НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»
спеціальності «121 Інженерія програмного забезпечення»

на тему: Мобільна гра в жанрі Shooter з елементами RogueLike

Виконав студент IV курсу, групи ІП-12
(шифр групи)

Корнієнко Валерій Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент, Зарічковий О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент асистент кафедри ІСТ, PhD, Альбрехт Й. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії
Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 121 Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Корнієнку Валерію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Мобільна гра в жанрі Shooter з елементами RogueLike
керівник проєкту Зарічковий Олександр Анатолійович, асистент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. № 1705-с
2. Термін подання студентом проєкту «16» червня 2025 року
3. Вихідні дані до проєкту: технічне завдання
4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, аналіз відомих програмних продуктів, аналіз відомих алгоритмічних та технічних рішень, опис бізнес-процесів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, розроблення функціональних вимог, розроблення нефункціональних вимог.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення, аналіз безпеки даних.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості програмного забезпечення, опис процесів тестування, опис контрольного прикладу.
4) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.
6) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання

2) Схема структурна діяльності

3) Схема структурна компонентів архітектури станів гри

4) Схема структурна компонентів процесу генерації рівнів

5) Схема структурна компонентів архітектури ШІ ворогів

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» березня 2025 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	20.03.2024	
3	Постановка та формалізація задачі	25.03.2024	
4	Розробка інформаційного забезпечення	30.03.2024	
5	Алгоритмізація задачі	05.04.2024	
6	Обґрунтування вибору використаних технічних засобів	10.04.2024	
7	Розробка програмного забезпечення	15.04.2024	
8	Налагодження програми	10.05.2024	
9	Виконання графічних документів	20.05.2024	
10	Оформлення пояснювальної записки	29.05.2024	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Валерій КОРНІЄНКО

(підпис) (ініціали, прізвище)

Керівник _____ Олександр ЗАРІЧКОВИЙ

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 37 таблиць, 35 рисунків та 52 джерела – загалом 82 сторінки.

Дипломний проєкт присвячений розробці мобільної гри в жанрі Shooter з елементами RogueLike.

Метою розробки є автоматизація та спрощення створення ігрових рівнів шляхом застосування комбінованого алгоритму процедурної генерації, а також розробка комплексної й адаптивної системи штучного інтелекту для ефективного налаштовування поведінки ігрових персонажів та ворогів.

У першому розділі наведено аналіз галузі відеоігор, зокрема мобільних roguelike, проведено порівняльний аналіз відомих програмних продуктів та алгоритмічних рішень для процедурної генерації рівнів, алгоритмів ШІ для ворогів, описано бізнес-процеси.

Другий розділ присвячений розробці варіантів використання, функціональних, нефункціональних і системних вимог. Також в цьому розділі описана оцінка економічної ефективності розробки та постановка задачі.

В третьому розділі описано архітектуру програмного забезпечення, алгоритм процедурної генерації рівнів та алгоритм ШІ для ворогів, проаналізовано безпеку даних та обґрунтовано вибір засобів розробки.

В четвертому розділі проведено аналіз якості ПЗ та проведено тестування розробленого програмного забезпечення, окремо наведено контрольний приклад.

В п’ятому розділі описано процес розгортання та супроводу програмного забезпечення.

КЛЮЧОВІ СЛОВА: МОБІЛЬНА ГРА, UNITY, ANDROID, SHOOTER, ROGUELIKE, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ГЕЙМПЛЕЙ, ІГРОВИЙ РІВЕНЬ, СЦЕНАРІЇ ВОРОГІВ, АЛГОРИТМИ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 37 tables, 35 figures and 52 sources – in total 82 pages.

The purpose of this diploma project is to automate and simplify the creation of game levels by applying a combined procedural-generation algorithm, as well as to design a comprehensive and adaptive artificial-intelligence system for effectively configuring the behavior of player characters and enemies.

In the first chapter, an analysis of the video-game industry—particularly mobile roguelikes—is presented. A comparative study of well-known software products and algorithmic solutions for procedural level generation and enemy AI algorithms is conducted, and the relevant business processes are described.

The second chapter is devoted to the development of use-case scenarios and to defining functional, non-functional, and system requirements. This chapter also includes an evaluation of the economic efficiency of the development and the statement of the problem.

In the third chapter, the software architecture is described, along with the procedural-generation algorithm for levels and the AI algorithm for enemies; data security is analyzed, and the choice of development tools is justified.

The fourth chapter presents a quality-analysis of the software and reports on the testing of the developed system, including a detailed control example.

The fifth chapter describes the process of deployment and maintenance of the software.

KEYWORDS: MOBILE GAME, UNITY, ANDROID, SHOOTER, ROGUELIKE, PROCEDURAL GENERATION, ARTIFICIAL INTELLIGENCE, GAMEPLAY, GAME LEVEL, ENEMY BEHAVIOR, ALGORITHMS.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1214.045490.01.91	Технічне завдання	13	
3	А4	КПІ.ІП-1214.045490.02.81	Пояснювальна записка	83	
4	А4	КПІ.ІП-1214.045490.03.12	Текст програми	56	
5	А4	КПІ.ІП-1214.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1214.045490.05.34	Керівництво користувача	11	
7	А3	КПІ.ІП-1214.045490.06.99	Графічний матеріал	5	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1214.045490.00.90		
Розроб.		Корнієнко В.С.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Зарічковий О.А.					
Н.контр.		Головченко М.М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-12	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

МОБІЛЬНА ГРА В ЖАНРІ SHOOTER З ЕЛЕМЕНТАМИ ROGUELIKE

Технічне завдання

КПІ.ІП-1214.045490.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ЗАРІЧКОВИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Валерій КОРНІЄНКО

Київ – 2025

Зміст

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: МОБІЛЬНА ГРА В ЖАНРІ SHOOTER З ЕЛЕМЕНТАМИ ROGUELIKE.

Галузь застосування: індустрія відеоігор.

Наведене технічне завдання поширюється на розробку мобільної гри в жанрі Shooter з елементами RogueLike «Chaos Corridor» [КПІ.ІП-1214.045490], котра використовується для спрощення процесу створення двовимірних ігрових рівнів у мобільних іграх, оптимізації налаштування ШІ для ворогів та призначена для проведення дозвілля загального кола мобільних користувачів.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки мобільної гри в жанрі Shooter з елементами RogueLike є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для спрощення процесу створення двовимірних ігрових рівнів у мобільних іграх, а також для оптимізації налаштування штучного інтелекту ворогів.

Метою розробки є автоматизація та спрощення створення ігрових рівнів шляхом застосування комбінованого алгоритму процедурної генерації, а також розробка комплексної й адаптивної системи штучного інтелекту для ефективного налаштовувати поведінку ігрових персонажів та ворогів.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– відображення головного меню гри (Рисунок 4.1);

Рисунок 4.1 – Головне меню гри

– відображення користувацького інтерфейсу в грі (Рисунок 4.2);

Рисунок 4.2 – Користувацький інтерфейс в грі

– відображення меню павзи в грі (Рисунок 4.3);

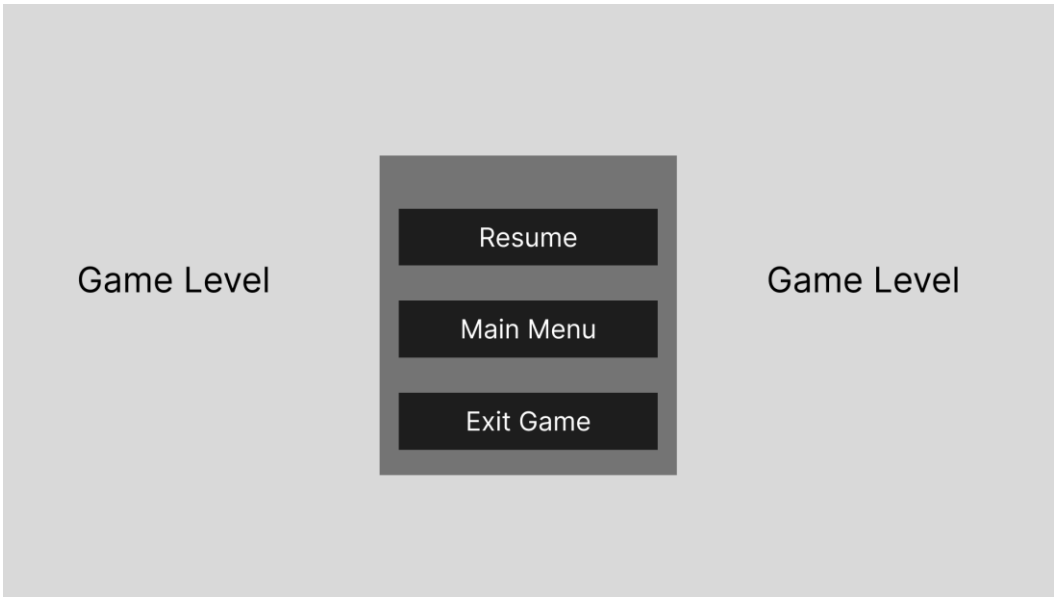


Рисунок 4.3 – Меню павзи в грі

– відображення меню програшу в грі (Рисунок 4.4);

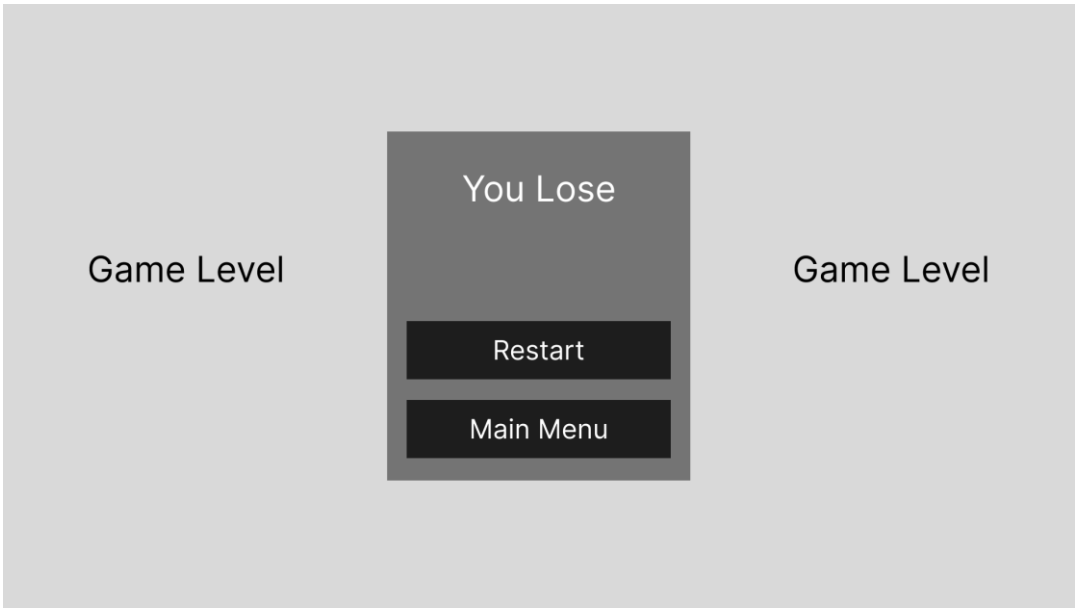


Рисунок 4.4 – Меню програшу в грі

– відображення меню вибору покращень в грі (Рисунок 4.5).

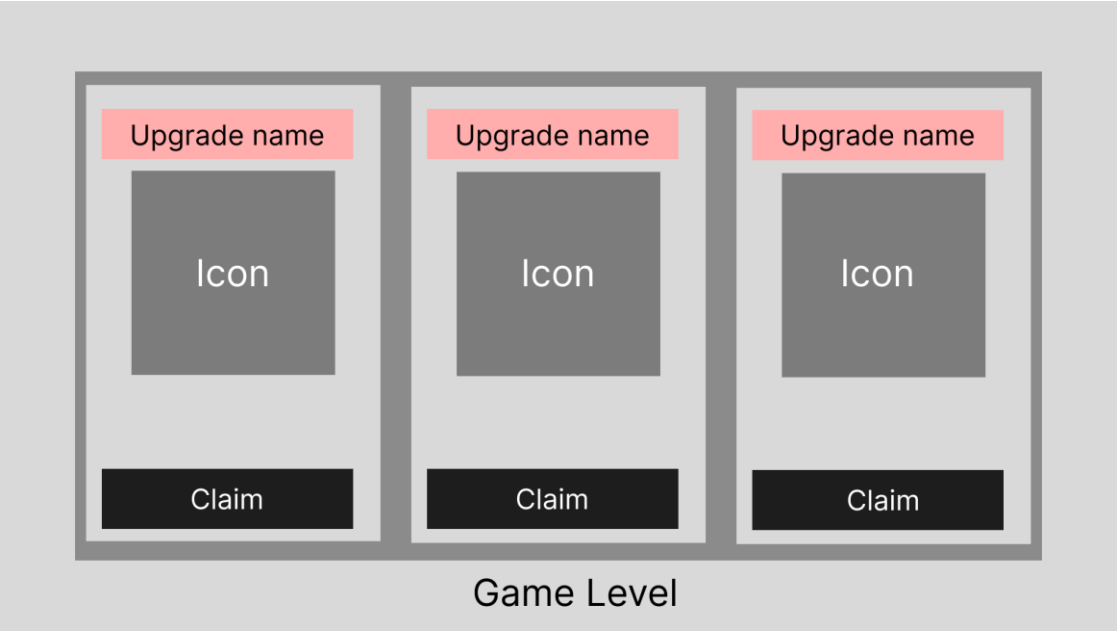


Рисунок 4.5 – Меню вибору покращень в грі

4.1.2 Для користувача:

– можливість генерації ігрового рівня;
– можливість почати гру;
– можливість рухатися по рівню;
– можливість реагувати в ворогами на рівні, з об’єктами рівня;
– можливість отримувати покращення впродовж гри;
– можливість поставити гру на павзу;
– можливість відновити гру;
– можливість вийти з гри.

4.1.3 Додаткові вимоги:

– відображення внутрішньо-ігрового рівня;
– реагування на дії гравця.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення призначене для роботи на мобільних пристроях під управлінням операційних систем Android та iOS.

Мінімальна конфігурація технічних засобів:

– операційна система: Android 8.0 (API level 26) або iOS 12.0;
– обсяг оперативної пам’яті: від 2 ГБ;
– вільне місце на накопичувачі: від 200 МБ;
– доступ до мережі Інтернет (опціонально).

Рекомендована конфігурація технічних засобів:

– операційна система: Android 11 / iOS 15 або новіші;
– обсяг оперативної пам’яті: від 4 ГБ;
– вільне місце на накопичувачі: від 500 МБ;
– дисплей із роздільною здатністю від 720p.

Вимоги до комп’ютера, що використовується для розробки гри:

– тип процесора: Intel Core i5 / Apple M1 або аналогічний;
– обсяг оперативної пам’яті: 8 Гб;
– вільне місце на диску: 20 ГБ;
– підключення до Інтернету зі швидкістю від 100 Мбіт/с;
– встановлене середовище розробки (Unity, Android Studio, Xcode).

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно забезпечувати коректну роботу на мобільних пристроях під управлінням операційних систем Android версії 8.0 (API level 26) або новіших та iOS версії 12.0 або новіших.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C#.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Unity за допомогою середовища JetBrains Rider.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді GitHub-репозиторію.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача;

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна діяльності;
– схема структурна компонентів архітектури станів гри;
– схема структурна компонентів процесу генерації рівнів;
– схема структурна компонентів архітектури ШІ ворогів.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	21.02	
2.	Розробка технічного завдання	15.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проєкту	20.05	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Мобільна гра в жанрі Shooter з елементами RogueLike**

КПІ.ІП-1214.045490.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI – Artificial Intelligence – штучний інтелект.

BSP – Binary Space Partitioning – метод розбиття простору, що використовується для процедурної генерації рівнів.

DI – Dependency Injection – шаблон проєктування, що полягає у передачі залежностей об'єкту ззовні.

FSM – Finite State Machine – скінченний автомат, структура для моделювання поведінки в системах.

IDE – Integrated Development Environment – інтегроване середовище розробки..

MVP – Model-View-Presenter – архітектурний шаблон проєктування інтерфейсу користувача.

UI – User Interface – користувацький інтерфейс.

ВСТУП

Впродовж останніх років індустрія відеоігор зазнала значного розвитку, залучаючи все більше користувачів та фінансів. Особливого зростання зазнала індустрія мобільних ігор. Це пояснюється декількома факторами, такими як різноманітність ігрових жанрів для мобільних девайсів, зростання доступності потужних мобільних девайсів, таких як смартфони, планшети тощо. Відповідно до сучасних досліджень, на 2021 рік на мобільні ігри припадало 52% всього доходу індустрії [1].

Одним із досить популярних жанрів серед відеоігор є Roguelike. Цей жанр відеоігор характеризується високою складністю на реграбельністю. Відмінною його рисою є перманентна смерть, коли після програшу в сесії необхідно починати з самого початку [2]. Це створює високий рівень напруженості геймплею та стимулює гравця до стратегічного планування та обережності.

Історія жанру Roguelike починається у 1980 році з гри «Rogue», яка стала революційним твором та створила хвилю подібних да неї ігор і в результаті призвела до створення терміна "roguelike". Ігри цього жанру були популярні серед студентів та програмістів 1980-х і 1990-х років, що призвело до сотень їх варіацій. З часом жанр розвивався та розширювався, поступово перетворюючись на один із найпопулярніших жанрів сьогодення.

Жанр Roguelike ідеально підходить для мобільного геймінгу завдяки коротким ігровим сесіям та високій реграбельності. Короткі ігрові сесії унікально вписуються у мобільність гравців, дозволяючи їм зануритися у гру під час перерв або вільних моментів, а випадково згенерований контент, наприклад мапи, події, поведінка ворогів, робить кожну ігрову сесію унікальною. Простота управління та низькі вимоги до апаратного забезпечення також сприяють зручності гри на сенсорних екранах.

Іншим популярним жанром відеоігор є жанр Shooter. Даний жанр характеризується високим акцентом механік на стрільбу з вогнепальної зброї [3]. Головна мета гравця в таких іграх полягає у перемозі над ворогами за

допомогою точності стрільби, тактичного мислення та швидкості реакції. Ігри даного жанру бувають як одиночні, так і багатокористувацькі та, аналогічно до жанру Roguelike, часто мають сесійний характер.

Метою дипломної роботи є створення ігрового програмного забезпечення в жанрі Shooter з елементами Roguelike. Дане програмне забезпечення буде містити алгоритми для просунутого штучного інтелекту для ворогів, випадковою генерацією ігрової мапи та геймплейні механіки жанру шутер у поєднанні з механіками жанру Roguelike. Таке поєднання жанрів має забезпечити реграбельність та напруженість жанру Roguelike з цікавий та стратегічний геймплеєм жанру Shooter.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

Дипломне проєктування передбачає виконання наступних завдань:

– аналіз предметної області та опис її ключових бізнес-процесів, визначення загального завдання розробки у рамках ДП;

– аналіз існуючих рішень (як програмних продуктів, так і алгоритмічних чи технічних рішень) обраного завдання розробки у рамках ДП;

– аналіз та моделювання бізнес-процесів;

– розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;

– аналіз економічних показників програмного забезпечення ДП;

– постановка завдання на розробку програмного забезпечення ДП;

– розроблення архітектури програмного забезпечення;

– розроблення архітектурних рішень та обґрунтування вибору засобів розробки програмного забезпечення;

– конструювання та розроблення програмного забезпечення;

– аналіз безпеки даних програмного забезпечення;

– аналіз якості та тестування програмного забезпечення;

– розгортання та супровід програмного забезпечення;

– створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

Відеогра є однією з найбільш популярних форм цифрових розваг, яка охоплює широкий спектр жанрів та платформ. Відеоігри, за своєю суттю, являють собою інтерактивне програмне забезпечення, що використовує графічні зображення для створення віртуального середовища, де користувачі

можуть здійснювати певні дії та досягати поставлених цілей. Станом на сьогодні, ринок відеоігор демонструє стабільне зростання, залучаючи мільйони користувачів по всьому світу та генеруючи значні доходи. Згідно з даними Statista, доходи від продажу відеоігор у 2020 році склали понад 159 мільярдів доларів США, що свідчить про економічну значимість цієї індустрії [4].

Мобільні ігри, зокрема, займають значну частку ринку завдяки своїй доступності та зручності використання. Смартфони та планшети, завдяки своїй портативності, дозволяють грати в ігри будь-де і будь-коли, що робить їх привабливими для широкої аудиторії. Водночас, розробка мобільних ігор має свої специфічні виклики та обмеження. Зокрема, мобільні пристрої мають обмежену обчислювальну потужність та графічні можливості у порівнянні з персональними комп'ютерами та консолями. Це вимагає від розробників значної оптимізації графіки та ресурсів для забезпечення плавного геймплею та високої продуктивності [5]. Крім того, сенсорне управління потребує розробки інтуїтивно зрозумілих інтерфейсів, що забезпечують зручність використання.

Жанр Roguelike є одним з жанрів відеоігор. Він характеризується такими ключовими особливостями, як процедурна генерація рівнів, перманентна смерть персонажа та висока складність гри. Процедурна генерація рівнів, яка використовується у Roguelike іграх, забезпечує створення унікальних ігрових середовищ при кожному новому проходженні, що підвищує повторюваність та інтерес до гри. Ця технологія використовує алгоритми, такі як Perlin Noise та Cellular Automata, для створення складних та непередбачуваних локацій [6].

Штучний інтелект (ШІ) у Roguelike іграх є критично важливим компонентом, який забезпечує виклик для гравця. Використання поведінкових дерев (Behaviour Tree) дозволяє створювати складні та адаптивні моделі поведінки ворогів. Поведінкові дерева представляють собою ієрархічну структуру, яка дозволяє визначати поведінку персонажів залежно від різних

умов та дій гравця [7]. Це забезпечує динамічний геймплей, де вороги можуть змінювати свою тактику в залежності від ситуації.

Жанр Shooter також є одним з найпопулярніших у відеоіграх. Шутери відрізняються швидким темпом подій, динамічним геймплеєм та високим рівнем взаємодії з оточенням. Цей жанр може бути реалізований як від першої особи (First Person Shooter, FPS), так і від третьої особи (Third Person Shooter, TPS). Основними характеристиками шутерів є реалістичність рухів та поведінки персонажів, якість графіки та складність штучного інтелекту ворогів.

Процес використання знань предметної області у програмному забезпеченні на поточний момент розвитку IT-технологій дозволяє ефективно імплементувати ключові особливості Roguelike та Shooter ігор на мобільних платформах. Використання сучасних алгоритмів процедурної генерації та складних моделей штучного інтелекту є стандартною практикою у розробці цих жанрів. Проте існують певні недоліки поточного стану речей. Основні проблеми пов'язані з обмеженою продуктивністю мобільних пристроїв, що може впливати на якість графіки та складність ігрових механік. Крім того, оптимізація сенсорного управління вимагає значних ресурсів та зусиль від розробників для забезпечення зручності та інтуїтивності управління [8].

Можливі шляхи покращення ситуації з розробками у сфері IT за предметною областю включають:

– оптимізація алгоритмів процедурної генерації рівнів;

– впровадження складніших моделей поведінки та використання методів машинного навчання для адаптації AI до стилю гри користувача;

– адаптація геймплейних механік для сенсорних екранів.

У рамках даного дипломного проєкту обрано шлях створення 2D Roguelike гри з акцентом на оптимізацію процедурної генерації рівнів та покращення AI ворогів за допомогою поведінкових дерев. Особлива увага приділяється адаптації гри для мобільних платформ, що включає оптимізацію

управління та продуктивності, забезпечуючи комфортний та захоплюючий геймплей для користувачів

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації мобільной 2д гри в жанрі Shooter з елементами Roguelike. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз гри Dead Cells

"Dead Cells" – це популярна гра в жанрі Roguelike, розроблена французькою студією Motion Twin та випущена в 2018 році [9]. Гра поєднує елементи Roguelike та Metroidvania [10], створюючи унікальний ігровий досвід. Основною особливістю "Dead Cells" є процедурна генерація рівнів, що забезпечує унікальність кожного проходження. Крім того, гра відрізняється високою якістю графіки та анімації, що надає їй візуальної привабливості. Гравці можуть досліджувати різноманітні рівні, стикаючись з різними типами ворогів та перешкод.

Важливим елементом геймплею є складний бойовий механізм, який включає в себе різноманітні комбінації атак, ухилянь та спеціальних здібностей. "Dead Cells" також використовує поведінкові дерева (Behaviour Tree) для управління штучним інтелектом ворогів, що робить їхню поведінку більш адаптивною та непередбачуваною. Гра здобула численні нагороди та високу оцінку критиків за інноваційний підхід та захоплюючий геймплей, ставши однією з найвизначніших у своєму жанрі.

Рисунок 1.1. – Банер гри «Dead Cells»

Рисунок 1.2. – Приклад геймплею гри «Dead Cells»

1.3.2 Аналіз гри The Binding of Isaac

"The Binding of Isaac" – це культова гра в жанрі Roguelike, створена Едмундом Макмілленом та випущена в 2011 році [11]. Гра натхненна біблійною історією про жертвоприношення Ісаака, що відображено в її сюжеті та атмосфері. Основні особливості "The Binding of Isaac" включають процедурну генерацію рівнів, що дозволяє створювати нові виклики при кожному проходженні, а також велику кількість предметів та їх комбінацій, які впливають на геймплей. Це робить кожну сесію гри унікальною для гравців.

Гра також відзначається високим рівнем складності, що вимагає від гравців стратегічного мислення та швидких реакцій. "The Binding of Isaac" стала однією з найбільш впливових у жанрі Roguelike, встановивши стандарти для багатьох подальших ігор. Вона отримала позитивні відгуки критиків за свою глибину, інноваційний геймплей та атмосферу, що поєднує елементи жаху та чорного гумору.

Рисунок 1.3. – Банер гри «The binding of Isaac»

Рисунок 1.4. – Приклад геймплею гри «The binding of Isaac»

1.3.3 Аналіз гри Pixel Dungeon

"Pixel Dungeon" – це мобільна гра в жанрі Roguelike, розроблена незалежним розробником Oleg Dolya та випущена в 2013 році [12]. Гра відрізняється класичним підходом до жанру з процедурною генерацією рівнів, випадковими предметами та ворогами, а також перманентною смертю персонажа. Графічний стиль гри – це проста піксельна графіка, яка не лише додає їй ретро-шарму, але й дозволяє легко запускати її на різних мобільних пристроях.

Популярність "Pixel Dungeon" серед гравців пояснюється її простотою, доступністю та захоплюючим геймплеєм. Незважаючи на простоту графіки,

гра пропонує глибокий ігровий досвід завдяки добре продуманій механіці та високій варіативності. Кожне проходження є унікальним завдяки випадковим елементам, що додає грі додатковий рівень виклику та інтриги.

Рисунок 1.5. – Банер гри «Pixel Dungeon»

Рисунок 1.6. –Приклад геймплею гри «Pixel Dungeon»

Для порівняння проєкту з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Дипломний проєкт	Dead Cells	The Binding of Isaac	Pixel Dungeon	Пояснення
Процедурна генерація рівнів	Так	Так	Так	Так	Усі аналоги використовують процедурну генерацію рівнів
Використання Behaviour Tree	Так	Так	Ні	Ні	Behaviour Tree використовується для AI ворогів в Dead Cells та дипломному проєкті
Перманентна смерть персонажа	Так	Так	Так	Так	Всі ігри жанру Roguelike використовують цю особливість
Гнучкість та адаптація бойового механізму під ситуацію	Так	Так	Ні	Ні	Dead Cells має гнучкий бойовий механізм, інші аналоги мають простіший лінійний підхід
Різноманітність предметів	Так	Так	Так	Так	Усі аналоги мають великий асортимент предметів
Shooter стрільба	Так	Так	Так	Ні	Pixel Dungeon не має механіки стрільби
Платформа	Мобільні пристрої	ПК, Консолі, Мобільні	ПК, Консолі	Мобільні пристрої	Дипломний проєкт зосереджений на мобільних платформах

1.3.4 Аналіз відомих алгоритмічних та технічних рішень

Процедурна генерація рівнів є одним із ключових компонентів рогалік-ігор, забезпечуючи унікальний ігровий досвід при кожному проходженні. Розглянемо найпоширеніші алгоритми та їх порівняльні характеристики.

1.3.5 Аналіз алгоритму Binary Space Partitioning (BSP)

BSP — алгоритм, який рекурсивно ділить прямокутний простір на менші прямокутні області [13]. Його основні переваги:

– гарантує створення зв'язаних, структурованих просторів;
– забезпечує природний поділ на кімнати різних розмірів;
– низька обчислювальна складність (o(n log n)).

