Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення телекомунікацій та електронних систем
(назва відділення)
Циклова комісія комп'ютерної інженерії
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітній ступінь)

на тему: **Розробка 2D-гри жанру Top-Down Shooter на ігровому рушії Unity**

Виконав: студент VI курсу, групи КІб-602

Спеціальності: **123 Комп'ютерна інженерія**
(шифр і назва спеціальності)

_____ Анатолій СИРИМУЛА
(підпис) (ім'я та прізвище)

Керівник _____ Леся ШТОКАЛО
(підпис) (ім'я та прізвище)

Рецензент _____
(підпис) (ім'я та прізвище)

Тернопіль – 2025

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерної інженерії
Освітній ступінь бакалавр
Освітньо-професійна програма: Комп'ютерна інженерія
Спеціальність 123 Комп'ютерна інженерія
Галузь знань: 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"06" травня 2025 року

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

Сиримулі Анатолію Олеговичу
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи: **Розробка 2D-гри жанру Top-Down Shooter на ігровому рушії Unity**

керівник роботи: Штокало Леся Ярославівна

затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя» від 05.05.2025 р. № 4/9-217.

2. Строк подання студентом кваліфікаційної роботи 20 червня 2025 року.

3. Вихідні дані до роботи: Unity 2022.3.62f1 Visual Studio 2022 GIMP 3.0.4 FL Studio 2024.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічний розділ. Безпека життєдіяльності, основи охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Опис методу розробки програмного забезпечення. Діаграма послідовності виконання функціональних етапів гри. Алгоритм бойової системи у грі. Структура файлів і каталогів. Варіанти дій гравця у грі. Таблиця техніко-економічних показників

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Оксана РЕДЬКВА заст. директора з НВР		
Безпека життєдіяльності, основи охорони праці	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	06.05	
2	Збір і узагальнення інформації по роботі	19.05	
3	Написання першого розділу	23.05	
4	Розробка технічного та робочого проекту	28.05	
5	Написання спеціального розділу	03.06	
6	Розрахунок економічної частини	05.06	
7	Написання розділу охорони праці	07.06	
8	Виконання графічної частини	12.06	
9	Оформлення проекту	16.06	
10	Проходження нормоконтролю	18.06	
11	Попередній захист роботи	20.06	
12	Захист роботи		

7. Дата видачі завдання 06 травня 2025 року

Студент _____ Анатолій СИРИМУЛА
(підпис) (ім'я та прізвище)

Керівник роботи _____ Леся ШТОКАЛО
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Метою кваліфікаційної роботи бакалавра є розробка 2D-гри жанру Top-Down Shooter на ігровому рушії Unity.

У кваліфікаційній роботі представлено розробку комп’ютерної 2D-гри у жанрі top-down shooter під назвою "Void Rush", створеної за допомогою рушія Unity.

Кваліфікаційна робота включає реалізацію основних ігрових механік: управління персонажем, стрільба, система здоров’я, ШІ ворогів, рівні прокачки, інтерфейс користувача. Детально описано процес розробки, побудову алгоритмів, створення графічного та звукового контенту.

У роботі проаналізовано інструменти розробки, обґрунтовано вибір рушія Unity та середовища Visual Studio.

Проведено тестування готового продукту, а також розраховано техніко-економічні показники. Гра орієнтована на демонстрацію ключових навичок у галузі геймрозробки в навчальному середовищі.

Ключові слова: 2D-гра, Unity, top-down shooter, рушій, ігрова механіка, графічний інтерфейс, програмування, геймдизайн, C#, штучний інтелект.

ANNOTATION

The purpose of the bachelor's thesis is to develop a 2D game of the Top-Down Shooter genre on the Unity game engine.

This qualification work presents the development of a 2D top-down shooter game titled “Void Rush”, created using the Unity game engine. The project implements core game mechanics including player movement, shooting system, health and enemy AI, experience-based level-up system, and user interface.

The thesis outlines the full cycle of game development – from initial concept and prototyping to implementation and testing. The software environment includes Unity 2022, C# programming in Visual Studio 2022, GIMP for graphics, and FL Studio for sound effects.

The economic and organizational aspects of development are also considered. The game is designed as an educational and demonstration product
in the field of independent game development.

Keywords: 2D game, Unity, top-down shooter, engine, game mechanics, UI, programming, game design, C#, AI.

ЗМІСТ

ВСТУП

Сучасна цифрова епоха характеризується стрімким розвитком інформаційних технологій, у тому числі й індустрії розробки ігор. Відеоігри вже давно вийшли за межі розваг і стали окремою формою мистецтва, способом взаємодії з користувачем, потужним інструментом навчання, а також полем для реалізації творчого потенціалу програмістів, дизайнерів та художників. Ігрова індустрія генерує мільярдні доходи та активно впливає на інші сфери людської діяльності, включаючи культуру, освіту, психологію та економіку.

У межах навчального процесу розробка гри є ефективним способом застосування на практиці теоретичних знань, отриманих під час вивчення курсів програмування, алгоритмізації, об’єктно-орієнтованого проєктування, систем керування даними, графіки, UI/UX дизайну та інших технічних дисциплін. Саме тому створення гри в середовищі Unity було обрано як тема кваліфікаційної роботи бакалавра.

Головною метою кваліфікаційної роботи бакалавра є створення функціональної 2D-гри жанру top-down shooter з назвою "Void Rush", яка реалізована на основі рушія Unity із використанням мови програмування C#. У процесі розробки даної роботи передбачено створення повного циклу гри — від ігрової логіки до графічного та звукового оформлення, включаючи інтерфейс користувача та механіку розвитку персонажа.

У роботі було приділено увагу всім аспектам створення ігрового програмного продукту, зокрема побудові архітектури гри, розробці скриптів, реалізації інтерфейсу, тестуванню, а також питанням техніки безпеки, екологічності та економічної доцільності роботи. Досліджено аналогічні розробки, обґрунтовано вибір технологій і засобів розробки, а також проаналізовано переваги та недоліки реалізованого рішення.

У процесі розробки передбачено охоплення повного циклу створення гри. Це включає розробку та впровадження основної ігрової логіки, що забезпечить динамічний та захопливий геймплей, характерний для жанру top-down shooter.

Особлива увага буде приділена графічному та звуковому оформленню, яке створить атмосферне та візуально привабливе середовище для гравця.

Крім того, не менш важливими аспектами є розробка інтуїтивно зрозумілого інтерфейсу користувача (UI), який полегшить взаємодію гравця з грою, та впровадження комплексної механіки розвитку персонажа.

Це дозволить гравцям покращувати свого героя, відкривати нові здібності та адаптувати ігровий процес під свій стиль гри.

Таким чином, кваліфікаційна робота бакалавра спрямована не лише на досягнення освітньої мети, але й на набуття практичного досвіду створення ігор, розуміння структури програмних продуктів і наближення до професійного рівня розробника програмного забезпечення.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Індустрія відеоігор є однією з найдинамічніших галузей цифрового світу, демонструючи стабільне зростання як у технічному, так і в творчому плані. Особливе місце в цьому сегменті займають 2D-ігри, які, попри відносну простоту реалізації, залишаються популярними завдяки своїй доступності, низьким вимогам до ресурсів та широким можливостям для реалізації ігрових ідей.

Одним із найпоширеніших під жанрів 2D-ігор є топ-даун шутери. У таких іграх гравець керує персонажем, якого видно згори (вигляд зверху), і взаємодіє з великою кількістю супротивників або об’єктів на рівні. Гравець, зазвичай, має можливість стріляти у будь-якому напрямку, переміщаючись по двовимірному простору. Цей жанр відзначається високою динамікою, простим керуванням і можливістю реалізації широкого спектру ігрових механік, що забезпечує гравцям цікаві та насичені враження.

Ці ігри отримали свою назву через особливу перспективу камери , яка розташована зверху, дивлячись прямо вниз на ігрове поле та персонажа . Така камера дозволяє гравцеві бачити значну частину навколишнього середовища, ворогів та об'єктів одночасно, що є ключовим для ігрового процесу.

У top-down шутерах гравець зазвичай керує персонажем, що вільно пересувається по карті та використовує різноманітну зброю для знищення ворогів.

Динаміка гри часто базується на швидких рефлексах, стратегічному позиціонуванні та ефективному використанні оточення. Прикладами таких ігор є класичні аркадні тайтли, такі як "Alien Breed" , "Hotline Miami" або сучасні інді-хіти, що використовують подібну механіку. Цей жанр відзначається своєю доступністю , адже освоїти базові механіки досить просто, але водночас він пропонує значну глибину через різноманітність ворогів, видів зброї, рівнів складності та можливостей для та-

ктичного маневру. Механіки розвитку персонажа, як от покращення зброї чи навичок, додатково збагачують ігровий досвід, дозволяючи гравцям відчути прогрес та адаптувати гру під свій стиль.

Топ-даун шутери мають довгу історію, що бере початок із класичних аркадних ігор, як-от Robotron: 2084 (1982), Smash TV (1990), а пізніше еволюціонують у сучасні успішні проєкти, зокрема The Binding of Isaac, Enter the Gungeon, Nuclear Throne тощо. Ці ігри використовують інтенсивну бойову систему, процедурну генерацію (у більшості), велику кількість зброї та ворогів, що сприяє високій реіграбельності.

З-поміж інструментів для створення 2D-ігор варто згадати такі рушії, як GameMaker Studio, Godot Engine, Construct та Unity.

– GameMaker Studio часто використовується для створення піксельних 2D-проєктів і підтримує як візуальне, так і текстове програмування (мова GML). GameMaker Studio - програма, кросплатформний ігровий рушій для створення 2D ігор будь-якого жанру. Має вбудовану мову програмування GML (GameMaker Language), схожу на JavaScript. У версії з 1.3 і старше є вбудована система YoYo Games Marketplace.

– Godot Engine є рушієм із відкритим вихідним кодом, підтримує 2D та 3D графіку, має власну мову GDScript, зручний для початківців і активно розвивається. Це безкоштовний, з відкритим вихідним кодом, кросплатформовий 2D і 3D ігровий рушій, ліцензований MIT. Розробники з Godot Engine Community працюють над ним. Раніше він використовувався в деяких компаніях в Латинській Америці до того, як був випущений як відкрите програмне забезпечення. [16]

Godot постачається з повноцінним редактором ігор в який інтегровані інструменти для вирішення найпоширеніших потреб. Він включає в себе редактор коду, редактор анімації, редактор карт плитки, редактор шейдерів, зневаджувач, профайлер і багато іншого. [16]

– Construct — це браузерне середовище, яке дозволяє створювати прості ігри без написання коду взагалі, однак має обмеження щодо складних механік.

Програма Construct — ідеальне рішення для початківців програмістів і розробки простих ігор різних жанрів і складності в 2D графіку. При використанні цього програмного забезпечення не потрібно знати мови програмування. Таке рішення підійде людям, які мають намір швидко і легко створити свою гру або намагаються відійти від застарілої технології Flash. Основою є движок, розроблений із застосуванням HTML5. Construct доступний в безкоштовній версії, але вона має безліч обмежень, що дозволяють використовувати до 100 різних подій. Включає безліч інструментів і плагінів для: обертання, масштабування об'єктів, роботи з графікою і звуковими ефектами; установник послідовності дій; фізичні моделі ігрових можливостей. Створені ігри такого типу в різних операційних системах, їх можна тестувати разом з друзями через браузер, створювати власні плагіни для збільшення продуктивності програмного забезпечення. [15]

На офіційному сайті виробника можна завантажити додаткові звуки, спецефекти або інші надбудови, що розширюють можливості редактора.

Особливу увагу варто приділити Unity, який було обрано для реалізації цієї кваліфікаційної роботи.

Unity — це потужний кросплатформенний рушій, який дозволяє створювати як 2D, так і 3D ігри. Його особливості включають:

– Гнучкий компонентний підхід до побудови об’єктів (через GameObject та компоненти);

– Мову програмування C#, яка є стандартом у середовищі розробки;

– Вбудовану фізику 2D та 3D, систему частинок, анімації, звуку, UI, системи навігації;

– Модульну архітектуру, що дозволяє розробляти проєкт поетапно та масштабувати його в майбутньому;

– Широку підтримку та документацію, а також доступ до Asset Store, де можна знайти безліч готових ресурсів.

Unity — багатоплатформовий інструмент для розроблення відеоігор і застосунків, і рушій, на якому вони працюють. Створені за допомогою Unity

програми працюють на настільних комп'ютерних системах, мобільних пристроях та гральних консолях у дво- та тривимірній графіці, та на пристроях віртуальної чи доповненої реальності. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL. [1]

Редактор Unity має інтерфейс, що складається з різних вікон, які можна розташувати на свій розсуд. Завдяки цьому можна проводити налагодження гри чи застосунка прямо в редакторі. Головні вікна — це оглядач ресурсів проєкту, інспектор поточного об'єкта, вікно попереднього перегляду, оглядач сцени та оглядач ієрархії ресурсів. [1]

Проєкт в Unity поділяється на сцени (рівні) — окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як об'єкти-моделі (ландшафт, персонажі, предмети довкілля тощо), так і порожні ігрові об'єкти — ті, що не мають моделі, проте задають поведінку інших об'єктів (тригери подій, точки збереження прогресу тощо). Їх дозволяється розташовувати, обертати, масштабувати, застосовувати до них скрипти. В них є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов'язково наявний компонент Transform — він зберігає в собі координати місця розташування, повороту і розмірів по всіх трьох осях. У об'єктів з видимою геометрією також за умовчанням присутній компонент Mesh Renderer, що робить модель видимою. Різні моделі можуть об'єднуватися в набори (ассети) для швидкого доступу до них. Наприклад, моделі споруд на спільну тему. [14]

Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів. [14]

У 2D іграх Unity переважно використовує спрайти. В 3D іграх Unity здебільшого використовує тривимірні моделі (меші), на які накладаються текстури (зумовлюють вигляд поверхні об'єктів), матеріали (зумовлюють як поверхня

реагуватиме на різні фактори) та шейдери (невеликі скрипти, за яким вираховується зміна кольору кожного пікселя згідно заданих параметрів, як-от розсіяння відбитого світла). В обох видах застосовуються системи часток для відображення субстанцій, таких як рідини чи дим. [14]

Unity підтримує стиснення текстур, міпмапінг і різні налаштування роздільності екрана для кожної платформи; забезпечує бамп-мапінг, мапінг відображень, паралакс-мапінг, затінення навколишнього світла у екранному просторі, динамічні тіні за картами тіней, рендер у текстуру та повноекранні ефекти обробки зображення, такі як зернистість, глибина чіткості, розмиття в русі, відблиски віртуальних лінз або ореол навколо джерел світла. [14]

У цій кваліфікаційній роботі використовувалася версія Unity 2022.3.62f1 — одна з останніх LTS (Long-Term Support) версій, яка забезпечує стабільність роботи. Для створення графіки було застосовано GIMP 3.0.4, для звукових ефектів — FL Studio 2024, а для написання коду — Visual Studio 2022.

