додавати нові компоненти, редагувати їх властивості через вікно інспектора (Inspector) та спостерігати зміни в режимі реального часу. Однак, коли мова йде про об'єкти, які часто повторюються (наприклад, NPC або будівлі), виникає потреба у створенні багаторазових шаблонів.

Для цього в Unity передбачено використання Prefabs - спеціального типу ресурсів, що дозволяє зберегти повну конфігурацію GameObject разом з усіма компонентами та їхніми налаштуваннями. Prefab слугує шаблоном, на основі якого можна створювати безліч однакових екземплярів. Головна перевага Prefabs — централізоване оновлення: змінивши базовий префаб, зміни автоматично застосовуються до всіх його копій у сценах. При цьому кожен екземпляр може мати індивідуальні налаштування — наприклад, змінений колір, позицію чи поведінку, зберігаючи зв’язок з оригіналом.

Слід зазначити, що хоча Prefab зберігається у вигляді файлу у вигляді asset, Asset сам по собі не є префабом. Його не можна напряму змінити або оснастити компонентами, як це можливо з Prefab. Prefab — динамічний, редагований шаблон, у той час як звичайні Asset-файли (наприклад, зображення або моделі) виконують роль ресурсу для подальшого використання в об’єктах [2].

Наприклад, у грі є багато інтерактивних об'єктів, таких як коробки, важелі, двері або кнопки, які мають однакову логіку взаємодії, але розміщуються на різних рівнях гри. Замість того щоб вручну створювати кожен об’єкт заново, розробник створює Prefab кнопки з усіма потрібними компонентами: Collider для виявлення натискання, скрипт взаємодії, анімацію натискання, звуковий ефект та інші параметри.

Після цього такий Prefab можна перетягувати у сцену, змінюючи лише

необхідні параметри — наприклад, колір, розмір чи швидкість спрацювання — без порушення загальної структури. Якщо розробник вирішить пізніше, що кнопка має, скажімо, світитися при наведенні, йому достатньо внести зміни один раз у Prefab, і це оновлення автоматично застосується до всіх кнопок у всіх сценах гри.

Це значно економить час, уніфікує логіку гри та зменшує ймовірність помилок, що є особливо важливим при масштабуванні проєкту.

2.4 Налаштування ігрових об’єктів

Щоб створити Prefab, необхідно перейти до Asset > Create > Prefab або просто перетягнути об’єкт зі сцени у відповідну папку в вікні Project, після чого в проєкті з’явиться новий Prefab. Надалі створювати екземпляри цього об’єкта дуже просто — достатньо перетягнути його зі списку Assets безпосередньо в сцену. У вікні Hierarchy такі об'єкти будуть позначатися синім кольором, тоді як звичайні об'єкти мають чорне маркування.

Як вже згадувалося, зміни, внесені до Prefab, автоматично застосовуються до всіх його копій у сцені. Водночас кожен екземпляр можна налаштувати індивідуально — наприклад, змінити матеріал, розміри чи інші параметри. Це особливо корисно, коли потрібно створити групу схожих об'єктів, таких як NPC з однаковою поведінкою, але різною зовнішністю для більшої правдоподібності.

Щоб відрізнити змінені властивості у конкретному екземплярі Prefab, Unity виділяє їх жирним шрифтом в Inspector. Якщо до екземпляра додається новий компонент, усі його поля також будуть позначені жирним — сигналізує, що зміни стосуються тільки цього конкретного екземпляра, а не оригінального Prefab [3].

Prefab-и особливо корисні при створенні об'єктів, які мають однакову логіку поведінки, керовану скриптами. Наприклад, можна створити Prefab

ворога, до якого буде прикріплений скрипт EnemyAI.cs, що відповідає за пересування, атаки або взаємодію з гравцем. Якщо потрібно внести зміни в логіку — достатньо відредагувати скрипт або сам Prefab, і зміни автоматично застосуються до всіх ворогів у сцені.

Ще один важливий випадок — анімація. Наприклад, якщо у вас є анімований персонаж або об'єкт, до якого прикріплений компонент Animator, ви можете зберегти його як Prefab з уже налаштованим аніматорним контролером (Animator Controller), в якому задані анімаційні стани (біг, стрибок, атака тощо). Потім ви просто створюєте нові екземпляри цього об’єкта в різних частинах сцени або навіть під час гри через скрипт (метод Instantiate), і всі вони працюють згідно заданої анімації.

Таким чином, Prefab є потужним інструментом повторного використання коду, графіки та логіки, значно пришвидшуючи процес розробки та зменшуючи ймовірність помилок

2.5 Мова програмування C#

Рушій Unity підтримує використання кількох мов програмування (зокрема C#, UnityScript, Boo тощо), проте на практиці основною мовою для розробки є C#, оскільки вона отримує найповнішу підтримку та найактуальніші оновлення. Це дозволяє розробникам працювати в звичному середовищі без необхідності освоювати нову мову з нуля.

У межах реалізації мого проєкту було обрано саме C# як основну мову програмування. Такий вибір обумовлений не лише тим, що я вже маю досвід роботи з цією мовою, а й тим, що C# забезпечує широкі можливості для створення функціонального та гнучкого коду. Вона є потужною об'єктно-орієнтованою мовою, що також підтримує компонентно-орієнтовану парадигму програмування — ідеально сумісну з архітектурою Unity, де вся логіка базується на компонентах.

C# має багато спільного з іншими мовами родини C-подібних (таких як C, C++ і Java), тому її синтаксис є інтуїтивно зрозумілим для тих, хто вже знайомий із цими мовами. Вона дозволяє створювати сучасні програмні компоненти у вигляді автономних модулів, що реалізують окремі функції застосунку. Кожен компонент у C# може містити властивості, методи, події та атрибути, які додають декларативну метаінформацію. Крім того, можливість автоматичного документування значно спрощує підтримку та масштабування проєкту.

Основною мовою програмування, яка використовується в Unity, є C#. Ця мова була обрана для реалізації проєкту з кількох причин:

— знання та досвід використання C# є знайомою для автора мовою, що дозволяє уникнути витрат часу на її вивчення;

— об’єктно-орієнтована модель C# повністю підтримує об’єктно та компонентно-орієнтоване програмування, що ідеально відповідає структурі Unity;

— схожість із іншими C-подібними мовами C#, як і C++, Java, має зрозумілий синтаксис, що сприяє швидшій розробці;

— можливості розширення C# надає зручні інструменти для створення модульних компонентів, що мають власні методи, властивості, події та атрибути.

Таким чином, використання Unity у поєднанні з мовою програмування C# дозволяє ефективно реалізовувати складну логіку гри, забезпечуючи кросплатформеність, масштабованість, зручність редагування та широкі можливості для оптимізації продукту. Саме тому дані інструменти стали основою технічної реалізації дипломного проєкту [4].

2.6 Налаштування графіки

Для налаштування графіки в Unity потрібно перейти в Edit > Project

Settings, а потім вибрати вкладку Graphics. Далі пропонується покрокове налаштування параметрів. Розглянемо детальніше доступні параметри налаштування:

— scriptable render pipeline settings (Налаштування скриптової системи рендерингу) — цей параметр дозволяє визначити набір команд для більш точного керування чергою відображення об’єктів на сцені.

— camera settings (Налаштування камери) — група параметрів, яка відповідає за різні типи рендерингу.

— transparency sort ode (Режим сортування прозорих об’єктів) визначає порядок відображення об’єктів згідно з їх відстанню до заданої осі, рендеринг в Unity здійснюється за певними критеріями, такими як номер шару або відстань до камери;

— default сортування об’єктів залежно від їхнього положення відносно камери;

— perspective сортування об’єктів залежно від перспективи;

— orthographic сортування об’єктів для ортографічного виду;

— transparency Sort Axis сортування прозорих об’єктів за осьовим напрямом, заданим користувачем.

Ці налаштування дозволяють точніше контролювати процес рендерингу та відображення об’єктів у грі, забезпечуючи необхідний рівень деталізації.

Окрім основних параметрів, варто звернути увагу на сумісність графічних налаштувань із обраною платформою розгортання (наприклад, ПК, мобільні пристрої, WebGL тощо). У процесі оптимізації важливо враховувати особливості цільової платформи, оскільки різні пристрої мають різний рівень підтримки шейдерів, тіней, постобробки та інших графічних ефектів. Хоча ці аспекти не є критичними на етапі первинного налаштування, їх врахування дозволяє уникнути потенційних проблем на пізніших стадіях розробки та

тестування. Графічні налаштування повинні відповідати можливостям обраної платформи (ПК, мобільні пристрої, WebGL) — обирайте лише ті шейдери, тіні та постобробні ефекти, які на ній підтримуються, щоб уникнути помилок під час тестування.

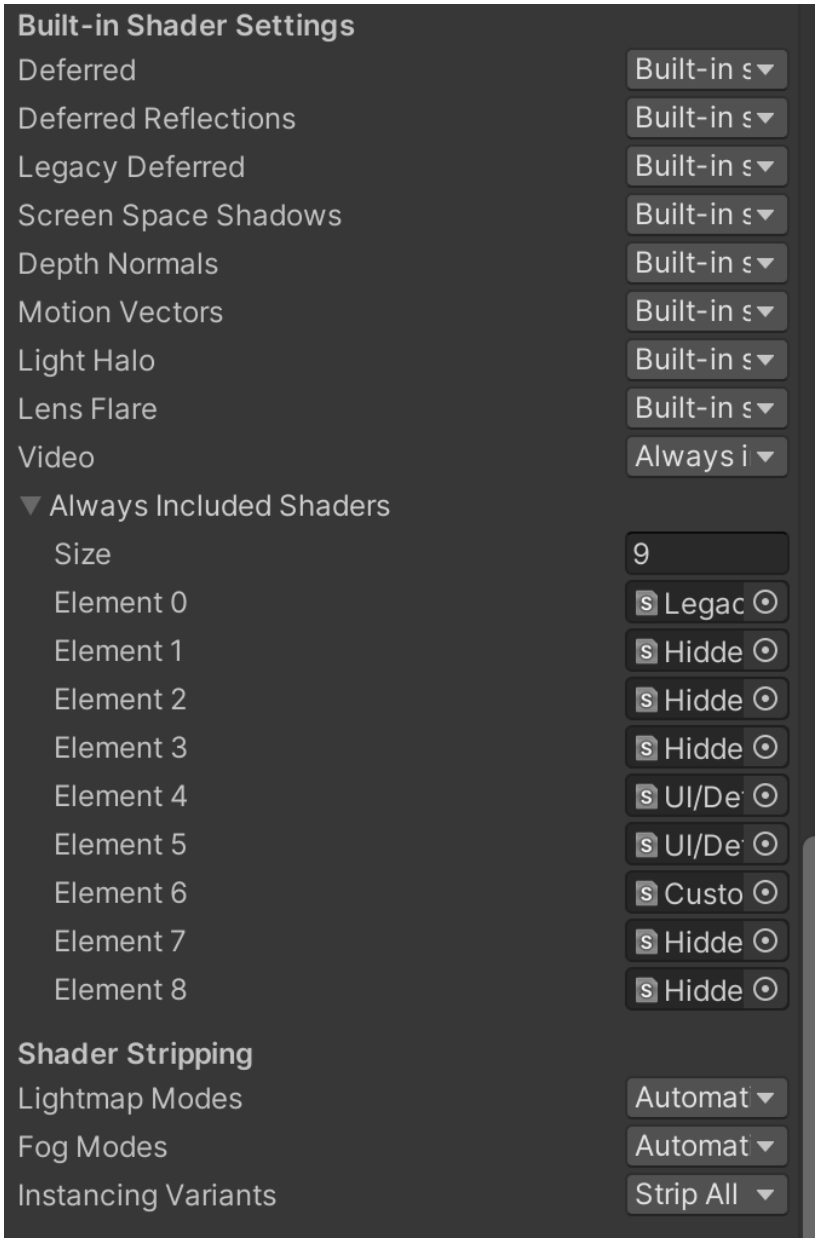

Рисунок 2.2 — Налаштування графіки

Tier Settings (Налаштування рівня) дозволяють вносити специфічні коригування для платформи в рендеринг і компіляцію шейдерів через налаштування як показано на (рис. 2.2). Це дає можливість поступово

виконувати рендеринг, а також завантажувати більш важкі матеріали на ранніх етапах, що допомагає уникнути ситуацій, коли проект вже завантажено, але матеріали ще не повністю завантажились [5].

