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

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

на тему: «Розробка методів і засобів 3D-гри з використанням двигуна Unity»

Виконав: студент II курсу
групи 3ПІ-22м спеціальності
121 – Інженерія програмного забезпечення
Воронін Євген Сергійович

Керівник: к.т.н., доц. каф. ПЗ Войтко В.В.
«08» грудня 2023 р.

Опонент: к.т.н., доц. каф. ОТ Савицька Л.А.
«08» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«08» грудня 2023 р.

ВНТУ – 2023

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

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

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

Вороніну Євгену Сергійовичу

1. Тема роботи – Розробка методів і засобів 3D-гри з використанням двигуна Unity.
Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від « 18 » вересня 2023 р. № 247.

2. Строк подання студентом роботи
5 грудня 2023 р.

3. Вихідні дані до роботи: середовище розробки Unity, мова розробки C#, система керування базами даних – Photon, операційна система – Windows 11, метод аналізу дій користувача та метод автоматичного підбору кімнат.

4. Зміст текстової частини: вступ; аналіз стану розвитку ігрових програм та постановка задач розробки; розробка методу, моделей та алгоритмів гри в жанрі FPS Shooter; розробка 3D гри з використання двигуна Unity; тестування програмного додатку; економічна частина; висновки; перелік посилань; додатки.

5. Перелік ілюстративного матеріалу: блок-схеми алгоритмів роботи додатку; структурні схеми додатку; тестування застосунку.

6. Консультанти розділів робот

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Войтко В.В., к.т.н., доц. кафедри ПЗ	19.09.2023	01.12.2023
5	Причепа І.В., к.е.н., доцент	22.11.2023	01.12.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку навчальних ігор та постановка задач дослідження	20.09.2023 30.09.2023	Вик.
2	Розробка методу та моделей гри	01.10.2023 17.10.2023	Вик.
3	Розробка гри HeillGunner за допомогою двигуна Unity	18.10.2023 04.11.2023	Вик.
4	Тестування програми	05.11.2023 21.11.2023	Вик.
5.	Економічна частина	22.11.2023 01.12.2023	Вик.

Студент _____ **Воронін Є. С.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Войтко В.В.**
(підпис) (прізвище та ініціали)

Анотація

УДК 004.912.032.26

Воронін Є. С. А. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 157 с.

На укр. мові. Бібліогр.: назва: розробка методів і засобів 3D-гри з використанням двигуна Unity; рисунків: 50; таблиць 11.

У магістерській кваліфікаційній роботі подано результати розробки гри у жанрі FPS Shooter із дослідженням технології розширеного підбору опонентів для гравців з різним досвідом та навичками. Удосконалено методи розробки ігрового додатку з використанням алгоритмів адаптивного пошуку опонентів з моніторингом показників ігрового процесу та урахуванням рівня досягнень гравця.

Магістерська кваліфікаційна робота містить аналіз аналогів, розроблених на двигуні Unity, з огляду на їх логіку підбору завдань для різних типів гравців. Розроблено моделі прецедентів взаємодії гравців та додатку, 3D моделі ігрових об’єктів та ігрового середовища, алгоритми роботи ігрових процесів, програмні модулі реалізації ігрової стратегії.

Розроблена комп’ютерна гра «HeillGunner» реалізована на двигуні Unity з використанням технології Photon SDK та мови програмування C#.

Створено ігрове 3D середовище з використанням ігрових кімнат. Гра виконує ігрову стратегію жанру FPS Shooter в мультиплеєрному режимі, проводить оцінку навичок гравців та надає можливість підбору суперників з урахуванням рівня досягнень кожного учасника, що забезпечує оптимальні умови для тренувань у середовищі ігрового процесу. У роботі проведено тестування та дослідження загального функціонування ігрової системи.

Ключові слова: відеогра, матчмейкінг, FPS, Shooter, Unity.

Abstract

Voronin E. S. Development of methods and means of 3D game implementation using the Unity engine. Master's qualification work on specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. 157 p.

In Ukrainian speech Bibliography: title: development of methods and means of 3D game implementation using the Unity engine; drawings: 50; Tables 11.

The master's thesis presents the results of the development of a game in the genre of FPS Shooter with a study of the technology of advanced selection of opponents for players with different experience and skills. The methods of developing a game application have been improved using algorithms for adaptive search of opponents with monitoring of gameplay indicators and taking into account the level of the player's achievements.

The master's qualification work contains an analysis of analogues developed on the Unity engine, considering their logic for selecting tasks for different types of players. Models of precedents of interaction between players and the application, 3D models of game objects and game environment, algorithms of game processes, software modules for game strategy implementation have been developed.

The developed computer game "HeillGunner" is implemented on the Unity engine using the Photon SDK technology and the C# programming language.

A 3D game environment was created using game rooms. The game implements the game strategy of the FPS Shooter genre in the multiplayer mode, evaluates the skills of the players and provides the opportunity to select opponents taking into account the level of achievement of each participant, which provides optimal conditions for training in the gameplay environment. In the work, testing and research of the general functioning of the game system was carried out.

Keywords: video game, matchmaking, FPS, Shooter, Unity.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Відеоігри – це популярний вид розваг у сучасному світі. Їх виникнення пов'язане з розвитком комп'ютерних технологій і відкриттям нових можливостей для взаємодії з віртуальним середовищем [1].

Існує безліч різних жанрів відеоігор, таких як стратегії, головоломки, гонки, спортивні ігри, симулятори, навчальні ігри та багато інших. Ця різноманітність дозволяє кожному гравцю знайти ігри, що відповідають його інтересам, потребам і вподобанням.

Ігрова індустрія у наш час є однією із найбільших площадок у сфері розваг на рівні кінематографу. Ринок постійно поповнюється новими більш технологічними та вишуканими з художньої точки зору ігровими програмами, зокрема, в жанрі FPS Shooter, який є одним із найпопулярніших та високобюджетних на світовому ринку. Існує навіть офіційна спортивна дисципліна та змагання з FPS Shooter, які визнав олімпійський комітет у 2017 році. FPS Shooter є відносно новим і відноситься до інтелектуальних та командних видів спорту.

FPS Shooter відзначається інтенсивним і динамічним геймплеєм, де головний акцент робиться на бойових діях, пригодах та взаємодії з оточуючим світом від першої особи. У більшості таких багатокористувацьких проєктів головною проблемою постає баланс між рівнем гри і навиками гравців, що, у свою чергу, створює прірву між вмінням та досягненнями новачків і досвідчених гравців.

Тому актуальною є розробка відеогри у жанрі FPS Shooter, в яку буде інтегрована система аналізу дій гравців під час ігрової сесії з метою покращення процесу підбору опонентів для наступних ігор. Це дозволить створити зручні умови для гри, де новачки можуть набирати досвід, а досвідчені гравці гратимуть із суперниками вищого рівня, що дозволить їм удосконалювати свої навички.

Система аналізу дій гравців збиратиме дані про їхню активність, стратегії гри, успішність, що стане основою для розумного підбору гравців у наступних ігрових сесіях. Перевагою такої системи є можливість забезпечення балансу між рівнями гравців в ігровому процесі. Це створить зручне середовище для навчання та вдосконалення навиків гравців без непропорційного випадкового підбору суперників.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до плану науково-дослідних робіт кафедри програмного забезпечення.

Мета та задачі дослідження. Метою роботи є підвищення якості процесу підбору суперників в мультиплеєрних іграх шляхом удосконалення методів та засобів аналізу дій гравця під час ігрових сесій, які враховують статистику росту його навичок, що дозволить автоматизувати процес підбору ігрових опонентів для забезпечення відповідних умов тренувань у середовищі ігрового процесу.

Основними задачами роботи є:

- визначити засоби моніторингу ігрового процесу та аналізу дій гравця;
- розробити метод аналізу та обробки дій гравця для коректного підбору ігрових суперників;
- розробити метод автоматичного підбору ігрових кімнат з урахуванням рівня досягнень гравця;
- розробити моделі та алгоритми роботи ігрової системи;
- розробити програмні модулі компонентів відеогри в жанрі FPS Shooter;
- провести тестування ігрової системи.

Об'єктом дослідження є процес розробки 3D гри з використанням двигуна Unity.

Предметом дослідження є методи та засоби реалізації 3D гри в жанрі FPS Shooter.

Методи дослідження. Для реалізації поставлених задач було використані:

- методи створення відеоігор на двигуні Unity та для розробки компонентів ігрової системи;
- методи аналізу та обробки дій гравця для ведення статистики його досягнень;
- методи теорії алгоритмів для розробки алгоритмів роботи ігрових процесів;
- методи побудови компонентів гри та реалізації їхніх взаємозв'язків для створення загальної ігрової системи;
- методи проєктування програмного забезпечення для виконання розробки програмного ігрового додатку.
- методи тестування програмних систем для підтвердження їх працездатності.

Наукова новизна отриманих результатів:

- подальшого розвитку отримав метод аналізу та обробки дій гравця, який, на відміну від існуючих, орієнтований на ведення статистики досягнень з урахуванням результатів параметричного аналізу ігрової сесії в багатокористувацькому режимі з використанням ресурсів сервера PUN, що дозволяє ведення моніторингу ігрового процесу та здійснення аналізу ключових параметрів у реальному часі для формування рекомендованого підбору суперників для наступного ігрового змагання [1].

- подальшого розвитку отримав метод автоматичного підбору ігрових кімнат, який, на відміну від існуючих, враховує рівень досягнень гравця за результатами моніторингового аналізу його дій у попередній ігровій сесії, що дозволяє забезпечити зручні умови проведення ігрових змагань.

Практичне значення одержаних результатів. Розроблена відеогра «HeillGunner» може використовуватися гравцями в індивідуальному режимі та в багатокористувацькому режимі для проведення FPS Shooter змагань. Гра може бути

розміщена на таких площадках продажу відеоігор для настільних пристроїв, як: Steam, GOG та EGS.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У науковій роботі, опублікованій у співавторстві, автору належить розробка моделі та алгоритму роботи ігрової програми.

Апробація матеріалів магістерської кваліфікаційної роботи. Результати магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ - 2023».

Публікації. Основні результати дослідження опубліковані в тезах доповіді на міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ - 2023» [4].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку використаних джерел, що містить 33 найменування, 4 додатки. Робота містить 50 ілюстрацій, 11 таблиць.

1 АНАЛІЗ РОЗВИТКУ ІГРОВИХ ПРОГРАМ ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз розвитку ігрових програм

Відеоігри – це популярний вид розваги у сучасному світі. Їх виникнення пов'язане з розвитком комп'ютерних технологій і відкриттям нових можливостей для взаємодії з віртуальним середовищем [2]. Відеоігри стали важливою частиною культури та розважальної індустрії, пропонуючи гравцям безмежну кількість пригод та можливість виражати себе у віртуальних світах.

Однією з найважливіших рис відеоігор є їхня здатність спонукати до розвитку різних навичок. Граючи в ігри, гравці можуть розвивати логічне мислення, реакцію, стратегічне мислення і навіть соціальні навички, коли вони грають у багатокористувацьких онлайн іграх та спілкуються з іншими гравцями. Ця можливість розвитку навичок робить відеоігри корисними для розвитку людини.

Як і в більшості продуктів, які стосуються індустрії розваг, кожен користувач обирає свій вид розваг. Відеоігри є інтерактивним продуктом, де гравець бере пряму участь у реалізації своєї ігрової стратегії. Тут, як і в кіноіндустрії, на кожен жанр знайдеться свій поціновувач.

Жанри відеоігор відіграють важливу роль у визначенні структури та стилю гри. Це різноманітні категорії, які вказують на специфіку геймплею та сюжету. Один і той же жанр може включати ігри з різним рівнем складності, тематикою та атмосферою. Серед популярних жанрів відеоігор [3] виділимо:

- Екшн. Цей жанр відомий своєю динамікою і енергійним геймплеєм. Гравцям пропонуються завдання, пов'язані з боротьбою, стріляниною та пригодами. Ігри в жанрі екшн можуть бути як від третьої особи, так і від першої особи і зазвичай акцентують увагу на рефлексах та навичках гравця.

- Рольові ігри (RPG). Цей жанр відзначається глибокою розробкою персонажів, розвитком сюжету та можливістю впливу на події у грі. Гравцям доводиться створювати та розвивати свого персонажа, обирати його навички та приймати рішення, які впливають на хід ігрової історії.

Існує безліч інших жанрів відеоігор, таких як стратегії, головоломки, гонки, спортивні ігри, симулятори та багато інших. Ця різноманітність дозволяє кожному гравцю знайти ігри, які відповідають його інтересам та вподобанням, та забезпечує багатий досвід в світі відеоігор.

Розберемо детальніше секцію жанрів екшн як базову для розвитку FPS Shooter ігор, орієнтованих на ведення ігрової стратегії від першої особи.

"Action" (екшн) є одним із найпопулярніших та найрізноманітніших жанрів відеоігор. Він відзначається інтенсивним і динамічним геймплеєм, де головний акцент робиться на бойових діях, пригодах та взаємодії з оточуючим світом. Серед ігор жанру екшн окремо виділяють жанри як складові підвиди базового жанру:

- Шутери (Shooter). Цей тип екшн-ігор зосереджений на бойових діях і використанні зброї. Гравці керують персонажем і ведуть бій зі злочинцями, ворогами або іншими героями гри. Шутери поділяються на ігри від першої особи (FPS – First-Person Shooter) та ігри від третьої особи (TPS – Third-Person Shooter).

- Платформери (Platformer). Цей піджанр гри акцентується на переміщенні героя через складні рівні, подоланні перешкод, збиранні предметів та розв'язанні головоломок. Найвідомішими представниками цього жанру є серія ігор "Super Mario" та "Sonic the Hedgehog".

- Бойовики (Beat 'em up). У цих іграх головний герой бере участь у бійках або батл-роялях проти численних ворогів, використовуючи різні прийоми та комбінації. Популярні бойовики включають серію ігор "Street Fighter" і "Mortal Kombat".

- Спеціалізовані Action-ігри. Окрім розглянутих типів ігрової продукції, існують ігри, які спеціалізуються на конкретних видах бойових дій або пригод. Наприклад, гра "Dark Souls" відзначається важкими боями і високою складністю, а гра "Assassin's Creed" комбінує бойові сцени зі стелс-елементами.

- Екшн-пригоди (Action-Adventure). Цей піджанр поєднує елементи екшну і пригод. Гравцям доводиться розв'язувати головоломки, досліджувати світи та брати участь у бойових сценах. Прикладами є серія ігор "Tomb Raider" та "Uncharted".

Жанр "Action" надає гравцям можливість переживати захоплюючі ігрові досвіди, де їх реакція та навички мають важливу роль. Цей жанр постійно еволюціонує та пропонує нові інновації у світі відеоігор, привертаючи широку аудиторію гравців.

У сучасному світі комп’ютерні ігри – це один із популярних способів для людей проводити дозвілля, що створює великий попит корситувачів на ігрову продукцію та появу великих бізнесів, які займаються створенням високоякісної цифрової продукції для массового користувача.

Серед основних платформ для ігрових застосунків можна виділити такі:

- настільний комп’ютер (Windows, Linux, AppleOS);
- ігрові консолі (XBOX, PLAYSTATION, NINTENDO);
- мобільні платформи (IOS, Android, HarmonyOS).

1.2 Порівняльний аналіз аналогів

FPS (First-Person Shooter) – це піджанр екшн-ігор, де гравці керують персонажем від першої особи й можуть бачити ігровий світ черз очима свого персонажа. Головна особливість FPS – це бойові дії з використанням різних видів

зброї, таких як стрільба з гвинтівки, гранати та інше.

FPS-ігри можуть мати багато різних режимів гри, включаючи одиночну кампанію, мультиплеєрні батл-роялі, кооперативні режими і багато інших. Цей жанр завжди залишається популярним і захоплюючим для гравців, які шукають бойові відчуття й потребують змагання та хочуть спостерігати за ігровим світом від першої особи.

Оскільки розробка гри «HeillGunner» буде проводитися на базі рушія Unity, то для порівняння оберемо аналоги, реалізовані за цією технологією, а саме: «ULTRAKILL», «Mist Hunter», «SUPERHOT», «7 days to die», «The Forest».

"ULTRAKILL" – це шутер-гра від першої особи, яка поєднує в собі швидкий та динамічний геймплей з ретро-стилем, подібним до шутерів 1990-х років. Гра відзначається розширеним арсеналом зброї, можливістю активного руху та битвами з численними ворогами і босами [4]. Вона пропонує гравцям екстремальний досвід у світі шутерів від першої особи (рис. 1.1).

Гра розроблена однією людиною на ім'я Hakita і відзначається низкою особливостей:

1. швидким і динамічний геймплеєм: гра "ULTRAKILL" відома своєю швидкістю та динамікою, гравці можуть очікувати швидких бойових дій і використання різних видів зброї для боротьби з ворогами;
2. ретро-стилем: гра намагається відтворити відчуття і стиль старих шутерів з 1990-х років, має специфічний піксельний графічний стиль та відповідне звукове оформлення;
3. великим арсеналом зброї: пропонує гравцям різні види зброї, включаючи мечі, рушниці, лазери тощо, кожна зброя має свої унікальні властивості для можливості здійснення атак.

4. безперервним рухом і використанням стрибків: персонажам дозволяється активно рухатися, стрибати та виконувати акробатичні трюки під час бою, що додає до геймплею додатковий рівень складності;
5. динамічними рівнями і битвами з босами: ігри включають велику кількість різних рівнів, в яких гравцям належить здолати ворогів і виконати різні завдання.

Рисунок 1.1 – Гра «ULTRAKILL»

"Mist Hunter" – це відносно невелика інді-гра, що поєднує в собі елементи roguelike та top-down shooter. У грі гравець бере на себе роль мисливця на чудовиськ, який досліджує ігровий світ та бореться з небезпечними істотами у таємничому тумані. Гра пропонує різноманітний арсенал і можливість вдосконалювати свого персонажа [4]. Різноманітність ворогів та екшн-геймплей роблять "Mist Hunter" цікавою грою для шукачів пригод і викликів (рис.1.2).

Серед великого функціоналу гри виділимо:

- рогалик-елементи – гра використовує класичні елементи жанру roguelike, такі як випадково генеровані рівні, постійний ризик втрати прогресу після смерті персонажа та потреба розвивати навички і здібності знову;
- таємничий світ – гра "Mist Hunter" розгортається в таємничому світі, який вкритий туманом, з таємничими та небезпечними істотами, які чекають на гравця в кожному кроці;
- великий арсенал зброї – гравцям доступний широкий вибір зброї, щоб боротися з ворогами, зокрема, вогнепальної зброї, лазерів, гранат тощо;
- можливість вдосконалення персонажа – гравцям дозволяється покращувати і розвивати свого персонажа, збільшуючи його здоров'я, ману і навички;
- чудову графіку й атмосферу – гра відзначається якісною піксельною графікою та створює атмосферний світ, який захоплює гравця в таємничий і містичний амбіанс.

Рисунок 1.2 – Гра «Mist Hunter»

"SUPERHOT" – це унікальна відеогра, яка поєднує в собі жанри шутера від першої особи і головоломки. Основна особливість гри полягає в тому, що час у грі змінюється тільки тоді, коли гравець рухається [5]. Це створює унікальну стратегічну динаміку, де гравцеві потрібно планувати свої дії, уникати куль і ворогів і вчасно реагувати на ситуації.

Гру характеризує її розвинений функціонал:

1. Механіка "Час рухається лише, коли ви рухаєтеся".Це основна механіка гри. Час у "SUPERHOT" сповільнюється або навіть зупиняється, коли гравець стоїть на місці. Це дозволяє гравцеві планувати свої дії, уникати куль і ворогів та стратегічно долати кожен рівень.

2. Мінімалістичний стиль. Гра має абстрактну графіку з відмінно відтвореними червоними об'єктами, які ідентифікують ворогів та зброю, а інший світ чорно-білий. Цей стиль підкреслює асиметрію і нелінійність гри.

3. Сюжет та атмосфера. Гра має абстрактний сюжет, який обговорює поняття віртуальної реальності, контролю і підконтрольності. Гравець отримує повідомлення від анонімного користувача, який відправляє його в різні віртуальні сценарії.

4. Інноваційний підхід до геймплею. "SUPERHOT" видається навіть більш стратегічною грою, ніж типові шутери, і стимулює гравця думати перед кожним рухом. Гравець повинен планувати свої дії, використовуючи навколишні об'єкти і зброю, щоб подолати ворогів.

5. Велика кількість викликів. Гра має багато різноманітних рівнів і завдань, що стають складнішими з кожним новим етапом. "SUPERHOT" надає можливість гравцеві відчути себе справжнім супергероєм, подолавши неймовірні виклики.

"SUPERHOT" – це ігра, яка вирізняється серед інших своїм унікальним підходом до геймплею і створює незабутній іммерсивний досвід для гравців (рис.1.3).

Рисунок 1.3 – Гра «SUPERHOT»

"7 Days to Die" – це відеогра в стилі відкритого світу в жанрі survival horror. Гра розгортається в постапокаліптичному світі, де гравці намагаються вижити в умовах зомбі-апокаліпсису [6]. Основні особливості гри включають функції виживання, будівництва бази, збирання ресурсів, боротьби з зомбі й іншими ворогами, а також генерацію випадкової карти (рис. 1.4). Гра також пропонує можливість грати як в персональному, так і в кооперативному з іншими гравцями режимах.

Гру характеризує її базовий функціонал:

1. Постапокаліптичний світ. Гра відбувається в постапокаліптичному світі, зруйнованому зомбі-апокаліпсисом, де гравці повинні збирати ресурси і боротися за виживання в небезпечному середовищі.
2. Будівництво та розвиток. Гравці можуть будувати власні бази, підсилювати їх і розвивати, щоб захистити себе від зомбі й інших небезпек. Це включає в себе створення оборонних споруд, які допоможуть виживати в найгірших ситуаціях.
3. Змінний геймплей. Гра пропонує змінний геймплей, де кожен ігровий день відрізняється від попереднього. Щоразу, коли настає "сьомий день", відбувається масова атака зомбі, і гравці повинні готуватися до бою.
4. Збирання ресурсів і ремонт обладнання. Гравці повинні збирати ресурси з навколишнього світу, виготовляти предмети та ремонтувати зброю й інше обладнання.
5. Багатокористувацький режим. "7 Days to Die" пропонує можливість грати в кооперативному з іншими гравцями режимі або змагатися проти них.
6. Моди та спільнота. Гра підтримує модифікації і має активну спільноту гравців, які розробляють власні зміни і додатки до гри.

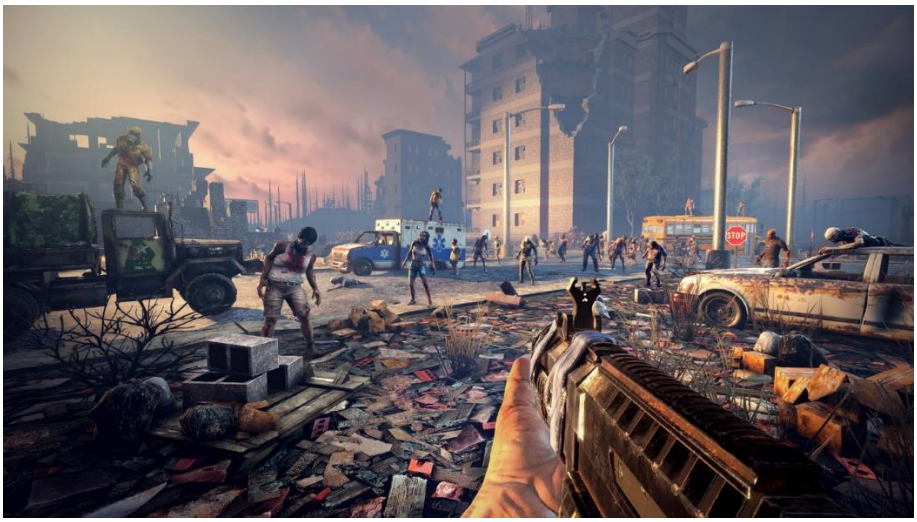

Рисунок 1.4 – Гра «7 Days to Die »

"The Forest" – це відеогра в жанрі відкритого світу і виживання, яка розгортається на острові, що заселений дикими і небезпечними істотами. Гравцям доводиться будувати укріплення, збирати ресурси, готувати їжу та боротися з ворогами, щоб вижити (рис. 1.5). Гра також пропонує індивідуальний режим та кооперативний режим гри разом з іншими гравцями [7].

- Гру "The Forest" характеризує її розширений функціонал:
- Виживання. Гравцям доводиться забезпечувати собі їжу, воду, притулок та інші ресурси, щоб вижити в небезпечному середовищі острова.
- Будівництво. Гра пропонує розвинену систему будівництва, де гравці можуть створювати різні типи споруд для захисту від ворогів і полегшення виживання.
- Ворожі істоти. Острів населяють надприродні і страшні істоти, які загрожують гравцям. Боротьба з ними є необхідною для виживання.
- Експлорація. Гравці можуть досліджувати різні частини острова, знаходячи ресурси, загублені артефакти та інші таємниці.
- Багатокористувацький режим. "The Forest" дозволяє грати у кооперативному режимі з іншими гравцями, що додає соціальний аспект до гри.

Рисунок 1.5 – Гра « The Forest »

Проаналізувавши аналоги, заповниму порівняльну таблицю (табл. 1.1), яка відображає головні характеристики ігрових програм.

Таблиця 1.1 – Порівняння аналогів

Назва гри / Характеристики	ULTRA KILL	Mist Hunter	SUPERHOT	7 Days to Die	The Forest	HeillGunner
Мультиплеєр	-	-	-	+	+	+
Кросплатформенність	-	-	+	-	+	+
Ігровий процес	+	+	+	+	+	+
Оптимізація	+	+	+	-	+	+
Безкоштовна гра	-	-	-	-	-	+
Реіграбельність	-	-	+	+	+	+

1.3 Постановка задач

Головним завданням магістерської кваліфікаційної роботи є створення високоякісного ігрового продукту зі зручним управлінням і простим інтерфейсом.

Гру «HeillGunners» буде розроблено на базі двигуна Unity 3D як платформи розробки та з використанням мови програмування C#.

Основними задачами роботи є:

- визначити засоби моніторингу ігрового процесу та аналізу дій гравця;
- розробити метод аналізу та обробки дій гравця для коректного підбору ігрових суперників;
- розробити метод автоматичного підбору ігрових кімнат з урахуванням рівня досягнень гравця;
- розробити моделі та алгоритми роботи ігрової системи;
- розробити програмні модулі компонентів відеогри в жанрі FPS Shooter;

- провести тестування ігрової системи.

З метою полегшення та зручного використання інтерфейсу потрібно створити меню з різними опціями, додати музичний супровід і надати можливість призупиняти гру з подальшим поверненням до головного меню гри.

Персонаж гравця буде знаходиться на невеличкій локації-арені, де також знаходитимуться його опоненти. Ціль гравця – набрати якнайбільше балів та перемогти усіх опонентів.