Недоліки включають:

– створення надто прямолінійних, геометрично правильних приміщень;
– обмеженість у генерації органічних структур;
– схильність до створення подібних макроструктур.

1.3.6 Аналіз алгоритму Cellular Automata

Метод, заснований на імітації життєвого циклу клітин за визначеними правилами [14]:

– чудово підходить для створення органічних, нерегулярних приміщень (печери, природні формації);
– створює візуально цікаві, непередбачувані простори;
– простий у реалізації та модифікації.

Обмеження:

– складно контролювати структуру та зв'язність;
– може створювати ізольовані області без проходів;
– непередбачуваність кінцевого результату.

1.3.7 Аналіз алгоритму Drunkard Walk

Алгоритм симулює випадковий рух ("п'яну ходу") [15], створюючи шлях:

– генерує органічні, звивисті шляхи та тунелі;
– простий у реалізації;
– добре підходить для створення лабіринтів.

Недоліки:

– не гарантує структурованість простору;
– може створювати надмірно звивисті, неприродні тунелі;
– часто потребує додаткової обробки для забезпечення грального балансу.

1.3.8 Аналіз алгоритму Random Room Placement

Підхід полягає у випадковому розміщенні кімнат у просторі з наступним з'єднанням [16]:

– забезпечує високу різноманітність макроструктур;
– простий у реалізації;
– гнучкий у налаштуванні розподілу кімнат.

Слабкі сторони:

– може створювати перекриття або нелогічні з'єднання;
– потребує додаткових алгоритмів для з'єднання кімнат;
– часто створює надто розріджені або щільні структури.

Порівняльний аналіз та обґрунтування вибору

При порівнянні алгоритмів ключовими критеріями були: контрольованість результату, органічність створюваних структур, обчислювальна ефективність та гральний баланс. Результати порівняння наведено в Таблиці 1.2.

Таблиця 1.2 – Порівняння алгоритмів процедурної генерації рівнів

Алгоритм	Контрольованість	Органічність	Обчислювальна ефективність	Ігровий баланс
BSP	Висока	Низька	Висока	Середній
Cellular Automata	Низька	Висока	Середня	Низький
Drunkard Walk	Середня	Висока	Висока	Середній
Random Room Placement	Середня	Низька	Висока	Низький
Комбінований підхід	Висока	Висока	Середня	Високий

Для розробки було обрано комбінований підхід, який включає:

– BSP для створення основної структури рівня та розподілу простору на кімнати;

– Cellular Automata для додавання органічності та деталізації окремих просторів;

– Drunkard Walk для створення тунелів між кімнатами;

– кастомну систему тегування просторів на трьох рівнях (глобальний, макро, мікро) для забезпечення контролю над генерацією та грального балансу.

Такий підхід дозволяє поєднати переваги різних алгоритмів, компенсуючи їхні недоліки. Трирівнева система тегів є оригінальним рішенням, що дозволяє більш тонко контролювати розміщення ігрових елементів, забезпечуючи як різноманітність, так і ігровий баланс.

При виборі архітектури ШІ для ворогів ключовими критеріями були: модульність, масштабованість, продуктивність та керованість поведінки. Результати порівняння наведено в Таблиці 1.3.

Таблиця 1.3 – Порівняння алгоритмів ШІ для ворогів

Алгоритм	Модульність	Масштабованість	Продуктивність	Керованість
FSM [17]	Низька	Низька	Висока	Середня
Behaviour Trees [7]	Висока	Висока	Середня	Висока
Utility-Based AI [18]	Середня	Середня	Низька	Середня
Neural Networks [19]	Низька	Висока	Низька	Низька
Behaviour Tree + Blackboard	Висока	Висока	Середня	Висока

Для реалізації ШІ ворогів було обрано комбінацію Behaviour Tree з Blackboard Architecture з наступних причин:

– Behaviour Tree забезпечує модульність, ієрархічну структуру поведінки та легкість розширення, що критично для створення різноманітних типів ворогів;

– Blackboard Architecture доповнює дерево поведінки, надаючи механізм обміну даними між різними вузлами та сутностями, що дозволяє реалізувати:

  – спільне використання інформації про середовище;

  – координацію дій між групами ворогів;

– персоналізовану поведінку окремих сутностей.

Трирівнева система контекстів (кімната, група, особистий) є оригінальним розширенням стандартної Blackboard Architecture, що забезпечує:

– оптимізацію обчислень шляхом обмеження доступу до інформації;
– природну ієрархію знань від глобальних до локальних;
– можливість реалізації як групової, так і індивідуальної тактики.

Така комбінація забезпечує оптимальний баланс між гнучкістю, продуктивністю та керованістю поведінки, дозволяючи створювати складні патерни поведінки ворогів без надмірного ускладнення системи.

1.4 Аналіз та моделювання бізнес-процесів

Для моделювання бізнес-процесу генерації рівня використовується BPMN модель. BPMN модель процесу генерації рівня знаходиться у графічному матеріалі, креслення 2.

Опис моделі бізнес-процесу генерації рівня:

– користувач починає гру;
– генератор починає створювати рівень;
– генератор розділює простір на кімнати;
– генератор змінює кожну кімнату;
– генератор генерує тунелі;
– гра розставляє об’єкти на сцені;
– гра завантажує всі інші об’єкти + інфраструктуру.

Висновки до розділу

У розділі сформульовано завдання розробки мобільної гри жанру Shooter з елементами Roguelike та проведено комплексний аналіз предметної області. Визначено, що сектор мобільних ігор активно розвивається, проте має

специфічні обмеження, пов'язані з продуктивністю пристроїв та особливостями сенсорного управління.

Проаналізовано ключові аналоги ("Dead Cells", "The Binding of Isaac", "Pixel Dungeon"), що дозволило визначити конкурентні переваги та недоліки існуючих рішень. Виявлено, що поєднання процедурної генерації рівнів з використанням Behaviour Tree для AI ворогів та механік Shooter створює перспективну нішу для розробки.

Досліджено та порівняно алгоритми процедурної генерації рівнів (BSP, Cellular Automata, Drunkard Walk, Random Room Placement) та підходи до реалізації штучного інтелекту (FSM, Behaviour Trees, Utility-Based AI, Neural Networks). На основі порівняльного аналізу обґрунтовано вибір комбінованого підходу для генерації рівнів та системи Behaviour Tree + Blackboard з трирівневою структурою контекстів для реалізації AI.

Проведене дослідження створює теоретичне підґрунтя для розробки мобільної гри, що поєднує інноваційні алгоритмічні рішення з ефективним використанням обмежених ресурсів мобільних платформ.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання, що відображає ключових акторів системи та їхню взаємодію з основними функціональними можливостями програмного забезпечення, знаходиться у графічному матеріалі, креслення 1.

В таблицях 2.1-2.12 докладно представлено повний і всебічний опис усіх варіантів використання розробленого програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Назва варіанту використання	Старт нової гри
Ідентифікатор варіанту використання	UC-01
Цілі	Розпочати новий ігровий процес
Актори	Гравець
Тригер	Гравець обирає "Нова гра" в головному меню
Передумови	Гра завантажена, головне меню доступне
Послідовність подій	Гравець запускає гру з головного меню, система ініціює процедурну генерацію рівня
Розширення	-
Постумова	Гра починається з новим рівнем, гравець може керувати персонажем

Таблиця 2.2 – Варіант використання UC-02

Назва варіанту використання	Вибір персонажу
Ідентифікатор варіанту використання	UC-02
Цілі	Дозволити гравцю вибрати персонажа для гри
Актори	Гравець
Тригер	Гравець натискає "Вибір персонажа" в меню старту нової гри
Передумови	Гра завантажена, гравець обрав старт нової гри
Послідовність подій	Гравець переглядає доступних персонажів і вибирає одного для гри
Розширення	Якщо персонаж заблокований, відображається вікно з вимогами для його розблокування
Постумова	Персонажа обрано, гра розпочинається з обраним персонажем

Таблиця 2.3 – Варіант використання UC-03

Назва варіанту використання	Бій з ворогом
Ідентифікатор варіанту використання	UC-03
Цілі	Забезпечити механіку бою між персонажем гравця і ворожими персонажами
Актори	Гравець, ворожі персонажі
Тригер	Гравець наближається до ворожого персонажа на ігровому полі
Передумови	Гравець керує персонажем, який зустрічає ворога на ігровому рівні

Продовження таблиці 2.3

Послідовність подій	1. Гравець зустрічає ворога. 2. Гравець ініціює атаку за допомогою ігрових контролів. 3. Ворог реагує в реальному часі, використовуючи свої атакуючі або оборонні дії. 4. Бій триває, поки показник здоров’я одного з учасників не досягне 0 або поки не буде виконана одна з додаткових умов передчасного завершення бою. 5. Ігрова система розраховує результати бою згідно з ігровими правилами. 6. Відображення результатів бою та зміни стану персонажів.
Розширення	Ворог може викликати підкріплення, якщо така поведінка передбачена його алгоритмом.
Постумова	Бій завершено. Якщо гравець переміг, він отримує досвід та/або предмети з переможеного ворога. Якщо персонаж гравця програв, гра може завершитись або продовжитись з певними наслідками.

Таблиця 2.4 – Варіант використання UC-04

Назва варіанту використання	Смерть гравця
Ідентифікатор варіанту використання	UC-04
Цілі	Обробити ситуацію смерті персонажа гравця та визначити наслідки цієї події для гри
Актори	Гравець
Тригер	Здоров'я персонажа гравця знижується до нуля
Передумови	Гравець активно бере участь в ігровому процесі, зокрема у боях з ворогами

Продовження таблиці 2.4

Послідовність подій	1. Здоров'я персонажа гравця досягає нуля внаслідок атак ворогів або інших внутрішньо ігрових чинників. 2. Ігрова система активує сценарій смерті персонажа. 3. Відображення анімації та ефектів смерті персонажа. 4. Вивід на екран повідомлення про смерть та можливі подальші опції гравця (наприклад, перезапуск рівня або вихід до головного меню). 5. Залежно від налаштувань гри, можливо збереження деяких досягнень або прогресу.
Розширення	1. Якщо в грі передбачена можливість відродження, гравець може використати предмети відродження або спеціальні здібності для повернення в гру. 2. Якщо гравець має захисні предмети, які автоматично відновлюють здоров'я при критичному стані, вони активуються.
Постумова	Гра завершується з смертю персонажа, або персонаж відроджується згідно з правилами гри. Всі зміни в ігровому стані фіксуються.

Таблиця 2.5 – Варіант використання UC-05

Назва варіанту використання	Збір предметів
Ідентифікатор варіанту використання	UC-05
Цілі	Дозволити гравцю збирати предмети, які знаходяться на ігровому полі
Актори	Гравець
Тригер	Гравець наближається до предмету на ігровому полі
Передумови	Предмети розміщені на ігровому полі генерації рівнів
Послідовність подій	Гравець підбирає предмет
Розширення	-
Постумова	Предмет підібрано та використано, зміни у стані гри зафіксовано.

Таблиця 2.6 – Варіант використання UC-06

Назва варіанту використання	Взаємодія з внутрішньоігровими елементами з особливою взаємодією (двері, сундуки)
Ідентифікатор варіанту використання	UC-06
Цілі	Дозволити гравцю взаємодіяти з деякими внутрішньоігровими об'єктами (двері, сундуки) шляхом нанесення урону
Актори	Гравець
Тригер	Гравець бажає відкрити двері або сундук, знаходячи їх на рівні
Передумови	Гравець знаходиться біля об'єкта з взаємодією, має зброю або інструмент для нанесення урону
Послідовність подій	1. Гравець виявляє двері або сундук на ігровому полі. 2. Гравець використовує атаку або засіб для нанесення урону по об'єкту. 3. Ігрова система розраховує урон та перевіряє, чи достатньо урону для відкриття або руйнування. 4. Якщо урон достатній, двері відчиняються або сундук ламається і випускає вміст.
Розширення	-
Постумова	Об'єкт відкрито або зруйновано, елементи взаємодії на ігровому полі змінено, можливий доступ до нових ареалів або ресурсів.

Таблиця 2.7 – Варіант використання UC-07

Назва варіанту використання	Підвищення рівня персонажа
Ідентифікатор варіанту використання	UC-07
Цілі	Дозволити гравцю підвищувати рівень свого персонажа, покращуючи його характеристики та здібності
Актори	Гравець

Продовження таблиці 2.7

Тригер	Гравець накопичив достатньо досвіду для переходу на наступний рівень
Передумови	Гравець отримав досвід з вбивства ворогів або шляхом взаємодії з внутрішньоігровими об’єктами на рівні
Послідовність подій	1. Гравець накопичує достатню кількість досвіду для підвищення рівня. 2. Ігрова система сповіщає гравця про можливість підвищення рівня. 3. Гравець переходить в вікно покращення, обирає одну з 3 опцій покращення 4. Підтвердження вибору та застосування покращень до персонажа. 5. Відображення оновлених характеристик та здібностей персонажа.
Розширення	1. Якщо доступні спеціальні бонуси за досягнення певного рівня, вони активуються.
Постумова	Персонаж гравця підвищений до наступного рівня з відповідними покращеннями характеристик та здібностей.

Таблиця 2.8 – Варіант використання UC-08

Назва варіанту використання	Застосування здібностей
Ідентифікатор варіанту використання	UC-8
Цілі	Дозволити гравцю використовувати здібності персонажа для впливу на ігровий процес
Актори	Гравець
Тригер	Гравець обирає застосувати певну здібність у контексті ігрової ситуації
Передумови	Гравець керує персонажем, який має доступні здібності; здібності доступні для використання (мають відновлений кулдаун, достатньо ресурсів, тощо)

Продовження таблиці 2.8

Послідовність подій	1. Гравець активує здібність за допомогою ігрових контролів. 3. Ігрова система перевіряє чи виконані умови для використання здібності. 4. Здібність застосовується, її ефект впливає на ігровий процес. 5. Ігрова система відображає ефекти здібності та оновлює відповідні параметри ігрового процесу. 6. Час перезарядки здібності оновлюється після використання здібності.
Розширення	1. Якщо здібність не може бути застосована (наприклад, не вистачає мани або персонаж під контролем), система відображає відповідне повідомлення. 2. Якщо здібність має спеціальні вимоги чи умови, вони враховуються перед її застосуванням.
Постумова	Здібність застосована, вплинувши на ігровий стан; відповідні параметри та стани персонажа оновлені.

Таблиця 2.9 – Варіант використання UC-09

Назва варіанту використання	Завершення гри
Ідентифікатор варіанту використання	UC-09
Цілі	Обробити завершення гри, включаючи збереження результатів, підведення підсумків, та відображення фінальної сцени або статистики
Актори	Гравець
Тригер	Гравець досягає кінця гри, наприклад, перемога над остаточним босом або завершення останнього рівня; або персонаж гравця гине і не може відродитися
Передумови	Гравець активно участвує в ігровому процесі та досягає логічного завершення або ендгейму

Продовження таблиці 2.9

Послідовність подій	1. Гравець досягає критичної точки гри (наприклад проходить останню кімнату на рівні). 2. Ігрова подія спричиняє запуск сценарію завершення гри. 3. Відображення фінальної анімації або кінцевих кредитів. 4. Відображення загальної статистики гри та результатів гравця. 5. Запропонувати гравцю переглянути свої досягнення або повернутися до головного меню.
Розширення	-
Постумова	Гра завершена; гравець отримує відображення результатів та можливість продивитися свої досягнення чи розпочати гру заново.

Таблиця 2.10 – Варіант використання UC-10

Назва варіанту використання	Вихід з гри
Ідентифікатор варіанту використання	UC-10
Цілі	Надати гравцю можливість безпечно завершити ігрову сесію та вийти з гри
Актори	Гравець
Тригер	Гравець обирає опцію виходу з гри з головного меню або під час гри
Передумови	Гра активна, доступне головне меню або пауза-меню
Послідовність подій	1. Гравець вибирає опцію "Вихід з гри". 2. Ігрова система закривається, гравець повертається на робочий стіл або платформу запуску ігор.
Розширення	
Постумова	Гра завершена та закрита, гравець повністю вийшов з ігрового середовища.

Таблиця 2.11 – Варіант використання UC-11

Назва варіанту використання	Поставити гру на павзу
Ідентифікатор варіанту використання	UC-11
Цілі	Надати гравцю можливість призупинити ігрову сесію
Актори	Гравець
Тригер	Гравець бажає зупинити ігровий процес
Передумови	Гра активна
Послідовність подій	Гравець натискає на кнопку павзи
Розширення	-
Постумова	Ігровий процес призупинено, відображається меню павзи

Таблиця 2.12 – Варіант використання UC-12

Назва варіанту використання	Відновлення гри
Ідентифікатор варіанту використання	UC-12
Цілі	Надати гравцю можливість відновити призупинену ігрову сесію
Актори	Гравець
Тригер	Гравець бажає продовжити гру
Передумови	Гра знаходиться на павзі
Послідовність подій	Гравець натискає на кнопку відновлення гри
Розширення	-
Постумова	Гра відновлена, продовжується з моменту коли була увімкнена павза

2.2 Розроблення функціональних вимог

В таблиці 2.13 наведено загальну модель вимог, а в таблиці 2.14 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.1.

Таблиця 2.13 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Меню та навігація	FR-1	Низький	Низький
2	Процедурна генерація рівня	FR-2	Високий	Високий
3	Управління гравцем	FR-3	Високий	Високий
4	Механіки стрільби	FR-4	Високий	Високий
5	Штучний інтелект ворогів	FR-5	Середній	Низький
6	Збір предметів та покращення	FR-6	Середній	Середній
7	Система здоров’я та шкоди	FR-7	Високий	Низький
8	Пауза та налаштування	FR-8	Низький	Низький

Таблиця 2.14 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Меню та навігація: Гравець бачить головне меню з кнопками «Нова гра», «Продовжити гру», «Налаштування», «Вихід з гри»
FR-2	Процедурна генерація рівня: Після старту нової сесії система генерує унікальний рівень із кімнатами, коридорами, точками спавну ворогів та розташуванням предметів.

Продовження таблиці 2.14

ID вимоги	Назва та опис
FR-3	Управління гравцем: Користувач керує персонажем через сенсорний джойстик або кнопки руху в чотирьох напрямках; кнопка «Вогонь» для стрільби; селектор зброї для перемикання; кнопки для стрибку та провалювання крізь платформи; кнопки для використання активних вмінь
FR-4	Механіки стрільби: Підтримка кількох типів зброї з параметрами швидкості стрільби, перезарядження, розкид; індикатор патронів і перезарядження.
FR-5	Штучний інтелект ворогів: Вороги мають складний патерн поведінки. Вони можуть патрулювати, виявляюти гравця та переслідувати/атакувати; Також можуть виконувати складні патерни атак (наприклад комбінації атак) або інші тактичні та стратегічні операції(телепортації, групування тощо)
FR-6	Збір предметів та покращення: Підбір аптечок, накопичення досвіду за вбивства ворогів із можливістю відкривати навички або покращення.
FR-7	Система здоров’я та шкоди: Відображення HP гравця; обробка нанесення шкоди; при втраті всіх HP – виконання дій, пов’язаних зі смертю ворога/гравця
FR-8	Пауза та налаштування: Кнопка «Пауза» відкриває меню з поверненням у головне меню або виходом із гри.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8
UC-01	+	+						
UC-02								
UC-03			+	+	+		+	
UC-04							+	
UC-05			+			+		
UC-06			+					
UC-07			+			+		
UC-08			+					
UC-09								
UC-10	+							
UC-11			+					+
UC-12	+		+					+

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Нефункціональні вимоги до програмного забезпечення наведені в таблиці 2.15

Таблиця 2.15 – Нефункціональні вимоги до програмного забезпечення

№	Категорія	Показник	Значення / Умова
1	Продуктивність	Частота кадрів (FPS)	≥ 60 при роздільній здатності 1080×1920 на Android 8+ / iOS 12+
2	Продуктивність	Час завантаження рівня	≤ 5 секунд
3	Продуктивність	Плавність ігрового процесу	Без фризів, стабільний геймплей
4	Сумісність	Підтримка ОС	Android 8.0+ та iOS 12.0+
5	Сумісність	Адаптивність інтерфейсу	Підтримка екранів зі співвідношенням 16:9–19.5:9
6	Надійність	MTBF (mean time between failures) [20]	≥ 100 годин активної гри без аварійних зупинок
7	Безпека	Захист локальних даних	Шифрування (наприклад, AES-256)
8	Зручність користування	Мінімальний розмір елементів керування	Не менше ніж 44×44 dp [21]
9	Зручність користування	Інтерфейс користувача	Інтуїтивно зрозумілий

2.4 Аналіз системних вимог

Для коректної та плавної роботи мобільного ігрового застосунку були сформовані системні вимоги. Вони наведені в таблиці 2.16

Таблиця 2.16 – Системні вимоги до програмного забезпечення

№	Категорія	Підкатегорія/ Показник	Вимоги
1	Платформи та ОС	Android	Android 8.0 (API level 26) або новіші
2	Платформи та ОС	iOS	iOS 12.0 або новіші
3	Графічний API	Android	OpenGL ES 3.0 або Vulkan
4	Графічний API	iOS	Metal
5	Оперативна пам’ять	RAM	Мінімум 2 ГБ
6	Сховище	Вільне місце	Приблизно 200 МБ для встанов
7	Додаткові вимоги	Сенсорний екран	Підтримка multi-touch

2.5 Аналіз економічних показників програмного забезпечення

2.5.1 Вихідні дані для розрахунку

Для розрахунку економічних показників розробки рогалік-гри використовуємо дані наведені в таблиці 2.17.

Таблиця 2.17 – Вихідні дані розрахунку економічних показників розробки

№	Показник	Значення
1	Кількість виконавців	1 розробник
2	Середня заробітна плата	550 USD / місяць
3	Тривалість робочого дня	8 годин
4	Кількість робочих днів у місяці	22 дні
5	Вартість програмного забезпечення	Unity Personal [22]
6	Амортизація обладнання	Ноутбук: 1200 USD
7	Тестові пристрої	Власний смартфон

2.5.2 Розрахунок трудомісткості розробки

На основі аналізу функціональних вимог (FR-1 — FR-8) та варіантів використання (UC-01 — UC-12) розробку проєкту розділено на етапи. В таблиці 2.18 наведено відповідні етапи розробки програмного забезпечення.

Таблиця 2.18 – етапи розробки програмного забезпечення

Етап	Трудомісткість (людино-годин)
Аналіз вимог і проєктування	80
Розробка системи процедурної генерації рівнів	120
Розробка механік руху персонажа та стрільби	100
Розробка штучного інтелекту ворогів	140
Розробка системи покращень і предметів	60
Розробка користувацького інтерфейсу	40
Тестування та налагодження	80
Оптимізація та фінальне полірування	60
Загальна трудомісткість	680

Виходячи з 8-годинного робочого дня, на виконання проєкту потрібно приблизно 85 робочих днів або 3,9 місяця.

2.5.3 Розрахунок витрат на розробку

В таблиці 2.19 наведено сумарні витрати на розробку програмного забезпечення

Таблиця 2.19 – Сумарні витрати на розробку

Категорія витрат	Сума (USD)
Оплата праці	2 145
Програмне забезпечення	0
Загальні витрати	2 145

2.5.4 Оцінка економічної ефективності

Для оцінки потенційних доходів використаємо дані про середню прибутковість інді мобільних ігор жанру рогалік:

– середня ціна завантаження (freemium [23] модель з внутрішніми покупками): 0 USD;

– середній дохід з одного активного користувача (ARPU) [24] 0,10 USD на місяць;

– прогнозована кількість завантажень за перший рік: 30 000;

– прогнозована кількість активних користувачів за перший рік: 5 000;

– прогнозовані доходи за перший рік: 5 000 × 0,10 USD × 12 = 6 000; USD

Розрахунок окупності

– сумарні витрати на розробку: 2 509 USD;

– прогнозовані доходи за перший рік: 6 000 USD;

– очікуваний прибуток за перший рік: 6 000 - 2 145 = 3 855 USD;

– термін окупності: приблизно 5 місяців.

2.5.5 Висновки щодо економічної ефективності

Розробка рогалік-гри з визначеним функціоналом є економічно обґрунтованою та привабливою з точки зору інвестицій. Очікуваний термін окупності складає приблизно 5 місяців, що є суттєво кращим показником порівняно з середнім по ринку інді-ігор (8-12 місяців).

Використання безкоштовної ліцензії Unity Personal та залучення junior розробника дозволяє значно знизити витрати на розробку порівняно з комерційними проєктами. При цьому, завдяки багатому функціоналу безкоштовної версії Unity, якість кінцевого продукту залишається на конкурентоспроможному рівні.

Економічна ефективність проєкту може бути додатково підвищена через:

– впровадження монетизації через рекламу;
– розширення гри через регулярні оновлення;
– локалізація гри для різних регіональних ринків;

Низькі початкові інвестиції та потенційно швидка окупність роблять проєкт привабливим для розробки навіть при консервативних прогнозах щодо популярності гри.

2.6 Постановка завдання на розробку програмного забезпечення

Розробити однокористувацький застосунок мобільної гри в жанрі Shooter з елементами Roguelike.

Основні завдання:

– адаптивний ШІ ворогів з реакцією на гравця та інших ворогів;
– процедурна генерація рівнів при кожному завантаженні;
– рух персонажа (включно зі стрибками) за допомогою екранних джойстиків;
– різні види зброї з унікальними характеристиками;
– активні вміння персонажа;
– прокачка характеристик і відкриття нових умінь протягом сесії.

Висновки до розділу

У розділі проведено формалізацію вимог до програмного забезпечення рогалік-гри. Розроблено діаграму варіантів використання та детально описано

12 основних сценаріїв взаємодії користувача з системою, від старту гри до її завершення.

Сформульовано 8 ключових функціональних вимог (FR-1 — FR-8) з визначенням їх пріоритетів та рівнів ризику. Найвищий пріоритет надано процедурній генерації рівня, управлінню гравцем, механікам стрільби та системі здоров'я. Для відстеження зв'язків між варіантами використання та функціональними вимогами створено матрицю трасування.

Визначено нефункціональні вимоги щодо продуктивності (FPS ≥ 60), сумісності (Android 8.0+, iOS 12.0+), надійності (MTBF ≥ 100 годин), безпеки (шифрування даних) та зручності користування. Проаналізовано системні вимоги, включаючи вимоги до операційних систем, графічних API, оперативної пам'яті (≥ 2 ГБ) та простору для зберігання (≈ 200 МБ).

На основі аналізу сформульовано чітке завдання на розробку, що визначає основні функціональні блоки: штучний інтелект ворогів, процедурну генерацію рівнів, механіки руху персонажа, систему стрільби, активні вміння, покращення характеристик та взаємодію з об'єктами. Це забезпечує структурований підхід до подальшого проєктування та реалізації гри відповідно до поставлених цілей.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

У межах розробки програмного забезпечення для гри було обрано монолітну архітектуру як основу структурування коду. Монолітна архітектура передбачає створення єдиної кодової бази, де всі функціональні компоненти тісно інтегровані між собою та взаємодіють у рамках одного процесу. Такий підхід забезпечує тісну взаємодію між компонентами, що є критичним для ігрових застосунків, де швидкодія та ефективна комунікація між підсистемами вирішально впливають на якість кінцевого продукту.

У розробці гри використовується патерн впровадження залежностей (Dependency Injection, DI) через контейнер, що дозволяє ефективно керувати залежностями між компонентами системи. DI Container забезпечує центральне місце для конфігурації залежностей та їх життєвого циклу. Цей підхід має наступні переваги:

– полегшення модульного тестування через можливість підміни залежностей;
– зменшення зв'язності між компонентами;
– спрощення конфігурації системи;
– покращення можливості повторного використання компонентів;

У реалізації архітектури також використано патерн Bootstrapper на двох рівнях: глобальному та сценовому. Глобальний Bootstrapper відповідає за ініціалізацію всієї гри, включаючи конфігурацію DI контейнера та запуск глобальної машини станів, у той час як кожна сцена має власний Bootstrapper, який відповідає за ініціалізацію лише локальних залежностей цієї сцени. Для управління потоком гри застосовано ієрархічну машину станів, де глобальна State Machine відповідає за основні режими (меню, геймплей, вихід), а кожна з них містить власну підмашину станів, яка керує вже більш конкретними внутрішніми станами (наприклад, екран налаштувань або пауза в межах

меню). Така ієрархія забезпечує чіткий контроль над переходами між режимами гри та спрощує масштабування логіки станів. C4 модель ієрархії машини станів та Bootstrapper-ів знаходиться у графічному матеріалі, креслення 3.

Для організації користувацького інтерфейсу застосовано архітектурний патерн Model-View-Presenter (MVP) [25]. MVP є варіацією класичного MVC, але з більш чітким розділенням відповідальності між компонентами, що особливо важливо в контексті ігрових застосунків. Основні компоненти MVP у нашій реалізації:

– Model — відповідає за дані та бізнес-логіку;
– View — відповідає за відображення даних користувачеві та передачу команд користувача до Presenter;
– Presenter — посередник між Model і View, обробляє взаємодію користувача та відповідає за оновлення View.