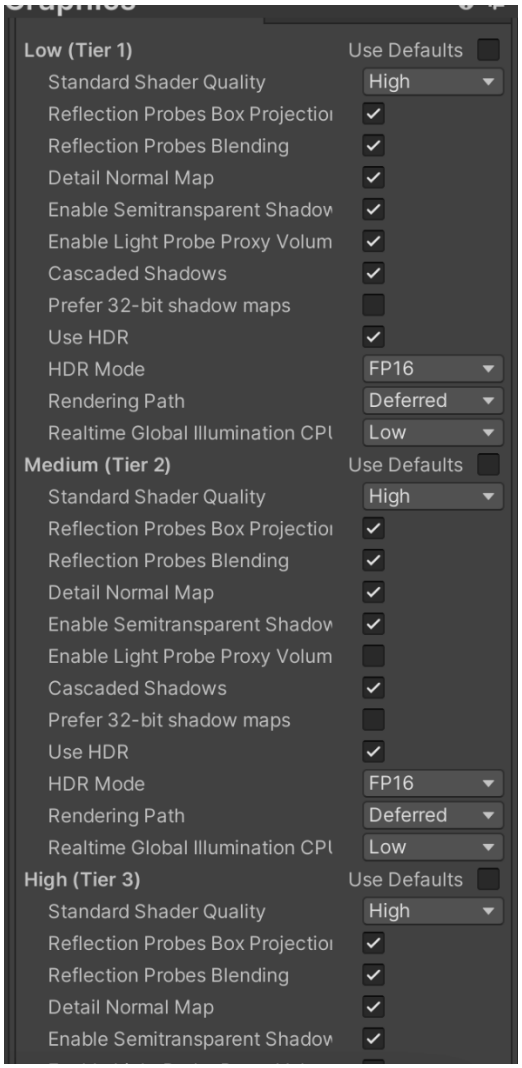

Рисунок 2.3 — Відкладений рендеринг

Tier Settings (Налаштування рівня) дозволяють вносити специфічні коригування для платформи в рендеринг і компіляцію шейдерів через налаштування. Вони містять вбудовані параметри, які визначені в бібліотеці Rendering.GraphicsTier (рис. 2.3).

Нижче наведено основні параметри, доступні для налаштування графіки в Unity.

Standard Shader Quality — налаштування якості стандартного шейдера на високий, середній або низький рівень.

Reflection Probes Box Projection — включає проекцію для відображення UV карт на Reflection Probes.

Reflection Probes Blending — активує змішування Reflection Probes.

Detail Normal Map — включає функцію детального нормального мапування, якщо вона необхідна.

Enable Semitransparent Shadows — дозволяє використовувати напівпрозорі тіні, що додає або видаляє визначення для компілятора шейдерів.

Enable Light Probe Proxy Volume — включає рендеринг 3D сітки з інтерпольованими Light Probes.

Cascaded Shadows — активує використання каскадних тіней.

Prefer 32-bit Shadow Maps — дозволяє використовувати 32-бітну карту тіней для платформ, що підтримують DirectX 11 або 12, або при розробці для консолей, таких як PS4.

Use HDR — активує рендеринг з високим динамічним діапазоном.

HDR Mode — дає можливість вибору формату для HDR буфера. За замовчуванням використовується формат FP16.

FP16 — формат, що використовує 16 біт з плаваючою точкою на канал.

Rendering Path — вибір способу рендерингу в Unity, який визначає продуктивність гри та спосіб обчислення освітлення і затінення. Доступні варіанти:

Forward — традиційний шлях рендерингу з підтримкою всіх основних графічних функцій.

Deferred — відкладене затінення, яке краще підходить для проектів з багатьма джерелами світла, але потребує апаратної підтримки.

Legacy Vertex Lit — шлях рендерингу з низькою точністю освітлення і без підтримки реальних тіней.

Legacy Deferred (light prepass) — застарілий варіант відкладеного рендерингу з компромісами в якості.

Realtime Global Illumination CPU Usage — визначає, скільки процесорних потоків використовуватиметься для обчислень освітлення під час виконання:

Low — 25% потоків.

Medium — 50% потоків.

High — 75% потоків.

Unlimited — 100% потоків.

Built-in Shader Settings — параметри для вибору шейдера, який буде використовуватися для різних функцій:

Deferred — для функцій відкладеного затінення.

Deferred Reflection — для Reflection Probes з відкладеним освітленням.

Screen Space Shadows — для каскадних тіней на ПК/консолях.

Legacy Deferred — для застарілих функцій відкладеного освітлення.

Motion Vectors — для обчислення векторів руху.

Always-included Shaders — список шейдерів, які завжди зберігаються в проєкті, навіть якщо не використовуються на сцені.

Shader Stripping — зменшує розмір даних збірки, видаляючи невикористовувані шейдери, що прискорює завантаження.

За замовчуванням властивість Lightmap modes має значення Automatic, що означає, що Unity вирішує, які варіанти sheder пропустити.

Щоб вказати, які режими використовувати самостійно, змініть цей параметр на призначений для користувача і увімкніть чи вимкніть такі режими lightmapping:

— Baked Non-Directional;

— Baked Directional;

— Realtime Non-Directional;

— Realtime Directional;

— Baked Shadowmask;

— Baked.

Те саме стосується і туману, тому для вказівки режиму використання самостійно, змініть параметр на призначений для користувача і увімкніть чи вимкніть такі режими fog:

— Linear;

— Exponential;

— Exponential Squared;

— Strip Unused (Default value) використовується коли Unity створює білд, він включає в себе тільки ті варіанти шейдеров, які щонайменше мають хоча б один матеріал з посиланням на шейдер і включеним Enable instancing. Unity видаляє всі шейдери, на які не посилаються матеріали з відключеними Enable instancing;

— Strip All видалення всіх примірників варіантів sheder, навіть якщо вони використовуються;

— Keep All зберігає всі варіанти примірників sheder, навіть якщо вони не використовуються.

Вкажіть список варіантів sheders з колекції Asset для попереднього завантаження під час завантаження гри. Зазначені в цьому списку варіанти sheders завантажуються на протязі всього терміну життя додатки. Використовуйте його для попереднього завантаження дуже часто використовуваних sheders [5].

2.6 Rendering Penpline

Перш ніж зображення з'явиться на екрані, здійснюється низка підготовчих та рендерингових етапів, які можна умовно поділити на дві частини: підготовка до рендерингу та сам рендеринг.

На етапі підготовки до рендерингу ігровий процесор визначає, які об'єкти активні, які моделі та текстури вони використовують, а також їх розташування в світі та подальші дії з ними. Також на цьому етапі визначається позиція камери та її напрямок. Після цього ігровий процесор передає цю інформацію візуалізатору.

Отримавши дані про сцену, графічний рушій приступає до рендерингу. Конвеєр рендерингу може варіюватися залежно від використовуваного 3D API, але загалом його можна уявити так:

Для кожної моделі візуалізатор спочатку оцінює її розмір та положення камери, визначаючи, чи модель буде видимою на екрані, чи потрапляє вона в поле зору камери, або вона настільки віддалена, що її не видно. Для цієї мети візуалізатор може застосовувати різні методи визначення видимості, наприклад, BSP дерева або методи порталів. Можна використовувати комбінацію цих методів. Полігони, які пройшли тест на відсічення зайвої геометрії, передаються далі візуалізатору.

Далі для кожного полігону візуалізатор проводить трансформацію за допомогою локальної математики (для анімації) і світової математики (для позиціонування моделі щодо камери). Потім перевіряються полігони на наявність нелицьових (невидимих) сторін об'єкта, і ці полігони відкидаються. Залишаються тільки ті, які піддаються освітленню згідно з джерелами світла на сцені. Візуалізатор визначає, які текстури має використовувати полігон, і перевіряє, чи підтримує API та відеокарта ці текстури. Потім полігони передаються на рендеринг API, після чого потрапляють на відеокарту.

У результаті формується зображення в буфері кадру на відеокарті, яке виводиться на екран монітора.

Швидкість проходження рендеринг-конвеєра безпосередньо впливає на продуктивність гри. Якщо гра працює повільно, це негативно позначається на загальному досвіді користувача. Тому підвищення продуктивності графічного

рушія є важливою задачею для розробників, і вони повинні знаходити оптимальний баланс між продуктивністю та якістю графіки. Щоб гра могла працювати на різних конфігураціях ПК, розробники повинні забезпечити можливість налаштування якості графіки користувачем. Тому такі налаштування мають бути вбудовані в графічний рушій з самого початку [5].

2.7 Використання Sheider-ів

Shader — програма, яка визначає вигляд поверхні об'єкта візуально, включаючи освітлення, текстурування, постобробку та інші аспекти. Шейдери виникли з концепцій Кука (Cook's shade trees) і Перліна (Perlin's pixel stream language). Програмовані шейдери вперше з'явилися в RenderMan компанії Pixar, де були визначені кілька типів шейдерів: light source shaders, surface shaders, displacement shaders, volume shaders, imager shaders. Ці шейдери зазвичай виконуються універсальними процесорами та не мають повної апаратної реалізації. Пізніше було розроблено низку мов, схожих на RenderMan, але призначених для апаратного прискорення. Peercy та його команда розробили техніку для виконання програм з умовами та циклами на традиційних апаратних архітектурах за допомогою кількох проходів рендеринга. Шейдери RenderMan розділялися на кілька проходів, результати яких комбінувалися в буфері кадру. З часом з'явилися мови, апаратно прискорені в DirectX та OpenGL, що дозволило адаптувати шейдери для графічних додатків реального часу.

На початку відеочіпи були програмованими, виконуючи лише заздалегідь запрограмовані дії (fixed-function), наприклад, алгоритми освітлення, які не можна було змінити. Проте з часом виробники відеочіпів поступово впроваджували програмовані елементи, хоча спочатку ці можливості були обмежені та не мали програмної підтримки в Microsoft DirectX API. Однак з появою Shader Model 2.0 (SM2) в DirectX 9 значно

розширилися можливості шейдерів реального часу, додавши підтримку складніших шейдерів і значно розширивши набір команд. Одним із найважливіших нововведень стало введення підтримки розрахунків з плаваючою комою для піксельних шейдерів. DirectX 9 також ознаменував появу мови високорівневих шейдерів — HLSL, схожої на мову C, та ефективного компілятора, який перетворює програми HLSL в низькорівневий код, зрозумілий для апаратних засобів.

Шейдери значно розширили можливості графічного конвеєра, дозволяючи виконувати трансформацію та освітлення вершин і індивідуальну обробку пікселів відповідно до вимог конкретного додатка.

Шейдери поділяються на два основні типи: вершинні та піксельні шейдери. Вершинні шейдери виконуються на відеочіпах і відповідають за математичні операції з вершинами. Вони дозволяють змінювати параметри вершин і їх освітлення. Кожна вершина визначається кількома змінними, наприклад, координатами (x, y, z), кольором, текстурами тощо. Вершинні шейдери змінюють ці дані під час роботи, наприклад, обчислюючи нові координати чи кольори. Вхідними даними для вершинного шейдера є інформація про поточну вершину моделі, яка обробляється. Результати роботи вершинного шейдера використовуються на наступних етапах конвеєра, зокрема растеризатором для лінійної інтерполяції даних для поверхні трикутника і подальшої обробки піксельним шейдером.

Піксельні шейдери виконуються на відеочіпах під час растеризації кожного пікселя зображення. Вони здійснюють вибірку з текстур і/або виконують математичні операції з кольором та глибиною (Z-buffer) пікселів. Усі інструкції піксельного шейдера виконуються по пікселях після завершення трансформації та освітлення геометрії. Результатом роботи піксельного шейдера є кінцеве значення кольору пікселя та значення Z для подальшого етапу конвеєра — змішування (blending) [6].

3 РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТІВ

3.1 Постановка задач

Бакалаврська дипломна робота являє собою розробку гри на ігровому рушію Unity. Головними завданнями для створення дипломного проєкту є:

— розробка управління головного героя;

— розробка камери для взаємодії з об’єктом;

— взаємодія з камерою;

— бар’єри і обмеження;

— використання shaders для перенесення обчислень на відео карту;

— використання post-procesing як спосіб задати атмосферу.

Для початку створюється прототип взаємодії з інтерактивними елементами меню налаштувань як взагалі має виглядати і взаємодія з ним взаємодії як на (рис. 3.1).

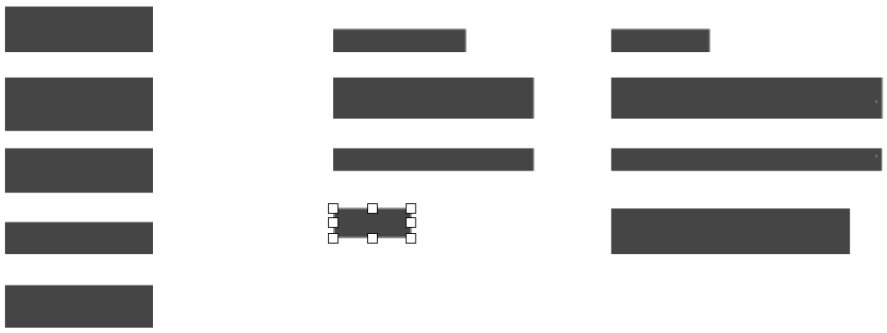

Рисунок 3.1 — Початкова блок-схема вікна для налаштувань гри

Наступним етапом створення взаємодії в меню, є що має відбуватися при натисканні кожної кнопки (рис. 3.2). Під час розробки будемо керуватися цією діаграмою для створення меню. Головним пункт це пункт налаштування тому що там найбільше роботи і взагалі майже вся взаємодія з ресурсами Unity.

Рисунок 3.2 — Блок схема, для переходів по пунктах меню

3.2 Управління персонажем

Головним в кожній грі є взаємодія персонажа, з ігровим світом, тобто як саме управляється головний персонаж. Управляння вирішив розбити на різні блоки по принципах проєктування Solid тобто кожен з блоків використовується окремо. Для переміщення використовується компонент rigibody — компонент який за фізичну взаємодію з світом це лістинг 3.1.

