Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___»_____2024р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»

на тему: Ігровий застосунок “Оборона стратегічного об’єкту”

Студентка групи КН-41 Анастасія БІЛА _____
(підпис)

Керівник роботи Ольга ПОСВЯТОВСЬКА _____
(підпис)

Консультанти:

з техніко-економічного обґрунтування Любов МЕЛЕНЧУК _____
(підпис)

нормоконтролер Надія ГАВРИШКІВ _____
(підпис)

Тернопіль – 2024

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
підпис
«___»_____ 2023р.

ЗАВДАННЯ

на кваліфікаційну роботу
на здобуття освітньо-професійного рівня «фаховий молодший бакалавр»
студентці Білій Анастасії Леонідівні
(прізвище, ім’я та по-батькові студента)

1. Тема роботи Ігровий застосунок “Оборона стратегічного об’єкту”
затверджена наказом по коледжу від “27” листопада 2023 р., № ____
2. Термін здачі студентом завершеної роботи “___” _____ 2024 р
3. Вихідні дані до роботи: аналіз існуючих ігрових застосунків жанру шутер від першої особи з елементами виживання, дослідження технологій розробки, вимоги до застосунків такого типу.
4. Перелік питань, які повинні бути розроблені:
а) основна частина оглял існуючих рішень і постановка завдання, проєктування, реалізація і тестування застосунку.
а) техніко-економічне обґрунтування аналіз ринку, розрахунок витрат на проєктування, обґрунтування необхідності розробки.
5. Перелік графічного матеріалу UML-діаграми прецедентів та класів, BPMN-діаграма проходження.
6. Консультанти роботи: Меленчук Л. І.

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	Меленчук Л. І. (вчена ступінь, звання П.І.Б. консультанта)		

КАЛЕНДАРНИЙ ПЛАН
Виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми та ознайомлення з вимогами до кваліфікаційної роботи.	23.11.23 р.	08.12.23 р.
2.	Огляд типових рішень та написання відповідного розділу роботи.	11.12.23 р.	25.12.23 р.
3.	Дослідження технологій реалізації та написання відповідного розділу роботи.	26.12.23 р.	15.01.24 р.
4.	Розробка функціональних вимог до роботи та робота над структурою програмного продукту. Написання відповідного розділу роботи.	16.01.24 р.	02.02.24 р.
5.	Встановлення та налаштування середовища реалізації та написання відповідного розділу роботи.	05.02.24 р.	23.02.24 р.
6.	Проєктування програмного засобу (функціоналу, інтерфейсу) та написання відповідного розділу роботи.	26.02.24 р.	11.03.24 р.
7.	Реалізація та налаштування програмного засобу та написання відповідного розділу роботи.	12.03.24 р.	19.04.24 р.
8.	Доопрацювання модулів.	22.04.24 р.	01.05.24 р.
9.	Тестування та налагодження програмного продукту.	02.05.24 р.	20.05.24 р.
10.	Опрацювання економічного розділу кваліфікаційної роботи та оформлення спеціального розділу.	21.05.24 р.	07.06.24 р.
11.	Робота над оформленням пояснювальної записки.	10.06.24 р.	16.06.24 р.
12.	Попередній захист кваліфікаційної роботи та доопрацювання.	17.06.24 р.	17.06.24 р.
13.	Підготовка до захисту кваліфікаційної роботи.	17.06.24 р.	24.06.24 р.
14.	Захист кваліфікаційної роботи.	27.06.24 р.	27.06.24 р.

7. Дата видачі завдання: “___” _____ 2023 р. Керівник _____/

Завдання прийняла до виконання _____/_____

Реферат

Кваліфікаційна робота. Ігровий застосунок «Оборона стратегічного об’єкту». 83 сторінок, 27 рисунків, 12 таблиць, 6 додатків, 17 джерел.

Об’єкт дослідження – існуючі ігрові застосунки спроєктовані і розроблені в жанрі шутер від першої особи з елементами виживання та засоби їх реалізації.

Метою кваліфікаційної роботи є розробка ігрового застосунку у жанрі шутер від першої особи з елементами виживання з використанням тривимірної графіки, орієнтованого на платформу ПК.

Завданнями роботи є аналіз існуючих ігрових застосунків у відповідному жанрі, дослідження засобів розробки ігрових механік та інтерфейсів, проєктування та реалізація системи управління ворогами, розробка користувацького інтерфейсу.

Для розробки системи було використано ігровий рушій Unity та програмне забезпечення Microsoft Visual Studio. Реалізація ігрового застосунку виконана на мові програмування C#.

У якості основного контенту гри розроблено систему хвиль ворогів, що атакують стратегічний об'єкт та головного героя, поведінка та навігація яких відбувається за допомогою штучного інтелекту. Розроблено моделі зброї з різним рівнем ушкодження, анімації, звукові ефекти та зручний користувацький інтерфейс.

Розроблений ігровий застосунок "Оборона стратегічного об'єкту" відповідає поставленим вимогам і може бути використаний для комерційного розповсюдження.

ІГРОВИЙ ЗАСТОСУНОК, ШУТЕР ВІД ПЕРШОЇ ОСОБИ, ВИЖИВАННЯ, UNITY, C#, 3D-ГРАФІКА, ШТУЧНИЙ ІНТЕЛЕКТ, ІНТЕРФЕЙС КОРИСТУВАЧА, UML-ДІАГРАМИ

Abstract

Qualification work. Game application "Defense of a strategic object". 83 pages, 27 figures, 12 tables, 6 appendices, 17 sources.

The object of research is existing game applications designed and developed in the genre of first-person shooter with survival elements and means of their implementation.

The purpose of the qualification work is to develop a game application in the genre of first-person shooter with survival elements using three-dimensional graphics, focused on the PC platform.

The tasks of the work are to analyze existing game applications in the relevant genre, research tools for developing game mechanics and interfaces, design and implement an enemy management system, and develop a user interface.

The system was developed using the Unity game engine and Microsoft Visual Studio software. The game application was implemented in the C# programming language.

The main content of the game is a system of waves of enemies attacking a strategic object and the protagonist, whose behavior and navigation are controlled by artificial intelligence. We developed weapon models with different levels of damage, animations, sound effects, and a user-friendly interface.

The developed game application "Defense of a Strategic Object" meets the requirements and can be used for commercial distribution.

GAMING APPLICATION, FIRST-PERSON SHOOTER, SURVIVAL, UNITY, C#, 3D GRAPHICS, ARTIFICIAL INTELLIGENCE, USER INTERFACE, UML DIAGRAMS

ЗМІСТ

					КР.КН 24.544.01.000 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Біла А. Л.			Ігровий застосунок «Оборона стратегічного об'єкту»	Літ.	Арк. 5 / Аркушів 63
Перев.		Посвятовська О. Б.					
Реценз.		Сиротюк О. Б.				ГФК.ВКТ. КН - 41	
Н.контр.		Гаврішків Н. Г..					
Зав. від.		Стефурак Н.А.					

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПК – персональний комп’ютер.

ОС – операційна система.

FPS – шутер від першої особи.

TPS – шутер від третьої особи.

AI – Artificial Intelligence.

UI – User Interface.

UML – Unified Modeling Language.

BPMN – Business Process Modeling Notation.

macOS – операційна система корпорації Apple.

Xbox – гральна консоль, розроблена компанією Microsoft.

PlayStation – гральна консоль, розроблена компанією Sony.

NPC – Non-Player Character.

UDP – User Datagram Protocol.

DirectX – набір інтерфейсів прикладного програмування для виконання завдань, пов’язаних із мультимедіа.

IDE – Integrated Development Environment.

ПДВ – Податок на додану вартість.

ЄСВ – Єдиний соціальний внесок.

ПДФО – Податок на доходи фізичних осіб.

ВСТУП

У сучасному світі комп’ютерні ігри не тільки визнані як засіб розваг, але і як ефективний інструмент для розвитку креативності, стратегічного мислення та вирішення проблем. У цьому контексті цікавим є поєднання елементів шутера від першої особи, виживання та стратегії захисту вежі, втілене у форматі 3-D гри з лінійним сюжетом.

Метою роботи є створення захопливого ігрового середовища, де гравець відчує адреналін бою та відповідальність за оборону важливого об'єкту. Інноваційний геймплей буде привертати увагу гравців і забезпечувати їм приємний досвід від гри. Така діяльність стимулюватиме гравців до розвитку тактичних та стратегічних навичок.

Основна проблема при створенні таких ігрових програм полягає в пошуку оптимального балансу між інтенсивністю шутер-елементів та стратегічними аспектами захисту об’єкту. Розробка гри, яка здатна зберегти баланс між особливостями жанрів, виглядає перспективним завданням, щоб задовольнити різні смаки гравців.

Актуальність роботи обґрунтовується високим попитом на ігри, які поєднують в собі різні жанри, що дозволяє гравцям розвивати та вдосконалювати свої навички в кількох напрямках. Крім того, за останні роки геймерська аудиторія стала більш вимогливою, шукаючи новаторські та цікаві геймплейні рішення. Усе це робить дану комп’ютерну гру актуальною в сучасній галузі розробки комп’ютерних ігор.

1 ДОСЛІДЖЕННЯ КЛАСИФІКАЦІЙ ТА АНАЛІЗ ІСНУЮЧИХ РОЗРОБОК КОМП’ЮТЕРНИХ ІГОР

1.1 Аналіз комп’ютерних ігор і загальна характеристика

Комп'ютерні ігри – це набір вказівок у вигляді слів, цифр, символів, кодів, схем або інших форм, які виражені у машинозчитувальному форматі та використовуються для керування комп'ютером [1]. У кожній грі є своя стратегія, дія та фантазія, що робить її унікальною та цікавою. Призначена для взаємодії між людиною (групою людей) і комп'ютером або між кількома людьми, які використовують комп'ютер з метою розваги, освіти або навчання [2]. У комп'ютерних іграх спеціалізоване програмне забезпечення за певними алгоритмами створює імітацію безпосередньої взаємодії у віртуальному просторі між персонажем і користувачем (або групою користувачів). Гравець впливає на гру за допомогою клавіатури, миші або джойстика, спостерігаючи за станом гри на екрані монітора. Комп'ютерні ігри схожі на відеоігри, але, на відміну від останніх, не потребують спеціального технічного обладнання (ігрової консолі або приставки).

Комп'ютерні ігри в основному забезпечують розвагу та веселощі, але також розвивають зорово-моторну координацію, стратегічне мислення та навички вирішення завдань. Онлайн-ігри часто надають можливість взаємодії з іншими гравцями по всьому світу, що сприяє формуванню соціальних навичок, командної роботи. Окрім цього індустрія комп’ютерних ігор є великим економічним сегментом, який створює робочі місця, привертає інвестиції та генерує значний обсяг доходу. Це підтверджує щорічна статистика Steam, яку публікує американська компанія Valve Corporation [3]: у 2023 році понад 500 ігрових проєктів заробили більше 3 мільйонів доларів США валового доходу.

Варто взяти до уваги і кіберспорт, визнаний офіційним видом спорту, як частину великої індустрії комп’ютерних ігор, де гравці змагаються на професійному рівні у турнірах. Відомі кіберспортсмени стають впізнаваними та

впливають на молодше покоління, а розвиток кіберспорту сприяє технологічним інноваціям у галузі геймінгу, розвитку віртуальної та доповненої реальності, що має вплив на інші сфери технологій та бізнесу. Кіберспорт привертає інвестиції та створює нові робочі місця, а участь команд і гравців з різних країн створює міжнародні партнерства та можливості для розширення економічних зв'язків між країнами. Організація великих кіберспортивних заходів привертає увагу туристів, що сприяє розвитку готельного, ресторанного бізнесу, транспортних послуг [4]. Зокрема, українська кіберспортивна організація NAVI вважається однією з найталановитіших серед світових кібер-гравців в іграх Counter-Strike та Dota 2. Олександр Костильов, член команди, у 2021 році отримав звання найкращого кіберспортсмена року, а у 2022 – організація ESL Counter-Strike повідомила у Twitter щодо його першості в турнірі IEM Rio Major 2022. Гравець S1mple (сам О. Костильов) був названий найкращим гравцем за останні десятиліття [5].

Однак деякі дослідження вказують на те, що довготривала інтенсивна гра може призводити до таких проблем зі здоров'ям, як втома, стрес і порушення сну. У статті, опублікованій на сайті Harvard Health Publishing, який належить Гарвардському медичному факультету, зазначається важливість дотримання поміркованості в ігровому процесі та балансі між грою та іншими аспектами життя [6].

Розглянемо класифікацію комп'ютерних ігор за різними критеріями, такими як жанр, стиль геймплею, режим гри та рівень графіки. У минулому жанри ігор були чітко визначені, проте сучасний геймінг відрізняється великою різноманітністю, і зараз жанри вже не такі однозначні. Гравці все частіше стикаються з поєднанням елементів різних жанрів у межах однієї гри, що робить їх класифікацію більш складною. Жанри та їх піджанри стають дедалі різноманітнішими, особливо завдяки тому, що розробники ігор поєднують різні типи найнесподіванішими способами. Іншими словами, ландшафт

комп'ютерних ігор постійно розвивається. Один з найпоширеніших підходів до класифікації має наступний вигляд [7]:

1) Екшн – жанр, що визначається фізичними взаємодіями, швидкістю реакції гравця та інтенсивними сценами дії. великий акцент на боях з ворогами або навколишнім середовищем, які можуть включати стрілянину, ближній бій, використання зброї та магію. Гравець може розвивати та покращувати різні здібності свого персонажа, такі як сила, швидкість і здатність до бою. Багато екшн-ігор мають інтригуючі сюжети, пов’язані з завданнями та місіями, які вимагають активної участі гравця в розвитку сюжету.

2) У пригодницьких іграх більше уваги приділяється історії, сюжету та вирішенню головоломок. Гравець-людина повинен розв'язувати головоломки під час пригод, може взаємодіяти з іншими персонажами, розмовляти з ними, отримувати від них завдання та впливати на розвиток історії. Персонажі зазвичай можуть носити з собою предмети, такі як зброя, ключі, інструменти тощо.

3) У рольових іграх гравці можуть вибирати різноманітних персонажів з різними характеристики, такими як вид, раса, стать і рід занять, а також різні здібності, такі як сила і спритність. У віртуальному ігровому світі гравець виконує квести, бореться з монстрами і розширює можливості персонажа, використовуючи силу або магію. Багато рольових ігор також є мережевими, що дозволяє більшій кількості гравців взаємодіяти в одному ігровому світі через мережу, таку як Інтернет або локальну мережу, також відому як LAN .

4) У стратегічних іграх наголошується на використанні стратегії, а не на швидких діях або рефлексах. До традиційних стратегічних ігор належать шахи та монополія. В останніх популярних стратегічних іграх гравець може керувати багатьма бойовими одиницями, щоб вести бій проти одного або декількох супротивників. У цих іграх гравцеві потрібно вирішувати проблеми розподілу ресурсів, плануванням оборони та нападу.

5) Ігри-симулятори поділяються на два типи: управлінські симуляції та навчальні симуляції. Управлінські симулятори – це ігри, в яких гравці повинні керувати спільнотою, установо. чи імперією, використовуючи обмежені ресурси. Під навчальними симуляторами маються на увазі ігри, які намагаються навчити чогось, імітуючи реальні обставини. Симуляція допомагає гравцеві розвивати фізичні навички, такі як керування автомобілем, літаком тощо.

6) Основною характеристикою шутерів є акцент на взаємодії гравця з грою через стрільбу та боротьбу з ворогами. Жанр розгалузився на два основні піджанри: шутер від третьої особи (Third Person Shooter) і від першої особи (First Person Shooter). Проте багато сучасних ігор дозволяють перемикатися між точками зору. FPS показує тільки те, що бачить ігровий персонаж, а TPS – всього персонажа та навколишнє середовище.

7) Перегони – ігри, в яких гравці беруть участь у змаганнях на транспортних засобах. Можуть бути поділені на кілька категорій, таких як автомобільні, мотоциклетні, велосипедні, космічні, тощо. Змагання відбуваються як у режимі онлайн проти інших гравців, так і в одиночних іграх, де штучний інтелект грає проти гравця.

8) Спортивні ігри – це адаптації існуючих реальних видів спорту або їхні варіації.

9) Виживання – ігри, що акцентуються на виживанні гравця в умовах обмежених ресурсів, небезпек та невизначеності. Небезпеки зустрічаються в різних формах, такі як природні катастрофи, ворожі істоти або інші загрози, які гравець повинен подолати. Гравець повинен слідкувати за своїм здоров'ям, голодом та іншими важливими показниками, використовуючи доступні ресурси.

10) Ігри хорор орієнтовані на створення напруженого враження чи страху у гравця, ігри цього жанру часто включають складні сюжети, пов'язані з надприродними явищами, вампірами, зомбі, демонами чи іншими елементами

жаху. Виживання та хорор часто мають багато подібних рис, включаючи спроби обмежити ресурси гравця для підвищення рівня напруження та важкості гри [8].

