МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко

підпис

«___» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

3D-гра на платформі Unity

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.21910808

Студент _____В. І. Димінський

підпис

«__» _____ 2023 р.

Керівник канд. техн. наук, доцент _____ Г. В. Горбань

підпис

«__» _____ 2023 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

підпис

«__» _____ 2023 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення, канд.
техн. наук, доцент,
_____Є.О. Давиденко
«____»_____2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано студенту групи 408 факультету комп'ютерних наук

Димінському Віталію Івановичу
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи

«3D-гра на платформі Unity»

Затверджена наказом по ЧНУ від «17» березня 2023 р. № 60

2. Строк представлення кваліфікаційної роботи «___»_____ 2023 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є програмне забезпечення у вигляді 3D-гри у жанрі Shooter-Action

4. Перелік питань, що підлягають розробці

Аналіз предметної області, порівняльний аналіз аналогів, визначення вимог та функціоналу системи, моделювання, проектування програмного застосунку, кодування та тестування застосунку.

5. Перелік графічних матеріалів:

Презентація

6. Завдання до спеціальної частини

Охорона праці на робочих місцях фахівців з інформаційних технологій

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
А. О. Алексєєва	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи

канд. техн. наук, доцент Горбань Гліб Валентинович

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

Димінський Віталій Іванович

(прізвище, ім'я, по батькові студента)

_____

(підпис)

Дата видачі завдання «____» _____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: «3D-гра на платформі Unity».

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	17.03.2023р.	18.03.2023р.	виконано
2.	Огляд літератури за темою роботи	20.03.2023р.	22.03.2023р.	виконано
3.	Складання календарного плану КРБ	24.03.2023р.	24.03.2023р.	виконано
4.	Аналіз предметної області	26.03.2023р.	29.03.2023р.	виконано
5.	Моделювання та конструювання ПЗ	01.04.2023р.	05.04.2023р.	виконано
6.	Кодування, тестування розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	10.04.2023р.	01.06.2023	виконано
7.	Розробка спеціальної частини з охорони праці	23.05.2023	26.05.2023	виконано
8.	Відгук керівника КРБ	19.06.2023	19.06.2023	виконано
9.	Оформлення КРБ та презентації	01.06.2023	05.06.2023	виконано
10.	Попередній захист	06.06.2023	07.06.2023	виконано
11.	Рецензування	08.06.2023	12.06.2023	виконано
12.	Завершення оформлення КРБ та презентації	12.06.2023	18.06.2023	виконано
13.	Захист кваліфікаційної роботи	26.06.2023	27.06.2023	виконано

Розробив студент Димінський Віталій Іванович _____

(прізвище, ім'я, по батькові студента) (*підпис*)

«____»_____ 2023 р.

Керівник роботи канд. техн. наук, доцент Горбань Гліб Валентинович _____

(прізвище, ім'я, по батькові студента) (*підпис*)

«____»_____ 2023 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«3D-гра на платформі Unity»

Студент 408 гр.: Димінський Віталій Іванович

Керівник: канд. техн. наук, доцент Горбань Г. В.

Захищена: «_27_» червня 2023 р.

Короткий зміст праці:

У сучасному світі розробка ігор для мобільних платформ стала однією з найбільш перспективних та прибуткових галузей розвитку програмного забезпечення. Жанр Shooter-action на Android-платформі є одним з найбільш популярних, але на даний момент не так багато доступних 3D-ігор такого жанру з високою якістю графіки та інтерактивності.

Розробка таких ігор має велику актуальність, оскільки вони забезпечують більш глибокий та реалістичний досвід гри, що включає в себе високоякісну графіку, відмінний звуковий супровід та більш реалістичну фізику поведінки об'єктів в грі. Розробка Android-гри у жанрі Shooter-action у середовищі розробки Unity є об'єктом роботи, а предметом роботи є аналіз технологій та алгоритмів, що використовуються для реалізації ігрового процесу в сучасних проєктах, які призначені для мобільних пристроїв з операційною системою Android.

Метою є покращення розвитку галузі мобільної ігрової індустрії шляхом аналізу технологій та алгоритмів, які використовуються в реалізації ігрового процесу.

Об'єкт роботи: Дослідження ігрового процесу у Android-іграх.

Предмет роботи: Технології та алгоритми, що використовуються для реалізації ігрового процесу в сучасних проєктах, які призначені для мобільних пристроїв з ОС Android.

Мета: Вдосконалення навиків у напрямку ігрової індустрії за рахунок розробки гри жанру Shooter-Action, яка демонструє можливості та переваги використання платформи Unity.

Для досягнення поставленої цілі необхідно виконати наступні завдання:

1. Дослідити основні елементи та інструменти платформи Unity;
2. Провести аналіз існуючих додатків з подібними функціями;
3. Визначити вимоги та специфікації до розроблюваного додатку;
4. Реалізувати різні режими гри з можливістю вибору рівня складності;
5. Створити прототип гри.

КРБ викладена на _77_ сторінок, вона містить _4_ розділи, _69_ ілюстрацій, _7_ таблиць, _18_ джерел в переліку посилань.

ABSTRACT

of the Bachelor's Thesis

"3D Game on the Unity Platform"

Student of group 408: Dyminskyi Vitalii Ivanovich

Supervisor: Ph. D. in Technical Science, Associate Professor Horban H. V.

Summary of work:

In the modern world, game development for mobile platforms has become one of the most promising and profitable areas of software development. The shooter-action genre on the Android platform is one of the most popular, but currently there are not many available 3D shooter-action games with high-quality graphics and interactivity.

The development of such games is highly relevant, as they provide a deeper and more realistic gaming experience, including high-quality graphics, excellent sound design, and more realistic physics of objects in the game. The development of an Android game application in the shooter-action genre on the Unity development platform is the object of this thesis, and the subject of the work is the analysis of technologies and algorithms used to implement the game process in modern projects designed for mobile devices running the Android operating system.

The aim is to improve the development of the mobile gaming industry by analyzing technologies and algorithms used in the implementation of the game process.

Object of the thesis: Research on the gameplay process in Android games.

Subject of the thesis: Technologies and algorithms used for implementing the gameplay process in modern projects designed for mobile devices with the Android operating system.

Goal: Improving skills in the gaming industry through the development of a Shooter-Action genre game that showcases the capabilities and advantages of using the Unity platform.

To achieve the set goal, the following tasks must be performed:

1. Explore the main elements and tools of the Unity platform;
2. Conduct an analysis of existing applications with similar functions;
3. Define requirements and specifications for the developed application;
4. Implement various game modes with the ability to choose the level of difficulty;
5. Create a game prototype

The bachelor's thesis is presented on _77_ pages, it contains _4_ sections, _69_ illustrations, _7_ tables, _18_ sources in the list of references.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

МП – мова програмування

ПЗ – програмне забезпечення

ОС – операційна система

БД – база даних

SA – Shooter-Action

AI – Artificial Intelligence

API – Application programming interface

ВСТУП

Актуальність теми полягає в тому що у сучасному світі, де мобільні технології швидко розвиваються та стають все більш популярними, розробка ігор для мобільних платформ стала однією з найбільш перспективних та прибуткових галузей розвитку ПЗ. Одним з найбільш популярних жанрів ігор є SA, який дозволяє гравцям зануритися у захоплюючі пригоди та боротьбу зі злом. Android-платформа стала однією з найбільш популярних платформ для розробки ігор, завдяки своїй відкритій архітектурі та широким можливостям для розробників. Також на Android-платформі можна знайти безліч різних ігор, які пропонують різні жанри та стилі гри. Однак, на даний момент, на Android-платформі не так багато доступних 3D-ігор жанру SA, особливо тих, які мають високі вимоги до графіки та інтерактивності. Це створює великий потенціал для розробки нових та цікавих 3D-ігор жанру SA для Android-платформи. Такі ігри можуть мати великий успіх серед гравців, оскільки вони забезпечують більш глибокий та реалістичний досвід гри, що включає в себе високоякісну графіку, відмінний звуковий супровід та більш реалістичну фізику поведінки об'єктів в грі. Крім того, такі ігри можуть бути корисні для розвитку навичок координації та реакції, що є важливим для людей різного віку та професій. Таким чином, розробка Android 3D-ігор жанру SA має велику актуальність, оскільки це дозволить розширити ринок ігор на Android-платформі та задовольнити попит на цікаві та захоплюючі ігри з високоякісною графікою та інтерактивністю. Крім того, розробка таких ігор може стати важливим елементом розвитку галузі мобільних ігор, що в свою чергу дозволить залучати більше інвестицій та прискорити розвиток цієї галузі.

Об'єкт роботи: дослідження ігрового процесу у Android-іграх.

Предмет роботи: технології та алгоритми, що використовуються для реалізації ігрового процесу в сучасних проєктах, які призначені для мобільних пристроїв з ОС Android.

Мета: вдосконалення навиків у напрямку ігрової індустрії за рахунок розробки гри жанру Shooter-Action, яка демонструє можливості та переваги використання платформи Unity.

Для досягнення поставленої цілі необхідно виконати наступні **завдання:**

- аналіз актуальності сфери;
- огляд аналогів ігор за обраним жанром;
- визначення функціоналу та користувачів гри;
- дослідження основних функцій платформи Unity;
- оформлення специфікацій вимог до ПЗ;
- створення загального алгоритму розробки гри.

Сфера застосування: Android SA забезпечують користувачеві можливість відволіктися від повсякденних турбот і підвищити свій настрій завдяки динамічним ігровим сценам та адреналіновим відчуттям від гри. Оскільки цей жанр ігор є дуже популярним серед підлітків та молоді, розробники часто ставлять перед собою завдання створити гру з якісним інтерфейсом та захопливим сенсом, що збільшує конкуренцію на ринку геймінгу. Також, Android-ігри жанру SA можуть мати певний навчальний аспект, якщо вони містять в собі елементи стратегії та логіки, що сприяє розвитку певних навичок і вмінь.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність предметної сфери

Мобільні 3D-ігри на платформі Unity мають велику актуальність в сучасному світі, особливо в контексті зростаючої популярності мобільних пристроїв. За останні кілька років кількість користувачів мобільних пристроїв значно зросла, а це спричинило значне збільшення попиту на якісні мобільні ігри.

Unity - це одна з найпопулярніших платформ для створення мобільних ігор, яка дозволяє створювати як 2D, так і 3D-гри. Для розробки мобільних 3D-ігор на Unity можна використовувати різні МП, такі як C# або JavaScript, що забезпечує більшу гнучкість в розробці [1].

Також, мобільні 3D-ігри на платформі Unity дозволяють розширювати межі традиційних ігор, дозволяючи користувачам більш вільно взаємодіяти з ігровим світом і отримувати набагато більш реалістичний досвід гри. Крім того, мобільні 3D-ігри на Unity можуть бути створені для різних платформ, таких як iOS або Android, що розширює аудиторію користувачів. Також, платформа Unity дозволяє взаємодіяти з іншими платформами, такими як Oculus або Steam, що дозволяє користувачам грати в одну гру на різних пристроях.

У світі ігрової індустрії, мобільні 3D-ігри на платформі Unity стали особливо популярними в останні роки. Це забезпечує зростання конкуренції в цій сфері, що стимулює розробників створювати якісніші та більш оригінальні ігри [1].

Платформа Unity дозволяє створювати багатофункціональні ігри з різноманітними механіками гри, що дозволяє розробникам створювати оригінальні та унікальні проекти. Крім того, платформа дозволяє розробникам легко підтримувати та оновлювати ігри, що забезпечує більшу гнучкість в розробці.

Також, зростання популярності віртуальної реальності, дозволяє розробникам створювати ще більш інноваційні та захоплюючі ігри з використанням платформи Unity. Розробка мобільних 3D-ігор на Unity також є дуже перспективною галуззю для маркетингу та реклами, оскільки такі ігри дозволяють більш ефективно залучати та зберігати увагу користувачів [1].

За останні кілька років мобільні ігри стали одними з найбільш прибуткових галузей розважальної індустрії. Наприклад, у 2020 році глобальні прибутки від мобільних ігор склали понад 77 мільярдів доларів, що становить більше половини всіх прибутків розважальної індустрії. І зростання цього сектору прогнозується і надалі [3].

Розробка мобільних 3D-ігор на платформі Unity є важливою галуззю для створення ігор, які мають високу якість графіки, звуку та функціональності. Більшість сучасних мобільних пристроїв мають потужні процесори та графічні картки, що дозволяє розробникам використовувати складну 3D-графіку, реалістичні ефекти та фізику, яка дозволяє створювати захоплюючі та реалістичні ігри [3].

Крім того, розробка мобільних 3D-ігор на платформі Unity дозволяє створювати гнучкі та прості інтерфейси, що робить ігри більш доступними для користувачів. Завдяки цьому розробники можуть створювати ігри для різних категорій користувачів – від дітей до дорослих, що дозволяє збільшити аудиторію та зберегти її увагу на довгий час.

Unity дозволяє розробникам легко інтегрувати різні сервіси та мережі, що підвищує ефективність рекламних кампаній та монетизації ігор. Реклама у мобільних іграх є дуже ефективним інструментом просування товарів та послуг, що дозволяє залучати нових користувачів і отримувати додатковий дохід [3].

У рекомендаціях Google з архітектури надаються сучасні методи для створення програм для Android, які дозволяють створювати високопродуктивні та стабільні програми. Компоненти архітектури Android є

чудовою основою для розробки більш надійних програм, зокрема з використанням сучасних компонентів архітектури для корпоративних мобільних застосунків, що дозволяють зменшити кількість шаблонного коду та більше уваги приділяти розробці самого програмного коду. Використання архітектури застосунків Android допомагає спільноті розробників створювати стабільні програми та спрощує шаблон програмування, такий як MVC, MVP або MVVM [2].

Для розробки корпоративних мобільних застосунків використовуються два компоненти: Enterprise Mobile Management (EMM) та Application Performance Management (APM). Вони дозволяють мінімізувати повільність мобільного застосунку та перетворити його на засіб перевірки якості ПЗ, що допомагає забезпечувати безпеку мобільних застосунків та ефективно обмінюватись даними через мобільні гаджети [2].

Застосування AI та машинного навчання в розробці мобільних додатків дозволяє значно покращити їх функціональність. Наприклад, додатки з підтримкою AI та ML дозволяють спрощувати повсякденну роботу користувачів та надавати більш точні пропозиції. Netflix є найкращим прикладом такої платформи, яка аналізує, які шоу переглядають користувачі, і рекомендує контент відповідно до їх вподобань. Декілька API-інтерфейсів можуть бути інтегровані з такими сервісами, як Firebase або Amazon Web Services, що дозволяє розробникам Android легко підключатись до хмарних рішень та використовувати їх у своїх додатках. Крім того, використання хмарних сервісів може значно знизити вартість розробки та забезпечити більшу масштабованість та доступність додатків [2].

Хмарні додатки для Android дозволяють користувачам зберігати та синхронізувати дані між різними пристроями, що робить їх більш доступними та зручними у використанні. Також вони можуть зберігати більш об'ємні дані, такі як відео та фотографії, що раніше могли займати багато місця на телефоні.

Загалом, хмарні додатки для Android дозволяють користувачам зберігати та отримувати доступ до даних більш ефективно та безпечно [4].

Нарешті, мобільні 3D-ігри на платформі Unity є важливим інструментом для розвитку технологічної галузі. Розробка таких ігор вимагає високої компетенції та знань у сфері програмування та дизайну. Відомі розробники 3D-ігор на платформі Unity є ключовими гравцями на ринку технологій, що дозволяє створювати нові можливості для відомих компаній та початківців у цій галузі.

Отже, мобільні 3D-ігри на платформі Unity є дуже актуальною та перспективною галуззю в сфері розваг та технологій. Розробка таких ігор дозволяє створювати високоякісні ігри з реалістичною графікою та фізикою, що залучає користувачів та збільшує їх увагу до гри. Крім того, такі ігри є ефективним інструментом для монетизації та реклами, що дозволяє збільшувати прибуток від ігор [4].

1.2 Огляд аналогів

Zombie Gunship Survival [6]

У грі ви берете на себе роль оператора з авіабази, який управляє зомбі-винищувачем з пушкою, який надає підтримку наземним силам у боротьбі з живими мерцями. Гравці повинні захищати свою базу від нападів зомбі, використовуючи зомбі-винищувач для знищення ворогів. Гра має різні рівні складності та множину зброї, що дає можливість гравцям відчувати виклик та індивідуальний досвід гри. Крім того, у грі є можливість співпраці з іншими гравцями та обміну ресурсами, що дозволяє взаємодіяти з гравцями з усього світу.

Таблиця 1.1 – Опис «Zombie Gunship Survival»

Назва	Zombie Gunship Survival

Виробник	Flaregames
Мова реалізації	C#
Функції	1. Бойові дії в ролі стрілка з літака AC-130. 2. Оновлення зброї. 3. Інтенсивна боротьба з зомбі.
Переваги	1. гра повністю безкоштовна. 2. Розвинена система зброї 3. Соціальний аспект
Недоліки	1. Можливість донату 2. Енергійні обмеження 3. Монотонність
Вебсайт	https://play.google.com/store/apps/details?id=com.flaregames.zgs

Рисунок 1.1 – Обкладинка гри «Zombie Gunship Survival»

Gunship Operator 3D [7]

"Gunship Operator 3D" - це військовий симулятор, який дозволяє гравцю брати під свій контроль бойовий корабель, щоб перемогти ворогів. Гравець може здійснювати місії по всьому світу, забезпечуючи захист своєї країни та наносячи удари по ворожим силам.

Таблиця 1.2 – Опис 3tier application «Clash of Clans»

Назва	**Gunship Operator 3D**
Виробник	VOODOO
Мова реалізації	C#
Функції	1. Виконання місій по всьому світу. 2. Багато різних завдань як для одного користувача; 3. Збирання та додавання до свого арсеналу різноманітної зброї та інструментів. Оператор бойового гелікоптера
Переваги	1. Гра залежить від користувача, від його рішень та вмінь які впливають на результат гри; 2. Різноманітні місії та сценарії.
Недоліки	1. Присутня донатна система. 2. Велика кількість реклами.
Вебсайт	https://play.google.com/store/apps/details?id=com.IronEqual.GunshipOperator3D

Рисунок 1.2 – Обкладинка гри «**Gunship Operator 3D**»

Drone Shadow Strike 3[8]

Це найінтенсивніший безкоштовний симулятор розвідки дронів на мобільних пристроях з розвинутими військовими діями та передовим військовим арсеналом.

Таблиця 1.3 – Опис 3tier application «Drone Shadow Strike 3»

Назва	Drone Shadow Strike 3
Виробник	Tilting Point
Мова реалізації	C#,
Функції	1. Використання дронів у військових операціях; 2. Розвиток та налаштування дронів. 3. Система нагород та досягнень
Переваги	1. Розробники створили гру повністю безкоштовну; 2. Гра не потребує особливих здібностей, все швидко пізнається у процесі гри.

	3. Різноманітні місії
Недоліки	1. Велика кількість реклами, без нагород; 2. Можливість донату.
Вебсайт	https://play.google.com/store/apps/details?id=com.reliancegames.drones3

Рисунок 1.3 – Обкладинка гри «Drone Shadow Strike 3»

Проведений аналіз аналогів ігор жанру SA дозволив зрозуміти, що цей жанр має велику популярність серед гравців і відомий своїми насиченими бойовими сценами та швидким геймплеєм. Багато ігор жанру мають широкі можливості настройки та багато позитивних відгуків від гравців. Аналіз також показав, що успішні ігри жанру часто мають добре продуманий інтерфейс та ігрову механіку, які сприяють захопленню гравця від перших хвилин гри. Загалом, аналіз підтверджує важливість ретельного проектування гри жанру SA з урахуванням основних елементів успішних ігор цього жанру.

1.3 Визначення технічного завдання для ігрового застосунку

Технічне завдання полягає в розробці та впровадженні унікального ігрового застосунку, який забезпечить довгий геймплей для користувача. Застосунок має задовольняти такі вимоги [4]:

1. При запуску ігрової програми повинно бути доступне меню, де користувач може готуватись до виконання завдань.
2. Користувач повинен мати можливість переглядати інформацію про кількість ресурсів та їх зміни від часу його відсутності.
3. Всі функціональні кнопки для взаємодії з інтерфейсом користувача повинні розміщуватись в нижній частині екрана, щоб забезпечити зручність використання на мобільних пристроях в горизонтальній орієнтації.
4. Основне ігрове поле має бути реалізовано в перспективній проекції.
5. Програма повинна працювати на Android-ОС з версією не нижче 4.3.

Насамперед необхідно вибрати платформу для розробки гри, яка задовольнятиме наступні вимоги:

1) можливість розробки для мобільних пристроїв;
2) можливість розробки 3D ігор;
3) зручність використання;
4) наявність якісної документації;
5) безкоштовне розповсюдження.

Unity є однією з найпопулярніших платформ для розробки ігор на сьогоднішній день, і її обрано було з кількох причин, які відповідають вище наведеним вимогам [1]:

1. Можливість розробки для мобільних пристроїв: Unity підтримує розробку для різних мобільних платформ, таких як iOS та Android. Це означає, що розробник може створювати ігри, які працюють на різних пристроях, включаючи смартфони та планшети.

2. Можливість розробки 3D ігор: Unity є потужним інструментом для створення 3D-ігор. Вона надає широкий спектр інструментів для моделювання, анімації, освітлення та інших аспектів 3D-розробки.

3. Зручність використання: Unity має досить простий та зрозумілий інтерфейс користувача, що дозволяє розробникам зосередитися на самій розробці, а не на вивченні складних інструментів.

4. Наявність якісної документації: Unity має велику кількість документації, включаючи офіційну документацію, курси та ресурси спільноти. Це дозволяє розробникам швидко знайти відповіді на свої запитання та вирішити проблеми.

5. Безкоштовне розповсюдження: Unity надає безкоштовну версію свого продукту, яка має достатні функціональність для розробки більшості ігор. Крім того, з Unity можна збирати ігри для різних платформ без додаткових витрат.

Unity має простий інтерфейс, який складається з таких вікон як: Scene, де можна розглянути ігрове поле з потрібного ракурсу; Inspector, який містить всі властивості виділеного об'єкта та його компоненти; Project, де знаходяться всі матеріали проекту. Крім того, Unity має інтегровані послуги для залучення, утримання та монетизації гравців.