FL Studio (раніше Fruity Loops) — редактор-секвенсер для написання музики, створений 1997 року програмістом Дідьє Дембреном (також відомим під псевдонімом «gol»), який розробляв цю програму вісім років, і що випускається компанією Image-Line Software. [7]

Музика створюється шляхом запису і зведення (звукозапису) аудіо-, або MIDI-матеріалу. Готова композиція може бути записана у файл з розширенням WAV, MP3 або OGG. Програма написана мовою програмування Delphi. [17]

Visual Studio — серія продуктів фірми Майкрософт, які містять інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці продукти дають змогу розробляти як консольні програми, так і програми з графічним інтерфейсом, включно з підтримкою технології Windows Forms, а також вебсайти, вебзастосунки, вебслужби як у рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight. [18]

Visual Studio 2022 включає масштаб для роботи над проектами будь-якого розміру і складності в 64-бітному IDE. Код за допомогою нового редактора Razor, який може рефакторировать між файлами. Діагностувати проблеми з візуалізацією асинхронних операцій і застосуванням автоматичних аналізаторів. [8]

У контексті розробки даної кваліфікаційної роботи важливо відзначити, що більшість популярних ігор жанру мають розширені функції, зокрема роуглайк-структуру, мережеву гру, кооперативний режим, величезну кількість апгрейдів тощо. Водночас у рамках кваліфікаційної роботи доцільно зосередитись на реалізації базових механік, які можуть бути легко масштабовані в майбутньому: управління, стрільба, ШІ ворогів, прокачка та інтерфейс.

Таким чином, кваліфікаційна робота гри орієнтований на поєднання класичних механік жанру з помірною складністю реалізації, що дозволяє в повній мірі продемонструвати навички програмування, дизайну та логіки в рамках академічної розробки.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Тема даної кваліфікаційної роботи: «Розробка 2D-гри жанру Top-Down Shooter на ігровому рушії Unity».

Область застосування: навчальний, демонстраційний і розважальний програмний продукт, розроблений у рамках кваліфікаційної роботи для бакалаврського рівня підготовки. Гра призначена для запуску на персональних комп’ютерах під керуванням операційної системи Windows. Вона демонструє базові принципи побудови ігрових систем, взаємодії між об’єктами, реалізації ШІ, графічного інтерфейсу та звукового супроводу.

Програмне забезпечення може бути використане для демонстрації навичок програмування, створення інтерактивного інтерфейсу користувача, а також як

платформа для подальшої розробки більш складних ігор або навчальних проєктів у галузі розробки ігор.

1.2.2 Призначення розробки

Метою даної розробки є створення навчально-демонстраційного програмного продукту — 2D-гри у жанрі top-down shooter. В процесі розробки її було названо "Void Rush".

Головне призначення гри — продемонструвати основні принципи побудови ігрової логіки, взаємодії об’єктів у віртуальному просторі, реалізації базових алгоритмів штучного інтелекту ворогів, а також інтеграції користувацького інтерфейсу та звукового супроводу.

Гра також служить засобом закріплення знань у сфері об’єктно-орієнтованого програмування, роботи з ігровими рушіями (зокрема, Unity), розробки графічного контенту, побудови логіки гри та управління процесами в реальному часі.

Програмний продукт може бути застосований у навчальному процесі як приклад реалізації ігрового проєкту, а також як шаблон для розширення функціоналу в майбутніх індивідуальних або командних проєктах з ігрової розробки.

1.2.3 Вимоги до програмного забезпечення

У процесі розробки гри висуваються наступні вимоги до програмного забезпечення:

Функціональні вимоги:

– Можливість керування гравцем у двовимірному просторі (рух у 4 напрямках);

– Реалізація бойової механіки: стрільба, шкоди, знищення ворогів;

– Наявність ворогів зі спрощеним штучним інтелектом (слідування за гравцем);

– Система здоров’я для гравця і ворогів;

– Збір досвіду, система рівнів і покращень (Level Up);

– Відображення інтерфейсу користувача (панелі HP, XP, меню апгрейду);

– Головне меню та екран завершення гри.

– Нефункціональні вимоги:

– Висока стабільність роботи (відсутність критичних помилок);

– Продуктивність: стабільна частота кадрів на ПК середнього рівня;

– Простота і зручність управління за допомогою клавіатури і миші;

– Підтримка екранів з різною роздільною здатністю;

– Можливість розширення функціоналу у майбутньому.

Програмне середовище:

- Unity 2022.3.62f1 — рушій гри;
- Visual Studio 2022 — розробка логіки на мові C#;
- GIMP 3.0.4 — створення/редагування графіки;
- FL Studio 2024 — створення звукового супроводу.

1.2.4 Вимоги до програмної документації

Програмна документація до кваліфікаційної роботи повинна забезпечувати повну інформованість користувача та розробника про особливості функціонування гри, правила використання, встановлення та можливості подальшої модифікації.

Документація має включати:

- титульний аркуш із загальними відомостями про розробку;
- пояснювальну записку до кваліфікаційної роботи (структурована відповідно до ДСТУ);
- технічне завдання із зазначенням усіх функціональних і нефункціональних вимог;

- інструкцію з інсталяції програмного забезпечення (розділ 3.1);
- інструкцію з використання гри (розділ 3.3);
- блок-схеми, схеми алгоритмів, приклади коду для ілюстрації логіки функціонування гри;
- коментарі до коду та специфікації класів і об’єктів;
- інструкцію з використання тестових наборів (розділ 3.2) — за наявності;
- перелік використаних джерел і додатків.

Вся документація має бути оформлена згідно з вимогами до кваліфікаційної роботи, з дотриманням стандартів ДСТУ 8302:2015 щодо посилань, шрифтів, структури та графічного представлення матеріалів.

1.2.5 Техніко-економічні показники

Дана кваліфікаційна робота має некомерційний характер, однак під час його реалізації доцільно враховувати основні техніко-економічні показники, які визначають ефективність витрат ресурсів і трудових зусиль. До таких показників належать витрати часу на розробку, обсяг використаних інструментів, рівень автоматизації, а також потенційна користь розробки в навчальному або демонстраційному середовищі.

Основні показники:

- Тривалість розробки гри — 4 місяці (від постановки завдання до тестування);
- Використано безкоштовні інструменти: Unity (версія Personal), GIMP, Visual Studio Community, FL Studio Demo;
- Витрати на програмне забезпечення — 0 грн;
- Витрати на графіку, звук, ефекти — 0 грн (використано вільні ресурси або створено самостійно);
- Людські ресурси — одна особа (студент-розробник);

– Орієнтовна трудомісткість — 180 годин (враховуючи проєктування, розробку, тестування, написання документації).

– Ефективність:

– Розробка дозволила здобути практичні навички програмування в Unity, роботи з графікою, анімацією та UI;

– Гра може бути використана як демонстраційний зразок у навчальних закладах або як основа для подальших ігрових розробок;

– Завдяки відкритій архітектурі, роботу легко масштабувати або адаптувати до інших потреб.

Комп’ютерна техніка, що використовувалась для розробки:

Процесор: AMD Ryzen 5 3600, 6 ядер, 12 потоків, 3.6–4.2 GHz

Оперативна пам’ять: 16 GB DDR4

Відеокарта: NVIDIA GeForce GTX 1660 Super, 6 GB

Накопичувач: SSD 512 GB (NVMe)

Операційна система: Windows 10 x64

Монітор: Full HD 1920×1080, 60 Гц

1.2.6 Стадії та етапи розробки

Процес створення програмного забезпечення гри включав декілька послідовних етапів, кожен із яких мав свої цілі, задачі та результати. Розробка проходила за класичною структурою життєвого циклу програмного продукту з орієнтацією на практичну реалізацію функціоналу.

1. Постановка задачі:

- Визначення мети, теми та обсягу роботи;
- Формування вимог і технічного завдання;
- Ознайомлення з жанром top-down shooter та аналіз аналогів.

2. Проєктування:

- Розробка структури гри, ігрових об’єктів та їхніх взаємозв’язків;

- Вибір рушія (Unity) та інструментів (GIMP, FL Studio, Visual Studio);
- Проєктування інтерфейсу користувача.

3. Реалізація:

- Створення ігрових сцен, налаштування об’єктів;
- Програмування логіки гравця, ворогів, стрільби та апгрейдів;
- Додавання графіки, анімацій і звукового супроводу.

4. Тестування:

- Перевірка ігрового циклу на наявність помилок;
- Тестування інтерфейсу, апгрейдів, логіки ШІ;
- Оптимізація продуктивності.

5. Документування:

- Написання пояснювальної записки;
- Підготовка до захисту проєкту;
- Формування інструкцій для користувача.

6. Презентація:

- Демонстрація роботи гри;
- Пояснення архітектури та реалізованого функціоналу;
- Обговорення перспектив удосконалення.

1.2.7 Порядок контролю та прийому

Контроль та прийом програмного продукту здійснюється відповідно до вимог, зазначених у технічному завданні, та загальноприйнятих стандартів контролю якості програмного забезпечення.

Контроль та прийом гри розробником — це критично важливі етапи, що забезпечують якість, функціональність та відповідність готового продукту початковим вимогам. Цей процес охоплює як внутрішній контроль протягом розробки, так і фінальну перевірку перед релізом.

Нижче наведені ключові кроки цього процесу:

1. Внутрішній контроль якості (QA) протягом розробки

Цей процес відбувається постійно протягом усього циклу розробки, починаючи з перших прототипів.

Регулярне тестування функціоналу:

– Модульне тестування: Перевірка окремих компонентів та команд, щоб переконатися, що вони працюють правильно. Наприклад, чи коректно рухається гравець, чи спрацьовує постріл, чи віднімається здоров'я.

– Інтеграційне тестування: Перевірка взаємодії між різними модулями. Наприклад, як рух гравця взаємодіє з системою зіткнень, або як вороги реагують на постріли.

– Системне тестування: Тестування всієї гри як єдиної системи, щоб переконатися, що всі функції працюють разом без збоїв.

Тестування геймплею та балансу:

– Плейтестинг: Регулярні ігрові сесії для оцінки загального ігрового досвіду, складності, веселості та балансу. Це може бути як внутрішніми розробниками, так і запрошеними тестерами.

– Перевірка прогресії: Оцінка того, як гравці просуваються по рівнях, чи достатньо складність зростає, чи є достатня мотивація для продовження гри.

– Тестування продуктивності:

– Оцінка частоти кадрів (FPS): Перевірка стабільності FPS на різних конфігураціях обладнання, щоб гра працювала плавно.

– Використання пам'яті та ЦП: Моніторинг використання системних ресурсів для запобігання витоків пам'яті або надмірному навантаженню.

– Час завантаження: Оцінка часу, необхідного для завантаження гри та переходів між рівнями.

Тестування сумісності:

– Тестування на різних платформах/пристроях: Якщо гра призначена для різних платформ (ПК, мобільні пристрої, консолі), тестування на кожній з них з різними характеристиками.

– Тестування на різних дозволах екрану: Перевірка коректного відображення UI та ігрового світу при різних дозволах.

Тестування інтерфейсу користувача (UI) та користувацького досвіду (UX):

– Зручність навігації: Перевірка інтуїтивності меню, налаштувань, елементів управління.

– Читабельність тексту: Перевірка розміру шрифтів, контрастності, зрозумілості всіх текстових елементів.

– Доступність: Перевірка, чи гра є доступною для гравців з різними потребами (наприклад, колірний режим для дальтоніків).

2. Фінальний прийом гри

Цей етап відбувається, коли гра вважається повністю розробленою і готова до потенційного релізу або передачі замовнику.

Перевірка відповідності дизайн-документу (GDD):

– Порівняння реалізованих функцій, механік, візуального стилю та контенту з тим, що було описано в GDD.

– Виявлення будь-яких відхилень та обговорення їх з командою або замовником.

– Виявлення та виправлення критичних багів:

– Проведення ретельного тестування для виявлення всіх "блокуючих" багів (краші, зависання, неможливість пройти рівень) та пріоритетне їх виправлення.

– Виправлення "значних" багів (некоректна поведінка ворогів, помилки в розрахунках, візуальні глітчі, що псують досвід).

Перевірка локалізації (якщо є):

Якщо гра підтримує кілька мов, перевірка коректності перекладу, відсутність помилок у граматиці та відображенні тексту.

Фінальна оптимізація:

Останній етап оптимізації для досягнення найкращої продуктивності та стабільності на цільових платформах.

Формування "збірки" гри (Build):

Створення фінальної версії гри, яка буде готова до поширення. Це включає пакування всіх файлів, налаштування інсталятора (якщо потрібно) та підготовка до завантаження на платформи.

Підготовка супровідної документації:

Керівництво користувача, технічна документація для підтримки, опис відомих проблем (якщо такі є).

Приймальний акт:

У випадку роботи з замовником, підписання акту прийому-передачі, що підтверджує завершення робіт та відповідність гри встановленим вимогам.

У випадку власного проекту – внутрішнє затвердження командою, що гра відповідає очікуванням та готова до випуску.

Ефективний контроль та прийом гри вимагає систематичного підходу, чіткої комунікації та готовності до ітерацій. Це забезпечує, що кінцевий продукт буде не лише функціональним, але й приємним для гравців.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Постановка задачі на розробку програмного забезпечення

Мета даного розділу кваліфікаціної роботи полягає у детальному формулюванні завдання щодо створення навчального, демонстраційного та розважального програмного продукту — 2D-гри в жанрі top-down shooter, якій в процесі розробки було надано назву "Void Rush".

Основна увага зосереджена на повному циклі розробки ігрового проєкту — від постановки задачі та виконання роботи до реалізації і тестування. Робота має на меті продемонструвати навички використання сучасних інструментів програмування, ігрових рушіїв та методології розробки програмного забезпечення, що відповідає вимогам кваліфікаційної роботи бакалавра.