Стиль геймплею та архітектура рівнів поділяються на:

– Відкритий світ – гравці можуть вільно розвідувати великий, нелінійний світ гри, де вони мають велику ступінь вибору та можливості взаємодії з оточенням.

– Лінійний сценарій – гра має чітко визначений сюжет, і гравець має обмежений вибір у визначенні шляху чи впливу на розвиток подій.

– “Метроїдванія” або екшн-пригода з відкритим світом – гравці поступово розблоковують доступ до різних областей гри, які можуть бути пов'язані між собою, дозволяючи гравцеві повертатися назад для вдосконалення навичок чи знаходження нових можливостей.

Поділ ігор за режимом гри базується на тому, скільки гравців може одночасно брати участь у грі. Усі ігри класифікуються як одиночні та мультиплеєрні. Одиночні – ігри, в яких гравець грає самостійно без інших реальних гравців. Гравець контролює усі аспекти гри та розвиває сюжет незалежно від інших. Мультиплеєрні – ігри, де кілька гравців може грати одночасно, співпрацюючи чи конкуруючи один з одним. Це може включати в себе різні форми співробітництва або змагань в режимі реального часу. Багато ігор поєднують одиночну гру та мультиплеєр.

Класифікація ігор за рівнем графіки визначається тим, як гра виглядає та які технічні можливості використовуються для створення візуальної естетики гри. Основні категорії включають:

– Текстові – використовують лише текстовий опис і небагато графіки, щоб гравці могли спілкуватися один з одним.

– 2D графіка – спрощена двовимірна графіка, часто в плоскому стилі.

– 3D графіка – усі елементи гри розроблені за допомогою тривимірної графіки, щоб зробити світ більш реалістичним.

Після аналізу жанрів, стилю геймплею, режимів гри та рівнів графіки комп’ютерних ігор було прийнято рішення розробити одиночну тривимірну гру у жанрі шутер від першої особи та з лінійним сценарієм. Жанр має свою актуальність та перспективи розвитку через кілька факторів:

1) Інновації в геймдизайні: гравці завжди шукають нові, цікаві геймплей-враження.

2) Унікальність геймплею: комбінація жанрів завжди привертає увагу через нестандартний геймплей, охоплює більшу кількість аудиторії – прихильників стратегічного планування та динамічного шутера.

3) Перспективи розвитку: можливість використання інтерактивних технологій, віртуальної реальності, застосування ШІ для розвитку більш інтелектуальних стратегій та ворогів.

Вибір лінійного сюжету зумовлений простотою його реалізації. Використання тривимірної графіки в даному жанрі забезпечує відповідність сценарію гри та сприяє позитивному сприйняттю гри в цілому.

1.2 Огляд та аналіз існуючих розробок

Жанр шутер від першої особи з елементами виживання (Survival Shooter) об'єднує в собі елементи бойового шутера та виживання, надаючи гравцям можливість ведення бою від першої особи, стратегічної оборони та виживання. У таких іграх гравець зазвичай відіграє роль командира, який повинен захищати стратегічний об'єкт та себе від хвиль ворогів, особисто ведучи бій від першої особи. Основні риси цього жанру включають:

– Гравець спостерігає за грою очима свого персонажу, що надає високий рівень реалістичності та динаміки.

– Головний фокус гри – це бойові дії, де гравець веде вогонь ворожих сил, використовуючи різноманітні види зброї та навички.

– Для успішної оборони необхідне стратегічне планування, включаючи вибір відповідної зброї та тактику переміщення.

Ігри у жанрі шутер можуть містити як і фентезійні елементи з мальованою графікою, так і реалістичні стилістичні рішення, а головними героями можуть бути звичайні люди або міфічні істоти. Жанр FPS зародився в 1990-х роках з ігор, таких як "Doom" і "Quake". Ці ігри високого темпу, де гравець бачив світ очима свого персонажа, стали відомі своєю інтенсивністю та екшн-орієнтованим геймплеєм. Survival Shooter, у свою чергу, фокусується на виживанні проти численних ворогів і змушує гравця ефективно використовувати ресурси та навички, щоб успішно виконати місію.

Прикладами ігор цього жанру можуть бути "Left 4 Dead", "Dead Alliance", “DayZ”, де гравець комбінує елементи шутера та виживання для ефективного ведення бою та вдосконалення своєї оборони.

1.2.1 "Left 4 Dead"

"Left 4 Dead" – це серія відеоігор, що вважається однією з найкращих ігор цього жанру завдяки інтенсивному геймплею, кооперативній грі та чудовій атмосфері. Відеогра розроблена американською компанією Valve Corporation, вперше випущена у 2008 році для Windows та Xbox360, пізніше була портована на інші платформи. Серія відеоігор отримала схвальні відгуки критиків та завоювала низку нагород, включаючи “Best Multiplayer Game” на премії BAFTA Video Game Award [9]. Знімок екрану гри подано на рисунку 1.1.

Рисунок 1.1 – Знімок екрану з гри "Left 4 Dead"

Сюжет гри відбувається в постапокаліптичному світі, де четверо людей, які вижили, борються з ордами зомбі. Цими персонажами керують гравці, проходячи через різні місця, щоб дістатися до безпечного місця. Гра складається з кількох кампаній, кожна з яких закінчується битвами з босами. Гравці використовують різні стратегії та навички для виживання під час атак зомбі, включаючи «special infected» з особливими здібностями. Одним із переваг ефективної боротьби з ворогами є широкий вибір зброї та засобів захисту.

Переваги даної комп'ютерної гри наступні:

– кооперативна гра;

– завдяки системі штучного інтелекту, що використовується для динамічного контролю ігрових подій (AI Director), кожне проходження відчувається унікальним;

– реалістична графіка, потужний звуковий супровід та інтенсивний темп гри.

Разом з тим, до недоліків можна віднести:

– відсутність глибокого сюжету: гра більше сфокусована на дії, ніж на розгортанні складного сюжету;

– гра найкраще розкривається в кооперативному режимі, що робить її менш привабливою для тих, хто віддає перевагу соло-іграм;

– незважаючи на динамічність, деякі гравці можуть відчувати, що геймплей повторюється через тривалий час.

Гра "Left 4 Dead" успішно поєднує елементи виживання, кооперативного геймплею та динамічного розвитку сюжету, що робить її популярною серед шанувальників жанру та геймерської спільноти загалом.

1.2.2 "Dead Alliance"

"Dead Alliance" – командна відеогра, що поєднує в собі елементи шутера від першої особи та стратегії, розроблена і видана американською компанією IllFonic. У серпні 2017 року вона була випущена для Windows, PlayStation 4 та

Xbox One. Стала відомою нетрадиційним підходом до шутерів від першої особи, у якому гравці можуть використовувати зомбі для атаки на ворогів [10].

Сюжет розгортається в постапокаліптичному світі, де гравці змушені боротися за виживання, використовуючи зомбі як засоби для нападу на ворогів і виживання в небезпечних умовах. Кожен тип зомбі має свої унікальні властивості і здатності, що дозволяє гравцям використовувати їх у бойових операціях для досягнення стратегічних цілей. Знімок екрану гри зображено на рисунку 1.2.

Рисунок 1.2 – Знімок екрану з гри "Dead Alliance"

Визначені переваги цієї гри наступні:

– унікальне поєднання стратегії та шутера від першої особи;

– сюжет, а саме можливість грати за різні класи зомбі, кожен із яких володіє різними здібностями.

При детальному аналізі виявлено такі недоліки:

– технічні проблеми і баги в грі, що вплинули на загальний геймплей;

– недостатня оптимізація ігрового процесу на різних платформах;

– обмежені можливості для мультиплеєра.

У загальному, гра "Dead Alliance" була спрямована на інноваційність у жанрі шутерів від першої особи, проте через технічні труднощі та слабку графіку не змогла досягти значної популярності серед ігрової аудиторії.

1.2.3 “DayZ”

“DayZ” – відеогра, що розроблена чеською компанією Bohemia Interactive, створена у жанрі шутера від першої особи з виживанням та поєднує елементи відкритого світу і мультиплеєра. Спочатку була випущена як модифікація для гри "Arma 2" у 2012 році, а потім стала окремим проєктом (рис. 1.3) [11].

У центрі сюжету гри знаходиться небезпечний світ, де гравці повинні вижити, зустрічаючись з іншими гравцями, які також шукають ресурси для виживання. Зосереджена на виживанні від голоду та спраги, а також на захисті від зомбі та інших небезпечних гравців. Динамічний геймплей гри вимагає приймати важливі рішення для виживання, від збору ресурсів до взаємодії з іншими гравцями. Для того, щоб захистити себе або взяти участь у конфліктах з іншими учасниками можна використовувати різноманітні предмети, знаряддя та зброю. У «DayZ» середовище відкрите та постійно змінюється, що створює напружену атмосферу та підвищує відчуття реальності.

Рисунок 1.3 – Знімок екрану з гри “DayZ”

У даній комп’ютерній грі привертають увагу наступні переваги:

– реалістична графіка та модель виживання, що вимагає від гравців використовувати стратегію та тактичні навички;

– відкритий світ: безліч можливостей для дослідження, взаємодії з іншими гравцями та пошуку ресурсів;

– унікальна напружена й аутентична атмосфера постапокаліптичного світу.

До недоліків гри можна віднести:

– складний для новачків ігровий процес;

– гра фокусується на виживанні, але не має глибокого сюжету, що може розчарувати гравців, які шукають насичений сюжетний досвід.

Загалом, гра "DayZ" пропонує унікальне відчуття виживання у відкритому світі, хоча має свої концептуальні обмеження, які варто враховувати при оцінці гри.

Базуючись на огляді існуючих розробок та враховуючи їхні переваги та недоліки, вирішено спрямувати розробку проєкту на створення гри із більш сучасною тривимірною графікою, простим геймплеєм, стилізованим під реальний світ, дотриманням продуманого сюжету та збалансованим поєднанням різних жанрів.

1.3 Постановка завдань для реалізації ігрового застосунку

Розробка ігрового застосунку буде реалізована у жанрі шутера від першої особи з елементами виживання.

Задля реалізації ігрового застосунку необхідно виконати такі поставлені завдання:

1) Для реалізації геймплея та механіки

– Розробити основні геймплейні механіки, включаючи керування персонажем, стрільбу, переміщення та взаємодію з об'єктами оточення.

– Реалізувати ворожих персонажів з різними рівнями складності.

– Визначити та реалізувати основні геймплейні взаємодії, такі як завдання, квести та бонуси.

2) Для створення графіки та анімації:

– Створити та інтегрувати 3-D моделі персонажів, об'єктів та оточення.

– Розробити реалістичні анімації для персонажів, включаючи рух, стрільбу, атаку та поранення.

– Забезпечити високу якість текстур та освітлення для покращення реалізму гри.

3) Для реалізації інтерфейса користувача (UI):

– Розробити інтуїтивний та ефективний інтерфейс для гравця, включаючи меню, панелі здоров'я та боєзапасу, а також ігрові підказки.

– Впровадити систему управління камерою та виведення інформації на екран для покращення користувацького досвіду.

4) Для забезпечення звуку та музичного супроводу:

– Забезпечити звуковий супровід для різних геймплейних ситуацій, включаючи звуки стрільби, руху та ефекти оточення.

– Додати оригінальну музичну композицію та звукові ефекти для створення належного настрою гри.

5) Провести тестування та виправлення помилок:

– Провести широкомасштабне тестування гри на різних етапах розробки для виявлення та виправлення помилок.

– Взаємодіяти з гравцями та отримувати фідбек для подальшого вдосконалення геймплею та функціональності.

6) Передбачити підтримку та оновлення:

– Забезпечити систему підтримки гравців та вчасні відповіді на їхні запитання та проблеми.

– Розробити план оновлень та додаткового контенту для подальшого розвитку гри.

Ця постановка завдань надає чітку орієнтацію та визначає ключові елементи розробки гри.

2 ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

2.1 Опис сценарію гри

Гравець з обмеженими ресурсами здоров’я в грі є останнім захисником стратегічного об'єкту міста. Він повинен захищати об’єкт та себе від постійних атак ворогів, застосовуючи різноманітні стратегії для захисту та стримування нападу. Для цього у нього є доступ до різних видів зброї, яку може використовувати для відбиття нападів.

На початку гри у головного героя є обмежений запас здоров’я, тому закінчення гри може настати у двох випадках: вичерпання здоров’я гравця або проникнення ворогів у стратегічний об’єкт. Якщо сили гравця досягають нуля через постійний тиск ворожих хвиль та нанесення йому шкоди, гра закінчиться для нього поразкою і доведеться розпочати гру спочатку. Основна мета гравця – протриматися якнайдовше проти безперервних хвиль нападу ворогів.

Гра запропонує гравцю захоплюючий геймплей, який випробує його навички стратегічного мислення та військової тактики. Кожна нова хвиля нападу ворогів буде складнішою, а вороги будуть все сильнішими, тому гравець повинен бути готовий до будь-яких випробувань.

2.2 Опис функціональних та нефункціональних вимог

Спираючись на поставлені задачі до розробки ігрового застосунку, виділено ключові функціональні та нефункціональні вимоги до системи. Для того, щоб відобразити типи ролей та їх взаємодію із системою створено UML-діаграму варіантів використання. (рис. 2.1).

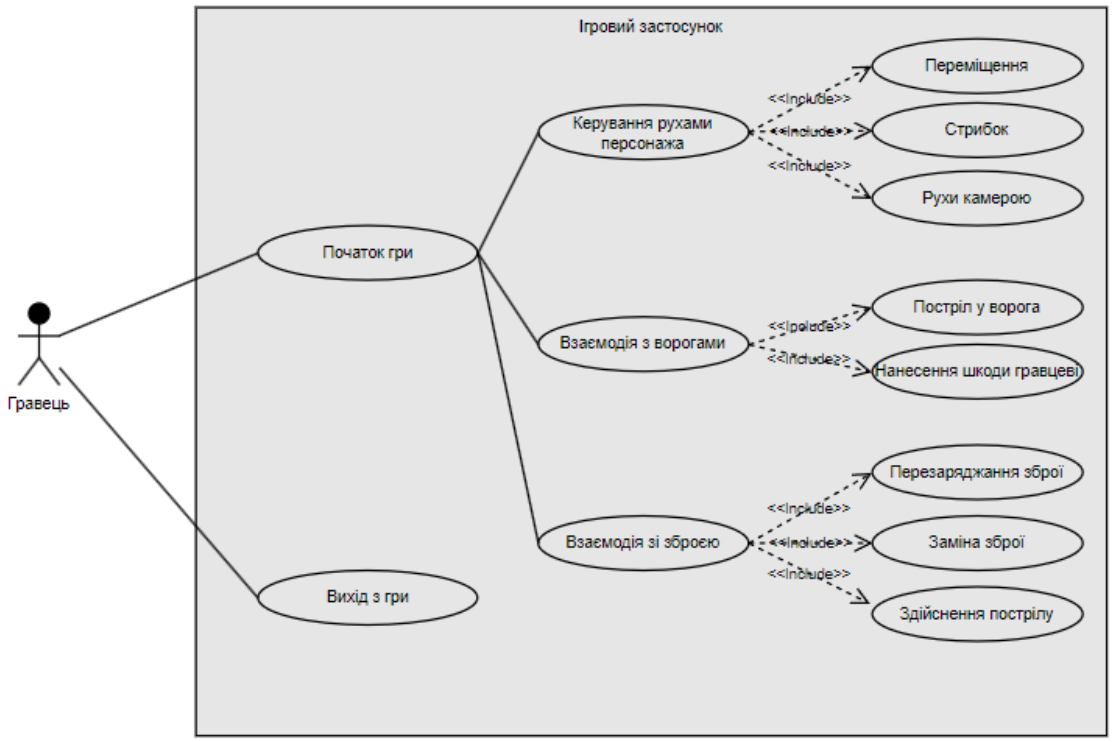

Рисунок 2. 1 – UML-діаграма прецедентів

Користувач, який виступає у ролі гравця, виступає основним актором. Таблиці 2.1–2.6 містять описи та варіанти дій користувача «Гравець».