Лістинг 3.1 — Клас PhysicsMovoment

```
using UnityEngine;
public class PhysicsMovoment : MonoBehaviour
{
   [SerializeField] private Rigidbody _rigidbody;
   [SerializeField] private float _speed;
   public void Move(Vector3 direction){
       Vector3 offset = direction * (_speed * Time.deltaTime);
   _rigidbody.MovePosition(_rigidbody.position + offset);
   }
```

У цьому випадку стрибок має бути керованим, щоб можна було змінювати його траєкторію через редактор. Для цього використовується

CurveAnimation. Однак для коректної взаємодії стрибка з іншими силами, такими як гравітація, швидкість та інерція, необхідно нейтралізувати ці сили. Для цього я написав допоміжний скрипт PureAnimation (лістинг 3.2), який усуває ці недоліки.

Лістинг 3.2 — Клас PureAnimation

```
public class PureAnimation : MonoBehaviour
{                   public Vector3 LastChange { get; private set; }
   private Coroutine _lastAnimation;
   private MonoBehaviour _context;
   public PureAnimation(MonoBehaviour context)
   {_context = context;}
public void Play(float duration, Func<float,Vector3> body)
         {if(_lastAnimation!=null) _context.StopCoroutine(_lastAnimation);
         _lastAnimation _context.StartCoroutine(GetAnimation(duration, body));}
         private IEnumerator GetAnimation(float duration, Func<float, Vector3> body)
         {float expiredSeconds = 0;float progress = 0; while (progress < 1){expiredSeconds
+= Time.deltaTime; progress = expiredSeconds / duration; LastChange =
body.Invoke(progress);
yield return null;}}}
```

Стрибок повинен бути керованим, тобто основні параметри, такі як висота, довжина, траєкторія та тривалість стрибка, мають задаватися безпосередньо в налаштуваннях скрипта в редакторі. Це спрощує подальшу роботу з створенням рівнів та налаштуванням метрик (лістинг 3.3).

Лістинг 3.3 — Клас PhysicsJump

```
public class PhysicsJump : MonoBehaviour{
[SerializeField] private JumpFX _fx;
[SerializeField] private float _lenght;
[SerializeField] private float _duration;
private PureAnimation _playTime;
private void Awake() => _playTime = new PureAnimation(this);
 public void Jump(Vector3 direction)
Vector3 target = transform.position + (direction * _lenght);
Vector3 startPosition = transform.position;
          PureAnimation fxPlayTime = _fx.PlayAnimation(transform, _duration);

          _playTime.Play(_duration, (progress) => transform.position =
```

Продовження лістингу 3.3

```
  Vector3.Lerp(startPosition, target, progress) + fxPlayTime.LastChange;
return transform.position;});}}
```

Також повинна бути реалізована інтеграція з інтерфейсом (лістинг 3.4) для покращення взаємодії та оптимізації. Для цього застосовано патерн Observer, при якому дані зчитуються не на кожному кадрі, а тільки при зміні вхідних параметрів.

Лістинг 3.4 — Клас PlayerUI

```
public class PlayerUI : MonoBehaviour
{   [SerializeField] private NewPlayer target;
   [SerializeField] private Image[] healths;
   [SerializeField] private Sprite activeHealths;
   [SerializeField] private Sprite notHealths;
              Private void Start() => UpdateValue(NewPlayer.Instance.health);
           private void OnEnable() => target.OnHealthChange += UpdateValue;
           private void OnDisable() => target.OnHealthChange -= UpdateValue;
           public void UpdateValue(int health){if (target == null) return;
      for (int i = 0; i < healths.Length; i++)
         if (health > i)
            healths[i].sprite = activeHealths;
         else
            healths[i].sprite = notHealths;}}
```

Для правильної роботи камери необхідно передавати дані позиції миші та згладжувати їх, щоб уникнути ривків при переміщенні (лістинг 3.5).

Лістинг 3.5 — Клас CameraController

```
public class CameraController : MonoBehaviour
        { [SerializeField] [Range(0.0f, 5f)] private float      mouseSmoothTime = 0.3f;
   [SerializeField] private float mouseSensitivity;
   [SerializeField] private Transform playerCamera;
   private float cameraPitch;
   private Vector2 currentMouseDelta = Vector2.zero;
              private Vector2 currentMouseDeltaVelocity = Vector2.zero;
  private void CameraUpadate(float mouseX,float mouseY){
   Vector2 targetMouseDelta = new Vector2(mouseX, mouseY);
         currentMouseDelta          =          Vector2.SmoothDamp(currentMouseDelta,
 targetMouseDelta, ref currentMouseDeltaVelocity, mouseSmoothTime);
cameraPitch -= currentMouseDelta.y * mouseSensitivity;
```

Продовження лістингу 3.5

```
cameraPitch = Mathf.Clamp(cameraPitch, -90.0f, 90.0f);
playerCamera.localEulerAngles = Vector3.right * cameraPitch;
        transform.Rotate(Vector3.up * currentMouseDelta.x * mouseSensitivity);}
        private void Update() => CameraUpadate(Input.GetAxis("Mouse X"),
  Input.GetAxis("Mouse Y"));}
```

3.3 Розробка головного меню

При створенні власної системи вибору елементів в UI використовуємо один менеджер та другорядні скрипти які приймають дані від менеджера, та реагують. Створювати власний менеджер взаємодії потрібний для керованого переміщення по інтерфейсу, без прив'язки до позицій на сцені (рис. 3.3).

Рисунок 3.3 — Меню

Створення загальної Input системи яка приймає дані з контролера і задає потрібний індекс для правильної взаємодії з інтерфейсом (лістинг 3.4).

Лістинг 3.6 — Клас MenuButton

```
public class MenuButton : MonoBehaviour
{
        [SerializeField] private MenuButtonController buttonController;
   [SerializeField] private AnimatorMenu animatorMenu;
   [SerializeField] private Animator animator;
```

Продовження лістинг 3.6

```
            [SerializeField] private AnimatorFunction animatorFunction;
        [SerializeField] private int thisIndex;
    [SerializeField] private bool activePresset = true;
    private void Update() => ButtonConroller();
private void ButtonConroller()
    {if (buttonController.index == thisIndex){
     animator.SetBool("Selected", true);
if (Input.GetAxis("Submit") >= 0.7f && activePresset){
animator.SetBool("Pressed", true);
animatorMenu.AnimationMenuNumber(thisIndex);}
else if (animator.GetBool("Pressed"))
animator.SetBool("Pressed", false);}
        else
animator.SetBool("Selected", false);}}
```

3.4 Процес завантаження даних

Також одним з головних етапів кожної гри є зміна сцени, і елементи його загрузки для цього створений клас LevelLoader який загружає всі елементи сцени загружаються паралельно, та з плавною анімацією переходів між сценами, це елемент який можна визивати з будь-якого місця коду тому що це утиліта яка використовує статичне посилання.

Тобто є два варіанти виклику загрузки сцени передавання прямим посиланням, або ж просто передати назву сцени як аргумент, цей скрипт можна використовувати як тригер для порталу (лістинг 3.7 та 3.8). Цей скрипт можна використовувати як тригер для порталу, що активується під час входу гравця в задану область або після виконання певної дії.

Лістинг 3.7 — Клас MenuButton

```
public class LevelLoader : MonoBehaviour
{
            [SerializeField] private bool shouldPlayOpeningAnimation = false;
private static LevelLoader instance;
private Animator animator;
private AsyncOperation loadingSceneOperation;
public static void SwitchToScene(string sceneName){
        instance.animator.SetTrigger("End");
                instance.loadingSceneOperation =
```

Продовження лістинг 3.7

```
  SceneManager.LoadSceneAsync(sceneName);
            instance.loadingSceneOperation.allowSceneActivation = true;}
private void Start(){
        animator = GetComponent<Animator>();
        instance = this;
if (shouldPlayOpeningAnimation)
            animator.SetTrigger("Start");}
    public void OnAnimationOver(){
        shouldPlayOpeningAnimation = true;
                    loadingSceneOperation.allowSceneActivation = true;}}
```

Лістинг 3.8 — Клас MenuButton

```
public class PortalNextZone : MonoBehaviour
{
 private void OnCollisionEnter(Collision collision)
    {
            If(collision.gameObject.GetComponent<NewPlayerInteractable>()){LevelLoade
  r.SwitchToScene("ForItchIO");}}}
```

3.5 Взаємодія з об’єктами

Жодна гра не може обійтися без інтерактивних об’єктів та взаємодії з ними, оскільки саме вони формують основу ігрового процесу, забезпечують зворотний зв’язок і дозволяють гравцеві активно впливати на події у віртуальному середовищі. До таких об’єктів належать як елементи керування, так і частини оточення, що реагують на дії користувача. Наприклад, кнопки, платформи, бар’єри, перемикачі або навіть умовно прості елементи, як-от куб, можуть виконувати роль інтерактивних елементів, забезпечуючи логіку подій або змін у сцені. Ці об’єкти можуть бути як одноразовими тригерами, так і багаторазовими механізмами, що змінюють свій стан залежно від контексту гри. Кожен із них має власну логіку роботи, яка реалізується на основі заданих умов або сценаріїв, розроблених дизайнером. Такий підхід дає змогу не лише урізноманітнити ігровий процес, а й створити відчуття живого, динамічного світу, що реагує на дії гравця (рис. 3.4).

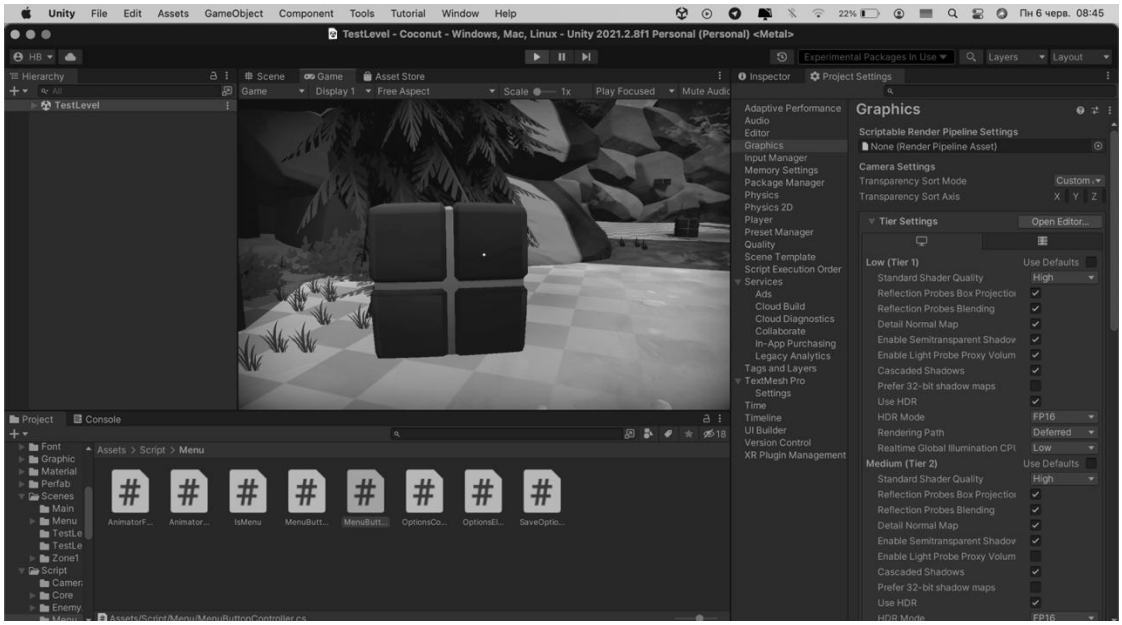

Рисунок 3.4 — Виділений об’єкт

Для взаємодії з об’єктами створений скрип PhysicsObject який відповідає за можливість взаємодіяти з курсором та переміщає об’єкт по його по бажанню на певну відстань (лістинг 3.9).

Лістинг 3.9 — Клас PhysicsObject

```
public class PhysicsObject : MonoBehaviour
{
   [SerializeField] private float waitOnPickup = 0.2f;
   [SerializeField] private float breakForce = 35f;
   [HideInInspector] public bool pickedUp = false;
   [HideInInspector] public NewPlayerInteractable player;
   private void OnCollisionEnter(Collision collision)
            {if (pickedUp && collision.relativeVelocity.magnitude > 
  breakForce)player.BreakConnection()}
         public IEnumerator PickUp(){yield return new 
  WaitForSecondsRealtime(waitOnPickup);
pickedUp = true;}}
```

Бар’єри — важлива частина, гри тому що всі головоломки налаштовані для того що б їх забрати. Вони використовують властивість unity з layer mask (рис. 3.5).