Коли персонаж гравця програє битву, він має відроджуватися, поки один з гравців не набере необхідну кількість балів для перемоги, після чого ігрова сесія завершується.

Для зручності усіх гравців має бути реалізовано супровідний функціонал:

- гра може бути призупинена у будь-який момент з можливістю переходу до головного меню;
- користувач на початку гри буде отримувати підказки;
- можливість переглянути таблицю з усіма гравцями та статистикою кожного гравця;
- можливість додати музичний супровід;
- можливість виключити музику і звуки.

Для запуску програми апаратне забезпечення має відповідати мінімальним вимогам:

- комп’ютер серії IBM PC з частотою 233 МГц і вище;
- 64МБ оперативної пам’яті;
- графічний адаптер SVGA (Super Video Graphic Adapter);
- відеокарта об’ємом пам’яті не менше 4МБ;
- ОС Windows Vista/7/XP;

- розмір дискового простору, що займає програма, 15 704 байт, розмір оперативної пам’яті, що займає програма, 3 500 КБайт.

1.4 Висновок

У першому розділі проведено аналіз розвитку ігрової індустрії. Здійснено порівняльний аналіз базових аналогів ігрової програми: "ULTRAKILL", "Mist Hunter", "SUPERHOT", "7 days to die", та "The Forest". Виявлені переваги й недоліки кожного аналогу дали змогу визначитися з вибором функціоналу власного ігрового застосунку.

Крім того, розглянуто різні жанри ігор, включаючи шутери (Shooter), платформери (Platformer), бойовики (Beat 'em up), спеціалізовані Action-ігри та екшн-пригоди (Action-Adventure), зокрема, в рамках жанру FPS Shooter як обраного для реалізації власної ігрової програми.

Проведено аналіз програмно-інформаційної підтримки при розробці розважальних комп’ютерних ігор, включаючи використання двигуна Unity 3D та мови програмування C#.

Сформульовано задачі, які необхідно виконати для розробки додатку.

2 РОЗРОБКА МЕТОДУ, МОДЕЛЕЙ ТА АЛГОРИТМІВ ГРИ В ЖАНРІ FPS SHOOTER

2.1 Аналіз інформаційного забезпечення ігрової програми

Розроблені теоретичні аспекти реалізації ігрової системи мають бути успішно використані в комп'ютерній грі "HeillGunners". Гра призначена для надання розважальних послуг чи може бути використана для тренування вправності ведення віртуального бою, швидкості реакції та вміння орієнтуватися на місцевості і передбачає отримання багатокористувацького досвіду в ігрвому середовищі. У "HeillGunners" відсутній сюжет, що робить її привабливою для тих, хто прагне отримати максимальне задоволення від шутер-гри в мультиплеєрному режимі.

Гра "HeillGunners" пропонує гравцям захоплюючі бої на невеликих картах-аренах, де кожен гравець грає сам за себе. Ця гра покликана розвивати навички реакції та стратегічного мислення, вимагаючи від гравців високого рівня майстерності та вміння виживати в безжальних боях. "HeillGunners" створена для того, щоб надати гравцям можливість насолоджуватися захоплюючим мультиплеєрним досвідом та відчути смак справжнього замагання.

Для створення кінцевого ігрового продукту потрібно провести розробку методів, моделей і алгоритмів роботи програми та реалізувати їх в програмному забезпеченні ігрової системи:

- створити інтерфейс, який легко зрозуміти користувачу;
- розробити методи, моделі та алгоритми гри;
- розробити графічні та анімаційні матеріали;
- здійснити проєктування гейм-дизайну;
- здійснити проєктування користувацького інтерфейсу;

- провести проєктування бази даних (БД) Pier-to-pier;
- розробити 3D моделі ігрових об’єктів та ігрового середовища;
- розробити програмні модулі ігрової систем та забезпечити їх взаємозв’язок та інтеграцію;
- провести тестування кінцевого програмного продукту.

2.2 Розробка структури інтерфейсу

Користувацький інтерфейс (або UI, скорочено від User Interface) – це спосіб взаємодії користувача з програмним продуктом або системою. Він включає в себе всі елементи, які дозволяють користувачеві взаємодіяти з програмою, зокрема: вікна, кнопки, меню, поля для введення тексту, іконки тощо. Користувацький інтерфейс розробляється таким чином, щоб бути легким у використанні, інтуїтивно зрозумілим та забезпечувати зручність та ефективність взаємодії гравця з ігровою системою [8].

Головним призначенням користувацького інтерфейсу є створення зручних та доступних умов для гравця, які дозволяють йому використовувати програму без зайвих труднощів і незручностей. Якщо інтерфейс добре спроєктований, користувач може легко виконувати завдання, взаємодіяти з програмою та досягати своїх цілей без зайвих зусиль і затрат часу [9].

Існують різні види користувацьких інтерфейсів, які використовуються в програмних продуктах і системах. Декілька основних видів включають [10]:

1. Графічний користувацький інтерфейс (GUI). Це найпоширеніший тип інтерфейсу, який використовує графічні об'єкти, такі як вікна, кнопки, меню, іконки та інші елементи для взаємодії з користувачем. GUI дозволяє користувачам виконувати дії за допомогою миші або сенсорного екрану.
2. Текстовий користувацький інтерфейс (TUI). Цей тип інтерфейсу

використовує текстові команди та введення для взаємодії з програмою. Текстовий інтерфейс найчастіше використовується в командних рядках або консольних додатках.

3. Інтерфейси віртуальної реальності (VR) та доповненої реальності (AR). Ці інтерфейси створюють іммерсивне оточення для користувача, де взаємодія відбувається віртуально або в поєднанні з реальним світом. Вони широко використовуються у відеоіграх, навчанні та симуляціях.
4. Голосовий інтерфейс. Цей тип інтерфейсу дозволяє користувачам взаємодіяти з програмою за допомогою голосових команд та розпізнавання мови. Голосові інтерфейси набирають популярності у голосових асистентах, мобільних додатках та інших сферах.
5. Сенсорний інтерфейс. Це інтерфейс, який взаємодіє з дотиком. Він використовує сенсорні дисплеї для керування та взаємодії з програмами, такими як смартфони та планшети.
6. Жестовий інтерфейс. У цьому типі інтерфейсу користувачі взаємодіють з програмою за допомогою жестів, таких як махання рукою, зум, обертання тощо. Він зазвичай використовується в сучасних сенсорних пристроях.
7. Дво- або багато-екранний інтерфейс. Використовує два або більше екрани для відображення інформації та взаємодії з користувачем. Цей тип інтерфейсу знайде застосування в мультимедійних пристроях та робочих столах.

Розглянуті типи користувацьких інтерфейсів комбінуються залежно від потреб конкретного програмного продукту та цільової аудиторії.

Інтерфейс гри «HeillGunners» передбачає доступ до головних засобів користувацької взаємодії, що включають:

- головне меню;

- меню паузи;
- інтерфейс ігрового процесу;
- таблицю статистики.

Головне меню (рис. 2.1) відображає заголовок «Game Servers» та таблицю із підібраними для гравця кімнатами або з можливістю створити власну кімнату.

Рисунок 2.1 – Головний екран

Ігровий екран (рис. 2.2) складається з активного ігрового екрану та форм для відображення кількості набоїв та здоров’я гравця.

Рисунок 2.2 – Ігровий екран

Екран паузи (рис. 2.3) складається з назви гри, кнопки для відновлення гри, кнопки для налаштувань та кнопки виходу в головне меню гри.

Рисунок 2.3 – Екран паузи

Таблиця статистики (рис. 2.4) складається з назви гри, імені гравця, кількості переможених опонентів, кількості власних поразок та набраних балів.

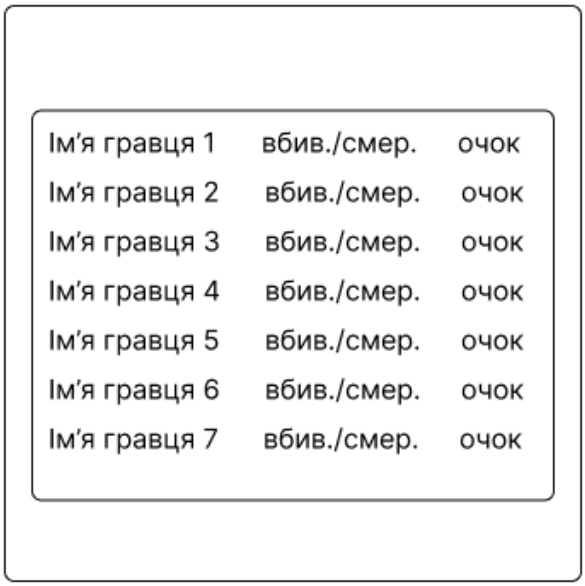

Рисунок 2.4 – Таблиця статистики

2.3 Розробка 3D моделей

Графіка в іграх включає в себе процес концептуалізації, що охоплює концепт-арт, який акумулює ескізи та ідеї, що формують візуальний стиль гри, а також комп'ютерну графіку, що використовується в грі [11]. Зазвичай художники, спеціалізовані на концептуальному мистецтві, тісно співпрацюють із гейм-дизайнерами для уточнення концепції, створюючи ескізи персонажів та оточення. Часто це включає і тривимірне моделювання. Після цього комп'ютерні художники беруть на себе створення персонажів та текстур, як для двовимірних спрайтів, так і для 3D-моделей, з яких формуються об'єкти в грі. Вони також відповідають за промальовку фонів й анімацію. Деякі компанії можуть мати фахівців, які відповідають за інтеграцію графіки в ігровий двигун [12].

Говорячи про тривимірну графіку, маємо справу зі створенням візуальних елементів, які надають грі глибину та об'єм. Тривимірна графіка включає в себе

процес моделювання тривимірних об'єктів та їх текстурування, щоб створити реалістичні об'єкти та оточення у віртуальному просторі [13].

Ключовим етапом є створення тривимірних моделей, які можуть бути персонажами, об'єктами чи оточенням у грі. Це включає в себе створення геометричних форм, надання їм текстур та матеріалів, які роблять об'єкти виглядом реалістичними. Після створення моделей робота включає в себе освітлення та анімацію, які надають рух та життя тривимірним об'єктам [14].

Процес створення тривимірної графіки може бути поділений на кілька етапів:

Моделювання. На цьому етапі розробники або 3D-моделювальники створюють геометричні форми об'єктів, які згодом з'являться у грі. Це включає створення об'єктів в тривимірному просторі, включаючи їх форму, розміри та структуру. Модель було розроблено у внутрішньому 3D редакторі Unity Зображення цього этапу наведено на рисунках 2.5, 2.6 та 2.7

Рисунок 2.5 – Модель персонажа (без текстур)

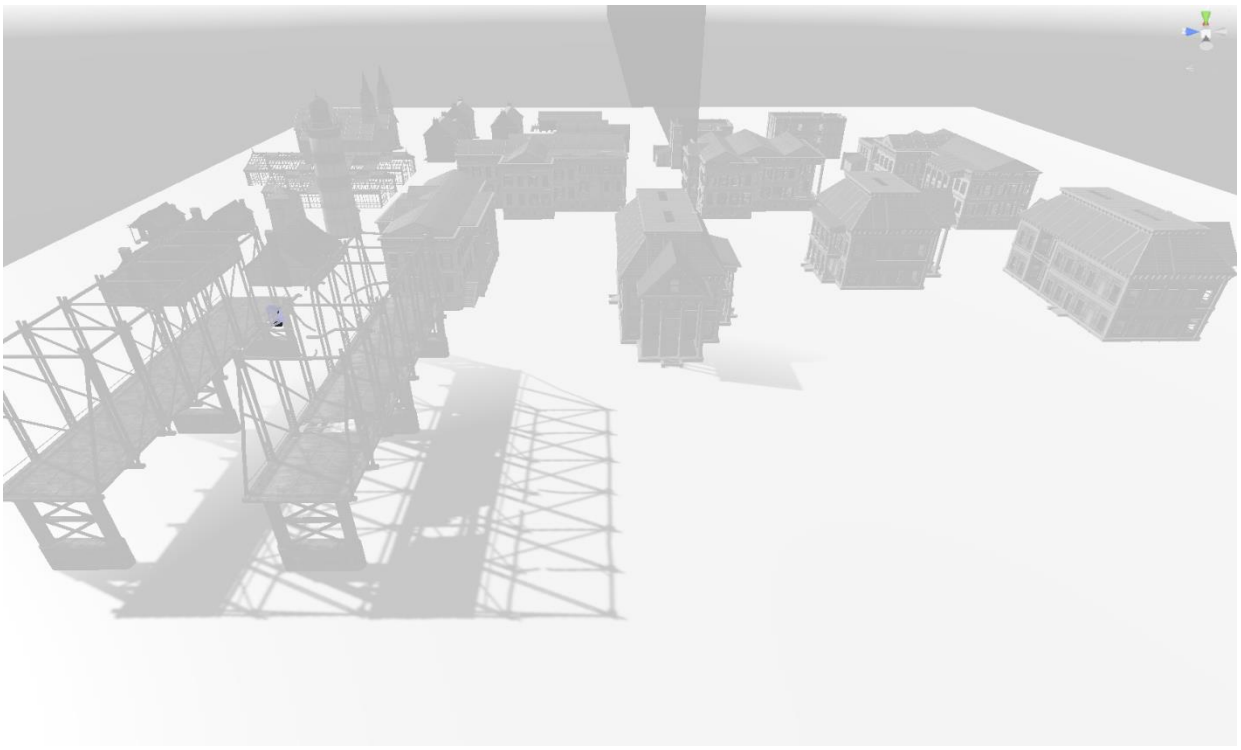

Рисунок 2.6 – Ігрова локація (без текстур)

Рисунок 2.7 – Модель зброї (без текстур)

Текстурування. Тривимірні моделі об'єктів отримують текстури, які надають їм вигляд та кольори. Це може включати в себе нанесення зображень, рисунків та текстур на поверхні об'єктів. Зображення цього этапу наведено на рисунках 2.8, 2.9 та 2.10.

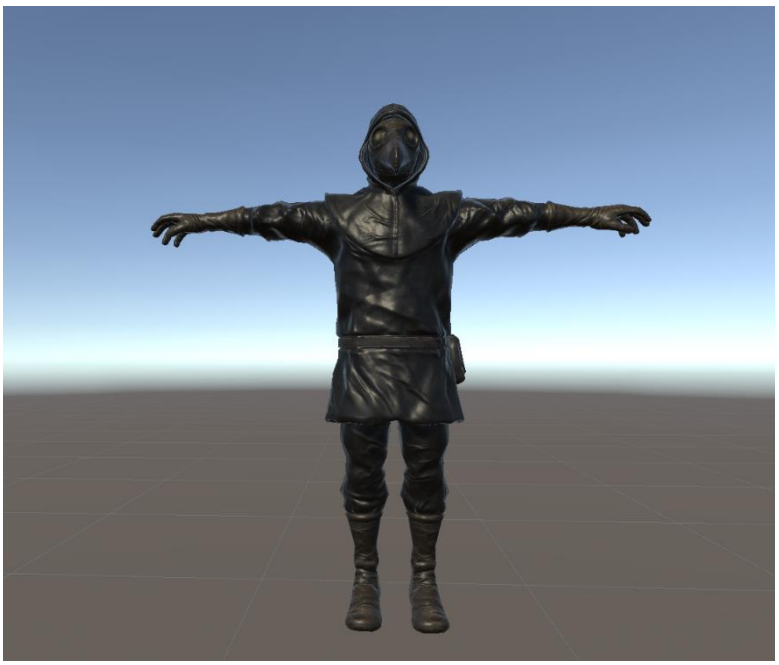

Рисунок 2.8 – Модель персонажа (з текстурами)

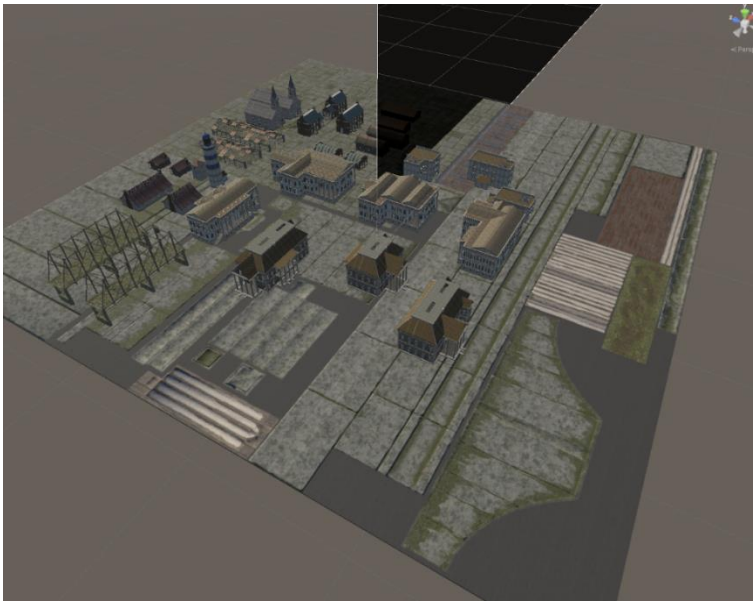

Рисунок 2.9 – Ігрова локація (з текстурами)

Рисунок 2.10 – Модель зброї (з текстурами)

Матеріали. Вибір для об'єктів різних матеріалів допомагає визначити їх фізичні властивості, такі як відбиття світла, прозорість чи матовість. Це додає реалізму графіці. Матеріали моделей гравця та зброї зображено на рисунку 2.11.

Рисунок 2.11 – Основні матеріали

Освітлення. Для створення реалістичного відображення тривимірних об'єктів у грі використовується освітлення. Це включає в себе розміщення джерел світла та визначення процесу взаємодії світла з об'єктами. Результат зображено на рисунках2.12 та 2.13.

Рисунок 2.12 – Модель гравця

Рисунок 2.13 – Модель зброї

Анімація. Об'єкти можуть бути анімовані для створення руху та інтерактивності у грі. Художники-аніматори створюють послідовність змін стану об'єктів для анімації їх руху, поведінки та дій.

2.4 Розробка моделі прецедентів взаємодії гравців та додатку

UML (Unified Modeling Language) – це мова моделювання, яка використовується для створення і візуалізації моделей програмних систем. UML надає стандартні засоби та нотацію для розробки діаграм, які допомагають інженерам розробки програмного забезпечення спроєктувати, аналізувати та документувати системи. UML використовується для спрощення спілкування між розробниками, архітекторами та стейкхолдерами проєкту [15].

UML включає в себе різні типи діаграм, такі як діаграми класів, діаграми послідовності, діаграми діяльності, діаграми прецедентів та багато інших. Кожен тип діаграми використовується для моделювання певних аспектів програмної системи. Наприклад, діаграми класів використовуються для моделювання структури системи, а діаграми послідовності - для показу послідовності операцій у системі [16].

Діаграма прецедентів (Use Case Diagram) у рамках UML (Unified Modeling Language) використовується для моделювання взаємодії між системою та зовнішніми агентами (користувачами чи іншими системами) з точки зору функціональності системи. Діаграма прецедентів допомагає визначити та відобразити ключові функціональні можливості системи та зовнішні взаємодії [17].

Основні елементи діаграми прецедентів включають у себе:

1. Прецеденти (Use Cases). Це функціональність або функціональні можливості системи, які доступні її користувачам чи іншим агентам. Кожен прецедент описує конкретну дію або послідовність дій, яку система може виконати.

2. Актори (Actors). Це зовнішні агенти, які взаємодіють з системою і використовують прецеденти. Актори можуть бути реальними користувачами, іншими системами або будь-якими сутностями, які мають ролі у взаємодії з системою.
3. Відносини (Relationships). Відносини між прецедентами та акторами визначають, які дії або послуги доступні для кожного актора. Зазвичай використовуються такі відносини:
    - включення (Inclusion) – показує, що один прецедент може включати інший прецедент;
    - розширення (Extension) – вказує, що один прецедент може розширювати інший прецедент;
    - успадкування (Generalization) – показує успадкування між прецедентами, де один є більш загальним, а інший – більш конкретним.

Діаграма прецедентів гри «HeillGunner» використовується для ілюстрації взаємодії між користувачами (акторами) та функціональними можливостями системи (прецедентами) з використанням візуальних зображень, як показано на рисунку 2.14. Ця діаграма надає візуальне уявлення про те, як різні користувачі взаємодіють з різними частинами гри та які функції доступні кожному актору. Кожен прецедент відображає конкретну функціональність гри, що може бути викликана користувачем, і дозволяє отримати уявлення про взаємодію між різними системними компонентами. Ця діаграма є важливим інструментом для розуміння обсягу та структури взаємодії в грі, сприяючи кращому проєкттуванню та реалізації функціоналу гри [18].

У додаток до цього, діаграма прецедентів гри допомагає уникнути можливих непорозумінь між розробниками та стейкхолдерами під час обговорення вимог до

гри. Комплексна візуалізація взаємодії відображає різноманітні сценарії використання, що сприяє більш глибокому розумінню функцій гри та її взаємодії з гравцями. Такий підхід покращує не лише процес розробки, але й якість взаємодії з грою для кінцевого користувача.

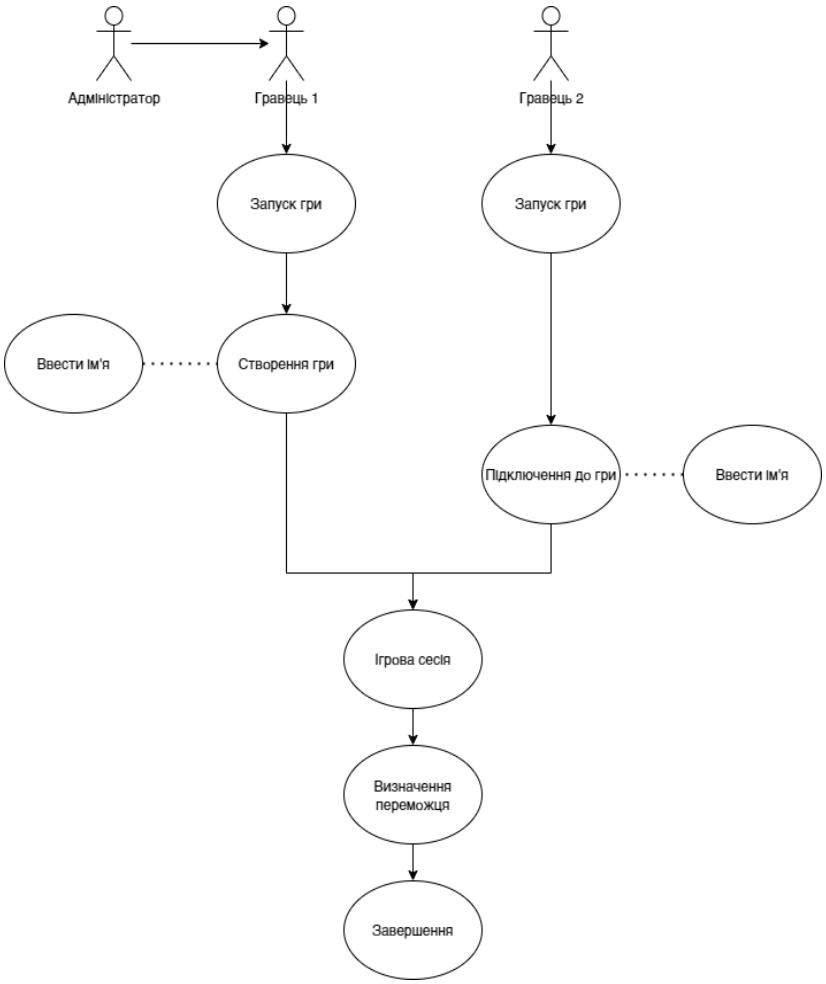

Рисунок 2.14 – Діаграма прецедентів

2.5 Розробка методу аналізу та обробки дій гравця з використанням ресурсів сервера PUN

Photon Unity Networking (PUN) – це розширення для Unity, яке надає широкий набір інструментів та сервісів для легкого та ефективного впровадження мережевого функціоналу у ваших ігрових проєкттах. Основним елементом PUN є

Photon Cloud – потужна хмарна платформа, яка надає серверні рішення для розподіленої обробки мережевого обміну даними між гравцями [19].

Основними компонентами PUN є:

- сервер PUN – центральна частина системи, відповідає за управління мережевим обміном даними та координацію між гравцями;
- API PUN – інтерфейс програмування додатків, який реалізується в коді Unity та надає доступ до функцій Photon Cloud для взаємодії з сервером.

Система авторитету сервера полягає в тому, що сервер PUN визначає, які дані вважаються "правдивими" у грі, щоб уникнути конфліктів та забезпечити синхронізацію між гравцями [20].

Масштабованість та продуктивність сервера PUN забезпечується функціоналом:

- сервер PUN може ефективно працювати з різними типами ігор, незалежно від їхнього обсягу та складності;
- з сервер PUN абезпечує високу продуктивність для гарантування плавності геймплею в режимі реального часу [21].

Лобі та Мастер-сервери. PUN дає можливість створювати лобі для гравців та використовувати мастер-сервер для відстеження доступних ігрових сесій [22].

Загальна ідея полягає в тому, що PUN використовується для оптимізації складного процесу розробки мережевих ігор, який надає інструменти та сервіси, що полегшують впровадження мережевого функціоналу та забезпечують ефективну спільноту гравців у реальному часі.

Схему роботи сервера PUN можна переглянути на рис. 2.15.

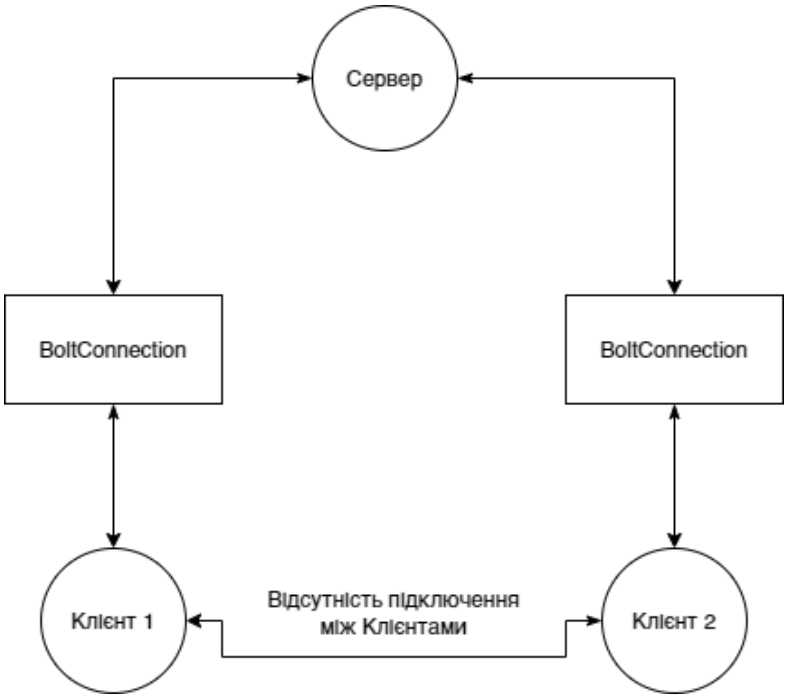

Рисунок 2.15 – Схема роботи сервера PUN

Bolt використовує модель, яку загалом називають "Клієнт/Сервер", що означає, що один комп'ютер вважається сервером, а всі інші є клієнтами і підключаються безпосередньо до сервера.