Таблиця 2.1 – Опис варіанту використання UC01

Назва	Початок гри
Опис	Можливість почати або завершити гру. Перейшовши до головного меню програми.
Учасник	Гравець
Передумови	Користувач запустив ігрову програму, гра закінчилась поразкою або натиснув клавішу «Esc» протягом гри, відкривши таким чином головне меню
Постумови	Користувач зайшов у головне меню
Основний сценарій	Користувач переходить до головного меню
Розширення сценаріїв	Користувач розпочинає нову гру або виходить з гри

Таблиця 2.2 – Опис варіанту використання UC02

Назва	Керування рухами персонажа
Опис	Процес керування рухами персонажа у тривимірному ігровому середовищі
Учасник	Гравець
Передумови	Користувач натиснув на клавіші «w», «a», «s», «d», «space», або на клавіші стрілок та здійснив рухи мишкою задля обертання камери
Постумови	Персонаж здійснив рухи персонажа в ігровому світі відносно інших об’єктів
Основний сценарій	Персонаж просунувся вперед. Персонаж просунувся назад. Персонаж просунувся вправо. Персонаж просунувся вліво
Розширення сценаріїв	Персонаж підстрибнув та опустився на початкове місце. Персонаж повернув камерою

Таблиця 2.3 – Опис варіанту використання UC03

Назва	Взаємодія з ворогами
Опис	Взаємодія гравця з некерованими персонажами (NPC), що є ворогами гравця
Учасник	Гравець
Передумови	Користувач перебуває у режимі гри, на карті присутні вороги
Постумови	Гравець продовжує свою гру після взаємодії з NPC ворогами. NPC вороги завдають шкоди гравцеві і гра припиняється

Продовження табл. 2.3

Основний сценарій	Гравець може використовувати зброю або встановлювати захисні вежі для атаки NPC ворогів. У разі успішної атаки NPC вороги можуть бути знищені або отримати шкоду. У разі невдалої атаки гравець може отримати пошкодження або бути повалений NPC ворогами

Таблиця 2.4 – Опис варіанту використання UC04

Назва	Взаємодія зі зброєю
Опис	Взаємодія гравця зі зброєю в грі
Учасник	Гравець
Передумови	Користувач перебуває в грі та має доступ до зброї
Постумови	Гравець може продовжувати з іншими елементами гри після використання зброї
Основний сценарій	Гравець вирішує використати зброю для атаки ворогів. Гравець обирає зброю, яку він хоче використовувати. Гравець наводить вогонь на ціль та виконує вистріл. Зброя виконує вистріл, випускаючи кулі в напрямку обраної цілі. Якщо кулі досягають цілі, ця ціль отримує шкоду від удару. Гравець може продовжувати використовувати зброю до тих пір, поки це потрібно для досягнення своїх цілей.
Розширення сценаріїв	Якщо у гравця закінчуються боєприпаси для зброї, він може змінити зброю або скористатися альтернативними способами захисту. Якщо зброя гравця виявиться неефективною проти певного типу ворогів, гравець може змінити зброю на більш ефективну.

Таблиця 2.5 – Опис варіанту використання UC05

Назва	Вихід з гри
Опис	Вихід з поточної гри та завершення сеансу гри
Учасник	Гравець
Передумови	Гра повинна бути запущена та гравець повинен перебувати в грі
Постумови	Гравець завершує гру та повертається до головного меню або виходить з програми
Основний сценарій	Гравець відкриває головне меню програми та натискає кнопку «Завершити гру»

Таблиці 2.6–2.13 містять визначення основних функціональних вимог до ПЗ на основі UML-діаграми та описаних вище варіантів використання.

Таблиця 2.6 – Опис функціональної вимоги FR-1

Назва	Початок гри
Опис	Натискання кнопки в головному меню програми для початку нової гри

Таблиця 2.7 – Опис функціональної вимоги FR-2

Назва	Вихід з гри
Опис	Натискання кнопки в головному меню програми для виходу з гри

Таблиця 2.8 – Опис функціональної вимоги FR-3

Назва	Виклик меню
Опис	Відкривання головного меню, призупинивши ігрові процеси

Таблиця 2.9 – Опис функціональної вимоги FR-4

Назва	Вихід з меню
Опис	Натискання потрібної клавіші клавіатури, що дозволить закрити головне меню та поновити ігрові процеси

Таблиця 2.10 – Опис функціональної вимоги FR-5

Назва	Пересування персонажа у різні сторони
Опис	Можливість переміщення персонажа вперед, назад, вліво, вправо за допомогою відповідних клавіш руху та забезпечення відповідної анімації рухів

Таблиця 2.11 – Опис функціональної вимоги FR-6

Назва	Стрибки персонажа
Опис	Можливість переміщення головного персонажа вгору та вниз відносно ігрових об'єктів за допомогою відповідних клавіш клавіатури та забезпечення відповідної анімації

Таблиця 2.12 – Опис функціональної вимоги FR-7

Назва	Система штучного інтелекту
Опис	Поведінка NPC ворогів, можливість взаємодії з ними

Таблиця 2.13 – Опис функціональної вимоги FR-8

Назва	Бойова система
Опис	Різноманітність видів зброї та їх характеристики. Реалістична модель пострілів та пошкоджень

Щодо нефункціональних вимог, які можуть бути важливими для розробки гри то це:

1) Продуктивність. Гра повинна мати низький рівень завантаження процесора та оперативної пам'яті для забезпечення плавного геймплею на різних пристроях.

2) Графіка та анімація. Оскільки гра розробляється у тривимірному середовищі, то графіка повинна бути якісною та привабливою, з високим рівнем деталізації. Анімації персонажів та об'єктів повинні бути плавними та реалістичними.

3) Звук та музика. Звукові ефекти та музика повинні бути належної якості та підкреслювати атмосферу гри.

4) Керування та інтерфейс. Управління грою повинно бути інтуїтивно зрозумілим і простим для користувачів. Інтерфейс користувача повинен бути добре організованим та забезпечувати швидкий доступ до основних функцій гри.

5) Сумісність та оптимізація. Гра повинна бути сумісною з різними операційними, а саме Windows, Linux, macOS.

Враховуючи те, що даний ігровий застосунок зосереджений на україномовну аудиторію, то інтерфейс користувача повинен підтримувати українську мову.

2.3 Проєктування ігрового застосунку

Проєктування програмного забезпечення є ключовим етапом, що включає в себе створення детальних архітектурних схем та визначення процесів для забезпечення відповідності функціональним і нефункціональним вимогам. Змодельовано архітектуру та процеси програми на основі вищезазначених функціональних та нефункціональних вимог. Для досягнення цього було розроблено схему проходження ігрового застосунку у вигляді BPMN-діаграми бізнес-процесів і взаємодії між учасниками в них (рис. 2.2).

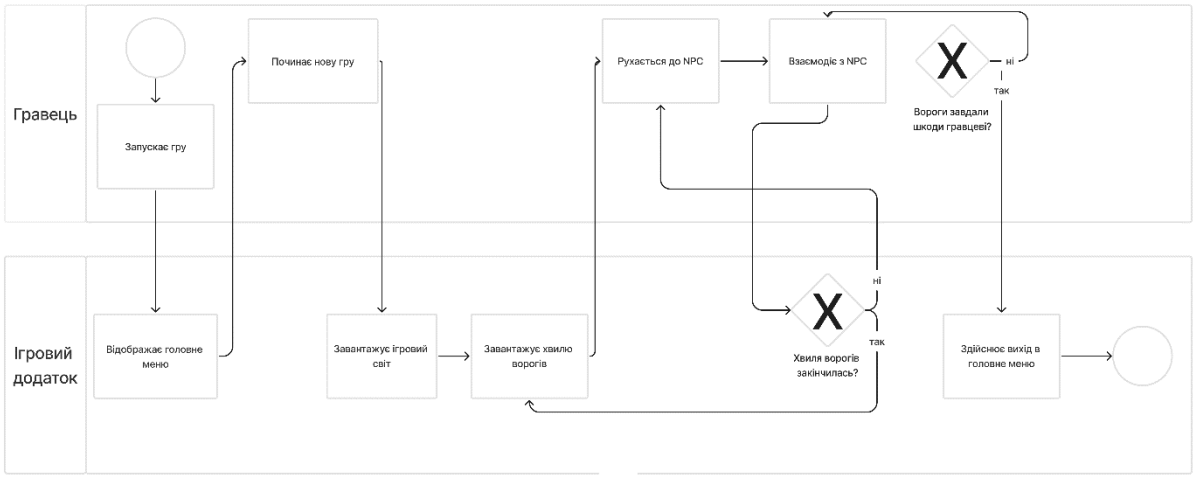

Рисунок 2.2 – BPMN-діаграма проходження ігрового застосунку

Процес проходження гри включає наступне:

– гравець починає гру, запускаючи її;

– відображається головне меню гри;

– гравець розпочинає нову гру, натиснувши на відповідну кнопку меню;

– гравець відображається в ігровому світі у стартовому місці;

– гравець дізнається про мету гри, сюжет та завдання;

– гравець та взаємодіє з NPC; у випадку завдання шкоди гравцеві та вичерпання усіх ресурсів здоров’я гра завершується; гра здійснює вихід у головне меню;

– якщо хвиля NPC ворогів завершується, то гравець переходить на новий рівень; інакше гравець продовжує взаємодію з NPC та повторює подальші дії.

Для представлення структури ігрового застосунку розроблено UML-діаграму класів, яка показує статичні елементи, такі як класи, типи даних, їхній зміст та їхні відношення (рис. 2.3).

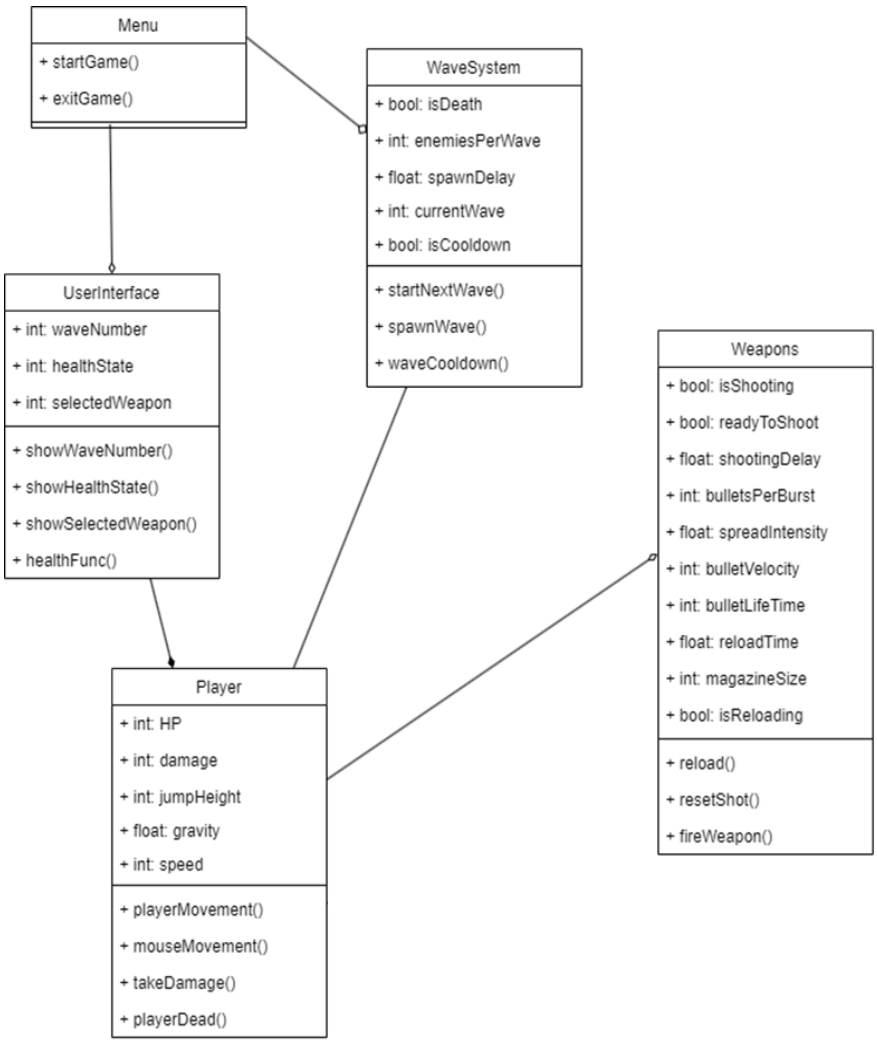

Рисунок 2.3 – UML-діаграма класів

Відповідно до діаграми:

– клас Menu містить тільки методи startGame() та ExitGame();

– клас UserInterface містить атрибути waveNumber (хвиля ворогів), healthState (стан здоров’я гравця), selectedWeapon (обрана зброя) та методи showWaveNumber(), showHealthState(), showSelectedWeapon(), healthFunc();

– клас Player містить атрибути HP (здоров’я гравця), damage (пошкодження), jumpHeight (висота стрибка), gravity (гравітація), speed (швидкість переміщення) та методи playerMovement(), mouseMovement(), takeDamage(), playerDead();

– клас WaveSystem містить атрибути isDeath (мертвий/живий), enemiesPerWave (кількість ворогів за хвилю), spawnDelay (затримання хвилі),

currentWave (поточна хвиля), isCooldown (зупинка хвилі) та методи startNextWave(), spawnWave(), waveCooldown;

– клас Weapons містить атрибути isShooting (стріляє/нестріляє), readyToShoot (готовність до пострілу), shootingDelay (затримка стрільби), bulletsPerBurst (кількість куль за постріл), spreadIntensity (інетнсивність поширення), bulletVelocity (швидкість кулі), bulletLifeTime (час життя кулі), reloadTime (час перезарядки), magazineSize (розмір магазину), isReloading (перезаряджається чи ні) та методи reload(), resetShot(), fireWeapon();

– встановлено відповідні зв’язки між класами.

2.4 Проєктування користувацького інтерфейсу ігрового застосунку

Ключовою метою будь-якого інтерфейсу є забезпечення ефективної взаємодії між користувачами та системою. Саме тому інтерфейс повинен відповідати наступним вимогам:

– легкість у використанні та зрозумілість для гравців усіх рівнів досвіду;

– відповідність стилістиці та тематиці гри;

– забезпечення можливості виконання потрібних функцій з мінімальними зусиллями;

– не повинен заважати гравцеві під час гри.

Знайомство користувача з грою розпочинається з меню програми, яке повинне бути інтуїтивно зрозумілим, простим у використанні та необтяжене зайвими елементами. Проєктування ігрового меню зображено на рисунку 2.4.

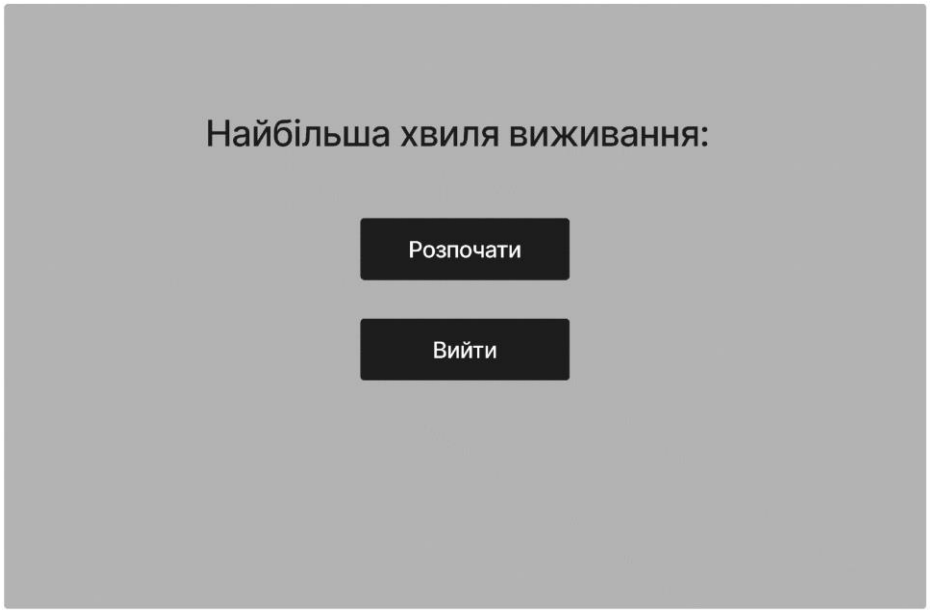

Рисунок 2.4 – Проєктування головного меню ігрового застосунку

Весь процес гри передбачає відображення основної інформації про стан героя, повідомлення про завдання чи досягнення. Ці елементи не повинні заважати ігровому процесу, а також бути зрозумілим для користувача (рис. 2.5).

Рисунок 2.5 – Проєктування ігрової сцени застосунку

Спроєктований інтерфейс відповідає усім сформульованим вимогам та є юзабельним для користувача.

2.5 Дослідження технологій та засобів реалізації ігрового застосунку

Перш ніж почати розробку ігрового застосунку, необхідно обрати

оптимальний варіант ігрового рушія, а саме програмного забезпечення, яке використовується для розробки відеоігор. Рушій надає розробникам набір інструментів, бібліотек і ресурсів для створення ігрових середовищ, персонажів, об'єктів, анімацій, фізики.

Для розробки даного ігрового застосунку оптимальним буде рушій, який підтримує платформу ПК та ОС Windows, розробку у тривимірному просторі для створення з невисокими системними вимогами. Одними з найбільш популярних та потужних рушіїв є Unreal Engine, Unity, Godot. Нижче наведено основні характеристики, сильні та слабкі сторони кожного рушія для порівняння та обґрунтування вибору.