Окрім зручного інтерфейсу, Unity також має потужний редактор, що дозволяє легко створювати складні 3D моделі та анімації. Крім того, для розробки ігор у Unity використовують МП C#, яка є досить простою для вивчення та має велику кількість ресурсів для самостійного навчання [1].

Unity також дозволяє легко експортувати гру на різні платформи, такі як iOS, Android, Windows та Mac OS. Більшість налаштувань можна здійснювати безпосередньо в редакторі Unity, що дозволяє зекономити час на розробку та тестування.

1.4 Специфікація вимог

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ [5]

Доступність

Гра доступна для будь-якого користувача з апаратним інтерфейсом та доступом до Інтернету.

Переносимість

ПЗ працює на ОС Android версії 4.3 та вище.

Продуктивність

Продуктивність роботи ПЗ залежить від швидкості підключення до мережі Інтернет.

Надійність

Дані, які користувачі надають під час реєстрації, є приватними і кожен користувач має свій власний ігровий процес.

Безпека

Кожен користувач авторизується через свій аккаунт.

ПРИЗНАЧЕННЯ ЗАСТОСУНКУ

Призначення застосунку, для якої розробляється програмне забезпечення

ПЗ призначене для використання користувачами у сфері розваг та дозвілля.

Погодження, що ухвалені в програмній документації

Для створення загального ПЗ та забезпечення його злагодженої роботи використовуються допоміжні ассети та бібліотеки Unity.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування

Застосунок призначений для використання користувачами для проведення дозвілля та відпочинку.

Характеристики користувачів

Основні характеристики користувачів: наявність телефону та доступу до мережі Інтернет.

Загальна структура і склад системи

Основними частинами ПЗ є сервер та БД.

Загальні обмеження

Єдиним обмеженням є наявність доступу до мережі Інтернет.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Джерела і зміст вхідної інформації (даних)

У цьому ПЗ основним джерелом вхідної інформації є користувач, який самостійно вводить дані, такі як досягнення та ім'я персонажу.

Вимоги до способів організації, збереження та ведення інформації

Щодо способів організації, збереження та ведення інформації, використовується реляційна БД SQLite. Ця БД має багато переваг, включаючи мультиплатформність та здатність забезпечувати компактність і швидку взаємодію з даними.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вимоги до ПЗ можуть бути визначені через ряд характеристик, які відображають якість та надійність роботи програми.

Коректність є однією з таких характеристик та визначається як здатність програми до безпомилкової реалізації необхідного алгоритму. При цьому, коректність може бути описана як властивість програми, яка свідчить про відсутність помилок, допущених розробником на різних етапах проектування - від специфікації та проектування алгоритму до структур даних та кодування.

Стійкість є ще однією важливою характеристикою ПЗ та визначається як здатність програмного засобу здійснювати необхідні перетворення інформації зберігаючи вихідні рішення у межах встановлених специфікацій.

Відновлюваність є ще однією характеристикою ПЗ та характеризує можливість програмного засобу адаптуватися до виявлення помилок та їх усунення.

Надійність, у свою чергу, може бути представлена як сукупність різних характеристик, таких як *цілісність* програмного засобу, його *живучість*, *завершеність* та *працездатність*. Цілісність описує здатність ПЗ захищатися від відмов та зберігати свою працездатність. Живучість відображає здатність програми до контролю вхідних даних та їх перевірки під час роботи. Завершеність описує бездефектність готового програмного засобу та якість його тестування. Працездатність відображає здатність програмного засобу відновлювати свої можливості після збоїв або непередбачуваних ситуацій.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

1. Операційна система: Android версії 4.3 та вище.
2. Процесор: ARMv7 або вище.
3. Оперативна пам'ять: 1 ГБ RAM.
4. Вільне місце: 200 МБ.
5. Екран: роздільна здатність 960x540 пікселів або вище.

1.5 Опис загального алгоритму виконання проекту

Для того щоб розробити гру, важливо спланувати роботу та визначити часові рамки. Зазвичай тривалість процесу розробки гри залежить від складності самого проекту. Розробка гри є складним процесом, що складається з кількох етапів. На кожному з етапів розробляється певний аспект гри, який потім буде інтегрований в кінцевий продукт [9].

Першим етапом є проєктування меню гри. Меню є одним з найважливіших аспектів гри, оскільки воно є головним інтерфейсом, через який користувач буде взаємодіяти з грою. На цьому етапі розроблюється дизайн інтерфейсу гри, обрано колірну палітру, функціональність меню, його структура та навігація.

Другим етапом є проєктування рівнів. Рівні гри є тим, чим гравець буде займатись протягом гри. На цьому етапі розроблюються загальні концепції рівнів, їх дизайн, рівень складності та головні завдання.

Третім етапом є моделювання. На цьому етапі розробляється 3D-моделювання персонажів, об'єктів, теренів та інших елементів гри. Для цього використовуються спеціальні програми, такі як Blender, Maya або 3DS Max.

Четвертим етапом є розробка. На цьому етапі розробляється ПЗ гри, яке буде реалізовувати всі функції, відповідати за геймплей та інші аспекти гри. Розробники використовують спеціальні програмні платформи, такі як Unity, Unreal Engine або GameMaker Studio [9].

Останнім етапом є тестування. На цьому етапі гра проходить ряд тестів, щоб перевірити її стабільність, функціональність та геймплей. Цей етап дуже важливий, оскільки він допомагає виявити та виправити помилки, баги та інші недоліки в грі перед тим, як вона буде випущена для широкого загалу.

На основі питань які виникають під час підготовки до виробництва створюється GDD (документ ігрового дизайну). Цей документ включає в себе такі елементи, як жанр, ідею, сюжет, персонажів, рівні та складність, геймплей. Цей список може бути доповнюваний протягом розробки гри. Все це необхідно для того, щоб у процесі розробки було менше питань та уникнути помилок при створенні проєкту. Крім того, це дозволить отримати повноцінний проєкт, який потребуватиме менше виправлень у кінці роботи. Проте, гра буде випущена у бета-версії, щоб змогти виправити деякі недоліки, які можуть бути пропущені розробниками під час створення проєкту [9].

Висновки до розділу 1

Майбутнє інформаційних технологій належить мобільним застосункам, які продовжують розвиватись з кожним роком. Сучасні мобільні пристрої можуть надати потужність та обчислювальну потужність, що часто перевищують можливості стаціонарних ПК. Це робить їх досить конкурентноздатними відносно стаціонарних комп'ютерів, особливо у випадку геймінгу. Ігри в жанрі SA мають багато аналогів, тому вирішення, що дозволило б відрізнити їх від конкурентів, стало важливою задачею. Однією з основних переваг цього застосунку стала можливість грати на різних платформах. Проте, він також має деякі недоліки, зокрема, AI, локалізація та кількість рекламних повідомлень, які потрібно враховувати для того щоб не втратити цільову аудиторію.

2 АНАЛІЗ ФРЕЙМВОРКІВ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

2.1 Вибір засобів розробки ігрового застосунку Unity

Unity - це інтегроване середовище розробки ігор, що дозволяє створювати ігри для різних платформ, таких як Windows, macOS, Linux, Android, iOS, Xbox, PlayStation та інші. Unity дозволяє створювати ігри за допомогою візуального редагування об'єктів та скриптів, що дозволяє програмістам та дизайнерам працювати в одному середовищі [9]. Основою Unity є двигун Unity, що дозволяє відтворювати графічні об'єкти, взаємодіяти з ними та створювати різні ефекти. Unity також має широкий вибір інструментів та компонентів, які дозволяють створювати ігри без додаткового програмування (рис. 2.1-2.2).

Рисунок 2.1 – Логотип «Unity»

Unity також підтримує різні мови програмування, такі як C#, JavaScript, Boo, що дозволяє розробникам вибирати ту мову програмування, з якою вони знаходяться у зручності. Крім того, Unity підтримує створення VR та AR додатків, що робить його популярним серед розробників [3].

Unity є досить популярним середовищем розробки ігор та додатків, особливо в світі мобільних ігор. Він дозволяє швидко створювати гру без значної витрати часу та зусиль, що робить його доступним для початківців та досвідчених розробників.

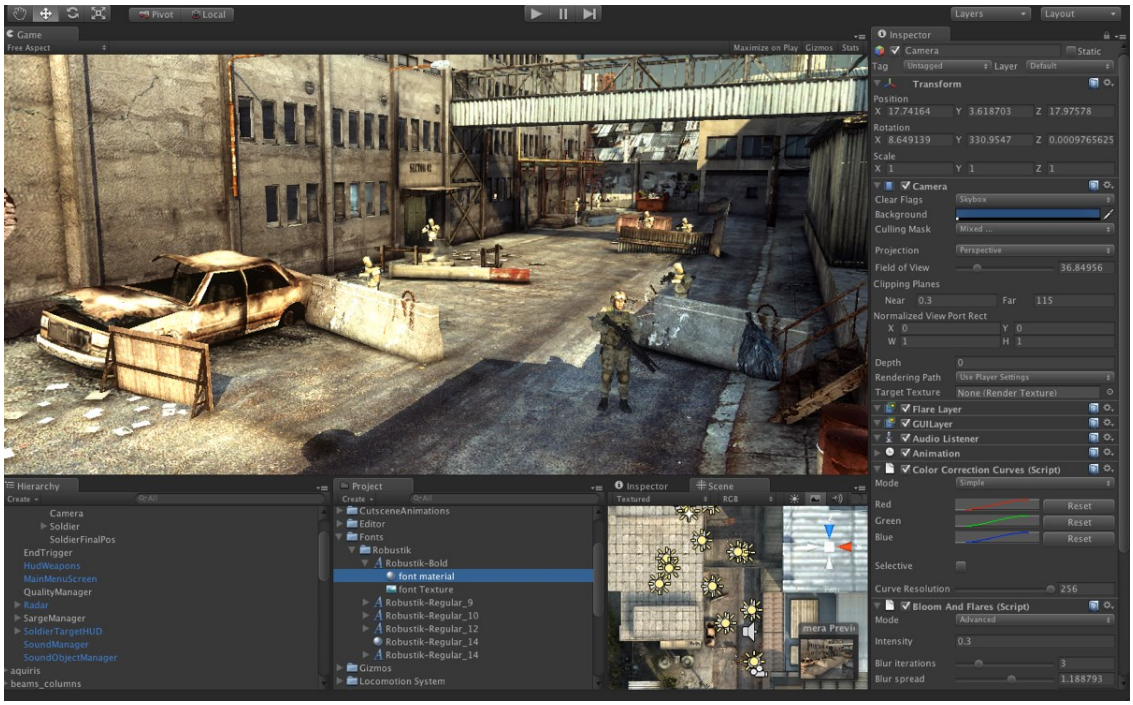

Рисунок 2.2 – Інтерфейс платформи «Unity»

Unreal Engine

Unreal Engine - це платформа для розробки ігор та візуалізації, створена компанією Epic Games. Вперше вона була представлена у 1998 році як рушій для першої особи шутерів, а зараз Unreal Engine використовують для створення ігор, візуалізації архітектурних та промислових проектів, а також в кіно та телебаченні [12].

Unreal Engine пропонує велику кількість інструментів для розробки ігор, таких як готові компоненти фізики, освітлення, анімації, штучного інтелекту та інші. Всі ці інструменти допомагають розробникам створювати різноманітні ігри від першої і третьої особи, ігри з відкритим світом, ігри з підтримкою віртуальної та доповненої реальності, та інші [12].

Рисунок 2.3 – Логотип «Unreal Engine»

Unreal Engine базується на мові програмування C++, але в ньому також можна використовувати скриптові мови, такі як Blueprint (подібний до візуальної мови програмування), Python та Lua.

Крім того, Unreal Engine має свій власний магазин активів (Unreal Marketplace), де можна придбати готові моделі, текстури, звуки та інші ресурси для створення ігор. Також в платформі є можливість робити рендеринг у режимі реального часу (real-time rendering), що дозволяє отримувати дуже високоякісні візуальні ефекти [12].

Рисунок 2.4 – Інтерфейс платформи «Unreal Engine»

CryEngine

CryEngine - це ігровий двигун, створений компанією Crytek. Він використовується для розробки відеоігор та візуалізації архітектурних та механічних об'єктів. CryEngine має вражаючі можливості, такі як високоякісна рендерингова система, фізична система та підтримка VR. Він також має потужні інструменти для розробки ігрового світу, такі як редагувальний режим з візуальним програмуванням та підтримка штучного інтелекту [13].

CryEngine підтримує кросплатформність, що дозволяє розробникам створювати ігри для різних платформ, включаючи PC, консолі та мобільні пристрої.

Рисунок 2.5 – Логотип «CryEngine»

Однак, в порівнянні з іншими ігровими двигунами, CryEngine може бути складним у використанні та має високу вимогу до апаратного забезпечення. Крім того, підтримка іншими користувачами та розвиток спільноти може бути обмеженим [13].

У загальному, CryEngine є потужним інструментом для розробки високоякісних відеоігор та візуалізації, якщо вам доступні необхідні ресурси та навички для роботи з цим двигуном.

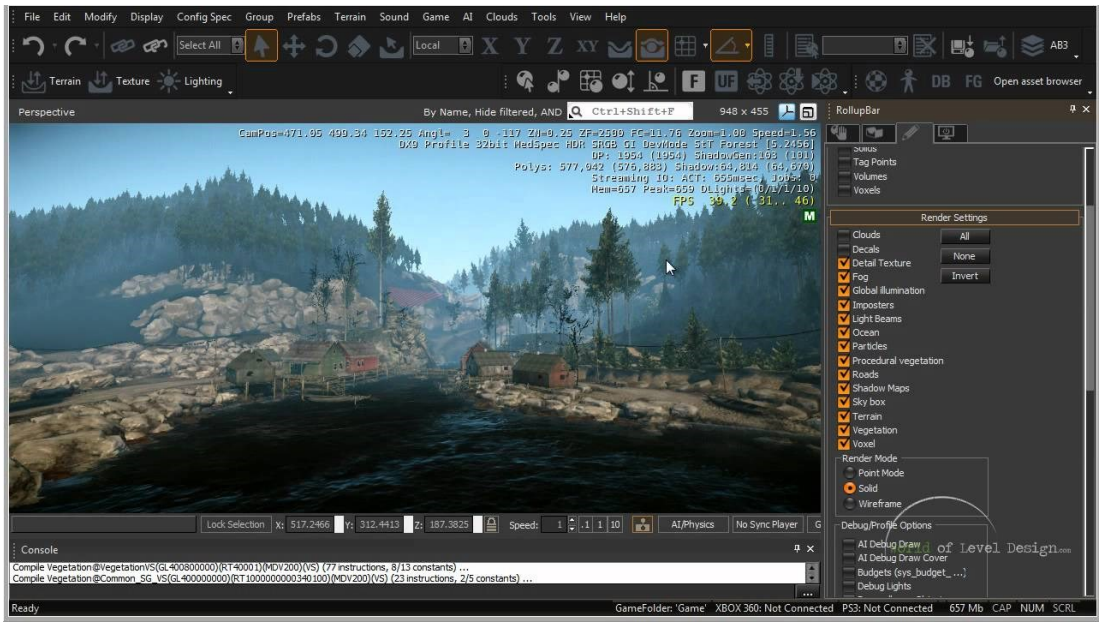

Рисунок 2.6 – Інтерфейс платформи «CryEngine»

Таблиця переваг та недоліків для порівняння платформ розробки (табл.2.1-2.3): Unity, Unreal Engine, CryEngine.

Таблиця 2.1 – Переваги та недоліки Unity [9]

№	**Unity**	
	Переваги	*Недоліки*
1	Широке застосування і підтримка різних платформ, включаючи мобільні пристрої	Обмежена можливість контролювання рендеринга та обробки даних
2	Велика спільнота користувачів та безкоштовні ресурси, що допомагають в навчанні та розробці проектів	Іноді можуть виникати проблеми з оптимізацією та продуктивністю

3	Простий та інтуїтивний інтерфейс	Обмежені можливості для розробки графіки, залежно від версії
4	Безкоштовний	

Таблиця 2.2 – Переваги та недоліки Unreal Engine[12]

№	Unreal Engine	
	Переваги	*Недоліки*
1	Висока якість графіки та візуальних ефектів	Висока вартість підписки на платформу
2	Широкі можливості для розробки ігор, включаючи VR та AR проекти	Складніше вивчення та розробка порівняно з Unity
3	Є можливість використовувати скриптові мови, такі як C++, Python, а також власні мови програмування	Через свої потужні особливості до графіки, має великі вимоги до обладнання
		Потребує великого досвіду у розумінні МП C++

Таблиця 2.3 – Переваги та недоліки CryEngine [13]

№	CryEngine	
	Переваги	*Недоліки*
1	CryEngine має вражаючу графіку, яка дозволяє створювати дуже деталізовані ігрові світи	Для роботи з CryEngine потрібна потужна комп'ютерна система, що може бути проблемою для деяких користувачів
2	CryEngine надає готові рішення для створення ігрових механік, таких як штучний інтелект та розумне освітлення	CryEngine підтримує лише деякі платформи, такі як Windows та Xbox, що може бути обмеженням для деяких розробників
3	CryEngine має досить зручний та потужний інтерфейс для розробників, що дозволяє швидко створювати та редагувати ігрові об'єкти	У порівнянні з Unity та Unreal Engine, спільнота CryEngine є досить маленькою, що може ускладнити пошук допомоги
4	Може працювати з дуже великими ігровими світами, що дозволяє створювати масштабні проекти	CryEngine вимагає від розробника високого рівня знань та навичок в програмуванні та редагуванні ігрових об'єктів

На основі порівняння різних ігрових движків та їхніх переваг і недоліків було обрано Unity як ігровий движок для створення 3D ігор. Основні переваги Unity полягають у простоті використання, доступності готових компонентів та ресурсів, а також підтримці різних платформ, включаючи мобільні пристрої. Крім того, Unity має велику спільноту користувачів та розвинену документацію, що дозволяє швидко розв'язувати проблеми. Unity є більш підходящим ігровим движком для початківців та середніх розробників, які шукають прості та доступні рішення для створення 3D ігор.

2.2 Вибір мови програмування

У сфері розробки мобільних ігор для платформи Android, зазвичай використовуються три мови програмування: C++, C#, та Java. Кожна з цих мов має свої переваги та недоліки, тому вибір конкретної мови залежить від потреб проекту та вмінь розробника. Наприклад, C++ часто використовують для створення графічно вимогливих ігор з високими вимогами до продуктивності. C# забезпечує зручний інтерфейс для створення гри, а також може бути використана для розробки ігор для різних платформ, включаючи Windows, Xbox та PlayStation [16]. Java часто використовують для створення мобільних ігор, оскільки вона є мультиплатформною та може запускатися на різних операційних системах без необхідності відробки коду для кожної платформи окремо.

Вибір мови програмування для розробки ігор залежить від різних факторів, таких як потреби проекту, особисті навички та знання розробника, наявність інструментів та ресурсів для розробки. Для розробки гри було обрано мову програмування C# [10].

Однією з особливостей C# є те, що вона є однією з найпопулярніших мов програмування для розробки ігор. Вона пропонує зручну розробку та використання графічного інтерфейсу, який допомагає швидко створювати ігри з гарними візуальними ефектами та анімацією. Крім того, мова C# є частиною

платформи .NET, яка пропонує широкий набір бібліотек та інструментів для розробки програмного забезпечення, що полегшує процес розробки ігор [10].

Іншою особливістю C# є те, що вона підтримує багатопоточність, що може допомогти забезпечити високу продуктивність ігри. Це важливо, особливо для ігор з великою кількістю об'єктів та складних алгоритмів, де розпаралелювання обчислень може допомогти підвищити швидкість гри.

Крім того, мова C# підтримує патерн проектування Model-View-Controller (MVC), що дозволяє легко розділити компоненти гри та забезпечити їх незалежність. Це зменшує ризик помилок та спрощує процес розробки [10].

Також, C# має високу рівень безпеки та стійкості, оскільки вона базується на принципах статичної типізації та контролю пам'яті. Це дозволяє попереджати баги та помилки на етапі компіляції, а також робить мову менш вразливою до вразливостей, що можуть бути використані для атак на програми.

Таким чином, C# є потужним та універсальним інструментом для розробки мобільних додатків, зокрема, ігор. Її особливості, такі як високий рівень безпеки, наявність інструментів для швидкої розробки, та можливість створення кросплатформних додатків, роблять її досить популярною серед розробників [11].

Обрано Visual Studio як середовище програмування для застосунку з кількома причинами. По-перше, воно надає багато корисних можливостей, таких як додавання проєкту, редагування коду та створення класів поведінки. Крім того, воно є інтегрованим середовищем розробки для Unity, що дозволяє легко і швидко експортувати код з Unity до Visual Studio та знову імпортувати назад зміни з Visual Studio до Unity. Також, середовище дозволяє легко визначити метод, який буде використовуватись, і створювати нові шейдери [11].

Visual Studio працює з тією ж системою проєкту, що й Unity, тому будь-які зміни, внесені до коду в Visual Studio, автоматично синхронізуються з

Unity. Це значно полегшує розробку та забезпечує більшу точність та швидкість в процесі розробки. Крім того, Visual Studio надає багато інструментів тестування та налагодження, що дозволяє швидко виявляти та виправляти помилки під час розробки [11].

2.3 Вибір програми для моделювання

У розробці мобільних ігор велике значення має візуальна складова, що включає моделі та дизайн. Все, що бачить користувач, є фантазією художників та розробників. При створенні мобільної гри потрібно приділити достатньо уваги дизайну, оскільки він є тим фактором, що відрізняє мобільні ігри однакового жанру. Для порівняння було обрано такі програми, як: Cinema 4D та Blender.

Рисунок 2.7 – Логотип «Cinema 4D»

Cinema 4D – це програма для 3D-моделювання, анімації та візуалізації, розроблена компанією Maxon. Вона використовується для створення різноманітних візуальних ефектів, анімаційних фільмів, комп'ютерних ігор, рекламних роликів та інших видів мультимедійного контенту [14].

Однією з ключових особливостей Cinema 4D є її інтуїтивно зрозумілий і легкий для вивчення інтерфейс. Програма підтримує широкий спектр

форматів файлів, що дозволяє працювати з іншими програмами для 3D-моделювання та візуалізації. Cinema 4D також має вбудовану бібліотеку моделей та матеріалів, що спрощує процес створення реалістичних сцен [14].