Розглянемо кілька важливих моментів, на які варто звернути увагу:

1. Сервер має один об'єкт BoltConnection на кожного клієнта. Усі підключення клієнтів можна знайти під BoltNetwork.clients. На самому сервері це поверне нуль елементів.
2. Клієнти мають лише одне BoltConnection, яке є з'єднанням з сервером. Таке з'єднання можна знайти за допомогою BoltNetwork.server. Ця властивість повертає null на самому сервері.
3. Не існує з'єднання, яке посилається на вас у будь-який спосіб, вони завжди являють собою посилання на інший комп'ютер.
4. Не існує прямого з'єднання між клієнтами, усі дані передаються через

сервер.

Розглянемо метод аналізу та обробки дій гравця, який орієнтований на ведення статистики досягнень з урахуванням результатів параметричного аналізу ігрової сесії в багатокористувацькому режимі з використанням ресурсів сервера PUN:

1. Збираємо дані про гравця. PUN (Photon Unity Networking) надає зручний спосіб синхронізувати стани гравців між клієнтами та сервером. Забезпечимо відстеження ключових параметрів, таких як координати гравця, напрямок його руху, стан здоров'я та стан амуніції. Використано PhotonCallbacks для обробки різних подій, що виникають у грі.
2. Визначаємо ключові події. Ретельно визначаємо події, які мають велике значення для геймплею. Це може включати стрільбу, попадання або промахи, зміну зброї та пересування в конкретні області. Важливо зафіксувати всі можливі впливи на гру.
3. Збір та обробка статистики. Створено систему для збору та зберігання статистики гравців, яка включає в себе не лише базову статистику, а й деталізовані дані про взаємодію гравців між собою.
4. Взаємодія з ігровим сервером. Використано Photon для передачі результатів аналізу на сервер. Встановлено двосторонню взаємодію між клієнтами та сервером для ефективного обміну даними та забезпечення коректності аналізу.
5. Реакція на аналіз. Механіки гри, які реагують на результати аналізу, що включає в себе винагороди або покарання для гравців, які демонструють високий рівень вмінь чи, навпаки, допускають багато помилок.

Загальну блок-схему алгоритму роботи методу аналізу та обробки дій гравця наведено на рисунку 2.16.

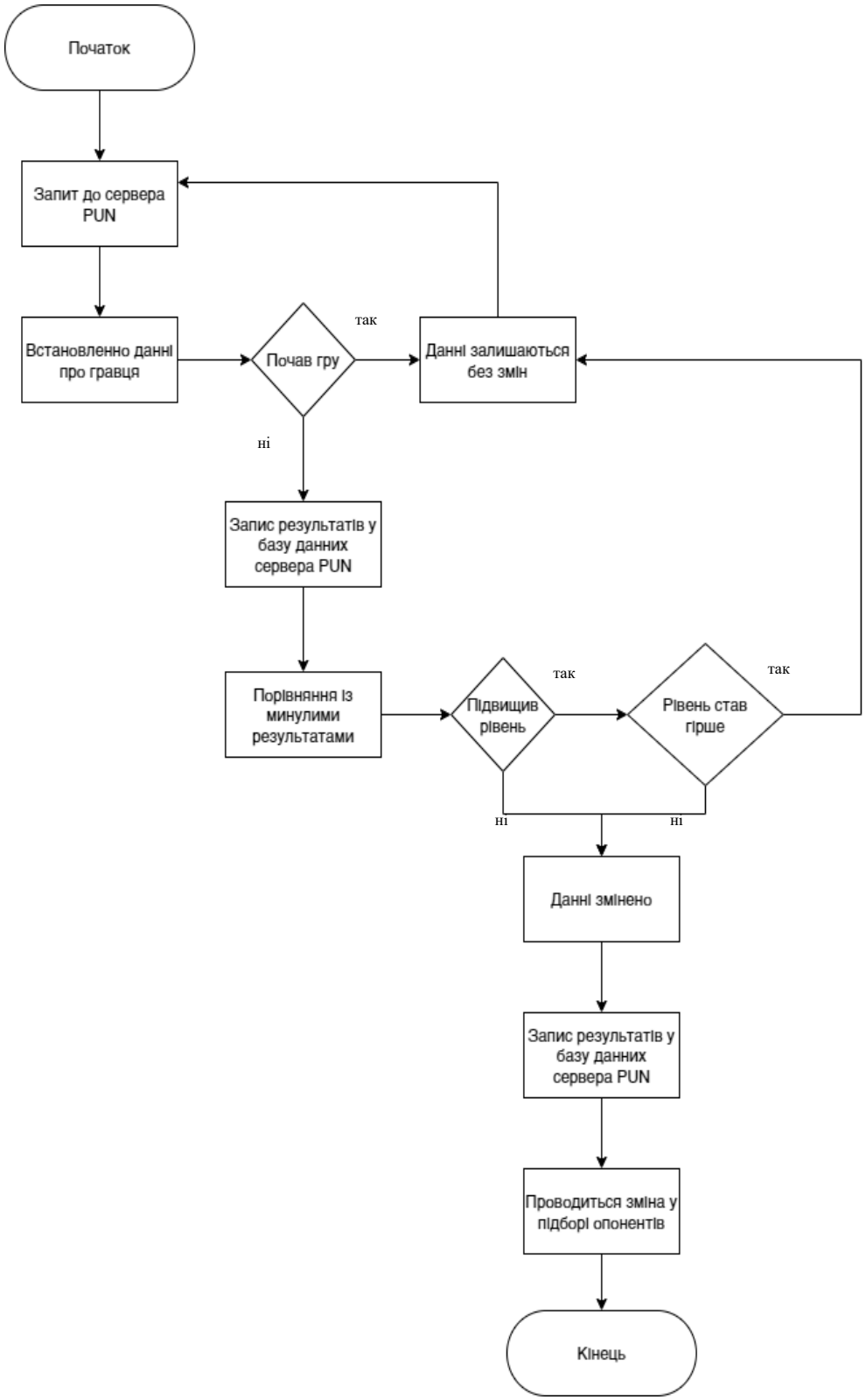

Рисунок 2.16 – Загальна блок-схема алгоритму роботи методу аналізу та обробки дій гравця

Запропонований метод аналізу та обробки дій гравця дозволяє ведення моніторингу ігрового процесу та здійснення аналізу ключових параметрів у реальному часі для формування рекомендованого підбору суперників для наступного ігрового змагання.

2.6 Розробка методу автоматичного підбору ігрових кімнат

Метод автоматичного підбору ігрових кімнат є важливим для досягнення оптимальної геймплейної динаміки та задоволення вимог різних гравців. Ця функціональність визначається ключовою у вирішенні низки проблем та вдосконаленні користувацького досвіду в ігрових середовищах [23]. Розглянемо деякі аспекти, що роблять метод автоматичного підбору ігрових кімнат важливим:

1. Зручність для гравців. Автоматичний підбір кімнат дозволяє гравцям швидко та без зусиль знаходити і приєднуватися до гри. Це полегшує їм вхід в ігровий світ і сприяє позитивному першому враженню від гри.

2. Ефективність розподілу ресурсів. Система автоматичного підбору дозволяє оптимізувати використання серверних ресурсів та забезпечує ефективний розподіл гравців по доступних кімнатах, що важливо для забезпечення стабільності гри й уникнення перевантажень.

3. Баланс гри. Здатність автоматично вибирати кімнати з врахуванням параметрів, таких як рівень гравців чи навички, сприяє створенню більш збалансованих і справедливих ігор. Це важливо для того, щоб уникнути ситуацій, коли досвід буває надто легким або надто складним.

4. Забезпечення соціальної взаємодії. Автоматичний підбір також сприяє соціальній взаємодії гравців, допомагаючи їм знаходити та взаємодіяти з іншими гравцями, які мають подібний рівень інтересів чи навичок.

5. Варіативність геймплею. Здатність автоматичного підбору адаптує геймплей до різних умов та вимог гравців. Це дозволяє створювати різноманітні

ігрові сценарії й надає можливість гравцям спробувати різні стилі гри.

Метод автоматичного підбору ігрових кімнат акумулює функціонал:

1. Збір даних. Система автоматичного підбору кімнат починає аналіз зі збору даних про навички гравців. Це може включати в себе рейтинг, статистику попередніх ігор, рівень навичок та інші важливі параметри.

2. Категоризація гравців. Гравці розподіляються за категоріями на основі експертизи їхнього рівня навичок. Наприклад, можна визначити категорії "початківці", "середні гравці" та "професіонали".

3. Визначення параметрів пошуку. Встановлюються параметри пошуку для автоматичного підбору ігрової кімнати. Це може включати в себе максимальний та мінімальний рівень навичок гравців, щоб забезпечити балансовані ігри.

4. Запуск пошуку. Система запускає пошук доступних ігрових кімнат, спираючись на визначені параметри. Мета пошуку полягає в тому, щоб знайти кімнату з гравцями, які мають схожий рівень навичок.

5. Відображення результатів. Гравцю надсилається інформація про знайдену кімнату, включаючи рівень навичок інших гравців та характеристики гри.

6. Приєднання. Гравець має можливість приєднатися до запропонованої кімнати або відхилити пропозицію та продовжити пошук.

7. Динамічне оновлення. Під час гри система динамічно оновлює рівень навичок гравців у кімнаті, враховуючи їхні досягнення та результати гри.

8. Автоматичне корегування. Залежно від результатів гри, гравці можуть бути перерозподілені між кімнатами для забезпечення постійного балансу.

9. Завершення ігрової сесії. Після завершення ігрової сесії гравець має можливість залишити кімнату, взяти участь у новій ігровій сесії чи вибрати інші ігрові опції.

2.7 Розробка алгоритму роботи ігрової системи

Алгоритм – це послідовний план або набір кроків, які потрібно виконати для досягнення поставленої мети. Алгоритм як основний інструмент програмістів дозволяє формалізувати й візуалізувати поетапний процес вирішення задачі. Алгоритм описується за допомогою програми, яка допомагає комп'ютеру розуміти, що саме потрібно робити і в якому порядку, щоб досягти бажаного результату [24].

Алгоритм роботи ігорової системи «HeillGunner» включає послідовність дій:

1. Відображається початковий екран під час запуску програми.
2. Відображається головне меню.
3. Запуск фонового музичного супроводу.
4. Пошук сервера.
5. Сервер знайдено/не знайдено.
6. Запуск гри.
7. Завершення гри перемогою/поразкою.
8. Повернення до головного меню.
9. Вихід з гри.

Блок-схему алгоритма роботи ігрової системи «HeillGunner» наведено на рис. 2.17 [1].

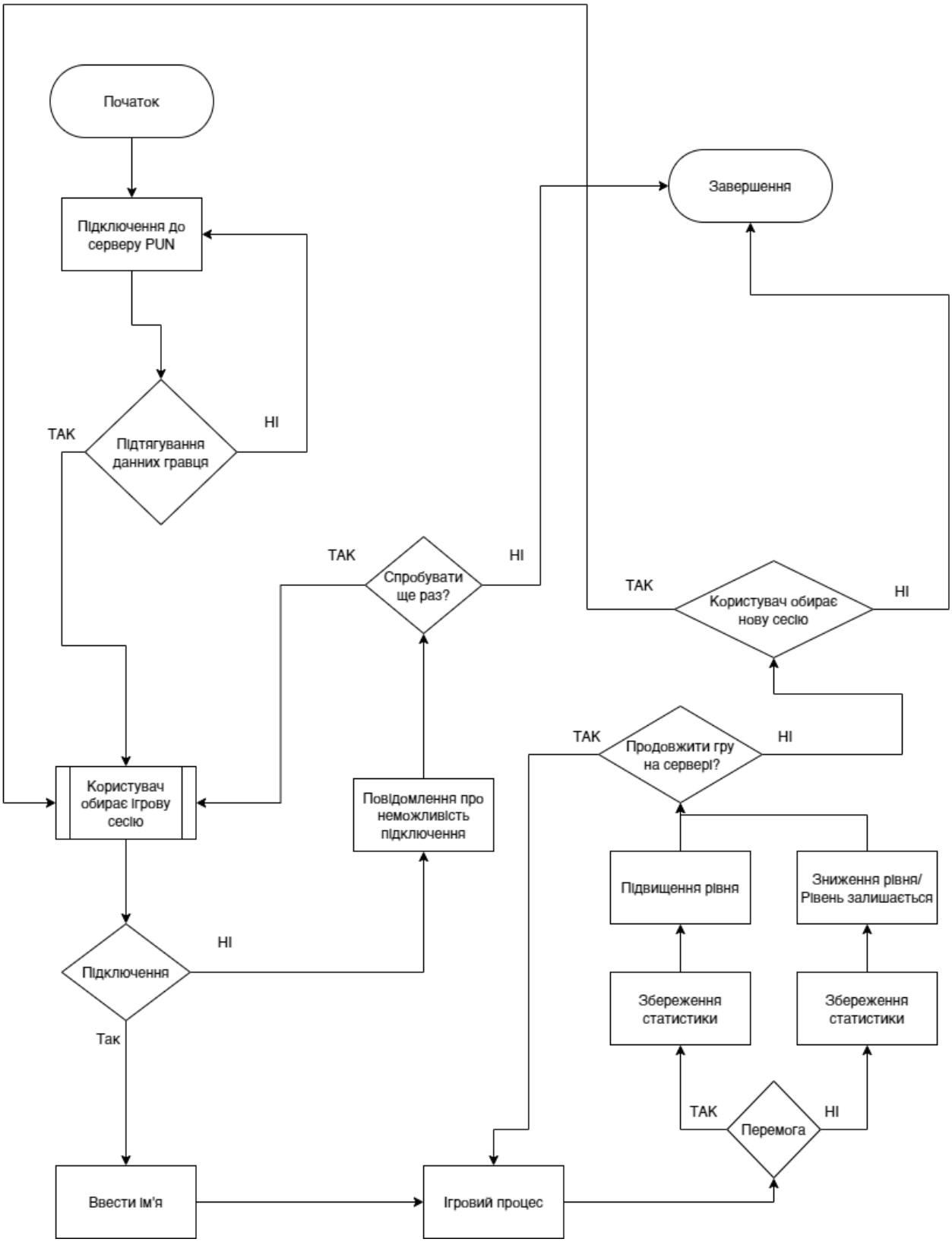

Рисунок 2.17 – Блок-схема алгоритму роботи ігрової системи

2.8 Висновок

У другому розділі розглянуто завдання роботи та окреслені кроки, які необхідно виконати для розробки високоякісного ігрового продукту. Проведено аналіз інформаційного забезпечення ігрового продукту. Розроблену структуру інтерфейсу, який є зручним та задовільняє усі необхідні потреби для широкого кола гравців. Розроблено 3D моделі ігрової локації, зброї та персонажу гравця. Розроблено модель програмного продукту за допомогою UML-діаграми (діаграми прецедентів). Розроблено базові методи, моделі й алгоритми роботи ігрової системи «HeillGunner» в жанрі FPS Shooter.

3 РОЗРОБКА 3D ГРИ З ВИКОРИСТАННЯМ ДВИГУНА UNITY

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного продукту

У рамках даного проєкту фокус ставиться на вирішенні науково-технічних викликів, пов'язаних з розробкою високотехнологічного FPS Shooter. Основним завданням є створення віртуальної ігрової реальності, де кожен кадр, кожен об'єкт та кожен алгоритм мають важливе значення. Використовуючи передовий графічний двигун Unity та Photon SDK для мультиплеєрної інтеграції, робота ґрунтується на наукових принципах, які дозволять створити інтенсивну гру з точністю та динамікою, що вимагають високотехнологічні вимоги [25].

Unity – це комплексне середовище розробки, що використовується для створення високоякісних інтерактивних ігор та додатків. Основна цінність Unity полягає в його здатності взаємодіяти з різноманітними технологіями та обладнанням, забезпечуючи високий рівень реалістичності та іммерсії [26].

Unity володіє потужними засобами для створення реалістичних візуальних ефектів, включаючи відмінну обробку графіки, анімацію та фізичне моделювання. Цей двигун дає можливість науковцям та розробникам розгорнути глибокий рівень взаємодії з ігровими об'єктами та середовищем, що відображається в якості гри та інтеракції [27].

Також важливо відзначити, що Unity включає в себе інструменти для розробки на різних платформах, що робить його ідеальним інструментом для створення багатоплатформових додатків та ігор. Його інтеграція з різними мовами програмування, такими як C#, дозволяє розробникам реалізовувати складні логічні моделі та взаємодію з ігровими об'єктами [28].

Photon SDK відіграє ключову роль у розробці ігрового застосунку, надаючи низку технічних переваг. Цей інструментарій дозволяє створювати динамічний та інтенсивний ігровий досвід, оснований на багатокористувацькій взаємодії гравців у режимі реального часу [29].

Перш за все, використання Photon SDK надає можливість гравцям взаємодіяти та змагатися один з одним, роблячи ігровий світ більш живим та цікавим. Можемо створити спільноту гравців, яка об'єднується для подолання викликів, колективної гри та обміну враженнями. Ця взаємодія є ключовою для досягнення цілей і надає можливість вивчати соціальну динаміку гри в реальному часі.

Друга важлива перевага Photon SDK полягає в тому, що він забезпечує стабільну та швидку мережеву інфраструктуру для гри. Це гарантує, що гравці можуть взаємодіяти без затримок та відчувати плавну інтеракцію. Це є критично важливим для створення реалістичних геймплей-вражень та забезпечення позитивної реакції гравців [30].

Крім того, Photon SDK дозволяє досліджувати та аналізувати взаємодію гравців у віртуальному середовищі. Це створює можливість для наукових досліджень щодо стратегій, тактики та соціальної поведінки гравців, що може бути корисним для подальшого розвитку гри та інтерактивних додатків.

У підсумку, Photon SDK допомагає досягнути поставлених метоцентричних цілей у розробці захоплюючої гри з мультиплеєрними можливостями, роблячи гру більш соціальною, стабільною та динамічною. Цей інструмент є важливим компонентом проєкту, що сприяє досягненню поставлених цілей та створенню незабутнього ігрового досвіду для користувачів.

3.2 Розробка програмного модуля кімнати

Першим етапом при розробці буде створення основного класу, який пов’язуватиме усі інші класи між собою та оригінальні біліотеки Unity та Photon SDK. Цим класом є RoomManager, алгоритм роботи якого наведено на рисунку3.1.

RoomManager – це клас, який відповідає за управління мультиплеєрними кімнатами і гравцями в грі, використовуючи Photon SDK для забезпечення мережевого взаємодії. Детальніше розглянемо його функціональність:

- public static RoomManager instance – статичне поле, що дозволяє отримати доступ до єдиного екземпляру класу RoomManager і використовується для створення єдиного контролера кімнати;
- public GameObject player – посилання на об'єкт гравця, яке використовується для створення нових гравців у грі;
- public Transform[] spawnPoints – масив точок для спавну гравців у грі, гравці будуть з'являтися на випадковій точці з цього масиву;
- public GameObject roomCam – посилання на об'єкт камери для ігрової кімнати, яка контролює, що відображається гравцям у грі;
- public GameObject nameUI – посилання на інтерфейс для введення імені гравця;
- public GameObject connectingUI – посилання на інтерфейс для відображення процесу підключення до ігрової кімнати;
- private string nickname = "unnamed" – змінна, що зберігає ім'я гравця за замовчуванням;
- public string roomNameToJoin = "test" – змінна, що зберігає ім'я ігрової кімнати, до якої гравець намагається приєднатися.

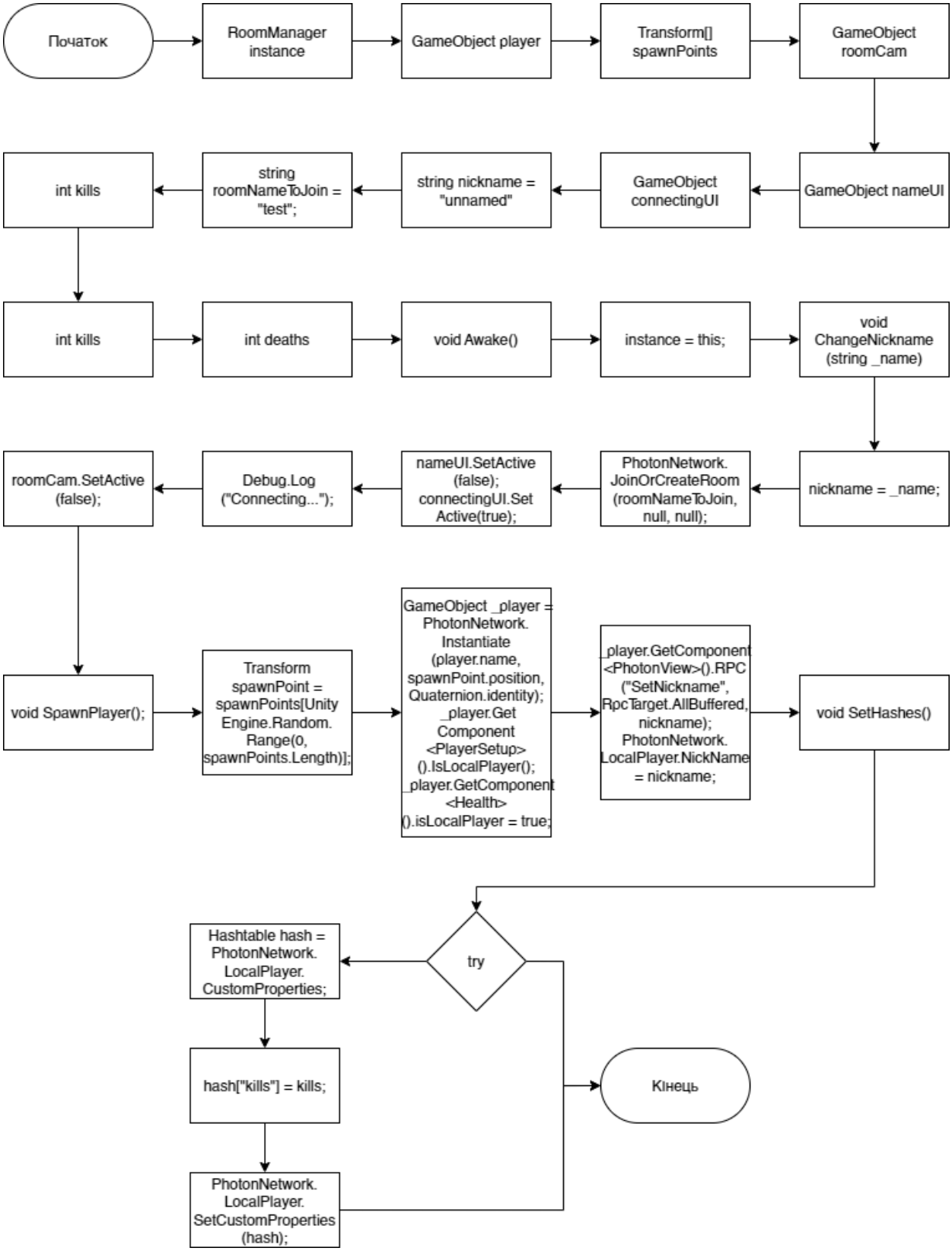

Рисунок 3.1 – Блок-схема алгоритму роботи RoomManager

Розглянемо методи, використані в процесі розробки програмного забезпечення ігрової програми:

- ChangeNickname(string _name) – дозволяє гравцю змінити своє ім'я в грі;
- JoinRoomButtonPressed() – викликається при натисканні кнопки "Приєднатися до кімнати", використовує Photon SDK для спроби приєднання гравця до вказаної кімнати та змінює інтерфейс підключення;
- OnJoinedRoom() – викликається, коли гравець успішно приєднується до ігрової кімнати, приховує об'єкт камери кімнати і викликає SpawnPlayer() для створення гравця;
- SpawnPlayer() – створює гравця на випадковій точці спавну з масиву точок спавну, надає гравцю ім'я та ініціалізує його параметри;
- SetHashes() – встановлює значення кількості вбивств та смертей гравця в користувацьких властивостях гравця у мережі.

Клас RoomManager у контексті розробки гри. Цей клас відповідає за управління різними аспектами взаємодії гравців у межах гри. Він включає поля, такі як статичний екземпляр для отримання доступу до єдиного екземпляру RoomManager, посилання на об'єкти гравців, точки спавну, камери кімнати та інтерфейси користувача для імен гравців і статусу підключення. Крім того, за замовчуванням визначаються значення для імені гравця та імені кімнати для приєднання.

Загалом, клас RoomManager є важливим для управління взаємодією гравців у межах гри, обробки спавну гравців, логіки підключення та мережевих властивостей. Він взаємодіє з Photon SDK для реалізації мультиплеєрної функціональності та надає методи для налаштування та взаємодії гравців.

3.3 Розробка програмного модуля пересування

Базовим класом у грі є Movement, алгоритм роботи якого зображено на рисунку 3.2.

Клас Movement відповідає за реалізацію руху персонажа гравця у 3D-середовищі гри. Основні елементи цього класу включають поля:

- public float walkSpeed = 8f – визначає швидкість руху персонажа гравця під час ходьби, що дозволяє контролювати швидкість персонажа, коли він рухається без бігу.
- public float sprintSpeed = 14f – встановлює швидкість руху персонажа гравця під час бігу, використовується для контролю швидкості персонажа, коли він вирішує прискорити свій рух;
- public float maxVelocityChange = 10f – визначає максимальну швидкість зміни для плавного переміщення персонажа гравця, що дозволяє плавно керувати швидкістю руху персонажа, уникнувши раптових змін швидкості;
- public float airControl = 0.5f – встановлює коефіцієнт керованості персонажа гравця у повітрі, що дозволяє гравцю керувати рухом свого персонажа під час стрибків або перебування у повітрі;
- public float jumpHeight = 5f – визначає висоту стрибка персонажа гравця, що дозволяє контролювати, наскільки високо персонаж може стрибати у грі.

Розглянемо методи, використані при розробці модуля пересування:

- Start() – ініціалізує початкові значення, встановлюючи змінні іншого класу;

- Update() – отримує введення від користувача, включаючи дані про рух уперед/назад та ліворуч/праворуч, а також про біг та стрибок;
- OnTriggerStay(Collider other) – викликається, коли інший колайдер перебуває в тому самому об'єкті тригера, встановлює змінну grounded для визначення того, чи знаходиться персонаж гравця на землі;
- FixedUpdate() – викликається з фіксованою частотою для обробки фізики гри, зокрема для управління рухом персонажа гравця та взаємодії з навколишнім середовищем;
- CalculateMovement(float _speed) – обчислює вектор швидкості для переміщення персонажа гравця залежно від введення користувача та обмежень швидкості, що дозволяє плавно керувати рухом персонажа та уникнути раптових змін швидкості.