У розробці програмного забезпечення застосовується принцип "композиція замість успадкування" (Composition over Inheritance) [26]. Цей підхід передбачає створення складних об'єктів шляхом комбінування простіших компонентів замість побудови глибоких ієрархій наслідування.

Компонентна модель реалізована таким чином, що кожен ігровий об'єкт складається з набору компонентів, які визначають його поведінку та характеристики. Це дозволяє:

– підвищити гнучкість системи;
– зменшити дублювання коду;
– спростити додавання нової функціональності;
– покращити тестованість окремих компонентів.

Для створення об'єктів використовується патерн Factory (Фабрика) [27]. Цей патерн інкапсулює логіку створення об'єктів, надаючи інтерфейс для створення об'єктів певного типу без прямої залежності від конкретних класів.

У контексті гри фабрики використовуються для створення різноманітних ігрових сутностей: персонажів, предметів, елементів оточення тощо.

Для оптимізації роботи з часто створюваними та знищуваними об'єктами впроваджено патерн Object Pool (Пул об'єктів) [28]. Цей патерн передбачає попереднє створення та збереження набору об'єктів, готових до використання, замість створення нових об'єктів на вимогу. Після використання об'єкти повертаються в пул замість знищення. Таким чином зменшується навантаження на систему керування пам'яттю та підвищується продуктивність.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

При виборі ігрового рушія розглядалося два основних варіанти: Unity [29] та Unreal Engine [30]. Обидва рушії є потужними інструментами для розробки ігор, проте для даного проєкту перевагу було надано Unity з наступних причин.

Unity має значно нижчий поріг входу порівняно з Unreal Engine, що дозволяє швидше розпочати розробку та прототипування. Мова програмування C#, яка використовується в Unity, є більш простою для освоєння та використання, ніж C++ в Unreal Engine, що прискорює процес розробки і тестування нових функцій. Це особливо важливо для проєктів, де необхідно швидко ітерувати дизайн гри та механіки.

Unity надає кращу підтримку 2D розробки, що є ключовим для рогалік-гри, яка розробляється в рамках даного проєкту [31]. Хоча Unreal Engine також має інструменти для 2D розробки, вони часто вважаються менш зрілими та інтегрованими порівняно з інструментами Unity.

Екосистема Unity включає Asset Store з великою кількістю готових рішень, інструментів та ресурсів, які можуть бути інтегровані в проєкт. Це дозволяє прискорити розробку та зосередитися на унікальних аспектах гри, а не витрачати час на створення базової функціональності.

Unity забезпечує простіше кросплатформне розгортання, що дозволяє з мінімальними змінами випускати гру на різних платформах (Windows, MacOS, Linux, мобільні платформи). Хоча Unreal Engine також підтримує кросплатформну розробку, Unity часто демонструє кращу продуктивність на менш потужних пристроях [32].

Для реалізації патерну впровадження залежностей (DI) у Unity було розглянуто два популярних рішення: Zenject [33] та VContainer [34]. Хоча обидва фреймворки вирішують аналогічні завдання, Zenject було обрано з наступних міркувань.

Zenject має більш розвинену та зрілу екосистему. Фреймворк існує довше, має велику спільноту та численні готові рішення для типових проблем у розробці ігор на Unity. Це забезпечує кращу документацію, більше прикладів використання та більший об'єм знань у спільноті розробників.

Zenject пропонує більш гнучку систему конфігурації залежностей через Installers, що дозволяє легко розділити конфігурацію за функціональними модулями гри. Це особливо корисно для проєктів середнього та великого розміру, де кількість залежностей може бути значною.

Zenject також надає механізми для керування життєвим циклом об'єктів та їх знищенням, що є критичним для ігрових застосунків, де ресурси часто потрібно звільняти для оптимізації продуктивності. VContainer, хоча і є більш легковагим, не пропонує такого ж рівня контролю над життєвим циклом.

Слід зазначити, що VContainer демонструє кращу продуктивність у деяких сценаріях, особливо при створенні великої кількості об'єктів [35]. Проте, перевага в продуктивності не є критичною для даного проєкту, оскільки більшість DI-операцій відбуваються під час завантаження сцен, а не в критичному циклі оновлення гри.

При проєктуванні архітектури користувацького інтерфейсу розглядалися два основних патерни: Model-View-Presenter (MVP) та Model-View-ViewModel (MVVM). Вибір було зроблено на користь MVP з наступних причин.

MVP забезпечує чіткіше розділення відповідальності між компонентами, що спрощує розробку та тестування. В контексті Unity та ігрової розробки, де структура часто дуже залежить від ігрового циклу та подій, MVP дозволяє краще ізолювати логіку UI від основної логіки гри [36].

Патерн MVP легше інтегрується з існуючою архітектурою Unity, оскільки View може бути реалізований як MonoBehaviour, що природно взаємодіє з системою подій Unity. MVVM, хоча і потужний патерн, часто вимагає додаткової інфраструктури для реалізації двостороннього зв'язування даних, що може бути надмірним у контексті ігрової розробки.

MVP також забезпечує кращу продуктивність порівняно з MVVM у сценаріях, де необхідна висока частота оновлення інтерфейсу, що є типовим для ігор. Це досягається за рахунок прямого управління оновленнями View з боку Presenter, без необхідності підтримувати систему подій для зв'язування даних [37].

Тестованість є ще однією перевагою MVP, особливо в контексті Unity, де юніт-тестування має певні обмеження. MVP дозволяє ізолювати логіку Presenter від MonoBehaviour-компонентів, що робить її більш доступною для стандартних методів юніт-тестування [38].

Для управління часто створюваними та знищуваними об'єктами було розглянуто два підходи: використання пулу об'єктів (Object Pool) та стандартне створення/знищення об'єктів. Пул об'єктів був обраний з наступних міркувань.

Пул об'єктів значно зменшує навантаження на збірник сміття (garbage collector), що є критичним для підтримки стабільної продуктивності гри. Часте створення та знищення об'єктів може призводити до "спайків" у роботі збірника сміття, що викликає помітні затримки під час гри [39].

В контексті рогалік-гри, де відбувається інтенсивне створення та знищення об'єктів (снаряди, вороги, ефекти), пул об'єктів дозволяє суттєво оптимізувати використання пам'яті та CPU. Дослідження показують, що при

правильній реалізації пул об'єктів може значно прискорити операції створення об'єктів порівняно зі стандартним Instantiate/Destroy в Unity [40].

Пул об'єктів також забезпечує більш передбачуване використання пам'яті, зменшуючи ризик фрагментації пам'яті, що є особливо важливим для мобільних платформ з обмеженими ресурсами [41].

Реалізація пулу об'єктів добре інтегрується з іншими архітектурними рішеннями проєкту, такими як Factory Pattern та DI Container, дозволяючи створити гнучку та розширювану систему управління об'єктами.

3.3 Конструювання програмного забезпечення

Система генерації ігрових рівнів є ключовим компонентом даної роботи. Вона реалізована з використанням кількох алгоритмів та підходів:

Binary Space Partitioning (BSP) використовується для розбиття простору рівня на менші області (кімнати). Алгоритм рекурсивно ділить простір на дві частини за допомогою розділяючих ліній, створюючи ієрархічну структуру просторів. Це дозволяє генерувати різноманітні компонування кімнат з природним розподілом простору.

Для створення органічних структур (печер, нерівних стін, тощо) використовується метод клітинних автоматів (Cellular Automata). Цей підхід симулює еволюцію дискретної сітки клітин за певними правилами, які визначають стан кожної клітини на основі стану її сусідів. За допомогою ітеративного застосування цих правил генеруються природні на вигляд структури з нерегулярними формами.

Для створення тунелів між кімнатами використовується алгоритм "п'яної ходи" (Drunkard Walk). Цей алгоритм симулює випадковий рух від однієї точки до іншої, створюючи при цьому прохід. Випадковість руху забезпечує природні, нелінійні шляхи між локаціями, що підвищує різноманітність ігрового простору.

Для ефективного управління компонентами рівня розроблено трирівневу систему тегування простору:

– глобальні теги — застосовуються до цілих структурних одиниць (кімнат, тунелів) та визначають їх загальні характеристики та призначення;

– макро-теги — визначають великі області в межах структурних одиниць (стіни, підлога, стеля тощо);

– мікро-теги — визначають конкретні точки інтересу (спавн ворогів, гравця, розміщення сундуків, дверей тощо).

Така система забезпечує ієрархічний контроль над генерацією та розміщенням елементів у просторі гри.

С4 діаграма генерації рівнів знаходиться у графічному матеріалі, креслення 4. На рисунку 3.1 зображено результат генерації рівня, представлений з використанням PNG Exporter інструменту.

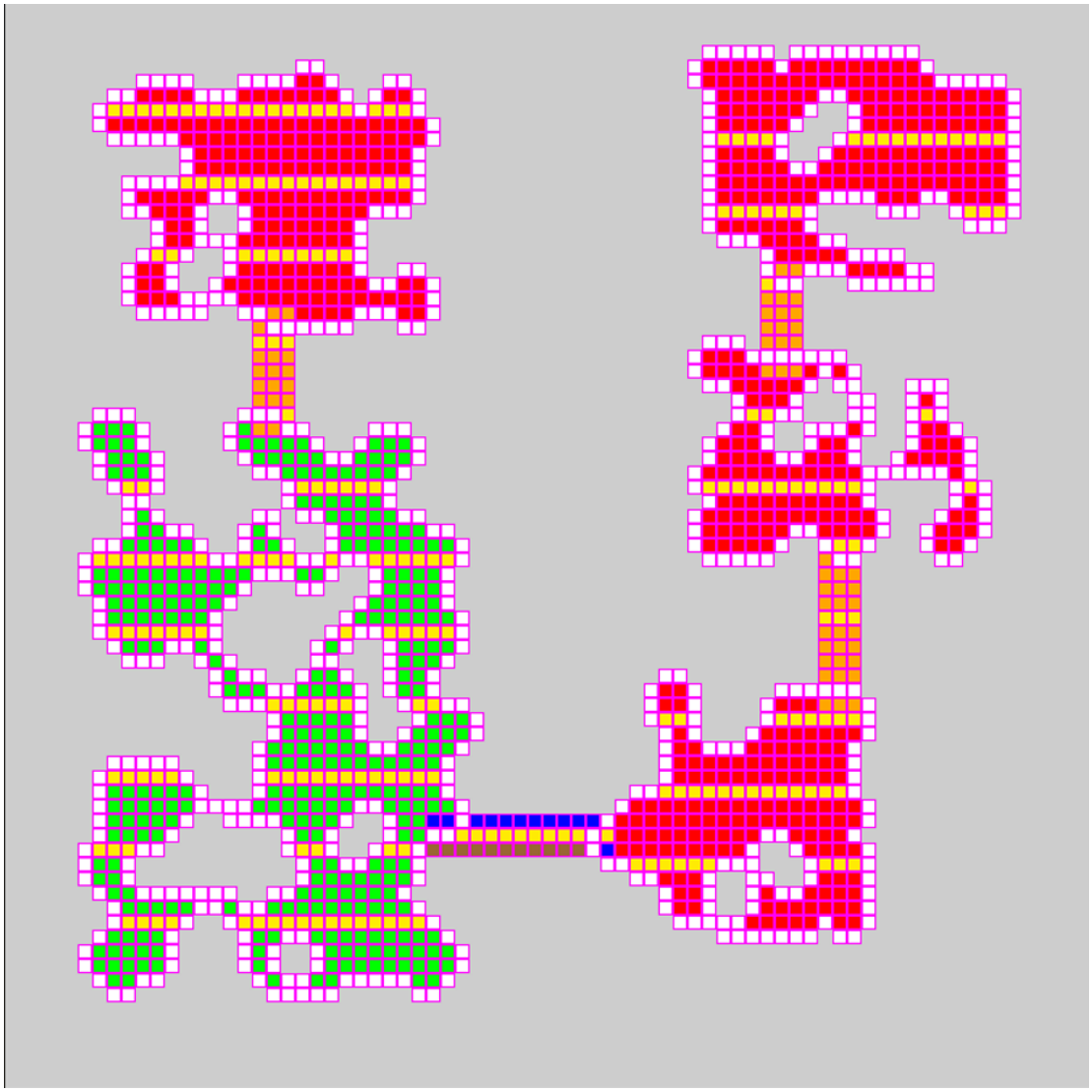


Рисунок 3.1 – Результат генерації рівня, представлений з використанням інструменту PNG Exporter

Для реалізації штучного інтелекту ворогів використовується комбінація патернів Behaviour Tree (Дерево поведінки) та Blackboard (Дошка). Behaviour Tree представляє ієрархічну структуру можливих дій, які ШІ може виконувати, з чіткими правилами переходу між ними. Blackboard виступає як сховище даних, до якого мають доступ всі вузли дерева поведінки, забезпечуючи обмін інформацією між різними частинами ШІ.

Реалізація контекстів знань включає три рівні:

– знання на рівні кімнати — інформація, доступна всім сутностям у межах однієї кімнати (наприклад, положення гравця, стан об'єктів середовища);

– знання на рівні групи — інформація, яка ділиться між певною групою сутностей (наприклад, тактична інформація для групи ворогів);

– персональні знання — інформація, специфічна для конкретної сутності (наприклад, рівень здоров'я, поточний стан).

Така багаторівнева система знань дозволяє створювати складні сценарії поведінки ворогів з елементами колективної та індивідуальної тактики. C4 діаграма архітектури ШІ ворогів знаходиться у графічному матеріалі, креслення 5.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.15.

Таблиця 3.1 – Опис утиліт, бібліотек та іншого стороннього програмного забезпечення

№ п/п	Назва утиліти	Опис застосування
1	Unity	Багатоплатформовий ігровий рушій, що використовується як основне середовище розробки гри. Забезпечує базову функціональність рендерингу, фізики, навігації та інших ключових аспектів ігрової розробки.

Продовження таблиці 3.1.

№ п/п	Назва утиліти	Опис застосування
2	Rider	Інтегроване середовище розробки (IDE) від JetBrains, оптимізоване для розробки на C# та Unity. Використовується для написання, редагування та відлагодження коду.
3	Zenject	Фреймворк для впровадження залежностей (Dependency Injection) у Unity. Використовується для реалізації DI Container, що забезпечує слабке зв'язування компонентів системи та спрощує тестування.
4	TextMeshPro	Бібліотека для просунутого рендерингу тексту в Unity. Використовується для відображення тексту з високою якістю, підтримкою різних шрифтів, стилів та ефектів для користувацького інтерфейсу гри.
5	Cinemachine	Система керування віртуальними камерами для Unity. Використовується для створення динамічних камер.
6	Addressables	Система керування ресурсами в Unity. Використовується для оптимізації завантаження ігрових ресурсів, що дозволяє динамічно завантажувати і вивантажувати контент за потреби.
7	UniTasks	Асинхронна бібліотека для Unity, що розширює стандартні можливості async/await.
8	DOTween	Бібліотека для анімації через код. Використовується для створення плавних анімацій інтерфейсу, переміщення об'єктів, зміни параметрів з можливістю точного налаштування часу та функцій плавності.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Серед ключових нормативних актів із захисту даних — Загальний регламент про захист даних (GDPR). Оскільки дане програмне забезпечення не збирає й не обробляє персональні дані користувачів і не потребує до них доступу, воно відповідає вимогам GDPR [42].

Висновки до розділу

У даному розділі розглянуто архітектуру та програмну реалізацію рогалік-гри. Проаналізовано та обґрунтовано ключові архітектурні рішення з порівнянням альтернативних підходів.

У підрозділі 3.1 описано монолітну архітектуру програмного забезпечення та застосовані патерни: Dependency Injection через DI Container, Model-View-Presenter для інтерфейсу, Factory і Object Pool для управління об'єктами, принцип "композиція замість успадкування". Детально представлено систему генерації рівнів з використанням Binary Space Partitioning, Cellular Automata та Drunkard Walk алгоритмів, а також трирівневу систему тегування простору. Розглянуто реалізацію штучного інтелекту на основі Behaviour Tree та Blackboard з трирівневою системою контекстів знань.

Підрозділ 3.2 містить порівняльний аналіз засобів розробки. Обґрунтовано вибір Unity замість Unreal Engine, Zenject замість VContainer для впровадження залежностей, MVP замість MVVM для архітектури UI та Object Pool замість стандартного створення/знищення об'єктів.

У підрозділі 3.3 представлено конструювання ключових компонентів програми з результатами генерації рівня. Систематизовано інформацію про використані сторонні утиліти та бібліотеки, включаючи Unity, Rider, та спеціалізовані інструменти.

Підрозділ 3.4 підтверджує відповідність програмного забезпечення вимогам GDPR, оскільки воно не працює з персональними даними користувачів.

Усі обрані рішення забезпечують оптимальний баланс між гнучкістю, швидкістю розробки та продуктивністю, що є ключовим для створення якісного ігрового продукту.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Етап аналізу якості є невід'ємною частиною процесу розробки програмного забезпечення.

Для загальної перевірки якості коду було використано інструмент статичного аналізу NDepend [43]. Цей засіб дозволяє детально дослідити структуру проєкту, виявити можливі архітектурні вади, обчислити такі метрики, як цикломатична складністі, індекс підтримуваності та покриття коду тестами.

Вибір NDepend був обумовлений його здатністю працювати зі складними архітектурами, характерними для ігрових проєктів на Unity. Гнучка система фільтрів дозволяє відфільтрувати тільки основну збірку програми: зокрема, зокремо було виключено Assembly-CSharp-Editor та Assembly-CSharp-Editor-firstpass [44] та збірки зовнішніх плагінів.Результати аналізу якості коду за допомогою NDepend наведено на рисунку 4.1.

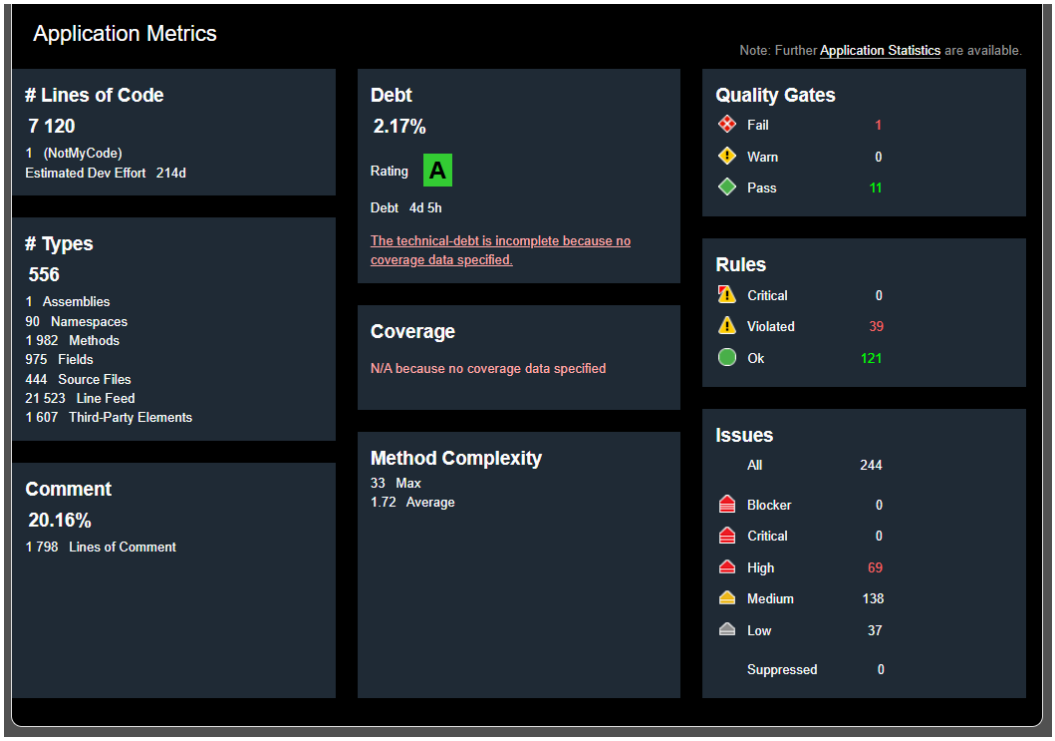


Рисунок 4.1 – Результати загальної перевірки коду з використанням NDepend

На рисунку 4.1 видно, що в проєкті виявлено 69 High Issues. Усі вони сконцентровані в кількох Unity-орієнтованих плагінах — зокрема в TMP_Pro [45], InputActions [46], CinemachineCamera [47]. Цей розподіл чітко показано на рисунку 4.2 Treemap View з перевірки NDepend.

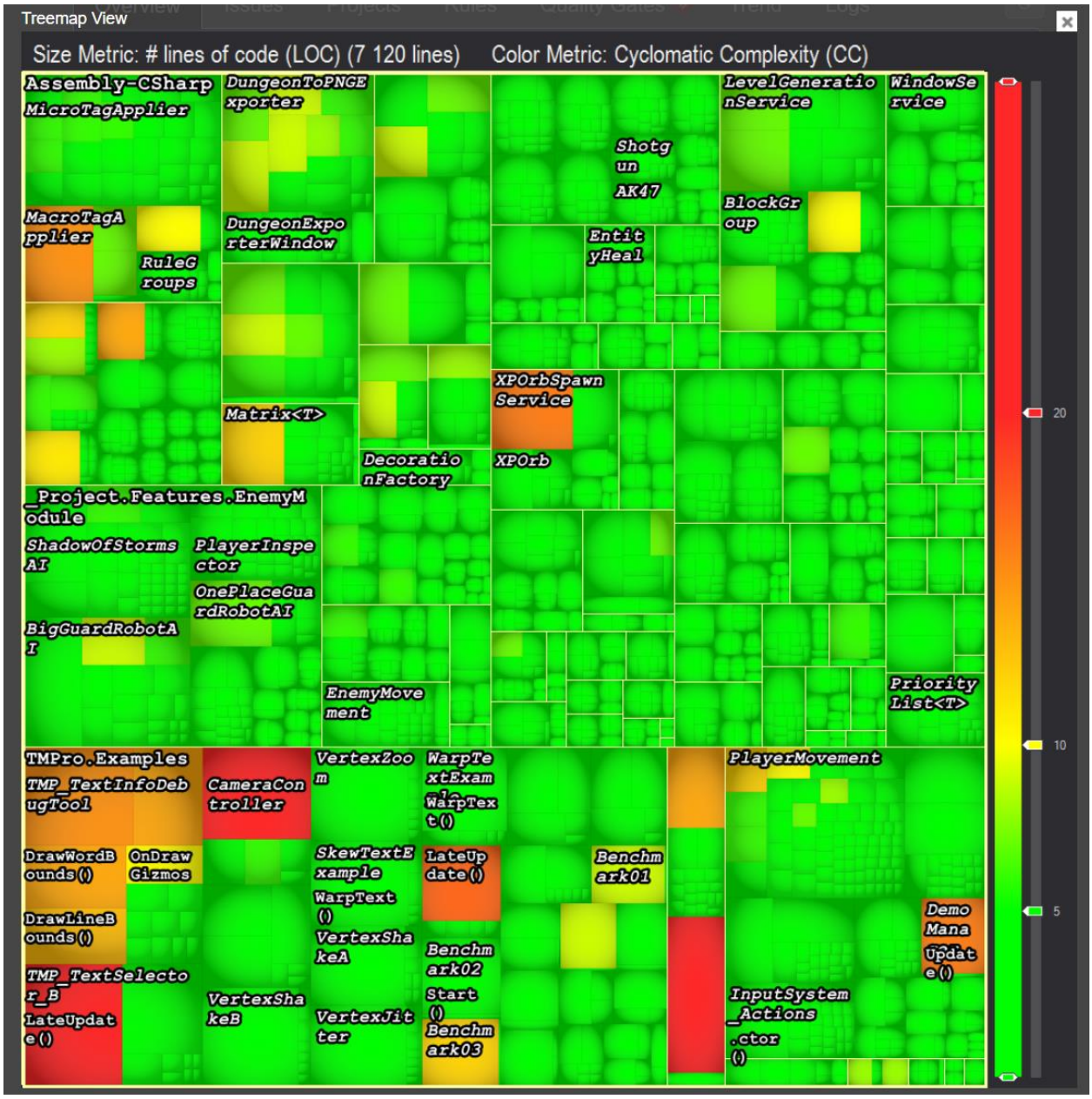


Рисунок 4.2 – Візуальне відображення проблемних місць кодобази з використанням NDepend

Метриками для оцінки якості ПЗ обрано наступні:

– підтримуваність коду;
– цикломатична складність;
– глибина успадкування;

– зв’язаність класів;
– рядки вихідного коду;
– рядки виконуваного коду.

Для кількісної оцінки якості коду було використано вбудований у Visual Studio інструмент Code Metrics [48], який базується на рекомендованих Microsoft стандартах вимірювання. На рисунку 4.3 наведено значення Maintainability Index [49] всього проєкту — 87, що за шкалою Microsoft свідчить про високий рівень підтримуваності коду.

Необхідно також зазначити що на результат також повпливали вищезгадані плагіни(TMP_Pro, InputActions, CinemachineCamera), аналогічно до аналізу з використанням NDepend.

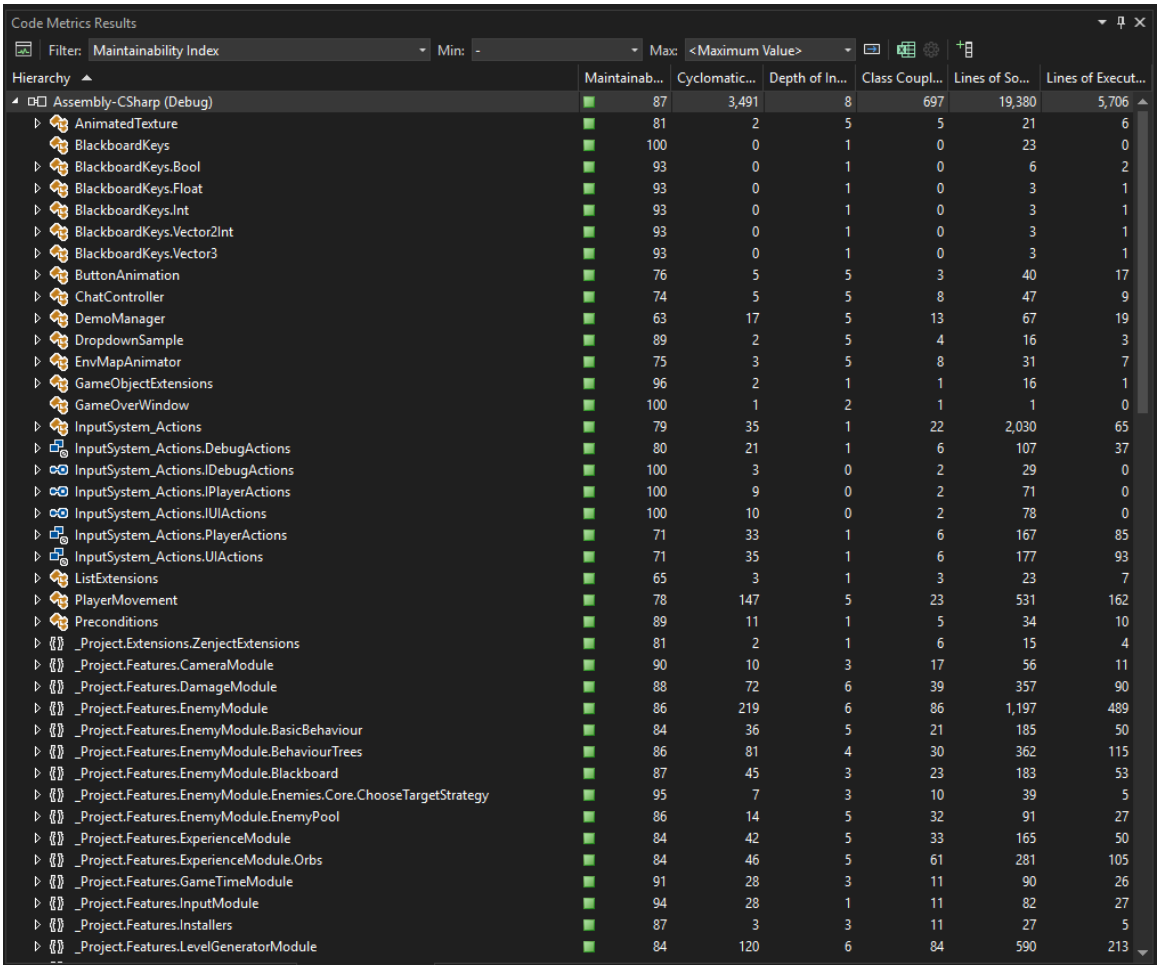


Рисунок 4.3 - Результати кількісного аналізу якості коду з використанням Code Metrics

4.2 Опис процесів тестування

Метою тестування є верифікація відповідності реалізованого ПЗ вимогам і специфікаціям, своєчасне виявлення та документування дефектів, а також підтвердження стабільності й надійності роботи системи в різних умовах експлуатації. Завдяки тестуванню забезпечується якість користувацького досвіду, коректність бізнес-логіки й зменшення ризиків при подальшому супроводі та розширенні проєкту.