Програма має широкі можливості для роботи з анімацією, зокрема, ієрархічну структуру об'єктів, скелетний аніматор, динаміку та інші інструменти для контролю над рухом та поведінкою об'єктів в сцені. Крім того, Cinema 4D підтримує створення 3D-тексту, включаючи мультитекст, інструменти для розтягування та викривлення тексту, що дозволяє створювати різноманітні заголовки та титри [14].

Ще одна важлива особливість Cinema 4D - це її рендеринг. Програма має вбудований двигун рендерингу, що дозволяє створювати візуально привабливі та реалістичні зображення. Крім того, Cinema 4D підтримує зовнішні рендерери, такі як Arnold, Octane та Redshift, що дозволяє працювати з різними технологіями рендерингу.

Одна з найбільш відомих особливостей Cinema 4D - це MoGraph, що є інструментом для створення складних анімацій, включаючи анімаційні графіки та візуалізації даних. За допомогою MoGraph можна створювати складні анімаційні ефекти, які можуть бути динамічними та змінюватися з часом [14].

Програма також має інтеграцію з After Effects та Adobe Illustrator, що дозволяє створювати комплексні анімаційні проекти, включаючи візуальні ефекти, рекламні ролики та інші види контенту. Крім того, Cinema 4D має можливість імпорту та експорту моделей у різноманітних форматах, що дозволяє працювати з іншими програмами для 3D-моделювання та візуалізації.

Рисунок 2.8 – Логотип «Blender»

Blender - це безкоштовна програма для 3D-моделювання, анімації та візуалізації. Вона розробляється та підтримується відкритою спільнотою, що забезпечує її постійне оновлення та розвиток [15].

Одна з головних особливостей Blender – це його інтуїтивний та ергономічний інтерфейс, що дозволяє простіше працювати з програмою. Blender також має розширення Python, яке дозволяє розробникам створювати власні інструменти та скрипти для автоматизації роботи з програмою.

Blender має вбудований набір інструментів для моделювання, включаючи стандартні інструменти для створення форм, такі як примітиви та меші. Крім того, в Blender є можливість створювати анімацію, використовуючи ключові кадри та криві руху, а також можливість візуалізувати дані за допомогою діаграм та графіків [15].

Blender також має багато інструментів для текстурування та освітлення, що дозволяє створювати складні візуальні ефекти. У Blender також є можливість використовувати фізичну симуляцію, таку як симуляція рідини, твердих тіл та м'яких тіл.

Крім того, Blender має вбудовану підтримку віртуальної реальності (VR), що дозволяє створювати вражаючі віртуальні досвіди для VR-гарнітур. Blender також має широку підтримку форматів файлів, що дозволяє взаємодіяти з іншими програмами для 3D-моделювання та візуалізації [15].

Отже, на підставі порівняння Blender і Cinema 4D, можна зробити висновок, що Blender є більш привабливим вибором для студій та індивідуальних розробників, які шукають безкоштовний та потужний 3D редактор з великою спільнотою та гнучкими можливостями розробки. Blender є більш універсальним у використанні, оскільки підтримує різні платформи та має можливість експортувати моделі у різні формати. Також Blender має відкритий вихідний код, що дозволяє розробникам створювати свої плагіни та інструменти. Крім того, Unity відносно недавно придбала компанію Blender, що може вказувати на те, що Blender буде більш інтегрованим та сумісним з Unity у майбутньому. Тому, при виборі програми для моделювання було обрано програму Blender [17].

Висновки до розділу 2

Під час роботи над розділом 2 було опрацьовано матеріал стосовно різновидів МП, проаналізовано різниці між еталонами сучасних рушіїв розробки та детально розглянули сучасні програми для моделювання 3д-простору та моделей для проекту.

Було обрано Unity як ігровий движок для створення 3D-гри. Основні переваги Unity полягають у простоті використання, доступності готових компонентів та ресурсів, а також підтримці різних платформ, включаючи мобільні пристрої.

Для розробки гри було обрано мову програмування C#. Однією з особливостей C# є те, що вона є однією з найпопулярніших мов програмування для розробки ігор. Вона пропонує зручну розробку та використання графічного інтерфейсу, який допомагає швидко створювати ігри з гарними візуальними ефектами та анімацією.

При виборі програми для моделювання було обрано програму Blender. Blender є більш привабливим вибором для студій та індивідуальних розробників, які шукають безкоштовний та потужний 3D редактор з великою спільнотою та гнучкими можливостями розробки.

3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

У проєктуванні обрано мову графічного опису для моделювання об’єктів UML. UML-діаграми відіграють важливу роль у створенні проєкту та розробці програмного забезпечення. Мова UML допомогає знайти помилки в структурі програми. Це дуже розповсюджена мова, що вважається найкращою для розробників, власників компаній, підприємців із різних галузей, фахівців та менеджерів проєктів. За допомогою UML-моделей можливо зробити генерацію коду, але UML не є мовою програмування. Для побудови діаграм було використано StarUML.

3.1 Написання usecase

Короткий usecase

Користувач повинен захищати свою базу від ворогів які нападають на неї. Для отримання інших завдань необхідно завершити попередні. Бонуси видаються за часте находження в грі, іх кількість збільшуєься від щоденного відвідування гри. Коричтувач може виконувати бонусні завдання, які появляються час від часу.

Поверхневий usecase

Користувач повинен захищати свою базу від ворогів які нападають на неї. Для отримання інших завдань необхідно завершити попередні. Бонуси видаються за часте находження в грі, іх кількість збільшуєься від проведеного часу в грі. Коричтувач може виконувати бонусні завдання, які появляються час від часу.

Альтернативний сценарій:

1) користувач не встиг знищити всіх ворогів та програв рівень;
2) користувач почав проходження сильного рівня не пройшовши попередні, як наслідок недостатність параметрів для знищення сильних ворогів;

3) користувач не заходив в гру, через що прогресс бонусу повертається до початкового;
4) користувач не встиг виконати тимчасове бонусне завдання, через що втратив можливість отримати додаткові нагороди.

Таблиця 3.1 – Повний usecase

Primary Actor	Користувач
Scope	Android-гра у жанрі SA на базі Unity
Level	Мета користувача
Preconditions	Користувач авторизований в застосунку
Stakeholders and interests	1. програміст: зацікавлений у розробці своєї задачі; 2. тестувальник: зацікавлений у знаходженні помилок у грі, програміст: для дороблення; 3. користувач: зацікавлений в здобутті досягнень та розвитку за найкоротший час.
Main Success Scenario: 1. Користувач встановлює гру. 2. Користувач вибирає завдання. 3. Користувач обирає спосіб розвитку у грі 4. Користувач виконує завдання яке він обрав 5. Користувач відмічає прогрес. 6. Користувач отримує нові завдання та відкриває нові досягнення. 7. Тестувальник слідкує за тим щоб не було помилок при отриманні завдань та отриманні нагород за прогресс у грі. 8. Програміст додає нові завдання, івенти та розроблює велику кількість нововведень. 9. Користувач продовжує грати та здобуває нові досягнення.	
Result	Користувач

Кінець таблиці 3.1

Extensions:	
1.	1. Користувач не заходить в гру: 1.1. Користувач втрачає прогресс бонусу. 1.2. a. Користувая втречає здобуття бонусів за відсутність в грі. 1.2. b. Користувач втрачає велику кількість ігрової валюти. 1.3. Користувач втрачає можливість проходження бонусних завдань.
2	2. Тестувльник знаходить помилку у грі: 2.1a. Помилка не глобальна. 2.2a. Програміст її швидко виправляє. 2.1b. Помилка глобальна 2.2b. Потрібно знаходити та вирішувати помилку та зупиняти гру на технічне обслуговування.
Special Requirements:	1) система перекладу тексту на декілька мов (українську, англійську); 2) адаптивний інтерфейс; 3) повне підключення до інтернету.
Frequency of Occurrence	Потрібно постійне підключення до інтернету.

3.2 Створення діаграми використання

Діаграми варіантів використання (Use Case diagrams) описують взаємовідносини та залежності між групами варіантів використання та дійових осіб, що беруть участь у процесі. Важливо розуміти, що діаграми варіантів використання не призначені для відображення проєкту та не можуть описувати внутрішній пристрій системи. Діаграми варіантів використання призначені для спрощення взаємодії з майбутніми користувачами системи [8], з клієнтами, і особливо знадобляться визначення необхідних характеристик системи. Іншими словами, діаграми варіантів використання говорять про те, що система повинна робити, не вказуючи самі методи, що застосовуються (рис. 3.1) [18].

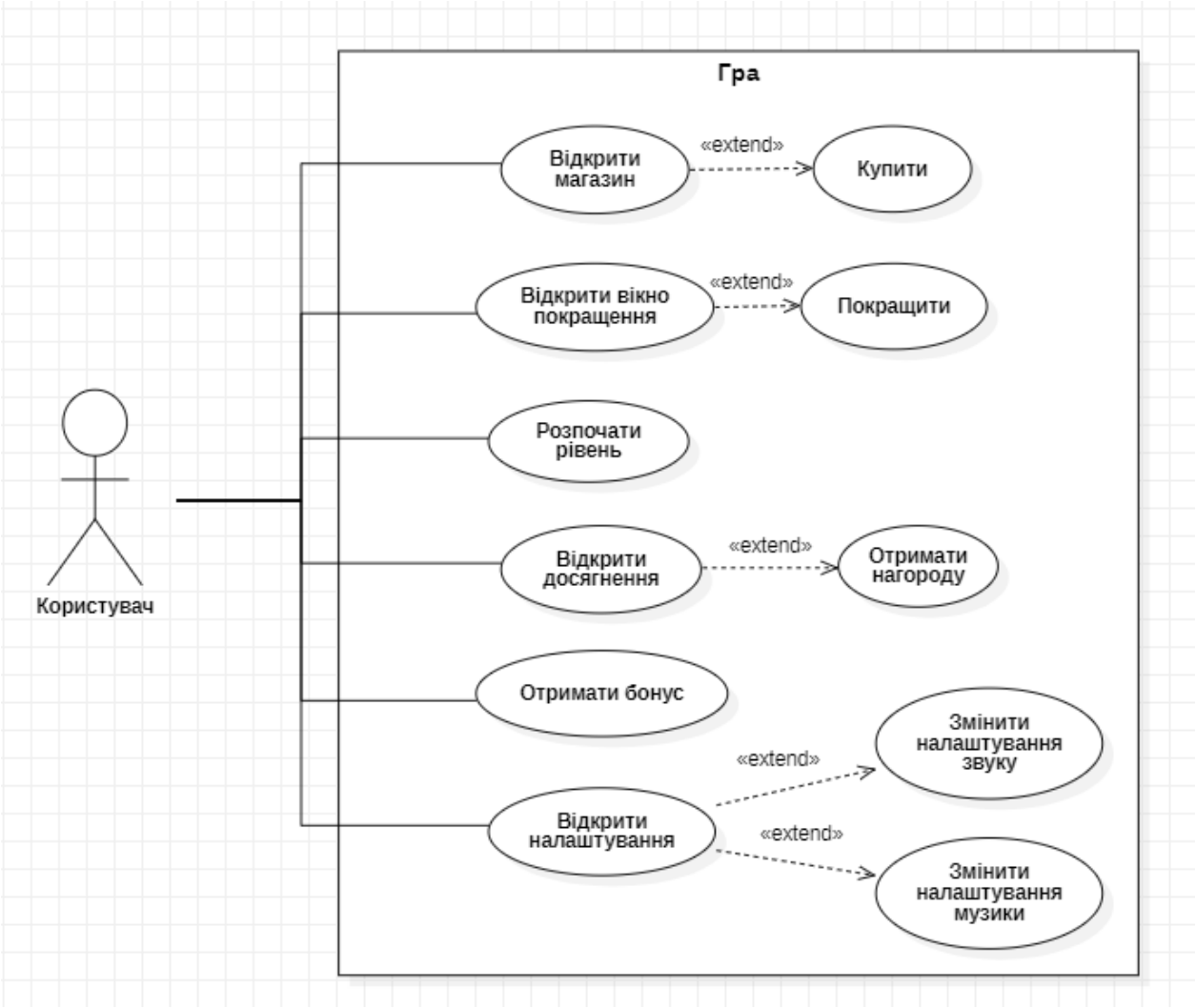

Рисунок 3.1 – Діаграма варіантів використання

Опис елементів моделі на діаграмах варіантів використання:

Актори – це прості користувачі, які взаємодіють з системою. Користувач може бути тільки людиною.

Підсистеми – у моделях UML підсистеми існують як тип стереотипних компонентів, поведінкові одиниці в системі [9]. Підсистеми використовуються в діаграмах класів, компонентів і варіантів використання для представлення великомасштабних компонентів у системі, яку ви моделюєте.

Зв'язки та відношення – зв’язки між елементами моделі. Відношення – тип елемента моделі, який додає семантику до моделі, допомогая визначити структуру та поведінку між елементами моделі.

Випадки використання – описує функцію, яку виконує система для досягнення мети користувача. Випадок використання повинен давати спостережуваний результат, який є цінним для користувача системи.

3.3 Побудова діаграм взаємодії (послідовності та кооперації)

Діаграми взаємодії (послідовності та кооперації) є графічними засобами моделювання поведінки системи. Вони використовуються для візуалізації послідовності виконання операцій між об'єктами або компонентами системи. Діаграми взаємодії описують взаємодію між об'єктами або компонентами в системі, показуючи послідовність виконання операцій та передачу повідомлень між ними. Діаграми взаємодії призначені для візуалізації та аналізу взаємодії об'єктів або компонентів в системі, але не призначені для детального опису внутрішньої реалізації об'єктів або документування архітектури системи [18].

В обох типах діаграм взаємодії можна використовувати такі елементи:

– Об'єкти: представляють ролі або класи об'єктів, які беруть участь у взаємодії.

– Повідомлення: показують обмін даними або виклики методів між об'єктами.

– Зв'язки: показують зв'язки між об'єктами, які вказують напрямок взаємодії.

1) Діаграми послідовності (Sequence Diagrams):

Діаграми послідовності показують взаємодію між об'єктами в системі в часовому порядку. Вони використовуються для моделювання викликів методів об'єктів та відповідних повернень результатів. Основні елементи діаграми послідовності включають об'єкти, повідомлення, роздільники часу та змінні.

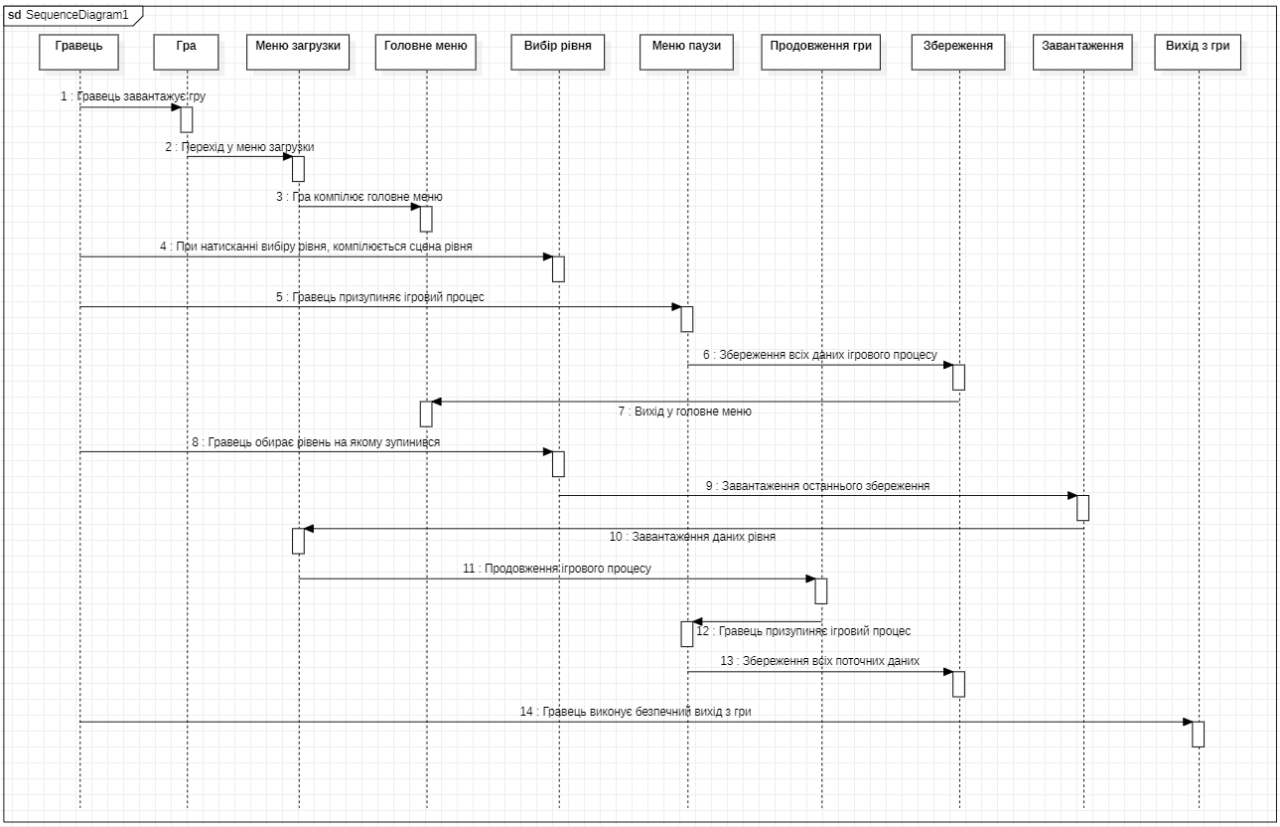

Рисунок 3.2 – Діаграма послідовності

2) Діаграми кооперації (Collaboration Diagrams):

Діаграми кооперації (також відомі як діаграми комунікації) представляють структуру взаємодії між об'єктами в системі. Вони показують об'єкти, повідомлення та зв'язки між об'єктами. Діаграми кооперації демонструють, як об'єкти спілкуються між собою для виконання певних функцій чи операцій [18].

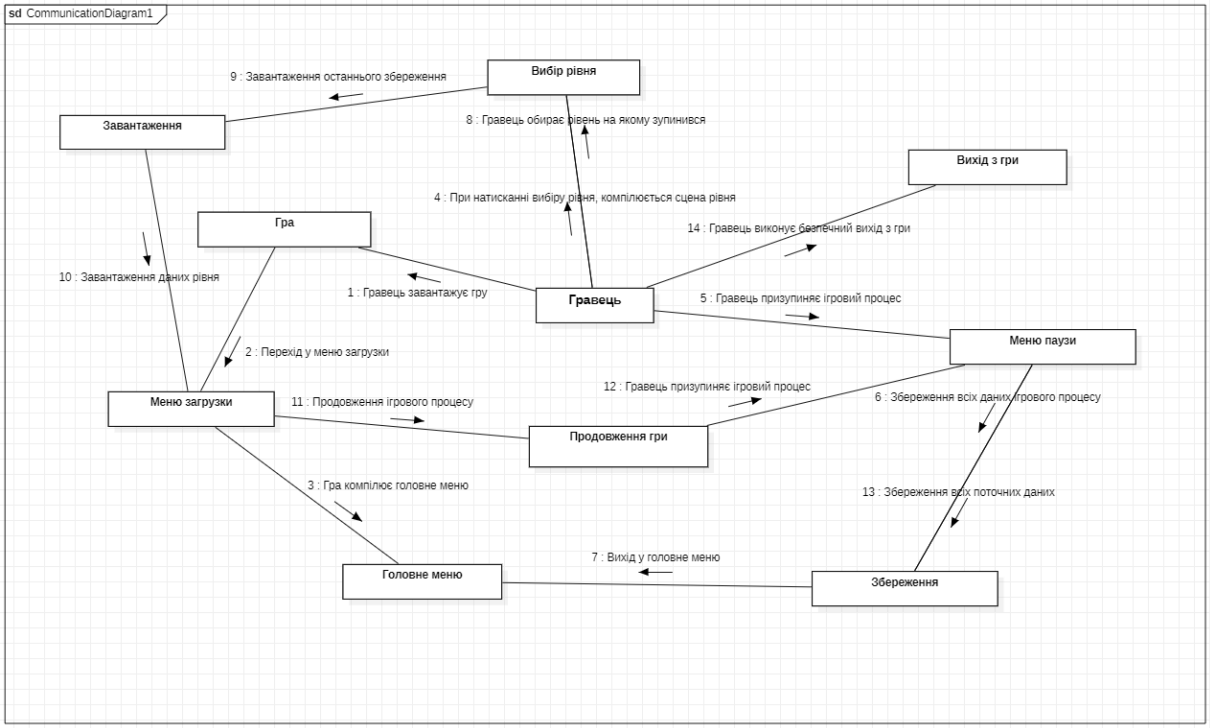

Рисунок 3.3 – Діаграма кооперації

3.4 Діаграми станів та переходів

Діаграми станів та переходів (Statechart diagrams) корисні для моделювання поведінки системи, виявлення різних станів та переходів, аналізу послідовностей дій та умов переходу, а також документування та спілкування між розробниками та зацікавленими сторонами. Вони допомагають зрозуміти і прогнозувати поведінку системи, з'ясувати взаємозв'язки між станами та діями та виявити можливі проблеми чи неточності у проекті системи [18].

Елементи діаграми станів та переходів включають наступні компоненти:

1. Стан (State): Представляє конкретний стан системи, в якому може перебувати об'єкт або компонент. Це може бути стан активності, очікування, виконання або будь-який інший визначений стан.

2. Подія (Event): Визначає зовнішню подію або внутрішнє сприйняття, яке спричиняє перехід з одного стану в інший. Наприклад, подія може бути натисканням кнопки, отриманням повідомлення або закінченням таймера.

3. Умова (Condition): Вказує умову або обмеження, які мають бути виконані, щоб здійснити перехід з одного стану в інший. Умова може бути логічним виразом, що оцінюється на істинність або виконання певної умови.

4. Дія (Action): Представляє дію, яка відбувається під час переходу з одного стану в інший. Це може бути виконання певного функціоналу, відправлення повідомлення, зміна значення змінних тощо.

5. Перехід (Transition): Вказує перехід між двома станами відповідно до певної події та умови. Він показує, як система реагує на подію та переходить з одного стану в інший, виконуючи відповідну дію.

6. Початковий стан (Initial state): Позначає стан, з якого починається система або об'єкт перед спрацюванням будь-яких подій.