Клас Movement має ключову роль у реалізації реалістичного та динамічного руху персонажа гравця в 3D-середовищі гри.

Параметри, визначені у полях класу, створюють можливість гнучко налаштовувати аспекти руху, такі як швидкість ходьби, бігу, максимальна швидкість зміни, а також висота стрибка. Це не лише дозволяє розробникам точно налаштовувати геймплей під конкретні вимоги гри, але і надає гравцеві можливість насолоджуватися різноманітністю рухів у грі.

Цей клас відіграє ключову роль у формуванні основ геймплею, дозволяючи розробникам індивідуалізувати геймплейний досвід.

Його значний внесок у визначенні динаміки гри та точного налаштування рухових параметрів робить цей клас критично важливим компонентом для створення реалістичних ігрових вражень.

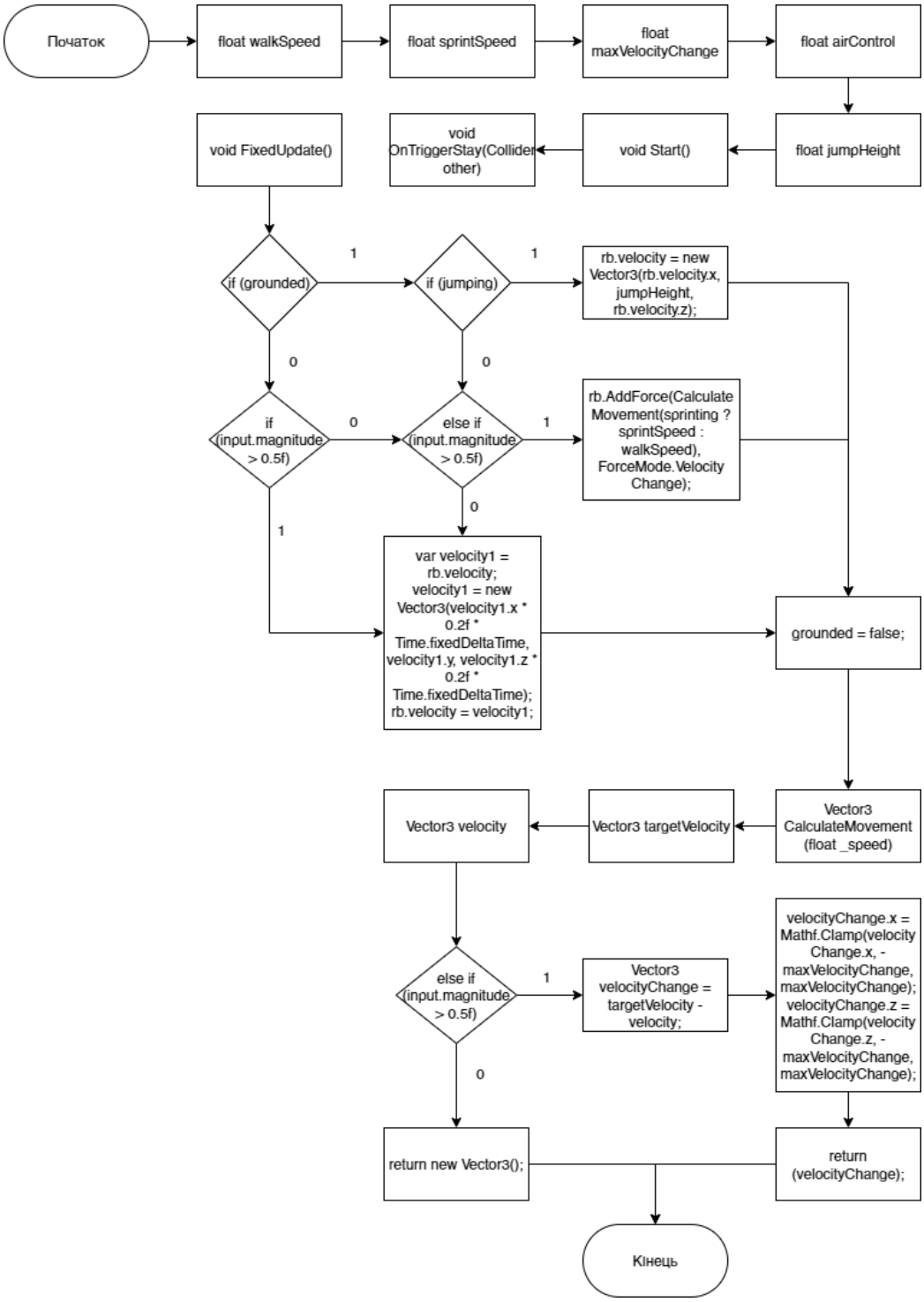

Рисунок 3.2 – Блок-схема алгоритму роботи модуля Movement

Клас Movement служить фундаментом для розширення геймплейних можливостей, наприклад, для використання системи оцінки дій гравця та

терактивних елементів гри. Гнучкість Movement дозволяє ефективно адаптувати рух до різноманітних ігрових сценаріїв, надаючи гравцям унікальні та динамічні ігрові ситуації.

3.4 Розробка програмного модуля гравця

Клас PlayerSetup відповідає за налаштування та управління властивостями гравця в мережевій мультиплеєрній грі.

Розглянемо фрагмент коду класу PlayerSetup (рис. 3.3).

```
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 1
public class PlayerSetup : MonoBehaviour
{
    public Movement movement;

    public GameObject camera;

    public string nickname;

    public TextMeshPro nicknameText;


    Ссылок: 1
    public void IsLocalPlayer()
    {
        movement.enabled = true;
        camera.SetActive(true);
    }

    [PunRPC]
    Ссылок: 0
    public void SetNickname(string _name)
    {
        nickname = _name;

        nicknameText.text = nickname;
    }
}
```

Рисунок 3.3 – Клас PlayerSetup

Розглянемо детальніше Клас PlayerSetup, визначимо його поля та властивості:

- public Movement movement – посилання на об'єкт класу Movement, що відповідає за рух персонажа гравця, використовується для управління рухом персонажа;

- public GameObject camera – посилання на об'єкт камери, пов'язаної з гравцем, яка слідкує за персонажем гравця та відображує його видимість;
- public string nickname – рядок, що містить ім'я гравця в мережевій грі;
- public TextMeshPro nicknameText – посилання на текстовий об'єкт (інтерфейсний елемент) для відображення імені гравця на екрані.

Модуль гравця використовує методи:

- IsLocalPlayer() – публічний метод, який викликається для встановлення локального гравця. У мережевих іграх зазвичай існує різниця між локальними та віддаленими гравцями. Метод активує рух персонажа гравця (movement.enabled = true) та камеру (camera.SetActive(true)), якщо гравець є локальним.
- [PunRPC] public void SetNickname(string _name) – метод, який викликається через Photon RPC (Remote Procedure Call) для встановлення імені гравця в мережі. Встановлює ім'я гравця на основі параметру _name та відображає його в інтерфейсі гравця, використовуючи nicknameText.text = nickname.

Цей клас дозволяє налаштовувати та керувати властивостями гравця в мережевій грі, включаючи ім'я, активацію руху та камери для локального гравця.

3.5 Розробка програмного модуля зброї

Клас Weapon відповідає за реалізацію зброї гравця в грі, включаючи управління вогнем, анімацію, збройовий стан та інші параметри (рис. 3.4).

Розглянемо детальніше поля та властивості класу Weapon:

- public Image ammoCircle – зображення для відображення залишку патронів у магазині;

- public int damage –кількість шкоди, яку завдає зброя при попаданні;
- public Camera camera – посилання на камеру, пов'язану зі зброєю, використовується для визначення напрямку вогню;
- public float fireRate – частота пострілів (кількість пострілів за секунду);
- public GameObject hitVFX – графічний ефект, який відтворюється при попаданні кулі в ціль;
- public int mag = 5 – розмір магазину (кількість патронів у магазині);
- public int ammo = 30 –кількість загальних патронів (поза магазином);
- public int magAmmo = 30 –загальна кількість патронів, яку гравець може мати (включаючи магазин);
- public TextMeshProUGUI magText – текстове поле для відображення кількості патронів у магазині;
- public TextMeshProUGUI ammoText – текстове поле для відображення загальної кількості патронів;
- public Animation animation – компонент анімації, який використовується для відтворення анімації перезарядки зброї;
- public AnimationClip reload – анімація перезарядки зброї;
- public float recoilUp = 1f – значення відведення зброї вгору під час вогню (реакція на віддачу);
- public float recoilBack = 0f – значення відведення зброї назад під час вогню (реакція на віддачу).

Модуль зброї використовує методи:

- SetAmmo() – метод для встановлення відображення в інтерфейсі залишку патронів;
- Start() – ініціалізує початкові значення, встановлює тексти патронів та обчислює час відновлення віддачі;
- Update() – оновлюється кожен кадр, відстежує введення гравця, вогонь, перезарядку та віддачу;
- Reload() – виконує перезарядку зброї, під час перезарядки змінюються тексти патронів та анімація перезарядки;
- Fire() – обробляє вистріл зброєю, створює променевий промінь від камери гравця та визначає, чи влучила куля у ціль, відтворює графічний ефект попадання та завдає шкоду цілі, якщо вона має компонент Health;
- Recoil() – управляє віддачею зброї, піднімаючи її вгору та опускаючи під час стрільби;
- Recovering() – відновлює позицію зброї після віддачі.

Цей клас також може включати в себе параметри, які визначають властивості зброї, такі як потужність пострілу, швидкість стріли та інші характеристики. Це дозволяє гравцеві взаємодіяти з різними видами зброї та відчувати різницю в їхньому використанні.

Клас Weapon також може містити логіку для взаємодії з оточуючими об'єктами, наприклад, перешкодами чи противниками, що може впливати на рух і дії гравця. Керування віддачею та точністю стрільби може бути враховано у логіці цього класу, сприяючи реалістичності гри.

Такий клас може використовувати різноманітні алгоритми для симуляції фізики куль, траєкторій стрільби та інших аспектів, що додає глибину геймплею.

Завдяки класу Weapon гравець може отримати вдосконалений досвід від гри, занурюючись у віртуальний світ з ще більшою інтенсивністю.

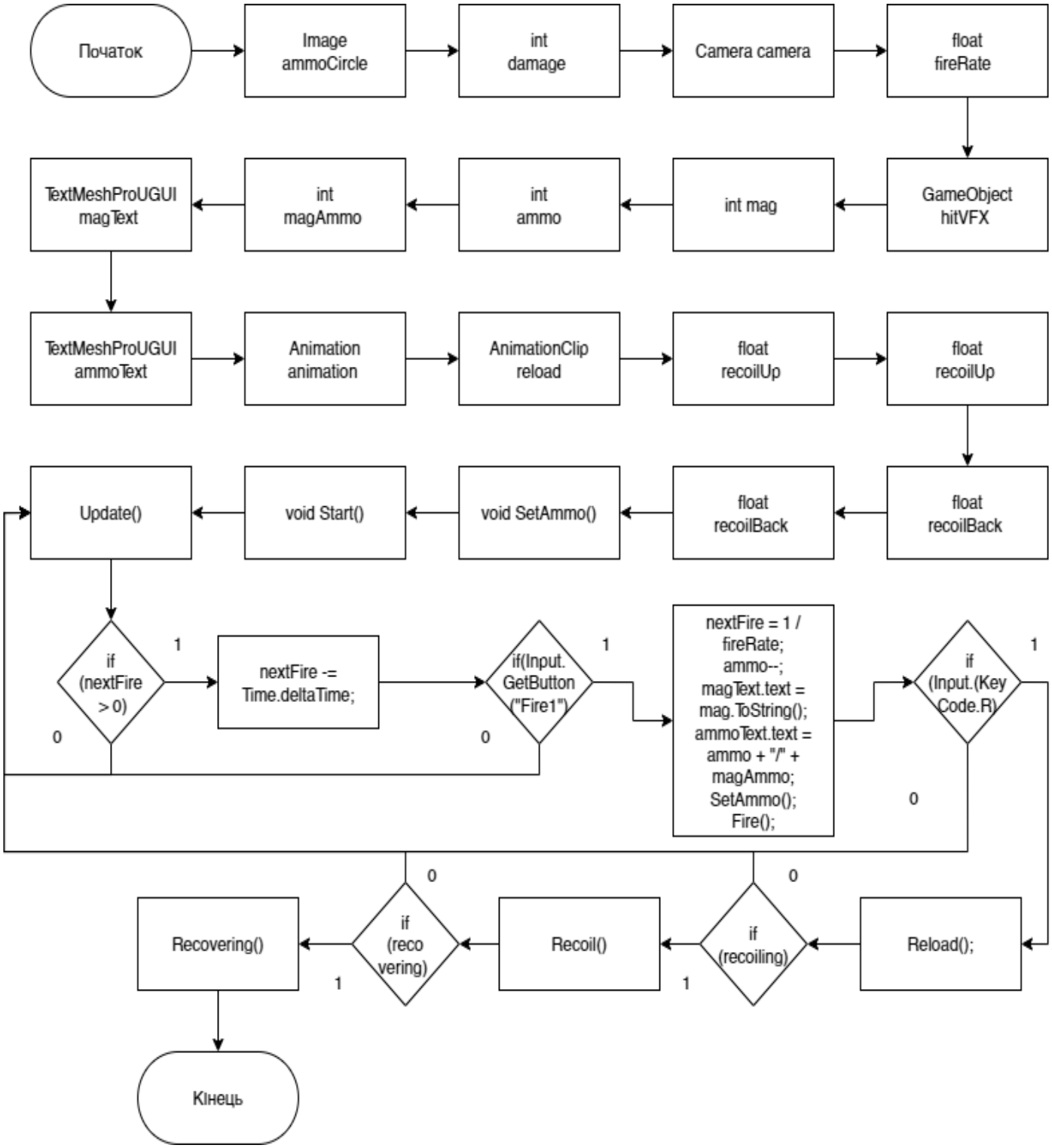

Рисунок 3.4 – Клас Weapon

3.6 Розробка програмного модуля здоров’я персонажа гравця

Клас Health відповідає за управління здоров'ям персонажа гравця в грі, включаючи відображення в інтерфейсі здоров'я та обробку отриманої шкоди.

Розглянемо фрагмент коду класу Health (рис. 3.5).

```
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 3
public class Health : MonoBehaviour
{
    public Image healthCircle;

    public int health;
    public bool isLocalPlayer;

    [Header("UI")]
    public TextMeshProUGUI helthText;

    Ссылок: 1
    void SetHelth()
    {
        healthCircle.fillAmount = (float)health;
    }

    [PunRPC]
    Ссылок: 0
    public void TakeDamage(int _damage)
    {
        health -= _damage;

        helthText.text = health.ToString();

        SetHelth();

        if (health <= 0 )
        {
            if (isLocalPlayer)
            {
                RoomManager.instance.SpawnPlayer();

                RoomManager.instance.deaths++;
                RoomManager.instance.SetHashes();
            }

            Destroy(gameObject);
        }
    }
}
```

Рисунок 3.5 – Клас Health

Розглянемо детальніше поля та властивості класу Health:

- public Image healthCircle – зображення для відображення в інтерфейсі залишку здоров'я;
- public int health – поточний рівень здоров'я персонажа гравця;

- public bool isLocalPlayer – прапорець, який вказує, чи належить персонаж локальному гравцю (цей персонаж контролюється локальним гравцем).

Модуль здоров’я персонажа гравця використовує методи:

- SetHelth() – метод для встановлення відображення в інтерфейсі залишку здоров'я;
- [PunRPC] public void TakeDamage(int _damage) – метод, який викликається через Photon RPC (Remote Procedure Call) для обробки отриманої шкоди. Зменшує кількість здоров'я на _damage, оновлює тексти здоров'я та відображення в інтерфейсі. Якщо здоров'я стає менше або дорівнює 0, то виконує додаткові дії: респаун гравця, збільшує лічильник смертей та оновлює дані гравця.

Клас Health може включати в себе методи для виявлення та обробки подій, пов'язаних зі здоров'ям, таких як отримання ушкоджень від атаки і відновлення здоров'я в результаті використання лікувальних засобів. Додатково він може мати механізми для регулювання рівня здоров'я в залежності від різних факторів, таких як характер ушкоджень або час відновлення. Клас також може взаємодіяти з іншими елементами гри, такими як системи бою чи об'єкти оточення, для створення більш комплексного геймплею та стратегічних можливостей.

Важливою функцією класу Health є відображення інформації про здоров'я персонажа гравця в графічному інтерфейсі, що надає гравцеві важливий контекст щодо його стану. Крім того, взаємодія з подіями втрати здоров'я та можливість обробки ситуацій смерті додають реалізму та викликів у гру, підсилюючи враження від мережевого геймплею. Такий клас стає важливим компонентом для створення захоплюючого та динамічного ігрового досвіду в мережевих іграх. Додатково, можливість синхронізації стану здоров'я між різними персонажами гравців у

реальному часі робить гру ще більш іммерсивною та реалістичною для усіх учасників.

3.7 Розробка програмного модуля ігрової кімнати

Розглянемо RoomList, блок-схема алгоритму роботи якого зображена на рисунках 3.6 та 3.7.

Клас RoomList відповідає за управління списком кімнат у мережевій грі та вибір ігрової кімнати для приєднання.

Розглянемо детальніше поля та властивості класу RoomList:

- public static RoomList Instance – посилання на єдиний екземпляр класу RoomList, використовується для доступу до цього класу з інших частин коду;
- public GameObject roomManagerGameobject – посилання на об'єкт, який належить класу RoomManager, використовується для активації об'єкта RoomManager при приєднанні до кімнати;
- public RoomManager roomManager – посилання на об'єкт класу RoomManager, який відповідає за управління кімнатами та гравцями в них;
- public Transform roomListParent – посилання на батьківський об'єкт для відображення списку кімнат в інтерфейсі;
- public GameObject roomListItemPrefab – префаб (зразок) для створення об'єктів-елементів списку кімнати в інтерфейсі;
- private List<RoomInfo> cachedRoomList = new List<RoomInfo>() – список кешованої інформації про кімнати для подальшого відображення у

списку.

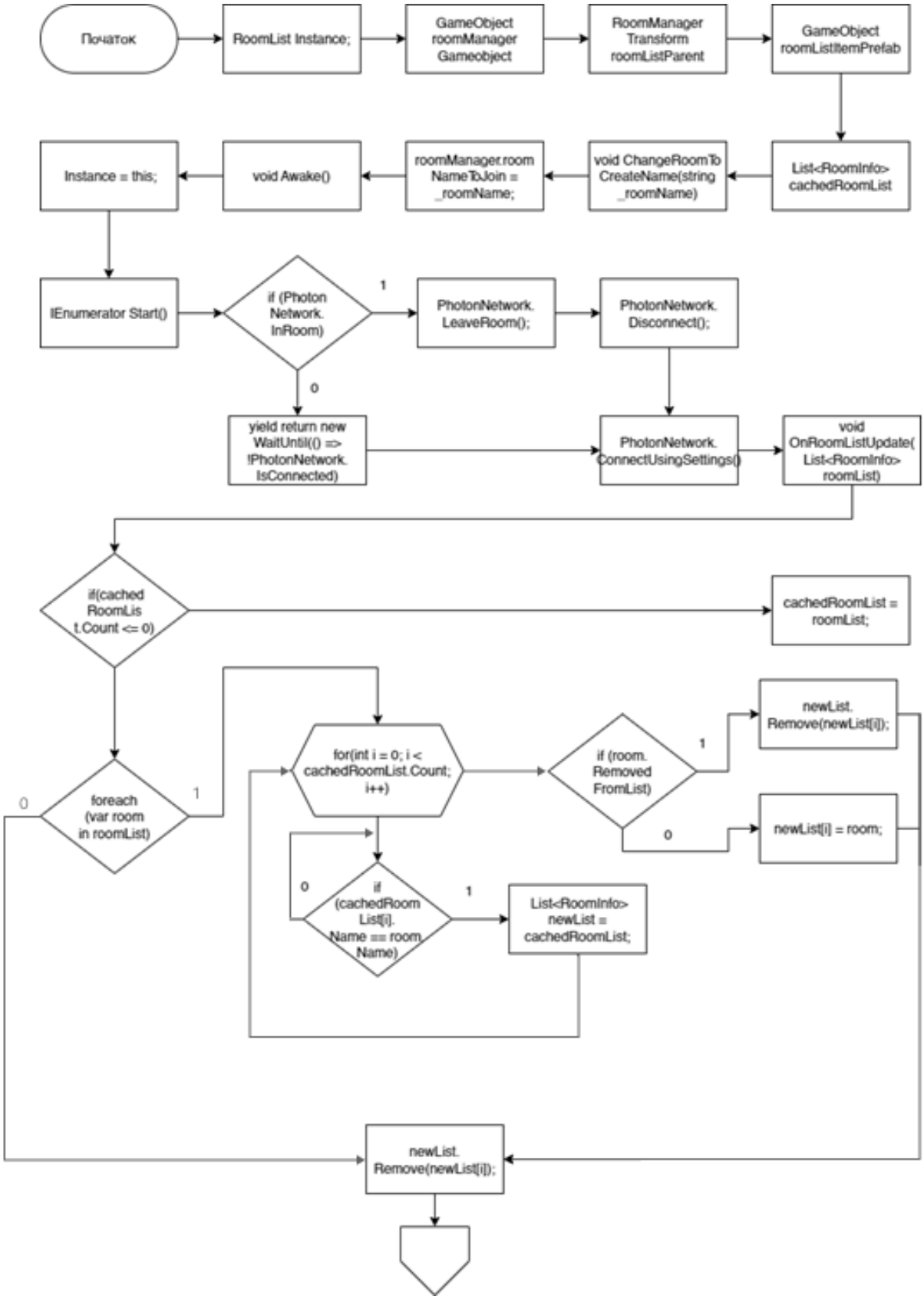

Рисунок 3.6 – Блок-схема алгоритму роботи модуля RoomList (частина 1)

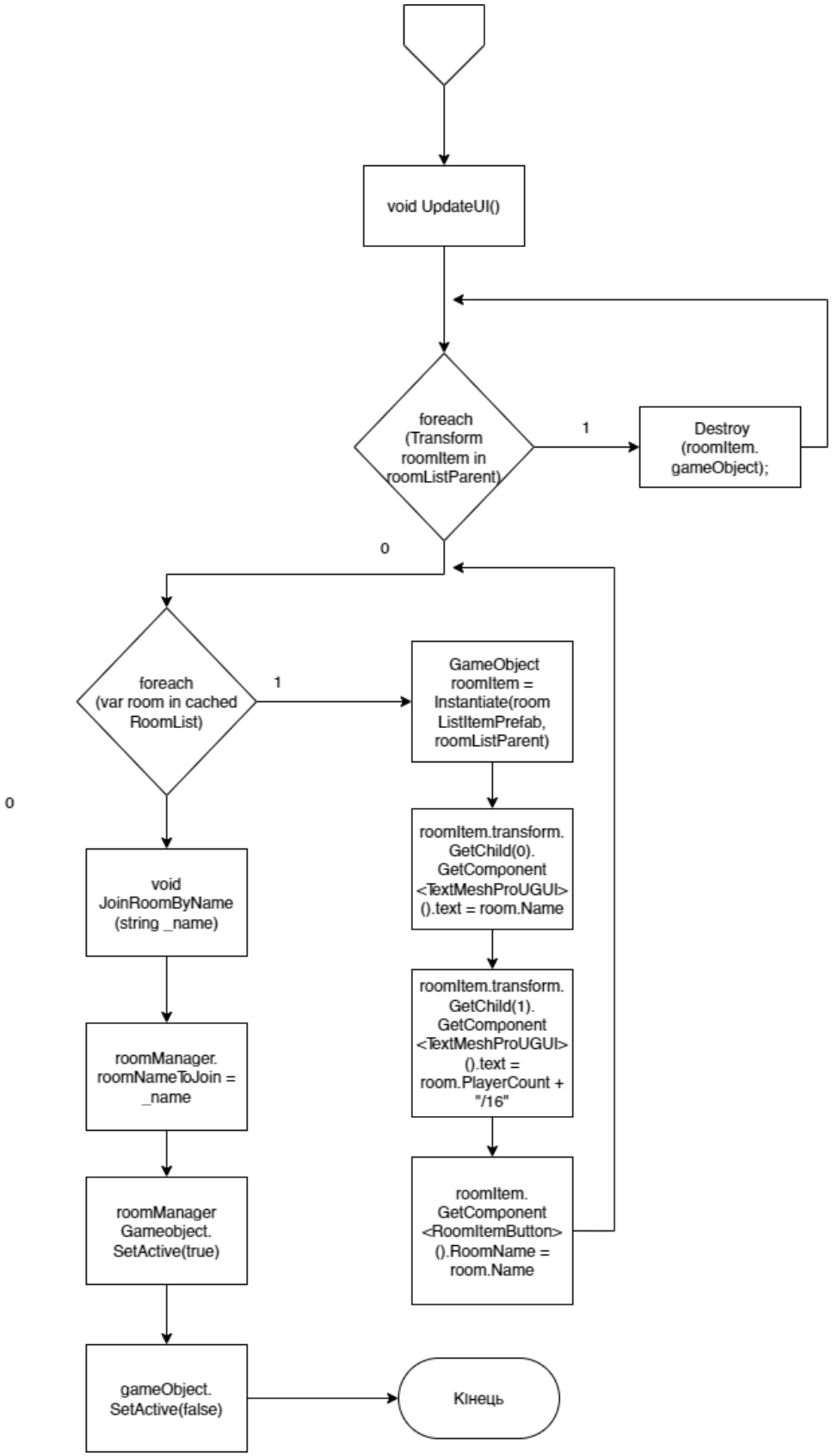

Рисунок 3.7 – Блок-схема алгоритму роботи модуля RoomList (частина 2)

Модуль ігрової програми використовує методи:

- ChangeRoomToCreateName(string _roomName) – метод для зміни імені кімнати, яку гравець хоче створити або приєднатися до неї;
- Awake() – метод, що викликається на старті гри і встановлює посилання на єдиний екземпляр класу RoomList;
- Start() – метод, що викликається при запуску гри, виконує підготовчі дії, такі як розрив з'єднання з попередньою кімнатою та підключення до мережі Photon;
- OnConnectedToMaster() – перевизначений метод, що викликається при підключенні до сервера Photon, він викликає JoinLobby() для приєднання до лобі;
- OnRoomListUpdate(List<RoomInfo> roomList) – перевизначений метод, що викликається при оновленні списку кімнат, оновлює кешований список кімнат і викликає метод UpdateUI() для оновлення інтерфейсу;
- UpdateUI() – очищує інтерфейс від попередніх елементів та створює нові елементи списку кімнат на основі кешованого списку;
- JoinRoomByName(string _name) – метод для приєднання до обраної кімнати за ім'ям, встановлює ім'я кімнати в об'єкті RoomManager, активує roomManagerGameobject та деактивує поточний об'єкт.