Unreal Engine від Epic Games відомий своєю високоякісною графікою та наявністю блупринтів (Blueprints Visual Scripting), що дозволяє розробляти додатки без написання коду. Серед переваг також можна відзначити готові рішення «з коробки», тобто готові шаблони, відкритий вихідний код та сучасний інструментарій. Для прикладу, системи Lumen та Nanite, які дозволяють створювати високодеталізовані сцени з динамічним освітленням без значного впливу на перформанс. Lumen – динамічна система глобального освітлення та відображень. Nanite – технологія автоматичного масштабування геометричної деталізації оточення в режимі реального часу в залежності від відстані до об'єкта. Також Unreal Engine готовий для роботи з консолями дев'ятого покоління PlayStation 5 та Xbox Series X/S, що дає розробникам можливість випускати ігри на всіх актуальних платформах.

Не зважаючи на велику кількість переваг, популярний рушій має і ряд недоліків. На противагу великому інструментарію фахівці стикаються з проблемою великої кількості непотрібних для них інструментів, які можуть впливати на рендер і створювати зайвий обсяг. Звичайно, шаблонні рішення корисні, але їх потрібно налаштувати під себе, що займе більше часу. Сучасні інструменти потребують вміння правильно застосовувати їх. І навпаки, технічна ситуація може бути хорошою, але візуальні елементи можуть

виглядати застарілими. Усі переваги Unreal Engine спрямовані на 3D-проєкти. З іншого боку, якщо потрібно створити 2D-гру, використання рушія здається просто незручним і саме створення може виявитися складнішим і дорожчим. На мобільних платформах створена гра матиме розмір на 70-100 Мб більший, ніж при створенні на Unity. Також системні вимоги рушія потребують потужного ПК. Мінімум – це чотириядерний процесор, 8 Гб оперативної пам'яті та відеокарта NVIDIA GeForce RTX 2000-Ї серії або AMD Radeon 6000-ї серії. [12]

Далі розглянуто рушій Unity. Загалом в Unity низький поріг входу, невеликі системні вимоги, зрозумілий користувацький інтерфейс. Серед переваг можна ще виділити універсальність, а саме успішність реалізації як простого 2D-проєкту, так і масштабного релізу. При цьому затрати коштів будуть меншими, якщо порівнювати з конкурентами. До редактора рушія можна додавати новий функціонал, що є простішим процесом, ніж у конкурентів. Завдяки активній спільноті, сформованій навколо Unity, ще одна перевага – дуже багатий маркетплейс з безліччю асетів, плагінів та рішень. Не всі вони безкоштовні, але їх використання допомагає заощадити час та спростити процес розробки. Unity пропонує безліч можливостей для роботи з двовимірним світом, чого немає у багатьох інших рушіїв. Також системні вимоги двигуна потребують мінімальних вимог системи. Мінімум – це ОС, починаючи від версії Windows 7, а також процесор, який підтримує DirectX 10-ї версії та архітектуру x64.

Серед недоліків можна виділити складніший процес створення багатокористувацьких ігор, закритий вихідний код, оновлення, які часто містять багато багів. Через закритий вихідний код позбутися проблем, які виникають з боку рушія, буває складно і процес розробки займає більше часу. У Unity є три системи рендерингу, проте вони є максимально різними, тобто не уніфікованими. Це може призвести до непередбачуваної роботи програми на різних пристроях або платформах та до невідповідностей у відтворенні графіки.

Наступним проаналізовано гральний двигун Godot Engine. Серед переваг: легкий для розуміння та зручний інтерфейс, відкритий системний код, 2D та 3D рендеринг, кросплатформенність. Рушій використовує сценічний дизайн, тобто все у грі організовано в сцени, кожну з яких можна завантажити та вивантижити за потреби. Це дозволяє грі використовувати пам’ять на низькому рівні, що сприятиме її плавній роботі. Підтримка багатокористувацької гри Godot базується на одноранговій мережевій бібліотеці ENet на основі UDP (протокол датаграм користувача).

Що стосується слабких сторін, то рушій використовує для написання сценаріїв свою особисту мову GDScript. Хоча мова нагадує Python, проте вимагає більше витраченого часу для адаптації. У зв’язку з меншою популярністю Godot Engine складніше знайти готові ресурси, такі як готовий шаблон, моделі, анімації. Інструменти рушія відстають у розробці 3D-проєктів, все-таки він більше підійде для реалізації 2-D мобільних ігор.

3 РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ

3.1 Вибір та обґрунтування засобів реалізації ігрового застосунку

Проаналізувавши переваги та недоліки ігрових рушіїв, які зазначені у пункті 2.5, було прийнято рішення розробки даного ігрового застосунку FPS у середовищі Unity. По-перше, Unity надає потужний та розширений набір інструментів для роботи з 3D-графікою, фізикою, анімацією та штучним інтелектом. Крім того, рушій має велику і активну спільноту розробників, яка надає безліч ресурсів, плагінів та розширень, що полегшує процес розробки. Також слід взяти до уваги велику кількість готових рішень та ресурсів для розробки ігор, включаючи шаблони, стартові проекти та різноманітні інструменти [13].

У якості середовища написання програмного коду було обрано найбільш популярне Microsoft Visual Studio. Причини вибору:

1) IDE (інтегроване середовище розробки) має широкий набір інструментів, а саме автодоповнення, розширення, інтегрована система керування версіями, візуальні редактори для різних типів файлів;

2) розширені інструменти для відлагодження коду, включаючи точки зупинки, крокування по коду, перегляд змін;

3) інтеграція з іншими продуктами: Azure, GitHub, Team Foundation Server, а також найбільш потрібна у даному проєкті – Unity;

4) активна спільнота розробників створює велику кількість розширень та додатків для покращення функціональності IDE, а саме засоби автоматизації та додатки для роботи з конкретними інструментами [14].

Для написання програмного коду при реалізації було обрано мову програмування C#. Спочатку слід зазначити, що C# є однією з найбільш поширених мов програмування для розробки ігор у геймдеві і обране середовище Unity підтримує цю мову як основну. Крім того, C# є мовою, яка дозволяє легко використовувати різні ресурси та бібліотеки для виконання

різних функцій у грі та дозволяє швидко розробляти та тестувати ігрові механіки.

Вибір мови програмування C# для створення ігрового застосунку пояснюється її популярністю, простотою використання та підтримкою в платформі Unity. Таким чином, Unity є потужною платформою для розробки ігор із великою кількістю інструментів, а також інтегрованим середовищем розробки Visual Studio, яке надає зручне середовище для написання коду. Розробка ігор із використання даних ресурсів є ефективним і продуктивним процесом.

3.2 Реалізація ігрового застосунку в середовищі Unity

Реалізація ігрового застосунку в Unity включає створення головного героя, карти, зброї, ворогів, звукових ефектів, анімації, користувацького інтерфейсу та меню. Для створення усіх звукових ефектів ігрових елементів використовується загальний скрипт, який подано у додатку А. Для додавання аудіокліпів використовується компонент AudioSource.

Слід розпочати з головного героя, оскільки він є ключовим елементом гри. Для керування рухом героя використовується компонент Character Controller, а для обробки введення з клавіатури або геймпада – скрипти на C#. Модель гравця складається з кількох важливих елементів, які забезпечують його функціонування в тривимірному середовищі: тіла (Body), головної камери (Main Camera), об’єкту перевірки поверхні (GroundCheck).

Основний елемент, що представляє тіло головного героя містить компоненти, які дозволяють персонажу бути присутнім у грі:

– Capsule Collider – коллайдер у формі капсули, який визначає фізичні межі персонажа та дозволяє персонажу взаємодіяти з іншими об'єктами в грі та реагувати на фізичні зіткнення;

– Mesh Renderer – компонент, який відповідає за візуалізацію моделі персонажа, рендерить сітку (mesh) персонажа, роблячи його видимим у грі.

Враховуючи те, що гра розробляється від першої особи, головна камера повинна бути розміщена в точці зору головного героя. Це забезпечує гравцю погляд з перспективи першої особи, де видно лише зброю. Головна камера складається з компонентів:

– Camera Component – основний компонент камери, що визначає її поведінку.

– Audio Listener – компонент, що дозволяє гравцеві чути звуки наче вони походять від місця розташування камери.

– GroundCheck – елемент, що використовується для визначення, чи знаходиться персонаж на поверхні (землі) або в повітрі. Він допомагає обробляти такі дії, як стрибки та падіння. GroundCheck зазвичай реалізується як невеликий об'єкт з коллайдером, який розташовується під персонажем і містить:

– Transform – компонет, що визначає позицію та орієнтацію GroundCheck об'єкта.

– скрипт для перевірки, який визначає чи знаходиться персонаж у контакті з поверхнею.

Вигляд моделі головного героя подано на рисунку 3.1, а скрипти персонажа – у додатку Б.

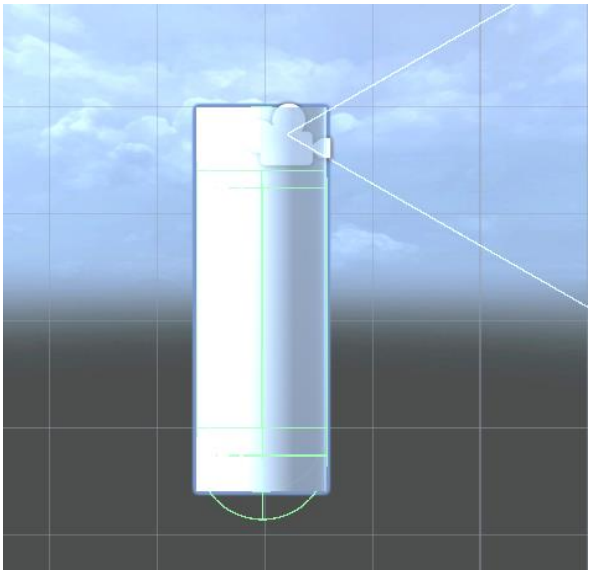

Рисунок 3.1 – Зображення моделі головного героя

У грі головний персонаж має низку важливих характеристик, які визначають його можливості та те, як він взаємодіє з ігровим середовищем. Швидкість пересування, висота стрибка, стан здоров’я та візуальний ефект при пораненні є частиною цих характеристик. Швидкість пересування персонажа визначає, як швидко він може переміщуватися по ігровій карті. Висота стрибка визначає, як високо персонаж може стрибати. Здоров'я персонажа визначає, скільки шкоди він може отримати, перш ніж померти. При отриманні шкоди, на екрані з'являється візуальний ефект "bloody screen", який сигналізує гравцю про поранення та додає атмосферності грі. Цей ефект частково забарвлює екран червоним кольором, що візуально нагадує про небезпеку та втрату здоров'я. Це допомагає гравцю швидше усвідомити необхідність ухилення або зцілення.

Зброя в грі є важливою частиною ігрового процесу. Об’єкт WeaponSpawn, який містить скрипт перемикання зброї, відповідає за зберігання та управління моделями зброї (додаток В). Кожна вогнепальна зброя має свою 3D-модель, анімації та звуки стрільби. Індивідуальні параметри кожної моделі зброї містяться в скрипті зброї (додаток В). Ці параметри включають збиток від кулі (weaponDamage), затримку між пострілами (shootingDelay), кількість куль у черзі (bulletsPerBurst), інтенсивність розсіювання кулі (spreadIntensity), префаб кулі, який зображено на рисунку 3.2 (bulletPrefab), місце появи кулі в ігровому світі (bulletSpawn), швидкість кулі (bulletVelocity), тривалість життя кулі (bulletPrefabLifeTime) та режими стрільби (Current Shooting Mode), а саме одиночний (Single), почерговий (Burst) та автоматичний (Auto). Параметри часу перезарядки (reloadTime), кількості куль у магазині (magazineSize) та залишку куль у магазині (bulletsLeft) контролюють систему перезарядки.

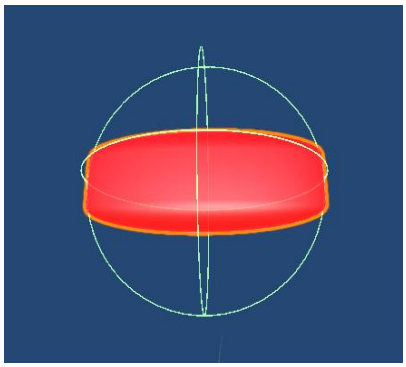

Рисунок 3.2 – Шаблон кулі

Як показано на рисунку 3.3, зброя реалізована як 3D-моделі.

Рисунок 3.3 – Зображення 3D-моделі зброї

Відповідні анімації, звуки та візуальні ефекти були додані, щоб зробити досвід використання зброї в грі реальним. Вони передають стани зброї під час різних дій, таких як стрільба, перезарядка та прицілювання. Анімації різних станів зброї створюються безпосередньо у середовищі розробки Unity за допомогою Animator Controller. Схеми анімацій для обох моделей зброї подано на рисунках 3.4, 3.5. Серед звукових елементів зброї – звуки стрільби, перезарядки та порожнього магазину (додаток А).

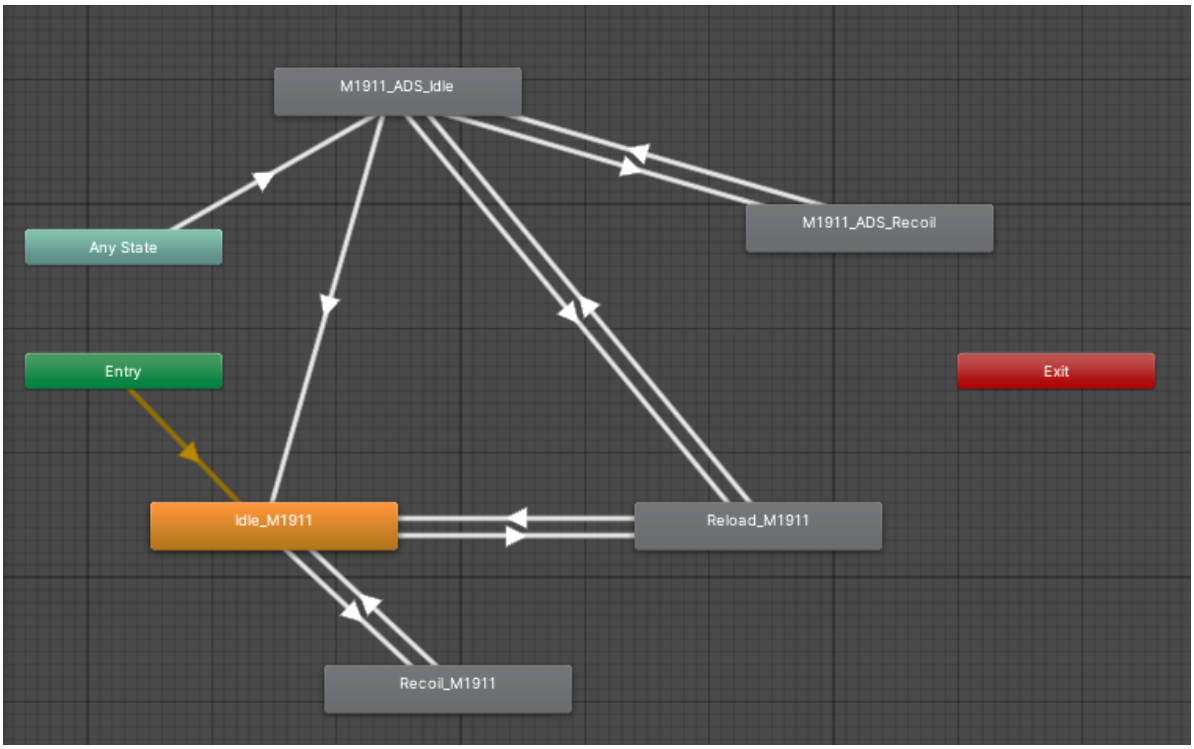

Рисунок 3.4 – Схема анімацій для моделі пістолета

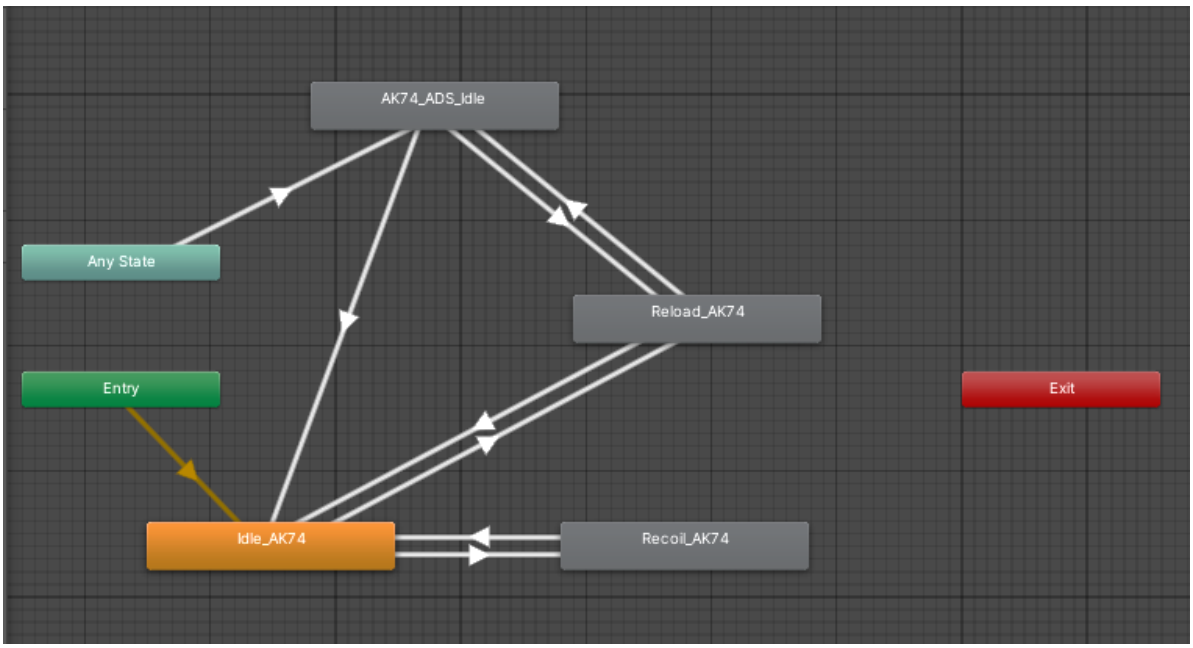

Рисунок 3.5 – Схема анімацій для моделі автомата

Вороги в грі мають 3D-моделі (рис. 3.6) та анімації.