7. Кінцевий стан (Final state): Позначає закінчений стан системи або процесу, коли він досягає свого кінцевого результату або завершується.

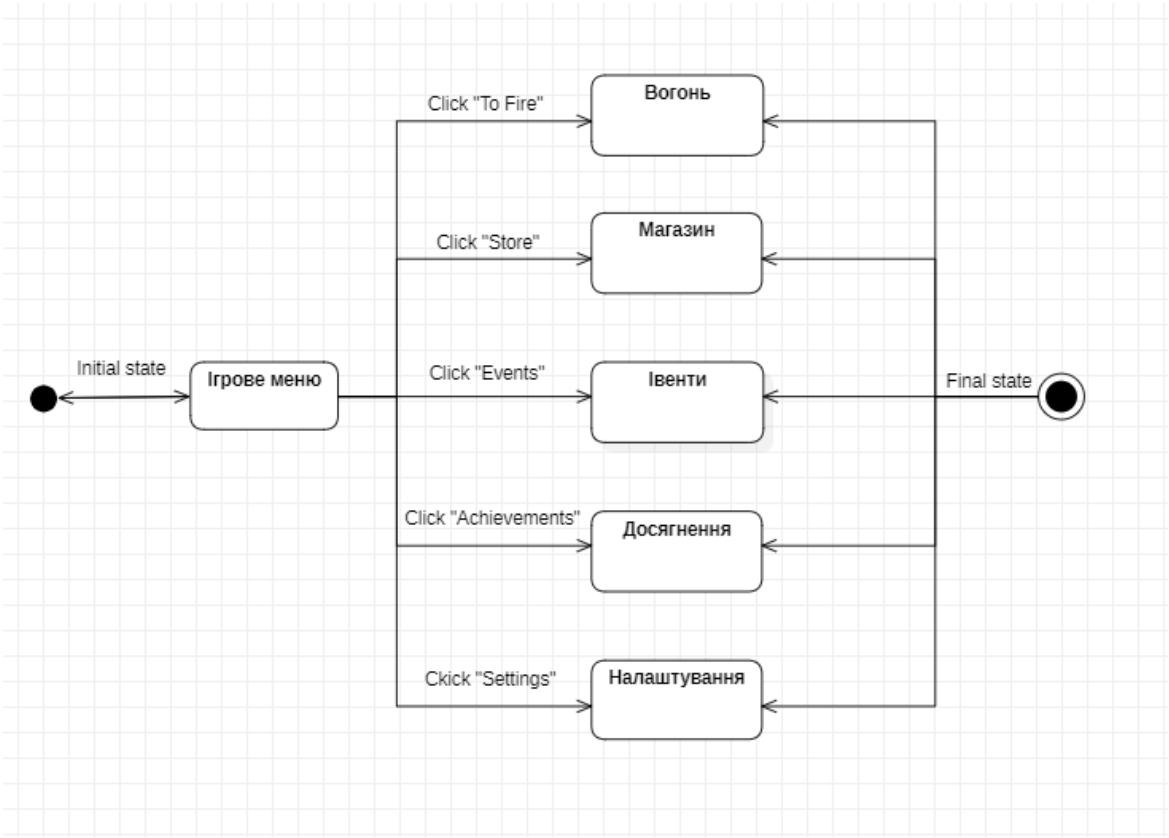

Рисунок 3.4 – Діаграма станів для всієї гри

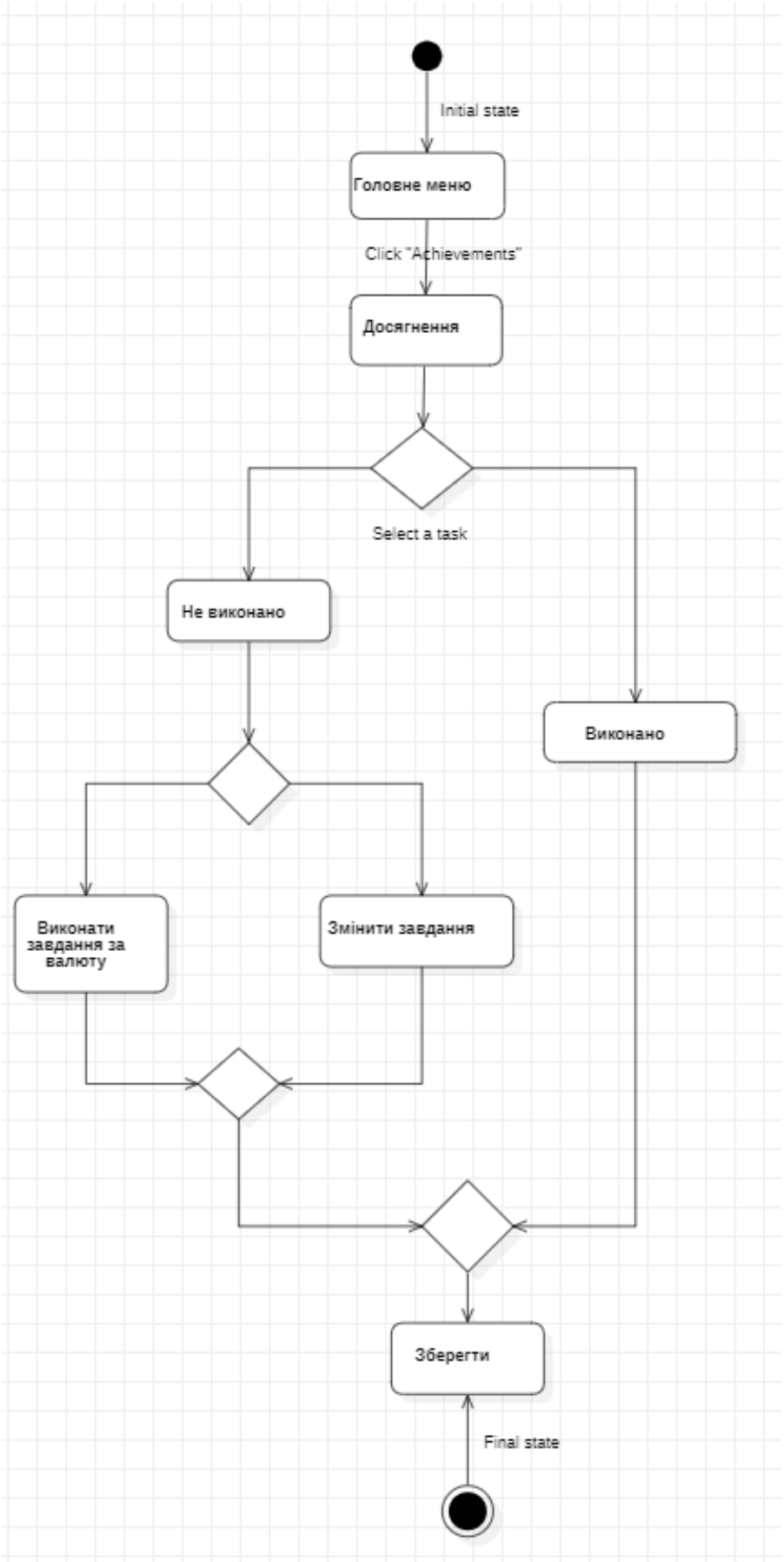

Рисунок 3.5 – Діаграма станів для виконання досягнень

В даних діаграмах описано основні можливості в грі, зі стартового меню є перехід майже до всіх функцій гри такі як: To Fire, Store, Events, Achievements, Settings. Також є діаграма виконання дисягнень, де можливо завершити виконане завдання або відмінити ще не виконане завдання для початку нового.

3.5 Діаграма діяльності

Діаграми діяльності (activity diagrams) відображають послідовність дій, що виконується в процесі реалізації певного варіанту використання або функціонування системи в цілому.Діаграми діяльності є аналогом блок-схеми будь якого алгоритму. Вони, як і діаграми станів та переходів, відображаються у вигляді орієнтованого графу, вершинами якого є дії, а ребрами – переходи між діями [18].

Елементи діаграми діяльності включають наступні компоненти:

– Дія (Action): Представляє окрему дію або крок, який виконується в процесі. Це може бути конкретна операція, функція, виконання певного блоку коду або інша дія, що відбувається в системі.

– Рішення (Decision): Показує розгалуження в процесі в залежності від певної умови або вибору. Він має два або більше варіанти виконання, які залежать від встановленої умови.

– Старт та кінець (Start/End): Позначають початок та завершення діаграми діяльності або певної послідовності дій. Вони показують початкову та кінцеву точки процесу або діяльності.

– Паралельність (Concurrent): Вказує на одночасне виконання двох або більше дій або процесів, які не залежать один від одного.

– Флоу (Flow): Вказує послідовність дій або переходів між ними. Використовує стрілки для показу напрямку виконання дій.

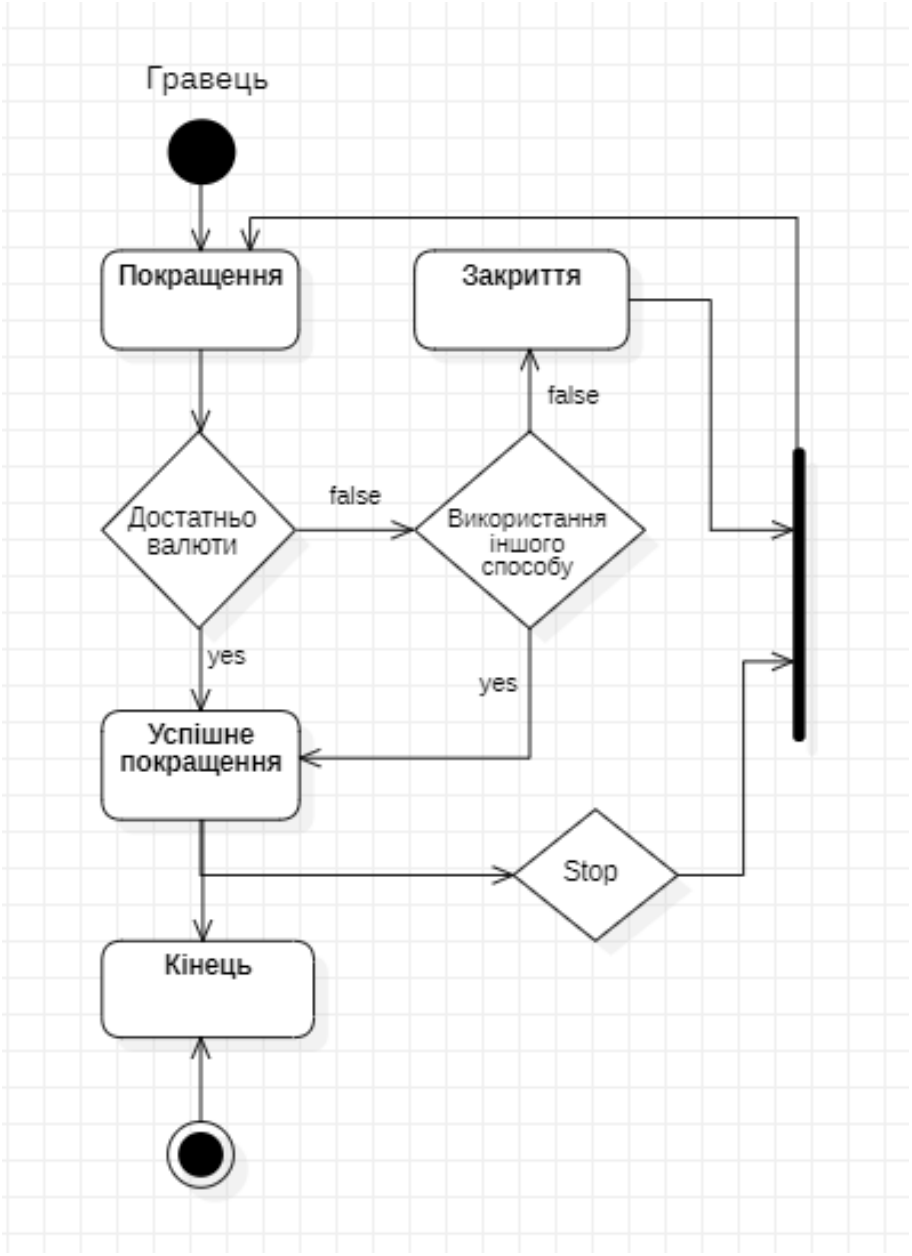

Рисунок 3.6 – Діаграма діяльності покращення

Також діаграми діяльності використовуються для відображення послідовності дій при моделюванні бізнес-процесів. При цьому використовується додатковий елемент діаграми, який має назву swimlane. Swimlane в дословному перекладі означає дорожка плавального басейну (по аналогії з графічним відображенням). В якості swimlanes на діаграмі можуть виступати фізичні особи, групи осіб, відділи підприємства, чи навіть окремі організації.

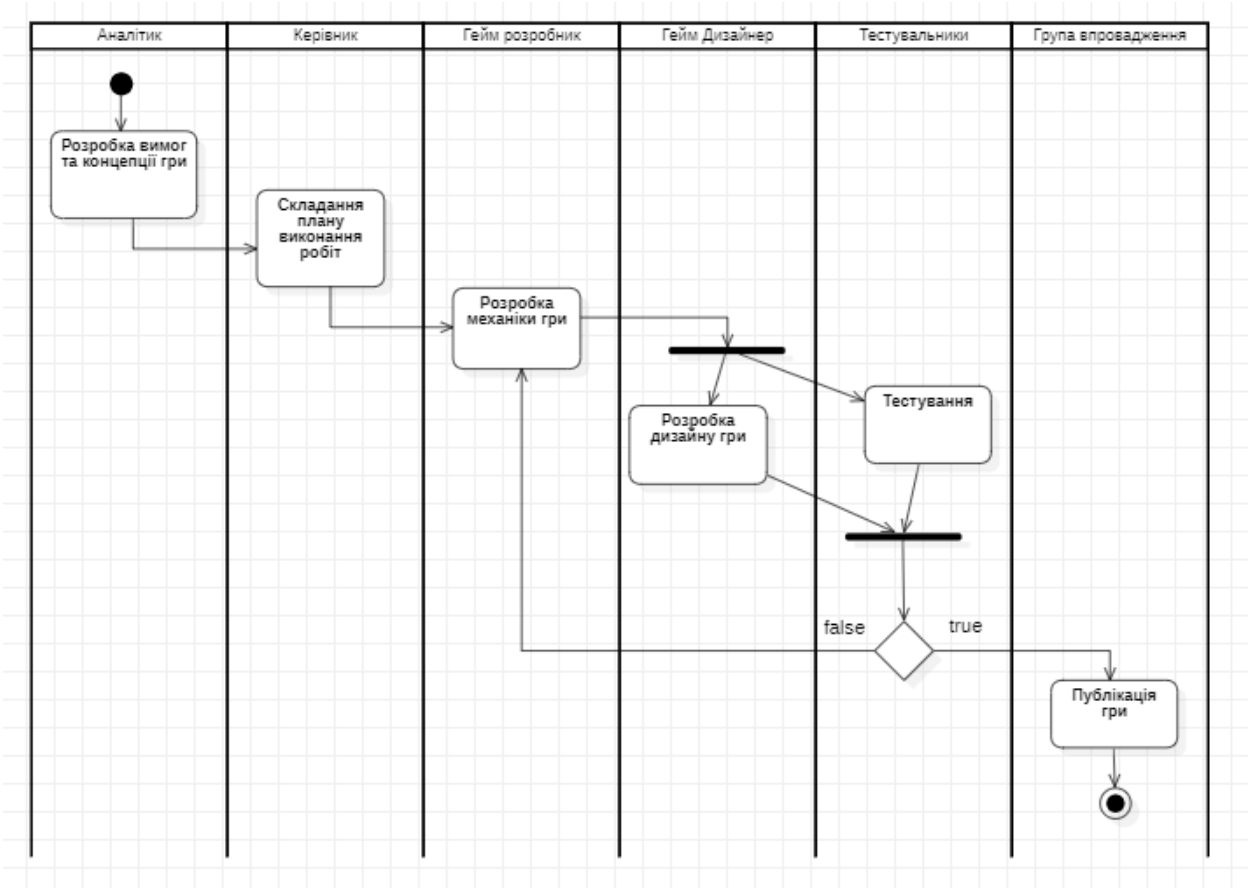

Рисунок 3.7 – Діаграма діяльності для відображення бізнес-процесів

Створено дві діаграми діяльності:

1) Покращення. Діаграма перевіряє на наявність ігрової валюти для оновлення та вибір за допомогою якої валюти зробити оновлення, також припинення всього процесу оновлення.

2) Діаграма відображення послідовності дій при моделюванні бізнес-процесів.

3.6 Розробка діаграм компонентів та розгортання

Діаграма компонентів — діаграма в UML, на якій відображаються основні компоненти системи та зв'язки між ними. Компоненти цієї діаграми відповідають окремим модулям або підсистемам системи, які мають визначені інтерфейси та функціональні можливості. Крім того, на діаграмі можуть бути показані залежності між компонентами, їх інтерфейси та залучені зовнішні

ресурси, такі як бази даних або сервіси. Діаграма компонентів допомагає зрозуміти структуру системи та взаємодію її компонентів [18].

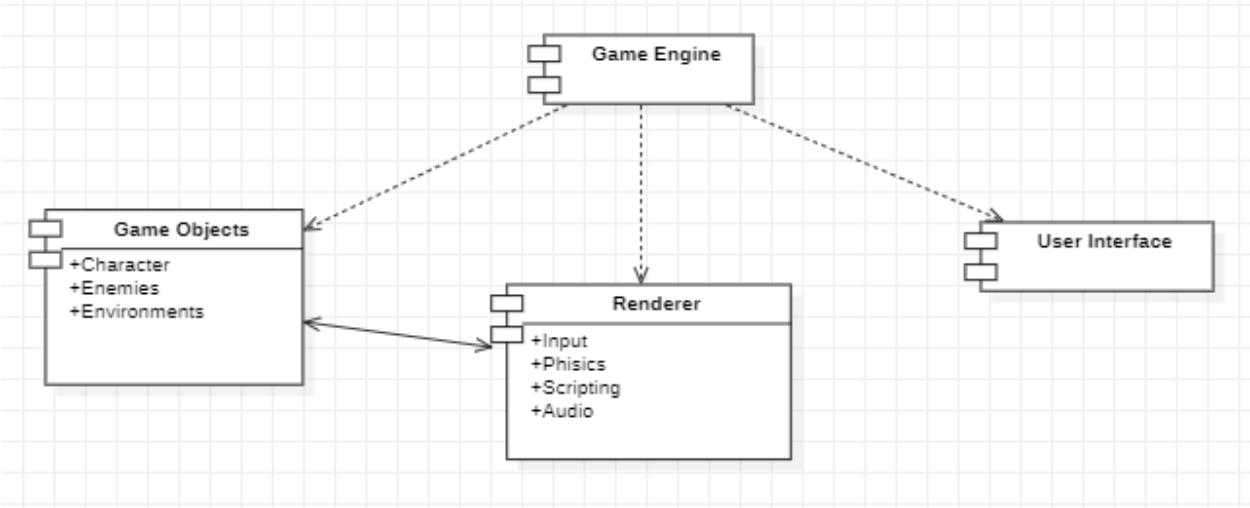

Рисунок 3.8 – Діаграма компонентів для Android-гри жанку SA

Діаграма розгортання — діаграма в UML (рис. 3.9), на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об’єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду.

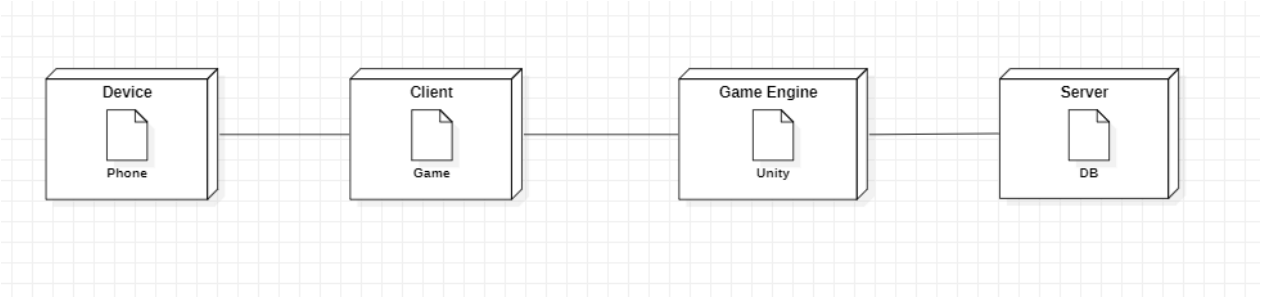

Рисунок 3.9 – Діаграма розгортання для Android-гри жанку SA

Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонентів. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв’язки між типами компонентів.

Висновки до розділу 3

У розділі проаналізовано розроблено діаграми, такі як діаграма використання, послідовностей, діяльності та інші. Ці діаграми створені детально та лаконічно, що спрощує процес створення гри і описує основний сюжет гри, а також основні можливості та функції.

У поверхневому usecase описано успішний та альтернативний сценарії гри. Це дозволяє визначити основний функціонал та можливості гри. У діаграмах взаємодії описано процес збереження гри та ігрового рекорду, а також процес збереження ігрового процесу після виходу з гри для подальшого проходження. Це забезпечує зручну управління грою та збереження прогресу гравця. У діаграмах діяльності описано процес покращення у грі з можливістю відміни покращення, якщо гравець допустив помилку у ході гри. Це дозволяє гравцеві вдосконалювати свої навички та коригувати свої дії в разі помилки.

Загалом, розділ 3 надає повний огляд проєктування та моделювання гри, описуючи основні компоненти, функції та можливості. Ці висновки допоможуть в подальшому у розробці гри.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ

4.1 Створення дизайну ігрового застосунку

UI-дизайн (дизайн користувацького інтерфейсу) в ігровому застосунку відіграє надзвичайно важливу роль, оскільки він визначає спосіб взаємодії користувачів з грою. Грамотно розроблений дизайн може значно покращити враження гравців від гри та забезпечити їм комфортну та просту взаємодію з геймплеєм.

В ігровому застосунку присутні такі важливі елементи інтерфейсу, як головне меню, меню налаштувань, меню паузи, меню покупок, меню апгрейду, меню рівнів, меню бонусу та досягнень. Для створення цього інтерфейсу були використані стандартні інструменти Unity, зокрема UI Canvas-компоненти, що включають різноманітні елементи, такі як кнопки, текстові поля, панелі та інші.