Клас RoomManager виступає важливим компонентом для ефективного управління мережевими аспектами гри, зокрема в контексті вибору та управління кімнатами гравців. Його основна функція полягає у відстеженні стану кімнат та координації взаємодії між гравцями, які перебувають у різних ігрових приміщеннях.

Клас RoomManager може включати методи для створення нових ігрових кімнат, видалення існуючих та керування загальним потоком гравців між ними. Цей клас може також забезпечувати механізми для синхронізації даних між гравцями у реальному часі, забезпечуючи стабільну і консистентну гру.

3.8 Розробка програмного модуля анімації розхитування

Розглянемо клас Sway (рис. 3.8). Він реалізує ефект розхитування зброї гравця під час руху миші, щоб зробити геймплей більш іммерсивним та реалістичним.

```
Скрипт Unity (ссылок на ресурсы: 2) | Ссылок: 0
public class Sway : MonoBehaviour
{
    [Header("Settings")]
    public float swayClamp = 0.09f;
    [Space]
    public float smoothing = 3f;


    private Vector3 origin;

    // Start is called before the first frame update
    Сообщение Unity | Ссылок: 0
    void Start()
    {
        origin = transform.localPosition;
    }

    // Update is called once per frame
    Сообщение Unity | Ссылок: 0
    void Update()
    {
        Vector2 input = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse Y"));

        input.x = Mathf.Clamp(input.x, -swayClamp, swayClamp);
        input.y = Mathf.Clamp(input.y, -swayClamp, swayClamp);

        Vector3 target = new Vector3(-input.x, -input.y, 0);

        transform.localPosition = Vector3.Lerp(transform.localPosition, target + origin, Time.deltaTime * smoothing)
    }
}
```

Рисунок 3.8 – Клас Sway

Розглянемо детальніше поля та властивості класу Sway:

- public float swayClamp = 0.09f – це значення визначає максимальну

амплітуду розхитування зброї в горизонтальному або вертикальному напрямку, воно обмежує, наскільки далеко може розходитися позиція зброї під час руху миші;

- public float smoothing = 3f – цей параметр відповідає за згладжування ефекту розхитування, він визначає, наскільки плавно зброя повертається до свого початкового положення після руху миші.

Модуль анімації розхитування використовує методи:

- Start() – цей метод викликається на старті гри. Він встановлює початкове положення зброї, яке використовується як точка відліку для розхитування.
- Update() – цей метод викликається кожен кадр. Він отримує значення руху миші в горизонтальному і вертикальному напрямках (Input.GetAxisRaw("Mouse X") і Input.GetAxisRaw("Mouse Y")). Після цього значення обмежуються за допомогою Mathf.Clamp так, щоб вони не перевищували swayClamp.

Далі визначається цільове положення зброї (target) на основі значень руху миші. Потім застосовується плавний перехід від поточного положення зброї до цільового положення за допомогою Vector3.Lerp. Це створює візуальний ефект розхитування зброї під час руху миші.

3.9 Розробка програмного модуля зміни гравцем зброї

Розглянемо клас WeaponSwitcher (рис. 3.9). Він відповідає за вибір та перемикання між різною зброєю гравця.

Розглянемо детальніше його поля та властивості:

- public Animation _animation – посилання на компонент анімації, Ввикористовується для відтворення анімації при зміні зброї;

- public AnimationClip draw – анімація "витягування" зброї, використовується для анімаційного ефекту при виборі нової зброї;
- private int selectedWeapon = 0 – індекс обраної зброї в масиві дочірніх об'єктів.

Модуль зміни гравцем зброї використовує методи:

- Start() – метод, який викликається на старті гри, викликає метод SelectWeapon(), щоб встановити початкову зброю гравця;
- Update() – метод, який викликається кожен кадр, відстежує дії гравця, такі як вибір зброї за допомогою клавіш 1-9 або колеса миші;
- SelectWeapon() – метод для вибору та перемикання між зброєю гравця, він приймає поточний індекс обраної зброї та встановлює видимість зброї відповідно до обраної зброї.

У методі Update() реалізована логіка вибору зброї за допомогою клавіш 1-9 або колеса миші. Також він слідкує за зміною обраної зброї та викликає SelectWeapon(), якщо обрана зброя змінюється.

У методі SelectWeapon() перебираються всі дочірні об'єкти (зброї) і встановлюється видимість обраної зброї, а решта зброї стає невидимою.

Також використовується анімація "витягування" зброї за допомогою _animation.Play(draw.name).

Цей клас дозволяє гравцю вибирати різні зброї та ефектно перемикатися між ними з використанням анімаційного ефекту "витягування" зброї.

```
void SelectWeapon()
{
    if(selectedWeapon >= transform.childCount)
    {
        selectedWeapon = transform.childCount - 1;
    }

    _animation.Stop();
    _animation.Play(draw.name);


    int i = 0;

    foreach(Transform _wepon in transform)
    {
        if(i == selectedWeapon)
        {
            _wepon.gameObject.SetActive(true);
        }
        else
        {
            _wepon.gameObject.SetActive(false);
        }

        i++;
    }
```

Рисунок 3.9 – Клас WeaponSwitcher

3.10 Розробка програмного модуля зчитування статистики гравців

Клас Leaderboard відповідає за оновлення та відображення лідерів у грі (рис.3.10).

Розглянемо детальніше поля та властивості класу Leaderboard:

- public GameObject playersHolder – посилання на контейнер для гравців у грі;

- public float refreshRate = 1f – частота оновлення лідерів;
- public GameObject[] slots – масив для відображення слотів лідерів;
- public TextMeshProUGUI[] scoreTexts – масив текстових полів для відображення результатів гравців;
- public TextMeshProUGUI[] nameTexts – масив текстових полів для відображення імен гравців;
- public TextMeshProUGUI[] kdTexts – масив текстових полів для відображення співвідношення вбивств та смертей персонажів гравців.

Модуль зчитування статистики гравців використовує методи:

- Start() – метод, який викликається на старті гри. Використовує InvokeRepeating для регулярного оновлення інформації про лідерів.
- Refresh() – метод для оновлення інформації про лідерів. Всі слоти лідерів спочатку стають невидимими.

  У Refresh() гравці сортуються за кількістю очок у порядку спадання отриманих результатів, інформація про них заповнюється у відповідні поля. Якщо ім'я гравця відсутнє, встановлюється значення "unnamed".

  Також у Refresh() відображається співвідношення вбивств до смертей, якщо воно доступне, або значення "0/0" у протилежному випадку.

```
public class Leaderboard : MonoBehaviour
{
    public GameObject playersHolder;

    [Header("Options")]
    public float refreshRate = 1f;

    [Header("UI")]
    public GameObject[] slots;

    [Space]
    public TextMeshProUGUI[] scoreTexts;
    public TextMeshProUGUI[] nameTexts;
    public TextMeshProUGUI[] kdTexts;


    Сообщение Unity | Ссылок: 0
    private void Start()
    {
        InvokeRepeating(nameof(Refresh), 1f, refreshRate);
    }


    Ссылок: 1
    public void Refresh()
    {
        foreach (var slot in slots)
        {
            slot.SetActive(false);
        }


        var sortedPlayerList =
            (from player in PhotonNetwork.PlayerList orderby player.GetScore() descending select player).ToList();

        int i = 0;
        foreach (var player in sortedPlayerList)
        {
            slots[i].SetActive(true);

            if(player.NickName == "")
               player.NickName = "unnamed";


            nameTexts[i].text = player.NickName;
            scoreTexts[i].text = player.GetScore().ToString();


            if (player.CustomProperties["kills"] != null)
            {
                kdTexts[i].text = player.CustomProperties["kills"] + "/" + player.CustomProperties["deaths"];
            }
            else
            {
                kdTexts[i].text = "0/0";
            }


            i++;
        }
    }
```

Рисунок 3.10 – Клас Leaderboard

Клас Leaderboard використовується для створення і поновлення лідерської дошки у грі, що дозволяє гравцям бачити та порівнювати свої результати з результатами інших учасників гри в реальному часі.

3.11 Висновки

У третьому розділі було обґрунтовано використання двигуна Unity та технології Photon SDK для розробки багатокористувацької гри у жанрі «FPS Shooter».

Було проведено розробку усіх компонентів та модулів ігрової системи «HeillGunner» з використанням класів: RoomManager, Movement, PlayerSetup, Weapon, Health, RoomList, Sway, WeaponSwitcher, Leaderboard.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Аналіз методів тестування

У розробці програмного продукту тестування є критично важливим етапом, оскільки воно спрямоване на забезпечення високої якості та надійності роботи програми. Наявні різні методики та види тестування, які мають свою специфіку та цільовий напрям.

Серед найбільш поширених видів тестування можна виділити такі:

- чорна скринька (Black Box Testing) – у цьому методі програма розглядається як "чорна скринька", тобто тестувальник не має доступу до внутрішньої структури програми і зосереджується на перевірці її функціональності та взаємодії з користувачем;
- біла скринька (White Box Testing) – уцьому випадку тестувальник має доступ до вихідного коду програми та використовує його для створення тестів, тестування базується на аналізі внутрішньої логіки програми;
- сіра скринька (Grey Box Testing) – цей метод поєднує аспекти чорної та білої скриньки, де тестувальник має обмежений доступ до внутрішньої структури програми.

Крім того, існують різні види тестування, які спрямовані на оцінку різних аспектів програми:

- функціональне тестування (Functional Testing) – перевірка функціональності програми з точки зору користувача;
- тестування на відповідність до вимог (Requirements Testing) – перевірка відповідності програми до встановлених вимог;

- тестування продуктивності (Performance Testing) – оцінка швидкості та продуктивності програми під навантаженням;
- тестування на відмови (Failure Testing) – перевірка поведінки програми під час відмови чи помилок;
- тестування на безпеку (Security Testing) – оцінка програми на вразливість до атак та забезпечення захисту від них;
- тестування на відновлення (Recovery Testing) – перевірка здатності програми відновлювати роботу після відмови чи збою;
- тестування на сумісність (Compatibility Testing) – оцінка сумісності програми на різних платформах, операційних системах та браузерах;
- тестування на витривалість (Endurance Testing) – оцінка роботи програми при тривалому використанні або при великих навантаженнях;
- тестування на відновлення даних (Data Recovery Testing) – перевірка можливості відновлення даних після аварійного завершення програми;
- тестування на гнучкість (Usability Testing) – оцінка зручності та користувацької дружності програми;
- тестування на помилки (Error Testing) – перевірка програми на виявлення помилок та їх обробку;
- тестування на стабільність (Stability Testing) – оцінка стабільності та надійності програми.

Для тестування розробленої комп’ютерної гри «HeillGunner» у жанрі FPS Shooter обираємо метод тестування «Чорна скринька».

4.2 Тестування компонентів системи

Для тестування гри було використано: комп’ютер з операційною системою Windows 11, CPU: Intel Core I5 6700k, CPU:16гб, GPU:GTX 1060 6гб.

Тестування роботи системи було здійснено у вигляді покрокового виконання з врахуванням основного робочого процесу для кінцевих користувачів.

Порядок проведення тестування ігрової системи:

1) Користувач відкриває додаток HeillGunner.exe (рисунок 4.1).

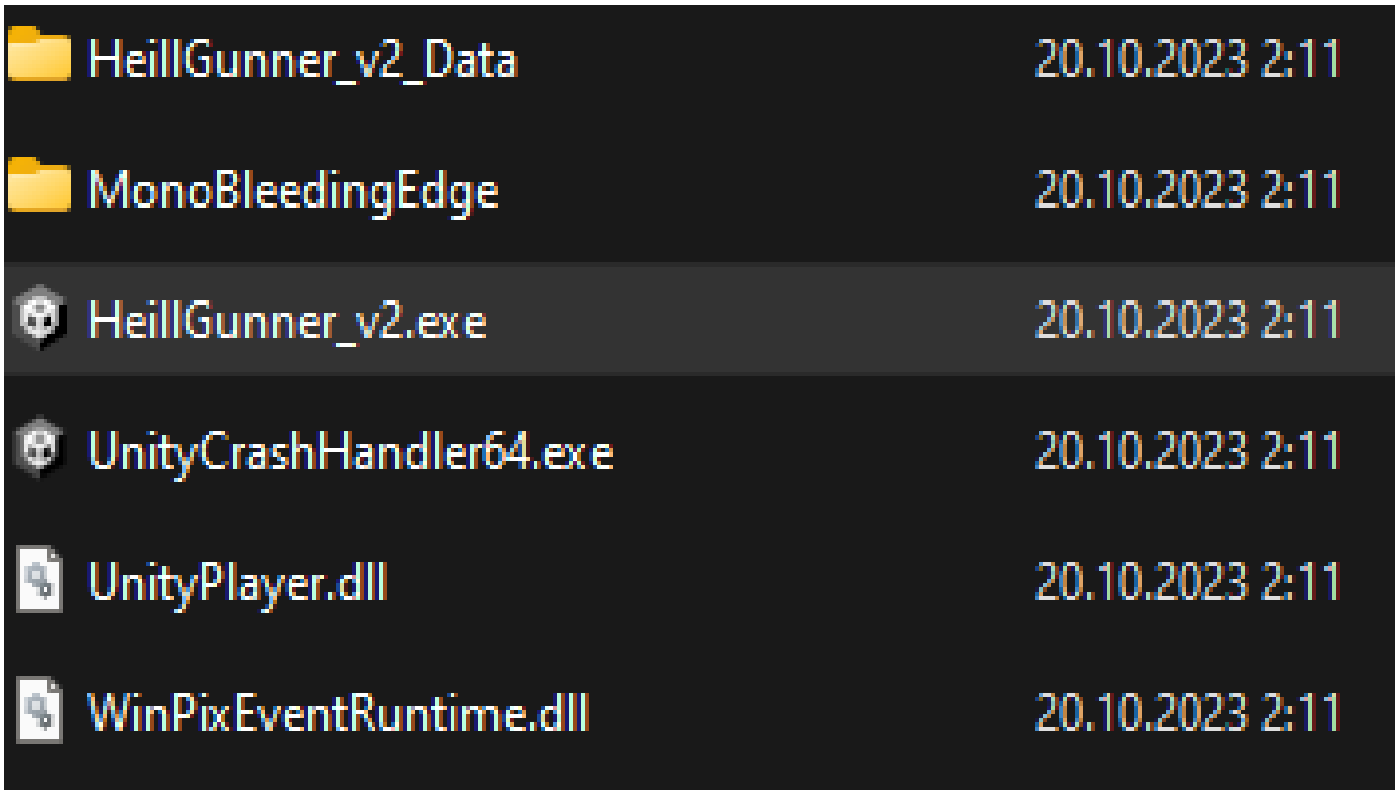

Рисунок 4.1 – Файл гри HeillGunner.exe

2) Коли користувач відкриває додаток, він переходить до меню, де йому пропонується або створити власний сервер, або доєднатися до існуючого (рис. 4.2).

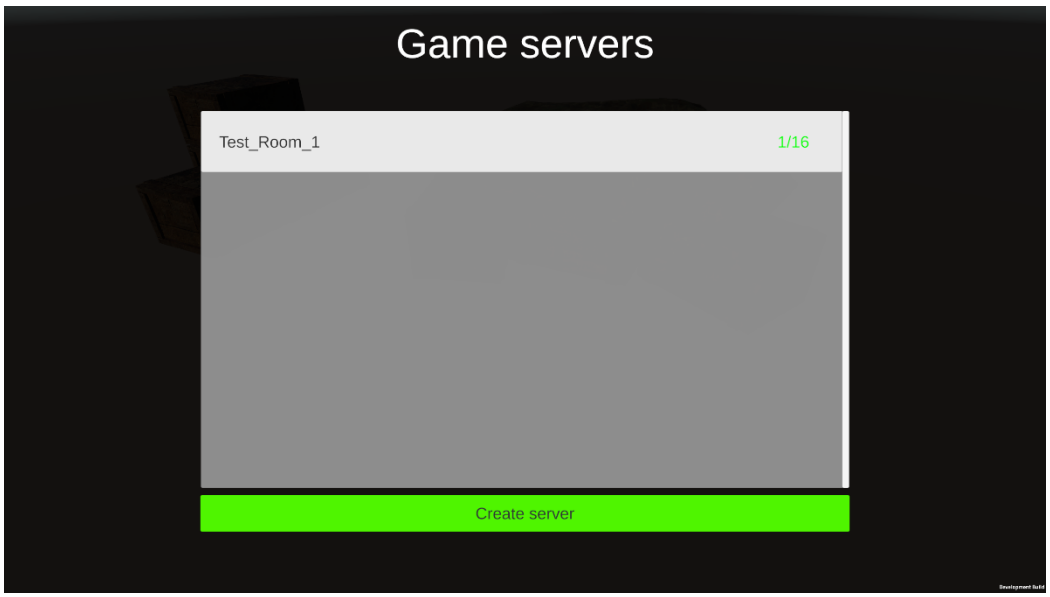

Рисунок 4.2 – Меню із списком серверів

3) Користувач натискає на сервер «Test_Room_1», після чого йому необхідно ввести своє ім’я (якщо користувач не вводить ім’я, за замовчуванням йому надається ім’я «Unknown») (рис. 4.3).

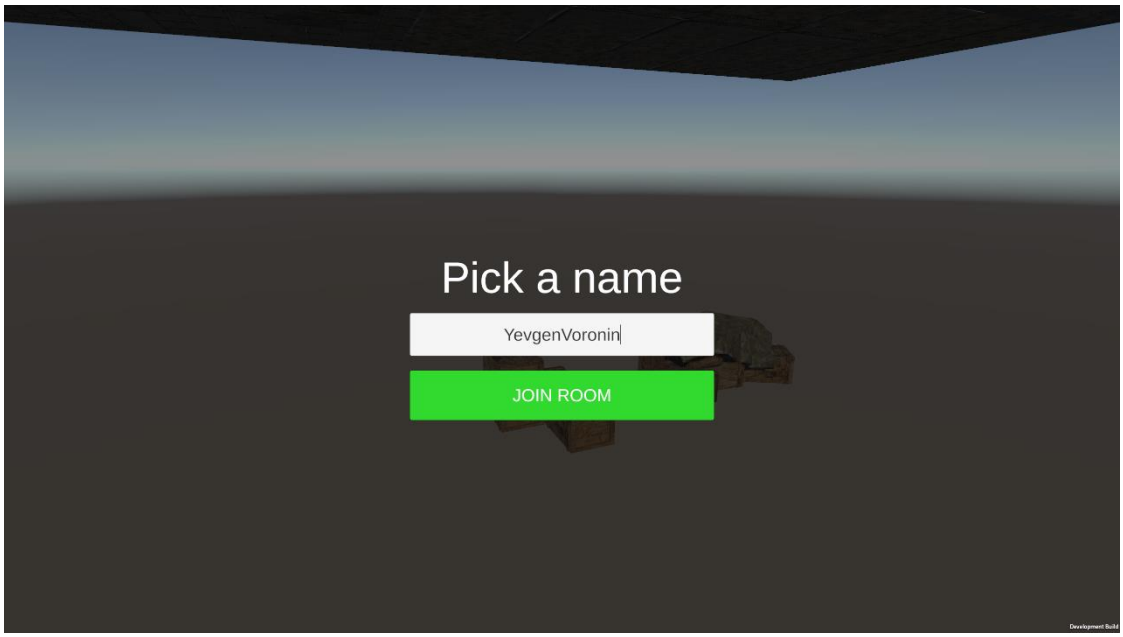

Рисунок 4.3 – Вікно для введення імені гравця

4) Користувач натискає кнопку «Join Room» та доєднується до ігрової сесії (рис. 4.4).

Рисунок 4.4 – Ігрова сесія

5) Перевіряємо рухи вперед, назад, право, ліво та коректність повороту персонажа мишкою.

6) Просимо ігрового партнера вистрілити в наш персонаж, щоб перевірити коректність реєстрації попадань на HelthBar (рис. 4.5).

Рисунок 4.5 – HelthBar

7) Стріляємо в персонаж суперника 4 рази та перевіряємо LeaderBoard на коректність фіксації та підрахунку результатів, натиснувши клавішу «TAB» (рис.4.6).

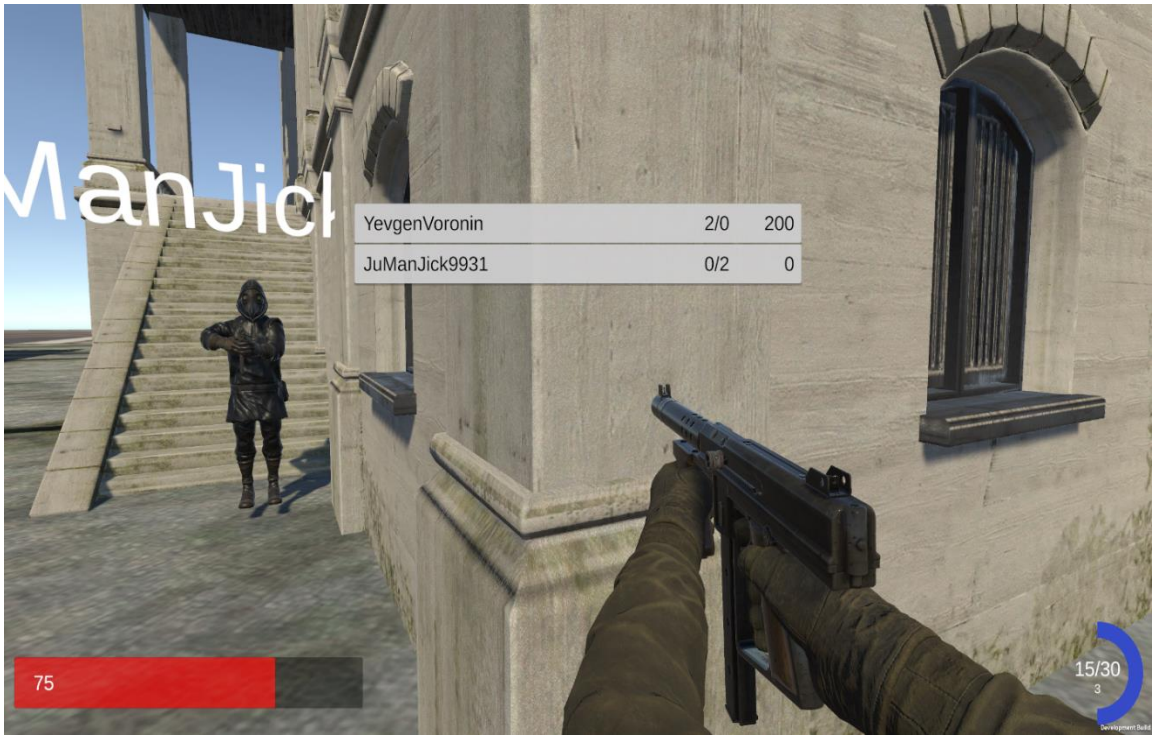

Рисунок 4.6 – LeaderBoard

8) Перевіряємо коректність підрахунку боєприпасів та магазинів, які має

гравець: було (ліворуч) та стало (праворуч) (рисунок 4.7);

Рисунок 4.7 – Інтерфейс підрахунку боєприпасів

9) Від’єднуємося від сервера.

10) Перевіряємо можливість створення власного сервера (рис. 4.8 та 4.9).

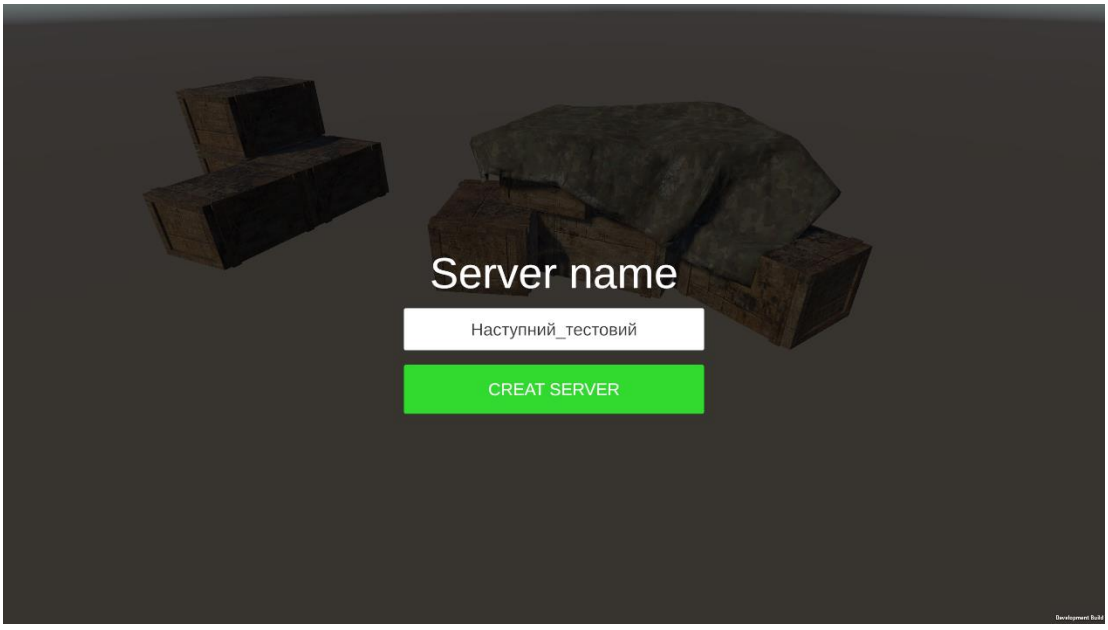

Рисунок 4.8 – Вікно створення власного сервера

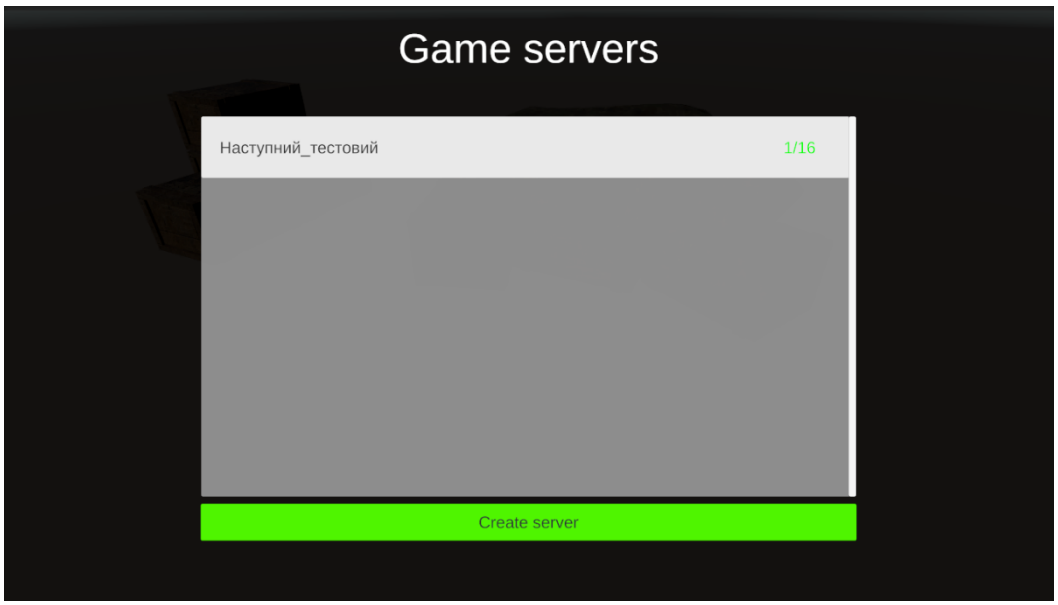

Рисунок 4.9 – Наш сервер на екрані іншого гравця

11) Перевіряємо колізію стін (рис. 4.10).