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.14.

Таблиця 4.1 – Тест 1.1 Запуск гри з процедурно згенерованим рівнем

Початковий стан системи	Користувач знаходиться в головному меню
Вхідні дані	Натиснута кнопка Грати
Опис проведення тесту	Натиснути кнопку Грати у головному меню
Очікуваний результат	Починається гра на процедурно згенерованому рівні
Фактичний результат	Збігається з очікуваним

Таблиця 4.2 – Тест 1.2 Перехід у режим павзи

Початковий стан системи	Гра запущена, ігровий процес активний
Вхідні дані	Натиснута кнопка Пауза
Опис проведення тесту	Натиснути кнопку Пауза під час гри

Продовження таблиці 4.2

Очікуваний результат	Ігровий процес зупиняється, відображається екран паузи
Фактичний результат	Збігається з очікуваним

Таблиця 4.3 – Тест 1.3 Смерть ворога при критичній шкоді

Початковий стан системи	Гравець поруч із ворогом
Вхідні дані	Нанесення ворогу критичної шкоди (> поточного HP)
Опис проведення тесту	Здійснити атаку, яка зменшує HP ворога до 0 або нижче
Очікуваний результат	Ворог помирає, його об’єкт видаляється з поля
Фактичний результат	Збігається з очікуваним

Таблиця 4.4 – Тест 1.4 Спавн об’єкта досвіду після смерті ворога

Початковий стан системи	Ворог щойно помер
Вхідні дані	—
Опис проведення тесту	Після смерті ворога перевірити спавн об’єкта досвіду
Очікуваний результат	На місці смерті ворога з’являється об’єкт досвіду
Фактичний результат	Збігається з очікуваним

Таблиця 4.5 – Тест 1.5 Екран смерті при зменшенні здоров’я гравця до 0

Початковий стан системи	Здоров’я гравця падає до 0 або нижче
Вхідні дані	Нанесення шкоди гравцю, що призводить до HP < 0
Опис проведення тесту	Здійснити атаку на гравця, поки HP не стане від’ємним
Очікуваний результат	Відображається екран смерті (Death Screen) із кнопками «Перезапустити» і «Вийти»
Фактичний результат	Збігається з очікуваним

Таблиця 4.6 – Тест 1.6 Відкриття вікна вибору покращень

Початковий стан системи	Гравець накопичив досвід, але не підвищив рівень
Вхідні дані	Досвід ≥ порогу для нового рівня
Опис проведення тесту	Накопичити достатньо досвіду для підвищення рівня
Очікуваний результат	Відкривається вікно вибору покращень (Upgrade Window)
Фактичний результат	Збігається з очікуваним

Таблиця 4.7 – Тест 1.7 Застосування покращення до статів гравця

Початковий стан системи	Вікно вибору покращень відкрите
Вхідні дані	Вибране покращення (наприклад +10% до HP)
Опис проведення тесту	Натиснути на іконку бажаного покращення

Продовження таблиці 4.7

Очікуваний результат	Стати гравця оновлюються відповідно до вибраного бонусу
Фактичний результат	Збігається з очікуваним

Таблиця 4.8 – Тест 1.8 Підтвердження вибору покращення

Початковий стан системи	Покращення застосоване, вікно все ще відкрите
Вхідні дані	Натиснута кнопка Підтвердити у вікні покращень
Опис проведення тесту	Натиснути Підтвердити після вибору покращення
Очікуваний результат	Вікно закривається, у HUD видно новий рівень гравця
Фактичний результат	Збігається з очікуваним

Таблиця 4.9 – Тест 1.9 Рух героя за допомогою джойстика

Початковий стан системи	Головний герой чекає введення керування
Вхідні дані	Рух за допомогою екранного джойстика
Опис проведення тесту	Пересувати віртуальний джойстик у різні боки
Очікуваний результат	Герой рухається в напрямку руху джойстика
Фактичний результат	Збігається з очікуваним

Таблиця 4.10 – Тест 1.10 Стрільба героя за допомогою джойстика

Початковий стан системи	Герой озброєний, готовий до стрільби
Вхідні дані	Стрільба за допомогою екранного джойстика
Опис проведення тесту	Натиснути/тримати віртуальний джойстик для стрільби
Очікуваний результат	Герой виконує постріли у напрямку, вказаному джойстиком
Фактичний результат	Збігається з очікуваним

Таблиця 4.11 – Тест 1.11 Зміна ігрового рівня при повторному запуску гри

Початковий стан системи	Гра запущена, але рівень ще не перезапускався
Вхідні дані	Повторне натискання Грати у головному меню
Опис проведення тесту	Вийти в меню та кілька разів запустити гру
Очікуваний результат	Кожен запуск відкриває новий процедурно згенерований рівень
Фактичний результат	Збігається з очікуваним

Таблиця 4.12 – Тест 1.12 Стрибок і падіння з платформи

Початковий стан системи	Гравець знаходиться біля платформи
Вхідні дані	Натискання кнопки стрибка на екрані
Опис проведення тесту	Натиснути віртуальну кнопку стрибка; потім утримання/натискання для зістрибку
Очікуваний результат	Герой застрибує на платформу та успішно зістрибне з неї без застрягання
Фактичний результат	Збігається з очікуваним

Таблиця 4.13 – Тест 1.13 Закриття гри за кнопкою виходу

Початковий стан системи	Гра запущена
Вхідні дані	Натиснута Вийти в меню або на екрані смерті
Опис проведення тесту	Натиснути Вийти
Очікуваний результат	Гра коректно закривається, процес завершено
Фактичний результат	Збігається з очікуваним

Таблиця 4.14 – Тест 1.14 Перезапуск рівня з екрану смерті

Початковий стан системи	Користувач на екрані смерті
Вхідні дані	Натискання кнопки Перезапустити рівень на екрані смерті

Продовження таблиці 4.14

Опис проведення тесту	Натиснути Перезапустити рівень
Очікуваний результат	Поточний рівень перезапускається з початковими параметрами
Фактичний результат	Збігається з очікуваним

4.3 Опис контрольного прикладу

Наступним кроком розглянемо контрольний приклад.

Після запуску гри відображається головне меню. На головному меню бачимо 2 опції: почати гру та вийти. При натисканні на кнопку вийти застосунок коректно закривається. При натисканні на кнопку грати завантажується ігровий рівень з процедурно згенерованою мапою. На рисунку 4.4 зображено головне меню гри.

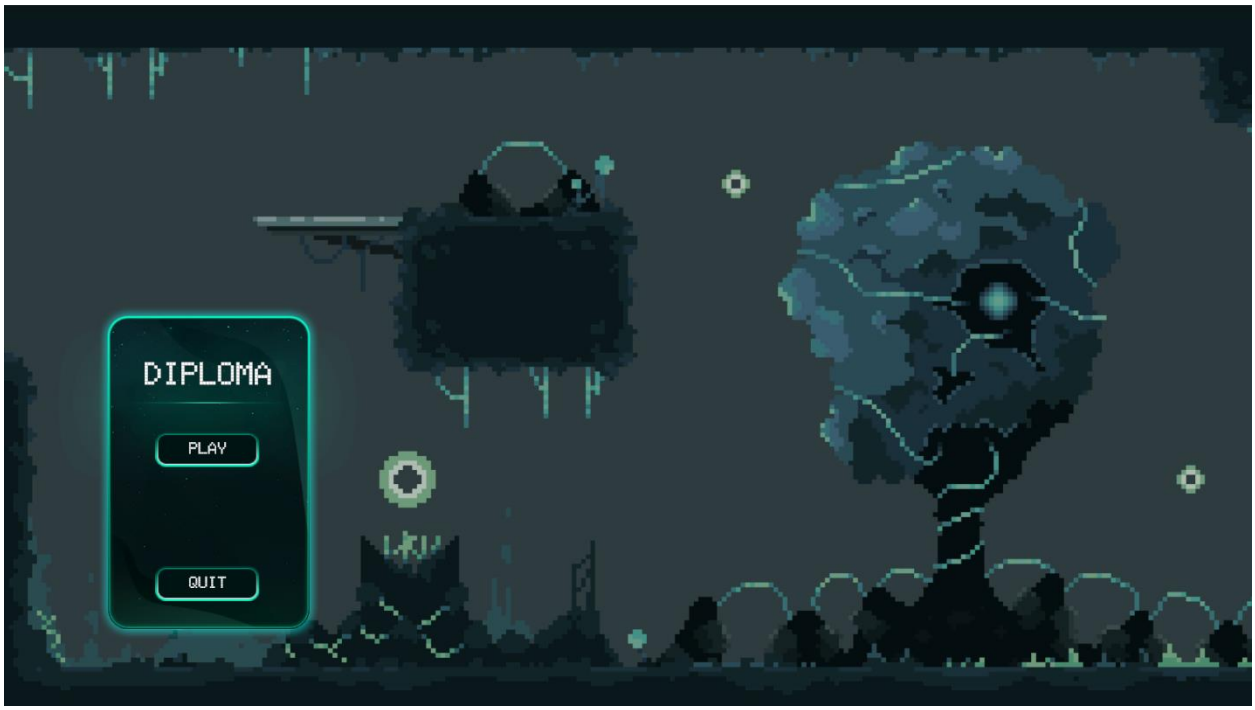


Рисунок 4.4 – Головне меню гри

Після завантаження рівня можемо маємо побачити головного героя в центрі екрану. Також на рівні розставляються платформи, декорації, вороги,

фон та елементи графічного інтерфейсу. Ігровий рівень зображено на рисунку 4.5.

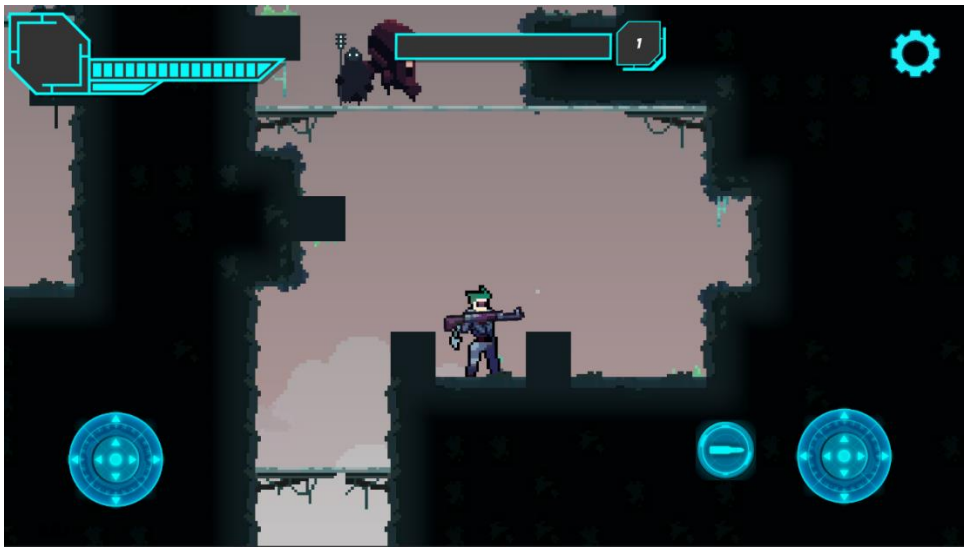


Рисунок 4.5 – Процедурно згенерований ігровий рівень з розміщеними елементами рівня, ворогами та графічним інтерфейсом

При використанні контролів на екрані або клавіатури гравець починає рухатися в сторону визначену гравцем. Також гравець може виконати стрибок, подвійний стрибок, застрибувати на платформи та зістрибувати з них. На рисунках 4.6 та 4.7 зображено рух гравця та зістрибування з платформи відповідно.

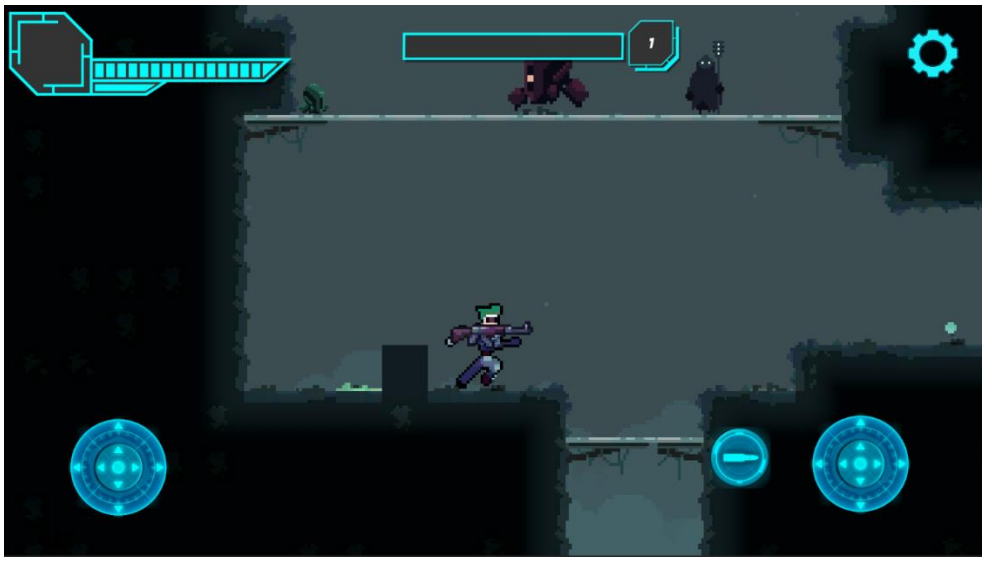


Рисунок 4.6 – Рух гравця по процедурно згенерованій мапі

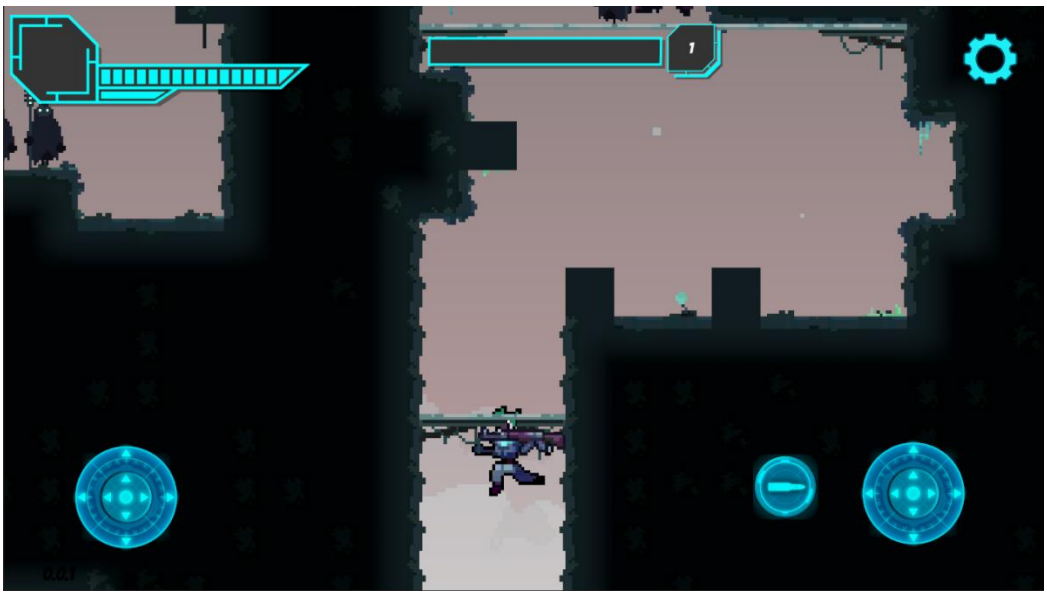


Рисунок 4.7 – Зістрибування гравця з платформи розміщеної на згенерованому рівні

Якщо гравець знаходиться у дистанції атаки ворога і ШІ останнього вирішує атакувати гравця, то ворог виконує свою атаку. При попаданні атаки по гравцю гравець отримує пошкодження, якщо атака не попадає по гравцю відіграється спеціальний ефект. На рисунку 4.8 зображено атаку одного з ворогів.

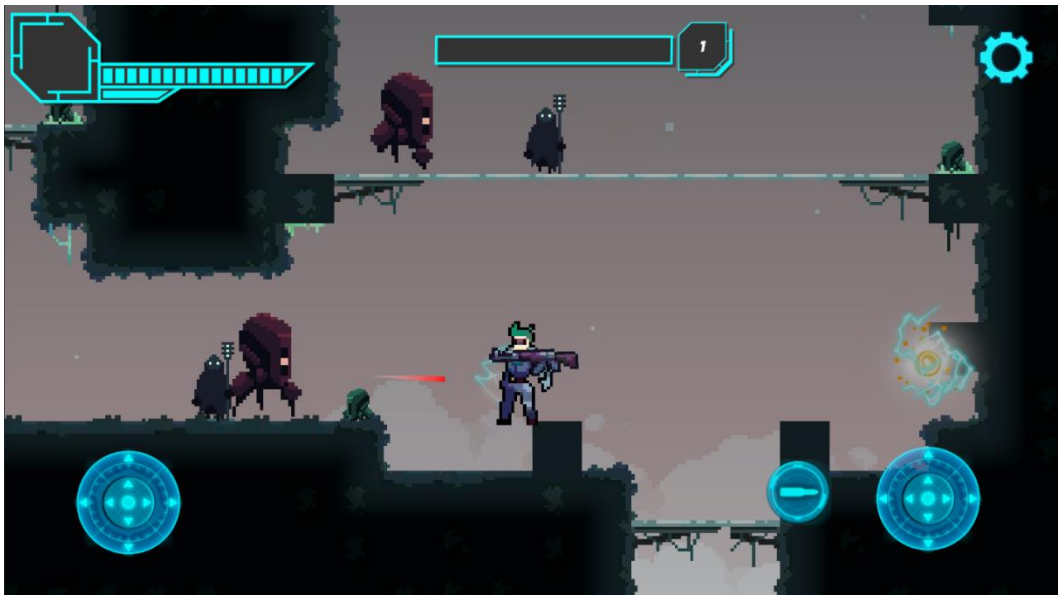


Рисунок 4.8 – Атака одного з ворогів

Якщо після отримання серії шкоди від ворогів здоров’я гравця опускається до нуля, гра відображає відповідне вікно на екрані з опціями виходу в меню та перезапуску рівня. На рисунку 4.9 зображено вікно гравця.

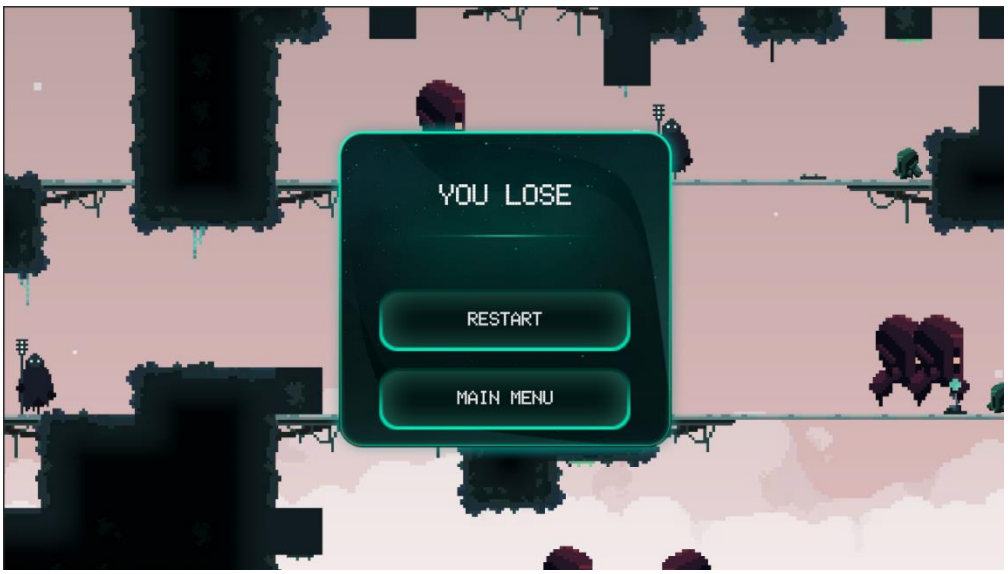


Рисунок 4.9 – Вікно програшу гравця після отримання критичної шкоди

Після нанесення ворогу критичної шкоди від атак гравця з ворогів з’являються ігрові об’єкти досвіду, якщо інакша логіка не прописана ШІ ворога або правилами гри. Після підбирання досвіду у гравця заповнюється шкала рівня. На рисунку 4.10 зображено досвід що випав з ворога та частково заповнена шкала досвіду.

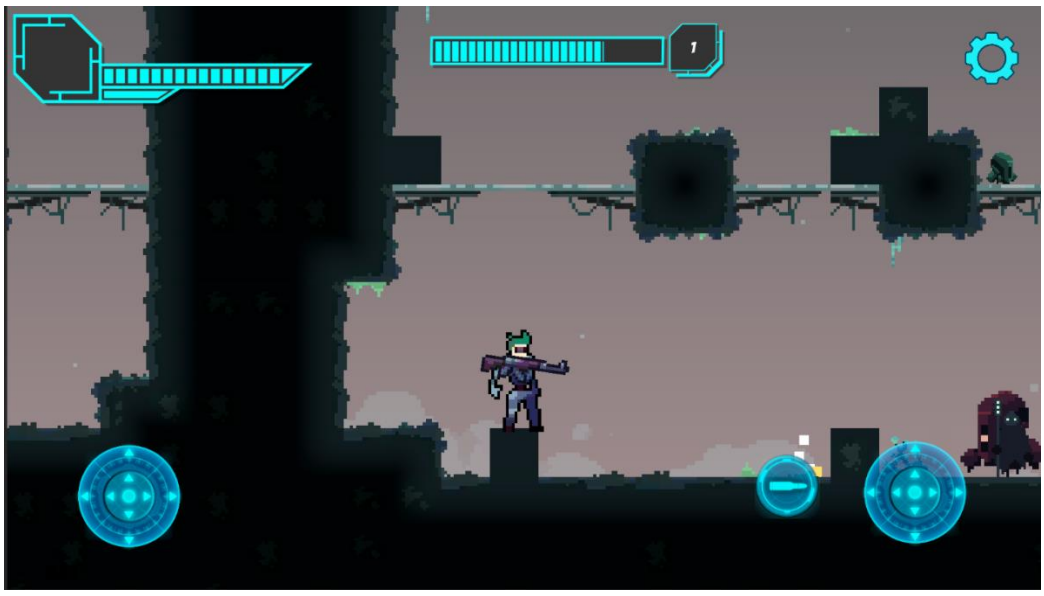


Рисунок 4.10 – Зображення досвіду що випадає з ворога

Після накопичення достатньої кількості досвіду гравцеві відкривається вікно вибору покращення з трьома опціями. Після вибору однієї з опцій дане вікно закривається, гравець отримує покращення, показник рівня гравця

збільшується на одну одиницю. На рисунках 4.11, 4.12 зображені вікно вибору покращення гравця та графічний інтерфейс з оновленим рівнем відповідно.

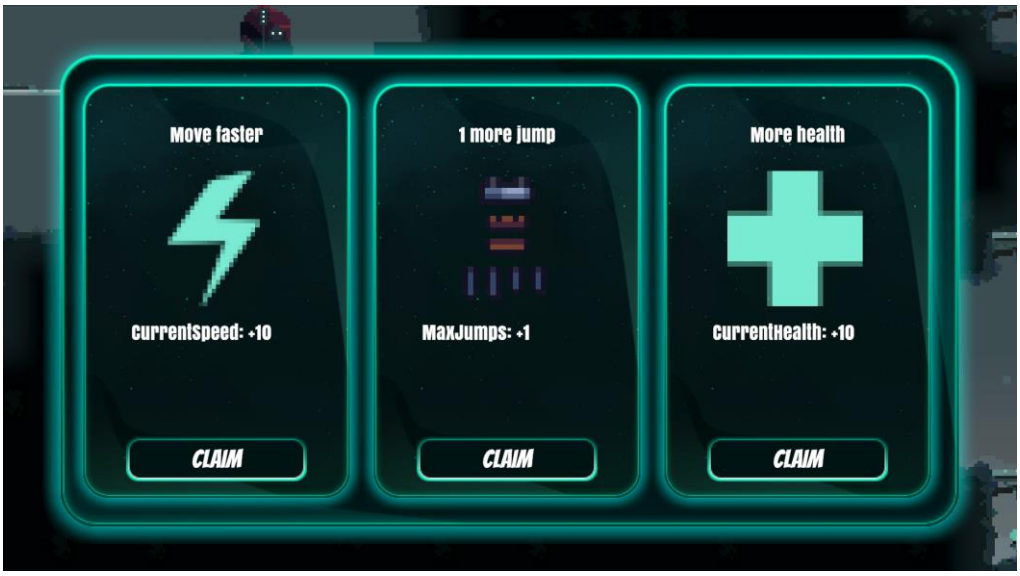


Рисунок 4.11 – Вікно вибору покращення гравця

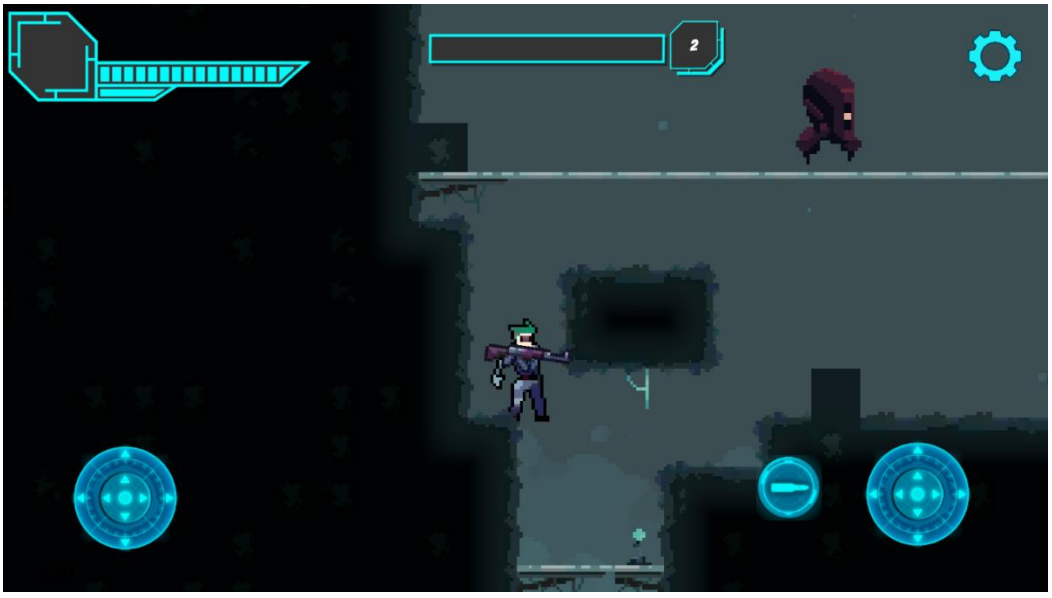


Рисунок 4.12 – Графічний інтерфейс з оновленим показником рівня

При натисканні кнопки ігрового меню в кутку екрану відкривається відповідне вікно з опціями продовження гри, виходу в головне меню та завершення ігрової сесії. При натисканні кожної з цих кнопок коректно виконується відповідна дія. На рисунку 4.13 зображено вікно внутрішньо-ігрового меню.

Рисунок 4.13 – Внутрішньо-ігрове меню гри

Висновки до розділу

У даному розділі було проведено комплексну оцінку якості розробленого програмного забезпечення та його тестування.

У підрозділі 4.1 було виконано статичний аналіз коду за допомогою інструмента NDepend, який виявив певну кількість проблем високого рівня, переважно сконцентрованих у зовнішніх Unity-плагінах. Кількісна оцінка якості коду засобом Code Metrics показала високий індекс підтримуваності — 87 балів за шкалою Microsoft. Також було проведено оцінку ключових метрик, таких як підтримуваність коду, цикломатична складність, глибина наслідування, зв'язаність класів та обсяг коду.

У підрозділі 4.2 було виконане мануальне тестування програмного забезпечення відповідно до документа «Програма та методика тестування». Проведено 14 тестових сценаріїв, що охоплюють ключові функції гри. Усі тести успішно пройдені — фактичні результати збігаються з очікуваними.

У підрозділі 4.3 було продемонстровано повний цикл роботи з програмним забезпеченням від запуску до завершення гри. Підтверджено коректність роботи всіх ігрових механік: навігації меню, процедурної генерації рівнів, системи керування персонажем, бойової системи, системи прогресії та досвіду, а також внутрішньо-ігрового інтерфейсу.