Головна задача полягає в реалізації повноцінної 2D-гри з базовим геймплейним циклом, що включає:

- управління ігровим персонажем;
- стрільбу по ворогах та бойову логіку;
- систему здоров’я, досвіду і прокачування;
- штучний інтелект ворогів;
- графічне оформлення та звуковий супровід;
- повноцінний користувацький інтерфейс.

Ключові підзадачі роботи:

- Реалізація переміщення гравця у двовимірному просторі за допомогою клавіш WASD або стрілок;
- Розробка системи стрільби у напрямку миші з обробкою колізій та візуальними ефектами;
- Впровадження логіки шкоди та смерті персонажів (гравця та ворогів);
- Розробка алгоритму ШІ ворогів, що здійснюють пошук та переслідування гравця;

- Побудова системи досвіду (XP), підвищення рівня (Level Up) та випадкового вибору покращень (апгрейдів);
- Відображення елементів інтерфейсу (панель здоров'я, досвіду, рівня, меню покращень, повідомлення тощо);
- Розробка головного меню, паузи, екрана завершення гри (Game Over);
- Інтеграція графіки, анімацій та звукового супроводу відповідно до обраної стилістики (космічний піксель-арт);
- Оптимізація ігрового процесу для стабільної роботи на персональних комп'ютерах середнього рівня.

Програмний продукт має відповідати таким вимогам:

- бути інтуїтивно зрозумілим для користувача;
- мати стабільну продуктивність (щонайменше 30 fps);
- запускатися у повноекранному режимі або у вікні;
- не мати критичних помилок під час запуску та гри;
- реагувати на основні події: старт, вихід, програш, апгрейд тощо.

Розробка реалізується в середовищі Unity з використанням мови програмування C#, що є однією з найбільш поширених комбінацій для створення 2D-ігор. Unity забезпечує зручну систему компонентів, потужну фізику та візуальний редактор, який дозволяє проектувати сцени швидко й ефективно. Використання C# дає змогу застосовувати об'єктно-орієнтовану модель програмування з чіткою структуризацією логіки. [2]

C# – це мова програмування, яка посідає визначне місце у світі розробки програмного забезпечення. Вона привертає увагу розробників з усього світу завдяки своїй потужності, універсальності та широкому спектру застосування. [19]

Розробка високопродуктивних додатків на C# вимагає ефективного та потужного середовища програмування.

C# має простий та інтуїтивно зрозумілий синтаксис, що робить його відносно простим в освоєнні для початківців-розробників. Він також має безліч вбудованих

функцій і бібліотек, які можуть допомогти розробникам прискорити процес розробки.

C# – це універсальна мова програмування, простий синтаксис, широка стандартна бібліотека та багата екосистема якої роблять її привабливим вибором для професіоналів і розробників-початківців. [19]

Особливу увагу приділено принципам об’єктно-орієнтованого програмування, використанню модульного підходу та чіткого розмежування функціональності між компонентами. Кожна ігрова сутність (гравець, ворог, снаряд, інтерфейс) має відповідний клас і набір властивостей, що забезпечує високу керованість та можливість масштабування проєкту. [2]

Ось кілька прикладів того, як C# може допомогти розробникам підвищити продуктивність їхніх додатків:

– C# є типобезпечною мовою, тобто компілятор перевіряє типи даних під час компіляції програми. Це допомагає уникнути помилок, які можуть призвести до зниження продуктивності.

– Об’єктно-орієнтоване програмування (ООП) дає змогу розробникам створювати складні додатки, які легко масштабуються та підтримувати. Це може підвищити продуктивність, особливо для додатків, які обробляють великий обсяг даних. [19]

– C# підтримує багатопоточність, тобто можна виконувати кілька завдань одночасно. Це підвищує продуктивність, особливо для додатків, які обробляють великий обсяг даних. Завдяки інтегруванню з різними хмарними технологіями, як-от Azure і AWS, розробники можуть підвищити продуктивність своїх додатків, переклавши частину навантаження на хмарну інфраструктуру.

Загалом, C# має безліч переваг, які роблять його чудовим вибором для розроблення високопродуктивних додатків. [19]

Окрім всього описаного вище, C# має потужну і сувору систему типів, яка допомагає запобігти безлічі помилок уже на етапі компіляції. Це гарантує створення більш надійних і продуктивних додатків. У системі типів C# також підтримуються

поліморфізм, успадкування та інтерфейси, що сприяє гнучкості та розширюваності коду. Розробка високопродуктивних додатків часто вимагає ефективної роботи з багатозадачністю та асинхронними операціями. C# надає багаті засоби для роботи з потоками виконання та асинхронними завданнями. Асинхронне ж програмування в C# дає змогу виконувати неблокувальні операції, як-от робота з мережею або файлами, без блокування основного потоку виконання. Це сприяє більш чуйному призначеному для користувача інтерфейсу і загальній продуктивності програми. [19]

Виходячи із вище описаного, гра "Void Rush", яка розробляється в даній кваліфікаційній роботі має стати ілюстрацією ефективної реалізації базових ігрових механік, зразком застосування сучасних технологій у навчальному процесі, а також платформою для майбутніх розширень. Крім того, вона може бути використана як навчальний шаблон для студентів молодших курсів або викладачів спеціальності, пов’язаної з розробкою програмного забезпечення та геймдизайном.

Таким чином, постановка задачі охоплює повний набір вимог до програмного забезпечення, що реалізується в межах кваліфікаційної роботи, і є підґрунтям для наступних етапів проєктування, програмування, тестування та документування продукту.

2.2 Опис та обґрунтування вибору структури та методу організації вхідних та вихідних даних

Одним із ключових елементів при проєктуванні ігрового програмного забезпечення є визначення логіки обміну даними між окремими елементами системи. У випадку даної кваліфікаційної роботи цей процес базується на можливостях рушія Unity і реалізується через компоненти, події та змінні класів. Застосовується структура об’єктно-орієнтованого підходу, де кожен логічний елемент гри представлений як окремий об’єкт із власними властивостями та поведінкою.

Вхідні дані:

– Клавіатурний ввід (рух гравця: клавіші WASD або стрілки);
– Маніпуляції миші (напрямок прицілу, постріли);
– Початкові параметри об’єктів, задані в інспекторі Unity (здоров’я, швидкість, тип ворога);
– Значення з конфігураційних скриптів або ScriptableObject (наприклад, параметри апгрейдів).

Вихідні дані:

– Відображення змін на екрані: пересування, анімації, ефекти пострілів;
– Зміни у UI: оновлення панелі HP, XP, меню покращень;
– Логічні події: смерть гравця, смерть ворога, перемога, поразка;
– Аудіо-вивід: звуки стрільби, шкоди, отримання XP.

Взаємодія реалізується через такі механізми:

– Unity Events (події при зіткненні, прицілюванні, взаємодії з UI);
– OnTriggerEnter2D/OnCollisionEnter2D для обробки фізичних подій;
– Публічні змінні, які передаються через інспектор або знаходяться скриптами;
– Делегати або методи виклику в інших компонентах (наприклад, GameManager, LevelUpSystem).

Вхідні та вихідні дані в грі не мають структури бази даних, проте логічно згруповані в окремі класи та об’єкти. Так, клас PlayerHealth відповідає за облік поточного та максимального здоров’я, PlayerShooting — за швидкість атаки, кулі та логіку стрільби, а PlayerXP — за рівень, досвід та прогрес прокачування.

Перевагою такої структури є:

– Простота розробки: всі об’єкти незалежні і легко модифікуються;
– Чітка взаємодія між компонентами через публічні посилання;
– Легкість у відлагодженні: інспектор Unity дозволяє бачити значення змінних у реальному часі;

– Розширюваність: будь-який об’єкт може бути вдосконалений або замінений без порушення загальної архітектури гри.

Вибір Unity як платформи для організації даних обумовлений його підтримкою великої кількості готових інструментів: Rigidbody2D, Collider2D, Canvas, AudioSource, Animation тощо.

Unity дозволяє ефективно поєднувати всі елементи, створюючи зручну екосистему для керування даними на всіх рівнях: від вводу користувача до обробки подій та виводу результатів на екран.

Таким чином, структура організації даних у "Void Rush" є простою, надійною, добре відлагодженою в межах середовища Unity та повністю відповідає задачам, поставленим у межах розробки кваліфікаційного проєкту.

2.3 Розробка алгоритму

Розробка алгоритмів є фундаментальною складовою створення будь-якого програмного забезпечення, а особливо — відеоігор, де взаємодія користувача з інтерфейсом і системами гри має відбуватися в реальному часі. У грі "Void Rush" алгоритми покривають широкий спектр функцій: від базового переміщення гравця до реалізації штучного інтелекту ворогів і динамічного формування меню покращень. У цьому підрозділі детально розглянемо як зовнішнє, так і внутрішнє проектування гри.

2.3.1 Зовнішнє проєктування програми

Зовнішнє проектування зосереджене на розробці користувацького інтерфейсу, зручності взаємодії з програмою та візуальних елементів. Метою є створення інтуїтивно зрозумілої структури управління, яка дозволяє гравцеві швидко адаптуватися до ігрового процесу та взаємодіяти з усіма необхідними елементами.

Ключові елементи зовнішнього проектування:

– Головне меню гри: дозволяє розпочати гру або вийти з програми. Всі кнопки оформлені у єдиному стилі з грою.

– Ігровий інтерфейс (HUD): містить індикатори здоров'я (HP), досвіду (XP), поточного рівня, та відображає інші важливі параметри стану гравця. Інтерфейс оновлюється в режимі реального часу.

– Меню покращень (Level Up): активується при підвищенні рівня. Гравцю пропонується вибір із трьох випадкових апгрейдів. Гра при цьому ставиться на паузу.

– Екран поразки (Game Over): з'являється при нульовому рівні здоров'я. Дозволяє гравцеві перезапустити гру або повернутись до головного меню.

Взаємодія: здійснюється через стандартні засоби вводу — клавіатура (рух, пауза), миша (орієнтація, стрільба, вибір меню).

На головному екрані користувач має змогу розпочати гру або вийти з неї. Меню виконано в мінімалістичному стилі, з використанням космічного фону. Інтерфейс інтуїтивно зрозумілий, що полегшує перший контакт користувача з грою. (див. рис. 2.1)

Рисунок 2.1 – Головне меню гри Void Rush

На початковій сцені відображається персонаж гравця в центрі екрану. Також видно панель здоров’я (HP), досвіду (XP) та фонову графіку. Сцена створена у піксельному стилі з використанням простих візуальних елементів. (див. рис. 2.2)

Рисунок 2.2 – Початкова сцена з гравцем

Після втрати всього здоров’я гравця з’являється екран поразки. Він містить повідомлення про кінець гри та кнопку для повернення до головного меню. Реалізовано логіку завершення гри та перезапуску сцени. (див. рис. 2.3)

Рисунок 2.3 – Екран поразки

2.3.2 Проєктування логіки програми

Логіка реалізації внутрішніх взаємодій між об’єктами гри сформована на основі принципів об’єктно-орієнтованого програмування (ООП) та компонентної моделі Unity. Кожен елемент гри представлений окремим скриптом або набором компонентів, які керують поведінкою відповідного об’єкта.

Основні логічні блоки:

Гравець:

– PlayerMovement.cs — переміщення у двовимірному просторі;

– PlayerShooting.cs — стрільба з урахуванням затримки (fireRate);

– PlayerHealth.cs — облік шкоди, смерть;

– PlayerXP.cs — облік досвіду, підвищення рівня;

– PlayerStats.cs — зберігання характеристик: швидкість, здоров’я, інтервал стрільби.

Вороги:

– EnemyFollow.cs — слідування за гравцем;

– EnemyAttack.cs — нанесення шкоди при зіткненні;

– TargetHealth.cs — зменшення HP, смерть.

Набої:

– Bullet.cs — переміщення, виявлення зіткнення з ворогом, знищення об’єкта, передача досвіду.

Інтерфейс:

– HealthBar, XPBar, LevelText, LevelUpPanel — синхронізація UI зі станом гравця;

– GameOverPanel — викликається GameManager’ом при загибелі гравця.

Менеджери:

– GameManager.cs — координує основні стани гри (початок, програш, перезапуск);

– LevelUpSystem.cs — управляє логікою покращень (рандомізація, застосування апгрейдів).

Під час гри гравець стріляє у ворогів із двох стволів. На рисунку зображено кадр, коли снаряди спрямовані у ворога. Видно також анімацію пострілу та рух ворога до гравця. (див. рис. 2.4)

Рисунок 2.4 – Момент стрільби у ворога

Після досягнення нового рівня з’являється панель вибору покращень. Гравець може обрати одну з трьох випадкових характеристик: збільшення здоров’я, швидкості або скорострільності. Це меню реалізовано з паузою гри та іконками апгрейдів. (див. рис. 2.5)

Рисунок 2.5 – Меню покращення характеристик

Алгоритм взаємодії основних елементів

Гравець → Стрільба → Ворог → Смерть → Досвід → Рівень → Покращення

1. Гравець натискає кнопку Fire1 — створюється куля;
2. Куля досягає ворога — викликається TakeDamage();
3. Ворог вмирає — через Die() нараховується XP гравцеві;
4. Коли XP перевищує xpToNextLevel — викликається LevelUp();
5. LevelUpSystem активує меню вибору покращення;
6. Гравець обирає один із трьох варіантів (HP, Speed, FireRate) — ефект застосовується до PlayerStats.

Розширення алгоритму

Гра побудована за принципом нескінченного виживання, де з кожною хвилею ворогів гравець отримує більше XP, що стимулює подальший розвиток і викликає дедалі більше ускладнень. Це досягається за допомогою:

- поступового збільшення швидкості спавну ворогів;
- додавання нових типів ворогів;

покращень, які дають змогу гравцю адаптуватися до нових викликів.

2.4 Визначення інформаційних зв’язків

У процесі розробки 2D-гри жанру Top-Down Shooter на ігровому рушії Unity одним із найважливіших завдань було формування чіткої системи інформаційних зв’язків між усіма компонентами програми.

Оскільки розробка здійснювалася в середовищі Unity, основна модель взаємодії побудована на основі публічних змінних, інтерфейсів, компонентної структури та подій.

Усі об’єкти гри — гравець, вороги, снаряди, елементи UI — тісно взаємодіють один з одним через чітко визначені канали передачі інформації.

Основні типи інформаційних зв’язків:

1. Гравець ↔ Вороги:

– Фізична взаємодія через колізії (OnCollisionEnter2D, OnTriggerEnter2D);

– Метод TakeDamage() викликається ворогом або снарядом при контакті з гравцем;

– Система EnemyAttack зменшує HP гравця під час зіткнення;

– Система Bullet або PlayerShooting взаємодіє з ворогом, викликаючи зменшення його здоров'я.