Кожен ворог має параметр здоров'я, який визначає скільки шкоди може витримати перед тим, як бути знищеним, та є частиною хвилі ворогів, яка

генерується об'єктом MilitarySpawner. Хвилі ворогів мають визначені параметри, а саме кількість ворогів за хвилю (InitialMilitaryPerWave), час очікування між хвилями (WaveCooldown) та затримку між появою окремих ворогів (SpawnDelay).

Рисунок 3.6 – 3-D модель ворогів

Штучний інтелект (AI) використовується для зміни того, як вороги поводяться. Вони повинні мати можливість пересуватись, переслідувати та атакувати гравця. Компонент NavMesh Agent використовується для навігації по карті, а компонент Animator використовується для анімації дій ворогів: ходьба, біг, атакування, поранення та смерть (рис. 3.7). AI поведінка ворогів реалізується за допомогою скриптів, які подано у додатку Г. Звук бігу, ходьби, атаки, поранення та смерті є відповідними звуковими ефектам, реалізацію яких подано у додатку А.

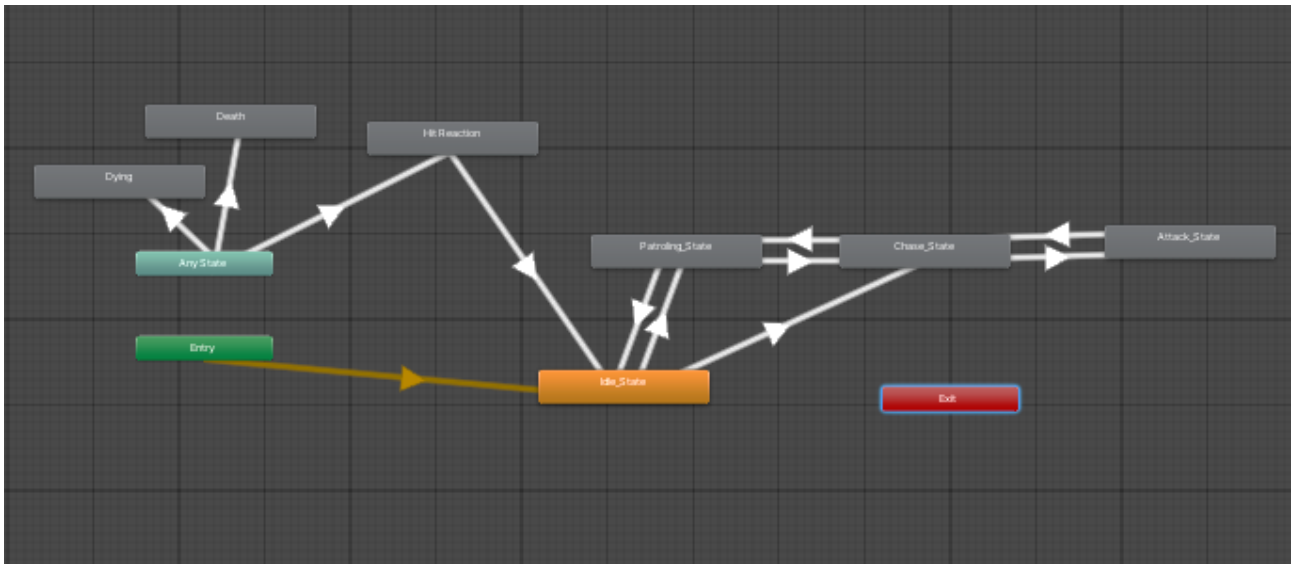

Рисунок 3.7 – Схема анімацій для ворогів

Карта гри створюється за допомогою редактора сцен Unity. Вона охоплює різні елементи навколишнього середовища, такі як будівлі, човни, мости, дороги, дерева, рельєф і водні об'єкти. Створення рельєфу здійснюється за допомогою компоненту Terrain, а додавання статичних об'єктів, таких як будівлі, човни, мости та дерева – GameObjects. Для забезпечення навігації ворогів по карті використовується компонент NavMesh. Звукові ефекти та анімації добавлено безпосередньо до кожного елемента. Усі елементи об'єднуються в одну сцену, де гравець може пересуватися, взаємодіяти з ворогами та навколишнім середовищем. Загальний вигляд карти зображено на рисунку 3.8.

Рисунок 3.8 – Зображення карти ігрового застосунку

У грі використовується кілька ключових елементів інтерфейсу користувача (рис. 3.9, 3.10), які забезпечують зворотній зв'язок гравцю про стан гри, здоров'я персонажа, поточну хвилю ворогів та інші важливі аспекти. Для реалізації текстових полів використовується компонент TextMeshPro, який забезпечує високу якість відображення тексту та широкі можливості для його налаштування.

Елемент Player Health UI призначений для відображення поточного рівня здоров'я гравця, що дозволяє гравцю бачити рівень здоров’я свого персонажа в будь-який час. Це текстовий інтерфейс, розташований у правому верхньому кутку екрану. Wave Number відображає поточну хвилю ворогів, з якою стикається гравець. Цей компонент розташований у видимій частині екрану та допомагає гравцю спостерігати за тим, як він прогресує в грі. Гравець завжди знає, на якій хвилі він знаходиться. Крім того, у нижньому правому кутку знаходиться індикатор залишку куль у магазину зброї для перезарядки. Він складається з текстового поля, яке відображає поточну кількість куль у магазині та загальну кількість куль, яку можна вмістити в магазин після перезарядки. Ці елементи користувацького інтерфейсу відображаються у головній сцені гри та подані на рисунку 3.9.

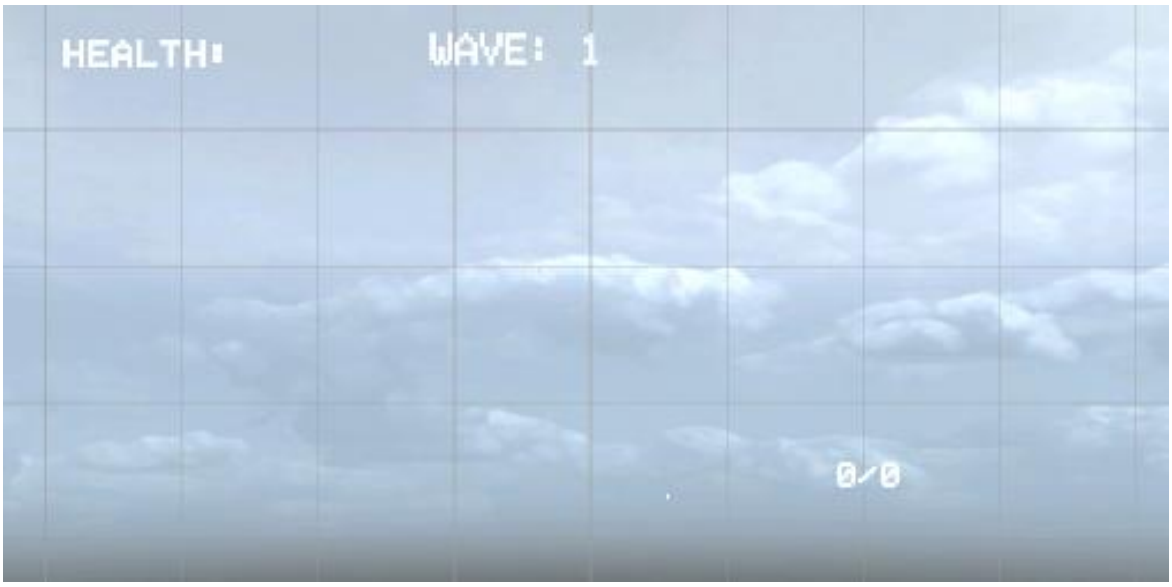

Рисунок 3.9 – Зображення користувацького інтерфейсу головної сцени

Після того, як персонаж гравця втрачає всі очки здоров’я та помирає, інтерфейс завершення гри з’являється на екрані (рис. 3.10). Ця частина включає повідомлення про завершення гри, яке супроводжується мелодією, та повернення до головного меню. Він інформує гравця про завершення поточної гри та дозволяє швидко розпочати нову гру.

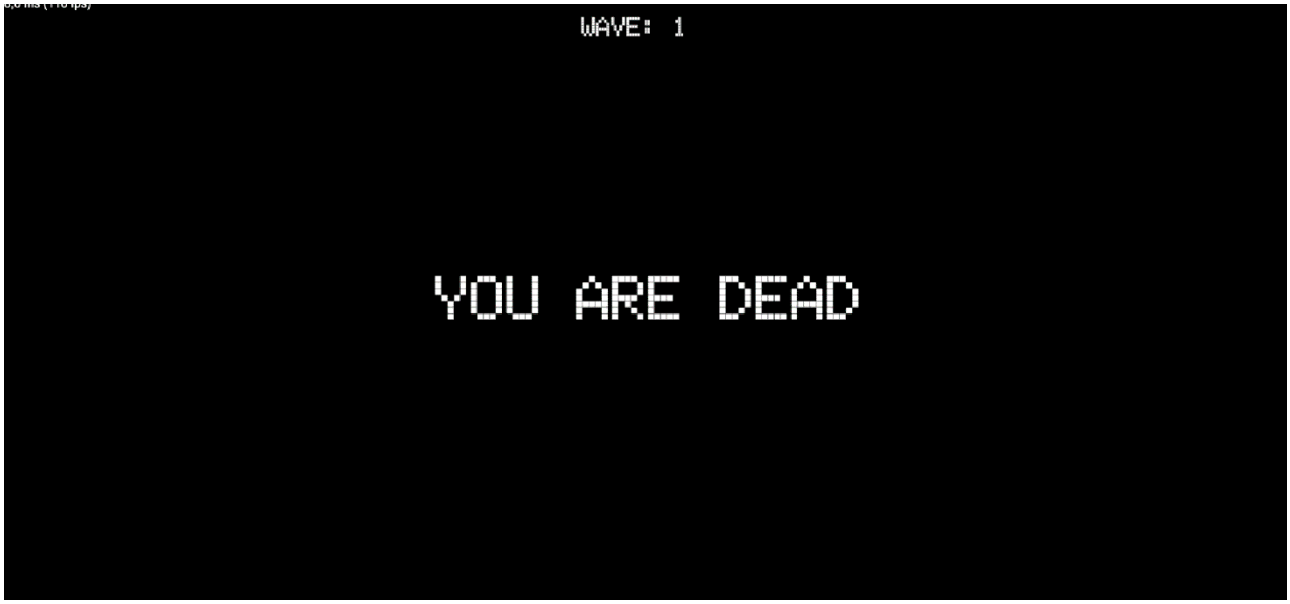

Рисунок 3.10 – Зображення користувацького інтерфейсу завершення гри

До користувацького інтерфейсу можна віднести головне меню ігрового застосунку, що є першим екраном, який бачить гравець при запуску гри. Воно включає привітальне повідомлення для гравця, яке інформує гравця про основні завдання у грі та мотивує його на успішне виконання місії. Головне меню також містить дві основні кнопки. Кнопка "Початок гри" запускає нову гру, завантажуючи дві сцени - SampleScene та Scene_A. Кнопка "Вихід з гри" дозволяє гравцеві закрити гру. Для реалізації головного меню використовуються скрипти, які подані у додатку Д.

Після реалізації основних механік проводиться оптимізація продуктивності гри та тестування для виявлення і виправлення помилок. Це включає налаштування графіки, звуку та інших параметрів для забезпечення плавного геймплею. Таким чином, використання Unity забезпечує ефективний

процес створення комплексного ігрового застосунку з головним героєм, картою, зброєю та ворогами.

3.3 Тестування гри

Метою тестування є перевірка коректності роботи всіх функціональних елементів гри, виявлення можливих помилок та забезпечення високої якості кінцевого продукту. Тестування буде здійснюватися за допомогою ручного тестування. Основні аспекти, які будуть протестовані:

- Головне меню.
- Ігровий процес.
- Графічний інтерфейс користувача (UI).
- Системи збереження і завантаження гри.
- Звукові ефекти та анімації.
- Продуктивність та стабільність.

Після завантаження ігрового застосунку відкривається головне меню (рис. 4.1), яке містить два основні функціональні елементи: кнопки “Почати гру” та “Вихід”. При натисканні кнопки "Почати гру" перевіряється, чи завантажуються без помилок сцени “SampleScene” (сцена з усіма ігровими елементами та головним персонажем) і “Scene_A” (сцена карти). Кнопка "Вихід" забезпечує коректне завершення гри.

Оскільки гра розроблена лише для однієї платформи, тестування кросплатформеності не передбачається для комп’ютерної гри. Для забезпечення найвищої якості графіки та візуального досвіду користувача акцент зроблено на широкій візуалізації.

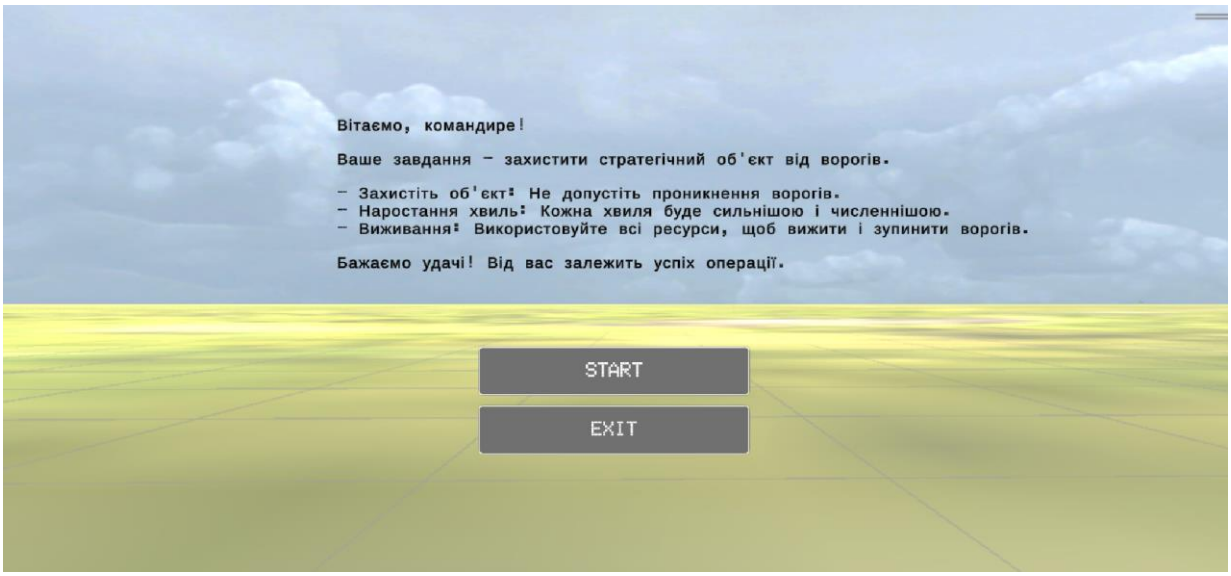

Рисунок 4.1 – Зображення головного меню ігрового застосунку

Після тестування всі елементи головного меню відображаються у правильній послідовності, кнопки виконують свої функції, звуковий супровід присутній.

Завантаження сцен відбувається успішно та без помилок, гравець з’являється у правильному місці. Елементи користувацького інтерфейсу та зброя коректно відображаються, що подано на рисунку 4.2.

Рисунок 4.2 – Зображення початку гри

Ігровий процес включає в себе кілька важливих аспектів, які підлягають

тестуванню. Після завершення тесту головний герой коректно пересувається за допомогою клавіш “W”, “A”, “S”, “D” або стрілок та може стрибати після натискання клавіші “Space”. Натискання лівої кнопки миші дозволяє йому стріляти та взаємодіяти з об’єктами, а натискання Ctrl + R дозволяє перезаряджати зброю. Зміна моделі зброї відбувається натисканням клавіш “1” та “2”.