Дизайн UI-інтерфейсу в ігровому застосунку виконує важливу функцію, не лише забезпечуючи взаємодію користувача з грою, але й впливаючи на загальне задоволення від геймплею. Правильно розроблений UI-дизайн може створити гармонійне та привабливе середовище, в якому гравці зможуть комфортно орієнтуватись та насолоджуватись грою.

Рисунок 4.1 – Головне меню

На рис. 4.1, показано головне меню, яке включає три основні кнопки: Play, Shop та Upgrade; та додаткові: Bonus, Achievements, Settings та кнопки отримання ігрової валюти. По натисненню на кнопку Play гравець перейде у меню рівнів де можна вибрати поточний активний рівень, та почати грати. Якщо гравець здолав ворогів то відкриється панель перемоги а якщо ні, то відкриється панель поразки.

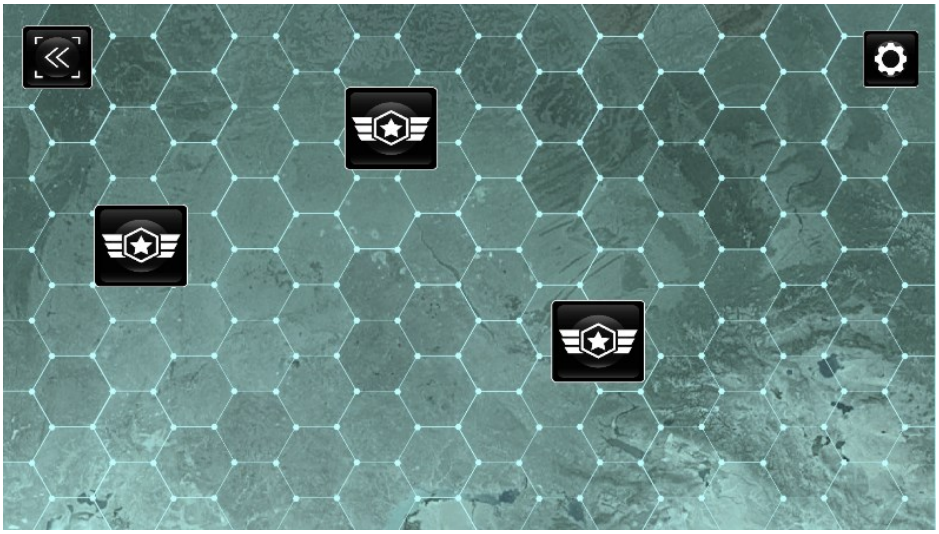

Рисунок 4.2 – Меню рівнів

При натисненні на кнопку Shop користувачу відкриється панель із налаштуваннями зброї, де можна купувати будь-який із трьох видів зброї та різні рівні якості цієї зброї. Також гравець має встановити цю зброю на свій літак.

Рисунок 4.3 – Меню налаштування зброї

При натисненні на кнопку Upgrade користувачу відкриється панель, із інвентарем купленої зброї з минулої панелі. Якщо гравець придбав якийсь вид зброї, то вона відображиться у цьому вікні, а вже тут є можливість покращити ту чи іншу зброю за ігрову валюту.

Рисунок 4.4 – Меню інвентарю

Це були основні кнопки для геймплею, та є ще додаткові, такі як: Bonus та Achievements. По натисканню на перший відкриється вікно з щоденним бонусом у вигляді ігрової валюти.

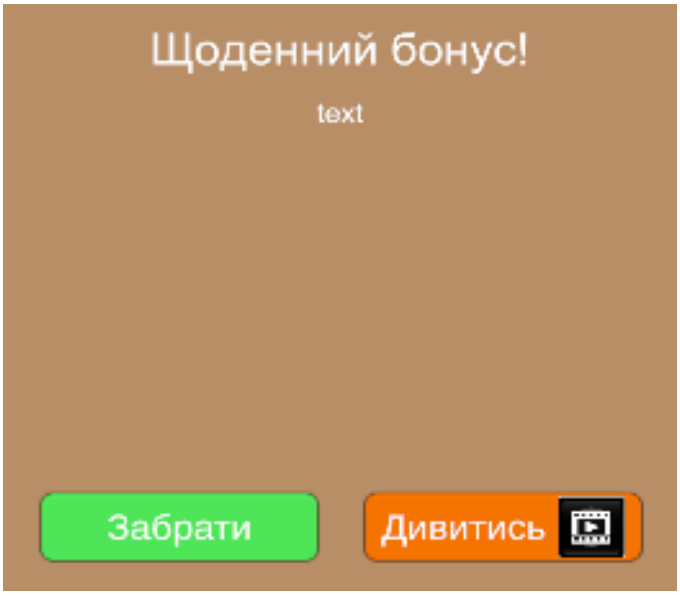

Рисунок 4.5 – Меню щоденного бонусу

А по натисканню на другий, тобто Achievements, відкриється вікно з доступними досягненнями у грі, наприклад: перегляд реклами, вбивство ворогів та ін.

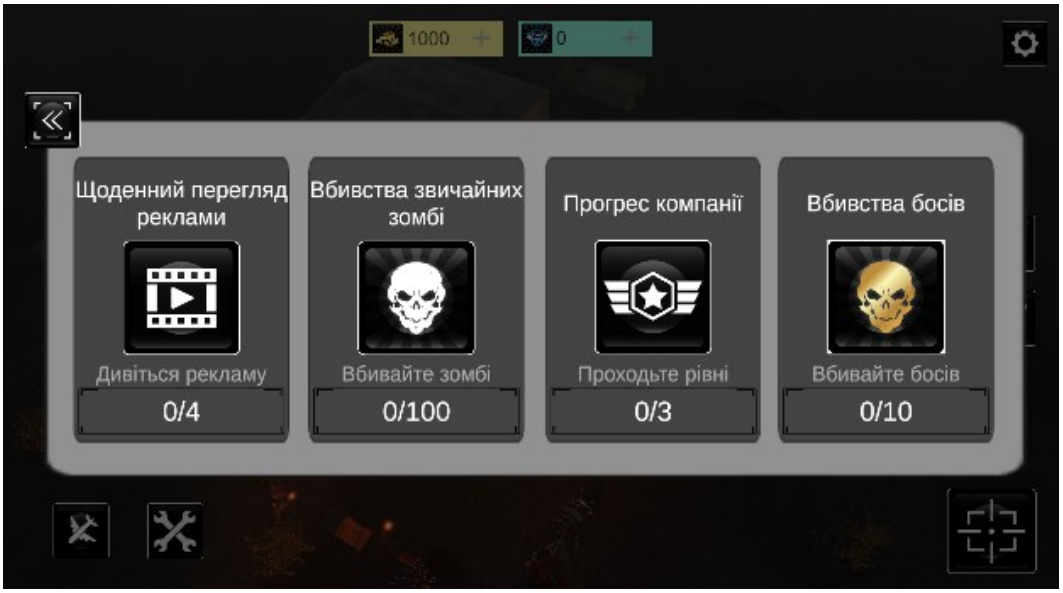

Рисунок 4.6 – Меню досягнень

Останнє що було зроблено, це меню Settings в якому гравець може вимкнути звуки або музику, якщо вони йому не подобаються та по натисканню на кнопку у самому рівні, поставити на паузу.

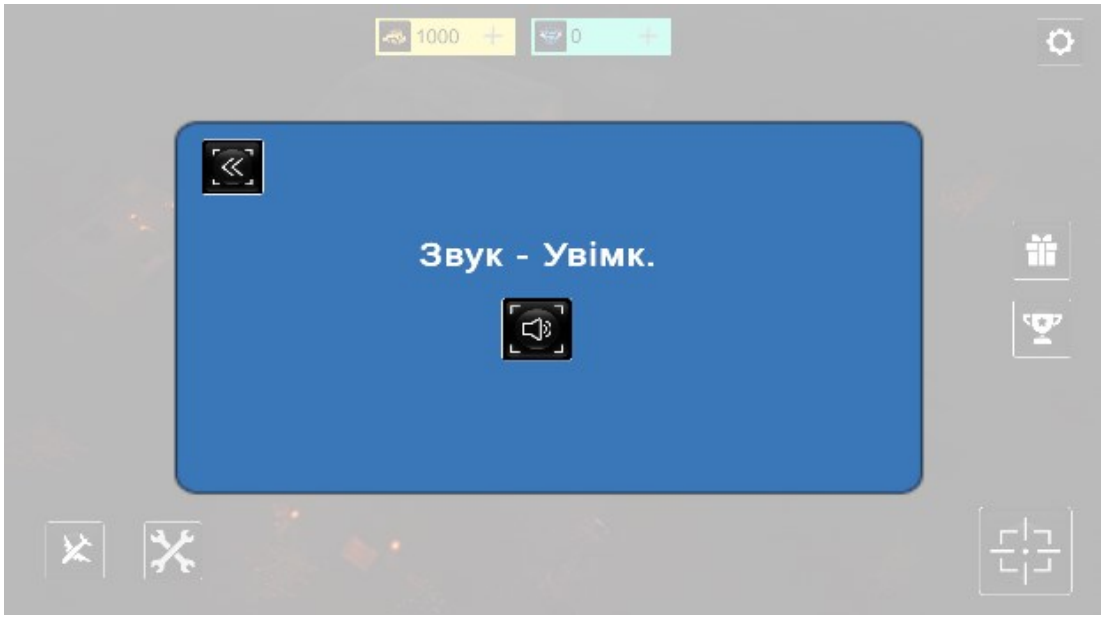

Рисунок 4.7 – Меню налаштувань

Як вже було описано вище, створення зручного та цікавого UI-інтерфейсу потрібен для дотримання принципів зручності й доступності для користувача. А також для зацікавлення користувача необхідні елементи для покращення якості життя гравця та полегшення у здобуванні контенту.

В результаті маємо такий сценарій: гравець заходить у налаштування літака, покупає зброю за початковий капітал, та йде проходити рівні для заробітку валюти щоб купити більш сильну зброю. Також по ходу гри виконуються досягнення для отримання додаткової валюти. І врешті гравець має можливість покращувати будь-яку куплену зброю за ту саму валюту.

4.2 Програмна реалізація UI-елементів до застосунку

Програмна реалізація UI-елементів до застосунку полягає в створенні логіки, що керує взаємодією між користувачем та грою через інтерфейс. Це охоплює різні аспекти, такі як обробка введення користувача, відображення графічних компонентів і реакція на події.

Зазвичай програмна реалізація UI-елементів включає створення об'єктів, що представляють графічні елементи, такі як кнопки, текстові поля, панелі тощо. Ці об'єкти налаштовуються з відповідними параметрами, такими як розмір, положення, колір, шрифт і т. д. Додатково, до цих елементів можуть бути додані різні компоненти та функції, що визначають їх поведінку, такі як реакція на кліки, анімації, перехід до інших інтерфейсних екранів тощо.

Для програмної реалізації ігрового меню було розроблено такі скрипти: Buttons – для функціонування кожної кнопки на головному меню, тобто ввімкнення та вимкнення потрібної панелі, які будуть спрацьовувати по натисканню на компонент Button на сцені.

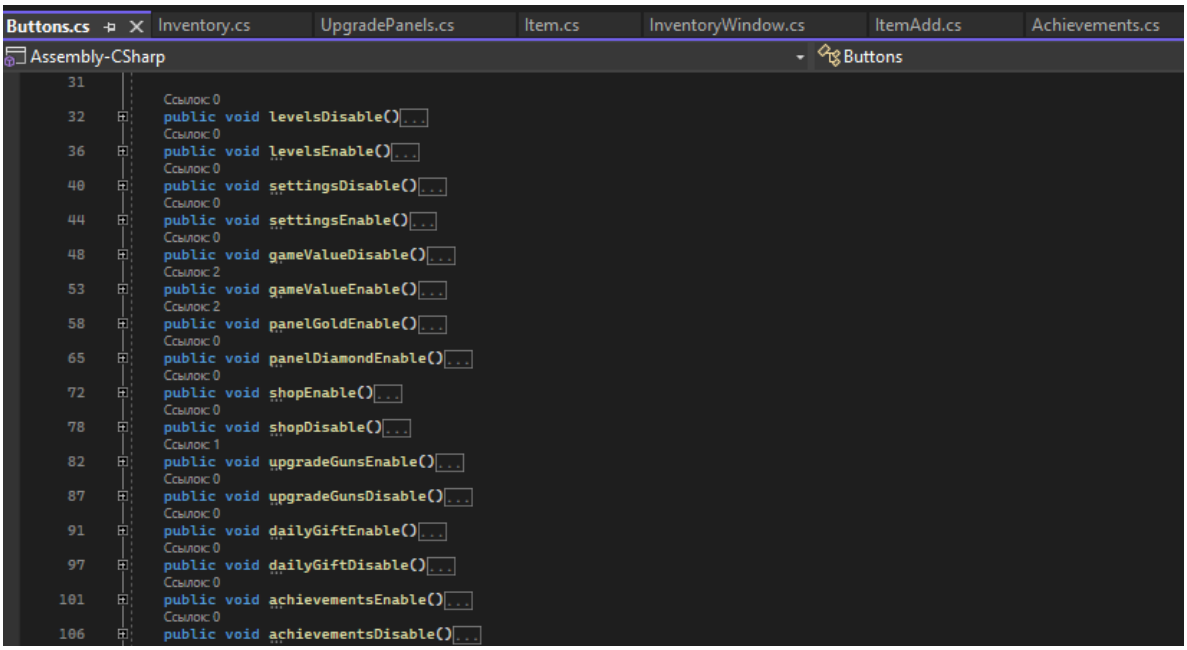

Рисунок 4.8 – Скрипт Buttons

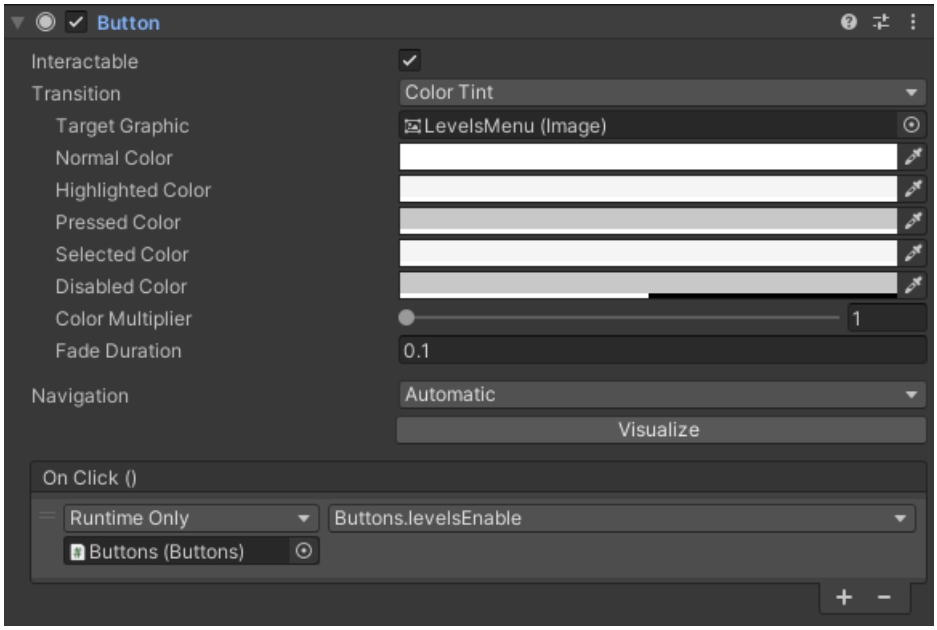

Рисунок 4.9 – Налаштування компоненту на сцені

Далі було розроблено скрипти: Item та ItemAdd, за допомогою них було реалізовано створення об’єктів зброї для подальшої роботи з ними та їх придбання.

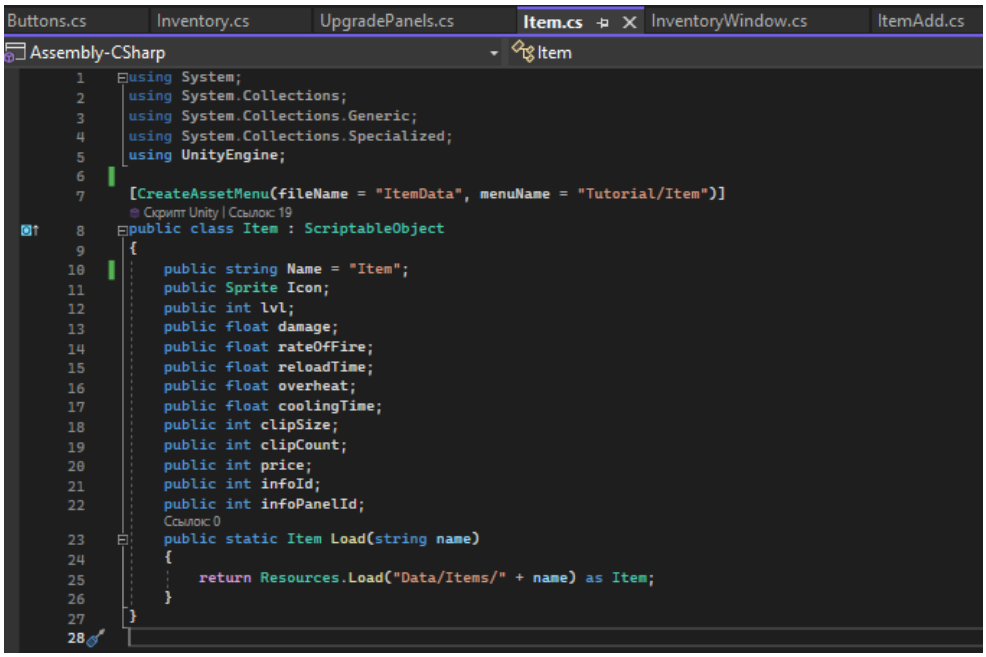

Рисунок 4.10 – Скрипт Item

Скрипт Item являється класом наслідуючий ScriptableObject, тобто за допомогою нього було створено об’єкти(ScriptableObjects) різних видів зброї з індивідуальними показниками для них.

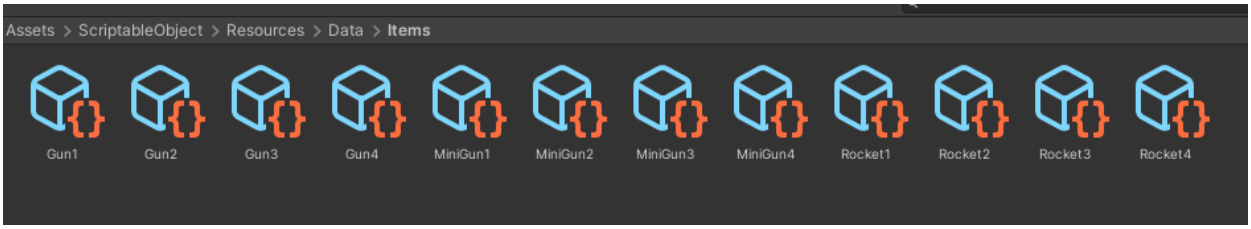

Рисунок 4.11 – Створені ScriptableObjects об’єкти зброї

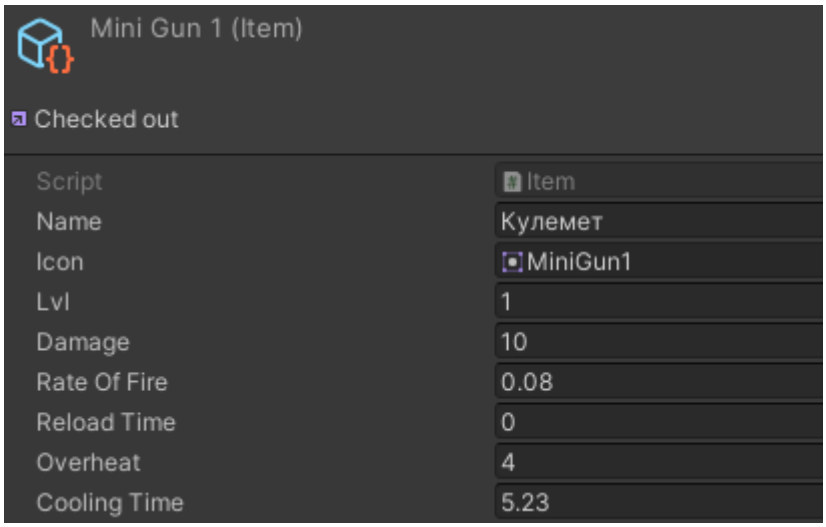

Рисунок 4.12 – Приклад одного ScriptableObject

За допомогою цих об’єктів ми маємо гнучку систему додавання та зміни кожного параметру окремо.

Рисунок 4.13 – Метод придбання зброї

У скрипті ItemAdd було створено метод додавання зброї до інвентарю, про який буде розказано нижче. Метод працює по натисканню на кнопку придбання зброї гравцем.