2. Гравець ↔ Інтерфейс (UI):

– Скрипти PlayerHealth та PlayerXP викликають методи оновлення HealthBar, XPBar, LevelText, синхронізуючи поточні значення зі шкалами та написами;

– При зміні здоров'я гравця автоматично оновлюється його індикатор здоров'я в UI;

– При здобутті досвіду та підвищенні рівня змінюється індикатор XP та рівня гравця;

– Меню Level Up викликається при досягненні нового рівня гравцем.

3. Гравець ↔ Менеджери (GameManager, LevelUpSystem):

– GameManager координує глобальні стани: запуск, поразку, перезапуск, зупинку гри;

– Меню Game Over активується при смерті гравця, зупиняючи таймер та вимикаючи елементи управління;

– Скрипт LevelUpSystem генерує випадкові покращення, відображає меню, чекає на вибір користувача і застосовує ефект.

4. Набої ↔ Вороги:

– Bullet при зіткненні з об'єктом, що має тег Enemy, викликає шкоду через метод TakeDamage();

– При знищенні ворога PlayerXP отримує сигнал про додавання досвіду;

– Можливе додавання ефектів (анімації вибуху, звук).

5. Меню ↔ Користувач:

– Unity UI викликає методи GameManager через кнопки (StartGame, ExitGame);

– Меню LevelUp – тимчасово зупиняє гру (Time.timeScale = 0) до вибору апгрейду;

– Game Over – дозволяє повертатись у головне меню або повторно запускати гру.

6. Скрипти ↔ Конфігураційні дані:

– Більшість параметрів (здоров'я, швидкість, шкода) задаються через інспектор Unity і передаються в скрипти через публічні поля;

– Передбачено можливість використання ScriptableObject для збереження типів покращень або глобальних налаштувань.

Характеристика інформаційних зв'язків:

– Односторонні зв'язки: приклад — снаряд вражає ворога, але не навпаки;

– Двосторонні зв'язки: взаємодія гравця з ворогами через зіткнення або атаки;

– Подвійні зв'язки: реалізовані через виклики методів між об'єктами або UnityEvent, що дозволяє масштабувати логіку без жорсткого зв'язування класів.

Графічно ці зв'язки можна відобразити у вигляді UML-діаграми, де кожен клас гри є окремим блоком із входами та виходами. Наприклад:

– Player ↔ Enemy: Damage();

– Bullet → Enemy: OnTrigger → TakeDamage → Die → XP++;

– PlayerXP → UI → LevelText + XPBar;

– PlayerHealth → HealthBar;

– GameManager → GameOverPanel.

Таким чином, проєкт реалізовано із дотриманням принципів слабкого зв'язування, високої модульності та прозорості інформаційного обміну між об'єктами гри, що є критично важливим для подальшої підтримки, рефакторингу та масштабування коду.

2.5 Написання текстів програм

Основна частина логіки 2D-гри жанру Top-Down Shooter на ігровому рушії Unity" (Void Rush") реалізована за допомогою мови програмування C# у середовищі розробки Visual Studio 2022 із повною інтеграцією в рушій Unity.

Програмна архітектура побудована за модульним принципом із дотриманням концепції розділення обов'язків, що дозволяє спростити масштабування, налагодження та повторне використання коду.

Всі компоненти гри поділені на окремі скрипти, кожен з яких виконує чітко визначену функцію.

Наприклад:

- PlayerMovement.cs — відповідає за обробку клавіш управління (W, A, S, D) та фізичне переміщення персонажа на сцені за допомогою Rigidbody2D.
- PlayerShooting.cs — керує орієнтацією гравця на мишу, створенням снарядів при натисканні кнопки стрільби, контролем інтервалу між пострілами.
- PlayerHealth.cs — реалізує обробку пошкоджень гравцем, оновлення індикатора здоров'я та зникнення персонажа після смерті.
- EnemyFollow.cs — відповідає за автоматичне переслідування гравця за допомогою пошуку напрямку та руху до нього.
- EnemyAttack.cs — реалізує атаку гравця при зіткненні з ворогом.
- Bullet.cs — управляє рухом куль, їхнім знищенням при зіткненні та нанесенням пошкоджень ворогам.
- PlayerXP.cs — накопичує досвід, підвищує рівень гравця та викликає LevelUp меню при досягненні певного порогу XP.
- LevelUpSystem.cs — відповідає за відображення меню покращень, вибір та застосування одного з трьох рандомних варіантів апгрейду.
- PlayerStats.cs — зберігає змінні швидкості, здоров'я, частоти стрільби та дозволяє їх змінювати через інші скрипти.

– GameManager.cs — головний контролер гри, який слідкує за статусом (гра/пауза/поразка), перемикає сцени та відображає GameOver.

Кожен скрипт містить:

– Заголовок із коротким описом.

– Коментарі до основних методів та змінних.

– Винесені публічні змінні для налаштування параметрів через інспектор Unity.

– Обробку винятків та відлагодження через Debug.Log().

Крім коду, до складу програми входять:

– Префаби гравця, ворогів, куль, UI панелей.

– Анімації для зброї, ворогів, ефекти пострілу.

– Звуки пострілу, музика, ефекти поразки.

– Спрайти для гравця, ворогів, кнопок, фону та іконок апгрейдів.

Програмна реалізація базується на найкращих практиках: код, чітка ієрархія об’єктів, використання Time.deltaTime для коректного подання та незалежності від FPS. Завдяки модульному підходу код легко тестувати й адаптувати під нові завдання без втручання в основну логіку гри.

Усі скрипти супроводжуються коментарями, що пояснюють призначення змінних, методів та умов. Об’єкти гри організовано у вигляді префабів з прив’язаними скриптами, що спрощує повторне використання та зміну конфігурації в редакторі Unity.

Під час написання коду дотримувалися основні принципи чистого програмування: розділення відповідальності, мінімізація залежностей, узгодженість імен змінних, використання інкапсуляції та перевірка на null.

Для зручності ознайомлення з усією програмною реалізацією, весь перелік використаних класів і скриптів з повним текстом коду представлено в Додатку А.

У додатку можна знайти повні версії основних скриптів, таких як PlayerMovement.cs, EnemyFollow.cs, PlayerHealth.cs, PlayerXP.cs, LevelUpSystem.cs,

GameManager.cs та інші. Також додаток містить приклади застосування логіки взаємодії між об’єктами, реалізації інтерфейсу, налаштування бойової системи та інших важливих частин гри.

2.6 Тестування та налагодження програм

Етап тестування та налагодження програмного забезпечення «Void Rush» був одним із найважливіших під час розробки гри. Його мета — забезпечення надійного функціонування всіх компонентів гри, виявлення помилок, багів, логічних або інтерфейсних недоліків, а також забезпечення стабільної та прогнозованої поведінки ігрових об’єктів.

Методи тестування

У рамках розробки були використані такі типи тестування:

1. Функціональне тестування — перевірка реалізації основного функціоналу відповідно до технічного завдання. Було протестовано:

- Переміщення гравця за допомогою клавіш WASD;
- Робота системи стрільби та знищення ворогів;
- Взаємодія гравця з ворогами (зіткнення, отримання пошкоджень);
- Система підвищення рівня (XP, Level Up);
- Робота головного меню, паузи та завершення гри;
- Активність UI-елементів (панель HP, панель XP, меню покращень).

2. Модульне тестування — перевірка окремих компонентів у відриві від системи:

- PlayerHealth.cs — тестування коректності віднімання HP та виклику методу Die();
- Bullet.cs — перевірка руху, часу життя, взаємодії з колайдерами;
- EnemyFollow.cs — правильність логіки слідування за гравцем;
- LevelUpSystem.cs — активація меню при досягненні рівня, коректне застосування апгрейду.

3. Інтеграційне тестування — перевірка взаємодії між різними модулями:

- Зв’язок PlayerXP з LevelUpSystem — активація меню та застосування покращень;
- Взаємодія Bullet з EnemyHealth і PlayerXP — отримання досвіду після знищення ворога;
- Спрацьовування GameManager при загибелі гравця.

4. Регресійне тестування — виконувалося після додавання нових механік:

- Перевірка, що нові скрипти не порушують вже реалізований функціонал;
- Повторне проходження повного ігрового циклу після кожного оновлення.

5. UI-тестування — оцінка відображення графічних елементів інтерфейсу:

- Відповідність індикаторів HP і XP реальному стану гравця;
- Активність кнопок у меню Level Up;
- Відображення рівня, кількості XP, анімацій у UI.
- Процес налагодження

Після виявлення помилок застосовувалися такі підходи до налагодження:

- Використання Debug.Log() для відстеження змін змінних у реальному часі;
- Використання вкладки Console у редакторі Unity для діагностики помилок, включаючи виключення (NullReferenceException, MissingComponentException, IndexOutOfRangeException);
- Налагодження префабів, компонентів та їх посилань у вікні Inspector;
- Вимкнення або ізоляція проблемних компонентів для локалізації багів.

Приклади помилок і шляхів вирішення:

- Проблема: Гравець отримував шкоду від власного снаряду. Рішення: Зміна шару об’єктів, що стріляють, та шарів взаємодії у Physics2D settings.

- Проблема: Меню Level Up не з’являлося після підняття рівня. Рішення: Виявлено, що виклик TriggerLevelUp() здійснювався з неактивного об'єкта. Вирішено через попередню активацію об’єкта на сцені або перенесення логіки виклику в інший компонент.
- Проблема: Некоректне оновлення UI (наприклад, HP-бар залишався незмінним). Рішення: Додано метод UpdateHealthUI() з викликом після кожної зміни значення здоров’я.
- Проблема: Звук пострілу повторювався надто часто або не відтворювався. Рішення: Додано перевірку доступності компоненту AudioSource перед викликом методу Play().
- Проблема: Гравець міг рухатись під час активного меню Level Up. Рішення: Під час відкриття меню встановлювалося Time.timeScale = 0, а також призупинення читання вводу користувача.

Усі ці дії допомогли виявити та усунути критичні помилки, що могли б завадити демонстрації гри.

Після налагодження всі компоненти працювали стабільно. Гра не видавала фатальних помилок під час тестових запусків, а також коректно реагувала на взаємодії між об’єктами, дії гравця та зміну стану гри.

Висновки за результатами тестування

Завдяки поетапному підходу до тестування, а також застосуванню багатьох типів перевірок, вдалося досягти стабільної, логічно завершеної та ігрової версії продукту.

Усі основні механіки, передбачені технічним завданням, були реалізовані, протестовані та успішно інтегровані в фінальну версію гри. Гра готова до демонстрації в рамках захисту кваліфікаційної роботи бакалавра.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з інсталяції програмного забезпечення

Для запуску 2D-гри жанру Top-Down Shooter на ігровому рушії Unity ("Void Rush") на персональному комп’ютері необхідно виконати наступні кроки:

1. Системні вимоги:

— Операційна система: Windows 10 або новіша;

— Процесор: Intel Core i3 або аналогічний AMD;

— Оперативна пам’ять: від 4 ГБ;

— Відеокарта: з підтримкою DirectX 10 або вище;

— Вільне місце на диску: не менше 500 МБ.

2. Інсталяція:

— Завантажити архів із грою з наданого джерела;

— Розпакувати архів у бажану директорію;

— Запустити файл VoidRush.exe;

— У випадку появи повідомлення системи безпеки — підтвердити запуск.

3. Додаткові рекомендації:

— Установити останню версію .NET Framework або Visual C++ Redistributable, якщо гра не запускається;

— Грати у віконному або повноекранному режимі (налаштовується у запускному вікні);

— Не змінювати структуру папок усередині архіву. (див. рис. 3.1)

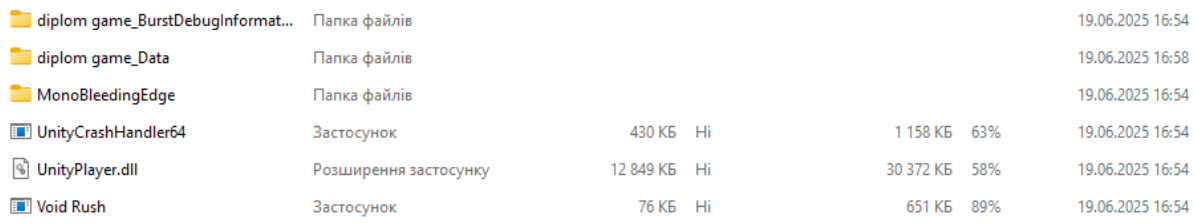


Рисунок 3.1 – Структуру папок усередині архіву

3.2 Інструкція з використання тестових наборів

У межах даної кваліфікаційної роботи бакалавра проводилося тестування основних функціональних компонентів гри з метою виявлення й усунення помилок. Для цього були сформовані умовні тестові сценарії, які не є автоматизованими наборами, а скоріше описами дій, що дозволяють перевірити роботу ключових функцій гри.

1. Тест керування гравцем:

— Очікувана поведінка: гравець плавно переміщується у всіх 4 напрямках;

— Вхідні дії: натискання клавіш W, A, S, D;

— Успішний результат: позиція змінюється, спрайт обертається відповідно до миші.

— Тест бойової системи:

— Вхідні дії: натискання кнопки миші для стрільби;

— Очікувана поведінка: випускаються кулі, які знищують мішень;

— Успішний результат: мішень зникає, кулі — самознищуються.

3. Тест системи досвіду та Level Up:

— Вхідні дії: влучити у ворога кілька разів;

— Очікувана поведінка: ворог зникає, зростає XP;

— Після досягнення порогу рівня з’являється меню покращень;

— Успішний результат: після вибору покращення гра продовжується.

4. Тест інтерфейсу (UI):

— Вхідні дії: пошкодження гравця, отримання XP;

— Очікувана поведінка: змінюється HP-бар, XP-бар, активується Level Up Panel;

— Успішний результат: інтерфейс оновлюється коректно.

5. Тест логіки Game Over:

— Вхідні дії: допустити повне знищення гравця;

— Очікувана поведінка: зупинка гри, поява меню програшу;

— Успішний результат: з'являється кнопка повернення до меню.

Рекомендації:

— Тестування проводити в редакторі Unity або в зібраному .exe-файлі;

— Кожен тест повторювати кілька разів для перевірки стабільності;

— Вести журнал виявлених багів для подальшого усунення.