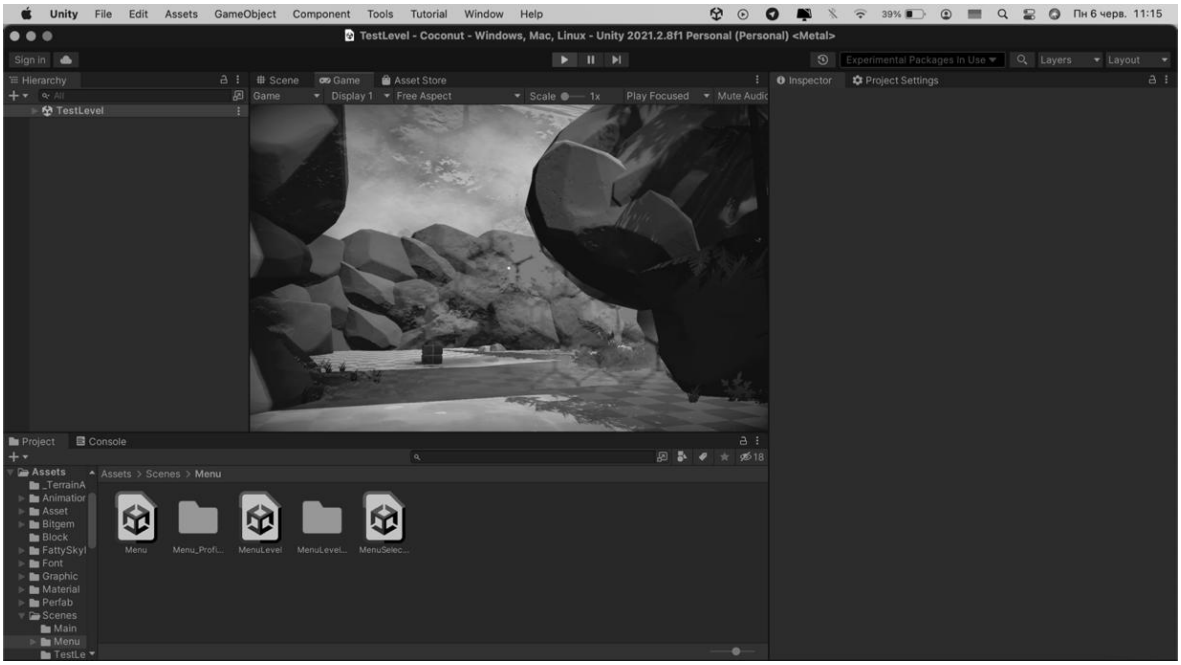

Рисунок 3.5 — Бар’єр та його shader

ShyBox — важлива частина проєкту яка впливає на загальний тон гри це зображення яке розгорнуте на 360 градусів, та складається з 6 малюнків (рис. 3.6).

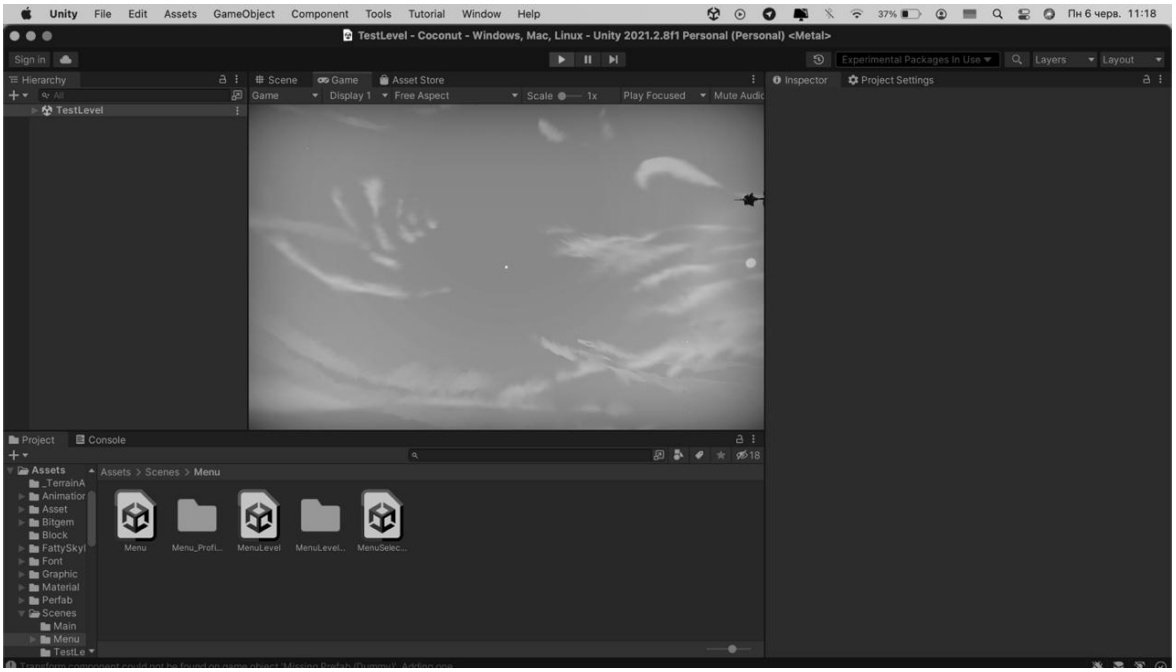

Рисунок 3.6 — SkyBox

Sheders — важлива частина проєкту яка впливає на візуальну складову, та добавляє візуального стилю та динаміку матеріалу яка добавляє, перенести навантаження з процесора на відеокарту (рис. 3.7).

Рисунок 3.7 — Shader води

3.6 Робота з основними механіками

У кожному ігровому проєкті існують основні механіки взаємодії з інтерактивними об'єктами. Це такі основні механіки:

— збільшення об'єктів цей процес реалізується за допомогою картин, що зменшують або збільшують об'єкт протягом певного часу, розмір змінюється залежно від вказаного часу та цільового розміру, до якого об'єкт має змінитися;

— робота з кнопками: при натисканні на кнопки відбуваються дії, зокрема заміна матеріалу на текстуру на самій кнопці, а також відкриваються бар'єри, ця взаємодія дозволяє використовувати і взаємодіяти з іншими об'єктами (лістинг 3.11 та 3.12).

Лістинг 3.11 — Клас PhysicsObject

```
IEnumerator ScaleObjectMax(Vector3 maxSize, float seconds)
   {startScales = true;
      Vector3 originalSize = startSize;
      Vector3 newSize = maxSize;float t = 0f;
      while (t <= 1.0){t += Time.deltaTime / seconds;
          gameObject.transform.localScale = Vector3.Lerp(originalSize, newSize, t);yield
  return null;}
          startScales = false;}IEnumerator ScaleObjectMin(Vector3 maxSize, float
  seconds){Vector3 originalSize = startSize;
Vector3 newSize = maxSize;startScales = true;float t = 0f;
while (t <= 1.0){t += Time.deltaTime / seconds;
          gameObject.transform.localScale = Vector3.Lerp(newSize, originalSize, t);yield
  return null;}startScales = false;}}
```

Лістинг 3.12 — Клас PhysicsObject

```
public class ObjectButton : MonoBehaviour
{
   [SerializeField] private GameObject button;
private delegate void ButtonAction();
private ButtonAction Action;
private void Start()
   {
      if (GetComponent<ButtonOpenDoor>() != null)
      Action = GetComponent<ButtonOpenDoor>().OpenDoor;
   }
        public void Button()
   {
      Action();
              button.GetComponent<MeshRenderer>().material.color = Color.green;
   }
}
```

І ще одна з головних механік, донесення історії — записки або діалоги, і для взаємодії з ними є важливою частиною інтерактивного світу проєкту, саме тому робота з ними має бути проста і зрозуміла, а самі записки не великими лістинг 3.13.

Лістинг 3.13 — Клас ObjectNote

```
public class ObjectNote : MonoBehaviour
```

Продовження лістингу 3.13

```
{
   public string Note;
   [SerializeField] private GameObject Noted;
   [SerializeField] private Text text;
   private bool isNote;
   public void Notes()
  {text.text = Note;
      if (!isNote){
        isNote = true;
          Noted.SetActive(true);
          Time.timeScale = 0;
      }
      else
      {
    isNote = false;
      Noted.SetActive(false);
          Time.timeScale = 1;}}}
```

Система частинок (particle) є однією з ключових складових проєкту, адже вона додає динаміки та візуальної естетики. Particle — система частинок, що контролює їх поведінку, зокрема, тривалість перебування на екрані та інші параметри (рис. 3.8 та рис. 3.9).

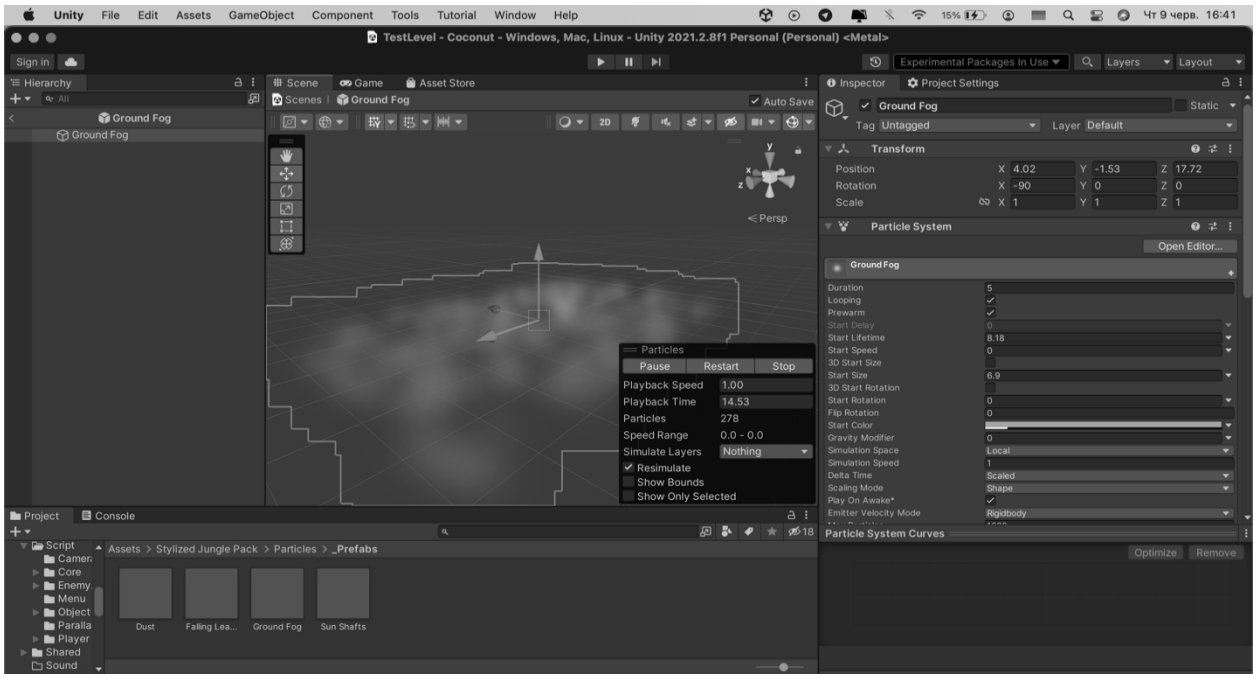

Рисунок 3.8 — Система частинок

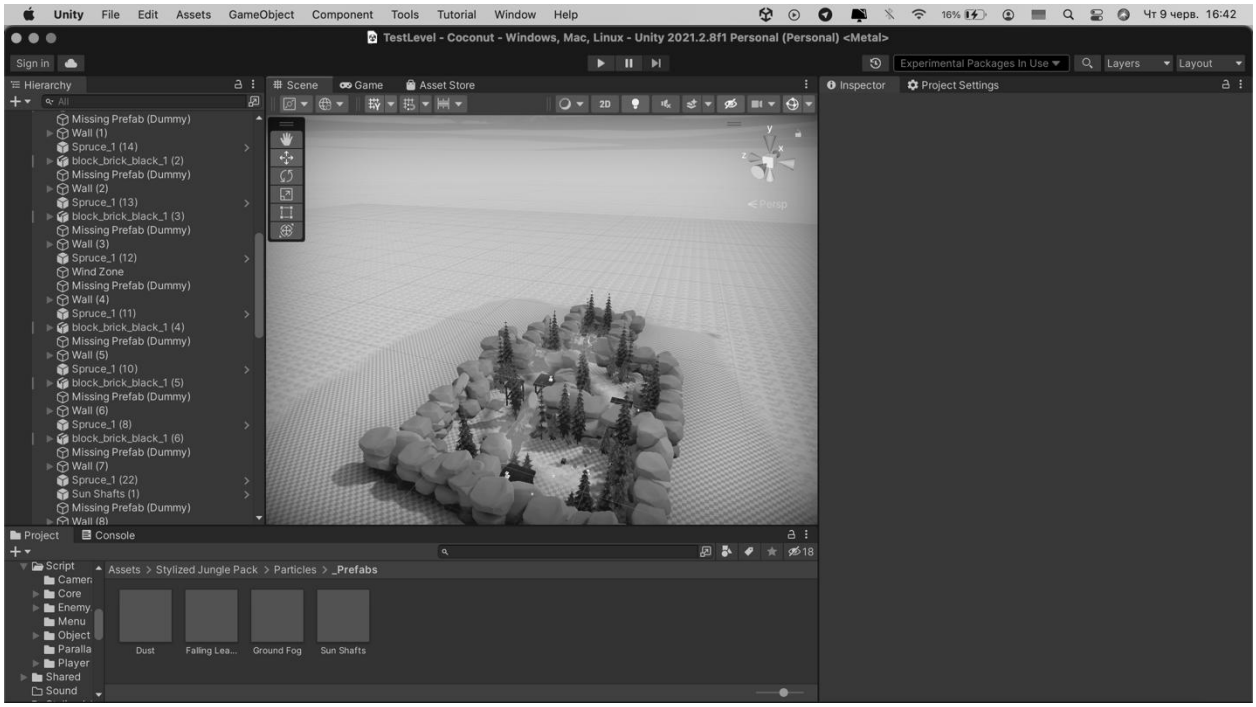

Рисунок 3.9 — Редактор Unity

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Аналіз методів тестування програмного забезпечення

У сучасному світі, де конкуренція на ринку програмного забезпечення постійно зростає, а вимоги і очікування користувачів стають дедалі вищими, тестування програмних систем набуває надзвичайної значущості. Цей процес є невід'ємною складовою циклу розробки, який дозволяє забезпечити високу якість, надійність, зручність використання та ефективність роботи програмних продуктів.