Результати тестування підтверджують якість розробленого продукту, його стабільність та відповідність поставленим вимогам. Виявлені недоліки стосуються виключно зовнішніх компонентів і не впливають на функціональність основної системи.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Розгортання мобільного додатку складається з двох етапів: створення та налаштування додатку на Google Play Console [50], створення збірки всередині Unity

Для створення мобільного додатку в Google Play Console треба перейти у відповідну вкладку в головному меню та заповнити інформацію стосовно додатку. На рисунку 5.1 зображено процес заповнення сторінку «Create App».

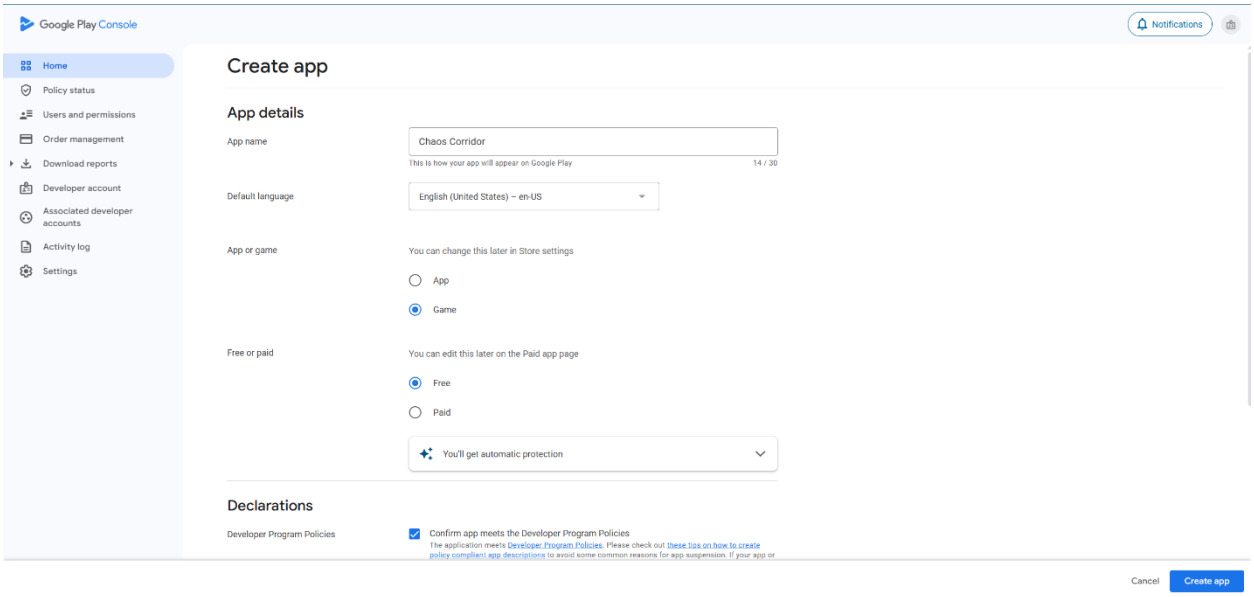


Рисунок 5.1 – Процес створення нового додатку в Google Play Console

Тепер перейдемо до створення збірки в Unity. Для початку перейдемо у вкладку Build Profiles. Тут обираємо платформу Android, також додаємо всі необхідні сцени та обираємо опцію білда Build App Bundle [51]. На рисунку 5.2 зображено обрані налаштування для Build Profiles застосунку.

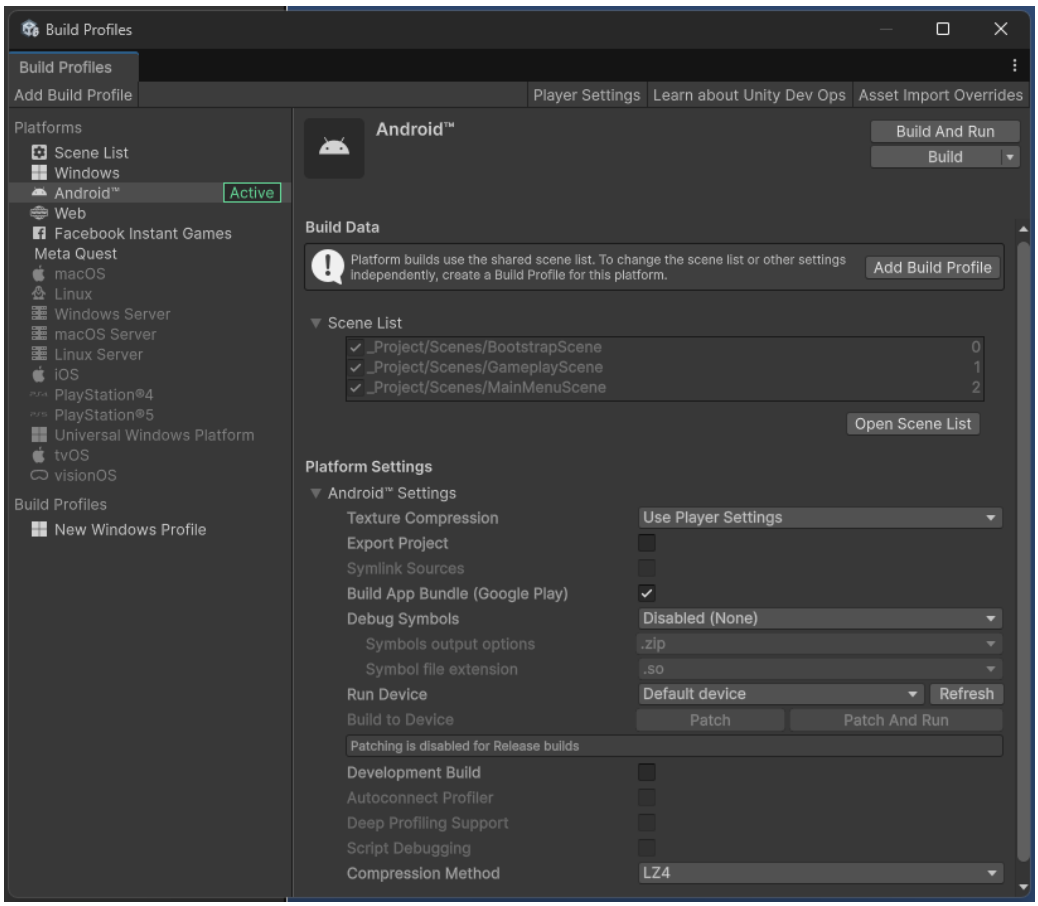


Рисунок 5.2 – Налаштування Build Profiles для створення мобільного білда для Google Play

Далі переходимо до Player Settings для подальшого налаштування іконки, назви гри, вікна застосунку, підтримуваних версій Android тощо. Використані налаштування Player Settings показано на рисунках 5.3 – 5.6.

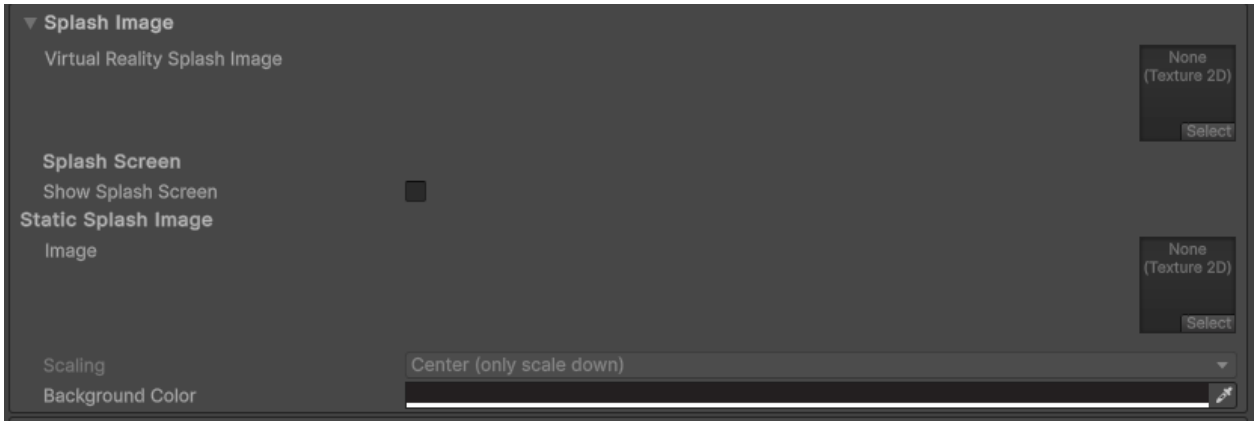


Рисунок 5.3 – Налаштування заставки гри в меню Player Settings

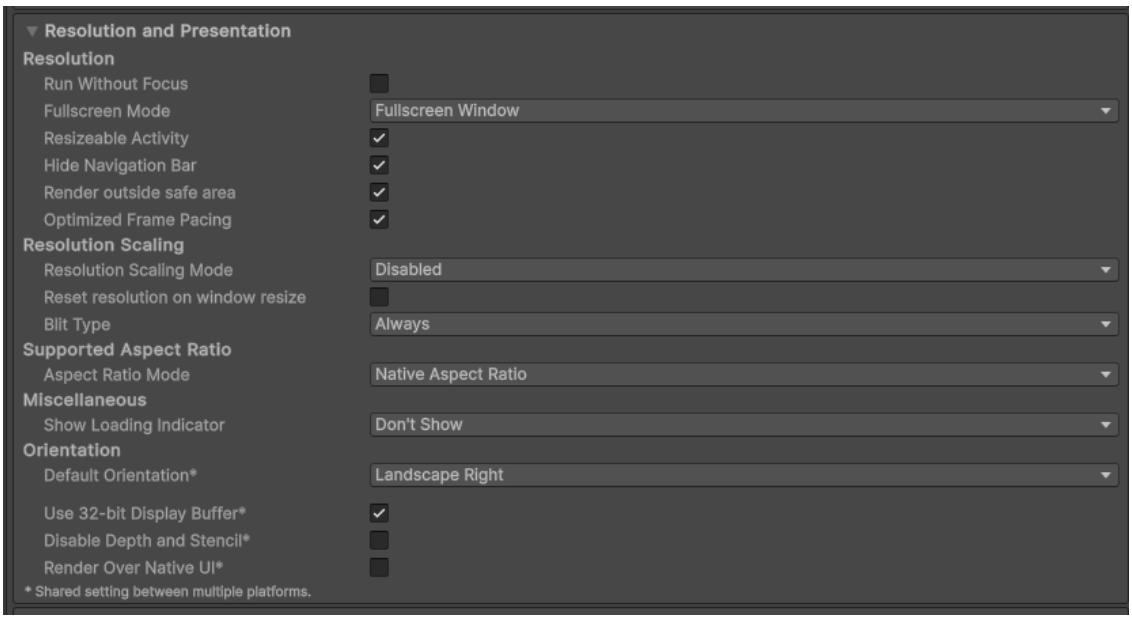


Рисунок 5.4 – Налаштування вікна застосунку в меню Player Settings

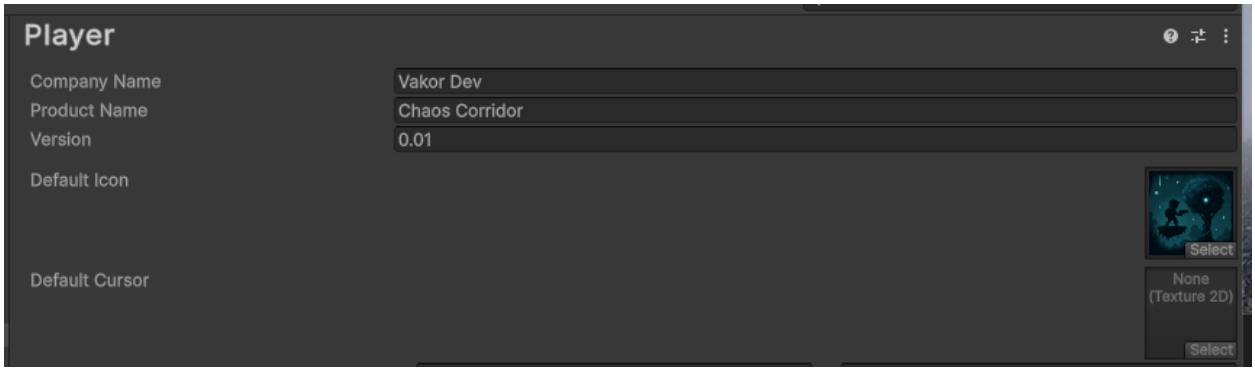


Рисунок 5.5 – Налаштування назви гри, її версії, назви компанії, іконки в меню Player Settings

Рисунок 5.6 – Налаштування підтримуваних версій Android, bundle code в меню Player Settings

Для розповсюдження гри через Google Play Store необхідно також створити Keystore – ключ завдяки якому можна створювати нові збірки. Даний ключ дозволяє запевнитись що навіть при втраті вихідного коду зловмисник не зможе створити нову збірку. На рисунках 5.7 – 5.9 показано процес створення Keystore, його заповнення в налаштуваннях Player Settings.

Рисунок 5.7 – Створення Keystore через меню Keystore Manager

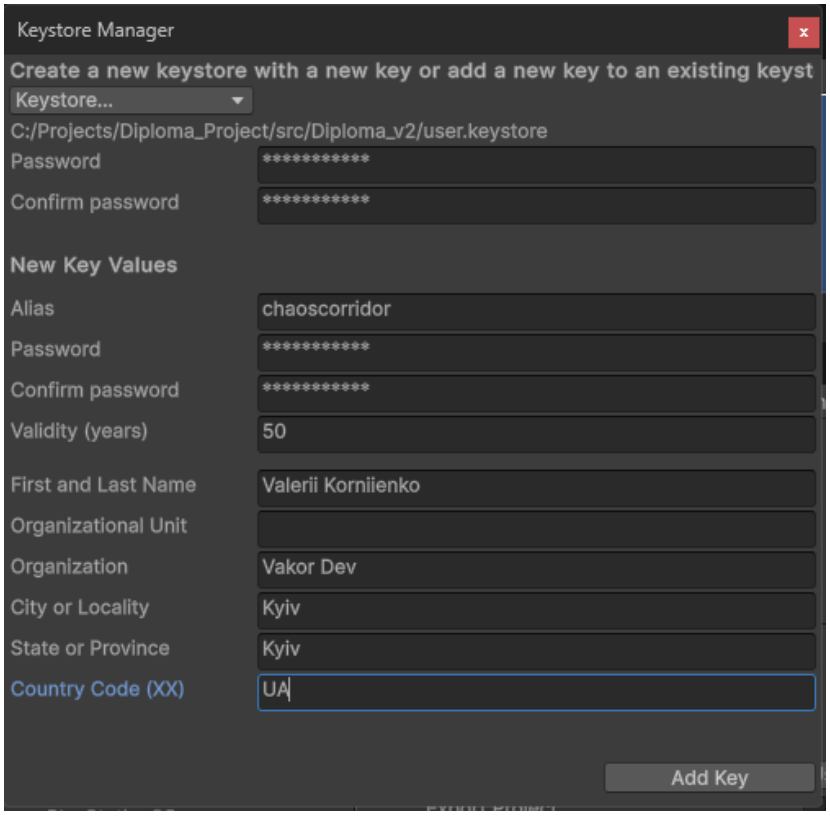


Рисунок 5.8 – Заповнення інформації про Keystore застосунку

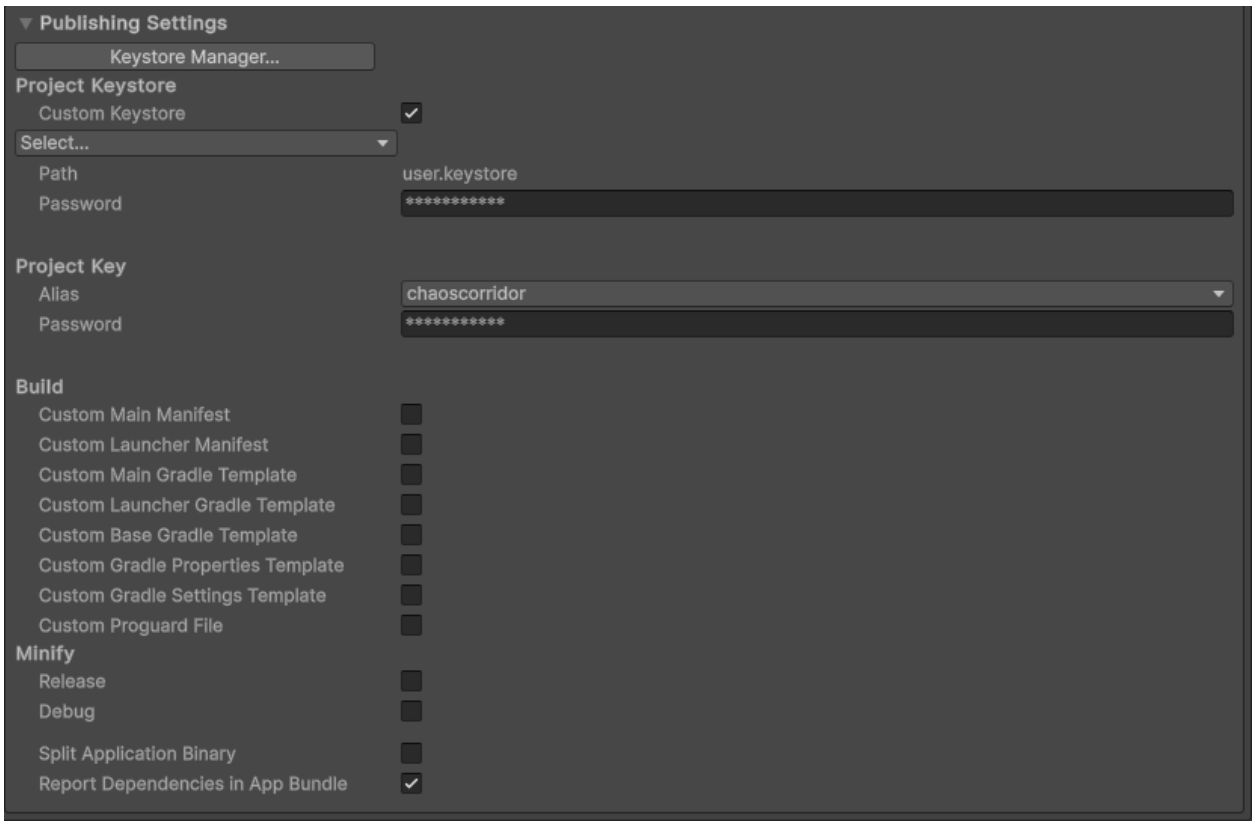


Рисунок 5.9 – Заповнення інформації про Keystore в меню Player Settings

Після заповнення інформації Player Settings, створення нового нового Keystore натискаємо на кнопку створення збірки і очікуємо результат виконання операції. На рисунку 5.10 зображено вихідні файли після успішного створення збірки

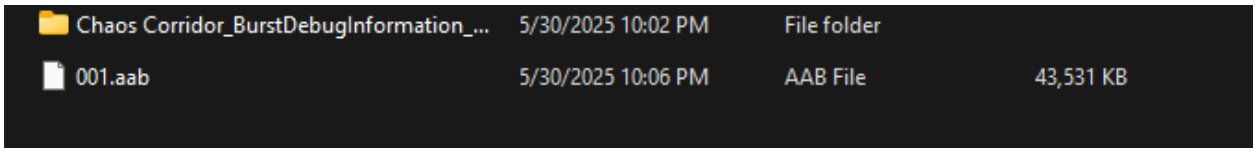


Рисунок 5.10 – Вихідні файли після успішного створення збірки застосунку

Після створення збірки застосунку переходимо в Google Play Console в меню Internal Testing. Сюди треба завантажити збірку, заповнити інформацію про реліз та опублікувати збірку. Після цього залишиться тільки натиснути кнопку Promote Release щоб збірка з грою опинилися у доступі в Google Play Store. На рисунках 5.11 – 5.12 зображено процес створення internal release [52] в Google Play Store.

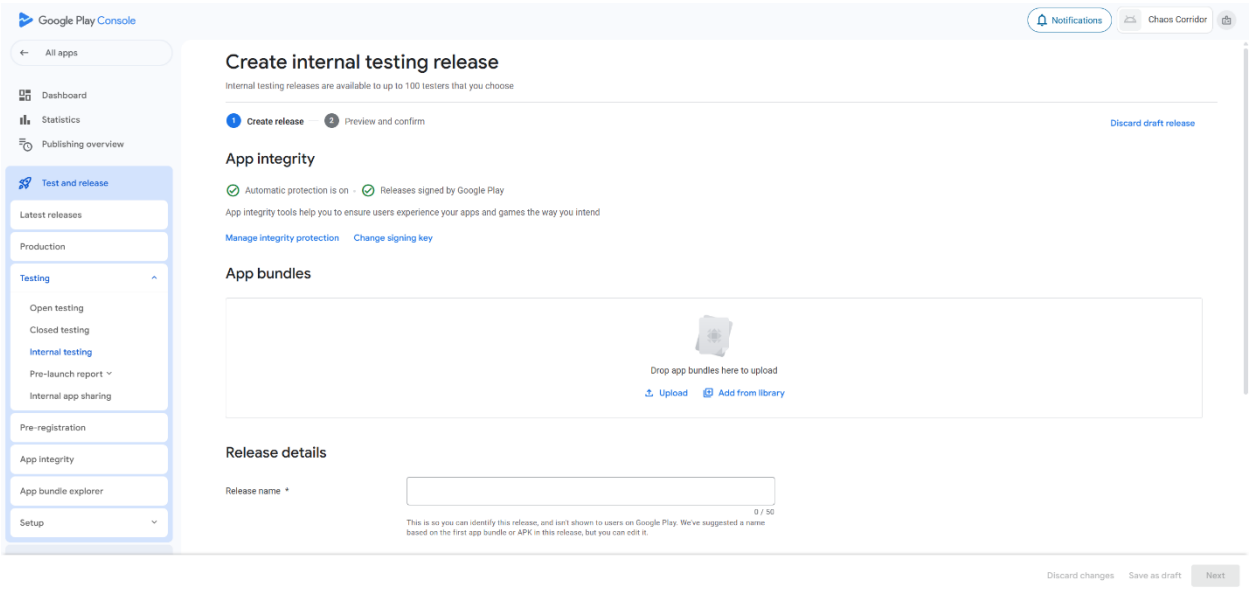


Рисунок 5.11 – Вікно створення internal testing release в Google Play Store

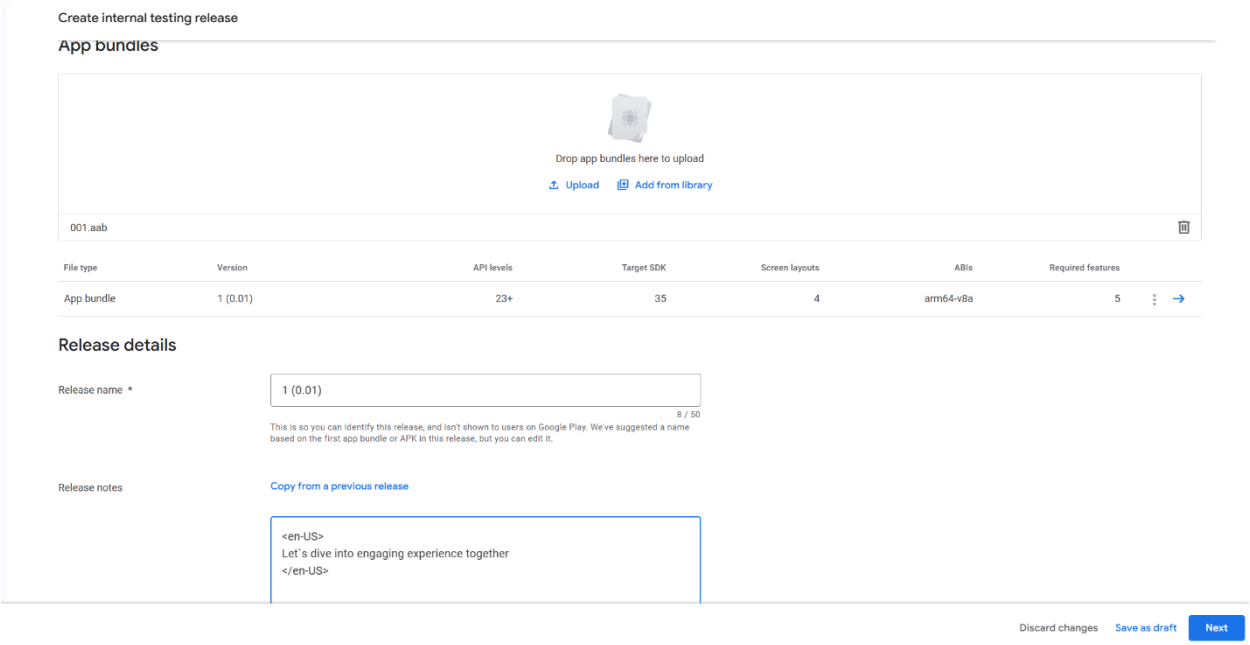


Рисунок 5.12 – Вікно створення internal testing release з заповненою інформацією в Google Play Store

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі «Керівництво користувача».

Для випуску нових версій застосунку необхідно створити збірку аналогічно до початкової збірки, цей процес був описаний в попередньому пункті. Відмінностями є лише збільшення версії гри в Player Settings та

збільшення Bundle Version Code. Збільшення версії гри та Bundle Version Code зображено на рисунках 5.13, 5.14 відповідно.

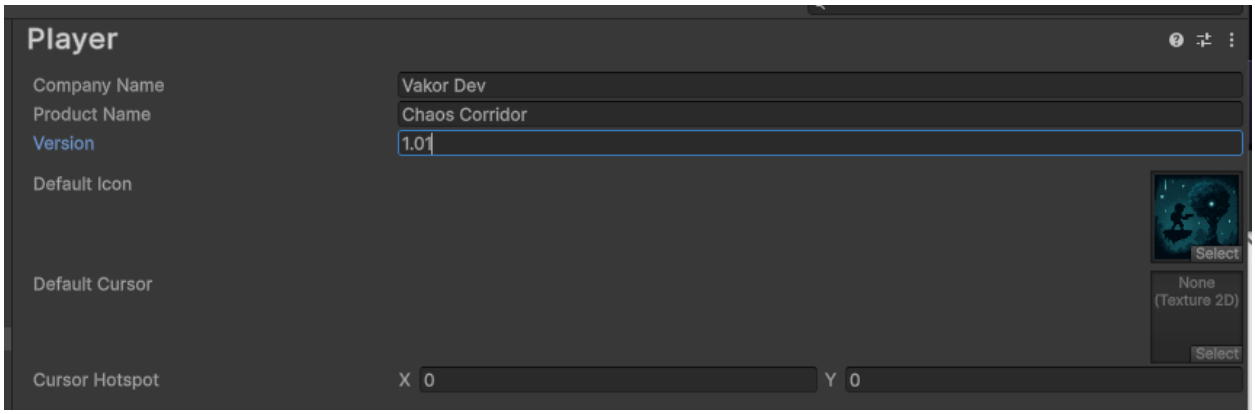


Рисунок 5.13 – Налаштування версії ігрового застосунку в меню Player Settings

Рисунок 5.14 – Налаштування Bundle Version Code в меню Player Settings

Після створення нової збірки її можна опублікувати в Google Play Store аналогічно до публікації першої збірки

Висновки до розділу

В даному розділі описаний двоетапний процес розгортання мобільного додатку: налаштування в Google Play Console та створення збірки в Unity.

У Google Play Console відбувається реєстрація додатку з необхідною інформацією. В Unity проводиться конфігурація Build Profiles, Player Settings та створення захисного Keystore для безпечного розповсюдження.

Після генерації збірки вона завантажується в Google Play Store через Internal Testing з подальшою публікацією.

Супровід програмного забезпечення передбачає окрему інструкцію користувача та стандартний процес випуску оновлень зі збільшенням версії програми.

ВИСНОВКИ

У дипломній роботі було реалізовано повний цикл розробки мобільного застосунку у форматі гри жанру Shooter з елементами Roguelike. Робота складалась із п’яти основних розділів, кожен із яких мав конкретну мету та охоплював окремий етап створення програмного забезпечення.