3.3 Інструкція з експлуатації програмного комплексу

Цей підпункт містить короткий опис порядку використання гри "Void Rush" кінцевим користувачем. Він охоплює запуск програми, управління, основні дії в грі та можливі сценарії користування.

Запуск гри:

1. Відкрити папку з розпакованим архівом гри (див. рис. 3.2);

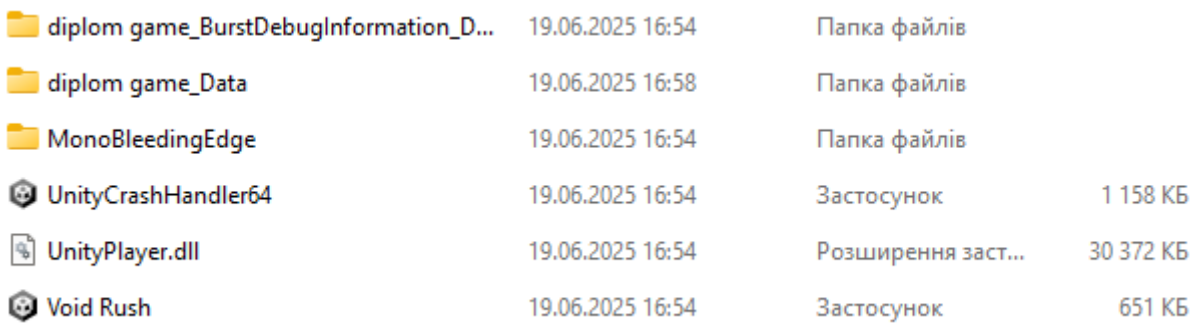


Рисунок 3.2 – Розпакований архів гри

2. Двічі клацнути по VoidRush.exe (див. рис. 3.3);

Рисунок 3.3 – VoidRush.exe

3. У стартовому меню обрати "Start Game" або "Exit"; (див. рис. 3.4)

Рисунок 3.4 – Головне меню гри Void Rush

4. Після вибору "Start Game" гравець переходить у сам ігровий процес. (див. рис. 3.5)

Рисунок 3.5 – Вікно відображення ігрового проесу

Управління:

— WASD — переміщення гравця вгору, вниз, вліво, вправо;

— Мишка — прицілювання (гравець повертається у напрямку курсора);

— ЛКМ (ліва кнопка миші) — стрільба;

— Esc — вихід до головного меню (опційно);

— Пробіл — підтвердження вибору в меню покращень.

Основні дії:

— Убивати ворогів, стріляючи у них (вороги самі наближаються);

— Збирати досвід (XP), який надається за знищення ворогів;

— При досягненні нового рівня — обирати одне з покращень;

— Виживати якнайдовше (гра без обмеження за часом);

— Якщо здоров’я гравця зменшиться до нуля — гра закінчується.

Інтерфейс:

— HP бар — у верхньому лівому куті, показує здоров’я гравця;

— XP бар — нижче HP, заповнюється при здобутті досвіду;

— Level Up меню — з’являється автоматично після підняття рівня;

— Game Over екран — з’являється після загибелі гравця.

Завершення роботи:

— Гру можна завершити через головне меню або закриттям вікна.

Рекомендації:

— Не переривати роботу гри примусово (Alt+F4), щоб уникнути втрати прогресу;

— При виникненні проблем перезапустити гру або звернутись до супровідної документації.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

4.1 Вибір і обґрунтування основних техніко-економічних показників

При розробці програмного забезпечення надзвичайно важливим є оцінка ефективності використаних ресурсів, як матеріальних, так і трудових. Економічний розділ спрямований на обґрунтування доцільності створення проєкту "Void Rush" з економічної точки зору, визначення витрат, необхідних для його реалізації, а також потенційної вигоди, яку може принести даний продукт у випадку комерційного використання або подальшого розвитку.

До основних техніко-економічних показників відносять: витрати на електроенергію, амортизацію обладнання, оплату праці розробника, витрати на оренду приміщення, допоміжні витрати та прогнозований прибуток. Всі ці показники дозволяють оцінити вартість створення програмного продукту та визначити його рентабельність у випадку подальшого впровадження.

Ці показники також дозволяють порівняти витрати на створення гри з потенційним прибутком від її реалізації. Такий аналіз важливий для прийняття рішень щодо комерціалізації або масштабування проєкту. В економічному розділі враховано як прямі, так і непрямі витрати, що забезпечує повноцінну картину ресурсного забезпечення.

4.2 Витрати на електроенергію

Розробка ігор потребує постійного використання комп’ютерної техніки, яка споживає електроенергію. Тому витрати на електроенергію є важливою частиною загальних витрат. Для забезпечення процесу розробки гри були використані персональний комп’ютер, монітор та освітлення. Витрати на електроенергію розраховуються за формулою:

Св = Р × Т × Це / 1000 ,

де: Р — потужність обладнання, Вт; Т — час використання, год; Це — тариф на електроенергію, грн/кВт·год.

Таблиця 4.1 – Витрати на електроенергію під час розробки гри

Обладнання	Потужність (Вт)	Час роботи (год)	Тариф (грн/кВт·год)	Витрати (грн)
Комп’ютер	450	180	2.64	213.84
Монітор	60	180	2.64	28.51
Освітлення	40	180	2.64	19.00
Разом	—	—	—	261.35

4.3 Амортизація обладнання

Амортизація обладнання — це процес поступового зменшення вартості технічних засобів у зв’язку з їх експлуатацією. Для розрахунку амортизації використовують формулу:

А = (Цоб × Н) / 100 × Т / 12 ,

де: Цоб — вартість обладнання, грн; Н — річна норма амортизації, %; Т — час використання, міс.

Цей показник враховує зношення обладнання протягом часу використання. Навіть у разі розробки некомерційного продукту цей показник дозволяє оцінити втрати вартості основних засобів.

(25 000 грн, 25%) → А = (25000 × 25 / 100) × 4 / 12 = 2083.33 грн

(6 000 грн, 25%) → А = (6000 × 25 / 100) × 4 / 12 = 500.00 грн

Разом: 2583.33 грн

4.4 Витрати на оплату праці

Витрати на оплату праці визначаються згідно з нормативними даними середньої заробітної плати програміста. Припустимо, студент виконував проєкт самостійно протягом 180 годин. Середня погодинна ставка — 100 грн/год.

$Зп = Т \times С$,

де Т — тривалість у годинах, С — ставка за годину.

$Зп = 180 \times 100 = 18000$ грн

$ЄСВ = Зп \times 22\% = 18000 \times 0.22 = 3960$ грн

Воп. = 18000 + 3960 = 21960грн.

Ці витрати є найбільшими серед усіх, оскільки людський ресурс є ключовим у створенні програмного забезпечення. Погодинна ставка обрана з урахуванням ринкових умов на позиції джуніор-розробника.

4.5 Витрати на оренду приміщення

Оренда розраховується виходячи з середньої вартості 1 м² офісного приміщення — 300 грн/м² на місяць. Площа — 5 м², термін — 4 міс.

$Опр. = Пл \times Цм^2 \times Т$,

$Опр. = 5 \times 300 \times 4 = 6000$ грн.

Цей показник включає витрати на приміщення, яке може бути домашнім робочим місцем або умовно розрахованим офісом. Такий підхід забезпечує реалістичну оцінку витрат.

4.6 Інші витрати

До інших витрат належать витрати на інтернет, канцелярію, резервне копіювання, хмарні сервіси, непередбачені витрати.

Інтернет (4 міс × 250 грн) = 1000 грн;

Канцелярія, флешка, блокнот = 500 грн.

Разом: 1500 грн

Додаткові витрати необхідно враховувати навіть у невеликих проєктах. Це дозволяє уникнути недооцінки загальної вартості проєкту.

4.7 Загальні витрати на проєкт

Св = Се + Са + Сзп + Со + Сдоп

Св = 261.35 + 2583.33 + 21960 + 6000 + 1500 =32304,68 грн.

Загальні витрати включають усі попередні статті й є основою для розрахунку прибутку та рентабельності. Такий комплексний підхід гарантує точність економічного аналізу.

4.8 Складання кошторису витрат та визначення собівартості НДР

Кошторис витрат являє собою зведений план усіх витрат підприємства на майбутній період виробничо-фінансової діяльності. Результати проведених вище розрахунків зведемо у таблиці 4.2

Таблиця 4.2 Кошторис витрат НДР

Зміст витрат	Сума, грн.	В % до загальної суми
Витрати на оплату праці (основну і додаткову заробітну плату)	18 000,00	55,71 %
Відрахування на соціальні заходи	3 960,00	12,26 %
Матеріальні витрати	7 500,00	23,22 %
Витрати на електроенергію	261,35	0,81 %
Транспортні витрати	1 615,23	5,00 %
Амортизаційні відрахування	2 583,33	7,99 %
Накладні витрати	2 196,00	6,80 %
Собівартість	36 115,91	100 %

Собівартість ($С_в$) НДР розрахуємо за формулою 4.1:

$$С_В=В_{о.п.}+В_{с.з.}+З_{м.в.}+З_в+Т_в+А+Н_в \quad (4.1)$$

СВ=18000+3960+7500+261,35+1615,23+2583,33+2196,00=36 115,91 грн

Отже, собівартість дорівнює $С_В$ = 36 115,91 грн

4.9 Розрахунок ціни НДР

Ціну НДР можна визначити за формулою 4.2: де Св – собівартість виконання НДР, $Р_{рен.}$ – рівень рентабельності; ПДВ – ставка податку на додану вартість, 20%.

$$Ц = Св \cdot (1+Ррен) \cdot (1+ПДВ), \quad (4.2)$$

де:

- Ррен. — рівень рентабельності = 30% = 0,3;
- ПДВ — податок на додану вартість = 20% = 0,2.

Ц=36115,91·1,3·1,2=56298,55 грн

4.10 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Прибуток розраховується за формулою 4.3:

$$П = Ц - С_в \qquad (4.3)$$

П = 56298,55−36115,91=20182,64 грн

Економічна ефективність ($Е_р$) полягає у відношенні результату виробництва до затрачених ресурсів і розраховується за формулою 4.4, де П – прибуток; $С_в$ – собівартість.

$$Е_р = П / С_в, \qquad (4.4)$$

$Е_р$ = 20182,64 / 36115,91 = 0,56

Поряд із економічною ефективністю розраховують (формула 4.5) термін окупності капітальних вкладень $Т_р$:

$$T_р = 1 / E_р \quad (4.5)$$

Допустимим вважається термін окупності до 5 років. В даному випадку:

$T_р=1/0,56=1,79$

Всі дані внесемо в зведену таблицю 4.3 техніко-економічних показників.

Таблиця 4.3 Техніко-економічні показники проєкту Розробка 2D-гри жанру Top-Down Shooter на ігровому рушії Unity

№п/п	Показник	Одиниця виміру	Значення
1.	Собівартість	грн.	36 115,91
2.	Плановий прибуток	грн.	20 182,64
3.	Ціна	грн.	56 298,55
4.	Економічна ефективність	-	0,56
5.	Термін окупності	рік	1,79

Висновки до економічного розділу

Ефективність виробництва — це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічний аналіз показав, що проєкт «Void Rush» має потенціал до комерційного використання та рентабельність понад 50%. Завдяки використанню безкоштовних інструментів розробки (Unity, GIMP, FL Studio DEMO, Visual Studio Community), проєкт не потребував ліцензійних витрат. Усі витрати були зведені до необхідного мінімуму, зосереджені на основних статтях: оплата праці, амортизація та оренда. Прогнозована вигода свідчить про економічну доцільність створення подібних продуктів у рамках кваліфікаційної роботи бакалавра.

5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

5.1 Розрахунок системи штучного освітлення для приміщення, де здійснюється розробка 2D-гри жанру Top-Down Shooter на ігровому рушії Unity

Організація якісного освітлення в робочому приміщенні відіграє надзвичайно важливу роль у забезпеченні безпечних, зручних та ефективних умов праці. Освітлення — це не лише засіб забезпечення видимості, але й один із чинників, що безпосередньо впливає на працездатність, стан здоров’я та загальну продуктивність працівника. Особливо це актуально для розробників програмного забезпечення, які більшість часу проводять перед монітором комп’ютера.

У контексті розробки 2D-гри жанру Top-Down Shooter на рушії Unity, комфортне освітлення є невід’ємною складовою продуктивного робочого процесу. Недостатній рівень освітленості, мерехтіння ламп, надмірна яскравість або ж неправильно розміщене світло можуть призводити до швидкої втомлюваності очей, головного болю, зниження концентрації уваги та, як наслідок, до погіршення якості програмного коду і зниження темпів розробки гри.

На основі чинного стандарту ДСТУ EN 12464-1:2022 "Світло та освітлення. Освітлення робочих місць у приміщеннях", встановлено, що для офісних приміщень, де здійснюється робота з комп’ютерною технікою, рекомендований рівень освітленості має становити не менше ніж 500 лк (люкс).

Приймемо такі вихідні параметри для розрахунку кількості необхідних світильників:

- Площа приміщення: 4 м × 5 м = 20 м²;
- Висота підвісу світильників: 2,5 м;

– Коефіцієнт запасу: 1,5 (з урахуванням пилу, зниження світловіддачі з часом);

– Коефіцієнт нерівномірності освітлення: 1,1;

– Світловий потік однієї лампи: 1500 лм (люмен);

– Коефіцієнт використання світлового потоку (η): 0,6.

Формула для розрахунку кількості світильників: $N = (E \times S \times Kз) / (Ф \times \eta \times Z)$, де:

– E — нормована освітленість (лк);

– S — площа приміщення (м²);

– Kз — коефіцієнт запасу;

– Ф — світловий потік одного світильника (лм);

– η — коефіцієнт використання світлового потоку;

– Z — коефіцієнт нерівномірності освітлення.

Підставимо значення: $N = (500 \times 20 \times 1,5) / (1500 \times 0,6 \times 1,1) \approx 15,15$.

Отже, для досягнення нормативного рівня освітлення вказаного приміщення необхідно встановити 15–16 світлодіодних світильників потужністю 1500 лм кожен. Це дозволить створити рівномірне, м’яке освітлення без відблисків та перевантаження зорової системи користувача.

Світильники повинні бути рівномірно розташовані по всій площі стелі приміщення. Наприклад, можна реалізувати схему 4 × 4 або 3 × 5 світильників з приблизно однаковими відстанями між ними. Такий підхід забезпечує не лише відповідність санітарно-гігієнічним нормам, але й підвищує естетичну привабливість та комфорт робочого середовища (див. рис. 5.1).