Рисунок 4.10 – Перевірка стикання моделі гравця з об'єктом будинку

12) Перевіряємо зсередини будівлі (рис. 4.11).

Рисунок 4.11 – Перевірка колізії зсередини будівлі

13) Перевіряємо колізію сходів (рис. 4.12).

Рисунок 4.12 – Перевірка колізії сходів

14) Перевіряємо колізію між гравцями (рис. 4.13 та 4.14).

Рисунок 4.13 – Колізія від особи 1-го гравця

Рисунок 4.14 – Колізія від особи 2-го гравця

15) Перевіряємо завантаженість сервера з більшою кількістю гравців (рис.4.15, 4.16, 4.17 та 4.18).

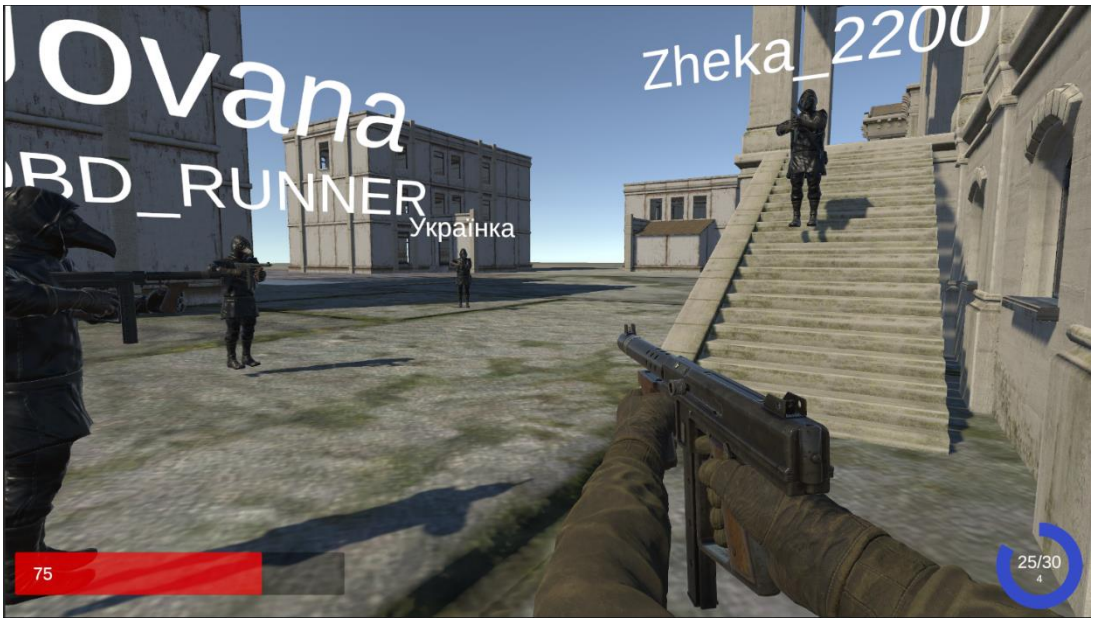

Рисунок 4.16 – Коли всі гравці в одному місці

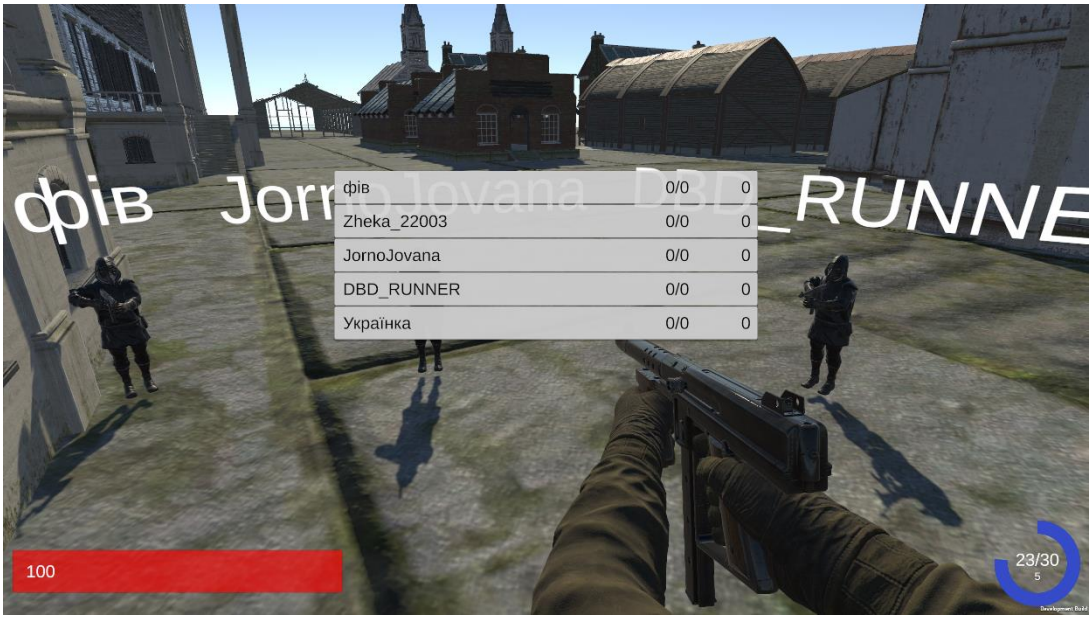

Рисунок 4.17 – Перевірка коректності відображення усіх гравців у грі

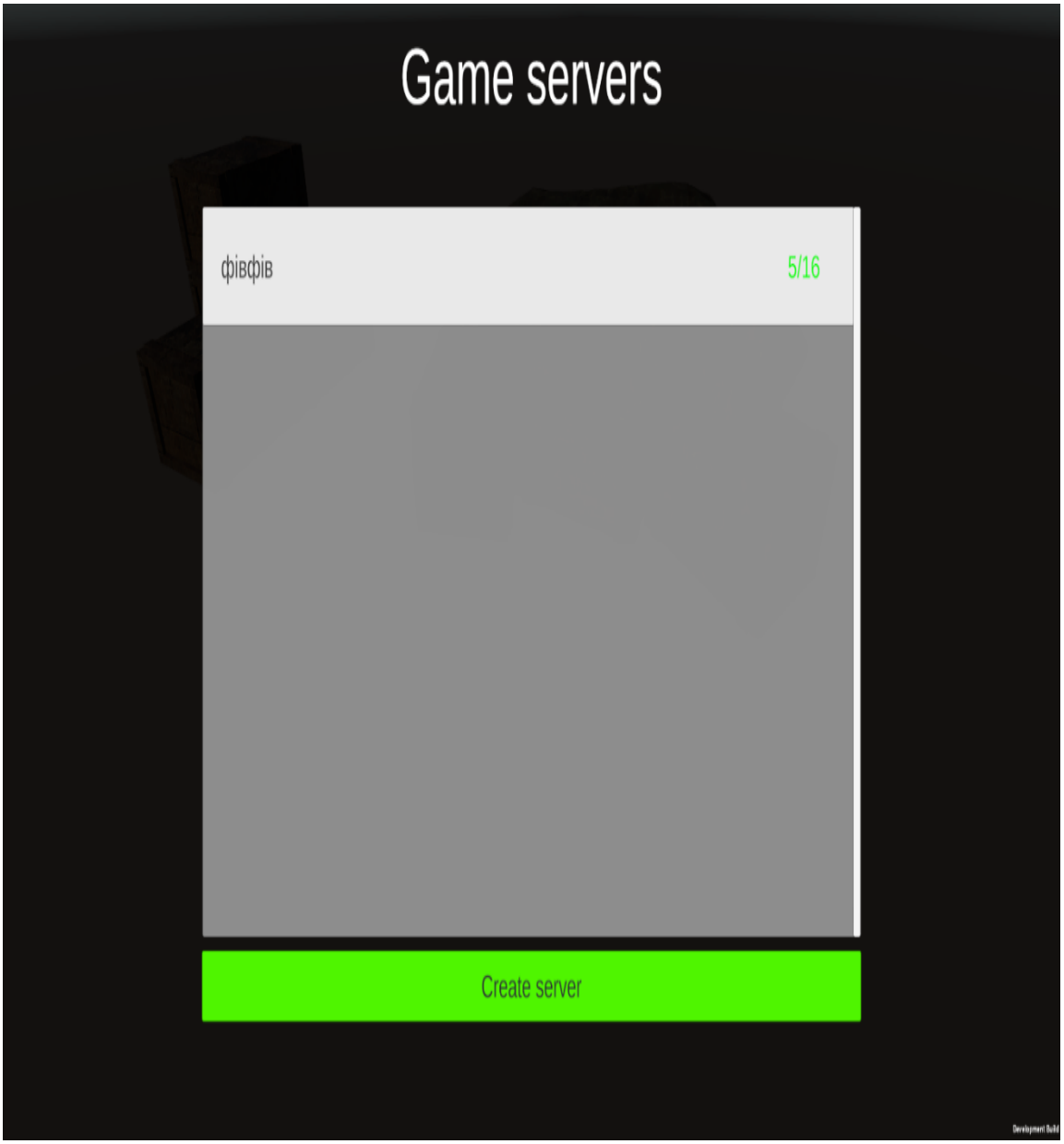

Рисунок 4.18 – Перевірка коректності відображення усіх гравців у кімнаті

16) Перевіряємо коректність роботи методу автоматичного підбору гравців за допомогою веб-інтерфейсу, який отримали завдяки Photon SDK. Ліворуч показується графік завантаженості сервера та кількість гравців в онлайні, з праворуч показано статистику усіх гравців з коефіцієнтом їх продуктивності (рис.4.19).

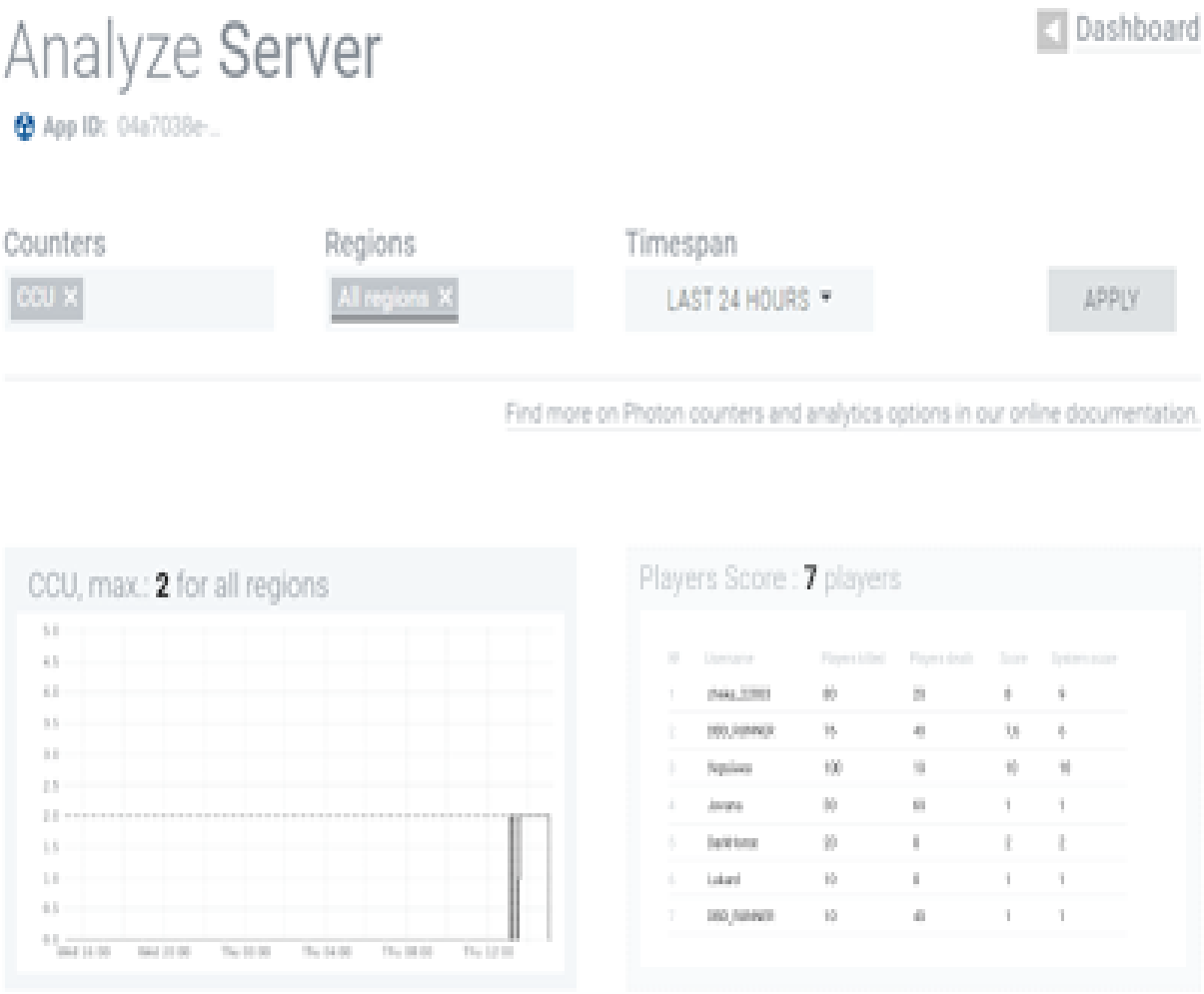

Рисунок 4.19 – Адміністративна панель Photon SDK

Рисунок 4.19 наводить статистику успіхів гравців та визначає можливості підбору гравців на наступну ігрову сесію.

4.3 Розробка інструкції користувача

Інструкція користувача складається з метою спрощення та прискорення процесу розгортання програмного забезпечення або комплексних програмних систем у середовищі кінцевого користувача, шляхом надання докладних та структурованих вказівок.

У таблиці 4.1 наведено технічні вимоги для розгортання розробленої системи.

Таблиця 4.1 – Технічні вимоги

Процесор	32-розрядний (x86) або 64-розрядний (x64) або ARM процесор з тактовою частотою 1 ГГц
Об’єм оперативної пам’яті	8 GB
Вільне місце на жорсткому диску	150 MB для базової версії настільного додатку
Операційна система	Windows 10, 11

Процес інсталяції настільного додатку відбувається за допомогою програми-розархівування файлу ZIP програмою WinRar .

Після завершення розархівування додаток готовий до використання.

4.4 Висновки

У четвертому розділі було проведено аналіз підходів до тестування програмного забезпечення та обрано метод тестування "чорної скриньки" як найбільш підходящий для випробування комплексної системи. Це обґрунтовується тим, що обраний метод дозволяє провести тестування в умовах, які найбільше наближені до реальних умов використання ігрової системи.

Після вибору методу тестування було проведено комплексне випробування системи, результати якого підтвердили її функціональність і відповідність до вимог.

Крім того, були визначені технічні характеристики для розгортання системи та розроблена інструкція для користувача щодо встановлення компонентів системи.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Розробка методів і засобів 3D-гри з використанням двигуна Unity» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка методів і засобів 3D-гри з використанням двигуна Unity» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки,

створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [31].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1.

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці 5.2.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	4	4	4
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	3	3
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	3	3
Сума балів	40	40	40
Середньоарифметична сума балів $СБ_c$	40		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [31].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за

темою «Розробка методів і засобів 3D-гри з використанням двигуна Unity» становить 40 балів, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вищий середнього).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка методів і засобів 3D-гри з використанням двигуна Unity», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою 5.1:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=21 день.

$$З_о = 25000{,}00 \cdot 63 / 21 = 75000 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 5.4.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	25000,00	1190,48	63,00	75000,00
Інженер-розробник програмного забезпечення	23000,00	1095,24	63,00	69000,00
Консультант-аналітик	20000,00	952,38	15,00	14285,71
Консультант з економічних питань	20000,00	952,38	5,00	4761,90
Всього				163047,62

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою 5.2:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою 5.3:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $М_М$=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 день;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1{,}1 \cdot 1{,}65 / (21 \cdot 8) = 72{,}38$ грн.
$З_{р1} = 72{,}38 \cdot 8{,}00 = 579{,}07$ грн.
Величина витрат на основну заробітну плату робітників наведена в табл. 5.5.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Встановлення допоміжного офісного обладнання	8	2	1,1	72,38	579,07
Монтаж робочого місця розробника графічних систем	12	2	1,1	72,38	868,61
Інсталяція програмного забезпечення	5	5	1,7	111,87	559,33
Інші допоміжні роботи	10	3	1,1	72,38	723,84
Всього					2730,85

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою 5.4:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \qquad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод} = (163047{,}62 + 2730{,}85) \cdot 12 / 100\% = 19893{,}42$ грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 5.5:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (163047,62 + 2730,85+ 19893,42) · 22 / 100% = 40847,81433грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою 5.6:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}\ , \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 2 · 200,00 · 1,1 - 0,000 · 0,00 = 440,0 грн.

Проведені розрахунки зведемо до таблиці 5.6.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір Calipso Plus A4-500-80	200	2	0	0	440
Папір для записів Calipso Parers Light A5	170	1	0	0	187
Набір офісного працівника JOBMAX, с наповненням (16 предметів)	200	2	0	0	440
Flesh-пам'ять Kingston 32 GB	160	2	0	0	352
Всього					1419

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка методів і засобів 3D-гри з використанням двигуна Unity» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Витрати на спецустаткування, які використовують при проведенні НДР на тему «Розробка методів і засобів 3D-гри з використанням двигуна Unity» відсутні.

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для

проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою 5.7:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.7)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг} = 5405 \cdot 1 \cdot 1{,}12 = 6053{,}6$ *грн.*

Отримані результати зведемо до таблиці 5.7:

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Unity Pro	1	5405	6053,6
Photon Gaming Circle Single	1	4504	5044,48
Всього			11098,08

5.2.7 Амортизація обладнання, програмних засобів та приміщень

У спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою 5.8:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} \ , \qquad (5.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (59400{,}00 \cdot 3) / (5 \cdot 12) =$ 2970,00 *грн.*

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
ПК (2 шт)	59400	5	3	2970,00
Маршрутизатор	1870	3	3	155,83
Unity Pro	6053,6	3	3	504,47
Photon Gaming Circle Single	5044,48	3	3	420,37
Всього				4050,67

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою 5.9:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \qquad (5.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 7,50 грн;

$K_{вп i}$ – коефіцієнт, що враховує використання потужності, $K_{вп i} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,3 \cdot 504,0 \cdot 7,50 \cdot 0,95 / 0,97 = 1110,62$ *грн.*

Проведені розрахунки зведемо до таблиці 5.9.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Блок живлення ПК 1	0,3	504	1110,62
Блок живлення ПК 2	0,3	504	1110,62
Блок живлення маршрутизатора	0,02	60	8,81
Робоче місце дослідника	0,15	504	555,31
Оргтехніка	0,45	100	330,54
Всього			3115,90

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 5.10:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \qquad (5.10)$$

де $H_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св} = 20\%$.

$B_{св}$ = *(163047,62 + 2730,85) · 20 / 100% = 33155,69 грн.*

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою 5.11:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.11)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$B_{сп}$ = *(163047,62 + 2730,85) · 30 / 100% = 49733,54 грн.*

5.2.11 Інші витрати

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

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою 5.12:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.12)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = *(163047,62 + 2730,85) · 50 / 100% = 82889,23 грн.*

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з

освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою 5.13:

$$B_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \quad (5.13)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 150%.

$B_{нзв} = (163047,62 + 2730,85) \cdot 150 / 100\% = 248\ 667,70$грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою 5.14:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (5.14)$$

$B_{заг}$= 660649,52 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою 5.15:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.15)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,7.

$ЗВ = 660649,52 / 0,7 = 943785,03$ грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи

іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Розробка методів і засобів 3D-гри з використанням двигуна Unity» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 4000 користувачів;

2-й рік – 6000 користувачів;

3-й рік – 5000 користувачів.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 100000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 115300 грн;

$\pm\Delta Ц_o$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 1000 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою 5.16:

$$\Delta П_i = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.16)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(1000,00·100000,00+116300,00·4000)·0,83·0,3·(1-0,18/100%)= 115819653,6 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(1000,00·100000,00+116300,00·(4000+6000))·0,83·0,3·(1-0,18/100%)=258811434 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(1000,00·10000,00+116300,00·(4000+6000+5000))·0,83·0,3·(1-0,18/100%)= 377971251 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки, визначається за формулою 5.17:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.17)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,3;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП =$$

$115819653,6/(1+0,3)^1+258811434/(1+0,3)^2+377971251/(1+0,3)^3=414274615,29$грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки, , визначається за формулою 5.18:

$$PV = k_{інв} \cdot ЗВ, \tag{5.18}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=4;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 943785,03 грн.

$PV = k_{інв} \cdot ЗВ = 4 \cdot 943785,03 = 3775140,121$грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме за формулою 5.19:

$$E_{абс} = ПП - PV \tag{5.19}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 414274615,29грн;

PV – теперішня вартість початкових інвестицій, 3775140,121 грн.

$E_{абс} = ПП - PV$ = 414274615,29- 3775140,121= 410499475,17 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, , визначається за формулою 5.20:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.20)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 410499475,17грн;

PV – теперішня вартість початкових інвестицій, 3775140,121 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+410499475,17/3775140,121)^{1/3} = 3,79.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ визначається за формулою 5.21:

$$\tau_{мін} = d + f, \quad (5.21)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,18.

$\tau_{мін}$ = 0,11+0,18 = 0,29 < 3,79 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка методів і засобів 3D-гри з використанням двигуна Unity» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, визначається за формулою 5.22:

$$T_{ок} = \frac{1}{E_в}, \tag{5.22}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 3,79 = 0,26 року.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методів і засобів 3D-гри з використанням двигуна Unity» становить 40 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вищий середнього).

Також термін окупності становить 0,26 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення

її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методів і засобів 3D-гри з використанням двигуна Unity».

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи було створено методи та інструменти для реалізації 3D-гри з використанням двигуна Unity. Для розробки використовувались Visual Studio 2023 та Unity. Робота оформлена згідно методичних вказівок [32, 33].

У межах дослідження був проведений детальний аналіз сучасного стану питання розвитку комп’ютерних ігрових технологій, розглянуті основні аналоги, реалізовані на Unity, визначено їх особливості та недоліки, проведено порівняльний аналіз з власним програмним продуктом. Для реалізації програми було обрано мову програмування C# та технологію Unity для розробки 3D-гри.

У роботі розроблено власні 3D моделі, модель прецедентності, метод автоматичного підбору кімнат та метод аналізу та обробки дій користувача з використанням ресурсів сервера PUN.

У роботі був проведений варіантний аналіз і обґрунтовано вибір програмного забезпечення для реалізації програмного продукту. Був розроблений інтерфейс програмного засобу після попереднього аналізу та вибору інтерфейсу. Розроблені алгоритми роботи програмних модулів, створені програмні модулі для пересування, гравця, зброї, здоров'я гравця, ігрової кімнати та анімації.

Тестування програми підтвердило повну працездатність програмного продукту та відповідність поставленому технічному завданню. Також була розроблена інструкція користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Войтко В.В., Черноволик Г.О., Денисюк А.В., Воронін Є.С. Розробка засобів реалізації адаптивної 3D гри з використанням ігрового рушія UNITY // Електронні інформаційні ресурси: створення, використання, доступ. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 20-21 листопада 2023 р. – Суми/Вінниця: НІКО/ КЗВО «Вінницька академія безперервної освіти», 2023. – С. 77-80.

2. Video Games Help With Learning?. URL: https://ablegamers.org/video-games-help-with-learning/?gclid=Cj0KCQjwsp6pBhCfARIsAD3GZuZ-S4gyv_zzOEFIRSVmF6K46GZmNgn0QfKbCJSl5VMtZ0E87JmaGD0aAnA_EALw_wcB.

3. Video Game Genres: Everything You Need to Know. URL: https://www.hp.com/us-en/shop/tech-takes/video-game-genres.

4. Ultrakill's Actual Review Score. URL: Режим доступу: https://www.reddit.com/r/Ultrakill/comments/13bh0rk/ultrakills_actual_review_score/.

5. Mist Hunter. URL: https://www.metacritic.com/game/mist-hunter/.

6. Superhot Review. URL: https://www.gamespot.com/reviews/superhot-review/1900-6416368/.

7. 7 Days to Die Review. URL: https://www.ign.com/articles/2016/07/13/7-days-to-die-review.

8. The Forest. URL: https://opencritic.com/game/6029/the-forest.

9. Level Up: A Guide to Game UI (with Infographic). URL: https://www.toptal.com/designers/gui/game-ui.

10. What is user interface (UI)? URL: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI#:~:text=The%20user%20interface%20(UI)%20is,an%20application%20or%20a%20website..

11. What Is a User Interface, and What Are the Elements That Comprise One? URL: https://careerfoundry.com/en/blog/ui-design/what-is-a-user-interface/

12. 3D models URL: https://doc.arcgis.com/en/3d/workflows/content/3d-models.htm#:~:text=3D%20modeling%20data%20(3D%20models,trains%2C%20cars%2C%20trees%20etc.

13. 3D Model Creation URL: https://cgifurniture.com/3d-model-creation-5-stages/

14. What is 3D modeling used for? URL: https://www.adobe.com/products/substance3d/discover/what-is-3d-modeling.html

15. What is 3D modelling and what is it used for? URL: https://www.futurelearn.com/info/blog/general/what-is-3d-modelling.

16. What is Unified Modeling Language (UML)? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#:~:text=UML%2C%20short%20for%20Unified%20Modeling,business%20modeling%20and%20other%20non.

17. Презентація "Діаграми UML. Діаграми прецедентів". URL: https://naurok.com.ua/prezentaciya-diagrami-uml-diagrami-precedentiv-238715.html.

18. UML-МОДЕЛЮВАННЯ ІНФОРМАЦІЙНО-АНАЛІТИЧНОЇ СИСТЕМИ. URL: https://lib.iitta.gov.ua/706851/1/aisnn_uml.pdf.

19. Photon Unity Networking 2. URL: https://doc-api.photonengine.com/en/pun/current/index.html.

20. Exciting Developments in Photon Networking for Unity. URL: https://www.linkedin.com/pulse/exciting-developments-photon-networking-unity-bleedingedgestudio/

21. Unity Multiplayer Introduction With Photon. URL: https://theslidefactory.com/unity-multiplayer-introduction-with-photon/.