Усі елементи користувацького інтерфейсу відображаються правильно та змінюють свої значення відповідно до взаємодії персонажа. Після завершення хвилі ворогів відбувається відлік до початку наступної хвилі (рис. 4.3). Кожна наступна хвиля генерує більшу кількість ворогів.

Рисунок 4.3 – Зображення елементів користувацького інтерфейсу

Зброя має анімацію, звукові та візуальні ефекти, і кожна модель має різний рівень пошкодження. На рисунку 4.4 показано, що ці компоненти працюють правильно.

Рисунок 4.4 – Відображення візуальних ефектів та анімації зброї

На рисунку 4.5 подано коректне відтворення анімації перезарядки та індикатор куль оновлюється правильно.

Рисунок 4.5 – Відображення перезарядки зброї

Вороги реагують на дії гравця, атакують та з'являються нові хвилі. Анімація та звукові ефекти ворогів, а саме ходьба, біг, атакування, ушкодження та смерть працюють коректно. Після ушкодження ворогами головного героя відображається екран ушкодження гравця та здоров’я зменшується, що зображено на рисунку 4.6.

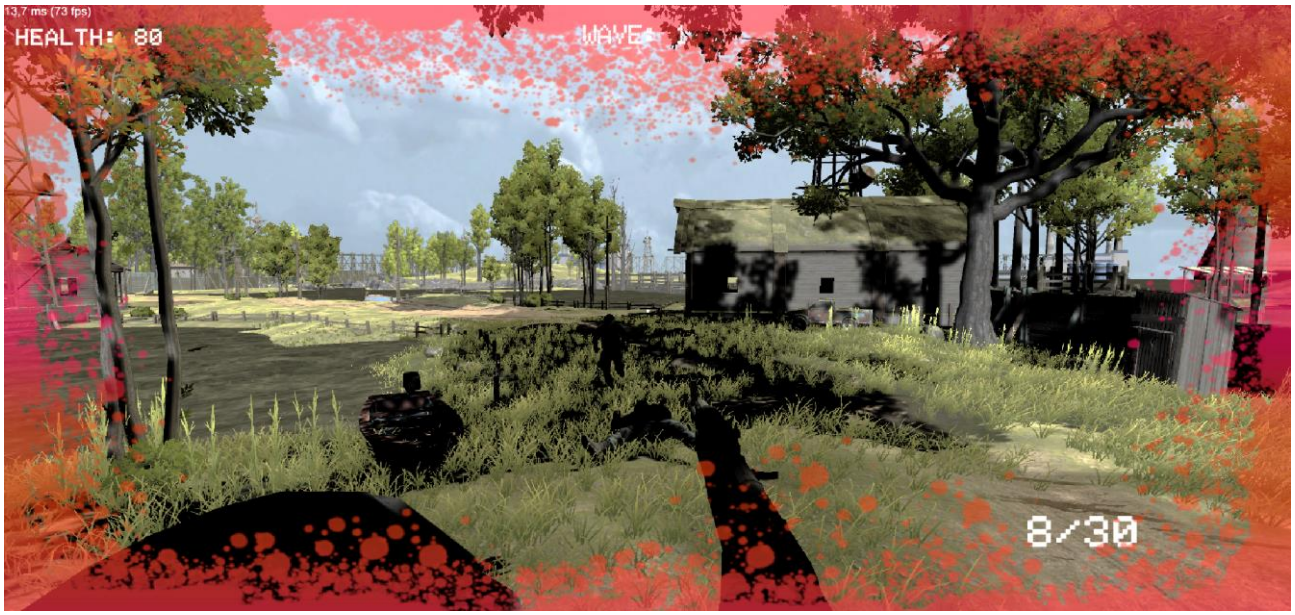

Рисунок 4.6 – Відображення ушкодження гравця

При смерті головного героя з'являється екран "Game Over" та звучить мелодія (рис. 4.7), через кілька секунд відбувається перехід до головного меню.

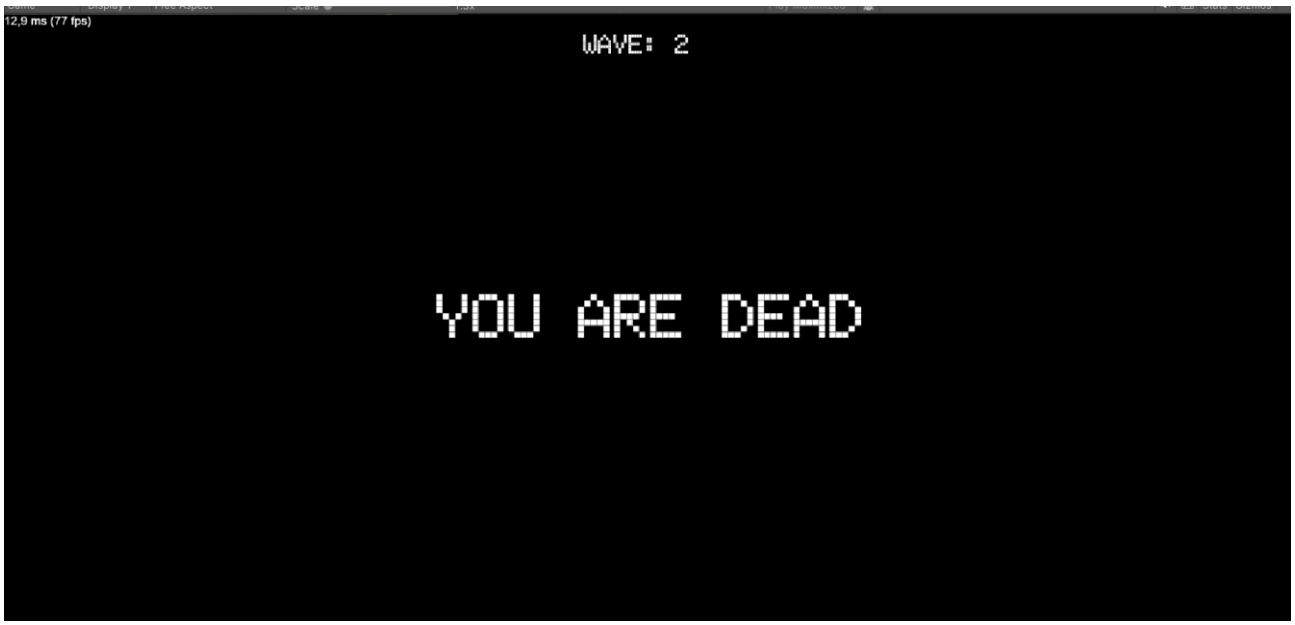

Рисунок 4.7 – Відображення смерті головного героя

Після ретельного тестування ігрового застосунку було встановлено, що всі основні компоненти та функції працюють правильно. Управління головним героєм, функціональність зброї, поведінка ворогів та робота користувацького інтерфейсу відповідають очікуванням. Інтеграційне тестування показало, що взаємодія між різними компонентами гри відбувається без конфліктів та

помилок, а системне тестування підтвердило стабільну продуктивність гри навіть під навантаженням. Зброя в грі демонструє хорошу анімацію, звукові та візуальні ефекти. Усі ці компоненти функціонують відповідно до задуму розробників. Звукові ефекти доповнюють візуальні елементи гри.

Проте, як і в будь-якому програмному продукті, завжди є можливості для удосконалення. Покращення штучного інтелекту ворогів є одним із можливих шляхів вдосконалення гри. Їх поведінка може бути більш реалістичною та адаптованою до дій гравця. Крім того, розширення арсеналу зброї додасть нових можливостей для гравців. Додавання нових місій, сценаріїв і історій до сюжетної лінії покращить ігровий досвід і надасть гравцям більше контенту для дослідження. Елементи користувацького інтерфейсу, такі як карти та індикатори місій, а також додаткова інформація про стан гравця та ворогів, зроблять гру більш зручною та інформативною.

Можна також подумати про впровадження мультиплеєрного режиму, який дозволить гравцям взаємодіяти один з одним, змагатися або працювати разом у командних місіях, що значно підвищить соціальний аспект. Доступність гри на різних платформах є важливою частиною розширення гри. Це дозволить більшій аудиторії насолоджуватися грою на різних пристроях.

Таким чином, незважаючи на те, що гра «Оборона стратегічного об’єкту» вже показує високий рівень якості, внесення вищезазначених змін може значно підвищити її привабливість і інтерес гравців, зробивши гру більш захоплюючою та різноманітною.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Світовий ігровий ринок продовжує демонструвати значне зростання, зумовлене низкою факторів, зокрема зміною споживчих уподобань, новими тенденціями та регіональними особливостями. Вподобання споживачів на ігровому ринку зміщуються в бік більш захоплюючих та інтерактивних ігор. За даними Statista на ринку у 2024 році дохід ігор сягає близько 455,30 млрд доларів США. До 2029 року прогнозується зростання обсягу ринку до 666,70 млрд доларів США за рахунок річного темпу зростання доходу 7,93% (рис. 4.1).

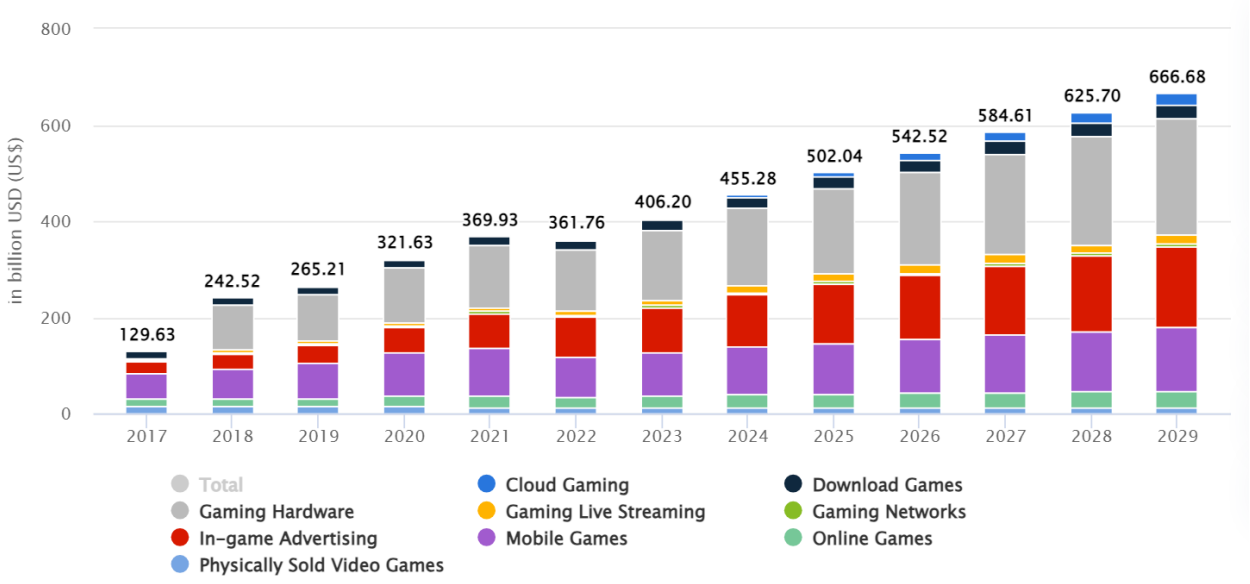

Рисунок 4.1 – Статистика доходу ігрового ринку

Розміри ринку жанру шутер станом на 2023 рік сягали 105,1 мільярдів доларів США, що показує ефективність монетизації таких ігрових проєктів. Лише культова франшиза “Call of Duty” принесла компанії Sony 1,5 мільярда доларів США за вартість самої гри, а беручи до уваги підписки, аксесуари та інше, дохід становив 15,9 мільярдів доларів США [15]. За всю історію ігрового маркетплейсу Steam найбільш прибутковою вважається і не менш популярна гра від компанії Valve “Counter-Strike: Global Offensive”. Дохід від продажів склав 6,7 мільярдів доларів США [16].

Аналітики стверджують, що до 2029 року на ринку ігор кількість користувачів становитиме 3,0 млрд. При цьому середній дохід на користувача складатиме 816,30 доларів США. (рис. 4.2) [17].

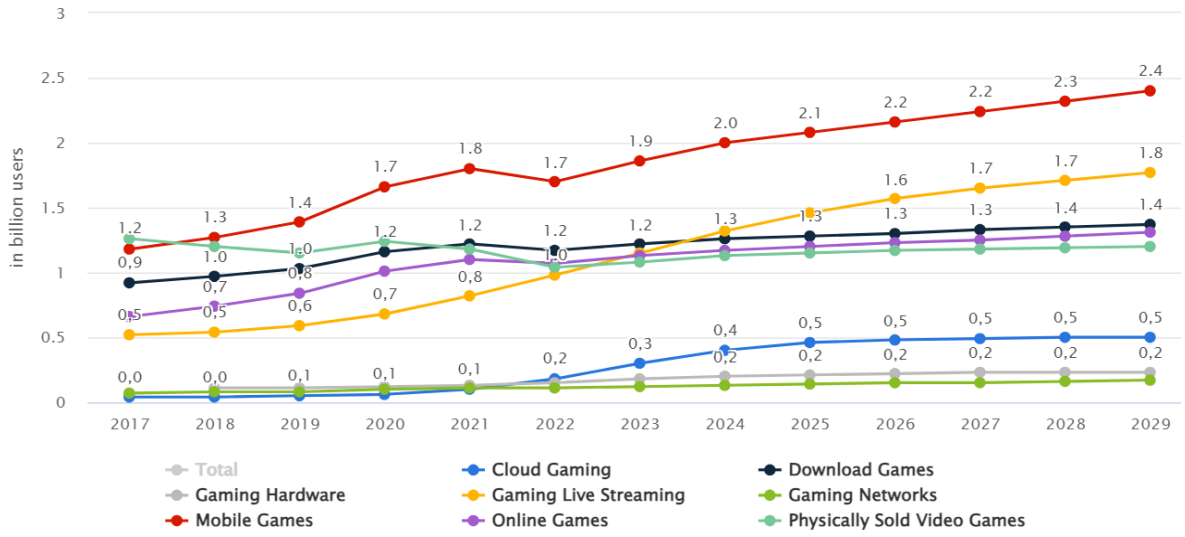

Рисунок 4.2 – Статистика користувачів на ринку ігор

Актуальність даного ігрового застосунку відповідає даним статистики та прогнозам аналітиків завдяки 3D-графіці та захоплюючому жанру. На ринку існує кілька конкурентів, які пропонують схожі продукти, їх перелік та характеристики зазначено у розділі 1.2 “Огляд та аналіз існуючих розробок”.

Потенційними замовниками (покупцями) ігрового застосунку є геймери різного віку, зокрема молодь та дорослі, які цікавляться жанром FPS та шукають високоякісні та захоплюючі ігрові продукти. Глобальні цифрові платформи, такі як Steam та Epic Games Store для ігрового досвіду на ПК, а також пізніше консолі PlayStation і Xbox, будуть основними ринками для збуту цього продукту.

Завдяки актуальності жанру та високій якості продукту очікується високий попит на гру. Цифрова дистрибуція через онлайн-магазини – це найкращий спосіб продажу, оскільки він дозволяє знизити витрати на логістику та охопити широку аудиторію. Організація сервісного обслуговування включатиме як допродажну підтримку (демонстрації, трейлери, рекламні кампанії), так і післяпродажне обслуговування (оновлення, патчі, додатковий

контент). Протягом перших років після запуску продукту очікується продаж тисяч копій, що забезпечить постійний дохід і дозволить інвестувати в подальший розвиток ігрової компанії.

4.2 Розрахунок витрат на проєктування

Заробітна плата розробників, соціальні витрати, прямі витрати, накладні витрати, накопичення та оподаткування – це усі витрати, пов’язані з розробкою гри. Розрахунок витрат на розробку застосунку з обґрунтуванням статей витрат подано у таблиці 1 додатку Е.

Як показано в таблиці, витрати на розробку проєкту становлять 2 248 984 грн. Основною частиною витрат є заробітна плата проектувальників у розмірі 817 678 грн для п’яти виконавців, яка визначається на основі діючих посадових окладів і кількості місяців роботи. Відрахування на соціальні потреби, включаючи податки та соціальні внески, становлять 56 631 грн. Інші прямі витрати становлять 327 071 грн, включаючи закупівлю обладнання, програмного забезпечення та матеріалів. Загальні господарські витрати становлять 360 414 грн накладних витрат. Планові накопичення становлять 312 359 грн для преміювання працівників і розвитку матеріально-технічної бази. Загальна кошторисна вартість проєкту до оподаткування становить 1 874 153 грн, а повна договірна вартість розробки – 2 248 984 грн із врахуванням ПДВ у розмірі 374 831 грн.