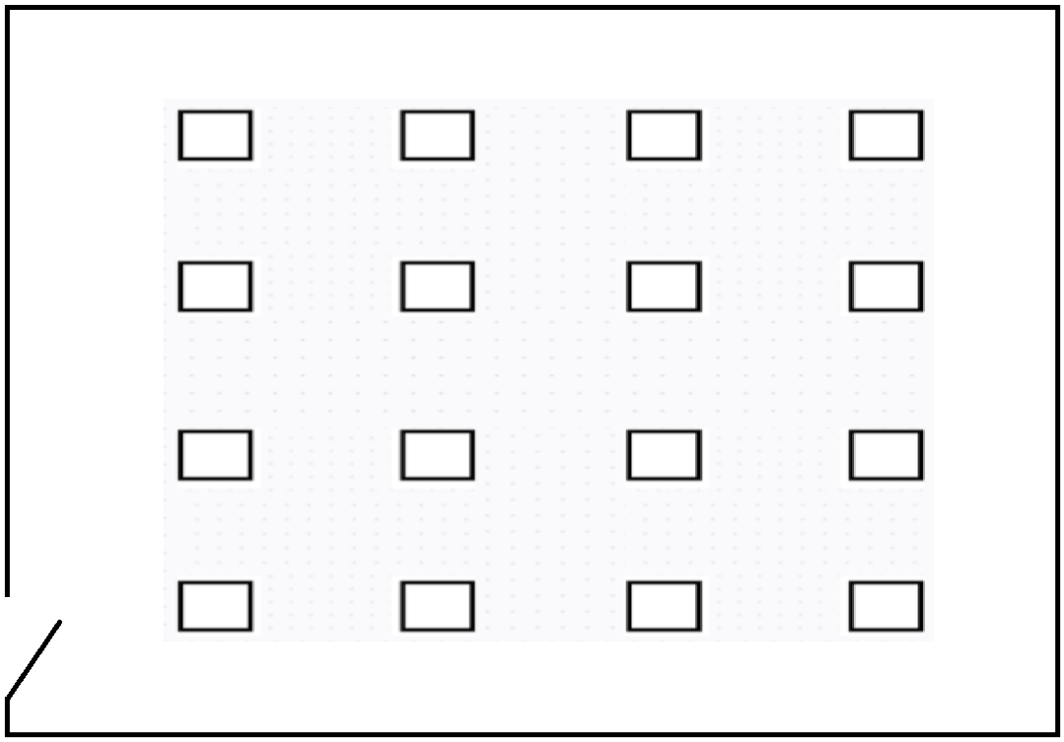


Рисунок 5.1 – Схема розташування світильників у приміщенні

Дотримання стандартів освітлення дозволяє зменшити ризики професійного зорового синдрому, підвищити мотивацію працівника та запобігти виникненню помилок під час виконання складних розробницьких задач.

У деяких випадках може бути доцільно використовувати лампи з регулюванням температури світла — від теплого до холодного. Це дає змогу адаптувати освітлення під індивідуальні потреби користувача. Наприклад, для вечірньої роботи більше підходить тепле м’яке світло, яке менше навантажує очі.

Застосування сучасних світлодіодних систем освітлення дозволяє не лише досягти нормативної освітленості, а й забезпечити енергоефективність. Вони споживають менше електроенергії, мають довший термін служби та краще підходять для тривалої експлуатації в офісних умовах.

Розміщення світильників також має враховувати розташування моніторів, щоб уникати утворення бліків на екрані. Ідеальним варіантом вважається непряме розсіяне світло, яке забезпечує рівномірне освітлення без тіней.

Важливо враховувати також колір стін та стелі: світлі поверхні краще відбивають світло і дозволяють досягти кращого рівня освітленості при меншій кількості джерел світла.

Додатково, рекомендується періодично проводити перевірку рівня освітлення та очищення світильників від пилу, що дозволить підтримувати ефективність освітлення на стабільному рівні.

Таким чином, правильне проєктування системи освітлення робочого місця розробника сприяє покращенню умов праці, підвищенню продуктивності та збереженню здоров’я.

5.2 Організація раціонального режиму праці та відпочинку користувачів ПК

Раціональний режим праці є одним із ключових елементів забезпечення охорони праці та запобігання професійним захворюванням. Сучасні професії, пов’язані з роботою за комп’ютером, вимагають високої концентрації уваги, інтенсивної роботи очей, а також тривалого перебування у сидячому положенні, що створює серйозне навантаження на організм людини. З огляду на це, впровадження науково обґрунтованого режиму праці та відпочинку є надзвичайно важливим.

Тривале перебування перед монітором без належних перерв може призвести до зниження концентрації, дратівливості, болю в очах і навіть до професійних захворювань. Тому надзвичайно важливо впровадити чіткий графік роботи з регулярними перервами.

Відповідно до сучасних рекомендацій з охорони праці:

- Після кожної години роботи за комп'ютером слід робити перерву тривалістю 10–15 хвилин;
- При роботі понад 4 години рекомендується планувати одну довгу перерву (30 хв) після 2 годин роботи;
- Під час перерв бажано виконувати вправи для очей (наприклад, метод 20-20-20: кожні 20 хвилин дивитись на об'єкт на відстані 20 футів протягом 20 секунд);
- Проводити легкі фізичні розминки: обертання плечима, нахили голови, вправи на розтяжку рук і спини;
- Дотримуватись оптимальної температури повітря в приміщенні (18–24°C) і вологості (40–60%).

З метою вдосконалення системи режимів праці та відпочинку доцільно враховувати також:

- Планування структури робочого дня: починати зі спрощених завдань, поступово переходячи до складніших;
- Створення індивідуального графіку, адаптованого під характер діяльності конкретного працівника;
- Забезпечення психологічного комфорту на робочому місці (ергономічні меблі, тиха обстановка, відсутність дратівливих чинників).

Впровадженню таких режимів повинна передувати робота щодо їх обґрунтування, заснована на урахуванні змісту праці, психофізіологічного навантаження та індивідуальних особливостей працівника. Особливо це актуально для осіб, що виконують творчу роботу за комп'ютером — таких як програмісти, дизайнери, інженери-програмісти, тощо. Для розробника ігор, який постійно працює з ПК, необхідно правильно планувати структуру робочого дня, щоб уникнути перевтоми та зниження працездатності.

Режими праці та відпочинку повинні враховувати:

- Етапність і плавність входження в робочий процес. На початку зміни варто починати з простих завдань, поступово переходячи до складніших, особливо у творчих видах діяльності.
- Ритмічність праці. Робота повинна бути структурованою таким чином, щоб розробник мав змогу рівномірно розподіляти навантаження між фазами роботи та перервами.
- Раціональне планування перерв. Перерви мають враховувати не лише час, але і вид навантаження. Наприклад, після тривалої роботи з мишею чи клавіатурою — доцільно виконати вправи для кистей, передпліч, плечового поясу.

Згідно з нормативними документами, користувачі ПК поділяються на три групи трудової діяльності:

- Група А — читання з екрана;
- Група Б — введення інформації;
- Група В — творча робота (налагодження, розробка, редагування).

Рівень навантаження поділяється на три категорії:

- I — низьке навантаження (до 2 год роботи);
- II — середнє (2–4 год);
- III — високе (понад 4 год).

На основі цих груп і категорій встановлюється тривалість регламентованих перерв, як показано в таблиці 5.1:

Таблиця 5.1 – Рекомендована тривалість перерв під час роботи за ПК залежно від категорії та характеру трудової діяльності

Категорія	Група А (зчитування), знаків	Група Б (введення), знаків	Група В (творча робота), год	Рекомендована тривалість перерв, хв
I	до 20 000	до 15 000	до 2	20
II	21 000–40 000	16 000–30 000	2,1–4	40
III	понад 40 000	понад 30 000	понад 4	60

При цьому безперервна робота за ПК не повинна перевищувати 2 годин. У разі нічної зміни тривалість перерв збільшується на 60 хвилин.

Під час перерв рекомендується:

- Виконувати вправи для очей, шиї та спини;
- Змінювати форму діяльності (наприклад, перейти від введення тексту до читання або перевірки результатів);
- Використовувати кімнати психологічного розвантаження (за наявності).

Також важливо враховувати індивідуальні особливості та рівень кваліфікації користувача. Наприклад, новачок потребує більше часу на адаптацію, тому йому варто надати додаткові інструкції, навчальні матеріали, короткі довідники з програмного забезпечення.

Особливу увагу слід приділяти інтенсивній творчій діяльності — наприклад, налагодженню ігрової логіки, створенню анімацій або оптимізації коду. У таких випадках рекомендовано використовувати чергування форм діяльності: перемикання між візуальними та логічними завданнями, перехід від роботи зі скриптами до графіки тощо.

Слід також зазначити, що ефективність режиму праці залежить від індивідуальних особливостей користувача, зокрема його рівня підготовки, фізіологічного стану та навіть мотивації. Тому розробка персоналізованих рекомендацій і ведення коротких інструкцій безпосередньо на робочому місці може значно підвищити результативність праці та покращити самопочуття працівника.

Таким чином, забезпечення раціонального режиму праці та відпочинку користувачів ПК є невід'ємною складовою ефективного процесу розробки програмного забезпечення і дозволяє зберігати здоров'я, зменшувати втому, уникати професійних ризиків і підвищувати загальну якість результату.

У випадку скарг на зорове перенапруження або інші прояви втоми слід розробляти індивідуальні режими праці, які дозволять уникнути перевантажень.

Висновок:
Впровадження науково обґрунтованого режиму праці та відпочинку користувачів ПК сприяє покращенню якості виконання інтелектуальних завдань, збереженню здоров'я розробників та підвищенню ефективності їхньої роботи. В умовах сучасної цифрової економіки такі заходи не лише рекомендовані, а й необхідні для успішного функціонування ІТ-команд та підтримки високого професійного рівня працівників.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи була успішно розроблена 2D-гра в жанрі top-down shooter під назвою "Void Rush", яка реалізує всі основні етапи життєвого циклу програмного продукту: від постановки задачі та вибору інструментів — до тестування, налагодження та створення документації.

Робота виконувалася із дотриманням принципів програмної інженерії, графічного дизайну, техніки безпеки та екологічних вимог.

Було реалізовано такі ключові компоненти гри:

- функціональне переміщення гравця та механіка стрільби;
- базова система ворогів із поведінкою, що реагує на гравця;
- система прокачки персонажа з використанням досвіду (XP) та меню вибору покращень;
- інтерфейс користувача, що відображає здоров’я, досвід та рівень;
- головне меню, екран завершення гри та пауза;
- графічне оформлення в стилі піксель-арт, адаптоване до космічної тематики;
- звукові ефекти для стрільби, фонова музика, звуки подій (поразка, меню тощо);
- зручність управління, оптимізація та стабільна робота гри.

Під час реалізації роботи були виявлені й усунені численні помилки, що дозволило досягти стабільної роботи гри. У процесі тестування перевірялися всі основні ігрові механіки, функціональність інтерфейсу та взаємодія між скриптами. Було проведено повноцінну налагоджувальну діяльність, яка охоплювала модульне, інтеграційне та функціональне тестування.

Дана кваліфікаційна робота відповідає цілям, поставленим на етапі технічного завдання, та демонструє високий рівень опанування навичками роботи з середовищем Unity, мовою програмування C#, а також знаннями у сфері побудови програмних систем. Розроблений продукт може слугувати основою для подальшого

розширення: додавання нових рівнів, типів ворогів, режимів гри, мережевої взаємодії або мобільної версії.

Таким чином, кваліфікаційна робота дозволила досягти поставлених цілей, закріпити знання, отримані під час навчання, та отримати безцінний досвід практичної реалізації даної розробки з нуля.

Розробка гри стала не лише навчальним завданням, а й інструментом самовираження, творчості та першого кроку до професійної діяльності у сфері розробки програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unity Technologies. Unity Documentation [Електронний ресурс]. — Режим доступу: https://docs.unity.com (дата звернення: 11.01.2025).
2. C# Programming Guide – Microsoft Learn [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 12.01.2025).
3. GameDev Market — Безкоштовні ресурси для розробників ігор [Електронний ресурс]. — Режим доступу: https://www.gamedevmarket.net (дата звернення: 16.01.2025).
4. itch.io — Безкоштовні та преміум ресурси для 2D-ігор [Електронний ресурс]. — Режим доступу: https://itch.io (дата звернення: 16.01.2025).
5. Stack Overflow — Спільнота розробників [Електронний ресурс]. — Режим доступу: https://stackoverflow.com (дата звернення: 16.06.2025).
6. GIMP 3.0.4 Documentation [Електронний ресурс]. — Режим доступу: https://docs.gimp.org (дата звернення: 12.01.2025).
7. FL Studio 2024. Офіційна документація [Електронний ресурс]. — Режим доступу: https://www.image-line.com/fl-studio/ (дата звернення: 03.02.2025).
8. Visual Studio 2022 Documentation [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/visualstudio/ (дата звернення: 12.01.2025).
9. Радченко О.В. Методичні вказівки до виконання дипломного проєкту. — ТК ТНТУ, 2022.
10. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання.
11. Deep-Fold. Pixel Planet Generator [Електронний ресурс]. — Режим доступу: https://deep-fold.itch.io/pixel-planet-generator (дата звернення: 18.01.2025).
12. Foozlecc. Void Main Ship Assets [Електронний ресурс]. — Режим доступу: https://foozlecc.itch.io/void-main-ship (дата звернення: 18.02.2025).

13. Screaming Brain Studios. Space Backgrounds [Електронний ресурс]. — Режим доступу: https://screamingbrainstudios.itch.io/seamless-space-backgrounds (дата звернення: 19.02.2025).

14. Створення ігор в Construct https://itschool.ua/blogua/movi-rozmitka-stili/stvorennya-igor-v-construct (дата звернення: 06.02.2025).

15. Документація до Godot Engine 4.4 українською мовою https://docs.godotengine.org/uk/4.x/getting_started/introduction/introduction_to_godot.html (дата звернення: 01.02.2025).

16. Про середовище програмування C# https://foxminded.ua/seredovyshche-prohramuvannia-si-sharp (дата звернення: 10.02.2025).