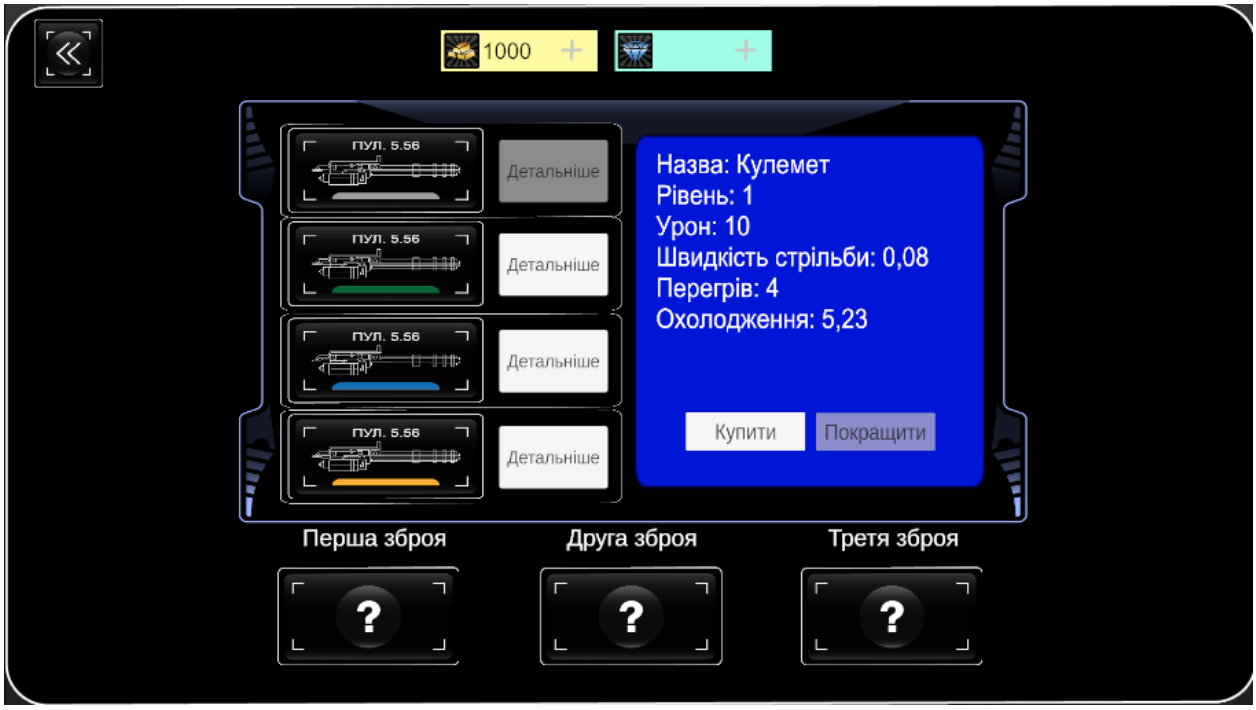

Рисунок 4.14 – Відображення характеристик та можливість придбання

Відображення характеристик обраної зброї було реалізовано за допомогою методу Redraw().

```
public void Redraw()
{

    var item = gunSlotInfo.allItems[slotId];
    if (gunSlotInfo.gunSlotId == 0) { textInfo.text = $"Назва: {item.Name}\nРівень: {item.lvl}\nУрон: {item.damage}\n" +
          $"Швидкість стрільби: {item.rateOfFire}\nПерегрів: {item.overheat}\nОхолодження: {item.coolingTime}"; }
    if (gunSlotInfo.gunSlotId == 1) { textInfo.text = $"Назва: {item.Name}\nРівень: {item.lvl}\nУрон: {item.damage}\n" +
          $"Швидкість стрільби: {item.rateOfFire}\nПерезарядження: {item.reloadTime}\nРозмір обойми: {item.clipSize}\n" +
          $"Кількість обійм: {item.clipCount}"; }
    if (gunSlotInfo.gunSlotId == 2) { textInfo.text = $"Назва: {item.Name}\nРівень: {item.lvl}\nУрон: {item.damage}\n" +
          $"Перезарядження: {item.reloadTime}\nКількість ракет: {item.clipSize}"; }
}

}
```

Рисунок 4.15 – Відображення характеристик зброї

Для відображення інвентаря придбаної зброї та для подальшого його покращення було створено скрипти: Inventory – для зберігання придбаної зброї та створення ігрової валюти, та InventoryWindow – для відображення всієї наявної зброї на сцені.

```
46      void Awake()
47      {
48          inventoryItems = new List<Item>();
49          for (var i = 0; i < startItems.Count; i++)
50          {
51              AddItem(startItems[i]);
52          }
53
54      }
        Ссылок: 2
55      public void AddItem(Item item)
56      {
57          startItems.Add(item);
58          onItemAdded?.Invoke(item);
59      }
```

Рисунок 4.16 – Метод для відображення придбаної зброї

Відображення зброї відбувається за рахунок створення нових об’єктів на панелі інвентаря у скрипті InventoryWindow, які беруть свої параметри через масив придбаних ScriptableObject об’єктів зброї.

Рисунок 4.17 – Відображення придбаної зброї у грі

```
List<GameObject> drawnIcons= new List<GameObject>();

Сообщение Unity | Ссылок 0
void Start()
{
    targetInventory.onItemAdded += OnItemAdded;
    Redraw();
}
Ссылок 1
void OnItemAdded(Item obj) => Redraw();
Ссылок 2
public void Redraw()
{
    ClearDrawn();
    for(var i = 0; i < targetInventory.startItems.Count; i++)
    {
        var item = targetInventory.startItems[i];

        var icon = new GameObject("Icon");
        icon.AddComponent<Image>().sprite = item.Icon;
        icon.transform.SetParent(itemsPanel);

        var button = icon.AddComponent<Button>();
        int indexGun = item.infoId;
        button.onClick.AddListener(() => upgradePanels.ShowObjectWindow(indexGun));
        button.onClick.AddListener(() => upgradePanels.Redraw(item));
        drawnIcons.Add(icon);
    }
}
Ссылок 1
public void ClearDrawn()
{
    for(var i = 0; i < drawnIcons.Count; i++)
    {
        Destroy(drawnIcons[i]);
    }
    drawnIcons.Clear();
}
}
```

Рисунок 4.18 – Створення об’єктів придбаної зброї у грі

Наступним було реалізовано встановлення придбаної зброї у визначене місце на літаку, для того щоб у рівні використовувалися актуальні параметри урону. Для цього було розроблено методи itemEquip() для кнопки втановлення та RedrawGunsEquip() для оновлення відображення активної зброї.

```
public void itemEquip()
{
    foreach (UnityEngine.UI.Button i in GunSlotInfo.equipInteractable)
    {
        i.interactable = true;
    }
    GunSlotInfo.equipInteractable[idGunSlotInfo.slotId].interactable = false;
    if (GunSlotInfo.gunSlotId == 0) { buttons.equipItem[0] = GunSlotInfo.allItems[idGunSlotInfo.slotId];
        for (int i = 0; i < GunSlotInfo.isActive1.Length; i++) { GunSlotInfo.isActive1[i] = false; } GunSlotInfo.isActive1[idGunSlotInfo.slotId] = true; } else
        if (GunSlotInfo.gunSlotId == 1) { buttons.equipItem[1] = GunSlotInfo.allItems[idGunSlotInfo.slotId];
        for (int i = 0; i < GunSlotInfo.isActive2.Length; i++) { GunSlotInfo.isActive2[i] = false; } GunSlotInfo.isActive2[idGunSlotInfo.slotId] = true; } else
        if (GunSlotInfo.gunSlotId == 2) { buttons.equipItem[2] = GunSlotInfo.allItems[idGunSlotInfo.slotId];
        for (int i = 0; i < GunSlotInfo.isActive3.Length; i++) { GunSlotInfo.isActive3[i] = false; } GunSlotInfo.isActive3[idGunSlotInfo.slotId] = true; }
    buttons.RedrawGunsEquip();
    buttons.SaveEquip();
    GunSlotInfo.SaveGunBought();
}
}
```

Рисунок 4.19 – Метод для кнопки встановлення зброї

```
public void RedrawGunsEquip()
{
    audioSource.PlayOneShot(audioClip);
    if (equipItem[0] != null) { buttonGunSlot[0].GetComponent<Image>().sprite = equipItem[0].Icon; }
    else buttonGunSlot[0].GetComponent<Image>().sprite = nullEquip;
    if (equipItem[1] != null) { buttonGunSlot[1].GetComponent<Image>().sprite = equipItem[1].Icon; }
    else buttonGunSlot[1].GetComponent<Image>().sprite = nullEquip;
    if (equipItem[2] != null) { buttonGunSlot[2].GetComponent<Image>().sprite = equipItem[2].Icon; }
    else buttonGunSlot[2].GetComponent<Image>().sprite = nullEquip;

}
```

Рисунок 4.20 – Метод для оновлення відображення активної зброї

Рисунок 4.21 – Відображення встановленої зброї

Далі було розроблено скрипт UpgradePanels в якому реалізовано вікно апгрейду зброї, на якій відображаються характеристики окремих видів зброї та відображення підвищених характеристик після апгрейду. Також було розроблено саме підвищення характеристик після покращення. Для цього у скрипті Inventory було додано метод Upgrade(), який викликається по натисканню на кнопку на панелі покращення.

```
public void ShowObjectWindow(int index)
{
    upgradePanels.SetActive(true);
    targetInventory.actualUpgradePanel = index;
}
Ссылок: 3
public void Redraw(Item item)
{
    GunIcon.GetComponent<Image>().sprite = item.Icon;
    GunIconUp.GetComponent<Image>().sprite = item.Icon;
    if (item.infoPanelId == 0)
    {
        textInfo.text = $"Назва: {item.Name}\nРівень: {item.lvl}\nУрон: {item.damage}\n" +
            $"Швидкість стрільби: {item.rateOfFire}\nПерегрів: {item.overheat}\nОхолодження: {item.coolingTime}";
        textInfoUp.text = $"Назва: {item.Name}\nРівень: {item.lvl+1}\nУрон: {item.damage+2}\n" +
            $"Швидкість стрільби: {item.rateOfFire}\nПерегрів: {item.overheat}\nОхолодження: {item.coolingTime}";
        textUpgradeCost.text = $"{item.price}";
    }
    if (item.infoPanelId == 1)
    {
        textInfo.text = $"Назва: {item.Name}\nРівень: {item.lvl}\nУрон: {item.damage}\n" +
            $"Швидкість стрільби: {item.rateOfFire}\nПерезарядження: {item.reloadTime}\nРозмір обойми: {item.clipSize}\n" +
            $"Кількість обійм: {item.clipCount}";
        textInfoUp.text = $"Назва: {item.Name}\nРівень: {item.lvl+1}\nУрон: {item.damage+15}\n" +
            $"Швидкість стрільби: {item.rateOfFire}\nПерезарядження: {item.reloadTime}\nРозмір обойми: {item.clipSize}\n" +
            $"Кількість обойм: {item.clipCount}";
        textUpgradeCost.text = $"{item.price}";
    }
    if (item.infoPanelId == 2)
    {
        textInfo.text = $"Назва: {item.Name}\nРівень: {item.lvl}\nУрон: {item.damage}\n" +
            $"Перезарядження: {item.reloadTime}\nКількість ракет: {item.clipSize}";
        textInfoUp.text = $"Назва: {item.Name}\nРівень: {item.lvl+1}\nУрон: {item.damage+50}\n" +
            $"Перезарядження: {item.reloadTime}\nКількість ракет: {item.clipSize}";
        textUpgradeCost.text = $"{item.price}";
    }
}
```

Рисунок 4.22 – Скрипт UpgradePanels

```
115     public void Upgrade()
116     {
117
118         int index = 0;
119         foreach (Item i in startItems)
120         {
121             if (i.infoId == actualUpgradePanel)
122             {
123                 index = startItems.LastIndexOf(i);
124                 actualId = index;
125             }
126         }
127         if (gold >= startItems[index].price)
128         {
129             gold = gold - startItems[index].price;
130             startItems[index].price = startItems[index].price + 50;
131             startItems[index].lvl = startItems[index].lvl + 1;
132             if (startItems[index].infoPanelId == 0) { startItems[index].damage += 2; }
133             else if (startItems[index].infoPanelId == 1) { startItems[index].damage += 15; }
134             else if (startItems[index].infoPanelId == 2) { startItems[index].damage += 50; }
135             UpdateGold();
136             upgradePanels.Redraw(startItems[index]);
137             SaveInventory();
138         }
139         else
140         {
141             enoughMoneyPanel.SetActive(true);
142             enoughMoneyText.text = $"Нехватает: {startItems[index].price - gold}";
143         }
144     }
```

Рисунок 4.23 – Метод Upgrade()

Рисунок 4.24 – Меню покращення зброї

Також було розроблено можливість перейти до апгрейду зброї з панелі придбання. Це було реалізовано за допомогою методу upgradeButton(), який бере масив придбаної зброї та по натисканню на кнопку переходе до панелі покращення.

```
public void upgradeButton()
{
    buttons.upgradeGunsEnable();
    int index = 0;
    foreach (Item i in targetInventory.startItems)
    {
        if (i.infoId == idGun)
        {
            index = targetInventory.startItems.LastIndexOf(i);
        }
    }
    upgradePanels.upgradePanels.SetActive(true);
    upgradePanels.Redraw(targetInventory.startItems[index]);

    targetInventory.actualUpgradePanel = idGun;
    idButtonGunSlot.gunSlotDisable();
}
```

Рисунок 4.25 – Метод upgradeButton()

Далі було розроблено систему досягнень. Для цього було створено скрипт Achievements [Додаток А]. Всього буде чотири досягнення. Перший полягає в тому щоб гравець дивився рекламу та отримував нагороду у вигляді валюти, але виконувати це досягнення можна один раз на день.

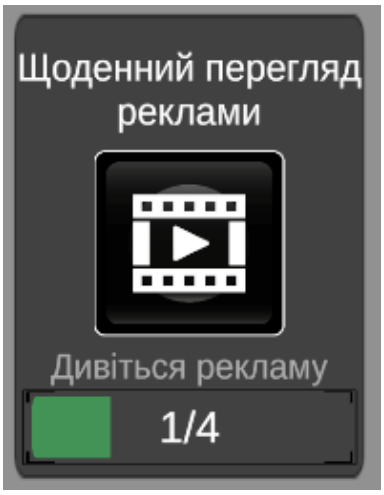

Рисунок 4.26 – Перше досягнення з рекламою

Для того щоб досягнення оновлювалося потрібно створити дві кнопки, на першу – буде програватися реклама, а на другу – якщо досягнення виконано, то з’являється кнопка забрати нагороду. Для перегляду реклами було створено скрипт AdsManager [Додаток В].

```
public void WatchAd()
{
    if (adsWatched != adsCount)
    {
        adsManager.ShowAds("Rewarded_Android");
        adsWatched++;
        fillImageAds.fillAmount = adsWatched / adsCount;
        adsText.text = $"{adsWatched}/{adsCount}";
        UpdateRewardsState();
        SaveAchievements();
    }
}
Ссылок: 0
public void ButtonAdClaim()
{
    targetInventory.gold += 300;
    targetInventory.UpdateGold();
    lastClaimTimeAds = System.DateTime.UtcNow;
    button.SetActive(false);
    UpdateRewardsState();
}
```

Рисунок 4.27 – Кнопки для роботи оновлення досягнення та нагороди

Друге досягнення буде виконуватись за допомогою знищення ворогів на рівні (рис. 4.6). Таким же чином було розроблено ще одне досягнення за вбивство але вже більш сильних ворогів та досягнення за проходження рівнів.

```
Ссылок: 2
public void UpdateRewardsUI2()
{
    fillImageZombie.fillAmount = zombieLiteKilled / zombieLiteCount;
    zombieText.text = $"{zombieLiteKilled}/{zombieLiteCount}";
    if(zombieLiteKilled >= zombieLiteCount) { buttonReceive2.SetActive(true); }
}
Ссылок: 0
public void RewardCLaim2()
{
    zombieLiteKilled -= zombieLiteCount;
    buttonReceive2.SetActive(false);
    UpdateRewardsUI2();
}
```

Рисунок 4.28 – Оновлення другого досягнення та сбір нагороди

Останнє що було зроблено, це нагорода за щоденний вхід. Гравець може забрати бонус у вигляді валюти та наступний бонус можна буде забрати через 24 години. Для цього було розроблено три скрипта: Reward, RewardPref та DailyRewards [Додаток Б].

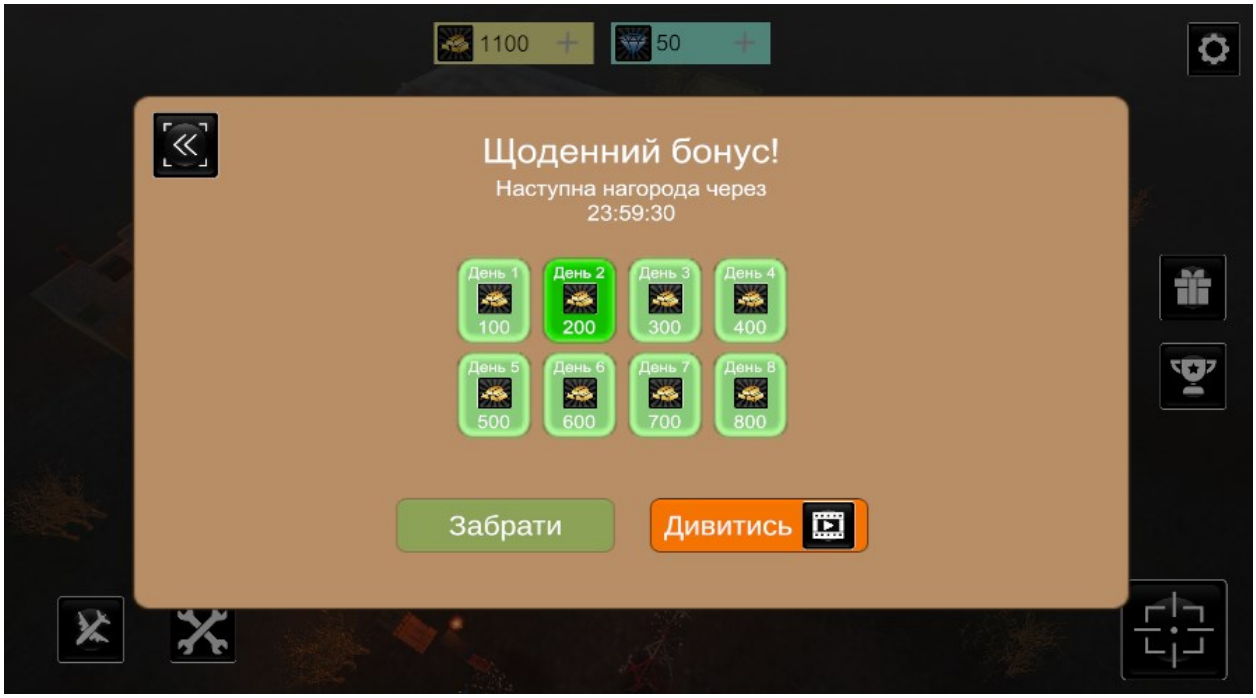

Рисунок 4.29 – Результат меню бонусу

Скрипт Reward використовуємо для типу нагороди та значень. RewardPref для того що відобразити значення на екран. А в DailyRewards основна логіка щоденної нагороди.

4.3 Програмна реалізація механіки рівня

Спочатку було створено логіку стрільби літака який буде стріляти різними видами зброї. Стрільбу реалізовано за допомогою скрипту Rocket в якому створюється об’єкт кулі, раніше створеного префабу кулі, яка летить у напрямку прицілу на землі [Додаток Г].

Рисунок 4.30 – Префаби для куль

Також було створено скрипт ButtonTypeGun для того щоб кулі були різні. Для цього на різні кнопки запуску снаряду було використано свій скрипт ButtonTypeGun із своїми унікальними параметрами і тому має різні види куль [Додаток Г].

Рисунок 4.31 – Стрільба з літака

Логіка стрільби полягає в тому що коли гравець натискає одну з кнопок то вмикається дозвіл на запуск, далі якщо час затримки перед стрільбою рівняється необхідній та немає перезарядки то відбувається постріл і віднімається кількість пострілів. Якщо пострілів нуль то вмикається перезарядка [Додаток Г].

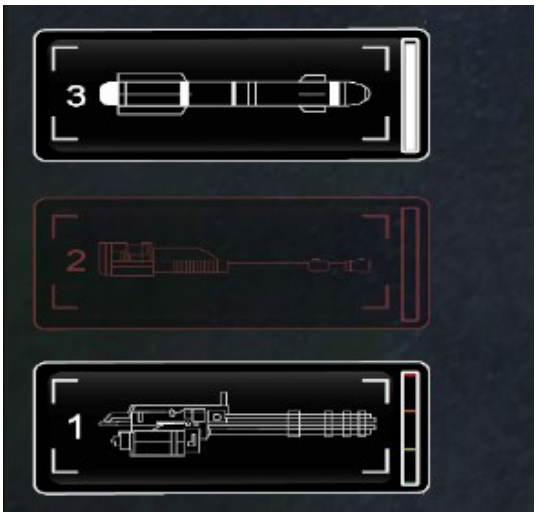

Рисунок 4.32 – Перезарядка зброї

Також окремо для кулемету було розроблено перегрів який досягає при постійній стрільбі з нього [Додаток Г].

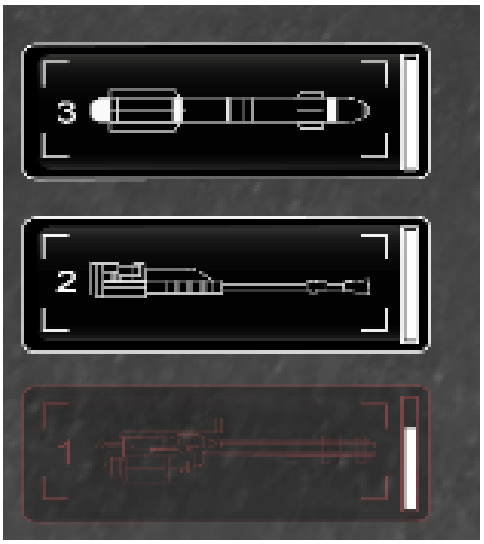

Рисунок 4.33 – Перегрів кулемету

Потім коли куля досягає землі було розроблено вибух, який є префабом для кожного типу зброї. Цей вибух перевіряє чи попав ворог у його радіус та наносить їм пошкодження [Додаток Д].