Тестування вже давно перестало бути лише формальною перевіркою функціональності. Воно стало ключовим фактором, що визначає конкурентоспроможність продукту на ринку, довіру користувачів та успіх розробників. Розглянемо основні причини, через які тестування систем є настільки важливим і необхідним:

— виявлення дефектів є однією з найважливіших задач тестування є ідентифікація помилок і несправностей у програмному забезпеченні, апаратних компонентах та інших елементах системи ще до того, як вони вплинуть на кінцевого користувача, раннє виявлення дефектів допомагає уникнути серйозних проблем у майбутньому, що дозволяє зекономити значні ресурси, адже виправлення багів на початкових етапах розробки завжди дешевше, ніж на пізніх;

— підтвердження відповідності якості в тестуванні дозволяє переконатися, що розроблена система відповідає всім встановленим вимогам та очікуванням користувачів, включає не лише перевірку коректності роботи функцій, але й оцінку зручності використання, стабільності, а також відповідність цілей і завдань, для яких система була створена;

— надійне тестування сприяє зростанню впевненості користувачів та зацікавлених сторін у тому, що система працюватиме без збоїв і виконуватиме покладені на неї функції та сприяє більш широкому прийняттю продукту,

покращенню репутації розробника та збільшенню кількості користувачів;

— зменшення ризиків є процесом тестування який допомагає мінімізувати ризики, пов’язані з розробкою, впровадженням та подальшою експлуатацією програмних систем, виявлення і усунення проблем на ранніх стадіях значно знижує ймовірність виникнення критичних несправностей, що можуть призвести до фінансових втрат, збоїв у роботі чи втрати клієнтів;

— покращення користувацького досвіду у тестуванні є також важливим інструментом для підвищення задоволеності користувачів, воно дозволяє виявити і виправити різні проблеми, які можуть негативно вплинути на комфорт використання — наприклад, недоліки в юзабіліті, знижену продуктивність чи проблеми з сумісністю це забезпечуючи простоту, швидкість і безпомилкову роботу програми, тестування сприяє кращому сприйняттю продукту і формуванню позитивного іміджу.

Отже, тестування програмних систем є фундаментальною частиною розробки, що дозволяє не лише підвищити якість та стабільність програмного забезпечення, але й створити конкурентний, зручний і надійний продукт, здатний відповідати сучасним вимогам ринку та очікуванням користувачів. Воно є ключовим інструментом для розробників, які прагнуть забезпечити успіх своїх проектів та довготривалу довіру з боку споживачів.

На сьогодні тестування програмного забезпечення – один з найбільш дорогих етапів життєвого циклу програмного забезпечення, на нього відводиться від 50% до 65% загальних витрат [15].

Серед основних методик тестування можна виділити наступні.

Модульне тестування (Unit Testing): перевірка окремих компонентів або модулів коду на предмет їхньої працездатності.

Інтеграційне тестування (Integration Testing): тестування взаємодії між різними модулями або компонентами системи.

Системне тестування (System Testing): тестування всієї системи в

цілому, включаючи її функціональність, продуктивність та безпеку.

Приймальне Тестування (Acceptance Testing): тестування, яке проводиться замовником або кінцевим користувачем для підтвердження, що система відповідає їхнім вимогам.

Регресійне тестування (Regression Testing): повторне тестування системи після внесення змін, щоб переконатися, що нові зміни не порушили існуючу функціональність.

Димове тестування (Smoke Testing): швидка перевірка основних функцій системи, щоб переконатися, що вона в принципі працює.

Тестування безпеки (Security Testing): перевірка системи на наявність вразливостей та захисту від несанкціонованого доступу.

Тестування зручності використання (Usability Testing): оцінка того, наскільки легко та зручно користувачам взаємодіяти з системою.

Тестування продуктивності (Performance Testing): визначення швидкодії, стабільності та масштабованості системи під навантаженням.

Стрес-тестування (Stress Testing): перевірка системи на здатність працювати в умовах перевищення нормальних навантажень.

Тестування на відмовостійкість (Fault Tolerance Testing): визначення здатності системи продовжувати роботу у разі виникнення помилок або збоїв.

Тестування сумісності (Compatibility Testing): перевірка роботи системи на різних платформах, браузерах або пристроях.

Отже, провівши аналіз методик функціонального тестування, було вирішено обрати тестування продуктивності, адже він дозволяє прискорити процес тестування, порівнюючи навантаження різних модулів системи.

4.2 Тестування системи

Для тестування системи було обрано GPU Visualizer [16] (рис. 4.1).

GPU Visualizer — потужний інструмент, який дає можливість детально

аналізувати використання графічного процесора (GPU) в реальному часі під час роботи програми. Завдяки широкому функціоналу цього засобу можна точно визначити, які саме елементи програмного забезпечення споживають найбільше ресурсів GPU, виявити місця виникнення затримок, а також розробити ефективні кроки для оптимізації продуктивності.

Проведення тестів в середовищі Unreal Engine із застосуванням GPU Visualizer дозволяє отримати комплексну інформацію про взаємодію програми з графічним процесором. Серед основних параметрів, що відслідковуються, — обсяг використаної відеопам’яті, частота кадрів у секунду, час виконання окремих фрагментів коду і графічних ефектів. Це дає змогу розробникам приймати обґрунтовані рішення щодо оптимізації як коду, так і використання ресурсів, а також тонко налаштовувати графічні налаштування для досягнення кращої продуктивності.

Таким чином, GPU Visualizer стає незамінним інструментом у роботі розробників, оскільки надає глибоке розуміння внутрішніх процесів, що відбуваються на графічному процесорі під час виконання програми. Отримані дані допомагають ідентифікувати «вузькі місця», що уповільнюють роботу, і дають змогу реалізувати заходи для підвищення ефективності системи. Це, у свою чергу, сприяє створенню більш плавного, стабільного і приємного досвіду для кінцевих користувачів.

Вся інформація, отримана під час тестування, активно використовується для подальшого вдосконалення програми, що забезпечує високу якість та оптимальну роботу програмного продукту на різних апаратних платформах.

GPU Visualizer дає розробникам деталізовані дані про роботу GPU під час виконання, допомагаючи виявляти вузькі місця та оптимізувати продуктивність. Отримані результати використовуються для вдосконалення програми й забезпечують стабільну роботу на різних апаратних платформах. Це допомагає швидше випускати оновлення з оптимізаціями.

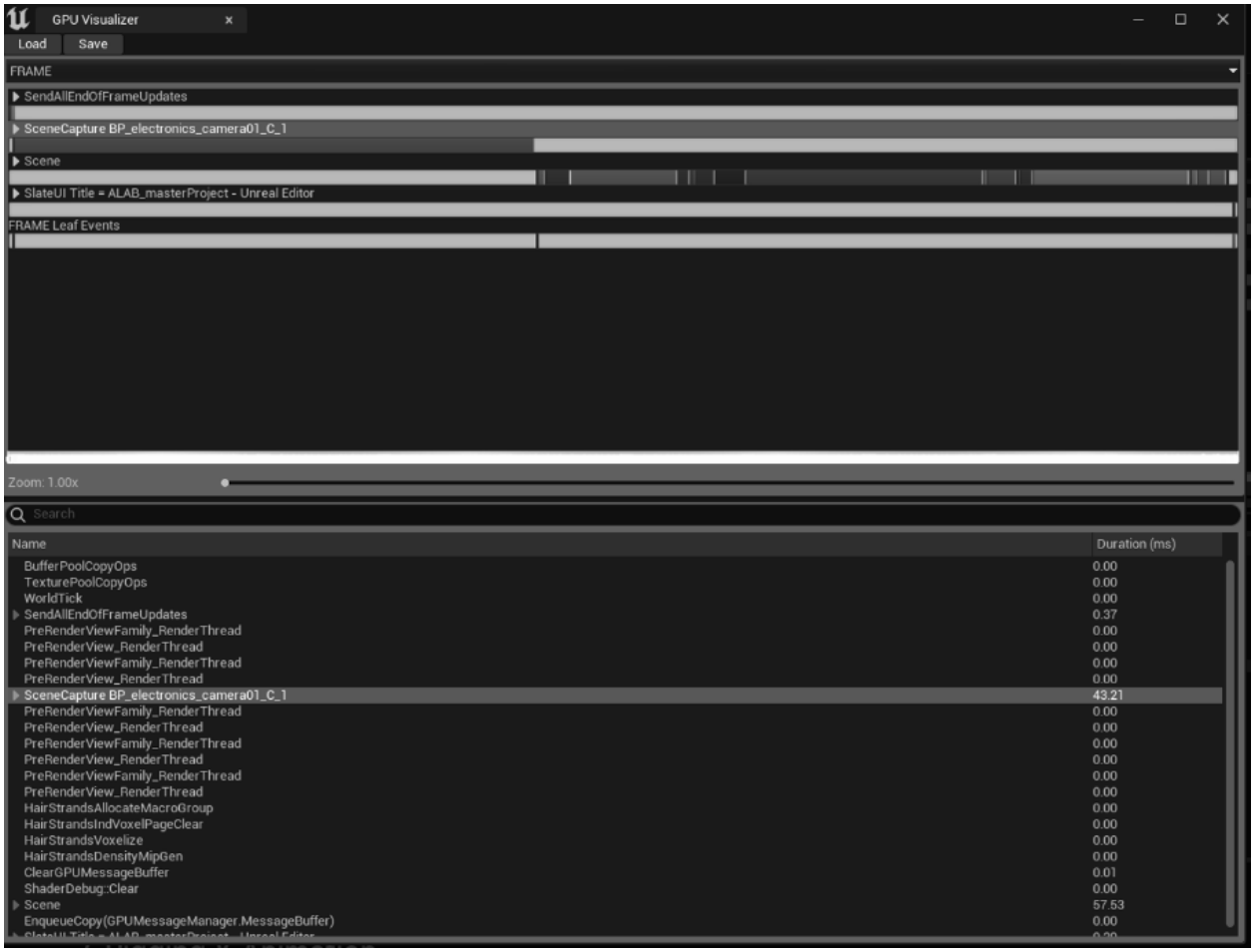

Рисунок 4.1 —Тестування продуктивності системи

4.3 Розробка інструкції користувача

У своїй статті «Зручність 101: Вступ до вивчення зручності» [17], опублікованій у блозі компанії Nielsen Norman Group, Якоб Нільсен описує основні характеристики зручності цифрових продуктів.

Серед них він виділяє такі аспекти:

— можливість навчання полягає в тому, наскільки легко користувачі можуть виконувати основні завдання під час першого ознайомлення з інтерфейсом;

— ефективність визначається тим, як швидко користувачі здатні виконувати завдання після того, як ознайомилися з дизайном;

— запам’ятовуваність описує, наскільки просто користувачам

відновити навички після певного періоду неактивного користування продуктом;

— помилки оцінюються за кількістю допущених користувачами помилок, їхньою серйозністю, а також за тим, наскільки легко користувачі можуть виправити помилки.

Виходячи з визначених критеріїв, було прийнято рішення розробити інтерфейс системи з акцентом на максимальну зручність для користувача.

Користувацький інтерфейс побудований за модульним принципом, що значно полегшує налаштування та подальше розширення функціональних можливостей системи. Кожен модуль має свій власний інтерфейс, який підтримує загальну стилістику програми, проте враховує особливості конкретних завдань, що він виконує.

При відкритті додатку користувача зустрічає екран завантаження, на якому розміщено логотип програми та відтворюється анімація, що відображає процес завантаження.(рис. 4.2).

Рисунок 4.2 — Екран завантаження

Після повного завантаження системи користувач потрапляє до

головного меню, де йому надається можливість обрати бажаний варіант подальшої взаємодії з програмою.

Функціональні кнопки, розташовані на головному екрані, спроектовані з урахуванням максимального комфорту та інформативності. Кожна кнопка містить чіткий текстовий напис, який одразу інформує користувача про її призначення, що значно спрощує навігацію. Завдяки продуманому дизайну та зручному розміщенню кнопок, користувач може швидко знаходити потрібні опції, що підвищує ефективність роботи з інтерфейсом.