Додатки

Додаток А

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

public class PlayerShooting : MonoBehaviour
{
    public Transform firePointLeft;
    public Transform firePointRight;
    public GameObject bulletPrefab;

    public Animator gunAnimatorLeft;
    public Animator gunAnimatorRight;
    public AudioClip shootSound;
    private AudioSource audioSource;

    private float nextFireTime = 0f;

    private PlayerStats playerStats;



    void Start()
    {
        playerStats = GetComponent<PlayerStats>();
        audioSource = GetComponent<AudioSource>();
    }

    void Update()
    {
        AimTowardsMouse();

        if (Input.GetButton("Fire1") && Time.time >= nextFireTime)
        {
            Shoot();
            nextFireTime = Time.time + playerStats.fireRate;
        }
    }
```

```
    void AimTowardsMouse()
    {
        Vector3 mousePos =
Camera.main.ScreenToWorldPoint(Input.mousePosition);
        Vector2 direction = (mousePos - transform.position).normalized;

        float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        transform.rotation = Quaternion.Euler(0f, 0f, angle - 90f);
    }

    void Shoot()
    {
        Instantiate(bulletPrefab, firePointLeft.position, firePointLeft.rotation);
        Instantiate(bulletPrefab, firePointRight.position, firePointRight.rotation);

        if (gunAnimatorLeft != null)
            gunAnimatorLeft.SetTrigger("Fire");

        if (gunAnimatorRight != null)
            gunAnimatorRight.SetTrigger("Fire");

        if (shootSound != null && audioSource != null)
        {
            audioSource.PlayOneShot(shootSound);
        }

    }
}

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

public class PlayerMovement : MonoBehaviour
{
    private Rigidbody2D rb;
    private Vector2 movement;

    private PlayerStats playerStats;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
```

```
        playerStats = GetComponent<PlayerStats>();
    }

    void Update()
    {
        movement.x = Input.GetAxisRaw("Horizontal");
        movement.y = Input.GetAxisRaw("Vertical");
    }

    void FixedUpdate()
    {
        rb.MovePosition(rb.position + movement.normalized *
playerStats.moveSpeed * Time.fixedDeltaTime);
    }
}
```

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

public class PlayerHealthBar : MonoBehaviour
{
    public Image fillImage;
    public PlayerStats playerStats;

    void Update()
    {
        fillImage.fillAmount = (float)playerStats.currentHealth /
playerStats.maxHealth;
    }
}
```

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

public class PlayerHealth : MonoBehaviour
{
    public Slider healthBar;

    private PlayerStats playerStats;
```

```
    void Start()
    {
        playerStats = GetComponent<PlayerStats>();
        playerStats.currentHealth = playerStats.maxHealth;

        healthBar.maxValue = playerStats.maxHealth;
        healthBar.value = playerStats.currentHealth;
    }

    public void TakeDamage(int amount)
    {
        playerStats.currentHealth -= amount;
        healthBar.value = playerStats.currentHealth;

        if (playerStats.currentHealth <= 0)
        {
            Die();
        }
    }

    void Die()
    {
        Debug.Log("Player died");
        gameObject.SetActive(false);
        FindObjectOfType<GameManager>().GameOver();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PlayerXP : MonoBehaviour
{
    public int currentXP = 0;
    public int level = 1;
    public int xpToNextLevel = 10;

    public Slider xpBar;
    public Text levelText;

    public LevelUpSystem levelUpSystem;
```

```
void Start()
{
    UpdateUI();
}

public void AddXP(int amount)
{
    currentXP += amount;

    if (currentXP >= xpToNextLevel)
    {
        LevelUp();
    }

    UpdateUI();
}

void LevelUp()
{
    level++;
    currentXP -= xpToNextLevel;
    xpToNextLevel += 5;

    Debug.Log("LEVEL UP!");

    levelUpSystem.TriggerLevelUp();

    UpdateUI();
}

void UpdateUI()
{
    if (xpBar != null)
    {
        xpBar.maxValue = xpToNextLevel;
        xpBar.value = currentXP;
    }

    if (levelText != null)
    {
        levelText.text = "Lv: " + level;
    }
```

```
    }
}

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

public class PlayerStats : MonoBehaviour
{
    public int maxHealth = 100;
    public int currentHealth = 100;
    public float moveSpeed = 5f;
    public float fireRate = 0.5f;
}

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

using System;

[Serializable]
public class UpgradeData
{
    public string upgradeName;
    public string description;
    public Action applyEffect;

    public UpgradeData(string name, string desc, Action effect)
    {
        upgradeName = name;
        description = desc;
        applyEffect = effect;
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

[System.Serializable]
public class UpgradeOption
```

```
{
    public string upgradeType;
    public string label;
    public Sprite icon;
}

public class LevelUpSystem : MonoBehaviour
{
    public GameObject levelUpPanel;
    public Button[] upgradeButtons;

    public List<UpgradeOption> upgradeOptions;

    public PlayerXP playerXP;
    public PlayerStats playerStats;

    void Start()
    {
        levelUpPanel.SetActive(false);
    }

    public void TriggerLevelUp()
    {
        Debug.Log("Викликається меню Level Up");
        Time.timeScale = 0f;
        levelUpPanel.SetActive(true);
        ShowRandomUpgrades();
    }

    void ShowRandomUpgrades()
    {
        List<UpgradeOption> optionsPool = new
List<UpgradeOption>(upgradeOptions);

        for (int i = 0; i < upgradeButtons.Length; i++)
        {
            int index = Random.Range(0, optionsPool.Count);
            UpgradeOption option = optionsPool[index];
            optionsPool.RemoveAt(index);


            Text labelText = upgradeButtons[i].GetComponentInChildren<Text>();
            if (labelText != null)
```

```
            labelText.text = option.label;


            Transform iconTransform = upgradeButtons[i].transform.Find("Icon");
            if (iconTransform != null)
            {
                Image iconImage = iconTransform.GetComponent<Image>();
                if (iconImage != null)
                    iconImage.sprite = option.icon;
            }


            upgradeButtons[i].onClick.RemoveAllListeners();
            upgradeButtons[i].onClick.AddListener(() =>
ApplyUpgrade(option.upgradeType));
        }
    }


    void ApplyUpgrade(string upgrade)
    {
        switch (upgrade)
        {
            case "HP":
                playerStats.maxHealth += 20;
                playerStats.currentHealth = playerStats.maxHealth;
                break;
            case "Speed":
                playerStats.moveSpeed += 0.5f;
                break;
            case "FireRate":
                playerStats.fireRate = Mathf.Max(0.1f, playerStats.fireRate - 0.05f);
                break;
        }

        levelUpPanel.SetActive(false);
        Time.timeScale = 1f;
    }
}

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

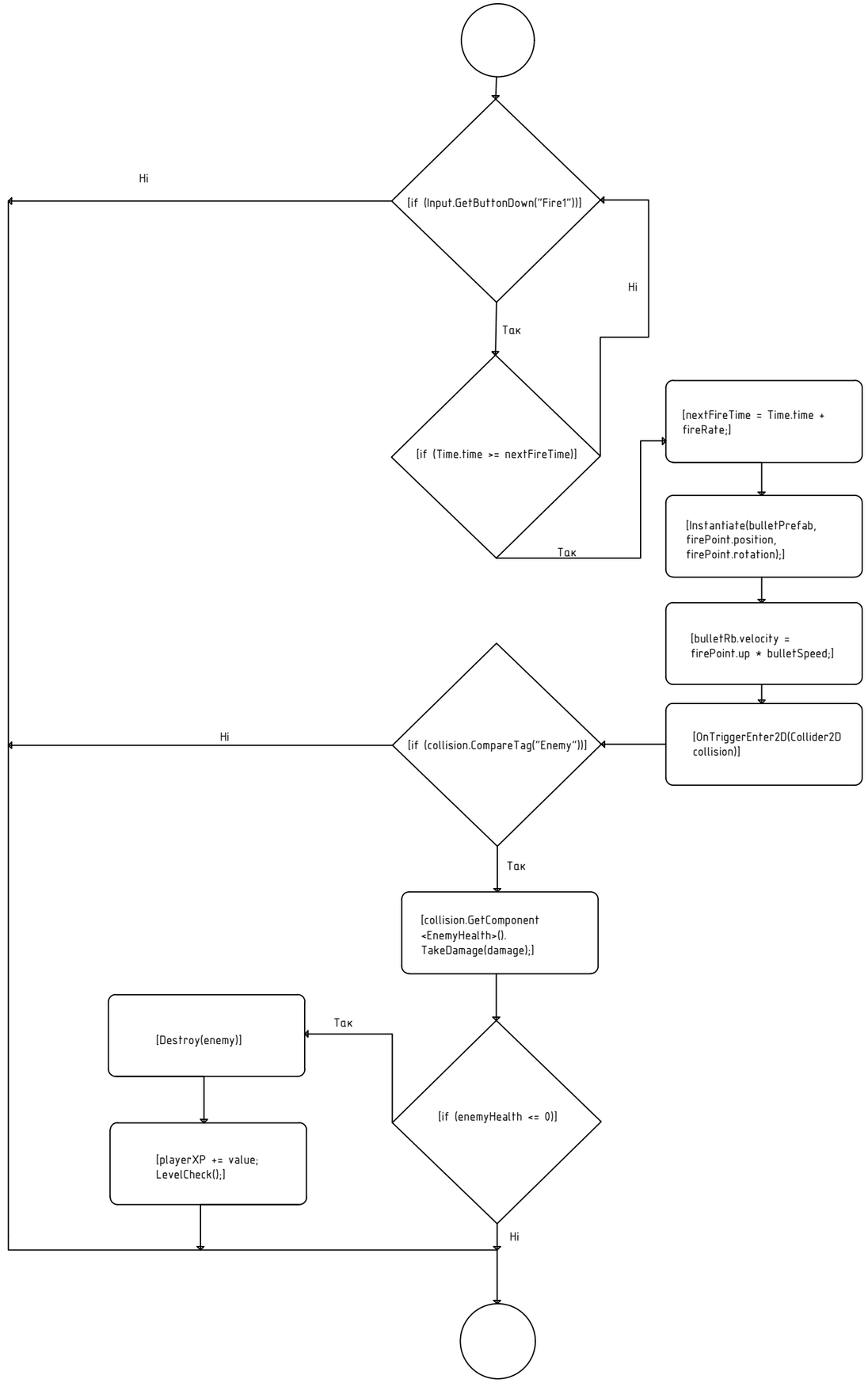
```
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    public GameObject gameOverPanel;
    public AudioSource gameOverAudio;

    public void GameOver()
    {
        if (gameOverAudio != null)
        {
            gameOverAudio.Play();
        }

        Time.timeScale = 0f;
        gameOverPanel.SetActive(true);
    }

    public void RestartGame()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public void LoadMainMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }
}

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

public class TargetHealth : MonoBehaviour
{
    public int health = 30;
    public int xpValue = 5;

    public GameObject explosionPrefab;
    public AudioClip deathSound;
```

```
    public void TakeDamage(int amount)
    {
        health -= amount;
        Debug.Log("Target HP: " + health);

        if (health <= 0)
        {
            Die();
        }
    }

    void Die()
    {
        PlayerXP playerXP =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerXP>();
        if (playerXP != null)
        {
            playerXP.AddXP(xpValue);
        }

        if (explosionPrefab != null)
        {
            Instantiate(explosionPrefab, transform.position, Quaternion.identity);
        }

        if (deathSound != null)
        {
            AudioSource.PlayClipAtPoint(deathSound, transform.position);
        }

        Destroy(gameObject);
    }
}

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

[System.Serializable]
public class EnemySpawnData
{
    public GameObject enemyPrefab;
    [Range(0f, 1f)] public float spawnChance;
```

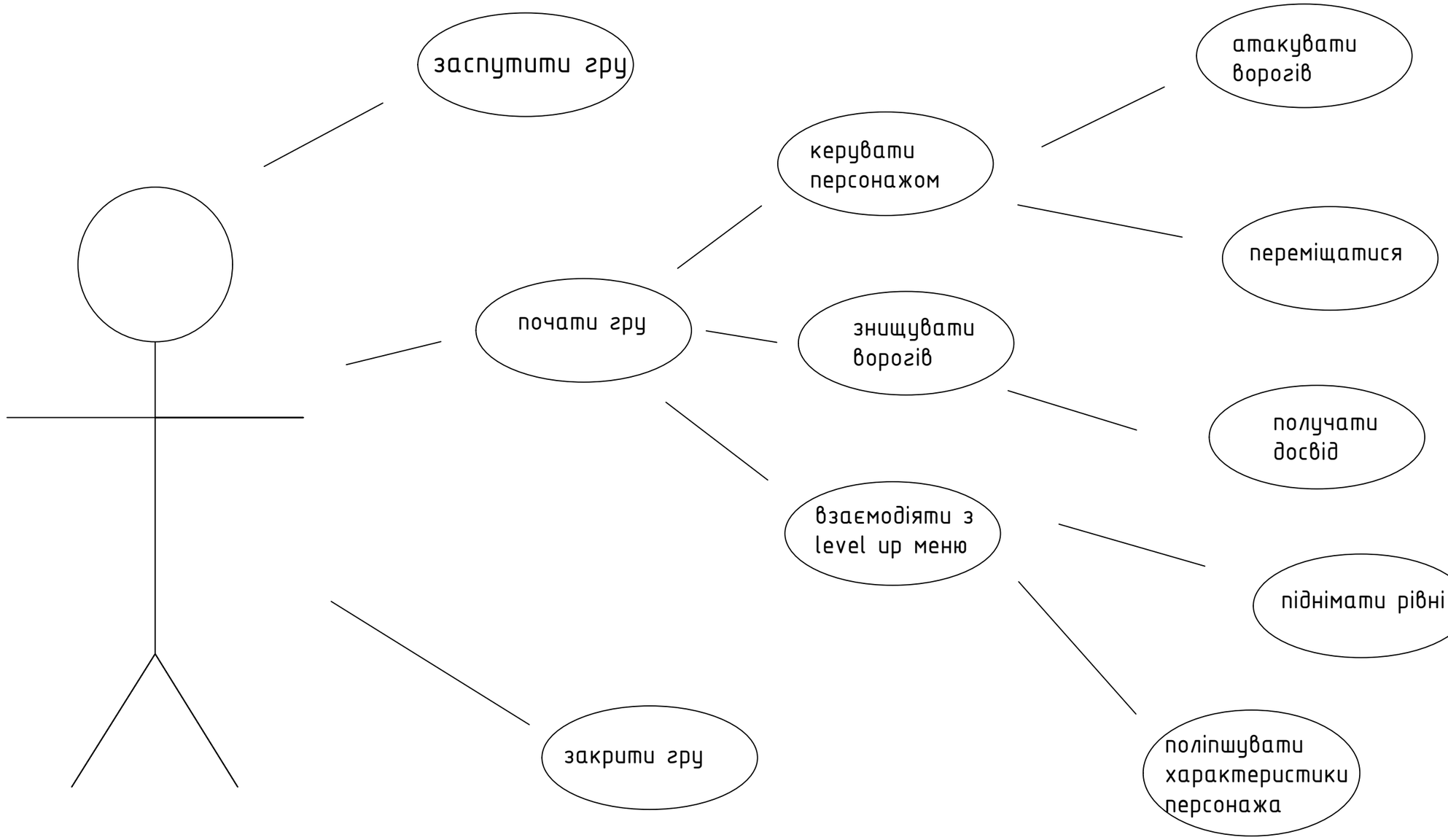
```
}

public class EnemySpawner : MonoBehaviour
{
    public List<EnemySpawnData> enemies;
    public float spawnInterval = 3f;
    public float spawnRadius = 10f;
    private float nextSpawnTime = 0f;

    void Update()
    {
        if (Time.time >= nextSpawnTime)
        {
            SpawnEnemy();
            nextSpawnTime = Time.time + spawnInterval;
        }
    }

    void SpawnEnemy()
    {
        GameObject prefabToSpawn = ChooseEnemy();

        if (prefabToSpawn != null)
        {
            Vector2 spawnPosition = (Vector2)transform.position +
Random.insideUnitCircle.normalized * spawnRadius;
            Instantiate(prefabToSpawn, spawnPosition, Quaternion.identity);
        }
    }

    GameObject ChooseEnemy()
    {
        float total = 0f;
        foreach (var enemy in enemies)
        {
            total += enemy.spawnChance;
        }

        float rand = Random.value * total;
        float cumulative = 0f;

        foreach (var enemy in enemies)
        {
```

```
            cumulative += enemy.spawnChance;
            if (rand <= cumulative)
                return enemy.enemyPrefab;
        }

        return null;
    }
}

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

public class EnemyFollow : MonoBehaviour
{
    public float speed = 2f;
    private Transform player;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").transform;
    }

    void Update()
    {
        if (player != null)
        {
            Vector2 direction = (player.position - transform.position).normalized;
            transform.position += (Vector3)(direction * speed * Time.deltaTime);

            float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
            transform.rotation = Quaternion.Euler(0f, 0f, angle - 90f);
        }
    }
}

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

public class EnemyAttack : MonoBehaviour
{
    public int damage = 10;
```

```
    public float damageRate = 1f;
    private float nextDamageTime = 0f;

    void OnCollisionStay2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player") && Time.time >=
nextDamageTime)
        {
            PlayerHealth playerHealth =
collision.gameObject.GetComponent<PlayerHealth>();
            if (playerHealth != null)
            {
                playerHealth.TakeDamage(damage);
                nextDamageTime = Time.time + damageRate;
            }
        }
    }
}

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

public class MainMenu : MonoBehaviour
{
    public void StartGame()
    {
        SceneManager.LoadScene("GameScene");
    }

    public void ExitGame()
    {
        Debug.Log("Вихід з гри");
        Application.Quit();
    }
}
```