22. Differences between PUN and Unity Networking URL: https://subscription.packtpub.com/book/game-development/9781849692328/2/ch02lvl1sec24/differences-between-pun-and-unity-networking.

23. Matchmaking Guide. URL: https://doc.photonengine.com/pun/current/lobby-and-matchmaking/matchmaking-and-lobby.

24. Algorithms. URL: https://edu.gcfglobal.org/en/computer-science/algorithms/1/.

25. Photon PUN SDK. URL: https://www.photonengine.com/sdks#pun.

26. Unity values URL: https://unity.com/our-company

27. Unity Game Engine Guide: How to Get Started with the Most Popular Game Engine Out There. URL: https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/.

28. Learn C# for Unity — Beginners Guide to Unity. URL: https://circuitstream.com/blog/learn-c-for-unity.

29. What is Photon SDK? URL: https://docs.photonsdk.org.

30. @photon-sdk/photon-lib. URL: https://www.npmjs.com/package/@photon-sdk/photon-lib.

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

32. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Розробники: А.О. Семенов, Л.П. Громова, Т.В. Макарова, О.В. Сердюк. Вінниця : ВНТУ, 2021. – 60 с.

33. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О.Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 50 с.

ДОДАТКИ

Додаток А (Обов'язковий). Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів і засобів 3D-гри з використанням двигуна Unity» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

к.т.н., доц. каф. ПЗ. В.В. Войтко

"_19_"____вересня_____2023 р.

Виконав:

студент гр.ЗПІ-22м Є. С. Воронін

"_19_"_____вересня_____2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і засобів 3D-гри з використанням двигуна Unity».

Галузь застосування – настільні комп’ютери.

2. Підстава для розробки.

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

3. Мета та призначення розробки.

Метою роботи є підвищення якості процесу підбору суперників в мультиплеєрних іграх шляхом удосконалення методів та засобів аналізу дій гравця під час ігрових сесій, які враховують статистику росту його навичок, що дозволить автоматизувати процес підбору ігрових опонентів для забезпечення відповідних умов тренувань у середовищі ігрового процесу.

Призначення роботи – розробка 3D гри за допомогою двигуна Unity.

4. Вихідні дані для проведення НДР

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

1. Video Games Help With Learning? [Електронний ресурс] – Режим доступу до ресурсу:https://ablegamers.org/video-games-helpwithlearning/?gclid=Cj0KCQjwsp6pBhCfARIsAD3GZuZS4gyv.
2. Photon Unity Networking 2 [Електронний ресурс] – Режим доступу до ресурсу: https://doc-api.photonengine.com/en/pun/current/index.html.
3. Unity Game Engine Guide. [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/.

5. Технічні вимоги

Вхідні дані до роботи – підключення користувача, ім’я користувача, пересування, стрільба; вихідні дані – графічний інтерфейс, ігровий 3D простір,

підбір гравців, виведення статистики.

6. Конструктивні вимоги.

Інтерфейс програми повинен відповідати естетичним та ергономічним вимогам та мати зручну навігацію і засоби керування.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку навчальних ігор та постановка задач дослідження	20.09.2023 30.09.2023
2	Розробка методу та моделей гри	01.10.2023 17.10.2023
3	Розробка гри HeillGunner за допомогою двигуна Unity	18.10.2023 04.11.2023
4	Тестування програми	05.11.2023 21.11.2023
5	Економічна частина	22.11.2023 01.12.2023

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б (Обов'язковий). Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методів і засобів 3D-гри з використанням двигуна Unity**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, ЗПІ – 22м

Науковий керівник: к.т.н., доц. каф. ПЗ. В.В. Войтко

Unicheck	
Оригінальність	**98 %**
Схожість	2 %

Аналіз звіту подібності

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

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

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck.

Автор роботи _____ Воронін С.С

Керівник роботи _____ Войтко В.В.

Додаток В (Довідниковий). Лістинг програми

RoomManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Pun.Demo.PunBasics;
using Hashtable = ExitGames.Client.Photon.Hashtable;

public class RoomManager: MonoBehaviourPunCallbacks {
  public static RoomManager instance;

  public GameObject player;

  [Space]
  public Transform[] spawnPoints;

  [Space]
  public GameObject roomCam;

  [Space]
  public GameObject nameUI;

  public GameObject connectingUI;

  private string nickname = "unnamed";
```

```
public string roomNameToJoin = "test";

[HideInInspector]
public int kills = 0;
[HideInInspector]
public int deaths = 0;

private void Awake() {
  instance = this;
}

public void ChangeNickname(string _name) {
  nickname = _name;
}

public void JoinRoomButtonPressed() {
  Debug.Log("Connecting...");

  PhotonNetwork.JoinOrCreateRoom(roomNameToJoin, null, null);

  nameUI.SetActive(false);
  connectingUI.SetActive(true);
}

public override void OnJoinedRoom() {
  base.OnJoinedRoom();
  Debug.Log("We're connected and in a room now");

  roomCam.SetActive(false);
```

```
    SpawnPlayer();
  }

  public void SpawnPlayer() {
    Transform spawnPoint = spawnPoints[UnityEngine.Random.Range(0, spawnPoints.Length)];

    GameObject _player = PhotonNetwork.Instantiate(player.name, spawnPoint.position,
Quaternion.identity);
    _player.GetComponent < PlayerSetup > ().IsLocalPlayer();
    _player.GetComponent < Health > ().isLocalPlayer = true;

    _player.GetComponent < PhotonView > ().RPC("SetNickname", RpcTarget.AllBuffered,
nickname);
    PhotonNetwork.LocalPlayer.NickName = nickname;
  }

  public void SetHashes() {
    try {
      Hashtable hash = PhotonNetwork.LocalPlayer.CustomProperties;

      hash["kills"] = kills;
      hash["deaths"] = deaths;

      PhotonNetwork.LocalPlayer.SetCustomProperties(hash);
    }
    catch {
      //Do nothing
    }
  }
```

```
}
```

PlayerSetup.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using TMPro;

public class PlayerSetup : MonoBehaviour
{
  public Movement movement;

   public GameObject camera;

   public string nickname;

   public TextMeshPro nicknameText;



   public void IsLocalPlayer()
   {
      movement.enabled = true;
      camera.SetActive(true);
   }

   [PunRPC]
   public void SetNickname(string _name)
   {
```

```
        nickname = _name;

        nicknameText.text = nickname;
    }
}
```

Health.cs

```
using Photon.Pun;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class Health : MonoBehaviour
{

    public int health;
    public bool isLocalPlayer;

    public RectTransform healthBar;
    private float originalHeathBarSize;

    [Header("UI")]
    public TextMeshProUGUI helthText;



    private void Start()
```

```
{
    originalHeathBarSize = healthBar.sizeDelta.x;
}

private void Update()
{
    //healthBar.sizeDelta = new Vector2(originalHeathBarSize * health / 100f, healthBar.sizeDelta.y);
}



[PunRPC]
public void TakeDamage(int _damage)
{
    health -= _damage;

    healthBar.sizeDelta = new Vector2(originalHeathBarSize * health / 100f, healthBar.sizeDelta.y);

    helthText.text = health.ToString();



    if (health <= 0 )
    {
        if (isLocalPlayer)
        {
            RoomManager.instance.SpawnPlayer();

            RoomManager.instance.deaths++;
            RoomManager.instance.SetHashes();
        }
```

```
            Destroy(gameObject);
        }
    }
}
```

Movement.cs

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


[RequireComponent(typeof(Rigidbody))]

public class Movement : MonoBehaviour
{
    public float walkSpeed = 8f;
    public float sprintSpeed = 14f;
    public float maxVelocityChange = 10f;


    [Space]
    public float airControl = 0.5f;


    [Space]
    public float jumpHeight = 5f;
```

```
private Vector2 input;
private Rigidbody rb;

private bool sprinting;
private bool jumping;

private bool grounded = false;




// Start is called before the first frame update
void Start()
{
    rb = GetComponent<Rigidbody>();
}

// Update is called once per frame
void Update()
{
    input = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"));
    input.Normalize();

    sprinting = Input.GetButton("Sprint");
    jumping = Input.GetButton("Jump");
}



private void OnTriggerStay(Collider other)
```

```
    {
        grounded = true;
    }



    void FixedUpdate()
    {
        if (grounded)
        {
            if (jumping)
            {
                rb.velocity = new Vector3(rb.velocity.x, jumpHeight, rb.velocity.z);
            }
            else if (input.magnitude > 0.5f)
            {
                rb.AddForce(CalculateMovement(sprinting ? sprintSpeed : walkSpeed),
ForceMode.VelocityChange);
            }
            else
            {
                var velocity1 = rb.velocity;
                velocity1 = new Vector3(velocity1.x * 0.2f * Time.fixedDeltaTime, velocity1.y, velocity1.z
* 0.2f * Time.fixedDeltaTime);
                rb.velocity = velocity1;
            }
        } else
        {
            if (input.magnitude > 0.5f)
            {
```

```
            rb.AddForce(CalculateMovement(sprinting ? sprintSpeed * airControl : walkSpeed * airControl), ForceMode.VelocityChange);
        }
        else
        {
            var velocity1 = rb.velocity;
            velocity1 = new Vector3(velocity1.x * 0.2f * Time.fixedDeltaTime, velocity1.y, velocity1.z * 0.2f * Time.fixedDeltaTime);
            rb.velocity = velocity1;
        }
    }

    grounded = false;
}



Vector3 CalculateMovement(float _speed)
{
    Vector3 targetVelocity = new Vector3(input.x, 0, input.y);
    targetVelocity = transform.TransformDirection(targetVelocity);

    targetVelocity *= _speed;

    Vector3 velocity = rb.velocity;



    if(input.magnitude > 0.5f)
    {
        Vector3 velocityChange = targetVelocity - velocity;
```

```
            velocityChange.x = Mathf.Clamp(velocityChange.x, -maxVelocityChange,
maxVelocityChange);
            velocityChange.z = Mathf.Clamp(velocityChange.z, -maxVelocityChange,
maxVelocityChange);



            velocityChange.y = 0;



            return (velocityChange);
        }
        else
        {
            return new Vector3();
        }
    }
}
```

Leaderboard.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
using Photon.Pun;
using TMPro;
using Photon.Realtime;
using Photon.Pun.UtilityScripts;

public class Leaderboard : MonoBehaviour
```

```
{
    public GameObject playersHolder;

    [Header("Options")]
    public float refreshRate = 1f;

    [Header("UI")]
    public GameObject[] slots;

    [Space]
    public TextMeshProUGUI[] scoreTexts;
    public TextMeshProUGUI[] nameTexts;
    public TextMeshProUGUI[] kdTexts;


    private void Start()
    {
        InvokeRepeating(nameof(Refresh), 1f, refreshRate);
    }


    public void Refresh()
    {
        foreach (var slot in slots)
        {
            slot.SetActive(false);
        }
```

```
    var sortedPlayerList =
        (from player in PhotonNetwork.PlayerList orderby player.GetScore() descending select
player).ToList();


    int i = 0;
    foreach (var player in sortedPlayerList)
    {
        slots[i].SetActive(true);


        if(player.NickName == "")
            player.NickName = "unnamed";



        nameTexts[i].text = player.NickName;
        scoreTexts[i].text = player.GetScore().ToString();



        if (player.CustomProperties["kills"] != null)
        {
            kdTexts[i].text = player.CustomProperties["kills"] + "/" +
player.CustomProperties["deaths"];
        }
        else
        {
            kdTexts[i].text = "0/0";
        }



        i++;
    }
```

```
    }

    private void Update()
    {
        playersHolder.SetActive(Input.GetKey(KeyCode.Tab));
    }
}
```

RoomList.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
using TMPro;

public class RoomList : MonoBehaviourPunCallbacks
{
    public static RoomList Instance;

    public GameObject roomManagerGameobject;
    public RoomManager roomManager;


    [Header("UI")]
    public Transform roomListParent;
    public GameObject roomListItemPrefab;
```

```
private List<RoomInfo> cachedRoomList = new List<RoomInfo>();


public void ChangeRoomToCreateName(string _roomName)
{
    roomManager.roomNameToJoin = _roomName;
}


private void Awake()
{
    Instance = this;
}


IEnumerator Start()
{
    //Precautions
    if (PhotonNetwork.InRoom)
    {
        PhotonNetwork.LeaveRoom();
        PhotonNetwork.Disconnect();
    }

    yield return new WaitUntil(() => !PhotonNetwork.IsConnected);


    PhotonNetwork.ConnectUsingSettings();
```

```
}


public override void OnConnectedToMaster()
{
    base.OnConnectedToMaster();


    PhotonNetwork.JoinLobby();
}

public override void OnRoomListUpdate(List<RoomInfo> roomList)
{
    if(cachedRoomList.Count <= 0)
    {
        cachedRoomList = roomList;
    }
    else
    {
        foreach(var room in roomList)
        {
            for(int i = 0; i < cachedRoomList.Count; i++)
            {

                if (cachedRoomList[i].Name == room.Name)
                {
                    List<RoomInfo> newList = cachedRoomList;

                    if (room.RemovedFromList)
```

```
                {
                    newList.Remove(newList[i]);
                }
                else
                {
                    newList[i] = room;
                }


                cachedRoomList = newList;

            }

        }
    }
    UpdateUI();
}


private void UpdateUI()
{
    foreach (Transform roomItem in roomListParent)
    {
        Destroy(roomItem.gameObject);
    }
```

```
        foreach (var room in cachedRoomList)
        {
            GameObject roomItem = Instantiate(roomListItemPrefab, roomListParent);

            roomItem.transform.GetChild(0).GetComponent<TextMeshProUGUI>().text = room.Name;
            roomItem.transform.GetChild(1).GetComponent<TextMeshProUGUI>().text =
room.PlayerCount + "/16";

            roomItem.GetComponent<RoomItemButton>().RoomName = room.Name;

        }
    }

    public void JoinRoomByName(string _name)
    {
        roomManager.roomNameToJoin = _name;
        roomManagerGameobject.SetActive(true);
        gameObject.SetActive(false);
    }

}
```

Weapon.cs

```
using Photon.Pun;
using Photon.Pun.UtilityScripts;
using System.Collections;
using System.Collections.Generic;
using TMPro;
```

```
using UnityEngine;
using UnityEngine.UI;

public class Weapon : MonoBehaviour
{

    public Image ammoCircle;

    public int damage;

    public Camera camera;

    public float fireRate;

    private float nextFire;

    [Header("VFX")]
    public GameObject hitVFX;

    [Header("Ammo")]
    public int mag = 5;

    public int ammo = 30;
    public int magAmmo = 30;

    [Header("UI")]
    public TextMeshProUGUI magText;
    public TextMeshProUGUI ammoText;
```

```
[Header("Animation")]
public Animation animation;
public AnimationClip reload;

[Header("Recoil Settings")]
//[Range(0, 1)]
//public float recoilPercent = 0.3f;

[Range(0, 2)]
public float recoverPercent = 0.7f;

[Space]
public float recoilUp = 1f;
public float recoilBack = 0f;


private Vector3 originalPosition;
private Vector3 recoilVelocity = Vector3.zero;

private float recoilLength;
private float recoverLength;


private bool recoiling;
public bool recovering;


void SetAmmo()
{
```

```
    ammoCircle.fillAmount = (float)ammo / magAmmo;
}

private void Start()
{
    magText.text = mag.ToString();
    ammoText.text = ammo + "/" + magAmmo;

    SetAmmo();

    originalPosition = transform.localPosition;

    recoilLength = 0;
    recoverLength = 1 / fireRate * recoverPercent;
}


// Update is called once per frame
void Update()
{
    if (nextFire > 0)
        nextFire -= Time.deltaTime;


    if(Input.GetButton("Fire1") && nextFire <= 0 && ammo > 0 && animation.isPlaying == false)
    {
        nextFire = 1 / fireRate;

        ammo--;
```

```
        magText.text = mag.ToString();
        ammoText.text = ammo + "/" + magAmmo;

        SetAmmo();

        Fire();
    }

    if (Input.GetKeyDown(KeyCode.R) && mag > 0)
    {
        Reload();
    }


    if (recoiling)
    {
        Recoil();
    }


    if(recovering)
    {
        Recovering();
    }
}

private void Reload()
{
```

```
    animation.Play(reload.name);

    if(mag > 0)
    {
        mag--;

        ammo = magAmmo;
    }

    magText.text = mag.ToString();
    ammoText.text = ammo + "/" + magAmmo;

    SetAmmo();
}

void Fire()
{
    recoiling = true;
    recovering = false;

    Ray ray = new Ray(camera.transform.position, camera.transform.forward);

    RaycastHit hit;

    //PhotonNetwork.LocalPlayer.AddScore(1);


    if (Physics.Raycast(ray.origin, ray.direction, out hit, 100f))
    {
```

```
        PhotonNetwork.Instantiate(hitVFX.name, hit.point, Quaternion.identity);

        if (hit.transform.gameObject.GetComponent<Health>())
        {
            //PhotonNetwork.LocalPlayer.AddScore(damage);

            if(damage >= hit.transform.gameObject.GetComponent<Health>().health)
            {
                //Kill

                RoomManager.instance.kills++;
                RoomManager.instance.SetHashes();

                PhotonNetwork.LocalPlayer.AddScore(100);
            }

            hit.transform.gameObject.GetComponent<PhotonView>().RPC("TakeDamage",
RpcTarget.All, damage);
        }
    }
}
void Recoil()
{
    Vector3 finalPosition = new Vector3(originalPosition.x, originalPosition.y + recoilUp,
originalPosition.z - recoilBack);

    transform.localPosition =
        Vector3.SmoothDamp(transform.localPosition, finalPosition, ref recoilVelocity, recoilLength);

    if(transform.localPosition == finalPosition)
```

```
        {
            recoiling = false;
            recovering = true;
        }
    }


    void Recovering()
    {
        Vector3 finalPosition = originalPosition;

        transform.localPosition =
            Vector3.SmoothDamp(transform.localPosition, finalPosition, ref recoilVelocity,
recoverLength);

        if (transform.localPosition == finalPosition)
        {
            recoiling = false;
            recovering = false;
        }
    } }
```

WeaponSwitcher.cs

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

public class WeaponSwitcher : MonoBehaviour
{
```

```
public Animation _animation;
public AnimationClip draw;

private int selectedWeapon = 0;


// Start is called before the first frame update
void Start()
{
    SelectWeapon();
}

// Update is called once per frame
void Update()
{
   int previouseSelectedWeapon = selectedWeapon;


    if (Input.GetKeyDown(KeyCode.Alpha1))
    {
        selectedWeapon = 0;
    }

    if (Input.GetKeyDown(KeyCode.Alpha2))
    {
        selectedWeapon = 1;
    }

    if (Input.GetKeyDown(KeyCode.Alpha3))
```

```
{
    selectedWeapon = 2;
}

if (Input.GetKeyDown(KeyCode.Alpha4))
{
    selectedWeapon = 3;
}

if (Input.GetKeyDown(KeyCode.Alpha5))
{
    selectedWeapon = 4;
}

if (Input.GetKeyDown(KeyCode.Alpha6))
{
    selectedWeapon = 5;
}

if (Input.GetKeyDown(KeyCode.Alpha7))
{
    selectedWeapon = 6;
}

if (Input.GetKeyDown(KeyCode.Alpha8))
{
    selectedWeapon = 7;
}
```

```
if (Input.GetKeyDown(KeyCode.Alpha9))
{
    selectedWeapon = 8;
}


if(Input.GetAxis("Mouse ScrollWheel") > 0)
{
    if(selectedWeapon >= transform.childCount - 1)
    {
        selectedWeapon = 0;
    }
    else
    {
        selectedWeapon += 1;
    }
}


if (Input.GetAxis("Mouse ScrollWheel") < 0)
{
    if (selectedWeapon <= 0)
    {
        selectedWeapon = transform.childCount - 1;
    }
    else
    {
        selectedWeapon -= 1;
    }
```

```
    }


    if (previouseSelectedWeapon != selectedWeapon)
    {
        SelectWeapon();
    }
}


void SelectWeapon()
{
    if(selectedWeapon >= transform.childCount)
    {
        selectedWeapon = transform.childCount - 1;
    }

    _animation.Stop();
    _animation.Play(draw.name);


    int i = 0;

    foreach(Transform _wepon in transform)
    {
        if(i == selectedWeapon)
        {
            _wepon.gameObject.SetActive(true);
        }
```

```
            else
            {
                _wepon.gameObject.SetActive(false);
            }

            i++;
        }
    }
}
```

Sway.cs

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

public class Sway : MonoBehaviour
{
    [Header("Settings")]
    public float swayClamp = 0.09f;
    [Space]
    public float smoothing = 3f;


    private Vector3 origin;

    // Start is called before the first frame update
    void Start()
    {
```

```
        origin = transform.localPosition;
    }

    // Update is called once per frame
    void Update()
    {
        Vector2 input = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse Y"));

        input.x = Mathf.Clamp(input.x, -swayClamp, swayClamp);
        input.y = Mathf.Clamp(input.y, -swayClamp, swayClamp);

        Vector3 target = new Vector3(-input.x, -input.y, 0);