Кнопка «Start Game» відповідає за запуск основного ігрового процесу. Натискання цієї кнопки переводить користувача до графічної сцени — просторого віртуального світу, де можна керувати заздалегідь запрограмованим персонажем та взаємодіяти з ігровим середовищем.

Кнопка «Options» є важливим елементом інтерфейсу, що відкриває меню налаштувань (рис. 4.3). Вона надає користувачеві широкий доступ до різноманітних параметрів програми, що дозволяє адаптувати її роботу під індивідуальні потреби та умови використання. Завдяки цьому користувач має змогу оптимізувати продуктивність і зручність використання програми відповідно до власних вимог.

У меню налаштувань користувач має можливість регулювати параметри дисплею, такі як роздільна здатність екрану та режим відображення, а також керувати параметрами відображення статистики.

Крім того, у цьому розділі доступні налаштування графіки, які включають автоматичне регулювання якості, вибір попередніх профілів якості, параметри роздільної здатності 3D, глобального освітлення, тіней, згладжування, дистанції перегляду, текстур, візуальних ефектів і відображення загалом. Такі можливості надають користувачу гнучкість у керуванні графічною складовою програми, що особливо важливо для дослідницьких цілей і оптимізації продуктивності.

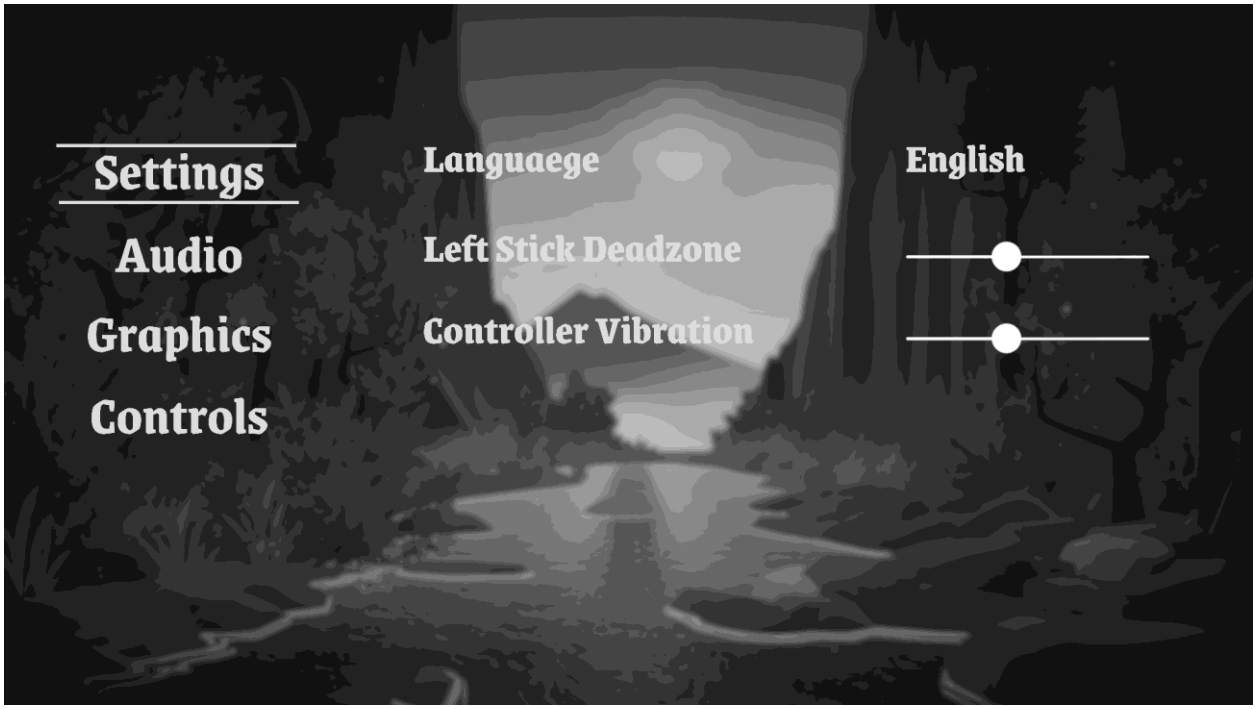

Рисунок 4.3 — Головний екран меню налаштувань

У разі, якщо нові налаштування не задовольняють користувача або викликають технічні проблеми, передбачена функція відновлення попередніх параметрів, що забезпечує безпеку і комфорт у роботі з програмою.

Для покращення зручності користування кожне налаштування супроводжується інформаційним вікном з детальним описом його функції, впливу на роботу системи та потенційних наслідків змін.

Окремо в меню передбачена кнопка «Exit Game», яка відповідає за вихід із програми. Перед завершенням роботи користувачу виводиться вікно з підтвердженням виходу, що запобігає випадковому закриттю застосунку (рис. 4.4). У разі, якщо нові налаштування несподівано виявляться незручними або викличуть технічні проблеми, передбачена функція миттєвого відновлення попередніх параметрів — достатньо одного кліку, щоб повернутися до випробуваної конфігурації й продовжити роботу без ризику збоїв. Для максимальної прозорості та зручності використання кожна опція супроводжується інформаційним вікном із детальним описом її призначення,

впливу на продуктивність системи та можливих наслідків зміни — це дає змогу користувачам чітко уявити, як кожен параметр позначиться на загальному досвіді. Окремий пункт меню «Exit Game» захищає від випадкових закриттів: натискання кнопки викликає спливаюче вікно з підтвердженням виходу, а перед цим автоматично зберігаються поточні налаштування, щоб жодна зміна не була втрачена збоїв.

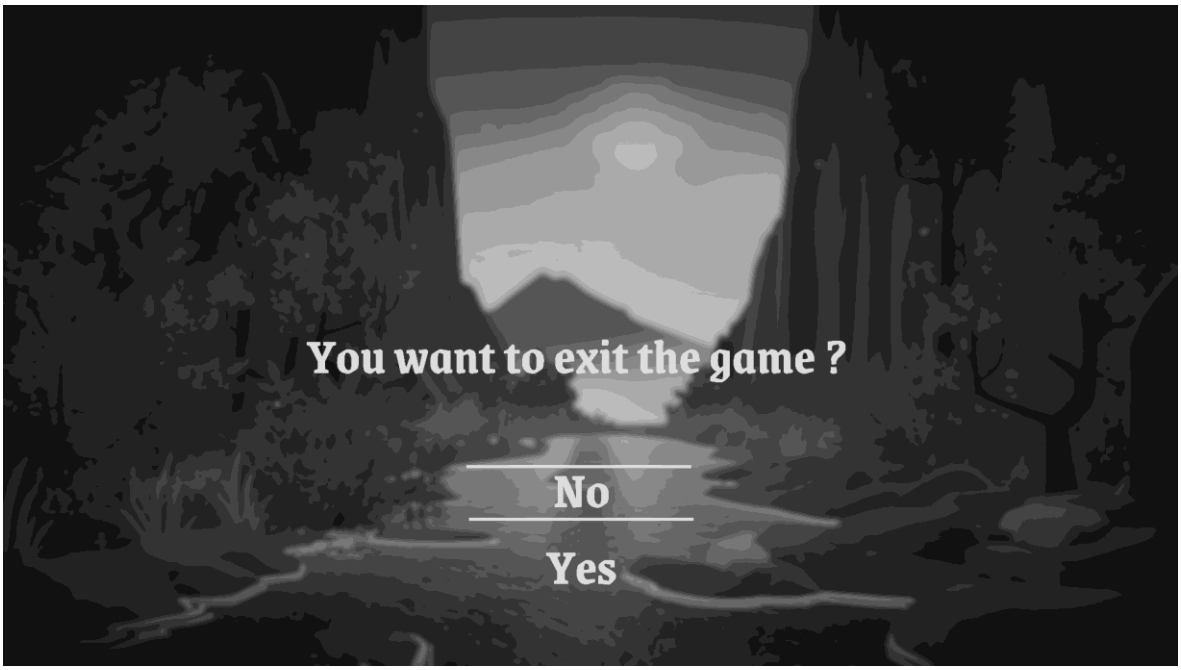

Рисунок 4.4 — Екран контекстне меню

Коли користувач знаходиться у головному вікні програми та натискає клавішу «Escape», на екрані відображається меню паузи (рис. 4.5), яке накладається поверх основного інтерфейсу. Це меню дозволяє користувачу виконувати низку дій, серед яких:

— перейти до вікна налаштувань. При виборі цього пункту поверх головного екрана відкривається відповідне меню налаштувань, де користувач може змінити параметри програми;

— вийти до головного меню. Обравши цю опцію, користувач повертається безпосередньо до головного меню програми;

— закрити меню паузи.

Також на (рис. 4.6) показаний скрін ігрового процесу.

Рисунок 4.5 — Меню паузи

Рисунок 4.6 — Ігровий процес

Графічний інтерфейс користувача було реалізовано за допомогою

інструментів розробки, доступних у середовищі Unreal Engine. Для створення інтуїтивно зрозумілого та гнучкого інтерфейсу було використано технологію Common UI, яка забезпечує ефективне створення елементів керування, адаптивну верстку під різні платформи та полегшує масштабування інтерфейсу в межах проєкту. Такий підхід дозволив забезпечити зручну навігацію, швидкий доступ до основних функцій та високу якість візуального оформлення.

4.4 Системні вимоги

Системні вимоги — перелік технічних характеристик апаратного забезпечення та програмного середовища, необхідних для інсталяції, запуску та стабільної роботи програмного продукту. Вони поділяються на два основні типи: мінімальні та рекомендовані. Такий поділ дозволяє враховувати різні конфігурації пристроїв користувачів і гарантувати належний рівень сумісності та продуктивності в залежності від можливостей системи.

У процесі розробки програмного забезпечення системні вимоги є ключовим елементом як на етапах планування й проєктування, так і під час тестування та впровадження. Вони забезпечують розробників орієнтиром для оптимізації коду, а користувачів — інформацією про те, чи зможе їхнє обладнання коректно працювати з програмою.

Мінімальні системні вимоги визначають базовий набір ресурсів, за якого програма здатна запуститися та виконувати основні функції. Хоча при такій конфігурації продуктивність може бути обмеженою, і частина функціоналу може працювати із затримками або зі зниженими параметрами якості, забезпечується доступність продукту для ширшого кола користувачів, зокрема власників старіших або малопотужних пристроїв. Таким чином, визначення мінімальних системних вимог є важливим фактором для підвищення охоплення потенційної аудиторії.

У свою чергу, рекомендовані вимоги відображають технічні характеристики, за яких продукт працює з максимальною ефективністю. Вони орієнтовані на забезпечення стабільної продуктивності, швидкої обробки даних, безперебійного відтворення графіки та повноцінного використання всіх доступних функцій. Для складних, ресурсоємних програм — таких як 3D-ігри, графічні редактори чи професійні інструменти моделювання — рекомендовані вимоги є критично важливими для досягнення високої якості взаємодії з продуктом.

Отже, чітке формулювання системних вимог дозволяє не лише підвищити рівень користувацького досвіду, а й сприяє ефективному поширенню та застосуванню програмного забезпечення в різноманітних технічних умовах.

Мінімальні вимоги до програмного продукту наведені нижче:

— Операційна система – Windows 8/10 64-bit
— Процесор – 2.4GHZ Чотирьохядерний процесор або кращий
— Оперативна пам’ять – 4 ГБ
— Відеокарта – Intel HD Graphic 4000 серії або еквівалент
— Місце на диску — 500 Мб доступного місця

Рекомендовані системні вимоги визначають технічну конфігурацію, яка забезпечує найвищу ефективність роботи програмного забезпечення, розкриваючи повний потенціал усіх його функцій. Вони гарантують комфортне користування продуктом без затримок, збоїв чи зниження продуктивності, що особливо актуально у випадку ресурсоємних додатків — зокрема, комп’ютерних ігор, програм для обробки графіки або відео, а також інструментів професійного призначення. Дотримання таких вимог дозволяє забезпечити плавну анімацію, швидку обробку даних, стабільну роботу інтерфейсу та підтримку високої якості графіки чи обчислень. Завдяки цьому користувач отримує максимально позитивний досвід взаємодії з програмою,

незалежно від складності виконуваних завдань.

Рекомендовані вимоги до програмного продукту наведені нижче:

— операційна система – Windows 10/11 64-bit;

— процесор 3.2 GHZ восьмиядерний процесор або кращий;

— оперативна пам’ять — 8 Гб;

— відеокарта GeForce GTX 640 або еквівалент;

— місце на диску 500 Мб доступного місця.

Мінімальні системні вимоги забезпечують стабільну роботу програми на більшості сучасних пристроїв, гарантують базову функціональність і доступність для широкого кола користувачів. У свою чергу, рекомендовані характеристики забезпечують найкращу продуктивність та повний доступ до всіх функцій і можливостей застосунку. Такий підхід дозволяє адаптувати продукт до різних апаратних конфігурацій, створюючи комфортні умови використання та сприятливий користувацький досвід незалежно від технічного забезпечення.