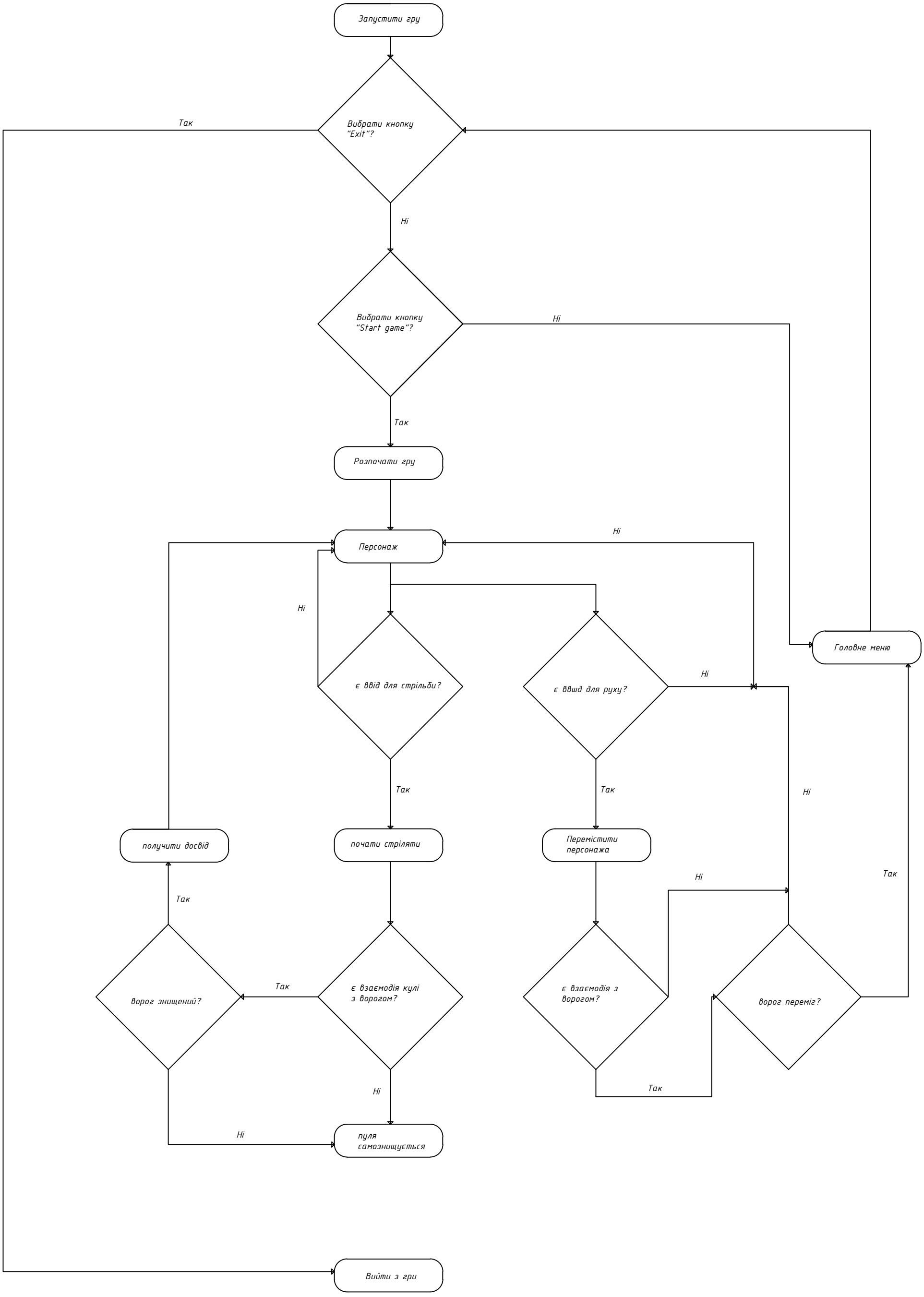
						2025.КРБ.123.602.25.00.00 БС	
Зм.	Арк.	№ документу	Підпис	Дата	Розробка 2D-гри жанру Top-Down Shooter на ігровому руші Unity. Алгоритм бойової системи у грі	Літ.	Маса / Масшт.
Розробив		Сиримула А.О					
Перевірив		Штокало Л.Я.				Аркуш	Аркушів 1
Консульт.							
Реценз.							
Н.Контр.		Приймак В.А.				ВСП"ТФК ТНТУ імені Івана Пулюя" Кіб-602п	
Зав. каф.							



						2025.КРБ.123.602.25.00.00 ВД	
Зм.	Арк.	№ документу	Підпис	Дата	Розробка 2D-гри жанру Top-Down Shooter на ігровому руші Unity Варіанти дій гравця у грі	Літ.	Маса / Масшт.
Розробив		Сиримула.А.О					
Перевірив		Штокало Л.Я.				Аркуш	Аркушів 1
Консульт.						ВСП"ТФК ТНТУ імені Івана Пулюя" Кіб-602п	
Рецензі.							
Н.Контр		Приймак В.А.					
Зав. каф.							

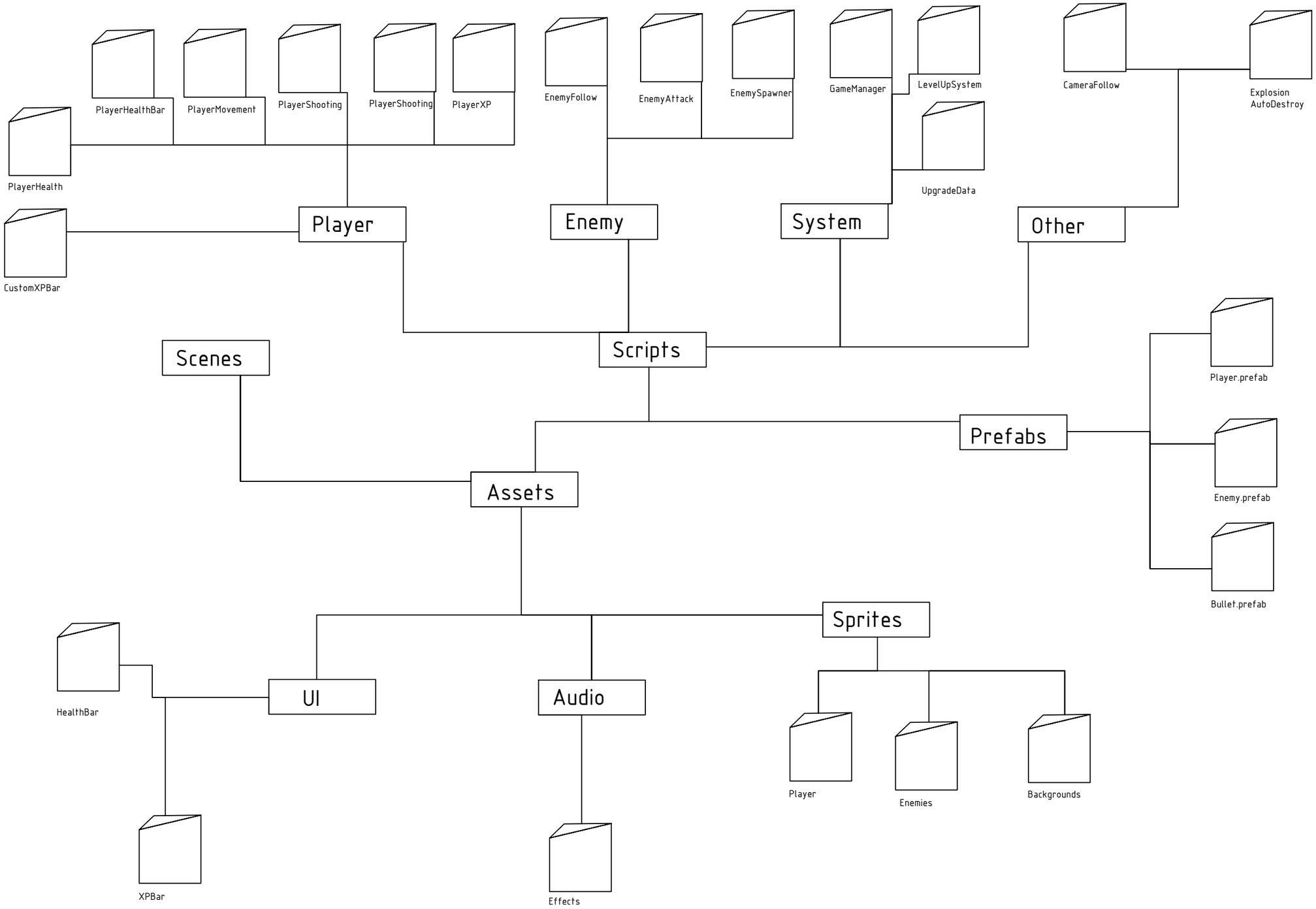
Таблиця техніко-економічних показників

№п/п	Назва показника	Значення	№п/п	Назва показника	Од.виміру	Значення
1	Мова програмування	C#	6	Загальна трудомісткість	години	180
2	платформа	Unity	7	Собівартість	грн	36115,91
3	Програма для створення графічної частини	Gimp	8	Орієнтовний прибуток	грн	60 000
4	Програма для створення аудіо	FL studio	9	Кількість ігрових об'єктів	одиниць	50+
5	Розмір збірки (exe-файл)	65 мб	10	Термін окупності	рік	1,79



						2025.КРБ.123.602.25.00.00 ДП	
Зм.	Арк.	№ документу	Підпис	Дата	Розробка 2D-гри жанру Top-Down Shooter на ігровому рушії Unity Діаграма послідовності виконання функціональних етапів гри	Літ.	Маса / Масшт.
Розробив		Сиримула.А.О.					
Перевірив		Штокало Л.Я.				Аркуш	Аркушів 1
Консульт.							
Реценз.						ВСП "ТФК ТНТУ імені Івана Пулюя" Кіб-602п	
Н.Контр		Приймак В.А.					
Зав. каф.							

```
void ShowRandomUpgrades()
 {
    List<UpgradeOption> optionsPool = new
List<UpgradeOption>(upgradeOptions);

    for (int i = 0; i < upgradeButtons.Length; i++)
    {
       int index = Random.Range(0, optionsPool.Count);
       UpgradeOption option = optionsPool[index];
       optionsPool.RemoveAt(index);


       Text labelText =
upgradeButtons[i].GetComponentInChildren<Text>();
       if (labelText != null)
          labelText.text = option.label;


       Transform iconTransform =
upgradeButtons[i].transform.Find("Icon");
       if (iconTransform != null)
       {
          Image iconImage =
iconTransform.GetComponent<Image>();
          if (iconImage != null)
             iconImage.sprite = option.icon;
       }


       upgradeButtons[i].onClick.RemoveAllListeners();
       upgradeButtons[i].onClick.AddListener(() =>
ApplyUpgrade(option.upgradeType));
    }
 }
```



						2025.КРБ.123.602.25.00.00 СС	
Зм.	Арк.	№ документу	Підпис	Дата	Розробка 2D-гри жанру Top-Down Shooter на ігровому рушії Unity. Структура файлів і каталогів проекту "Void Rush"	Літ.	Маса / Масшт.
Розробив		Сиримула А.О					
Перевірив		Штокало Л.Я.				Аркуш	Аркушів 1
Консульт.							
Рецензз.							
Н.Контр		Приймак В.А.				ВСП"ТФК ТНТУ імені Івана Пулюя" Кіб-602п	
Зав. каф.							