У першому розділі було сформульовано мету та завдання дипломного проєкту, обґрунтовано актуальність тематики та проведено аналіз предметної області. Здійснено огляд ринку мобільних ігор, визначено характерні риси жанру Roguelike та його особливості на мобільних платформах. Проведено порівняльний аналіз ігор-аналогів, таких як Dead Cells, The Binding of Isaac, Pixel Dungeon, що дозволило виявити ключові ігрові механіки та підходи до реалізації подібних проєктів. Також було досліджено алгоритми процедурної генерації рівнів (Binary Space Partitioning, Cellular Automata, Drunkard Walk) і моделі побудови штучного інтелекту (FSM, Behaviour Tree, Utility AI, нейронні мережі). На основі цього аналізу визначено доцільність використання комбінованого підходу до генерації рівнів та реалізації AI через структуру Behaviour Tree у поєднанні з контекстною моделлю знань.

У другому розділі здійснено формалізацію вимог до розроблюваного програмного забезпечення. Створено діаграму варіантів використання, яка охоплює основні сценарії взаємодії користувача з грою. Детально описано функціональні вимоги до системи, розподілено їх за пріоритетами та оцінено ризики. У розділі визначено нефункціональні вимоги до продуктивності (включаючи FPS, швидкість завантаження рівнів), сумісності (версії ОС та екранів), надійності (безаварійна робота), безпеки (шифрування локальних даних) та зручності інтерфейсу. Проведено аналіз системних вимог щодо операційної системи, графічного API, обсягу оперативної пам’яті та вільного місця для інсталяції гри. В результаті сформовано технічне завдання, яке визначає функціональні блоки гри: генерацію рівнів, бойову систему, систему управління, прогресію гравця та взаємодію з об’єктами.

У третьому розділі описано архітектурні рішення та реалізацію програмного забезпечення. Розглянуто структуру проєкту, обрану архітектуру (монолітна з внутрішнім розподілом за модулями) та шаблони проєктування, зокрема використання Dependency Injection через Zenject, застосування патернів MVP, Factory, Object Pool. Описано реалізацію процедурної генерації рівнів з використанням поєднання трьох алгоритмів: BSP, Cellular Automata та Drunkard Walk. Наведено структуру системи штучного інтелекту з Behaviour Tree та контекстами знань на основі Blackboard. Окрему увагу приділено вибору середовища розробки (Unity, JetBrains Rider), інструментів та бібліотек. Показано, як забезпечується масштабованість і модульність системи. Також перевірено відповідність програмного забезпечення вимогам щодо обробки даних (GDPR), з урахуванням того, що персональні дані не обробляються.

У четвертому розділі виконано тестування програмного забезпечення. Здійснено статичний аналіз коду за допомогою інструмента NDepend та засобу Code Metrics. Проведено оцінювання таких метрик, як цикломатична складність, підтримуваність, зв’язність класів, глибина наслідування. Крім того, здійснено мануальне тестування за визначеним тест-планом: виконано 14 сценаріїв, що охоплюють ключовий функціонал гри (рух, стрільба, генерація рівня, керування персонажем, інтерфейс). Здійснено верифікацію роботи ігрових механік на етапі запуску, проходження рівнів і завершення сесії. Виявлені проблеми були локалізовані у сторонніх компонентах і не стосувались критичної функціональності гри.

У п’ятому розділі описано процес розгортання програмного забезпечення. Представлено поетапну інструкцію з розміщення гри в Google Play: створення сторінки в Google Play Console, конфігурація збірки в Unity (налаштування Build Profiles, Player Settings, Keystore), створення App Bundle і завантаження його в систему внутрішнього тестування. Також описано етапи оновлення додатку та процедури супроводу, включаючи інструкції користувача та підготовку нових релізів.

У процесі виконання дипломного проєкту всі поставлені завдання були реалізовані відповідно до технічного завдання. Завдання розробки програмного забезпечення — включаючи проєктування архітектури, реалізацію ключових функціональних блоків, забезпечення відповідності системним та нефункціональним вимогам, а також проведення тестування та розгортання — виконано у повному обсязі. Кожен етап розробки відповідав встановленим критеріям, що дозволило забезпечити цілісність і функціональну завершеність програмного продукту.

У межах обраної предметної області подальший розвиток програмного забезпечення є технічно доцільним. Існує потенціал для розширення функціональності, зокрема — впровадження мультиплеєрного режиму, додаткових ігрових механік, системи монетизації, локалізації та кросплатформеної підтримки. Ці напрями можуть стати основою для наступних етапів розробки або окремих дослідницьких чи комерційних проєктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Buijsman M. The global games market will generate $187.7 billion in 2024. newzoo. URL: https://newzoo.com/resources/blog/global-games-market-revenue-estimates-and-forecasts-in-2024 (дата звернення: 10.06.2025).

2) Nutt C. 'Roguelikes': getting to the heart of the it-genre. Game Developer | Game Industry News, Deep Dives, and Developer Blogs. URL: https://www.gamedeveloper.com/business/-roguelikes-getting-to-the-heart-of-the-it-genre?utm_source=chatgpt.com (дата звернення: 14.06.2025).

3) Oya A. First-Person shooter videogames. First-Person shooter videogames. 2023. С. 1–100. URL: https://doi.org/10.1163/9789004691476_002 (дата звернення: 14.06.2025).

4) Mobile games - worldwide | statista market forecast. Statista. URL: https://www.statista.com/outlook/amo/media/games/mobile-games/worldwide (дата звернення: 14.06.2025).

5) Optimize your game performance for mobile. Unity. URL: https://unity.com/resources/unity-e-book-optimize-your-mobile-game-performance?utm_source=chatgpt.com (дата звернення: 14.06.2025).

6) Short T. X., Adams T. Procedural generation in game design / ред.: T. Short, T. Adams. Boca Raton : Taylor & Francis, CRC Press, 2017. : A K Peters/CRC Press, 2017. URL: https://doi.org/10.1201/9781315156378 (дата звернення: 14.06.2025).

7) Colledanchise M., Ögren P. Behavior trees in robotics and AI: an introduction. Taylor & Francis Group, 2018. 192 с.

8) Tidwell J. Designing interfaces: patterns for effective interaction design. O'Reilly Media, Incorporated, 2005.

9) Dead cells - the roguevania from motion twin. Dead Cells - Rogue-lite metroidvania with some souls-lite combat on top! Kill, die, learn, repeat. URL: https://dead-cells.com/?utm_source=chatgpt.com (дата звернення: 14.06.2025).

10) Stone A. Even without Hollow Knight: Silksong, these 10 games prove that 2024 might just be the best Metroidvania year ever. PC Gamer. URL:

https://www.pcgamer.com/games/best-metroidvania-games-2024/?utm_source=chatgpt.com (дата звернення: 14.06.2025).

11) The binding of isaac: repentance on steam. Welcome to Steam. URL: https://store.steampowered.com/app/1426300/The_Binding_of_Isaac_Repentance/ (дата звернення: 14.06.2025).

12) Pixel dungeon. Pixel Dungeon. URL: https://pixeldungeon.watabou.ru/ (дата звернення: 14.06.2025).

13) Shaker N., Togelius J., Nelson M. J. Procedural content generation in games. Cham : Springer International Publishing, 2016. URL: https://doi.org/10.1007/978-3-319-42716-4 (дата звернення: 14.06.2025).

14) B. Temuçin M., Kocabaş İ., Oğuz K. Using cellular automata as a basis for procedural generation of organic cities. European Journal of Engineering and Technology Research. URL: https://ej-eng.org/index.php/ejeng/article/download/2293/1005/9053 (дата звернення: 14.06.2025).

15) Koesnaedi A., Istiono W. Implementation drunkard's walk algorithm to generate random level in roguelike games. IJMRAP – International Journal of Multidisciplinary Research and Publications (IJMRAP)-ISSN (Online): 2581-6187. URL: https://ijmrap.com/wp-content/uploads/2022/07/IJMRAP-V5N2P27Y22.pdf (дата звернення: 14.06.2025).

16) Procedural dungeon generation analysis and adaptation. ResearchGate. URL: https://www.researchgate.net/publication/316848565_Procedural_Dungeon_Generation_Analysis_and_Adaptation (дата звернення: 14.06.2025).

17) Unity artificial intelligence programming - fourth edition. O'Reilly Online Learning. URL: https://www.oreilly.com/library/view/unity-artificial-intelligence/9781789533910/85337571-c489-4947-a87c-9ce45026ad71.xhtml#:~:text=Finite%20State%20Machines%20(FSMs)%20are,by%20the%20transitions%20between%20them. (дата звернення: 14.06.2025).

18) SmythOS - understanding utility-based AI agents and their applications. SmythOS. URL: https://smythos.com/managers/ops/utility-based-ai-agents/ (дата звернення: 14.06.2025).

19) Applications of neural-based agents in computer game design. IntechOpen - Open Science Open Minds | IntechOpen. URL: https://www.intechopen.com/chapters/10916 (дата звернення: 14.06.2025).

20) IBM. What is mean time between failure (MTBF)? | IBM. IBM - United States. URL: https://www.ibm.com/think/topics/mtbf#:~:text=Mean%20time%20between%20failure%20(MTBF)%20is%20a%20measure%20of%20the,will%20operate%20before%20it%20fails. (дата звернення: 14.06.2025).

21) Understanding success criterion 2.5.5: target size | WAI | W3C. W3C. URL: https://www.w3.org/WAI/WCAG21/Understanding/target-size.html (дата звернення: 14.06.2025).

22) Unity plans & pricing: pro, personal, enterprise, industry | unity. Unity. URL: https://unity.com/products (дата звернення: 14.06.2025).

23) Seufert E. B. Freemium economics: leveraging analytics and user segmentation to drive revenue. Elsevier Science & Technology Books, 2013.

24) What is average revenue per user and how is ARPU calculated? | Adjust. Accelerate your app’s growth with Adjust | Adjust. URL: https://www.adjust.com/glossary/arpu-definition/?utm_source=chatgpt.com (дата звернення: 14.06.2025).

25) Potel M. MVP: model-view-presenter the taligent programming model for C++ and java. Wildcrest Associates - Software products and consulting. URL: https://www.wildcrest.com/Potel/Portfolio/mvp.pdf (дата звернення: 14.06.2025).

26) Sierra K. Head first design patterns. O'Reilly Media, Incorporated, 2004.

27) Design patterns / R. Helm та ін. Addison Wesley, 2005.

28) Дібрівний Л. Як оптимізувати ігровий проєкт за допомогою патерну objectpool. посібник для unity-розробників. Gamedev DOU. URL:

https://gamedev.dou.ua/blogs/optimizing-a-game-project-with-object-pool-pattern/ (дата звернення: 14.06.2025).

29) Unity real-time development platform | 3D, 2D, VR & AR engine. Unity. URL: https://unity.com/ (дата звернення: 14.06.2025).

30) Unreal engine. Unreal Engine. URL: https://www.unrealengine.com/en-US (дата звернення: 14.06.2025).

31) Unity vs unreal engine: understanding the 7 key differences. RetroStyle Games. URL: https://retrostylegames.com/blog/unity-vs-unreal-engine/ (дата звернення: 14.06.2025).

32) Technologies A. Cross-Platform game development: unity vs. unreal engine. Medium. URL: https://medium.com/@appvintechnologies/cross-platform-game-development-unity-vs-unreal-engine-c661ca5a62dd (дата звернення: 14.06.2025).

33) GitHub - modesttree/zenject: dependency injection framework for unity3d. GitHub. URL: https://github.com/modesttree/Zenject (дата звернення: 14.06.2025).

34) About | VContainer. About | VContainer. URL: https://vcontainer.hadashikick.jp/ (дата звернення: 14.06.2025).

35) Comparing to Zenject | VContainer. About | VContainer. URL: https://vcontainer.hadashikick.jp/comparing/comparing-to-zenject (дата звернення: 14.06.2025).

36) GeeksforGeeks. Difference between MVP and MVVM architecture pattern in android - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/difference-between-mvp-and-mvvm-architecture-pattern-in-android/ (дата звернення: 14.06.2025).

37) Android architecture patterns: MVC vs MVVM vs MVP. daily.dev | Where developers suffer together. URL: https://daily.dev/blog/android-architecture-patterns-mvc-vs-mvvm-vs-mvp (дата звернення: 14.06.2025).

38) Testing for MVP: does it make a difference? - QA madness. QA Madness. URL: https://www.qamadness.com/testing-for-mvp/ (дата звернення: 14.06.2025).

39) Avoiding garbage collector performance spikes in Unity. Embrace. URL: https://embrace.io/blog/garbage-collector-spikes-unity/ (дата звернення: 14.06.2025).

40) Object pool · optimization patterns · game programming patterns. Game Programming Patterns. URL: https://gameprogrammingpatterns.com/object-pool.html (дата звернення: 14.06.2025).

41) Memory management on mobile devices / K. Sareen та ін. Steve Blackburn. URL: https://www.steveblackburn.org/pubs/papers/android-ismm-2024.pdf (дата звернення: 14.06.2025).

42) General data protection regulation (GDPR) – legal text. General Data Protection Regulation (GDPR). URL: https://gdpr-info.eu/ (дата звернення: 14.06.2025).

43) Improve your .NET code quality with NDepend. NDepend. URL: https://www.ndepend.com/ (дата звернення: 14.06.2025).

44) Unity - Manual: special folders and script compilation order. Unity - Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/6000.1/Documentation/Manual/script-compile-order-folders.html (date of access: 14.06.2025).

45) Class TextMeshProUGUI. Unity - Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/Packages/com.unity.textmeshpro@1.0/api/TMPro.TextMeshProUGUI.html (дата звернення: 14.06.2025).

46) Actions | input system | 1.14.0. Unity - Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.14/manual/Actions.html (дата звернення: 14.06.2025).

47) Cinemachine package | cinemachine | 3.1.4. Unity - Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/Packages/com.unity.cinemachine@3.1/manual/index.html (дата звернення: 14.06.2025).

48) How code metrics help identify risks - Visual Studio (Windows). Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022 (дата звернення: 14.06.2025).

49) Code metrics - Maintainability index range and meaning - Visual Studio (Windows). Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning?view=vs-2022 (дата звернення: 14.06.2025).

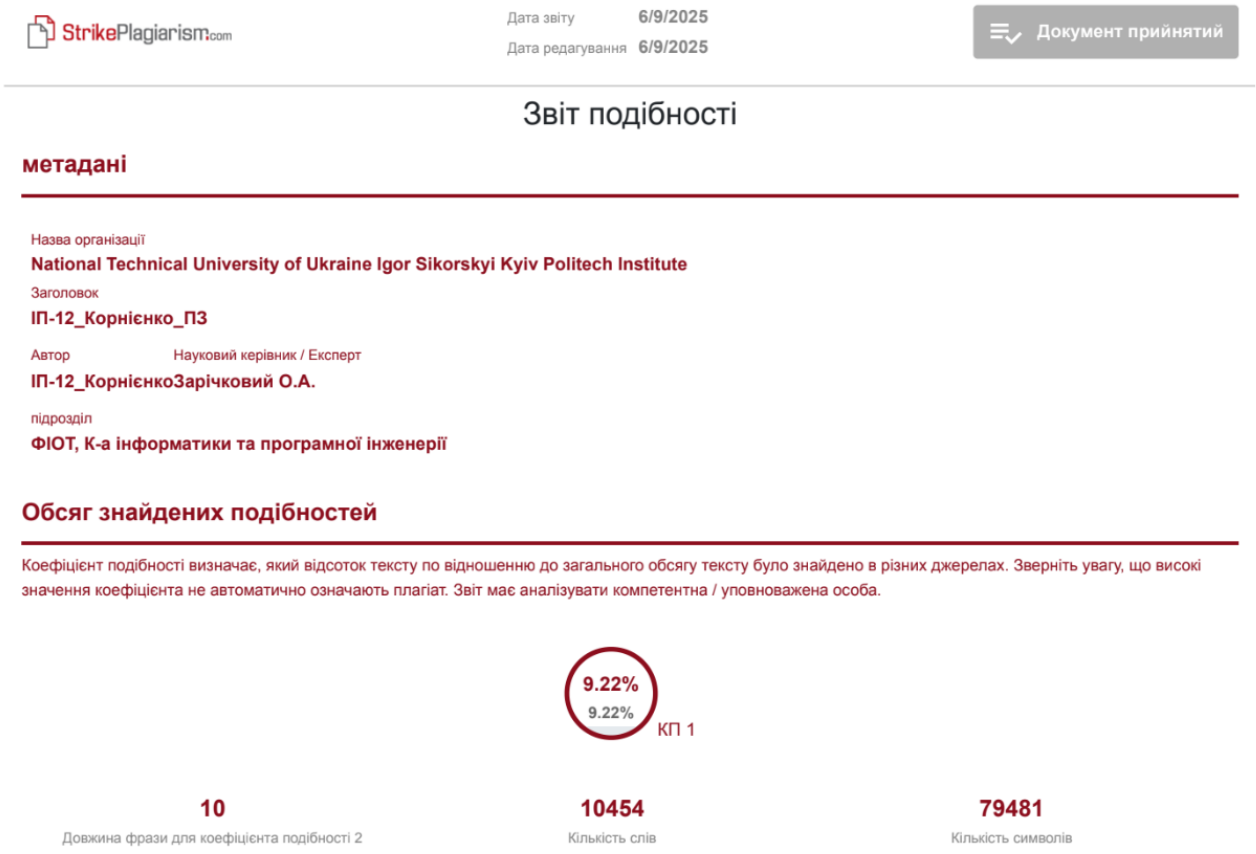
50) Android apps on google play. Android Apps on Google Play. URL: https://play.google.com/ (дата звернення: 14.06.2025).

51) Unity - manual: build your application for android. Unity - Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/6000.1/Documentation/Manual/android-BuildProcess.html (дата звернення: 14.06.2025).

52) Console G. P. Internal testing | google play console. Android Apps on Google Play. URL: https://play.google.com/console/about/internal-testing/ (дата звернення: 14.06.2025).

ДОДАТКИ

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



StrikePlagiarism.com

Дата звіту 6/9/2025
Дата редагування 6/9/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-12_Корнієнко_ПЗ
Автор Науковий керівник / Експерт
ІП-12_КорнієнкоЗарічковий О.А.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

9.22%
9.22%
КП 1

10	10454	79481
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

Мобільна гра в жанрі Shooter з елементами RogueLike

Текст програми

КПІ.ІП-1214.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ЗАРІЧКОВИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Валерій КОРНІЄНКО

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/vakor03/Diploma_Project

Файл MainMenuPresenter.cs
Реалізація функціональної вимоги «Меню та навігація»

```
using _Project.Infrastructure.MVP.Core;
using _Project.Scripts.Infrastructure.StateMachines;
using _Project.Scripts.Infrastructure.StateMachines.GlobalStates;
using JetBrains.Annotations;

namespace _Project.Features.UIModule.MainMenuUI {
    [PublicAPI]
    public class MainMenuPresenter : PresenterBehaviour<MainMenuViewBase> {
        private readonly GlobalStateMachine _globalStateMachine;

        public MainMenuPresenter(GlobalStateMachine globalStateMachine) =>
            _globalStateMachine = globalStateMachine;

        public override void OnViewSet() {
            View.OnPlayButtonClicked += OnPlayButtonClicked;
            View.OnExitButtonClicked += OnExitButtonClicked;
        }

        public override void OnDisposed() {
            View.OnPlayButtonClicked -= OnPlayButtonClicked;
            View.OnExitButtonClicked -= OnExitButtonClicked;
        }

        private void OnPlayButtonClicked() =>
            _globalStateMachine.Enter<GameplayState>();

        private void OnExitButtonClicked() =>
            _globalStateMachine.Enter<QuitGameState>();
    }
}
```

Файл MainMenuViewBase.cs
Реалізація функціональної вимоги «Меню та навігація»

```
using System;
using _Project.Infrastructure.MVP.Core;

namespace _Project.Features.UIModule.MainMenuUI {
    public abstract class MainMenuViewBase : ViewBehaviour {
        public event Action OnPlayButtonClicked;
        public event Action OnExitButtonClicked;

        protected void InvokeOnPlayButtonClicked() =>
            OnPlayButtonClicked?.Invoke();

        protected void InvokeOnExitButtonClicked() =>
            OnExitButtonClicked?.Invoke();
    }
}
```

Файл MainMenuView.cs
Реалізація функціональної вимоги «Меню та навігація»

```
using UnityEngine;
using UnityEngine.UI;

namespace _Project.Features.UIModule.MainMenuUI {
    public class MainMenuView : MainMenuViewBase {
        [SerializeField] private Button _playButton;
        [SerializeField] private Button _exitButton;

        private void OnEnable() {
            _playButton.onClick.AddListener(InvokeOnPlayButtonClicked);
            _exitButton.onClick.AddListener(InvokeOnExitButtonClicked);
        }

        private void OnDisable() {
            _playButton.onClick.RemoveListener(InvokeOnPlayButtonClicked);
            _exitButton.onClick.RemoveListener(InvokeOnExitButtonClicked);
        }
    }
}
```

Файл DungeonGeneratorService.cs
Реалізація функціональної вимоги «Процедурна генерація рівня»

```
using System.Collections.Generic;
using System.Linq;
using _Project.Features.MapGeneration.BSP;
using _Project.Features.MapGeneration.CA;
using _Project.Features.MapGeneration.Drukard;
using _Project.Features.MapGeneration.Matrix;
using _Project.Features.MapGeneration.RoomConnections;
using _Project.Features.MapGeneration.Tagging;
using _Project.Global.Collections;
using UnityEngine;

namespace _Project.Features.MapGeneration {
    public class DungeonGeneratorService : IDungeonGeneratorService {
        private readonly IMatrixFactory _matrixFactory;
        private readonly IMacroLayoutDungeonGenerationService
_macroLayoutDungeonGenerationService;
        private readonly ICaveRoomCarveService _caveRoomCarveService;
        private readonly IRoomConnectorService _roomConnectorService;
        private readonly ICorridorGeneratorService _corridorGeneratorService;
        private readonly IDungeonTagService _dungeonTagService;

        public DungeonGeneratorService(IMatrixFactory matrixFactory,
                                       IMacroLayoutDungeonGenerationService
macroLayoutDungeonGenerationService,
                                       ICaveRoomCarveService
caveRoomCarveService, IRoomConnectorService roomConnectorService,
                                       ICorridorGeneratorService
corridorGeneratorService, IDungeonTagService dungeonTagService) {
            _matrixFactory = matrixFactory;
            _macroLayoutDungeonGenerationService =
macroLayoutDungeonGenerationService;
            _caveRoomCarveService = caveRoomCarveService;
            _roomConnectorService = roomConnectorService;
            _corridorGeneratorService = corridorGeneratorService;
            _dungeonTagService = dungeonTagService;
```

```
        }

        public Dungeon GenerateDungeon(DungeonGenerationConfiguration config)
{
            Matrix<BlockType> dungeonMatrix =
_matrixFactory.CreateMatrix<BlockType>(config.DungeonSize.x,
config.DungeonSize.y);
            List<RectInt> macroGroup =
_macroLayoutDungeonGenerationService.Generate(config.DungeonSize,
config.BspDungeonGeneratorParams);
            List<Room> rooms = macroGroup.Select(room =>
_caveRoomCarveService.CarveRoom(dungeonMatrix, room,
config.CACaveRoomParams))
                .Where(room => room.Cells.Count > 0).ToList();

            List<(Vector2Int start, Vector2Int end)> connections =
_roomConnectorService.GetConnections(rooms);
            List<Tunnel> tunnels = connections.ConvertAll(tunnelEnds =>
                _corridorGeneratorService.CarveCorridor(dungeonMatrix,
tunnelEnds.start, tunnelEnds.end, config.CorridorParams));

            Dungeon dungeon = new Dungeon() {
                Matrix = dungeonMatrix,
                Rooms = rooms,
                Tunnels = tunnels,
            };

            DungeonTags dungeonTags =
_dungeonTagService.TagAllRegions(dungeon, GlobalPlaceTagRules(),
MacroTagRules(), MicroTagRules());

            dungeon.Tags = dungeonTags;

            return dungeon;
        }

        private PriorityList<IGlobalPlaceTagRule> GlobalPlaceTagRules() {
            PriorityList<IGlobalPlaceTagRule> rules = new();
            rules.Add(new InitialRoomGlobalPlaceTagRule(), 10);
            rules.Add(new DefaultRoomGlobalPlaceTagRule(), 0);
            rules.Add(new HorizontalTunnelGlobalPlaceTagRule(), 0);
            rules.Add(new VerticalTunnelGlobalPlaceTagRule(), 0);
            return rules;
        }

        private PriorityList<IMicroTagRule> MicroTagRules() {
            PriorityList<IMicroTagRule> rules = new();
            rules.Add(new PlatformMicroTagRule(), 11);
            rules.Add(new PlayerSpawnMicroTagRule(), 10);
            rules.Add(new EnemySpawnMicroTagRule(), 9);
            rules.Add(new BigFloorDecorationMicroTagRule(), 8);
            rules.Add(new SmallFloorDecorationMicroTagRule(), 7);
            rules.Add(new CeilingDecorationMicroTagRule(), 6);
            rules.Add(new WallDecorationMicroTagRule(), 5);
            return rules;
        }

        private PriorityList<IMacroTagRule> MacroTagRules() {
            PriorityList<IMacroTagRule> rules = new();
            rules.Add(new PlatformMacroTagRule(), 20);
            rules.Add(new FloorMacroTagRule(), 10);
            return rules;
        }
    }
```

```
    public enum BlockType {
        Wall = 0,
        EmptySpace = 1,
        Platform = 2,
    }
}
```

Файл BSPDungeonGenerator.cs

Реалізація функціональної вимоги «Процедурна генерація рівня»

```
using System.Collections.Generic;
using _Project.Features.MapGeneration.Matrix;
using _Project.Features.SeedModule;
using UnityEngine;

namespace _Project.Features.MapGeneration.BSP {
    public class BSPDungeonGenerator : IMacroLayoutDungeonGenerationService {
        private readonly ISeedService _seedService;

        private List<RectInt> Rooms { get; set; }

        private Vector2 _minLeafSize;

        public BSPDungeonGenerator(ISeedService seedService) =>
            _seedService = seedService;

        public List<RectInt> Generate(Vector2Int matrixSize,
BSPDungeonGeneratorParams @params) {
            _minLeafSize = @params.minLeafSize;
            Rooms = new List<RectInt>();

            BSPNode root = new(new RectInt(0, 0, matrixSize.x,
matrixSize.y));
            List<BSPNode> nodes = new() { root };

            for (int i = 0; i < nodes.Count; i++) {
                BSPNode node = nodes[i];
                if (node.Rect.width > _minLeafSize.x * 2 || node.Rect.height
> _minLeafSize.y * 2)
                    if (Split(node)) {
                        nodes.Add(node.Left);
                        nodes.Add(node.Right);
                    }
            }

            foreach (BSPNode leaf in nodes)
                if (leaf.IsLeaf)
                    CreateRoom(leaf, @params.minRoomSize,
@params.offsetFromBorders, @params.ShrinkageFactor);

            return Rooms;
        }

        private bool Split(BSPNode node) {
            bool horizontal = _seedService.GetRandom().NextDouble() > 0.5;
            if (node.Rect.width > node.Rect.height && node.Rect.height /
(float)node.Rect.width < 0.5f)
                horizontal = false;
            else if (node.Rect.height > node.Rect.width && node.Rect.width /
(float)node.Rect.height < 0.5f)
                horizontal = true;
```

```
            float minSize = horizontal ? _minLeafSize.y : _minLeafSize.x;
            int max = (horizontal ? node.Rect.height : node.Rect.width) -
Mathf.FloorToInt(minSize);

            if (max <= minSize)
                return false;

            int split =
_seedService.GetRandom().Next(Mathf.FloorToInt(minSize), max);

            if (horizontal) {
                node.Left = new BSPNode(new RectInt(node.Rect.x, node.Rect.y,
node.Rect.width, split));
                node.Right = new BSPNode(new RectInt(node.Rect.x, node.Rect.y
+ split, node.Rect.width, node.Rect.height - split));
            }
            else {
                node.Left = new BSPNode(new RectInt(node.Rect.x, node.Rect.y,
split, node.Rect.height));
                node.Right = new BSPNode(new RectInt(node.Rect.x + split,
node.Rect.y, node.Rect.width - split, node.Rect.height));
            }

            return true;
        }

        private void CreateRoom(BSPNode leaf, Vector2 minRoomSize, int
offset, float shrinkageRate) {
            int roomW = _seedService.GetRandom().Next((int)(leaf.Rect.width *
(1 - shrinkageRate)), leaf.Rect.width);
            int roomH = _seedService.GetRandom().Next((int)(leaf.Rect.height
* (1 - shrinkageRate)), leaf.Rect.height);

            roomW = Mathf.Max(roomW, Mathf.FloorToInt(minRoomSize.x));
            roomH = Mathf.Max(roomH, Mathf.FloorToInt(minRoomSize.y));

            int roomX = _seedService.GetRandom().Next(leaf.Rect.x + offset,
leaf.Rect.x + leaf.Rect.width - roomW - offset);
            int roomY = _seedService.GetRandom().Next(leaf.Rect.y + offset,
leaf.Rect.y + leaf.Rect.height - roomH - offset);

            RectInt room = new RectInt(roomX, roomY, roomW, roomH);
            leaf.Room = room;
            Rooms.Add(room);
        }
    }
}
```