Розмір заробітної плати розробників проєкту залежить від складності роботи, досвіду виконавців та кількості місяців їх участі у розробці. Отже, для реалізації застосунку було залучено керівника проєкту, програмістів-розробників, дизайнера гри та тестувальника, розрахунок заробітної плати яких подано у таблиці 2 додатку Е.

Відповідно до витрат на заробітні плати виконавців можна розрахувати відрахування на соціальні потреби, які включають в себе єдиний соціальний внесок, військовий збір і податок на доходи фізичних осіб.

Керівник проєкту отримує місячну зарплату 40 438 грн. Розрахунок відрахувань:

– 18% податку на доходи фізичних осіб (ПДФО) становить 7 278,84 грн на місяць;

– військовий збір – 1,5% від зарплати, що становить 606,57 грн/міс;

– 22% єдиного соціального внеску (ЄСВ) становить 8 896,36 грн/міс.

Після вирахування ПДФО та військового збору працівникові залишається до виплати 32 553 грн на місяць.

Програміст-розробник заробляє 80 873 грн на місяць. Розрахунок відрахувань:

– 18% ПДФО становить 14 557,32 грн на місяць;

– військовий збір становить 1 213,11 грн на місяць;

– 22% ЄСВ становить 17 792,28 грн на місяць.

Після вирахування ПДФО та військового збору до виплати працівникові залишається 65 104 грн на місяць.

Дизайнер гри отримує 70 900 грн на місяць, відрахування становлять:

– військовий збір становить 1,5% або 1 063,50 грн на місяць;

– ПДФО становить 18% або 12 762 грн на місяць;

– ЄСВ становить 22%, що дорівнює 15 598 грн на місяць.

Після вирахування ПДФО та військового збору до виплати працівникові залишається 57 074 грн на місяць.

Тестувальник отримує 65 200 грн на місяць, відрахування становлять:

– військовий збір у розмірі 1,5% або 978 грн на місяць;

– ПДФО становить 18% або 11 736 грн;

– 22% ЄСВ становить 14 344 грн на місяць.

Після вирахування ПДФО та військового збору працівникові залишається 52 486 грн на місяць, щоб отримати виплати.

4.3 Обґрунтування необхідності розробки

Розробка нових високоякісних проєктів із використанням сучасних технологій є важливою в сучасному світі комп’ютерних ігор, щоб задовольнити попит гравців. Проєкт, розроблений у тривимірному середовищі, має багато переваг, які роблять його актуальним і захоплюючим для гравців.

По-перше, популярність жанру FPS гарантує високий попит на подібні ігри. Важливо мати високоякісну графіку, реалістичну фізику та складні алгоритми штучного інтелекту, оскільки гравці цінують динамічність геймплею та можливість поглибленого занурення у віртуальний світ.

По-друге, створення гри в такому форматі має і певний навчальний аспект. Щоб розвивати свої навички реакції та концентрації, гравці повинні швидко реагувати на зміни в грі та ефективно взаємодіяти з NPC. Крім того, гравець отримає навички управління ресурсами, тактичного планування та стратегічного мислення. Для успішного захисту об’єкту від ворожих атак гравцям доведеться приймати швидкі та обдумані рішення.

Економічні аспекти успіху та актуальності розробки полягають у здатності залучати та утримувати гравців через постійні оновлення та розширення контенту. Це дозволяє заробляти кошти не лише від продажу основної гри, але й від додаткових матеріалів, доповнень, що може включати нові місії, зброю, карти та персонажів. Популярність розробки дозволяє розширювати бренд і, як наслідок, створювати продукти в інших галузях, завдяки впізнаваності персонажів. Цей підхід забезпечує постійний прибуток протягом тривалого часу.

Таким чином, гра не лише забезпечить користувачам захоплюючий геймплей, але й навчить їх стратегічному мисленню, плануванню та реакції. Це, в свою чергу, підвищить лояльність гравців та сприятиме довготривалому успіху проєкту на ринку, що є важливим фактором з економічної точки зору.

ВИСНОВКИ

Ігровий застосунок “Оборона стратегічного об’єкту” відповідає сучасним вимогам ігрової індустрії, яка висуває високі стандарти до якості графіки та геймплею. Використання ігрового рушія Unity Engine версії 2022.3.16f1 забезпечило можливість створення динамічного та інтерактивного середовища, що дозволяє гравцям максимально поглибитися у віртуальний світ гри.

У процесі проектування було реалізовано UML-діаграми для моделювання структури програмного забезпечення. Ці діаграми включали діаграми класів для визначення взаємодії компонентів, діаграми послідовності для моделювання процесів та взаємодій між об'єктами. Це сприяло кращому розумінню архітектури і гнучкому плануванню розробки.

Компоненти ігрового рушія забезпечили якісну графічну частину застосунку. Terrain було використано для створення великого та деталізованого ігрового світу з різноманітним ландшафтом. Завдяки використанню NavMesh Agent було забезпечено навігацію NPC по створеній карті. Це забезпечило реалізацію рухів персонажів у грі природними та адаптивними до змін у середовищі. Менеджер анімацій дозволив створити плавні та реалістичні анімації для персонажів, що підвищило їхню емоційну виразність та взаємодію з гравцем. Усі звукові та візуальні ефекти були взяті з відкритих джерел інтернету, що зекономило витрати на розробку.

Застосунок було написано на мові програмування C# з використанням ряду бібліотек, а саме UnityEngine.AI, TMPro та UnityEngine.SceneManagement. Завдяки використанню UnityEngine.AI була реалізовано штучний інтелект NPC, що дозволило ворожим персонажам реалістично реагувати на дії гравця та взаємодіяти з оточенням. Для зручного відображення інформації про статус гравця та ігрові події було використано бібліотеку TMPro для користувацького інтерфейсу та текстового виводу. Бібліотека UnityEngine.SceneManagement

використовувалася для управління сценами гри, що забезпечило плавний перехід між рівнями та етапами гри, та іншими складовими інтерфейсу.

У процесі роботи було проведено аналіз ринку та конкурентних товарів, щоб розробити стратегію розвитку та просування розробки. У кінцевому результаті ігровий застосунок відповідає сучасним вимогам геймерів і конкурентоспроможний на ринку. Він не тільки задовольняє високі технічні стандарти, але й пропонує захоплюючий ігровий досвід, який розвиває стратегічне мислення та навички управління ресурсами гравців.

Результати дослідження та детального аналізу предметної області, програмного продукту висвітлювались у виступі на науково–практичній конференції в рамках "Днів науки – 2024", що проводилась у Галицькому коледжі імені В’ячеслава Чорновола [18].

У подальшому передбачено розширення функціоналу ігрового застосунку "Оборона стратегічного об'єкту", що включатиме нові рівні складності, додаткові можливості оборони та вдосконалення інтерфейсу користувача для зручності гравців. Планується розповсюдження ігрового застосунку "Оборона стратегічного об'єкту" на провідних ігрових платформах, що дозволить максимально охопити цільову аудиторію гравців і забезпечити доступність гри для широкого кола користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Комп'ютерна гра. *Енциклопедія Сучасної України ЕСУ*. URL: https://esu.com.ua/article-4393 (дата звернення: 10.02.2024).

2. Мисенко О. Комп'ютерна гра та авторське право. *Блоги про бізнес, політику, юридичну систему | LIGA.net - LIGA*. URL: https://blog.liga.net/user/amyisenko/article/38093 (дата звернення: 10.02.2024).

3. У 2023 році понад 500 ігор заробили більш ніж $3 мільйони доходу в Steam. *Українська спільнота GameDev DOU*. URL: https://gamedev.dou.ua/news/steam-games-ravenue/ (дата звернення: 13.02.2024).

4. Українська правда. Кіберспорт офіційно визнали видом спорту в Україні. *Українська правда*. URL: https://www.pravda.com.ua/news/2020/09/7/7265572/ (дата звернення: 13.02.2024).

5. Ukrainian Became the Best Cyberplayer of the Decade. *GTInvest*. URL: https://good-time-invest.com/blog/ukrainian-became-the-best-cyberplayer-of-the-decade/ (дата звернення: 13.02.2024).

6. The health effects of too much gaming – Harvard Health. *Harvard Health*. URL: https://www.health.harvard.edu/blog/the-health-effects-of-too-much-gaming-2020122221645 (дата звернення: 16.02.2024).

7. Computer Games. *Nanyang Technological University*. URL: https://www3.ntu.edu.sg/home/asschui/Computer%20Games.PDF (дата звернення: 16.02.2024).

8. Video Game Genres: Everything You Need to Know. *HP Development Company, L.P.* URL: https://www.hp.com/us-en/shop/tech-takes/video-game-genres (дата звернення: 16.02.2024).

9. Contributors to Wikimedia projects. Left 4 Dead - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Left_4_Dead (дата звернення: 19.02.2024).

10. Dead Alliance™ on Steam. *Welcome to Steam*. URL: https://store.steampowered.com/app/626200/Dead_Alliance/ (дата звернення: 19.02.2024).

11. Contributors to Wikimedia projects. DayZ (video game) - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/DayZ_(video_game) (дата звернення: 19.02.2024).

12. Unity проти Unreal Engine – який рушій обрати для гри та чому. Зважуємо всі за та проти з розробниками. *Українська спільнота GameDev DOU*. URL: https://gamedev.dou.ua/articles/unity-or-unreal-engine/ (дата звернення: 03.04.2024).

13. Godot vs Unity 2022: Which Game Engine is Best for You? | Pingle Studio. Pingle Studio. URL: https://pinglestudio.com/blog/full-cycle-development/godot-vs-unity-2022 (дата звернення: 03.04.2024).

14. Contributors to Wikimedia projects. Visual Studio - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Visual_Studio (date of access: 04.04.2024).

15. Call of Duty: Mobile Shoots Past $1.5 Billion in Lifetime Player Spending. Sensor Tower - Market-Leading Digital Intelligence. URL: https://sensortower.com/blog/call-of-duty-mobile-shoots-past-usd1-5-billion-in-lifetime-player-spending (дата звернення: 09.05.2024).

16. Analysts: CS:GO sales brought Valve record for Steam $6.7 billion. csgo.com. URL: https://csgo.com/news/96509-analysts-csgo-sales-brought-valve-record-for-steam-67-billion (дата звернення: 09.05.2024).

17. Games - Worldwide | Statista Market Forecast. Statista. URL: https://www.statista.com/outlook/amo/media/games/worldwide#key-players (дата звернення: 10.05.2024).

18. Створення комп’ютерної гри в популярному жанрі шутер. Біла А.Л. ЗБІРНИК тез за матеріалами студентських науково-практичних конференцій в рамках Днів Науки 2024 в Галицькому фаховому коледжі імені В’ячеслава Чорновола (секція «Комп’ютерних технологій» та секція «Фізико-математичних та природничих дисциплін») – Тернопіль: Галицький фаховий коледж імені В’ячеслава Чорновола, 2024. ___с.

ДОДАТКИ

Додаток А

Лістинг програмного коду звукових ефектів елементів гри

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

public class SoundManager : MonoBehaviour
{
    public static SoundManager Instance { get; set; }

    public AudioSource ShootingChannel;

    public AudioClip M1911;
    public AudioClip AK74Shot;

    public AudioSource reloadingSoundAK74;
    public AudioSource reloadingSoundM1911;

    public AudioSource emptyMagazineSoundM1911;

    public AudioClip militaryWalking;
    public AudioClip militaryChase;
    public AudioClip militaryAttack;
    public AudioClip militaryHurt;
    public AudioClip militaryDeath;

    public AudioSource militaryChannel;
    public AudioSource militaryChannel2;

    public AudioSource playerChannel;
    public AudioClip playerHurt;
    public AudioClip playerDie;

    public AudioClip gameOverMusic;

    public AudioClip bg_music;
    public AudioSource main_channel;

    private void Awake()
    {
        main_channel.PlayOneShot(bg_music);

        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
        }
```

```
        else
        {
            Instance = this;
        }
    }

    public void PlayShootingSound(WeaponModel weapon)
    {
        switch (weapon)
        {
            case WeaponModel.M1911:
                ShootingChannel.PlayOneShot(M1911);
                break;
            case WeaponModel.AK74:
                ShootingChannel.PlayOneShot(AK74Shot);
                break;
        }
    }

    public void PlayReloadSound(WeaponModel weapon)
    {
        switch (weapon)
        {
            case WeaponModel.M1911:
                reloadingSoundM1911.Play();
                break;
            case WeaponModel.AK74:
                reloadingSoundAK74.Play();
                break;
        }
    }
}
```

Додаток Б

Програмний код для головного персонажа

Лістинг Б.1 – Програмний код рухів головного персонажа

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

public class PlayerMovement : MonoBehaviour
{
    private CharacterController controller;

    public float speed = 12f;
    public float gravity = -9.81f * 2;
    public float jumpHeight = 3f;

    public Transform groundCheck;
    public float groundDistance = 0.4f;
    public LayerMask groundMask;

    Vector3 velocity;

    bool isGrounded;
    bool isMoving;

    private Vector3 lastPosition = new Vector3(0f,0f,0f);

    void Start()
    {
        controller = GetComponent<CharacterController>();
    }

    void Update()
    {
        isGrounded = Physics.CheckSphere(groundCheck.position,
groundDistance, groundMask);

        if (isGrounded && velocity.y < 0 )
        {
            velocity.y = -2f;
        }

        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");

        Vector3 move = transform.right * x + transform.forward *
z;

        controller.Move(move * speed * Time.deltaTime);

        if(Input.GetButtonDown("Jump") && isGrounded)
```

```
        {
            velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
        }

        velocity.y += gravity * Time.deltaTime;

        controller.Move(velocity * Time.deltaTime);

        if(lastPosition != gameObject.transform.position &&
isGrounded == true)
        {
            isMoving = true;
        }

        else
        {
            isMoving = false;
        }

        lastPosition = gameObject.transform.position;
    }
}
```

Лістинг Б.2 – Основний програмний код головного героя

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using static Military;
using static ScreenFader;
using UnityEngine.SceneManagement;


public class Player : MonoBehaviour
{
    public int HP = 100;

    public GameObject bloodyScreen;

    public TextMeshProUGUI playerHealthUI;
    public GameObject gameOverUI;

    public TextMeshProUGUI distanceToObjectiveUI;

    public bool isDead;

    private void Start()
    {
        playerHealthUI.text = $"Health: {HP}";
    }
```

```
    private void Update()
    {
        Vector3 playerPosition = transform.position;
        Vector3 strategicObjectPosition =
StrategicObject.GetStrategicObjectPosition();

        float distance = Vector3.Distance(playerPosition,
strategicObjectPosition);

        distanceToObjectiveUI.text = $"Distance to object:
{distance:F2}";
    }

    public void TakeDamage(int damageAmount)
    {
        HP -= damageAmount;

        if (HP <= 0)
        {
            print("Життя втрачено");
            PlayerDead();
            isDead = true;

        }
        else
        {
            print("Отримано удар");
            StartCoroutine(BloodyScreenEffect());
            playerHealthUI.text = $"Health: {HP}";

SoundManager.Instance.playerChannel.PlayOneShot(SoundManager.Insta
nce.playerHurt);
        }
    }

    private void PlayerDead()
    {
SoundManager.Instance.playerChannel.PlayOneShot(SoundManager.Insta
nce.playerDie);
        SoundManager.Instance.militaryChannel.Stop();

        SoundManager.Instance.playerChannel.clip =
SoundManager.Instance.gameOverMusic;
        SoundManager.Instance.playerChannel.PlayDelayed(2f);

        GetComponent<MouseMovement>().enabled = false;
        GetComponent<PlayerMovement>().enabled = false;

        GetComponentInChildren<Animator>().enabled = true;
        playerHealthUI.gameObject.SetActive(false);

        GetComponent<ScreenFader>().StartFade();
```

```
        StartCoroutine(ShowGameOverUI());
    }

    private IEnumerator ShowGameOverUI()
    {
        yield return new WaitForSeconds(1f);
        gameOverUI.gameObject.SetActive(true);

        StartCoroutine(ReturnToMainMenu());
    }

    private IEnumerator ReturnToMainMenu()
    {
        yield return new WaitForSeconds(8f);

        SceneManager.LoadScene("MainMenu");
    }


    private IEnumerator BloodyScreenEffect()
    {
        if(bloodyScreen.activeInHierarchy == false)
        {
            bloodyScreen.SetActive(true);
        }

        var image = bloodyScreen.GetComponentInChildren<Image>();

        Color startColor = image.color;
        startColor.a = 1f;
        image.color = startColor;

        float duration = 2f;
        float elapsedTime = 0f;

        while (elapsedTime < duration)
        {
            float alpha = Mathf.Lerp(1f, 0f, elapsedTime /
duration);

            Color newColor = image.color;
            newColor.a = alpha;
            image.color = newColor;

            elapsedTime += Time.deltaTime;

            yield return null;
        }


        if (bloodyScreen.activeInHierarchy)
        {
```

```
            bloodyScreen.SetActive(false);
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("MilitaryHand"))
        {
            if (isDead == false)
            {
                MilitaryHand militaryHand =
other.gameObject.GetComponent<MilitaryHand>();
                if (militaryHand != null)
                {