Рисунок 4.34 – Префаби для взривів

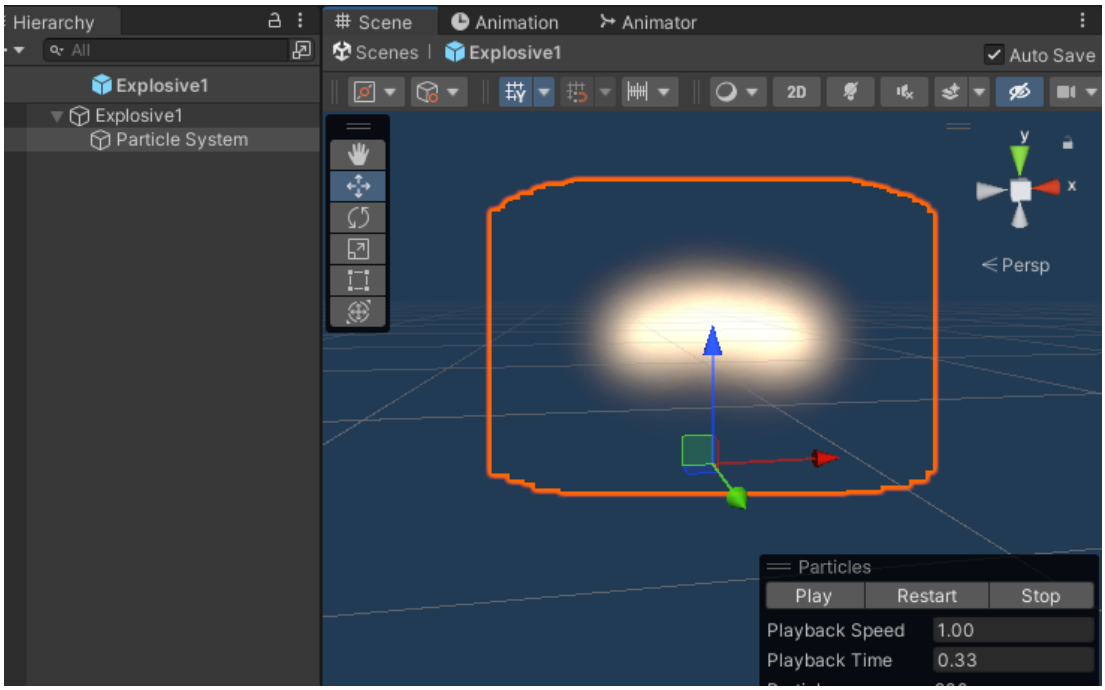

Рисунок 4.35 – Вигляд взриву через ParticleSystem

Рисунок 4.36 – Вигляд зіткнення кулі із землею на сцені

Далі потрібно було створити ворогів та їх логіку переміщення. Для цього було створено скрипт Spawn_Zombie в якому реалізовано спавн зомбі, з завчасно зроблених префабів зомбі, за вказаний час. Також створено метод який перевіряє чи всі вороги були знищені, якщо перевірка пройшла то вмикається панель перемоги [Додаток Е].

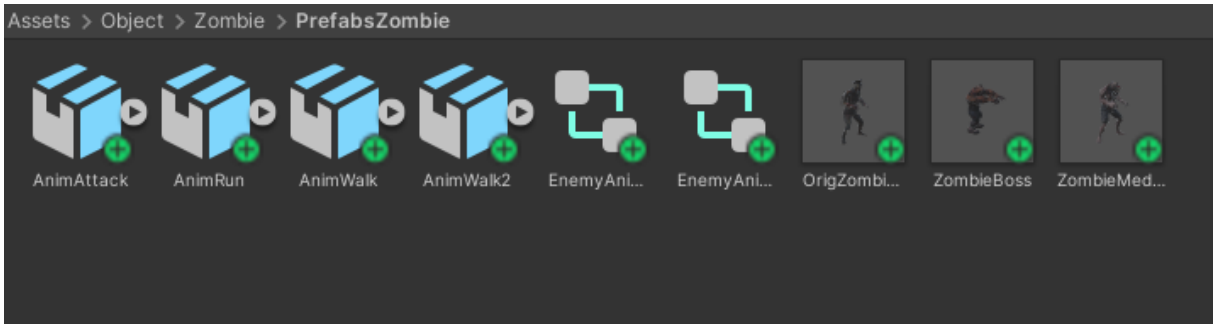

Рисунок 4.37 – Префаби зомбі та анімації

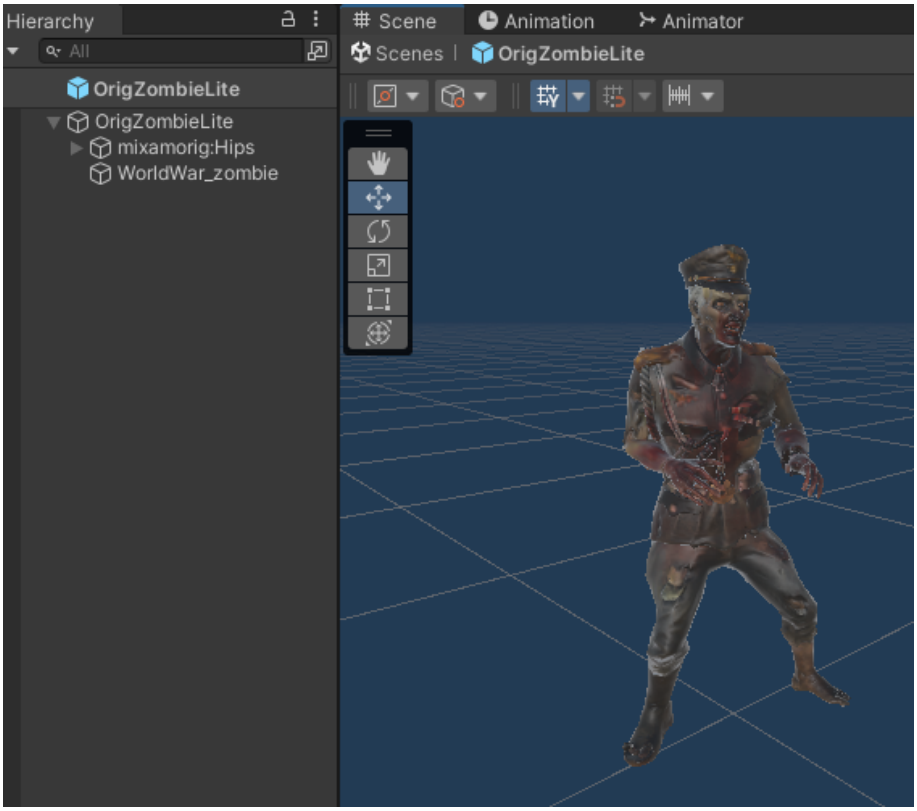

Рисунок 4.38 – Вигляд звичайного зомбі

Рисунок 4.39 – Заспавнені зомбі

Потім було розроблено скрипт ZombieMovement для переміщення цих ворогів та нанесення пошкоджень базі [Додаток Е].

Для переміщення було використано компонент NavMeshAgent, який використовується для створення площі по якій рухаються зомбі. У методі Update оброблюється інформація дистанції до цілі та рух до неї, в той же чай оброблюється метод чи готовий зомбі до атаки цілі. Також реалізований метод нанесення пошкоджень цілі, завдяки якому зомбі мають змогу перемогти користувача.

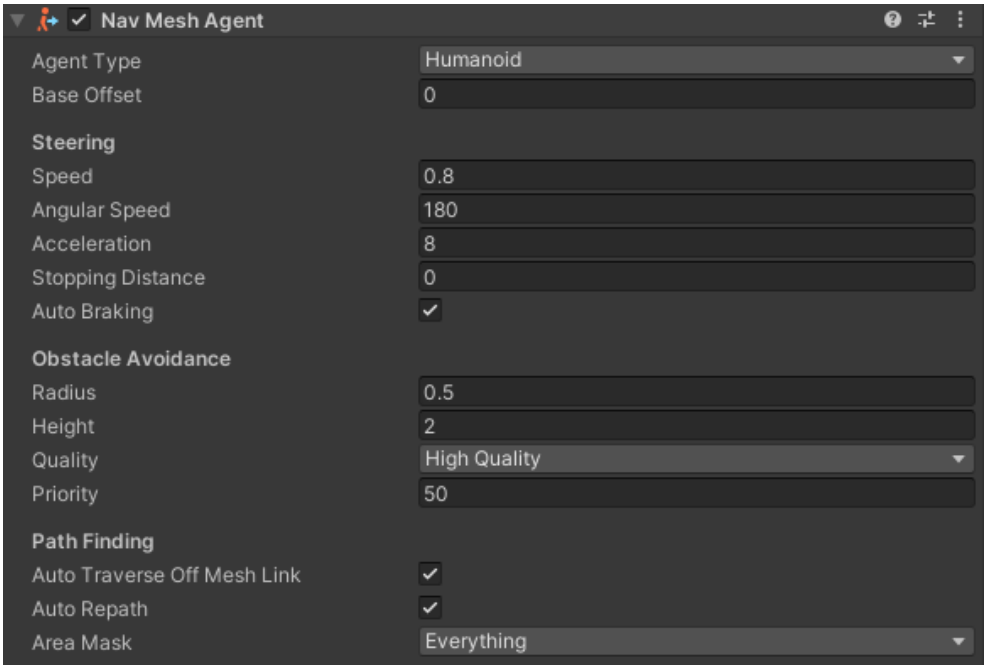

Рисунок 4.40 – Компонент NavMeshAgent на зомбі

Нарешті було створено скрипт Health для надання здоров’я для зомбі.

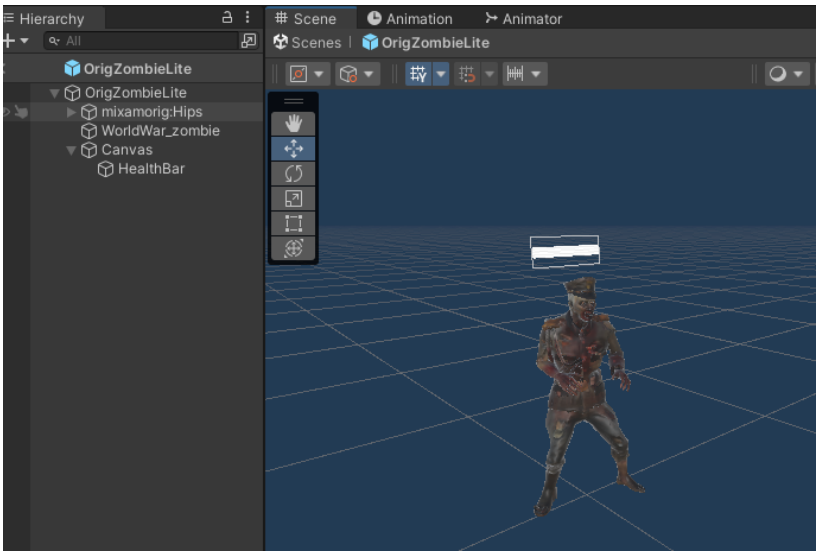

Рисунок 4.41 – Здоров’я зомбі

Для того щоб зомбі міг отримувати пошкодження було реалізовано механіку отримування пошкоджень зомбі від куль користувача. Також для того щоб гравець мав змогу перемогти було реалізовано метод смерті зомбі та знищення об’єкту зомбі зі сцени.

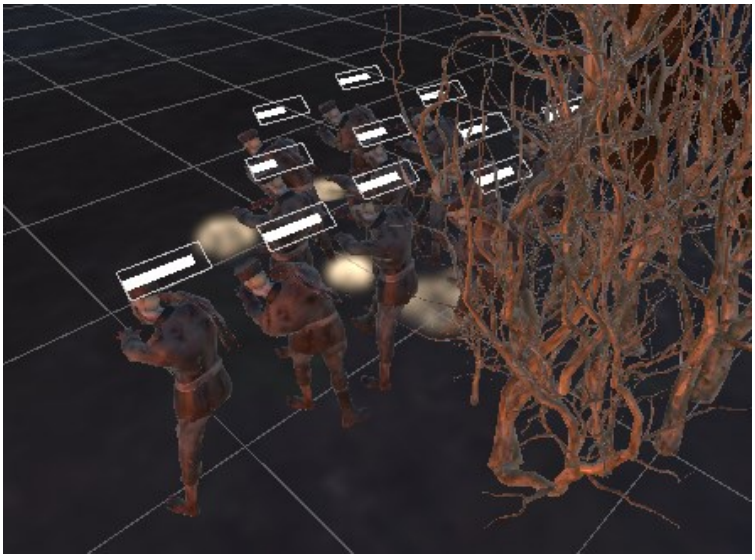

Рисунок 4.42 – Нанесення пошкоджень зомбі

Так як користувач грає ніби за оператора бойового літака, необхідно було реалізувати переміщення камери та прицілу для зміни точки місця призначення кулі [Додаток Ж].

Було реалізовано скрипт для камери CameraMovement де описано траєкторію руху камери симулюючи радіус руху літака довкола цілі. Також було зроблено так, щоб камера оператора(Користувача) завжди переслідувала таргет задля фіксації на місцевості прицілу користувача.

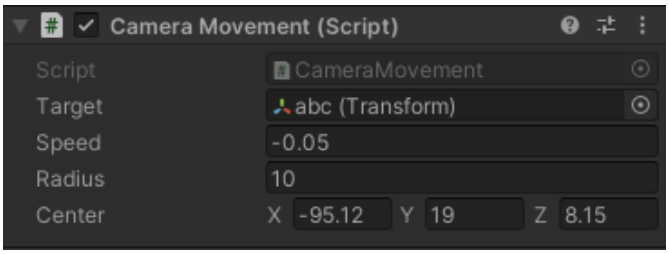

Рисунок 4.43 – Компонент скрипту на камері

Далі було розроблено скрипт TouchControl в якому реалізовано механіку яка повинна оброблювати рух пальця гравця по екрану в двовимірному просторі та переводити у рух об’єкт таргету у тривимірному просторі [Додаток Е]. Задля того щоб рух прицілу не був різким та грубим було використовувано вбудований метод Lerp() для згладжування переміщення.

Також у гравця має бути можливість приближати та віддаляти за допомогою рухів пальців по екрану камеру для зручного веденню вогню. Для цього було написано скрипт CameraZoom в якому описано механіку зуму прицілу.

Рисунок 4.44 – Приближене зображення

Для імітування інфрачервоного зображення камери у грі, було створено скрипт CameraEffect в якому реалізовано метод для накладування ефекту на зображення камери щоб було схоже на ведення бою з бойового літака [Додаток Е].

Рисунок 4.45 – Інфрачервоний ефект

Також було розроблено звуки у грі за допомогою компоненту Audio Source у інспекторі та підготовлені звукові дорожки для різних взаємодій. Таким чином, за допомогою коду: «audioSource.Play();» , де audioSource прив’язка до компоненту, відбувається програвання звуку. Було створено звуки для: натискання будь-якої клавіші, різних пострілів з літаку, самого літаку та взривів від пострілів.

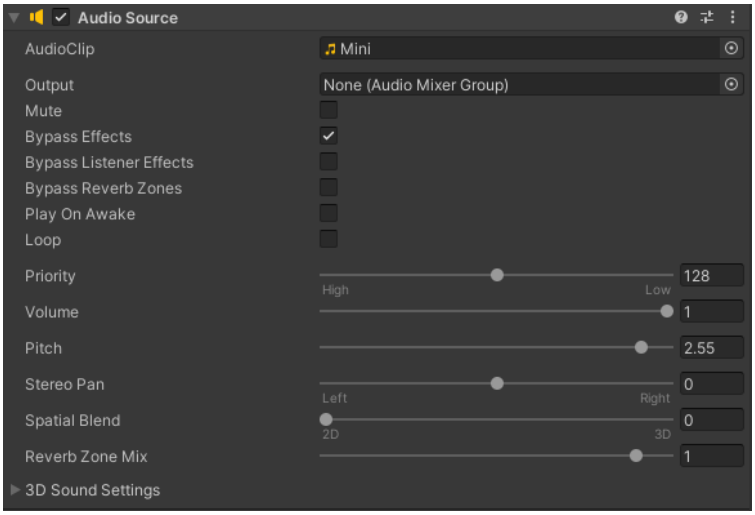

Рисунок 4.46 – Компонент Audio Source

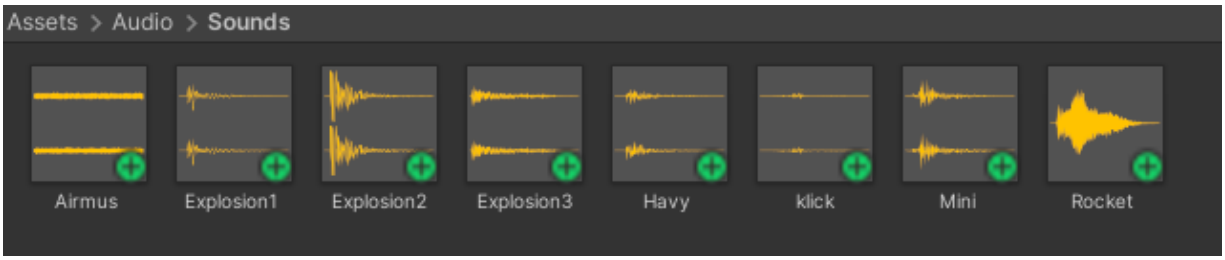

Рисунок 4.47 – Всі розроблені звуки

Накінець було створено пару анімацій для зомбі, а саме: переміщення та атака. Це було реалізовано за допомогою компонента Animator у інспекторі та підготовлених анімацій для заміни стану.

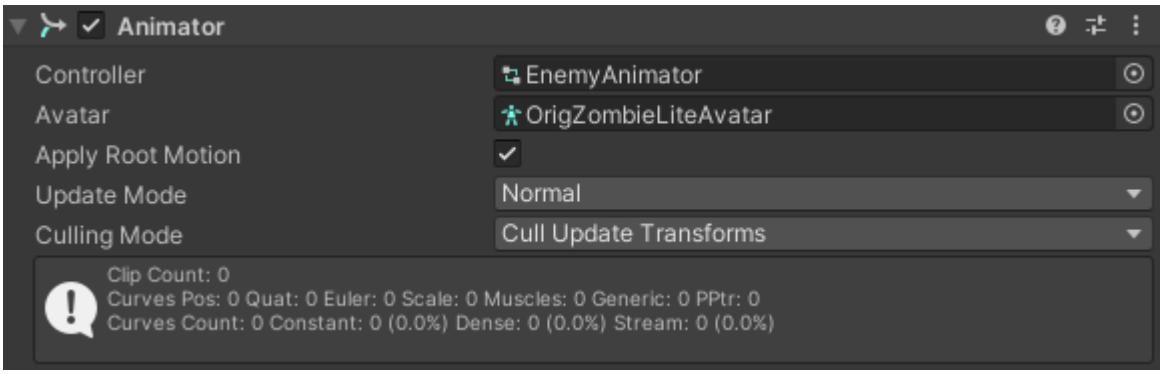

Рисунок 4.48 – Компонент Animator

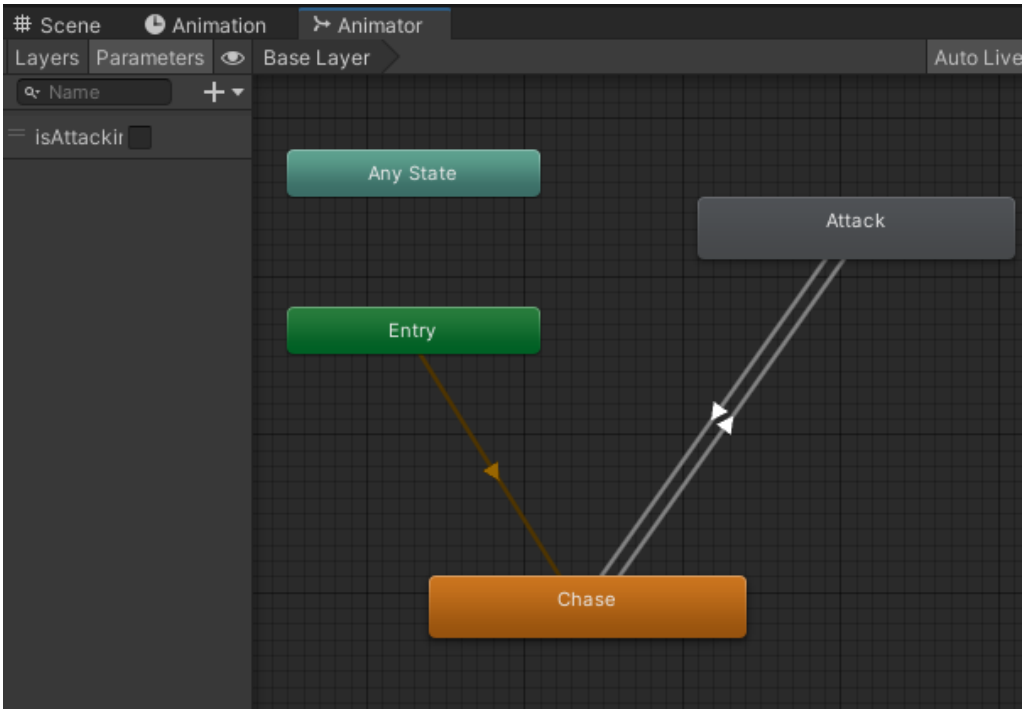

Рисунок 4.49 – Зміна станів анімацій у контролері

Висновки до розділу 4

У четвертому розділі було показано розробку ігрового дизайну та основних елементів користувацького інтерфейсу (головне меню, меню інвентарю, меню досягнень, ігровий інтерфейс, поліпшення зброї та вибір рівня). Було описано реалізацію ігрових механік роботи з інвентарем, досягненнями за проходження ігрових рівнів, можливість поліпшення придбаної зброї. Наведено реалізацію переміщень камери та взаємодію з аудіо супроводом у скриптах.

Інтерфейс реалізовано за допомогою інструмента UnityUI, Було використано елементи, які відповідають за певний функціонал для налаштування сцен та дизайну.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра проведено аналіз актуальності сфери Android-ігор і розглянуто декілька аналогів за обраним жанром. Були проаналізовані їх функціональні можливості, переваги та недоліки. Також був проведений системний аналіз для визначення цілей гри, користувачів та сценаріїв використання.

Крім того, було проаналізовано види засобів розробки для Android-ігор, зокрема Unity, Unreal Engine та Cry Engine, зосередившись на створенні 3D-ігор. Був проведений огляд програм для моделювання 3D-моделей для розробки гри, порівняння доступних варіантів та вибір найкращого. Також були розглянуті придатні мови програмування для розробки мобільних застосунків у середовищі розробки Unity.

У рамках роботи було реалізовано поставлене технічне завдання щодо створення ігрового застосунку. Для досягнення цієї мети були виконані такі завдання:

– Проведено аналіз актуальності сфери та ігор-аналогів обраного жанру.

– Визначено основні функції та створено специфікацію вимог до програмного забезпечення.

– Розроблено загальний алгоритм розробки гри.

– Створено діаграми сценаріїв використання, діаграми взаємодії, діаграми станів та переходів, діаграми діяльності, а також діаграми компонентів та розгортання.

– Реалізовано мобільний ігровий застосунок з використанням мови програмування C# у ігровому руші Unity.