Файл CellularAutomataCaveGeneratorService.cs

Реалізація функціональної вимоги «Процедурна генерація рівня»

```
using System.Collections.Generic;
using System.Linq;
using _Project.Features.MapGeneration.BSP;
using UnityEngine;
using _Project.Features.MapGeneration.Matrix;
using _Project.Features.SeedModule;

namespace _Project.Features.MapGeneration.CA {
    public class CellularAutomataCaveGeneratorService : ICaveRoomCarveService
{
        private readonly ISeedService _seedService;
```

```
        private static readonly int[,] NeighborDirections8 = new int[8, 2] {
            { -1, -1 }, { -1, 0 }, { -1, 1 }, { 0, -1 },
            { 0, 1 }, { 1, -1 }, { 1, 0 }, { 1, 1 },
        };

        private static readonly int[,] Directions4 = new int[4, 2] {
            { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 }
        };

        public CellularAutomataCaveGeneratorService(ISeedService seedService)
=>
            _seedService = seedService;

        public Room CarveRoom(Matrix<BlockType> matrix, RectInt region,
CAConfig config) {
            BlockType[,] cells = InitializeCells(region, config);

            SmoothCells(ref cells, config);
            List<Vector2Int> largest = GetLargestRegion(cells);
            BlockType[,] mask = CreateRegionMask(cells, largest);
            BlitToMatrix(matrix, region, mask, config.offsetFromBorders);

            Room room = new Room {
                PartitionBounds = region,
                RoomBounds = CalculateRoomBounds(region, config, largest),
                Cells = largest.ConvertAll(positionLocal => positionLocal +
new Vector2Int(region.x + config.offsetFromBorders, region.y +
config.offsetFromBorders)),
            };

            return room;
        }

        private RectInt CalculateRoomBounds(RectInt region, CAConfig config,
List<Vector2Int> largest) {
            int minX = largest.Min(el => el.x);
            int minY = largest.Min(el => el.y);
            int maxX = largest.Max(el => el.x);
            int maxY = largest.Max(el => el.y);

            int width = maxX - minX + 1;
            int height = maxY - minY + 1;
            int x = minX + region.x + config.offsetFromBorders;
            int y = minY + region.y + config.offsetFromBorders;
            RectInt roomBounds = new RectInt(x, y, width, height);
            return roomBounds;
            // return new(region.x + config.offsetFromBorders, region.y +
config.offsetFromBorders,
                // region.width - config.offsetFromBorders * 2, region.height
- config.offsetFromBorders * 2);
        }

        private BlockType[,] InitializeCells(RectInt region, CAConfig config)
{
            int offset = config.offsetFromBorders;
            int width = region.width - offset * 2;
            int height = region.height - offset * 2;
            BlockType[,] cells = new BlockType[width, height];
            System.Random rng = _seedService.GetRandom();

            for (int x = 0; x < width; x++) {
                for (int y = 0; y < height; y++) {
                    double r = rng.NextDouble();
                    cells[x, y] = (r < config.fillProbability) ?
```

```
BlockType.Wall : BlockType.EmptySpace;
                }
            }

            return cells;
        }

        private void SmoothCells(ref BlockType[,] cells, CAConfig config) {
            int width = cells.GetLength(0);
            int height = cells.GetLength(1);

            for (int step = 0; step < config.steps; step++) {
                BlockType[,] newCells = new BlockType[width, height];

                for (int x = 0; x < width; x++) {
                    for (int y = 0; y < height; y++) {
                        int wallCount = CountWallNeighbors(cells, x, y,
width, height);
                        if (wallCount > config.birthLimit)
                            newCells[x, y] = BlockType.Wall;
                        else if (wallCount < config.deathLimit)
                            newCells[x, y] = BlockType.EmptySpace;
                        else
                            newCells[x, y] = cells[x, y];
                    }
                }

                cells = newCells;
            }
        }

        private List<Vector2Int> GetLargestRegion(BlockType[,] cells) {
            int width = cells.GetLength(0);
            int height = cells.GetLength(1);
            bool[,] visited = new bool[width, height];
            List<List<Vector2Int>> regions = new List<List<Vector2Int>>();

            for (int x = 0; x < width; x++) {
                for (int y = 0; y < height; y++) {
                    if (!visited[x, y] && cells[x, y] ==
BlockType.EmptySpace) {
                        List<Vector2Int> region = FloodFill(cells, visited,
x, y, width, height);
                        regions.Add(region);
                    }
                }
            }

            List<Vector2Int> largest = new List<Vector2Int>();
            int maxSize = 0;
            foreach (List<Vector2Int> region in regions) {
                int size = region.Count;
                if (size > maxSize) {
                    maxSize = size;
                    largest = region;
                }
            }

            return largest;
        }

        private BlockType[,] CreateRegionMask(BlockType[,] cells,
List<Vector2Int> region) {
            int width = cells.GetLength(0);
```

```
            int height = cells.GetLength(1);
            BlockType[,] mask = new BlockType[width, height];

            foreach (Vector2Int cell in region)
                mask[cell.x, cell.y] = BlockType.EmptySpace;

            return mask;
        }

        private void BlitToMatrix(Matrix<BlockType> matrix, RectInt region,
BlockType[,] mask, int offset) {
            int width = mask.GetLength(0);
            int height = mask.GetLength(1);

            for (int x = 0; x < width; x++)
            for (int y = 0; y < height; y++) {
                int globalX = region.x + offset + x;
                int globalY = region.y + offset + y;

                matrix[globalX, globalY] = mask[x, y] == BlockType.EmptySpace
? BlockType.EmptySpace : BlockType.Wall;
            }
        }

        private int CountWallNeighbors(BlockType[,] cells, int x, int y, int
width, int height) {
            int count = 0;
            int length = NeighborDirections8.GetLength(0);
            for (int i = 0; i < length; i++) {
                int dx = NeighborDirections8[i, 0];
                int dy = NeighborDirections8[i, 1];
                int nx = x + dx;
                int ny = y + dy;
                if (nx < 0 || ny < 0 || nx >= width || ny >= height ||
cells[nx, ny] == BlockType.Wall) {
                    count++;
                }
            }
            // count += cells[x-1,y] + cells[x+1,y];

            return count;
        }

        private List<Vector2Int> FloodFill(BlockType[,] cells, bool[,]
visited, int startX, int startY, int width, int height) {
            List<Vector2Int> region = new List<Vector2Int>();
            Queue<Vector2Int> queue = new Queue<Vector2Int>();
            visited[startX, startY] = true;
            queue.Enqueue(new Vector2Int(startX, startY));

            int dirCount = Directions4.GetLength(0);
            while (queue.Count > 0) {
                Vector2Int cell = queue.Dequeue();
                region.Add(cell);

                for (int i = 0; i < dirCount; i++) {
                    int dx = Directions4[i, 0];
                    int dy = Directions4[i, 1];
                    int nx = cell.x + dx;
                    int ny = cell.y + dy;
                    if (nx >= 0 && ny >= 0 && nx < width && ny < height &&
!visited[nx, ny] && cells[nx, ny] == BlockType.EmptySpace) {
                        visited[nx, ny] = true;
                        queue.Enqueue(new Vector2Int(nx, ny));
```

```
                    }
                }
            }

            return region;
        }
    }
}
```

Файл DrunkardCorridorGeneratorService.cs

Реалізація функціональної вимоги «Процедурна генерація рівня»

```
using System.Collections.Generic;
using System.Linq;
using _Project.Features.MapGeneration.BSP;
using _Project.Features.MapGeneration.Matrix;
using _Project.Features.SeedModule;
using UnityEngine;

namespace _Project.Features.MapGeneration.Drukard {
    public class DrunkardCorridorGeneratorService : ICorridorGeneratorService
    {
        private readonly ISeedService _seedService;

        public DrunkardCorridorGeneratorService(ISeedService seedService) =>
            _seedService = seedService;

        public Tunnel CarveCorridor(Matrix<BlockType> matrix, Vector2Int
start, Vector2Int end, CorridorConfig config)
        {
            HashSet<Vector2Int> cells = new HashSet<Vector2Int>();
            System.Random rng = _seedService.GetRandom();
            Vector2Int current = start;
            CarveCellBlock(matrix, current, config.CorridorWidth, cells);

            while (!IsAtTarget(current, end))
            {
                Vector2Int step = StepTowardsTarget(current, end, rng,
config.WanderChance);
                current = new Vector2Int(current.x + step.x, current.y +
step.y);
                CarveCellBlock(matrix, current, config.CorridorWidth, cells);
            }

            return new Tunnel {
                Start = start,
                End = end,
                Cells = cells.ToList()
            };
        }

        private Vector2Int StepTowardsTarget(Vector2Int current, Vector2Int
target, System.Random rng, float wanderChance) {
            int dx = target.x - current.x;
            int dy = target.y - current.y;
            bool shouldWander = rng.NextDouble() < wanderChance;

            if (shouldWander) {
                return GetRandomWanderStep(rng);
            }

            return GetBiasedStep(dx, dy);
        }
```

```
        private Vector2Int GetRandomWanderStep(System.Random rng) {
            bool horizontal = rng.NextDouble() < 0.5;
            int stepX = 0;
            int stepY = 0;

            if (horizontal) {
                stepX = rng.Next(0, 2) == 0 ? 1 : -1;
            } else {
                stepY = rng.Next(0, 2) == 0 ? 1 : -1;
            }

            return new Vector2Int(stepX, stepY);
        }

        private Vector2Int GetBiasedStep(int dx, int dy) {
            if (Mathf.Abs(dx) >= Mathf.Abs(dy)) {
                int xStep = dx > 0 ? 1 : -1;
                return new Vector2Int(xStep, 0);
            }

            int yStep = dy > 0 ? 1 : -1;
            return new Vector2Int(0, yStep);
        }

        private void CarveCellBlock(Matrix<BlockType> matrix, Vector2Int
center, int width, HashSet<Vector2Int> corridorCells) {
            int half = width / 2;
            int startOffset = -half;
            int endOffset;

            if (width % 2 == 0) {
                endOffset = half - 1;
            } else {
                endOffset = half;
            }

            for (int ox = startOffset; ox <= endOffset; ox++) {
                for (int oy = startOffset; oy <= endOffset; oy++) {
                    int x = center.x + ox;
                    int y = center.y + oy;

                    if (x >= 0 && y >= 0 && x < matrix.Width && y <
matrix.Height) {
                        corridorCells.Add(new(x, y));
                        matrix[x, y] = BlockType.EmptySpace;
                    }
                }
            }
        }

        private bool IsAtTarget(Vector2Int current, Vector2Int target)
        {
            return current.x == target.x && current.y == target.y;
        }
    }
}
```

Файл DungeonTagService.cs

Реалізація функціональної вимоги «Процедурна генерація рівня»

```
using _Project.Features.MapGeneration.BSP;
using _Project.Features.MapGeneration.Tagging.TagAppliers;
```

```
using _Project.Global.Collections;

namespace _Project.Features.MapGeneration.Tagging {
    public class DungeonTagService : IDungeonTagService {
        private readonly IGlobalPlaceTagApplier _globalPlaceTagApplier;
        private readonly IMacroTagApplier _macroTagApplier;
        private readonly IMicroTagApplier _microTagApplier;

        public DungeonTagService(
            IGlobalPlaceTagApplier globalPlaceTagApplier,
            IMacroTagApplier macroTagApplier,
            IMicroTagApplier microTagApplier) {
            _globalPlaceTagApplier = globalPlaceTagApplier;
            _macroTagApplier = macroTagApplier;
            _microTagApplier = microTagApplier;
        }

        public DungeonTags TagAllRegions(Dungeon dungeon,
                                         PriorityList<IGlobalPlaceTagRule>
globalPlaceTagRules,
                                         PriorityList<IMacroTagRule>
macroTagRules,
                                         PriorityList<IMicroTagRule>
microTagRules) {
            DungeonTags dungeonTags = new();

            SetContextForRules(globalPlaceTagRules, macroTagRules,
microTagRules, dungeon);

            _globalPlaceTagApplier.Apply(globalPlaceTagRules, dungeon,
dungeonTags);
            _macroTagApplier.Apply(macroTagRules, dungeon, dungeonTags);
            _microTagApplier.Apply(microTagRules, dungeon, dungeonTags);

            return dungeonTags;
        }

        private void SetContextForRules(PriorityList<IGlobalPlaceTagRule>
globalRules,
                                        PriorityList<IMacroTagRule>
macroRules,
                                        PriorityList<IMicroTagRule>
microRules,
                                        Dungeon dungeon) {
            foreach (IGlobalPlaceTagRule rule in globalRules)
                rule.Context = dungeon;
            foreach (IMacroTagRule rule in macroRules)
                rule.Context = dungeon;
            foreach (IMicroTagRule rule in microRules)
                rule.Context = dungeon;
        }
    }
}
```

Файл GlobalPlaceTagApplier.cs

Реалізація функціональної вимоги «Процедурна генерація рівня»

```
using _Project.Extensions.EnumerableExtensions;
using _Project.Features.MapGeneration.BSP;
using _Project.Features.SeedModule;
using _Project.Global.Collections;

namespace _Project.Features.MapGeneration.Tagging.TagAppliers {
```

```
    public class GlobalPlaceTagApplier : IGlobalPlaceTagApplier {
        private readonly ISeedService _seedService;

        public GlobalPlaceTagApplier(ISeedService seedService) =>
            _seedService = seedService;

        public void Apply(PriorityList<IGlobalPlaceTagRule> rules, Dungeon
dungeon, DungeonTags dungeonTags) {
            foreach (IGlobalPlaceTagRule rule in rules)
                ApplyGlobalTagRule(dungeon, dungeonTags, rule);
        }

        private void ApplyGlobalTagRule(Dungeon dungeon, DungeonTags
dungeonTags, IGlobalPlaceTagRule rule) {
            int appliedCount = 0;

            foreach (Room room in
dungeon.Rooms.InRandomOrder(_seedService.GetRandom())) {
                if (dungeonTags.GetGlobalPlaceTagForRoom(room) ==
GlobalPlaceTag.None && rule.IsValidForRoom(room)) {
                    dungeonTags.AddGlobalPlaceTag(rule.GlobalPlaceTag, room);
                    appliedCount++;
                }

                if (appliedCount >= rule.MaxCount)
                    break;
            }

            if (appliedCount >= rule.MaxCount)
                return;

            foreach (Tunnel tunnel in dungeon.Tunnels) {
                if (dungeonTags.GetGlobalPlaceTagForTunnel(tunnel) ==
GlobalPlaceTag.None && rule.IsValidForTunnel(tunnel)) {
                    dungeonTags.AddGlobalPlaceTag(rule.GlobalPlaceTag,
tunnel);
                    appliedCount++;
                }

                if (appliedCount >= rule.MaxCount)
                    break;
            }
        }
    }
}
```

Файл MacroTagApplier.cs

Реалізація функціональної вимоги «Процедурна генерація рівня»

```
using System.Collections.Generic;
using _Project.Extensions.EnumerableExtensions;
using _Project.Features.MapGeneration.BSP;
using _Project.Features.SeedModule;
using _Project.Global.Collections;
using UnityEngine;

namespace _Project.Features.MapGeneration.Tagging.TagAppliers {
    public class MacroTagApplier : IMacroTagApplier {
        private readonly ISeedService _seedService;

        public MacroTagApplier(ISeedService seedService) =>
            _seedService = seedService;
```

```
        public void Apply(PriorityList<IMacroTagRule> rules, Dungeon dungeon,
DungeonTags dungeonTags) {
            Dictionary<GlobalPlaceTag, PriorityList<IMacroTagRule>>
rulesByGlobalTag = new();
            PriorityList<IMacroTagRule> rulesForAllTags = new();

            foreach (IMacroTagRule rule in rules) {
                int? priority = rules.GetPriority(rule);
                if (!priority.HasValue) continue;

                if (rule.GlobalPlaceTagFilter.AllowedTags.Count == 0) {
                    rulesForAllTags.Add(rule, priority.Value);
                }
                else {
                    foreach (GlobalPlaceTag tag in
rule.GlobalPlaceTagFilter.AllowedTags) {
                        if (!rulesByGlobalTag.ContainsKey(tag))
                            rulesByGlobalTag[tag] = new
PriorityList<IMacroTagRule>();
                        rulesByGlobalTag[tag].Add(rule, priority.Value);
                    }
                }
            }

            foreach (Room room in
dungeon.Rooms.InRandomOrder(_seedService.GetRandom())) {
                GlobalPlaceTag roomGlobalTag =
dungeonTags.GetGlobalPlaceTagForRoom(room);

                if (rulesByGlobalTag.TryGetValue(roomGlobalTag, out
PriorityList<IMacroTagRule> specificRules)) {
                    foreach (IMacroTagRule rule in specificRules) {
                        ApplyMacroTagToRoom(rule, room, roomGlobalTag,
dungeonTags);
                    }
                }

                foreach (IMacroTagRule rule in rulesForAllTags) {
                    ApplyMacroTagToRoom(rule, room, roomGlobalTag,
dungeonTags);
                }
            }

            foreach (Tunnel tunnel in
dungeon.Tunnels.InRandomOrder(_seedService.GetRandom())) {
                GlobalPlaceTag tunnelGlobalTag =
dungeonTags.GetGlobalPlaceTagForTunnel(tunnel);

                if (rulesByGlobalTag.TryGetValue(tunnelGlobalTag, out
PriorityList<IMacroTagRule> specificRules)) {
                    foreach (IMacroTagRule rule in specificRules) {
                        ApplyMacroTagToTunnel(rule, tunnel, tunnelGlobalTag,
dungeonTags);
                    }
                }

                foreach (IMacroTagRule rule in rulesForAllTags) {
                    ApplyMacroTagToTunnel(rule, tunnel, tunnelGlobalTag,
dungeonTags);
                }
            }
        }

        private void ApplyMacroTagToRoom(IMacroTagRule rule, Room room,
```

```
GlobalPlaceTag globalPlaceTag, DungeonTags dungeonTags) {
            if (!rule.GlobalPlaceTagFilter.IsAllowed(globalPlaceTag))
                return;

            int appliedCount = 0;

            foreach (Vector2Int position in
room.Cells.InRandomOrder(_seedService.GetRandom())) {
                if (dungeonTags.GetMacroTag(position) != MacroTag.None)
                    continue;

                if (rule.IsValidForPosition(position, room, globalPlaceTag,
dungeonTags)) {
                    dungeonTags.AddMacroTag(rule.MacroTag, position);
                    appliedCount++;
                }

                if (appliedCount >= rule.MaxCount)
                    break;
            }
        }

        private void ApplyMacroTagToTunnel(IMacroTagRule rule, Tunnel tunnel,
GlobalPlaceTag globalPlaceTag, DungeonTags dungeonTags) {
            if (!rule.GlobalPlaceTagFilter.IsAllowed(globalPlaceTag))
                return;

            int appliedCount = 0;

            foreach (Vector2Int position in
tunnel.Cells.InRandomOrder(_seedService.GetRandom())) {
                if (dungeonTags.GetMacroTag(position) != MacroTag.None)
                    continue;

                if (rule.IsValidForPosition(position, tunnel, globalPlaceTag,
dungeonTags)) {
                    dungeonTags.AddMacroTag(rule.MacroTag, position);
                    appliedCount++;
                }

                if (appliedCount >= rule.MaxCount)
                    break;
            }
        }
    }
}
```