ВИСНОВКИ

У результаті виконання бакалаврської роботи було розроблено повноцінний ігровий продукт на базі одного з найпотужніших сучасних рушіїв — Unity, орієнтований на кінцевого користувача. Основною метою роботи було створення гри, яка не тільки б забезпечувала розважальний функціонал, а й включала елементи навчання, логічного мислення, розвитку когнітивних навичок та занурення у сюжетну лінію через проходження головоломкових рівнів. Успішна реалізація всіх поставлених завдань свідчить про досягнення мети та підтверджує актуальність обраної тематики.

Одним із ключових технічних рішень під час реалізації проєкту стала відмова від сторонніх плагінів і використання виключно внутрішніх інструментів Unity. Це забезпечило кілька важливих переваг:

— оптимізацію продуктивності;

— спрощення налагодження та супроводу проєкту;

— повну контрольованість кожного елемента геймплею;

— високу сумісність з різними платформами без потреби у додаткових адаптаціях.

Такий підхід дозволив зробити розробку не тільки технологічно завершеною, а й гнучкою, масштабованою та легко модифікованою для подальших етапів розвитку або створення нових ігрових проєктів.

В основі геймплею реалізовано механіку зміни розміру об’єктів як головний рушій взаємодії з навколишнім середовищем. Це рішення дозволяє будувати неординарні рівні з головоломками, які вимагають від гравця креативного підходу до вирішення задач. Крім цього, в грі реалізовано сюжетну складову, яка розкривається поступово на кожному рівні, що посилює ефект занурення у гру та створює додаткову мотивацію для гравця проходити гру до кінця.

Значну увагу під час розробки було приділено модульності коду,

повторному використанню скриптів через систему префабів та інтеграції інтерфейсів користувача для інтуїтивного керування. Кожен компонент гри було спроектовано таким чином, щоби його легко можна було змінити чи використати в інших проєктах. Так, наприклад, реалізована система стрибків, руху камери, керування персонажем, обробки подій взаємодії з об’єктами — усе це зроблено із використанням патернів програмування, що сприяє оптимізації ресурсоємних операцій і зменшенню навантаження на графічний конвеєр.

Окрему роль у візуальному оформленні відіграє система частинок, яка використовується для створення ефектів, що підсилюють емоційне сприйняття сцен. Це дозволяє зробити гру привабливішою та «живішою», надає відчуття глибини та динаміки, навіть при роботі зі статичними об’єктами.

Ще однією сильною стороною створеного ігрового продукту є його розвивальний потенціал. Гра орієнтована не лише на задоволення від процесу гри, але й сприяє розвитку мислення, вміння аналізувати ситуації в умовах обмеженої інформації, приймати рішення, що базуються на логіці, причинно-наслідкових зв’язках і спостереженнях. У грі спеціально реалізовано елементи, що змушують гравця пробувати різні підходи до проходження рівня, вчитися на власних помилках та адаптуватися до нових умов.

Особливо важливим є формування навичок рефлексії — осмислення своїх дій у грі, спроба зрозуміти, чому попередній вибір не спрацював, і що можна змінити для досягнення результату. Такий тип ігрового процесу несе високу освітню цінність, адже допомагає тренувати навички, що можуть бути корисними і в реальному житті — від адаптивності до планування дій у складних умовах.

Завдяки грамотному використанню функцій Unity, зокрема системи анімацій, фізичного рушія, UI, обробки подій, вбудованої підтримки шейдерів

та інструментів оптимізації, вдалося створити гру, яка відповідає сучасним вимогам до інді-проєктів: технічно довершена, зручна для користувача, гнучка у налаштуванні та цікава для проходження.

Таким чином, виконання цієї бакалаврської роботи стало не лише демонстрацією володіння інструментами Unity, а й комплексним підходом до проектування, моделювання, програмування та дизайну гри. Проєкт доводить, що з використанням правильних інженерних рішень можна створити конкурентоспроможний ігровий продукт, який має освітній, пізнавальний і розважальний потенціал одночасно.

Unity підтвердив свою ефективність як багатофункціональний рушій для створення 2D і 3D-ігор, зручно інтегруючи можливості розробки, тестування й оптимізації в одному середовищі. У майбутньому проєкт може бути розширений новими рівнями, інтеграцією мультиплеєра або застосуванням нових механік без кардинальних змін архітектури, що демонструє його масштабованість.

Отже, результатом бакалаврської роботи стало не лише створення гри як програмного продукту, а й здобуття досвіду повного циклу розробки ігор — від ідеї до реалізації, від технічних налаштувань до дизайну користувацького досвіду. Проєкт може бути успішно використаний як база для подальших досліджень у сфері геймдизайну, когнітивної психології, освітніх технологій, а також як демонстраційна робота для портфоліо розробника.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity ігровий рушій [Електронний ресурс] Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html

2. Створення ігрових об’єктів [Електронний ресурс] Режим доступу до ресурсу: https://docs.unity3d.com/Manual/GameObjects.html

3. C# опис мови [Електронний ресурс] Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/

4. Налаштування рендеру [Електронний ресурс] Режим доступу до ресурсу: https://docs.unity3d.com/Manual/render-pipelines.html

5. Створення Shader [Електронний ресурс] Режим доступу до ресурсу: https://docs.unity3d.com/Manual/Shaders.html

6. Mean Squared Error Wiki [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Mean_squared_error

7. Speeded Up Robust Features Wiki [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Speeded_up_robust_features

8. Unity [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/products/unity-engine

9. Unreal Engine Experience Definition [Електронний ресурс] – Режим доступу до ресурсу: https://x157.github.io/UE5/LyraStarterGame/Experience

10. HUD Wiki [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/HUD_(video_games)

11. C++ Wiki [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/C%2B%2B

12. Blueprints Unreal Engine Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://is.gd/tAWJTl

13. Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/

14. What is render resolution [Електронний ресурс] – Режим доступу до ресурсу: https://is.gd/Vgk0gB

15. Дідковська М.В. Процес тестування програмного забезпечення – Київ: Київський політехнічний інститут, 6 с.

16. GPU Visualizer Unreal Engine Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://is.gd/cc64KF

17. Usability 101: Introduction to Usability [Електронний ресурс] – Режим доступу до ресурсу: https://www.nngroup.com/articles/usability-101-introduction-to-usability

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д..

«21» березня 2025 року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«Комп’ютерна гра-головоломка на ігровому рушії Unity»

08-54.БКР.002.00.000 ТЗ

Керівник phd, ст, виклад, каф. ОТ

Обертюх М. Р.

Студент групи 1КІ-21б

Бондар А. М.

ВНТУ 2025

1 Підстава для виконання комплексної бакалаврської кваліфікаційної роботи БКР

1.1 Актуальність теми полягає у стрімкому розвитку індустрії ігор та зростанні інтересу до ігор з головоломками та наративною складовою. Створення гри з інтерактивним геймплеєм, пов'язаним із зміною розміру об'єктів у Unity, є актуальною та такою, що відображає сучасні тенденції геймдизайну та розробки.

1.2 Наказ ВНТУ №97 про затвердження теми БКР від 20.03.2025.

2 Мета БКР і призначення розробки

2.1 Мета роботи — створення 3D-гри жанру Puzzle/Adventure із геймплеєм, що базується на зміні розмірів об'єктів та різноманітних інтерактивних рівнях.

2.2 Призначення розробки — створення технічно завершеної та модульної гри з освітнім та розвивальним потенціалом.

3 Вихідні дані для виконання БКР

3.1 Базові знання середовища Unity та мови програмування C#.

3.2 Принципи роботи фізичного рушія Unity та системи анімацій.

3.3 Основи 3D-моделювання, використання префабів та UI-елементів.

3.4 Методи оптимізації ігрового процесу, зокрема шейдери та GPU-навантаження.

3.5 Сучасні підходи до інтерактивного дизайну рівнів і геймплею.

4 Вимоги до виконання БКР

4.1 Провести аналіз сучасних підходів до розробки інді-ігор.

4.2 Створити ігровий світ із механіками зміни розміру об'єктів та інтерактивними тригерами.

4.3 Реалізувати функціонал взаємодії гравця з об'єктами, включаючи логіку натискань, переміщення та масштабування.

4.4 Забезпечити оптимізацію роботи гри шляхом застосування шейдерів, системи частинок та налаштування рівнів графіки.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати наведено в таблиці А.1

Таблиця А.1 — Етапи БКР

№ з/п	Назва етапу	Початок	Кінець	Очікувані результати
1	Аналіз задачі	21.03.2025	30.03.2025	Визначення мети, завдань та інструментів реалізації
2	Аналітичний огляд технологій	31.03.2025	10.04.2025	Огляд інструментів Unity, аналіз геймдизайну
3	Проєктування гри	11.04.2025	20.04.2025	Схема архітектури, логіка механік
4	Розробка ключових механік та геймплею	21.04.2025	30.04.2025	Реалізація руху, UI, масштабування, тригерів
5	Тестування та оптимізація	01.05.2025	10.05.2025	GPU-аналіз, баланс між продуктивністю й графікою
6	Оформлення пояснювальної записки та презентації	11.05.2025	20.05.2025	ПЗ, графічні матеріали, презентація

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні та ілюстративні матеріали, лістинг програмного забезпечення, протокол

попереднього захисту на кафедрі, відгук наукового керівника, рецензія, анотації українською та іноземною мовами.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно з установленими термінами. Захист БКР відбувається на засіданні екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення та порядок виконання БКР

8.1 При оформленні БКР використовуються:

— ДСТУ 3008:2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302:2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— інші документи, зазначені у вищевказаних.

8.2 Порядок виконання БКР викладено в «Положенні про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи:_____ Комп'ютерна гра-головоломка на ігровому рушії Unity _

Тип роботи:_____бакалаврська кваліфікаційна робота_____
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ_____кафедра обчислювальної техніки_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібностітекстових запозичень, виявлених у роботі системою StrikePlagiarism__6,5__%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату,фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішенняпро оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявленоознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____Азаров О.Д., зав. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

____Крупельницький Л.В., доц. каф ОТ_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку_____ ___Захарченко С.М.____
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник_____ _____Обертюх М.Р._____
(підпис) (прізвище, ініціали, посада)

Здобувач_____ _____Бондар А.М.____
(підпис) (прізвище, ініціали)