Результатом кваліфікаційної роботи є готовий ігровий застосунок, який відповідає поставленим вимогам і досягає поставлених цілей. Цей застосунок може бути використаний на мобільних пристроях під управлінням операційної системи Android.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мануал розробки під Android. URL: https://docs.unity3d.com/ru/530/Manual/android-GettingStarted.html (дата звернення: 02.05.2023)

2. Методика розробки URL: http://um.co.ua/8/8-7/8-7041.html (дата звернення: 02.05.2023)

3. Розробка ігор за допомогою UNITY3D. URL: https://ec.europa.eu/programmes/erasmus-plus/project-result-content/7c7d9761-4402-467c-adab-5b09579cb8fd/2017_KNTU_Module2.pdf(дата звернення 02.05.2023)

4. Розробка мобільних додатків URL: https://azure.microsoft.com/ru-ru/resources/ (дата звернення: 02.05.2023)

5. Актуальність розробки мобільних додатків під Android. URL: https://ela.kpi.ua/bitstream/123456789/38101/1/Rozrobka_ta_analiz_KP.pdf(дата звернення: 02.05.2023)

6. Опис аналога Zombie Gunship Survival. URL: https://play.google.com/store/apps/details?id=com.flaregames.zgs(дата звернення: 02.05.2023)

7. Опис аналога Gunship Operator 3D. URL: https://play.google.com/store/apps/details?id=com.IronEqual.GunshipOperator3D (дата звернення 02.05.2023)

8. Опис аналога Gunship Operator 3D. URL: https://play.google.com/store/apps/details?id=com.IronEqual.GunshipOperator3D(дата звернення 02.05.2023)

9. Лекції по Unity. URL: https://unity.com/ru/learn/get-started(дата звернення: 02.05.2023)

10. Windows Presentation Foundation about C#. URL: https://dic.academic.ru/dic.nsf/ruwiki/132037 (дата звернення: 02.05.2023)

11. Розробка простих консольних додатків за допомогою Microsoft Visual Studio.URL: https://learn.microsoft.com/ru-ru/visualstudio/get-started/csharp/tutorial-console?view=vs-2022 (дата звернення: 02.05.2023)

12. Unreal Engine. URL: https://vokigames.com/ua/unreal-engine(дата звернення: 02.05.2023)

13. CryEngine URL: http://3das.com.ua/igrovij-dvizhok-vibrati-unity-udk-abo-cryengine/ (дата звернення: 02.05.2023)

14. Програма для моделювання 3д ігор - Cinema4D URL : https://uk.wikipedia.org/wiki/Cinema_4D (дата звернення: 02.05.2023)

15. Програма для моделювання 3д ігор - Blender URL: https://uk.wikipedia.org/wiki/Blender (дата звернення: 02.05.2023)

16. На яких мовах програмування пишуть мобільні ігри. URL: https://lviv.logos-academy.com/movy-programuvannya (дата звернення: 02.05.2023)

17. Найкраща програма для 3Д моделювання. URL: https://cgaward.com.ua/ua/publikacii/soft/best-software-for-3d (дата звернення: 02.05.2023)

18. Конструювання usecases та UML-діаграм. URL: https://moodle3.chmnu.edu.ua/course/view.php?id=27767#section-6 (дата звернення 02.05.2023)

Додаток А

Скрипт Achievements

```
private System.DateTime? lastClaimTimeAds
 {
    get
    {
       string data = PlayerPrefs.GetString("lastClaimTimeAds", null);
       if (!string.IsNullOrEmpty(data)) return System.DateTime.Parse(data);
       return null;
    }
    set
    {
       if (value != null) PlayerPrefs.SetString("lastClaimTimeAds", value.ToString());
       else PlayerPrefs.DeleteKey("lastClaimTimeAds");
    }
 }
private bool canClaimReward;
private float claimCooldown = 24f;
void Start()
 {
    UpdateRewardsState(); Redraw();
    StartCoroutine(RewardsStateUpdater());
 }
public IEnumerator RewardsStateUpdater()
 {
    while (true)
    {
       UpdateRewardsState();
       yield return new WaitForSeconds(1);
    }
 }
public void UpdateRewardsState()
 {
    canClaimReward = true;
    if (lastClaimTimeAds.HasValue)
    {
       var timeSpan = System.DateTime.UtcNow - lastClaimTimeAds.Value;
       if (timeSpan.TotalHours < claimCooldown) { canClaimReward = false; }
    }
    UpdateRewardsUI();
 }
public void UpdateRewardsUI()
 {
    if(adsWatched != adsCount)
    {
       fillImageAds.fillAmount = adsWatched / adsCount;
       adsText.text = $"{adsWatched}/{adsCount}"; button.SetActive(false);
    }
    else if (adsWatched == adsCount)
    {
       button.SetActive(canClaimReward);
       if (canClaimReward) adsText.text = "Забрати!";
       else
       {
           var nextClaimTime = lastClaimTimeAds.Value.AddHours(claimCooldown);
           var currentClaimCooldown = nextClaimTime - System.DateTime.UtcNow;
           remainingTime = currentClaimCooldown.TotalSeconds;
           string CD = $"{currentClaimCooldown.Hours:D2}:{currentClaimCooldown.
Minutes:D2}:{currentClaimCooldown.Seconds:D2}"; adsText.text = $"{CD}";
           if((int)remainingTime <= 0) { adsWatched = 0; SaveAchievements(); }
       }
     }
```

```
}
```

Додаток Б

Скрипти щоденної нагороди

```
public class RewardPref : MonoBehaviour
{
    [SerializeField]
    private Image background;
    [SerializeField]
    private Color defaultColor;
    [SerializeField]
    private Color currentStreakColor;

    [SerializeField]
    private Text dayText;
    [SerializeField]
    private Text rewardValue;

    [SerializeField]
    private Image rewardIcon;
    [SerializeField]
    private Sprite rewardGold;

    public void SetRewardData(int day, int currentStreak, Reward reward)
    {
        dayText.text = $"День {day + 1}";

        rewardIcon.sprite = rewardGold;
        rewardValue.text = reward.Value.ToString();

        background.color = day == currentStreak ? currentStreakColor : defaultColor;
    }
}

public class Reward
{
    public enum RewardType
    {
        GOLD
    }

    public RewardType Type;
    public int Value;
    public string Name;
}

public class DailyRewards : MonoBehaviour
{
    [SerializeField]
    private Text status;
    [SerializeField]
    private Button claimBtn;

    [SerializeField]
    private RewardPref rewardPref;
    [SerializeField]
    private Transform rewardsGrid;

    [SerializeField]
    private List<Reward> rewards;

    private int currentStreak
```

```
{
    get => PlayerPrefs.GetInt("currentStreak", 0);
    set => PlayerPrefs.SetInt("currentStreak", value);
}
private System.DateTime? lastClaimTime
{
    get
    {
        string data = PlayerPrefs.GetString("lastClaimedTime", null);

        if (!string.IsNullOrEmpty(data))
            return System.DateTime.Parse(data);

        return null;
    }
    set
    {
        if (value != null)
            PlayerPrefs.SetString("lastClaimedTime", value.ToString());
        else
            PlayerPrefs.DeleteKey("lastClaimedTime");
    }
}

private bool canClaimReward;
private int maxStreakCount = 8;
private float claimCooldown = 24f;
private float claimDeadline = 48f;

private List<RewardPref> rewardPrefabs;

void Start()
{
    InitPrefabs();
    StartCoroutine(RewardsStateUpdater());
}

public void InitPrefabs()
{
    rewardPrefabs = new List<RewardPref>();
    for (int i = 0; i < maxStreakCount; i++)
        rewardPrefabs.Add(Instantiate(rewardPref, rewardsGrid, false));
}

public IEnumerator RewardsStateUpdater()
{
    while (true)
    {
        UpdateRewardsState();
        yield return new WaitForSeconds(1);
    }
}

public void UpdateRewardsState()
{
    canClaimReward = true;

    if (lastClaimTime.HasValue)
    {
        var timeSpan = System.DateTime.UtcNow - lastClaimTime.Value;

        if (timeSpan.TotalHours > claimDeadline)
        {
            lastClaimTime = null;
            currentStreak = 0;
```

```
        }
        else if (timeSpan.TotalHours < claimCooldown)
            canClaimReward = false;
    }
    UpdateRewardsUI();
  }

  public void UpdateRewardsUI()
  {
    claimBtn.interactable = canClaimReward;

    if (canClaimReward)
      status.text = "Заберіть свій подарунок!";
    else
    {
      var nextClaimTime = lastClaimTime.Value.AddHours(claimCooldown);
      var currentClaimCooldown = nextClaimTime - System.DateTime.UtcNow;

      string CD =
$"{currentClaimCooldown.Hours:D2}:{currentClaimCooldown.Minutes:D2}:{currentClaimCooldown.Seconds:D2}";

      status.text = $"Наступна нагорода через {CD}";
    }

    for (int i = 0; i < rewardPrefabs.Count; i++)
      rewardPrefabs[i].SetRewardData(i, currentStreak, rewards[i]);
  }

  public void ClaimReward()
  {
    if (!canClaimReward)
      return;

    var reward = rewards[currentStreak];
    switch (reward.Type)
    {
      case Reward.RewardType.GOLD:
        FindObjectOfType<Inventory>().AddGold(reward.Value);
        break;
    }

    lastClaimTime = System.DateTime.UtcNow;
    currentStreak = (currentStreak + 1) % maxStreakCount;

    UpdateRewardsState();
  }
}
```

Додаток В

Скрипт AdsManager

```
public class AdsManager : MonoBehaviour, IUnityAdsInitializationListener
{
  [SerializeField] private bool testMod = true;

  private string gameID = "5318323";

  void Start()
  {
    if (Advertisement.isSupported)
    {
      Advertisement.Initialize(gameID, testMod, this);
    }
  }

  public void ShowAds(string adsID)
  {
    Advertisement.Show(adsID);
  }
  public void OnInitializationComplete()
  {
    Debug.Log("Unity Ads initialization complete.");
  }

  public void OnInitializationFailed(UnityAdsInitializationError error, string message)
  {
    Debug.LogError("Unity Ads initialization failed: " + error.ToString() + " - " + message);
  }
}
```

Додаток Г

Скрипти механіки стрільби

```
public class Rocket : MonoBehaviour
{
  void Update()
  {
    if (checkFire == true )
    {
      if ( BTG[currentRocket].bulletCountCurrent > 0)
      {
        LaunchRocket(currentRocket);
      }
      if(currentRocket != 0 )
      {
        BTG[currentRocket].CheckCurrentBullet();
      }
    }
    Gun.LookAt(targ.transform.position );
  }
  public void LaunchRocket(int current)
  {
    current = currentRocket;
    Vector3 spreadOffset = new Vector3();
    if (current != 2) { spreadOffset = new Vector3(Random.Range(-bulletSpread, bulletSpread), Random.Range(-bulletSpread,
bulletSpread), Random.Range(-bulletSpread, bulletSpread)); }
    GameObject rocket = Instantiate(rocketPrefab[current], rocketSpawnPoint.position , rocketSpawnPoint.rotation);

    Rigidbody rb = rocket.GetComponent<Rigidbody>();
    rb.transform.LookAt(targ.transform.position);
    rb.velocity = (transform.forward + spreadOffset) * rocketSpeed;
    if(current == 0) { Destroy(rocket, 3f); }
  }
}

public class ButtonTypeGun : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
  void Update()
  {
    if (fireDelay <= timeFireDelay2)
    {
      fireDelay += Time.deltaTime;
    }
    if (reloadTimeCurrent > 0.004f)
    {
      reloadTimeCurrent -= Time.deltaTime * 1.1f;
      this.OverheatTimeDown();
      GetComponent<Button>().interactable = false;
      GetComponent<Image>().color = targetColor;
    }
    if (reloadTimeCurrent <= 0.004f && bulletCountCurrent > 0)
    {
      GetComponent<Button>().interactable = true;
      GetComponent<Image>().color = defoultColor;
      if (checkFire == true)
      {
        if (fireDelay >= timeFireDelay2)
        {
          RS.checkFire = true;
          fireDelay = 0;
        }
        else RS.checkFire = false;
        if (this.currentFire == 0) { this.OverheatTimeUp(); }
```

```
        }
        else if (checkFire == false && this.currentFire == 0) { this.OverheatTimeDown(); }
      }
      SetAmmoStatus();
    }
    public void OnPointerDown(PointerEventData eventData)
    {
        RS.currentRocket = currentFire;
        RS.rocketSpeed = rocketSpeed2;
        checkFire = true;
    }
    public void OnPointerUp(PointerEventData eventData)
    {
      checkFire = false;
      RS.checkFire = false;
    }
    public void CheckCurrentBullet()
    {
      if (bulletCountCurrent == 0)
      {
        RS.checkFire = false;
        reloadTimeCurrent = reloadTime;
        if (currentFire != 0)
        {
          StartCoroutine(DelayedMethod(reloadTime));
        }
      }else if (bulletCountCurrent > 0) { bulletCountCurrent--; }
    }
    public void OverheatTimeUp()
    {
      if (overheatTime < overheatTimeMax) { overheatTime += Time.deltaTime * 2; }
      else
      {
        reloadTimeCurrent = reloadTime;
        checkFire = false;
        RS.checkFire = false;
      }
    }
    public void OverheatTimeDown()
    {
      if(overheatTime > 0.004f) { overheatTime -= Time.deltaTime * 1.1f; }
    }
    public void SetAmmoStatus()
    {
      if(currentFire == 0)
      {
        fillImage.fillAmount = overheatTime / overheatTimeMax;
      }else fillImage.fillAmount = bulletCountCurrent / bulletCountMax;
    }
    IEnumerator DelayedMethod(float delay)
    {
      yield return new WaitForSeconds(delay);
      if(clipSize > 0)
      {
        bulletCountCurrent = bulletCountMax;
        clipSize--;
      }
    }
}
```

Додаток Д

Скрипти механіки взриву

```
public class RocketDestroy : MonoBehaviour
{
   Rocket RT;
   public int currRocket;

   void Start()
   {
      RT = FindObjectOfType<Rocket>();
      currRocket = RT.currentRocket;
   }

   private void OnTriggerEnter(Collider other)
   {
      if (other.gameObject.tag == "Ground")
      {
         GameObject explosive = Instantiate(RT.explosivePrefab[currRocket], gameObject.transform.position,
RT.explosivePrefab[currRocket].transform.rotation);

         Destroy(gameObject);
         Destroy(explosive, 1f);
      }
}
}public class Explosive : MonoBehaviour
{
   public int explosivePower;
   public float explosiveRange;
   public float damage;
   public float damageRange;
   void Start()
   {
      Collider[] hitColliders = Physics.OverlapSphere(transform.position, damageRange);

      foreach (var iter in hitColliders)
      {
         var navMesh = iter.GetComponent<NavMeshAgent>();

         if (navMesh != null )
         {
            if(navMesh.gameObject.CompareTag("zombie"))
            {
               Health health = navMesh.gameObject.GetComponent<Health>();
               if (health.enabled == true)
               {
                  health.TakeDamage(damage);
                  if (health.currentHealth <= 0)
                  {
                     Rigidbody[] rigidbody= navMesh.gameObject.GetComponentsInChildren<Rigidbody>();
                     foreach (var i in rigidbody)
                     {
                        i.AddExplosionForce(explosivePower, transform.position, explosiveRange);
                     }
                     health.Die();
                  }
               }
            }
         }
      }
   }
}
```

Додаток Е

Скрипти механіки ворогів

```
public class Spawn_Zombie : MonoBehaviour
{
  void Start()
  {
    StartCoroutine(ZombieKilledUpdater());
  }

  void Update()
  {
    if (waitingTimer > 0) { waitingTimer -= Time.deltaTime; } else
    {
      spawnTimer1 += Time.deltaTime;
      spawnTimer2 += Time.deltaTime;
      spawnTimer3 += Time.deltaTime;

      if (spawnTimer1 >= spawnInterval1 && spawnCountLite > 0)
      {
        SpawnZombieLite();
        spawnTimer1 = 0f;
      }else if (spawnTimer2 >= spawnInterval2 && spawnCountMedium > 0)
      {
        SpawnZombieMedium();
        spawnTimer2 = 0f;
      }else if (spawnTimer3 >= spawnInterval3 && spawnCountBoss > 0)
      {
        SpawnZombieBoss();
        spawnTimer3 = 0f;
      }
    }
  }

  void SpawnZombieLite()
  {
    GameObject newZombie = Instantiate(zombiePrefab1, spawnPoint.transform.position, Quaternion.identity);
    spawnCountLite--;
    spawnedZombie++;
  }

  void SpawnZombieMedium()
  {
    GameObject newZombie = Instantiate(zombiePrefab2, spawnPoint.transform.position, Quaternion.identity);
    spawnCountMedium--;
    spawnedZombie++;
  }

  void SpawnZombieBoss()
  {
    GameObject newZombie = Instantiate(zombiePrefab3, spawnPoint.transform.position, Quaternion.identity);
    spawnCountBoss--;
    spawnedZombie++;
  }

  private IEnumerator ZombieKilledUpdater()
  {
    while (true)
    {
      UpdateKilledZombie();
      yield return new WaitForSeconds(waitingSeconds);
    }
  }
```

```
    private void UpdateKilledZombie()
    {
        waitingSeconds = 1.5f;

        if (zombieLiteKilled - achievementsData.zombieLiteKilledData == countLiteMax && zombieKilled == spawnedZombie)
        {
            SaveAchievements();
            winPanel.SetActive(true);
        }
    }
}

public class ZombieMovement : MonoBehaviour
{
    public GameObject targetPoint;
    NavMeshAgent agent;
    Animator anim;

    float attackRange = 2;
    public float damage;

    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        targetPoint = GameObject.FindGameObjectWithTag("Target");
        anim = GetComponent<Animator>();
    }

    void Update()
    {
        float distance = Vector3.Distance(agent.gameObject.transform.position, targetPoint.transform.position);
        agent.destination = targetPoint.transform.position;

        if (distance < attackRange)
        {
            anim.SetBool("isAttacking", true);
        }
    }

    public void Damage()
    {
        targetPoint.GetComponentInParent<Health>().TakeDamage(damage);
        if (targetPoint.GetComponentInParent<Health>().currentHealth <= 0)
        {
            GameOver();
        }
    }

    public void GameOver()
    {
        Time.timeScale = 0f;
        GameObject.FindObjectOfType<IdLevel>().GameOverPanelEnable();
    }
}
public class Health : MonoBehaviour
{
    public HealsBar healthBar;
    public Camera targ;
    public float maxHealth;

    public float currentHealth;
    public int id;
```

```
    private void Start()
    {
        currentHealth = maxHealth;
        targ = FindObjectOfType<Camera>();
    }

    private void Update()
    {
        healthBar.gameObject.transform.LookAt(targ.transform.position);
    }

    public void TakeDamage(float damageAmount)
    {
        currentHealth -= damageAmount;
        healthBar.SetHealth(currentHealth, maxHealth);

        if (currentHealth <= 0)
        {
            if(id == 0)
            {
                GameObject.FindGameObjectWithTag("Spawner").GetComponent<Spawn_Zombie>().zombieLiteKilled++;
            }else if (id == 2) {
GameObject.FindGameObjectWithTag("Spawner").GetComponent<Spawn_Zombie>().zombieBossKilled++; }
            GameObject.FindGameObjectWithTag("Spawner").GetComponent<Spawn_Zombie>().zombieKilled++;
        }
    }

    public void Die()
    {
      Destroy(gameObject, 1.5f);
      enabled = false;
    }
}

public class HealsBar : MonoBehaviour
{
    public Image fillImage;
    public void SetHealth(float healthPercent, float health)
    {
        fillImage.fillAmount = healthPercent / health;
    }
}
```

Додаток Ж

Скрипти механіки камери

```
public class CameraMovement : MonoBehaviour
{
  public Transform target;
  public float speed = 0.2f;
  public float radius = 2f;
  public Vector3 center = Vector3.zero;
  private float angle = 0f;

  void Update()
  {
    angle += speed * Time.deltaTime;
    Vector3 offset = new Vector3(Mathf.Sin(angle), 0f, Mathf.Cos(angle)) * radius;

    transform.position = center + offset;
    transform.LookAt(target);
  }
}

public class TouchControl : MonoBehaviour, IDragHandler
{
  public Transform ABC;
  public float movementSpeed = 5f;

  public void OnDrag(PointerEventData eventData)
  {
    float deltaX = eventData.delta.x;
    float deltaY = eventData.delta.y;
    float deltaZ = 0f;

    Quaternion cameraRotation = Camera.main.transform.rotation;

    float cameraAngle = cameraRotation.eulerAngles.y;

    Vector3 newRotation = new Vector3(0f, cameraAngle, 0f);
    Quaternion rotationY = Quaternion.AngleAxis(cameraAngle, Vector3.up);

    ABC.rotation = rotationY;

    Vector3 targetPosition = ABC.position + rotationY * new Vector3(deltaX, deltaZ, deltaY) * movementSpeed;
    ABC.position = Vector3.Lerp(ABC.position, targetPosition, Time.deltaTime);
  }
}

public class CameraZoom : MonoBehaviour
{
  public Camera cameraToZoom;
  public float zoomSpeed = 0.5f;
  public float minFOV = 1f;
  public float maxFOV = 100f;

  void Update()
  {
    if (Input.touchCount == 2)
    {
      Touch touchZero = Input.GetTouch(0);
      Touch touchOne = Input.GetTouch(1);

      Vector2 touchZeroPrevPos = touchZero.position - touchZero.deltaPosition;
      Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;
```

```
            float prevTouchDeltaMag = (touchZeroPrevPos - touchOnePrevPos).magnitude;
            float touchDeltaMag = (touchZero.position - touchOne.position).magnitude;

            float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;

            float newFOV = cameraToZoom.fieldOfView + deltaMagnitudeDiff * zoomSpeed;
            cameraToZoom.fieldOfView = Mathf.Clamp(newFOV, minFOV, maxFOV);
        }
        if (Input.GetKey(KeyCode.W))
        {
            cameraToZoom.fieldOfView = Mathf.Clamp(zoomSpeed, minFOV, maxFOV);
        }
        if (Input.GetKey(KeyCode.S))
        {
            cameraToZoom.fieldOfView = Mathf.Clamp(zoomSpeed, maxFOV, minFOV);
        }
    }
}

public class CameraEffect : MonoBehaviour
{
    public Material material;

    private void OnRenderImage(RenderTexture source, RenderTexture destination)
    {
        if(material == null)
        {
            Graphics.Blit(source, destination);
            return;
        }
        Graphics.Blit(source, destination, material);
    }
}
```