        transform.localPosition = Vector3.Lerp(transform.localPosition, target + origin, Time.deltaTime *
smoothing);
    }
}
```

Додаток Г (Довідниковий). Ілюстративна частина

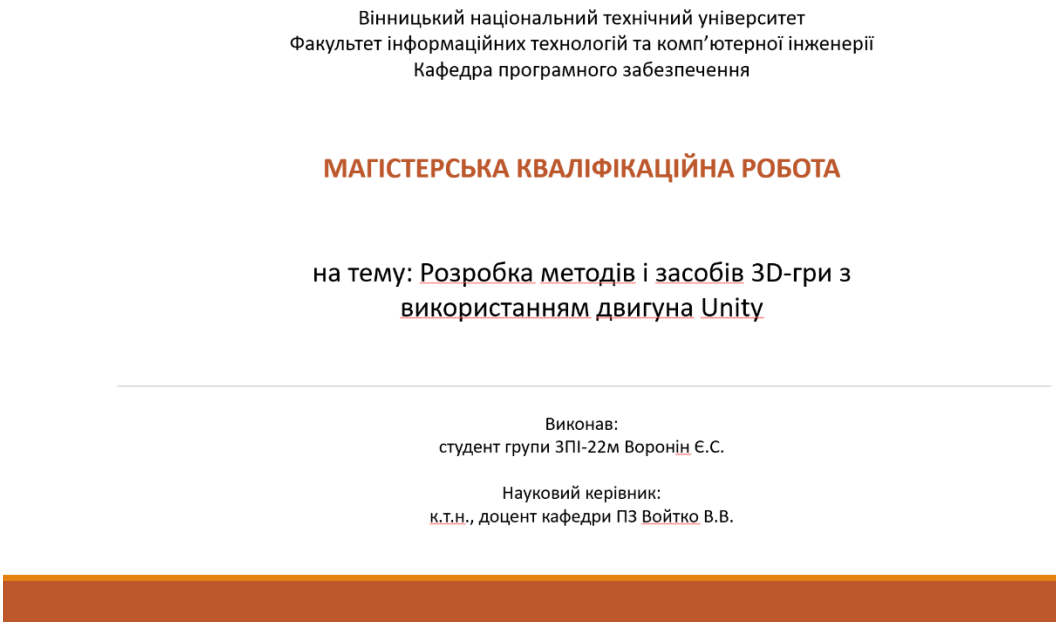

Рисунок Г.1 – Титульна сторінка

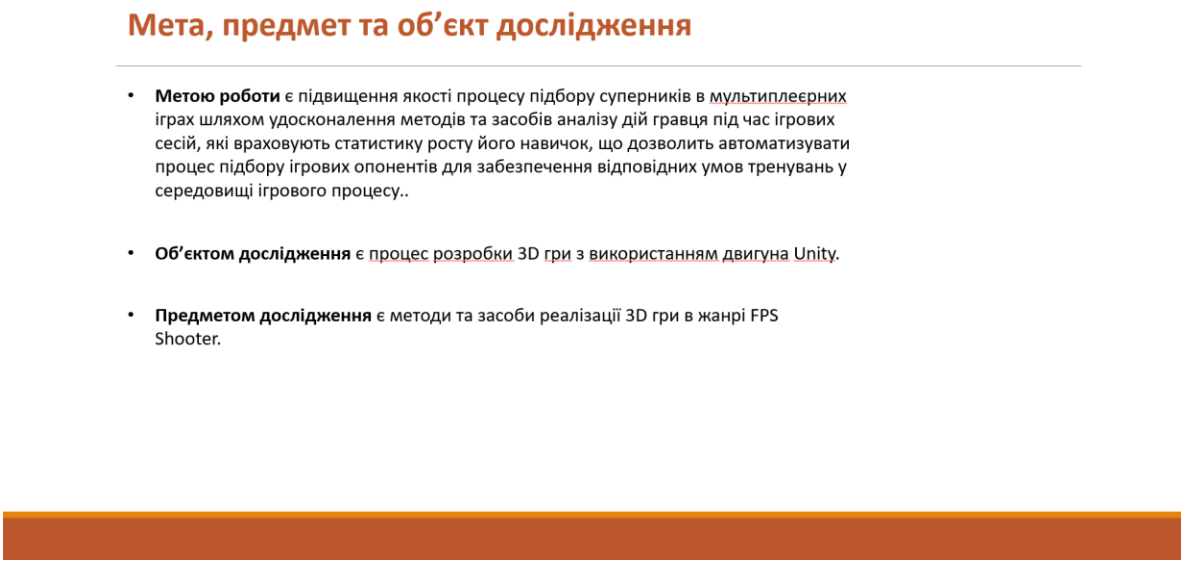

Рисунок Г.2 – Мета, об'єкт та предмет роботи

Постановка задач:

- Визначити засоби моніторингу ігрового процесу та аналізу дій гравця;
- Розробити метод аналізу та обробки дій гравця для коректного підбору ігрових суперників;
- Розробити метод автоматичного підбору ігрових кімнат з урахуванням рівня досягнень гравця;
- Розробити моделі та алгоритми роботи ігрової системи;
- Розробити програмні модулі компонентів відеогри в жанрі FPS Shooter;
- Провести тестування ігрової системи.

Рисунок Г.3 – Постановка задач

Наукова новизна розробки

- Подальшого розвитку отримав метод аналізу та обробки дій гравця, який, на відміну від існуючих, орієнтований на ведення статистики досягнень з урахуванням результатів параметричного аналізу ігрової сесії в багатокористувацькому режимі з використанням ресурсів сервера PUN, що дозволяє ведення моніторингу ігрового процесу та здійснення аналізу ключових параметрів у реальному часі для формування рекомендованого підбору суперників для наступного ігрового змагання
- Подальшого розвитку отримав метод автоматичного підбору ігрових кімнат, який, на відміну від існуючих, враховує рівень досягнень гравця за результатами моніторингового аналізу його дій у попередній ігровій сесії, що дозволяє забезпечити зручні умови проведення ігрових змагань.

Рисунок Г.4 – Наукова новизна

Практичне значення одержаних результатів

Розроблена відеогра «HeillGunner» може використовуватися гравцями в індивідуальному режимі та в багатокористувацькому режимі для проведення FPS Shooter змагань. Гра може бути розміщена на таких площадках продажу відеоігор для настільних пристроїв, як: **Steam, GOG та EGS**.

Ігрова платформа
Steam

Ігрова платформа
GOG

Ігрова платформа
EGS

Рисунок Г.5 – Практичне значення

Аналіз аналогів

Критерій для порівняння	ULTRAKILL	Mist Hunter	SUPERHOT	7 Days to Die	The Forest	Власна розробка
Мультиплеєр	-	-	-	+	+	+
Кросплатформенність	-	-	+	-	+	+
Ігровий процес	+	+	+	+	+	+
Оптимізація	+	+	+	-	+	+
Безкоштовна гра	-	-	-	-	-	+
Реіграбельність	-	-	+	+	+	+

Рисунок Г.6 – Аналіз аналогів

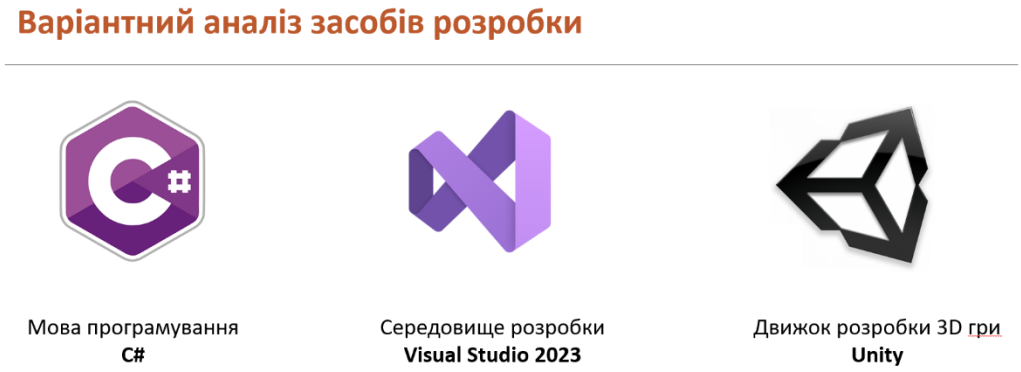

Рисунок Г.7 – Варіативний аналіз засобів розробки

Рисунок Г.8 – Структура інтерфейсу

Рисунок Г.9 – 3D – модель (Персонаж)

Рисунок Г.10 – 3D – модель (Зброя)

Рисунок Г.11 – 3D – модель (Ігрова локація)

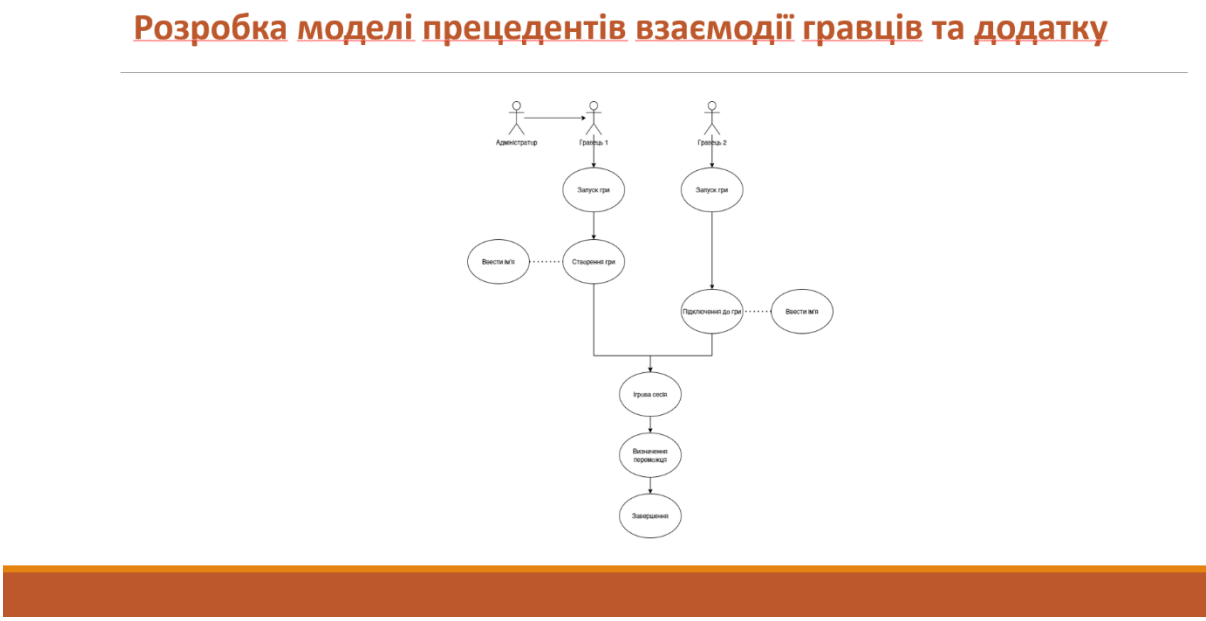

Рисунок Г.12 – Модель прецедентності

Розробка методу аналізу та обробки дій гравця з використанням ресурсів сервера PUN

Розглянемо метод аналізу та обробки дій гравця, який орієнтований на ведення статистики досягнень з урахуванням результатів параметричного аналізу ігрової сесії в багатокористувацькому режимі з використанням ресурсів сервера PUN:

1. Збираємо дані про гравця. PUN (Photon Unity Networking) надає зручний спосіб синхронізувати стани гравців між клієнтами та сервером. Забезпечимо відстеження ключових параметрів, таких як координати гравця, напрямок його руху, стан здоров'я та стан амуніції. Використано PhotonCallbacks для обробки різних подій, що виникають у грі.

2. Визначаємо ключові події. Ретельно визначаємо події, які мають велике значення для геймплею. Це може включати стрільбу, попадання або промахи, зміну зброї та пересування в конкретні області. Важливо зафіксувати всі можливі впливи на гру.

3. Збір та обробка статистики. Створено систему для збору та зберігання статистики гравців, яка включає в себе не лише базову статистику, а й деталізовані дані про взаємодію гравців між собою.

4. Взаємодія з ігровим сервером. Використано Photon для передачі результатів аналізу на сервер. Встановлено двосторонню взаємодію між клієнтами та сервером для ефективного обміну даними та забезпечення коректності аналізу.

5. Реакція на аналіз. Механіки гри, які реагують на результати аналізу, що включає в себе винагороди або покарання для гравців, які демонструють високий рівень вмінь чи, навпаки, допускають багато помилок.

Рисунок Г.13 – Метод обробки дій гравця (Частина 1)

Блок-схема алгоритму роботи методу аналізу та обробки дій гравця

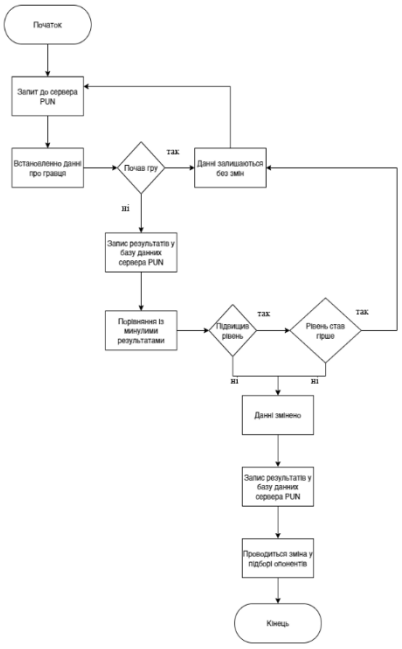

Рисунок Г.14– Метод обробки дій гравця (Частина 2)

Розробка методу автоматичного підбору ігрових кімнат

Метод автоматичного підбору ігрових кімнат акумулює функціонал:

1. Збір даних. Система автоматичного підбору кімнат починає аналіз зі збору даних про навички гравців. Це може включати в себе рейтинг, статистику попередніх ігор, рівень навичок та інші важливі параметри.
2. Категоризація гравців. Гравці розподіляються за категоріями на основі експертизи їхнього рівня навичок. Наприклад, можна визначити категорії "початківці", "середні гравці" та "професіонали".
3. Визначення параметрів пошуку. Встановлюються параметри пошуку для автоматичного підбору ігрової кімнати. Це може включати в себе максимальний та мінімальний рівень навичок гравців, щоб забезпечити балансовані ігри.
4. Запуск пошуку. Система запускає пошук доступних ігрових кімнат, спираючись на визначені параметри. Мета пошуку полягає в тому, щоб знайти кімнату з гравцями, які мають схожий рівень навичок.
5. Відображення результатів. Гравцю надсилається інформація про знайдену кімнату, включаючи рівень навичок інших гравців та характеристики гри.
6. Приєднання. Гравець має можливість приєднатися до запропонованої кімнати або відхилити пропозицію та продовжити пошук.
7. Динамічне оновлення. Під час гри система динамічно оновлює рівень навичок гравців у кімнаті, враховуючи їхні досягнення та результати гри.
8. Автоматичне корегування. Залежно від результатів гри, гравці можуть бути перерозподілені між кімнатами для забезпечення постійного балансу.
9. Завершення ігрової сесії. Після завершення ігрової сесії гравець має можливість залишити кімнату, взяти участь у новій ігровій сесії чи вибрати інші ігрові опції.

Рисунок Г.15 – Метод автоматичного підбору ігрових кімнат

Розробка алгоритму роботи ігрової системи

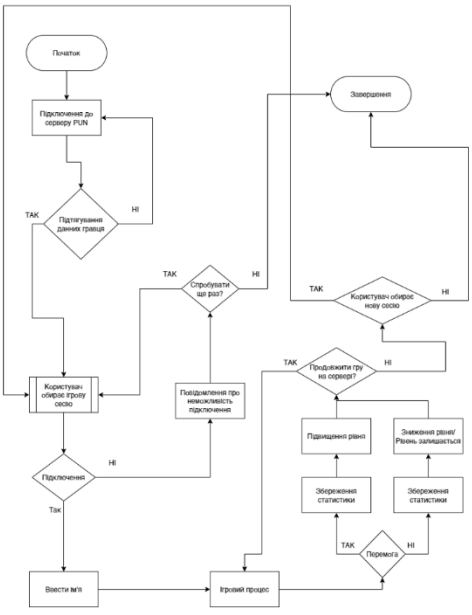

Рисунок Г.16 – Алгоритм роботи ігрової системи

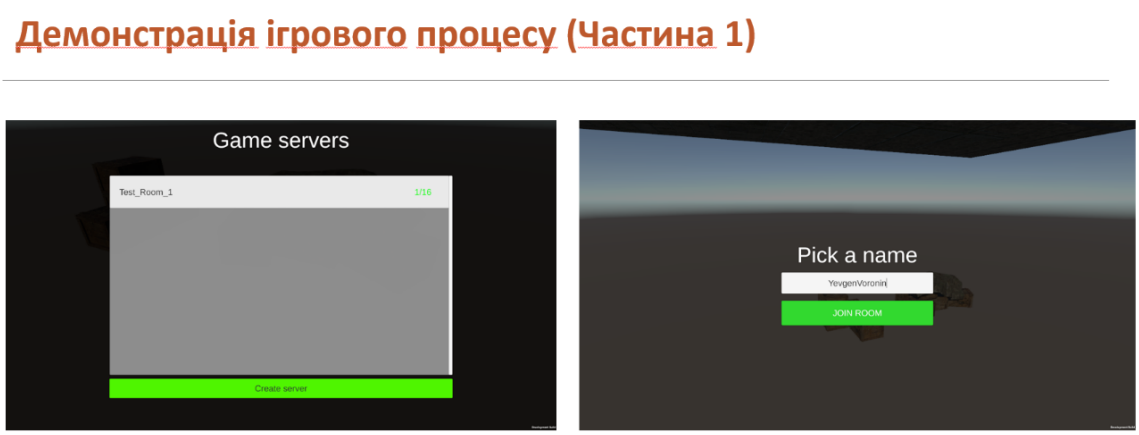

Рисунок Г.17 – Демонстрація ігрового процесу (Частина 1)

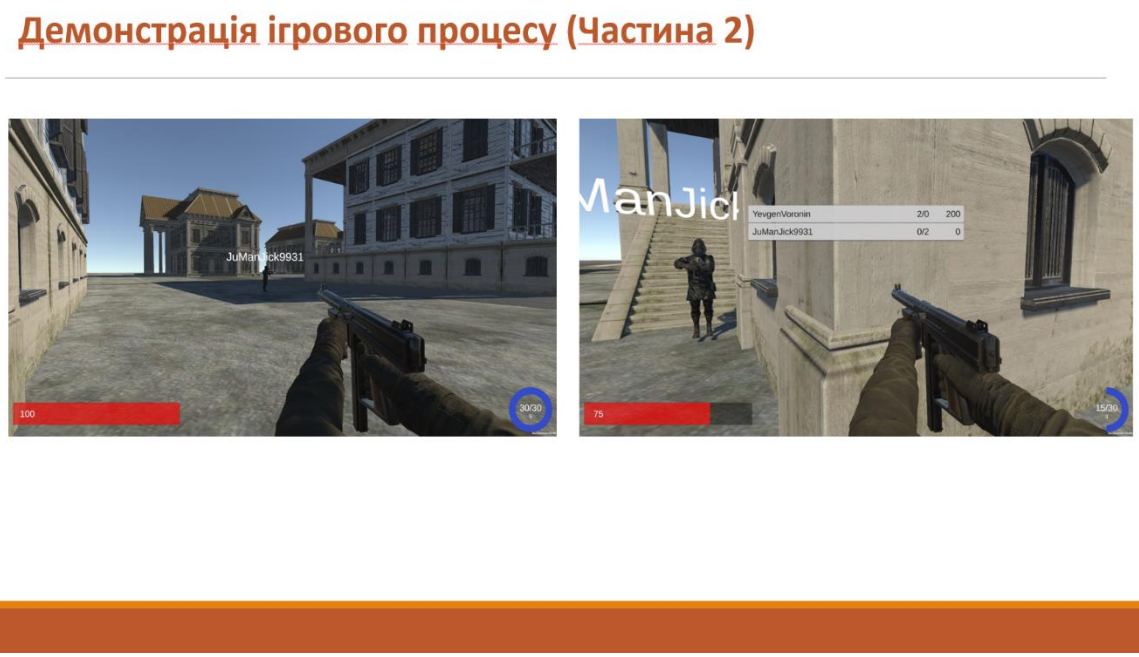

Рисунок Г.18 – Демонстрація ігрового процесу (Частина 2)

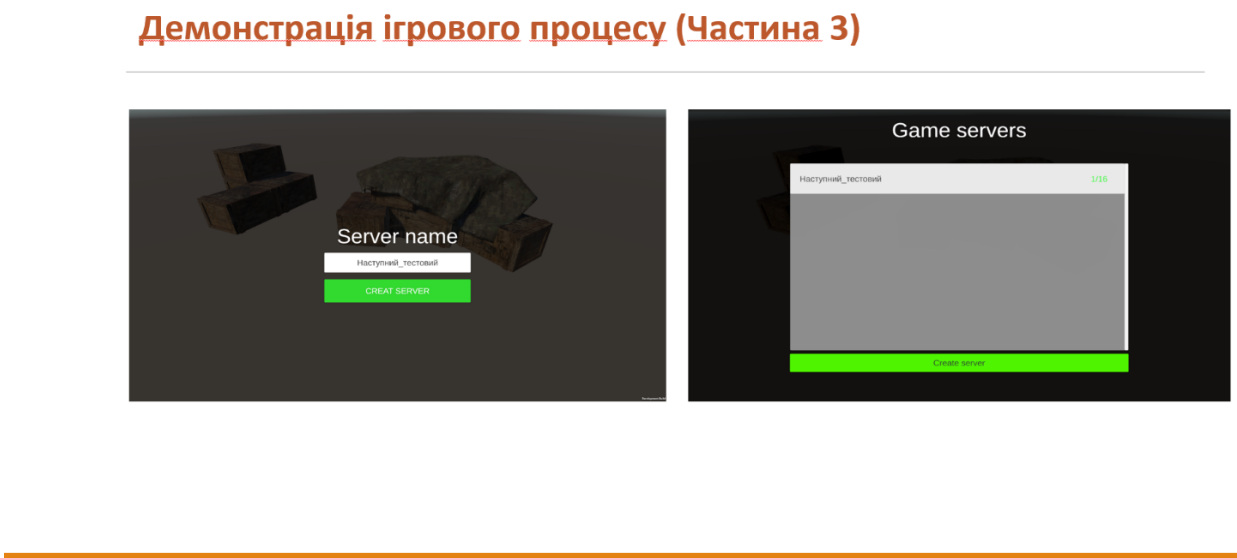

Рисунок Г.19 – Демонстрація ігрового процесу (Частина 3)

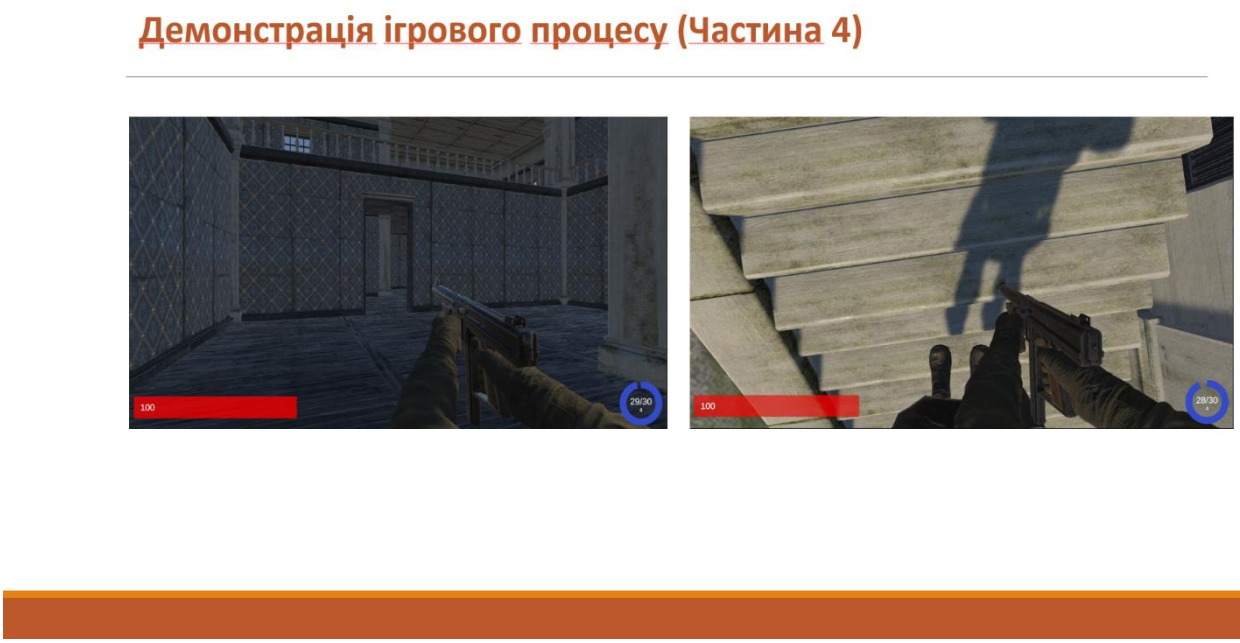

Рисунок Г.20 – Демонстрація ігрового процесу (Частина 4)

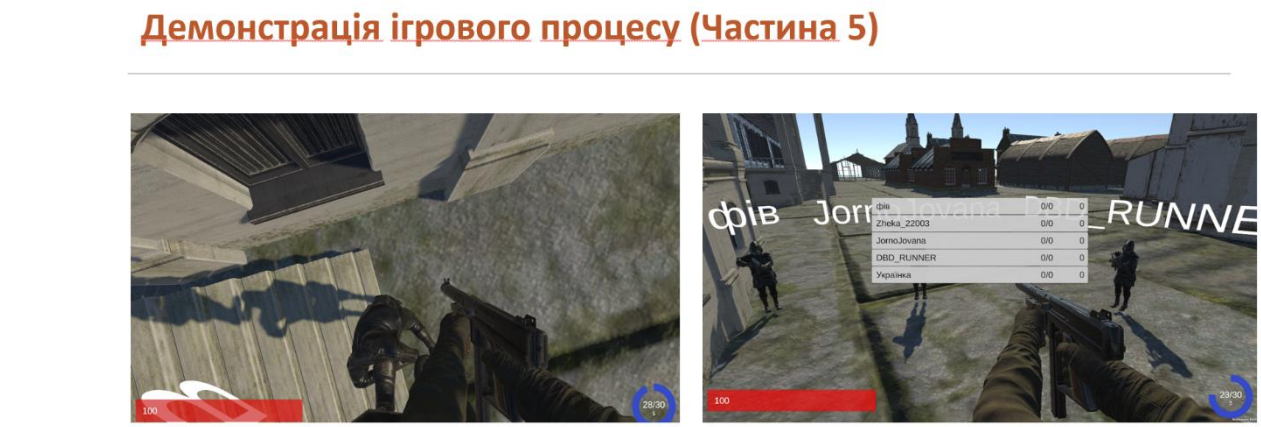

Рисунок Г.21 – Демонстрація ігрового процесу (Частина 5)

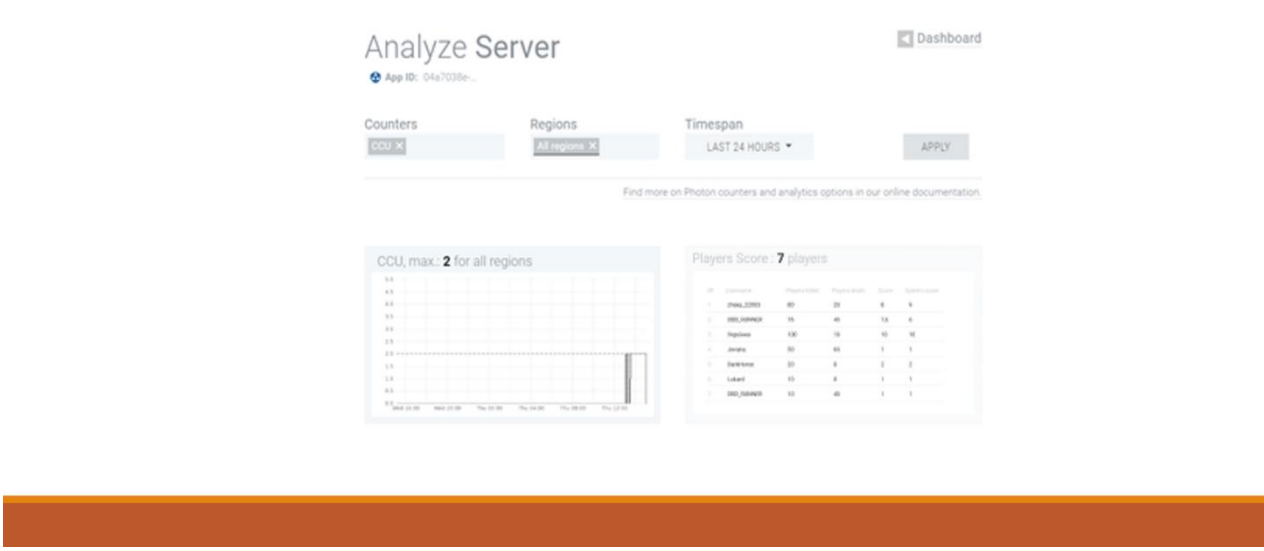

Рисунок Г.22 – Адміністративна панель сервера PUN

Апробація та публікація

- Апробація матеріалів магістерської кваліфікаційної роботи. Результати магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ - 2023»
- Публікація. Войтко В.В., Черноволик Г.О., Денисюк А.В., Воронін Є.С. Розробка засобів реалізації адаптивної 3D гри з використанням ігрового рушія UNITY // Електронні інформаційні ресурси: створення, використання, доступ. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 20-21 листопада 2023 р. – Суми/Вінниця: НІКО/ КЗВО «Вінницька академія безперервної освіти», 2023. – С. 77-80.

Рисунок Г.23 – Апробація та публікація

Висновки

- Визначено засоби моніторингу ігрового процесу та аналізу дій гравця;
- Розроблено метод аналізу та обробки дій гравця для коректного підбору ігрових суперників;
- Розроблено метод автоматичного підбору ігрових кімнат з урахуванням рівня досягнень гравця;
- Розроблено моделі та алгоритми роботи ігрової системи;
- Розроблено програмні модулі компонентів відеогри в жанрі FPS Shooter;
- Проведено тестування ігрової системи.

Рисунок Г.24 – Висновки

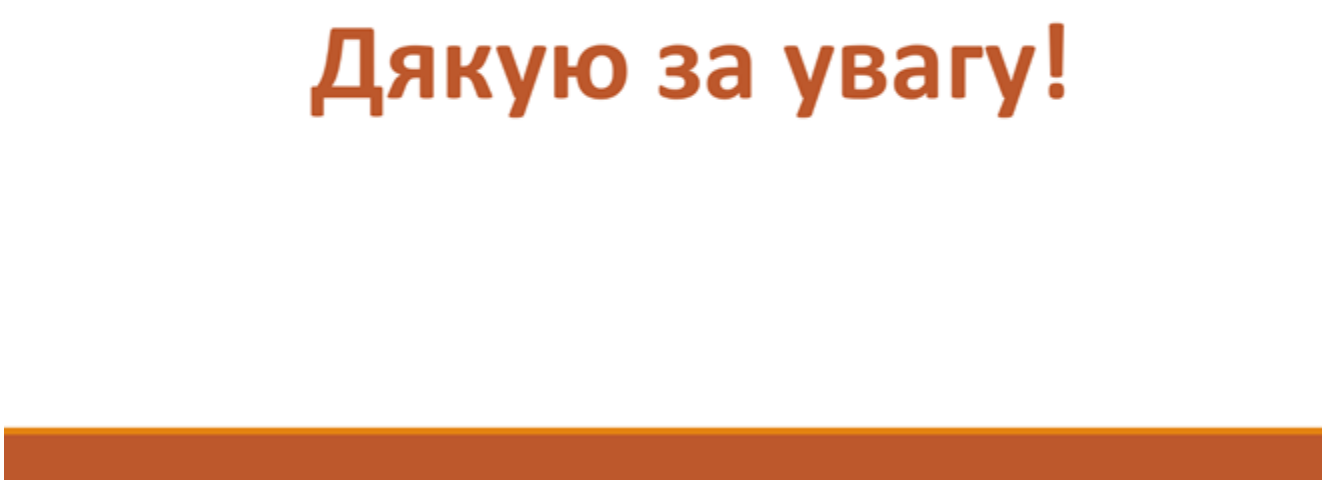

Рисунок Г.25 – Фінальний слайд