TakeDamage(other.gameObject.GetComponent<MilitaryHand>().damage);
                }
                else
                {
                    Debug.LogError("MilitaryHand компонент не
знайдено на об'єкті!");
                }
            }
        }
    }
}
```

Додаток В

Програмний код для зброї

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using static AmmoManager;
using static SoundManager;
using static WeaponSwitching;

public class Weapon : MonoBehaviour
{

    public int weaponDamage;

    public bool isShooting, readyToShoot;
    bool allowReset = true;
    public float shootingDelay = 2f;

    public int bulletsPerBurst = 3;
    public int burstBulletsLest;

    public float spreadIntensity;

    public GameObject bulletPrefab;
    public Transform bulletSpawn;
    public float bulletVelocity = 30;
    public float bulletPrefabLifeTime = 3f;

    public GameObject muzzleEffect;
    private Animator animator;

    public float reloadTime;
    public int magazineSize, bulletsLeft;
    public bool isReloading;

    public enum WeaponModel
    {
        M1911,
        AK74
    }

    public WeaponModel thisWeaponModel;

    bool isADS;

    public enum ShootingMode
    {
        Single,
        Burst,
        Auto
```

```
    }

    public ShootingMode currentShootingMode;

    private void Awake()
    {
        readyToShoot = true;
        burstBulletsLest = bulletsPerBurst;
        animator = GetComponent<Animator>();

        bulletsLeft = magazineSize;
    }

    void Update()
    {
        if (Input.GetKey(KeyCode.LeftControl) &&
Input.GetKeyDown(KeyCode.R))
        {
            if (isADS)
            {
                animator.SetTrigger("exitADS");
            }
            else
            {
                animator.SetTrigger("enterADS");
            }
            isADS = !isADS;
        }

        if (bulletsLeft == 0 && isShooting)
        {
            SoundManager.Instance.emptyMagazineSoundM1911.Play();
        }

        if(currentShootingMode == ShootingMode.Auto)
        {
            isShooting = Input.GetKey(KeyCode.Mouse0);
        }

        else if (currentShootingMode == ShootingMode.Single ||
            currentShootingMode == ShootingMode.Burst)
        {
            isShooting = Input.GetKeyDown(KeyCode.Mouse0);
        }

        if (Input.GetKeyDown(KeyCode.R) && bulletsLeft <
magazineSize && isReloading == false)
        {
            Reload();
        }
```

```
        if(readyToShoot && isShooting == false && isReloading ==
false && bulletsLeft <= 0)
        {
            Reload();
        }

        if (readyToShoot && isShooting && bulletsLeft > 0)
        {
            burstBulletsLest = bulletsPerBurst;
            FireWeapon();
        }

        if(AmmoManager.Instance.ammoDisplay != null)
        {
            AmmoManager.Instance.ammoDisplay.text =
$"{bulletsLeft/bulletsPerBurst}/{magazineSize/bulletsPerBurst}";
        }
    }

     private void FireWeapon ()
    {
        bulletsLeft--;

        muzzleEffect.GetComponent<ParticleSystem>().Play();

        if(isADS)
        {
            animator.SetTrigger("RECOIL_ADS");
        }
        else
        {
            animator.SetTrigger("RECOIL");
        }


        SoundManager.Instance.PlayShootingSound(thisWeaponModel);


        readyToShoot = false;

        Vector3 shootingDirection =
CalculateDirectionAndSpread().normalized;

        GameObject bullet = Instantiate(bulletPrefab,
bulletSpawn.position, Quaternion.identity);

        Bullet bul = bullet.GetComponent<Bullet>();
        bul.bulletDamage = weaponDamage;

        bullet.transform.forward = shootingDirection;
```

```
bullet.GetComponent<Rigidbody>().AddForce(bulletSpawn.forward.norm
alized * bulletVelocity, ForceMode.Impulse);

        StartCoroutine(DestroyBulletAfterTime(bullet,
bulletPrefabLifeTime));

        if(allowReset)
        {
            Invoke("ResetShot", shootingDelay);
            allowReset = false;
        }

        if(currentShootingMode == ShootingMode.Burst &&
burstBulletsLest > 1)
        {
            burstBulletsLest--;
            Invoke("FireWeapon", shootingDelay);
        }
    }

    private void Reload()
    {
        SoundManager.Instance.PlayReloadSound(thisWeaponModel);

        animator.SetTrigger("RELOAD");

        isReloading = true;
        Invoke("ReloadCompleted", reloadTime);
    }

    private void ReloadCompleted ()
    {
        bulletsLeft = magazineSize;
        isReloading = false;
    }

    private void ResetShot()
    {
        readyToShoot = true;
        allowReset = true;
    }

    public Vector3 CalculateDirectionAndSpread()
    {
        Ray ray = Camera.main.ViewportPointToRay(new Vector3(0.5f,
0.5f, 0));
        RaycastHit hit;

        Vector3 targetPoint;
        if(Physics.Raycast(ray, out hit))
        {
```

```
            targetPoint = hit.point;
        }
        else
        {
            targetPoint = ray.GetPoint(100);
        }

        Vector3 direction = targetPoint - bulletSpawn.position;

        float x = UnityEngine.Random.Range(-spreadIntensity,
spreadIntensity);
        float y = UnityEngine.Random.Range(-spreadIntensity,
spreadIntensity);

        return direction + new Vector3(x, y, 0);
    }

    private IEnumerator DestroyBulletAfterTime(GameObject bullet,
float delay)
    {
        yield return new WaitForSeconds(delay);
        Destroy(bullet);
    }
}
```

Додаток Г

Програмний код для ворогів

Лістинг Г.1 – Основний програмний код для ворогів

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

public class Enemy : MonoBehaviour
{
    [SerializeField] private int HP = 100;
    private Animator animator;

    private NavMeshAgent navAgent;

    public bool isDead;

    public Transform target;
    public float attackDistance = 2.5f;
    public int damageAmount = 10;

    void Start()
    {
        animator = GetComponent<Animator>();
        navAgent = GetComponent<NavMeshAgent>();
        target =
GameObject.FindGameObjectWithTag("StrategicObject").transform;
    }

    void Update()
    {
        if (isDead) return;

        navAgent.SetDestination(target.position);

        if (Vector3.Distance(transform.position, target.position)
<= attackDistance)
        {
            animator.SetTrigger("isAttacking");
        }
    }

    public void TakeDamage(int damageAmount)
    {
        HP -= damageAmount;

        if (HP <= 0)
        {
            int randomValue = Random.Range(0,2);
```

```
            if (randomValue == 0)
            {
                animator.SetTrigger("DIE1");
            }
            else
            {
                animator.SetTrigger("DIE2");
            }

            isDead = true;


SoundManager.Instance.militaryChannel2.PlayOneShot(SoundManager.In
stance.militaryDeath);
        }
        else
        {
            animator.SetTrigger("DAMAGE");


SoundManager.Instance.militaryChannel2.PlayOneShot(SoundManager.In
stance.militaryHurt);
        }
    }

    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, 2.5f);

        Gizmos.color = Color.blue;
        Gizmos.DrawWireSphere(transform.position, 100f);

        Gizmos.color = Color.green;
        Gizmos.DrawWireSphere(transform.position, 101f);
    }
}
```

Лістинг Г.2 – Програмний код для стану очікування ворогів

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


public class EnemyPatrolingState : StateMachineBehaviour
{
    float timer;
    public float patrolingTime = 10f;

    Transform player;
    NavMeshAgent agent;
```

```
    public float detectionArea = 18f;
    public float patrolSpeed = 2f;

    List<Transform> waypointsList = new List<Transform>();

    override public void OnStateEnter(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        player =
GameObject.FindGameObjectWithTag("Player").transform;
        agent = animator.GetComponent<NavMeshAgent>();

        agent.speed = patrolSpeed;
        timer = 0;

        GameObject waypointCluster =
GameObject.FindGameObjectWithTag("Waypoints");
        foreach (Transform t in waypointCluster.transform)
        {
            waypointsList.Add(t);
        }

        Vector3 nextPOsition = waypointsList[Random.Range(0,
waypointsList.Count)].position;
        agent.SetDestination(nextPOsition);
    }

    override public void OnStateUpdate(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        if (SoundManager.Instance.militaryChannel.isPlaying ==
false)
        {
            SoundManager.Instance.militaryChannel.clip =
SoundManager.Instance.militaryWalking;
            SoundManager.Instance.militaryChannel.PlayDelayed(1f);
        }

        if (agent.remainingDistance <= agent.stoppingDistance)
        {
            agent.SetDestination(waypointsList[Random.Range(0,
waypointsList.Count)].position);
        }

        timer += Time.deltaTime;
        if (timer > patrolingTime)
        {
            animator.SetBool("isPatroling", false);
        }
```

```
        float distanceFromPlayer =
Vector3.Distance(player.position, animator.transform.position);

        if (distanceFromPlayer < detectionArea)
        {
            animator.SetBool("isChasing", true);
        }
    }


    override public void OnStateExit(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        agent.SetDestination(agent.transform.position);

        SoundManager.Instance.militaryChannel.Stop();
    }
}
```

Лістинг Г.3 – Програмний код для стану нападу ворогів

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

public class EnemyAttackState : StateMachineBehaviour
{
    Transform player;
    UnityEngine.AI.NavMeshAgent agent;

    public float stopAttackingDistance = 2.5f;


    override public void OnStateEnter(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        player =
GameObject.FindGameObjectWithTag("Player").transform;
        agent =
animator.GetComponent<UnityEngine.AI.NavMeshAgent>();
        agent.isStopped = true;
    }

    override public void OnStateUpdate(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        if (SoundManager.Instance.militaryChannel.isPlaying ==
false)
        {

SoundManager.Instance.militaryChannel.PlayOneShot(SoundManager.Ins
tance.militaryAttack);
        }
```

```
        LookAtPlayer();

        float distanceFromPlayer =
Vector3.Distance(player.position, animator.transform.position);

        if (distanceFromPlayer < stopAttackingDistance)
        {
            animator.SetBool("isAttacking", true);

        }

        else
        {
            animator.SetBool("isAttacking", false);
            animator.SetBool("isChasing", true);
        }

    }

    override public void OnStateExit(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        agent.isStopped = false;
        SoundManager.Instance.militaryChannel.Stop();
    }

    private void LookAtPlayer()
    {
        Vector3 direction = player.position -
agent.transform.position;

        direction.y = 0;
        Quaternion lookRotation =
Quaternion.LookRotation(direction);
        agent.transform.rotation =
Quaternion.Slerp(agent.transform.rotation, lookRotation,
Time.deltaTime * 5f);
    }
}
```

Лістинг Г.4 – Програмний код для стану спокою ворогів

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

public class EnemyIdleState : StateMachineBehaviour
{
    float timer;
    public float idleTime = 0f;

    Transform player;
```

```
    public float detectionAreaRadius = 18f;

    override public void OnStateEnter(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        timer = 0;
        player =
GameObject.FindGameObjectWithTag("Player").transform;
    }

    override public void OnStateUpdate(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        timer += Time.deltaTime;
        if (timer > idleTime)
        {
            animator.SetBool("isPatroling", true);
        }

        float distanceFromPlayer =
Vector3.Distance(player.position, animator.transform.position);

        if (distanceFromPlayer < detectionAreaRadius)
        {
            animator.SetBool("isChasing", true);
        }
    }
}
```

Лістинг Г.5 – Програмний код для хвилі ворогів

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Random = UnityEngine.Random;
using TMPro;

public class MilitarySpawnController : MonoBehaviour
{
    public int initialMilitaryPerWave = 5;
    public int currentMilitaryPerWave;

    public float spawnDelay = 0.5f;

    public int currentWave = 0;
    public float waveCooldown = 10.0f;

    public bool inCooldown;
    public float cooldownCounter = 0;

    public List<Enemy> currrentMilitaryAlive;
```

```
    public GameObject militaryPrefab;

    public TextMeshProUGUI WaveOverUI;
    public TextMeshProUGUI cooldownCouterUI;
    public TextMeshProUGUI currentWaveUI;

    void Start()
    {
        currentMilitaryPerWave = initialMilitaryPerWave;

        GlobalReferences.Instance.waveNumber = currentWave;

        StartNextWave();
    }

    private void StartNextWave()
    {
        currrentMilitaryAlive.Clear();

        currentWave++;

        GlobalReferences.Instance.waveNumber = currentWave;

        currentWaveUI.text = "Wave: " + currentWave.ToString();

        StartCoroutine(SpawnWave());
    }

    private IEnumerator SpawnWave()
    {
        for(int i = 0; i < currentMilitaryPerWave; i++)
        {
            Vector3 spawnOffset = new Vector3(Random.Range(-1f,
1f), 0f, Random.Range(-1f, 1f));
            Vector3 spawnPosition = transform.position +
spawnOffset;

            var military = Instantiate(militaryPrefab,
spawnPosition, Quaternion.identity);

            Enemy enemyScript = military.GetComponent<Enemy>();

            currrentMilitaryAlive.Add(enemyScript);
            yield return new WaitForSeconds(spawnDelay);
        }
    }

    void Update()
    {
        List<Enemy> militaryToRemove = new List<Enemy>();
        foreach(Enemy military in currrentMilitaryAlive)
        {
```

```
            if(military.isDead)
            {
                militaryToRemove.Add(military);
            }
        }

        foreach(Enemy military in militaryToRemove)
        {
            currrentMilitaryAlive.Remove(military);
        }

        militaryToRemove.Clear();

        if(currrentMilitaryAlive.Count == 0 && inCooldown ==
false)
        {
            StartCoroutine(WaveCooldown());
        }

        if(inCooldown)
        {
            cooldownCounter -= Time.deltaTime;
        }

        else
        {
            cooldownCounter = waveCooldown;
        }

        cooldownCouterUI.text = cooldownCounter.ToString("F0");
    }

    private IEnumerator WaveCooldown()
    {
        inCooldown = true;
        WaveOverUI.gameObject.SetActive(true);

        yield return new WaitForSeconds(waveCooldown);

        inCooldown = false;
        WaveOverUI.gameObject.SetActive(false);

        currentMilitaryPerWave *= 2;
        StartNextWave();
    }
}
```

Додаток Д

Програмний код головного меню

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

public class MainMenu : MonoBehaviour
{
    string newGameScene = "SampleScene";
    string additionalScene = "Scene_A";

    public AudioClip bg_music;
    public AudioSource main_channel;

    void Start()
    {
        main_channel.PlayOneShot(bg_music);
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }

    public void StartNewGame()
    {
        main_channel.Stop();

        SceneManager.LoadScene(newGameScene);

        SceneManager.LoadScene(additionalScene, LoadSceneMode.Additive);
    }

    public void ExitApplication()
    {
#if UNITY_EDITOR
    UnityEditor.EditorApplication.isPlaying = false;
#else
        Application.Quit();
#endif
    }


}
```

Додаток Е

Таблиця Е.1 – Кошторис витрат на проєктування

Найменування статей витрат	Сума, грн	Обґрунтування
1 Зарплата проектувальників	817 678	Заробітна плата працівників розраховується на основі п’яти виконавців, діючих посадових окладів і кількості місяців, проведених ними під час розробки проекту.
2. Відрахування на соціальні потреби	56 631	Включає в себе відрахування на соціальні програми та потреби працівників такі, як податки на доходи фізичних осіб, військовий збір, єдиний внесок.
3. Контрагентські роботи і послуги	0	Не передбачено.
4. Витрати на відрядження	0	Не передбачено.
5. Інші прямі витрати	327 071	Витрати на закупівлю матеріалів, програмне забезпечення, обладнання.
6. Усього прямих витрат	1 201 380	Загальна вартість прямих витрат на реалізацію проєкту.
7. Накладні витрати	360 414	Загальногосподарчі витрати на виконання роботи включають витрати на опалення та електроенергію, а також інші загальні витрати.
8. Планові накопичення	312 359	Прибуток, що йде на розвиток матеріально-технічної бази організації і преміювання працівників.
9.Усього, кошторисна вартість проєкту	1 874 153	Сума всіх вищезазначених витрат, яка визначає загальну кошторисну вартість проєкту.
10. Податок на додану вартість	374 831	Відображає податок на додану вартість.
11. Загалом, договірна ціна розробки	2 248 984	Сума, за яку виконавець зобов’язаний виконати розробку в рамках договору, що включає суму всіх вищезазначених витрат та прибуток.

Таблиця Е.2 – Розрахунок заробітної плати проєктувальників

№ п/п	Посада виконавця	Оклад, грн/міс	Відрахування, грн/міс	Кількість		Сума з/п, грн.
				чол.	місяців	
1	Керівник проєкту	40 438	7 885,41	1	4	130 212
2	Програміст-розробник	80 873	15 770,43	2	4	520 832
3	Дизайнер гри	70 900	13 825,5	1	2	114 148
4	Тестувальник	65 200	12 714	1	1	52 486
		Усього зарплати:				817 678