ДОДАТОК В

Графічна частина

КОМП’ЮТЕРНА ГРА-ГОЛОВОЛОМКА НА ІГРОВОМУ РУШІЇ UNITY

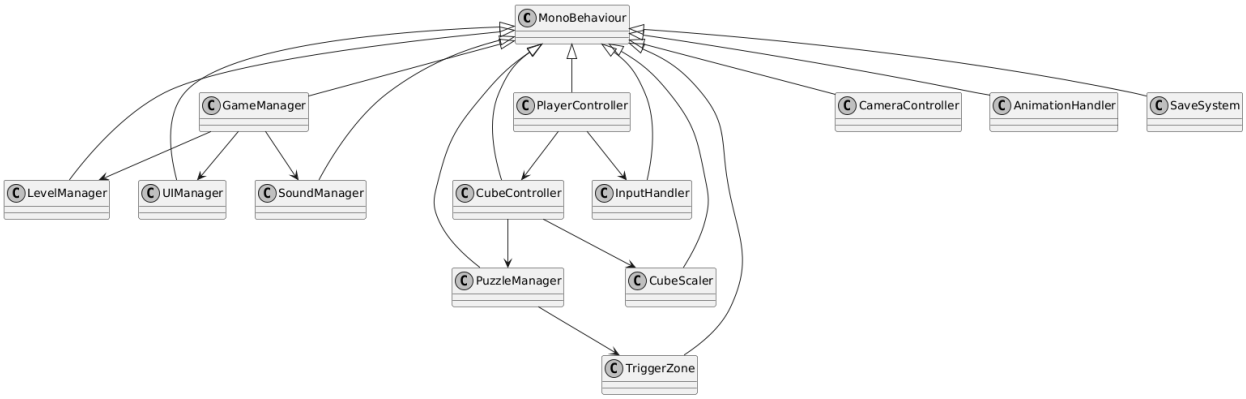

Рисунок В.1 — UML-діаграма класової структури скриптів

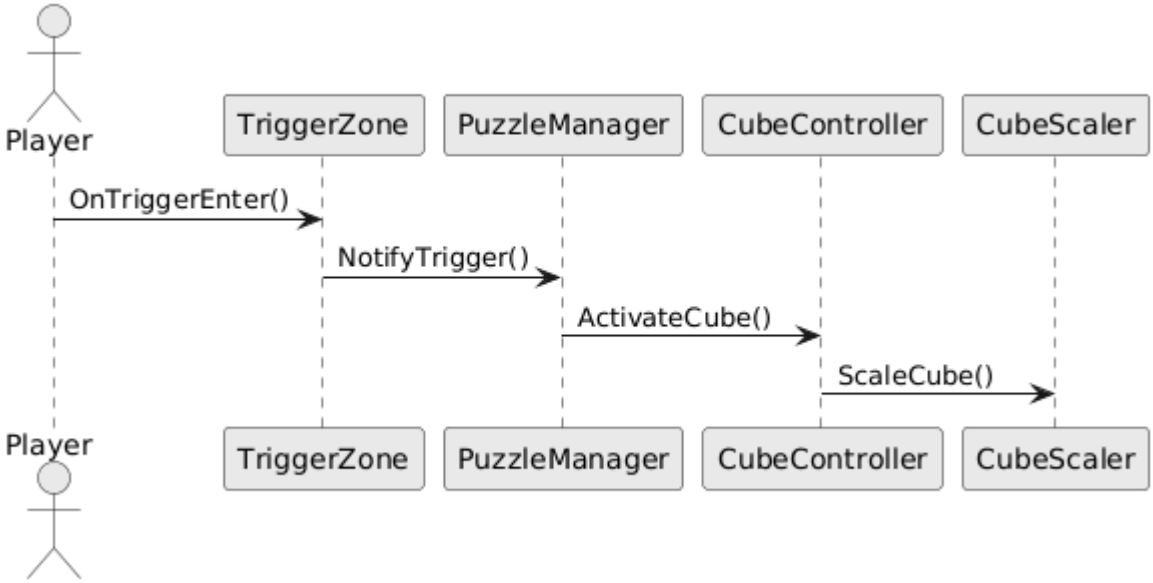

Рисунок В.2 — Діаграма послідовності обробки TriggerZone

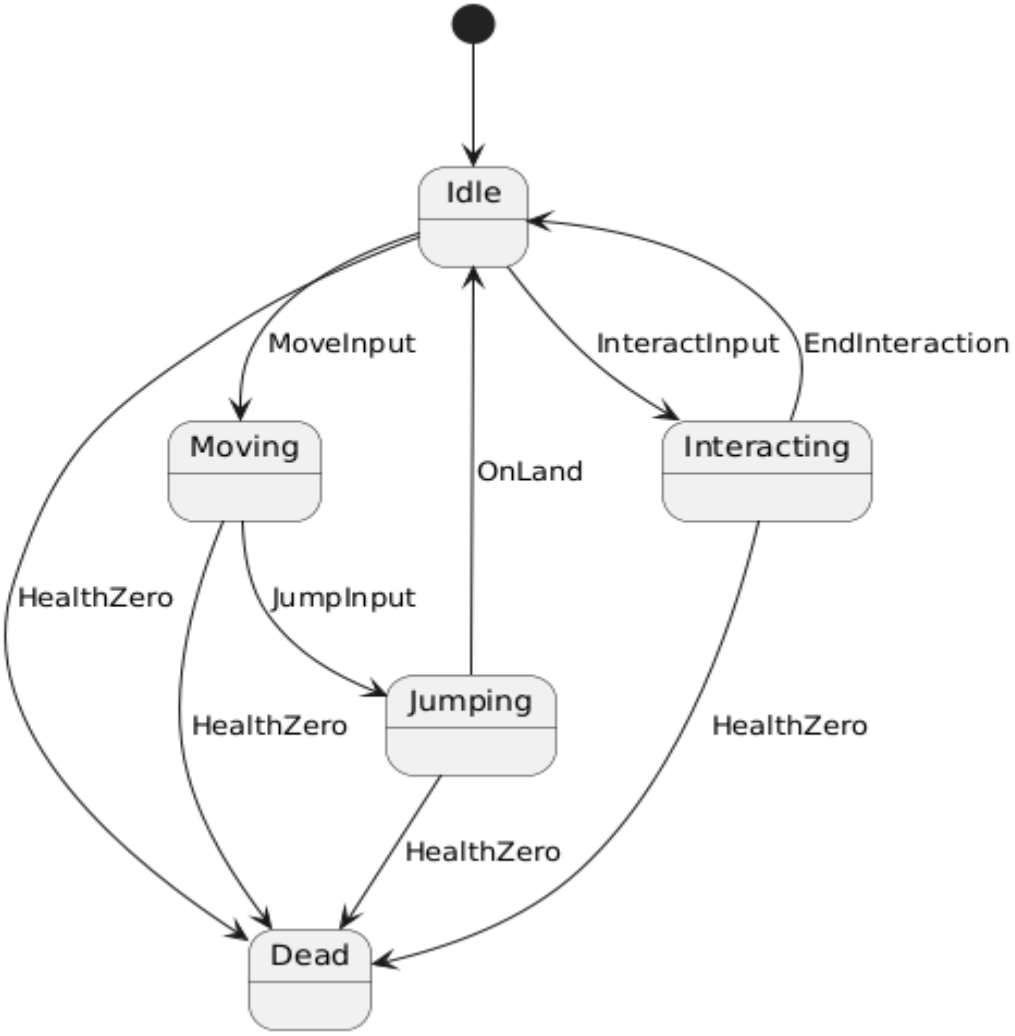

Рисунок В.3 — Діаграма послідовності обробки Idle

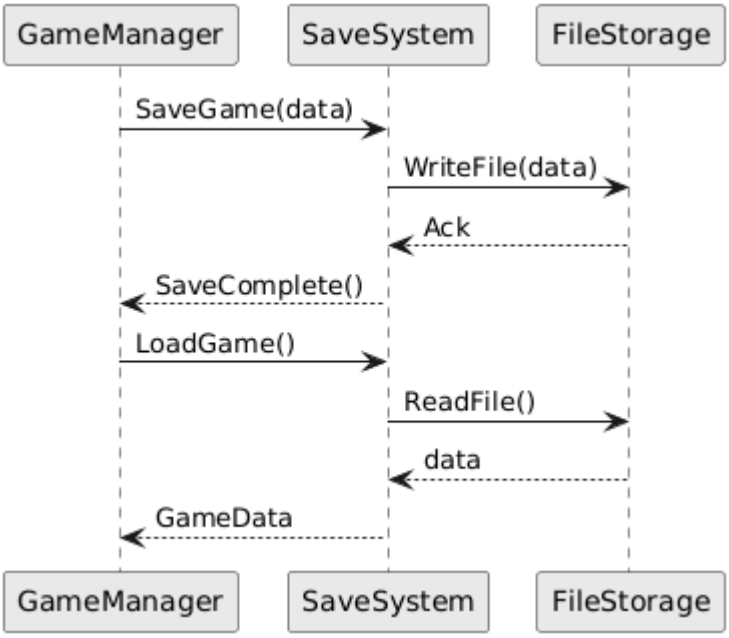

Рисунок В.4 — Послідовність викликів для збереження та завантаження гри

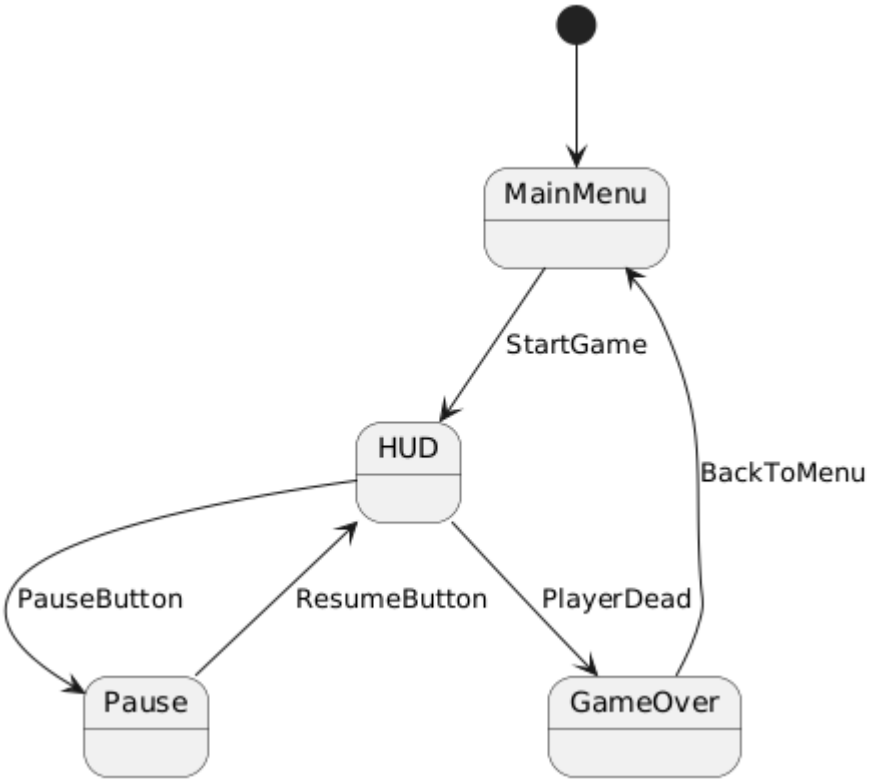

Рисунок В.5 — Діаграма переходів між UI-панелями

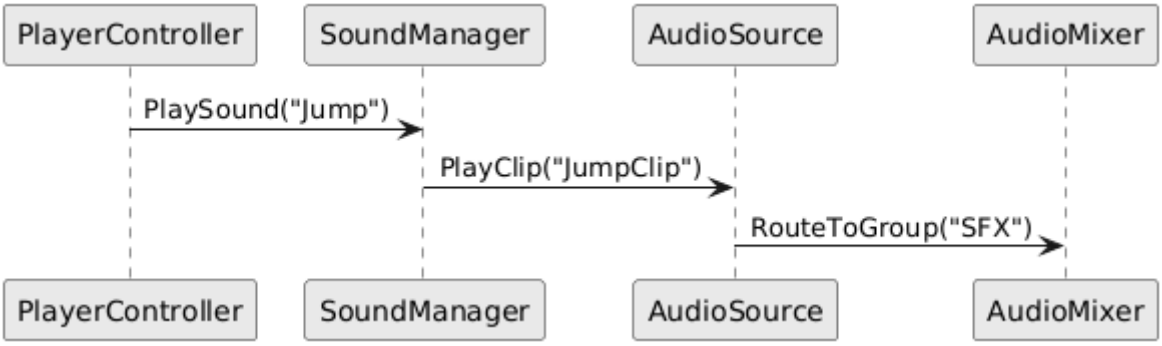

Рисунок В.5 — Послідовність відтворення звукового ефекту через SoundManager

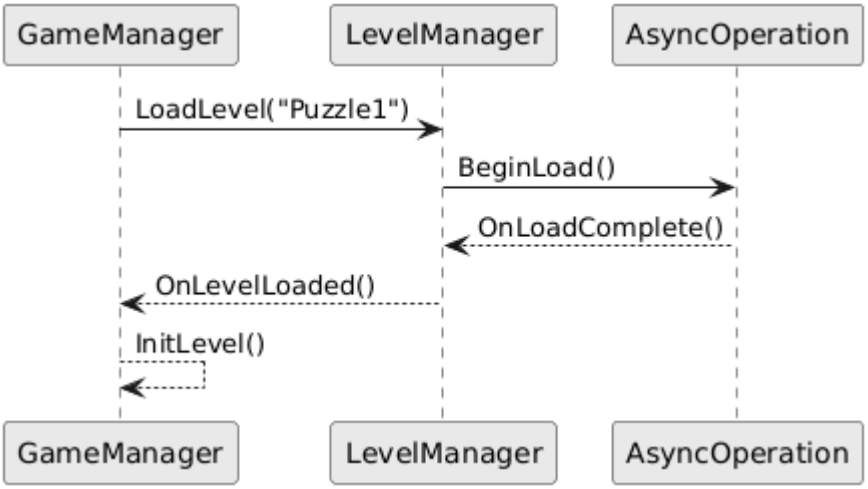

Рисунок В.6 — Послідовність завантаження та ініціалізації рівня

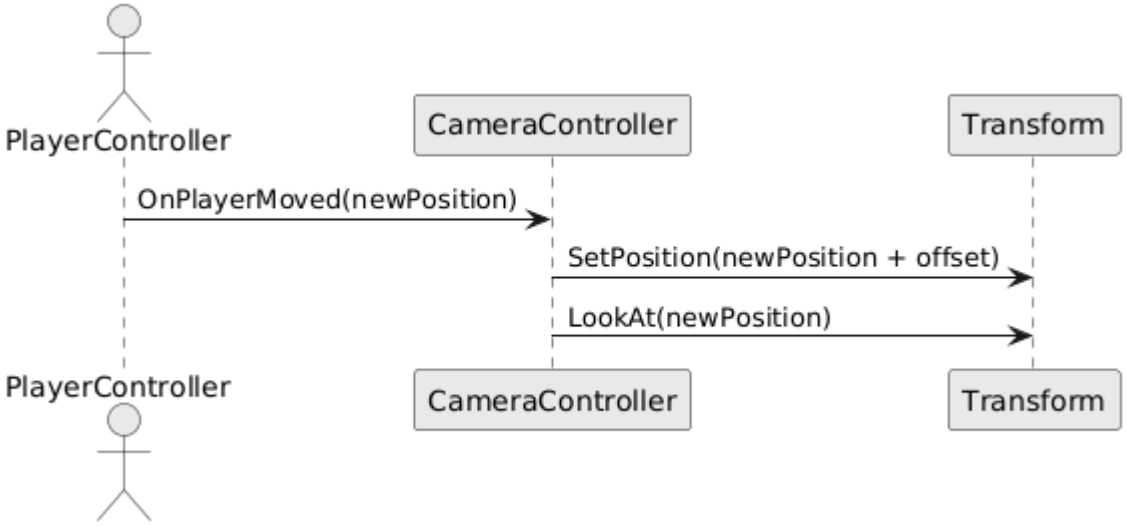

Рисунок В.7 — Послідовність оновлення положення та орієнтації камери через CameraController