Файл MicroTagApplier.cs

Реалізація функціональної вимоги «Процедурна генерація рівня»

```
using System.Collections.Generic;
using _Project.Extensions.EnumerableExtensions;
using _Project.Features.MapGeneration.BSP;
using _Project.Features.SeedModule;
using _Project.Global.Collections;
using UnityEngine;

namespace _Project.Features.MapGeneration.Tagging.TagAppliers {
    public class MicroTagApplier : IMicroTagApplier {
        private readonly ISeedService _seedService;

        public MicroTagApplier(ISeedService seedService) =>
            _seedService = seedService;
```

```
        public void Apply(PriorityList<IMicroTagRule> rules, Dungeon dungeon,
DungeonTags dungeonTags) {
            Dictionary<IMicroTagRule, MicroTagRuleCounts> ruleCounts =
InitializeRuleCounts(rules);
            RuleGroups ruleGroups = GroupRulesByFilters(rules);

            ApplyToRooms(dungeon, dungeonTags, ruleGroups, ruleCounts);
            ApplyToTunnels(dungeon, dungeonTags, ruleGroups, ruleCounts);
        }

        private Dictionary<IMicroTagRule, MicroTagRuleCounts>
InitializeRuleCounts(PriorityList<IMicroTagRule> rules) {
            Dictionary<IMicroTagRule, MicroTagRuleCounts> ruleCounts = new();

            foreach (IMicroTagRule rule in rules)
                ruleCounts[rule] = new MicroTagRuleCounts();

            return ruleCounts;
        }

        private RuleGroups GroupRulesByFilters(PriorityList<IMicroTagRule>
rules) {
            RuleGroups groups = new();

            foreach (IMicroTagRule rule in rules) {
                int? priority = rules.GetPriority(rule);
                if (!priority.HasValue) continue;

                CategorizeRule(rule, priority.Value, groups);
            }

            return groups;
        }

        private void CategorizeRule(IMicroTagRule rule, int priority,
RuleGroups groups) {
            bool hasGlobalFilter =
rule.GlobalPlaceTagFilter.AllowedTags.Count > 0;
            bool hasMacroFilter = rule.MacroTagFilter.AllowedTags.Count > 0;

            if (!hasGlobalFilter && !hasMacroFilter) {
                AddRuleToAllTags(rule, priority, groups);
            }
            else if (hasGlobalFilter && hasMacroFilter) {
                AddRuleWithBothFilters(rule, priority, groups);
            }
            else if (hasGlobalFilter) {
                AddRuleWithGlobalFilter(rule, priority, groups);
            }
            else {
                AddRuleWithMacroFilter(rule, priority, groups);
            }
        }

        private void AddRuleToAllTags(IMicroTagRule rule, int priority,
RuleGroups groups) {
            groups.RulesForAllTags.Add(rule, priority);
        }

        private void AddRuleWithBothFilters(IMicroTagRule rule, int priority,
RuleGroups groups) {
            foreach (GlobalPlaceTag globalTag in
rule.GlobalPlaceTagFilter.AllowedTags) {
```

```
                foreach (MacroTag macroTag in
rule.MacroTagFilter.AllowedTags) {
                    (GlobalPlaceTag, MacroTag) key = (globalTag, macroTag);
                    EnsureKeyExists(key, groups.RulesByTags);
                    groups.RulesByTags[key].Add(rule, priority);
                }
            }
        }

        private void AddRuleWithGlobalFilter(IMicroTagRule rule, int
priority, RuleGroups groups) {
            foreach (GlobalPlaceTag globalTag in
rule.GlobalPlaceTagFilter.AllowedTags) {
                EnsureKeyExists(globalTag, groups.RulesByGlobalTag);
                groups.RulesByGlobalTag[globalTag].Add(rule, priority);
            }
        }

        private void AddRuleWithMacroFilter(IMicroTagRule rule, int priority,
RuleGroups groups) {
            foreach (MacroTag macroTag in rule.MacroTagFilter.AllowedTags) {
                EnsureKeyExists(macroTag, groups.RulesByMacroTag);
                groups.RulesByMacroTag[macroTag].Add(rule, priority);
            }
        }

        private void EnsureKeyExists<TKey>(TKey key, Dictionary<TKey,
PriorityList<IMicroTagRule>> dictionary) {
            if (!dictionary.ContainsKey(key))
                dictionary[key] = new PriorityList<IMicroTagRule>();
        }

        private void ApplyToRooms(Dungeon dungeon, DungeonTags dungeonTags,
RuleGroups ruleGroups,
                                  Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            foreach (Room room in
dungeon.Rooms.InRandomOrder(_seedService.GetRandom()))
                ProcessRoom(room, dungeonTags, ruleGroups, ruleCounts);
        }

        private void ProcessRoom(Room room, DungeonTags dungeonTags,
RuleGroups ruleGroups,
                                 Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            GlobalPlaceTag roomGlobalTag =
dungeonTags.GetGlobalPlaceTagForRoom(room);
            Dictionary<IMicroTagRule, int> localCounts = new();

            foreach (Vector2Int position in
room.Cells.InRandomOrder(_seedService.GetRandom())) {
                if (dungeonTags.GetMicroTag(position) != MicroTag.None)
continue;
                ProcessRoomPosition(position, room, roomGlobalTag,
dungeonTags, ruleGroups, localCounts, ruleCounts);
            }
        }

        private void ProcessRoomPosition(Vector2Int position, Room room,
GlobalPlaceTag roomGlobalTag,
                                         DungeonTags dungeonTags, RuleGroups
ruleGroups,
                                         Dictionary<IMicroTagRule, int>
localCounts,
```

```
                                            Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            MacroTag macroTag = dungeonTags.GetMacroTag(position);

            if (TryApplySpecificRules(position, room, roomGlobalTag,
macroTag, dungeonTags,
                                      ruleGroups, localCounts, ruleCounts))
return;

            if (TryApplyGlobalOnlyRules(position, room, roomGlobalTag,
macroTag, dungeonTags,
                                       ruleGroups, localCounts, ruleCounts))
return;

            if (TryApplyMacroOnlyRules(position, room, roomGlobalTag,
macroTag, dungeonTags,
                                      ruleGroups, localCounts, ruleCounts))
return;

            TryApplyUnfilteredRules(position, room, roomGlobalTag, macroTag,
dungeonTags,
                                    ruleGroups, localCounts, ruleCounts);
        }

        private bool TryApplySpecificRules(Vector2Int position, Room room,
GlobalPlaceTag roomGlobalTag, MacroTag macroTag,
                                           DungeonTags dungeonTags,
RuleGroups ruleGroups,
                                           Dictionary<IMicroTagRule, int>
localCounts,
                                           Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (ruleGroups.RulesByTags.TryGetValue((roomGlobalTag, macroTag),
out PriorityList<IMicroTagRule> combinedRules)) {
                foreach (IMicroTagRule rule in combinedRules) {
                    if (TryApplyMicroTagToPosition(rule, position, room,
roomGlobalTag, macroTag, dungeonTags, localCounts, ruleCounts))
                        return true;
                }
            }
            return false;
        }

        private bool TryApplyGlobalOnlyRules(Vector2Int position, Room room,
GlobalPlaceTag roomGlobalTag, MacroTag macroTag,
                                             DungeonTags dungeonTags,
RuleGroups ruleGroups,
                                             Dictionary<IMicroTagRule, int>
localCounts,
                                             Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (dungeonTags.GetMicroTag(position) != MicroTag.None) return
false;

            if (ruleGroups.RulesByGlobalTag.TryGetValue(roomGlobalTag, out
PriorityList<IMicroTagRule> globalRules)) {
                foreach (IMicroTagRule rule in globalRules) {
                    if (rule.MacroTagFilter.IsAllowed(macroTag) &&
                        TryApplyMicroTagToPosition(rule, position, room,
roomGlobalTag, macroTag, dungeonTags, localCounts, ruleCounts))
                        return true;
                }
            }
            return false;
```

```
        }

        private bool TryApplyMacroOnlyRules(Vector2Int position, Room room,
GlobalPlaceTag roomGlobalTag, MacroTag macroTag,
                                            DungeonTags dungeonTags,
RuleGroups ruleGroups,
                                            Dictionary<IMicroTagRule, int>
localCounts,
                                            Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (dungeonTags.GetMicroTag(position) != MicroTag.None) return
false;

            if (ruleGroups.RulesByMacroTag.TryGetValue(macroTag, out
PriorityList<IMicroTagRule> macroRules)) {
                foreach (IMicroTagRule rule in macroRules) {
                    if (rule.GlobalPlaceTagFilter.IsAllowed(roomGlobalTag) &&
                        TryApplyMicroTagToPosition(rule, position, room,
roomGlobalTag, macroTag, dungeonTags, localCounts, ruleCounts))
                        return true;
                }
            }
            return false;
        }

        private bool TryApplyUnfilteredRules(Vector2Int position, Room room,
GlobalPlaceTag roomGlobalTag, MacroTag macroTag,
                                             DungeonTags dungeonTags,
RuleGroups ruleGroups,
                                             Dictionary<IMicroTagRule, int>
localCounts,
                                             Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (dungeonTags.GetMicroTag(position) != MicroTag.None) return
false;

            foreach (IMicroTagRule rule in ruleGroups.RulesForAllTags) {
                if (TryApplyMicroTagToPosition(rule, position, room,
roomGlobalTag, macroTag, dungeonTags, localCounts, ruleCounts))
                    return true;
            }
            return false;
        }

        private void ApplyToTunnels(Dungeon dungeon, DungeonTags dungeonTags,
RuleGroups ruleGroups,
                                    Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            RuleGroups tunnelGroups =
GroupRulesByFilters(ruleGroups.GetAllRules());

            foreach (Tunnel tunnel in
dungeon.Tunnels.InRandomOrder(_seedService.GetRandom())) {
                ProcessTunnel(tunnel, dungeon, dungeonTags, tunnelGroups,
ruleCounts);
            }
        }

        private void ProcessTunnel(Tunnel tunnel, Dungeon dungeon,
DungeonTags dungeonTags, RuleGroups tunnelGroups,
                                   Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            GlobalPlaceTag tunnelGlobalTag =
dungeonTags.GetGlobalPlaceTagForTunnel(tunnel);
```

```
            Dictionary<IMicroTagRule, int> localCounts = new();

            foreach (Vector2Int position in
tunnel.Cells.InRandomOrder(_seedService.GetRandom())) {
                ProcessTunnelPosition(position, tunnel, tunnelGlobalTag,
dungeonTags, tunnelGroups, localCounts, ruleCounts);
            }
        }

        private void ProcessTunnelPosition(Vector2Int position, Tunnel
tunnel, GlobalPlaceTag tunnelGlobalTag,
                                           DungeonTags dungeonTags,
RuleGroups tunnelGroups,
                                           Dictionary<IMicroTagRule, int>
localCounts,
                                           Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            MacroTag macroTag = dungeonTags.GetMacroTag(position);

            if (TryApplySpecificTunnelRules(position, tunnel,
tunnelGlobalTag, macroTag, dungeonTags,
                                            tunnelGroups, localCounts,
ruleCounts)) return;

            if (TryApplyGlobalOnlyTunnelRules(position, tunnel,
tunnelGlobalTag, macroTag, dungeonTags,
                                              tunnelGroups, localCounts,
ruleCounts)) return;

            if (TryApplyMacroOnlyTunnelRules(position, tunnel,
tunnelGlobalTag, macroTag, dungeonTags,
                                             tunnelGroups, localCounts,
ruleCounts)) return;

            TryApplyUnfilteredTunnelRules(position, tunnel, tunnelGlobalTag,
macroTag, dungeonTags,
                                          tunnelGroups, localCounts,
ruleCounts);
        }

        private bool TryApplySpecificTunnelRules(Vector2Int position, Tunnel
tunnel, GlobalPlaceTag tunnelGlobalTag, MacroTag macroTag,
                                                 DungeonTags dungeonTags,
RuleGroups tunnelGroups,
                                                 Dictionary<IMicroTagRule,
int> localCounts,
                                                 Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (tunnelGroups.RulesByTags.TryGetValue((tunnelGlobalTag,
macroTag), out PriorityList<IMicroTagRule> combinedRules)) {
                foreach (IMicroTagRule rule in combinedRules) {
                    if (TryApplyMicroTagToTunnelPosition(rule, position,
tunnel, tunnelGlobalTag, macroTag, dungeonTags, localCounts, ruleCounts))
                        return true;
                }
            }
            return false;
        }

        private bool TryApplyGlobalOnlyTunnelRules(Vector2Int position,
Tunnel tunnel, GlobalPlaceTag tunnelGlobalTag, MacroTag macroTag,
                                                   DungeonTags dungeonTags,
RuleGroups tunnelGroups,
                                                   Dictionary<IMicroTagRule,
```

```
int> localCounts,
                                                    Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (dungeonTags.GetMicroTag(position) != MicroTag.None) return
false;

            if (tunnelGroups.RulesByGlobalTag.TryGetValue(tunnelGlobalTag,
out PriorityList<IMicroTagRule> globalRules)) {
                foreach (IMicroTagRule rule in globalRules) {
                    if (rule.MacroTagFilter.IsAllowed(macroTag) &&
                        TryApplyMicroTagToTunnelPosition(rule, position,
tunnel, tunnelGlobalTag, macroTag, dungeonTags, localCounts, ruleCounts))
                        return true;
                }
            }
            return false;
        }

        private bool TryApplyMacroOnlyTunnelRules(Vector2Int position, Tunnel
tunnel, GlobalPlaceTag tunnelGlobalTag, MacroTag macroTag,
                                                    DungeonTags dungeonTags,
RuleGroups tunnelGroups,
                                                    Dictionary<IMicroTagRule,
int> localCounts,
                                                    Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (dungeonTags.GetMicroTag(position) != MicroTag.None) return
false;

            if (tunnelGroups.RulesByMacroTag.TryGetValue(macroTag, out
PriorityList<IMicroTagRule> macroRules)) {
                foreach (IMicroTagRule rule in macroRules) {
                    if (rule.GlobalPlaceTagFilter.IsAllowed(tunnelGlobalTag)
&&
                        TryApplyMicroTagToTunnelPosition(rule, position,
tunnel, tunnelGlobalTag, macroTag, dungeonTags, localCounts, ruleCounts))
                        return true;
                }
            }
            return false;
        }

        private bool TryApplyUnfilteredTunnelRules(Vector2Int position,
Tunnel tunnel, GlobalPlaceTag tunnelGlobalTag, MacroTag macroTag,
                                                    DungeonTags dungeonTags,
RuleGroups tunnelGroups,
                                                    Dictionary<IMicroTagRule,
int> localCounts,
                                                    Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (dungeonTags.GetMicroTag(position) != MicroTag.None) return
false;

            foreach (IMicroTagRule rule in tunnelGroups.RulesForAllTags) {
                if (TryApplyMicroTagToTunnelPosition(rule, position, tunnel,
tunnelGlobalTag, macroTag, dungeonTags, localCounts, ruleCounts))
                    return true;
            }
            return false;
        }

        private bool TryApplyMicroTagToTunnelPosition(IMicroTagRule rule,
Vector2Int position, Tunnel tunnel,
                                                      GlobalPlaceTag
```

```
globalPlaceTag, MacroTag macroTag, DungeonTags dungeonTags,

Dictionary<IMicroTagRule, int> localCounts,

Dictionary<IMicroTagRule, MicroTagRuleCounts> ruleCounts) {
            if (ruleCounts[rule].GlobalCount >= rule.GlobalMaxCount) return
false;

            InitializeLocalCount(rule, localCounts);

            if (localCounts[rule] >= rule.LocalMaxCount) return false;

            if (rule.IsValidForPosition(position, tunnel, globalPlaceTag,
macroTag, dungeonTags)) {
                ApplyTag(rule, position, dungeonTags, localCounts,
ruleCounts);
                return true;
            }

            return false;
        }

        private bool TryApplyMicroTagToPosition(IMicroTagRule rule,
Vector2Int position, Room room,
                                                GlobalPlaceTag
globalPlaceTag, MacroTag macroTag, DungeonTags dungeonTags,
                                                Dictionary<IMicroTagRule,
int> localCounts,
                                                Dictionary<IMicroTagRule,
MicroTagRuleCounts> ruleCounts) {
            if (ruleCounts[rule].GlobalCount >= rule.GlobalMaxCount) return
false;

            InitializeLocalCount(rule, localCounts);

            if (localCounts[rule] >= rule.LocalMaxCount) return false;

            if (rule.IsValidForPosition(position, room, globalPlaceTag,
macroTag, dungeonTags)) {
                ApplyTag(rule, position, dungeonTags, localCounts,
ruleCounts);
                return true;
            }

            return false;
        }

        private void InitializeLocalCount(IMicroTagRule rule,
Dictionary<IMicroTagRule, int> localCounts) {
            if (!localCounts.ContainsKey(rule)) {
                localCounts[rule] = 0;
            }
        }

        private void ApplyTag(IMicroTagRule rule, Vector2Int position,
DungeonTags dungeonTags,
                              Dictionary<IMicroTagRule, int> localCounts,
                              Dictionary<IMicroTagRule, MicroTagRuleCounts>
ruleCounts) {
            dungeonTags.AddMicroTag(rule.MicroTag, position);
            localCounts[rule]++;
            ruleCounts[rule].GlobalCount++;
        }
    }
```

```
    public class MicroTagRuleCounts {
        public int GlobalCount { get; set; }
    }

    public class RuleGroups {
        public Dictionary<(GlobalPlaceTag, MacroTag),
PriorityList<IMicroTagRule>> RulesByTags { get; } = new();
        public Dictionary<GlobalPlaceTag, PriorityList<IMicroTagRule>>
RulesByGlobalTag { get; } = new();
        public Dictionary<MacroTag, PriorityList<IMicroTagRule>>
RulesByMacroTag { get; } = new();
        public PriorityList<IMicroTagRule> RulesForAllTags { get; } = new();

        public PriorityList<IMicroTagRule> GetAllRules() {
            PriorityList<IMicroTagRule> allRules = new();

            AddRulesFromDictionary(RulesByTags.Values, allRules);
            AddRulesFromDictionary(RulesByGlobalTag.Values, allRules);
            AddRulesFromDictionary(RulesByMacroTag.Values, allRules);
            AddRulesFromCollection(RulesForAllTags, allRules);

            return allRules;
        }

        private void AddRulesFromDictionary<TKey>(Dictionary<TKey,
PriorityList<IMicroTagRule>>.ValueCollection collections,
                                                  PriorityList<IMicroTagRule>
allRules) {
            foreach (var rules in collections) {
                AddRulesFromCollection(rules, allRules);
            }
        }

        private void AddRulesFromCollection(PriorityList<IMicroTagRule>
sourceRules, PriorityList<IMicroTagRule> targetRules) {
            foreach (IMicroTagRule rule in sourceRules) {
                int? priority = sourceRules.GetPriority(rule);
                if (priority.HasValue) {
                    targetRules.Add(rule, priority.Value);
                }
            }
        }
    }
}
```

Файл PlayerMovement.cs

Реалізація функціональної вимоги «Управління гравцем»

```
using _Project.Features.PlayerModule;
using _Project.Features.StatsModule;
using UnityEngine;
using Zenject;

public class PlayerMovement : MonoBehaviour
{
    public PlayerRunData Data;
    private int _jumpsRemaining;

    [field:SerializeField] public Rigidbody2D RB { get; private set; }

    public bool IsFacingRight { get; private set; }
    public bool IsJumping { get; private set; }
    public bool IsWallJumping { get; private set; }
    public bool IsSliding { get; private set; }
```

```
    public float LastOnGroundTime { get; private set; }
    public float LastOnWallTime { get; private set; }
    public float LastOnWallRightTime { get; private set; }
    public float LastOnWallLeftTime { get; private set; }

    private bool _isJumpCut;
    private bool _isJumpFalling;

    private float _wallJumpStartTime;
    private int _lastWallJumpDir;

    private Vector2 _moveInput;
    public float LastPressedJumpTime { get; private set; }

    private IStatService<EntityStats> _statService;

    [Header("Checks")]
    [SerializeField] private Transform _groundCheckPoint;
    [SerializeField] private Vector2 _groundCheckSize = new Vector2(0.49f,
0.03f);
    [Space(5)]
    [SerializeField] private Transform _frontWallCheckPoint;
    [SerializeField] private Transform _backWallCheckPoint;
    [SerializeField] private Vector2 _wallCheckSize = new Vector2(0.5f, 1f);

    [Header("Layers & Tags")]
    [SerializeField] private LayerMask _groundLayer;

    [SerializeField] private Transform _playerVisuals;

    [Inject]
    private void InjectDependencies(IStatService<EntityStats> statService)
    {
        _statService = statService;
    }

    private void Start()
    {
        InitializePlayer();
    }

    private void InitializePlayer()
    {

SetGravityScale(Data.CalculateGravityScale(_statService.GetStat(EntityStats.J
umpHeight)));
        IsFacingRight = true;
        ResetJumps();
    }

    private void ResetJumps()
    {
        _jumpsRemaining = (int)_statService.GetStat(EntityStats.MaxJumps);
    }

    private void Update()
    {
        UpdateTimers();
        HandleInput();
        CheckCollisions();
        HandleJumpLogic();
        HandleSlideLogic();
        UpdateGravity();
```

```
    }

    private void UpdateTimers()
    {
        LastOnGroundTime -= Time.deltaTime;
        LastOnWallTime -= Time.deltaTime;
        LastOnWallRightTime -= Time.deltaTime;
        LastOnWallLeftTime -= Time.deltaTime;
        LastPressedJumpTime -= Time.deltaTime;

        if (LastOnGroundTime > 0)
        {
            ResetJumps();
        }
    }

    private void HandleInput()
    {
        _moveInput.x = Input.GetAxisRaw("Horizontal");
        _moveInput.y = Input.GetAxisRaw("Vertical");

        if (_moveInput.x != 0)
            CheckDirectionToFace(_moveInput.x > 0);

        CheckJumpInputs();
    }

    private void CheckJumpInputs()
    {
        if(Input.GetKeyDown(KeyCode.Space) || Input.GetKeyDown(KeyCode.C) ||
Input.GetKeyDown(KeyCode.J))
        {
            OnJumpInput();
        }

        if (Input.GetKeyUp(KeyCode.Space) || Input.GetKeyUp(KeyCode.C) ||
Input.GetKeyUp(KeyCode.J))
        {
            OnJumpUpInput();
        }
    }

    private void CheckCollisions()
    {
        CheckGroundContact();
        CheckWallContact();
    }

    private void CheckGroundContact()
    {
        if (Physics2D.OverlapBox(_groundCheckPoint.position,
_groundCheckSize, 0, _groundLayer) && !IsJumping)
        {
            LastOnGroundTime = Data.coyoteTime;
        }
    }

    private void CheckWallContact()
    {
        if (IsJumping || IsWallJumping) return;

        bool frontWallContact =
Physics2D.OverlapBox(_frontWallCheckPoint.position, _wallCheckSize, 0,
_groundLayer);
```

```
        bool backWallContact =
Physics2D.OverlapBox(_backWallCheckPoint.position, _wallCheckSize, 0,
_groundLayer);

        if (IsFacingRight)
        {
            if (frontWallContact && !IsWallJumping) LastOnWallRightTime =
Data.coyoteTime;
            if (backWallContact && !IsWallJumping) LastOnWallLeftTime =
Data.coyoteTime;
        }
        else
        {
            if (frontWallContact && !IsWallJumping) LastOnWallLeftTime =
Data.coyoteTime;
            if (backWallContact && !IsWallJumping) LastOnWallRightTime =
Data.coyoteTime;
        }

        LastOnWallTime = Mathf.Max(LastOnWallLeftTime, LastOnWallRightTime);
    }

    private void HandleJumpLogic()
    {
        UpdateJumpingState();
        ProcessJumpRequests();
    }

    private void UpdateJumpingState()
    {
        if (IsJumping && RB.linearVelocity.y < 0)
        {
            IsJumping = false;

            if(!IsWallJumping)
                _isJumpFalling = true;
        }

        if (IsWallJumping && Time.time - _wallJumpStartTime >
Data.wallJumpTime)
        {
            IsWallJumping = false;
        }

        if (LastOnGroundTime > 0 && !IsJumping && !IsWallJumping)
        {
            _isJumpCut = false;

            if(!IsJumping)
                _isJumpFalling = false;
        }
    }

    private void ProcessJumpRequests()
    {
        if (CanJump() && LastPressedJumpTime > 0)
        {
            PerformJump();
        }
        else if (CanWallJump() && LastPressedJumpTime > 0)
        {
            PerformWallJump();
        }
    }
```

```
private void PerformJump()
{
    IsJumping = true;
    IsWallJumping = false;
    _isJumpCut = false;
    _isJumpFalling = false;
    _jumpsRemaining--;
    Jump();
}

private void PerformWallJump()
{
    IsWallJumping = true;
    IsJumping = false;
    _isJumpCut = false;
    _isJumpFalling = false;
    _wallJumpStartTime = Time.time;

    int jumpDir = DetermineWallJumpDirection();

    WallJump(jumpDir);

    ResetJumps();
}

private int DetermineWallJumpDirection()
{
    if (LastOnWallRightTime > LastOnWallLeftTime)
    {
        _lastWallJumpDir = -1;
    }
    else
    {
        _lastWallJumpDir = 1;
    }

    return _lastWallJumpDir;
}

private void HandleSlideLogic()
{
    bool shouldSlide = CanSlide() &&
                       ((LastOnWallLeftTime > 0 && _moveInput.x < 0) ||
                        (LastOnWallRightTime > 0 && _moveInput.x > 0));

    IsSliding = shouldSlide;
}

private void UpdateGravity()
{
    if (IsSliding)
    {
        SetGravityScale(0);
    }
    else if (IsFastFalling())
    {
        ApplyFastFallGravity();
    }
    else if (_isJumpCut)
    {
        ApplyJumpCutGravity();
    }
    else if (IsAtJumpApex())
```

```
        {
            ApplyJumpApexGravity();
        }
        else if (IsFalling())
        {
            ApplyFallingGravity();
        }
        else
        {

SetGravityScale(Data.CalculateGravityScale(_statService.GetStat(EntityStats.J
umpHeight)));
        }
    }

    private bool IsFastFalling()
    {
        return RB.linearVelocity.y < 0 && _moveInput.y < 0;
    }

    private void ApplyFastFallGravity()
    {

SetGravityScale(Data.CalculateGravityScale(_statService.GetStat(EntityStats.J
umpHeight)) * Data.fastFallGravityMult);
        RB.linearVelocity = new Vector2(RB.linearVelocity.x,
Mathf.Max(RB.linearVelocity.y, -Data.maxFastFallSpeed));
    }

    private void ApplyJumpCutGravity()
    {

SetGravityScale(Data.CalculateGravityScale(_statService.GetStat(EntityStats.J
umpHeight)) * Data.jumpCutGravityMult);
        RB.linearVelocity = new Vector2(RB.linearVelocity.x,
Mathf.Max(RB.linearVelocity.y, -Data.maxFallSpeed));
    }

    private bool IsAtJumpApex()
    {
        return (IsJumping || IsWallJumping || _isJumpFalling) &&
               Mathf.Abs(RB.linearVelocity.y) < Data.jumpHangTimeThreshold;
    }

    private void ApplyJumpApexGravity()
    {

SetGravityScale(Data.CalculateGravityScale(_statService.GetStat(EntityStats.J
umpHeight)) * Data.jumpHangGravityMult);
    }

    private bool IsFalling()
    {
        return RB.linearVelocity.y < 0;
    }

    private void ApplyFallingGravity()
    {

SetGravityScale(Data.CalculateGravityScale(_statService.GetStat(EntityStats.J
umpHeight)) * Data.fallGravityMult);
        RB.linearVelocity = new Vector2(RB.linearVelocity.x,
Mathf.Max(RB.linearVelocity.y, -Data.maxFallSpeed));
    }
```

```
    private void FixedUpdate()
    {
        HandleMovement();
    }

    private void HandleMovement()
    {
        if (IsWallJumping)
            Run(Data.wallJumpRunLerp);
        else
            Run(1);

        if (IsSliding)
            Slide();
    }

    public void OnJumpInput()
    {
        LastPressedJumpTime = Data.jumpInputBufferTime;
    }

    public void OnJumpUpInput()
    {
        if (CanJumpCut() || CanWallJumpCut())
            _isJumpCut = true;
    }

    public void SetGravityScale(float scale)
    {
        RB.gravityScale = scale;
    }

    private void Run(float lerpAmount)
    {
        float currentSpeed = _statService.GetStat(EntityStats.CurrentSpeed);
        float targetSpeed = _moveInput.x * currentSpeed;
        targetSpeed = Mathf.Lerp(RB.linearVelocity.x, targetSpeed,
lerpAmount);

        float accelRate = CalculateAccelerationRate(targetSpeed);
        ApplyRunForce(targetSpeed, accelRate);
    }

    private float CalculateAccelerationRate(float targetSpeed)
    {
        float currentSpeed = _statService.GetStat(EntityStats.CurrentSpeed);
        float accelRate;
        bool isAccelerating = Mathf.Abs(targetSpeed) > 0.01f;

        if (LastOnGroundTime > 0)
        {
            accelRate = isAccelerating ?
Data.CalculateRunAccelAmount(currentSpeed) :
Data.CalculateRunDeccelAmount(currentSpeed);
        }
        else
        {
            accelRate = isAccelerating ?
                Data.CalculateRunAccelAmount(currentSpeed) * Data.accelInAir
:
                Data.CalculateRunDeccelAmount(currentSpeed) *
Data.deccelInAir;
        }
```

```
        if (IsAtJumpApex())
        {
            accelRate *= Data.jumpHangAccelerationMult;
            targetSpeed *= Data.jumpHangMaxSpeedMult;
        }

        if (ShouldConserveMomentum(targetSpeed))
        {
            accelRate = 0;
        }

        return accelRate;
    }

    private bool ShouldConserveMomentum(float targetSpeed)
    {
        return Data.doConserveMomentum &&
               Mathf.Abs(RB.linearVelocity.x) > Mathf.Abs(targetSpeed) &&
               Mathf.Sign(RB.linearVelocity.x) == Mathf.Sign(targetSpeed) &&
               Mathf.Abs(targetSpeed) > 0.01f &&
               LastOnGroundTime < 0;
    }

    private void ApplyRunForce(float targetSpeed, float accelRate)
    {
        float speedDif = targetSpeed - RB.linearVelocity.x;
        float movement = speedDif * accelRate;
        RB.AddForce(movement * Vector2.right, ForceMode2D.Force);
    }

    private void Turn()
    {
        Vector3 scale = _playerVisuals.localScale;
        scale.x *= -1;
        _playerVisuals.localScale = scale;

        IsFacingRight = !IsFacingRight;
    }

    private void Jump()
    {
        LastPressedJumpTime = 0;

        if (_jumpsRemaining ==
(int)_statService.GetStat(EntityStats.MaxJumps) - 1)
        {
            LastOnGroundTime = 0;
        }

        float force =
Data.CalculateJumpForce(_statService.GetStat(EntityStats.JumpHeight));
        if (RB.linearVelocity.y < 0)
            force -= RB.linearVelocity.y;

        RB.AddForce(Vector2.up * force, ForceMode2D.Impulse);
    }

    private void WallJump(int dir)
    {
        LastPressedJumpTime = 0;
        LastOnGroundTime = 0;
        LastOnWallRightTime = 0;
        LastOnWallLeftTime = 0;
```

```
        Vector2 force = new Vector2(Data.wallJumpForce.x,
Data.wallJumpForce.y);
        force.x *= dir;

        if (Mathf.Sign(RB.linearVelocity.x) != Mathf.Sign(force.x))
            force.x -= RB.linearVelocity.x;

        if (RB.linearVelocity.y < 0)
            force.y -= RB.linearVelocity.y;

        RB.AddForce(force, ForceMode2D.Impulse);
    }

    private void Slide()
    {
        float speedDif = Data.slideSpeed - RB.linearVelocity.y;
        float movement = speedDif * Data.slideAccel;

        float maxForce = Mathf.Abs(speedDif) * (1 / Time.fixedDeltaTime);
        movement = Mathf.Clamp(movement, -maxForce, maxForce);

        RB.AddForce(movement * Vector2.up);
    }

    public void CheckDirectionToFace(bool isMovingRight)
    {
        if (isMovingRight != IsFacingRight)
            Turn();
    }

    private bool CanJump()
    {
        return (LastOnGroundTime > 0 && !IsJumping) || (_jumpsRemaining > 0
&& !IsWallJumping);
    }

    private bool CanWallJump()
    {
        bool hasJumpInput = LastPressedJumpTime > 0;
        bool isOnWall = LastOnWallTime > 0;
        bool isNotOnGround = LastOnGroundTime <= 0;

        bool validWallJumpDirection = !IsWallJumping ||
                                      (LastOnWallRightTime > 0 &&
_lastWallJumpDir == 1) ||
                                      (LastOnWallLeftTime > 0 &&
_lastWallJumpDir == -1);

        return hasJumpInput && isOnWall && isNotOnGround &&
validWallJumpDirection;
    }

    private bool CanJumpCut()
    {
        return IsJumping && RB.linearVelocity.y > 0;
    }

    private bool CanWallJumpCut()
    {
        return IsWallJumping && RB.linearVelocity.y > 0;
    }

    public bool CanSlide()
```

```
    {
        return LastOnWallTime > 0 && !IsJumping && !IsWallJumping &&
LastOnGroundTime <= 0;
    }

    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.green;
        Gizmos.DrawWireCube(_groundCheckPoint.position, _groundCheckSize);

        Gizmos.color = Color.blue;
        Gizmos.DrawWireCube(_frontWallCheckPoint.position, _wallCheckSize);
        Gizmos.DrawWireCube(_backWallCheckPoint.position, _wallCheckSize);
    }
}
```

Файл DropThroughPlatformController.cs

Реалізація функціональної вимоги «Управління гравцем»

```
using System.Linq;
using _Project.Features.InputModule;
using _Project.Features.PhysicsModule;
using Sirenix.OdinInspector;
using UnityEngine;
using Zenject;

namespace _Project.Features.PlayerModule {
    public class DropThroughPlatformController : MonoBehaviour {
        [Header("Settings")]
        [SerializeField] private Collider2D _platformCollider;
        [SerializeField] private CollisionHandler2D _platformDetector;

        [Inject] private IInputService _inputService;

        private bool _isDropping = false;

        private void Update() {
            HandleDropThroughInput();
            if (_isDropping)
                HandleIsDropping();
            else
                HandleIsNotDropping();
        }

        private void HandleIsNotDropping() {
            if (ColliderCanBeEnabledBack())
                _platformCollider.enabled = true;
        }

        private void HandleIsDropping() =>
            _platformCollider.enabled = false;

        private bool ColliderCanBeEnabledBack() =>
            !_platformDetector.TriggeredColliders.Any();

        private void HandleDropThroughInput() =>
            _isDropping = _inputService.GetMoveDirection().y < 0;

        [Button]
        private void TryDropThrough() {
            _isDropping = true;
            _platformCollider.enabled = false;
        }
```

```
    }
}
```

Файл PlayerWeaponController.cs
Реалізація функціональної вимоги «Механіки стрільби»

```
using System;
using _Project.Features.InputModule;
using _Project.Features.WeaponModule;
using _Project.Features.WeaponsModule.Scripts.Weapons.WeaponsCoreModule;
using UnityEngine;
using Zenject;
using IShootable = _Project.Features.WeaponModule.IShootable;

namespace _Project.Features.PlayerModule
{
    public class PlayerWeaponController : MonoBehaviour
    {
        [Inject] private IInputService _inputService;
        [Inject] private EntityWeaponDataHolder _weaponDataHolder;

        private void OnEnable() =>
            _inputService.OnAttackStarted += FireWeapons;

        private void Update() {
            RotateWeaponsInLookDirection();
            if (_inputService.IsAttacking) {
                FireWeapons();
            }
        }

        private void OnDisable() =>
            _inputService.OnAttackStarted -= FireWeapons;

        private void FireWeapons()
        {
            foreach (IWeapon weapon in _weaponDataHolder.EquippedWeapons) {
                switch (weapon) {
                    case IShootable shootable:
                        shootable.Shoot();
                        break;
                    case IDirectionalShootable dir:
                        dir.Shoot(_inputService.GetLookDirection());
                        break;
                }
            }
        }

        private void RotateWeaponsInLookDirection()
        {
            Vector2 lookDirection = _inputService.GetLookDirection();
            if (lookDirection == Vector2.zero)
            {
                return;
            }

            foreach (IWeapon weapon in _weaponDataHolder.EquippedWeapons)
                if (weapon is IRotatable rotatable)
                    rotatable.RotateInDirection(lookDirection);
        }
    }
}
```

Файл AK47_Raycast.cs
Реалізація функціональної вимоги «Механіки стрільби»

```
using System;
using _Project.Features.StatsModule;
using _Project.Features.WeaponModule;
using
_Project.Features.WeaponsModule.Scripts.Projectiles.ProjectilesCoreModule;
using _Project.Features.WeaponsModule.Scripts.Projectiles.ProjectilesPool;
using _Project.Features.WeaponsModule.Scripts.Weapons.WeaponConfigurations;
using _Project.Features.WeaponsModule.Scripts.Weapons.WeaponsCoreModule;
using _Project.Features.WeaponsModule.Scripts.Weapons.WeaponSpreadModule;
using Features.WeaponsModule.Scripts.Weapons.WeaponsInstances;
using Sirenix.OdinInspector;
using UnityEngine;
using Zenject;
using IShootable = _Project.Features.WeaponModule.IShootable;

namespace _Project.Features.WeaponsModule.Scripts.Weapons.WeaponsInstances {
    public class AK47_Raycast : MonoBehaviour, IWeapon, IShootable,
IReloadable, IRotatable {
       [SerializeField] private Transform _firePoint;
       [SerializeField] private Vector2 _attackDirection;

       private const WeaponType WEAPON_TYPE = WeaponType.AK47_Raycast;
       private const ProjectileType PROJECTILE_TYPE =
ProjectileType.BulletRaycast;

       private ProjectilesObjectPool _projectilesPool;
       private WeaponConfiguration _weaponConfiguration;
       private IWeaponConfigurationService _weaponConfigurationService;
       private IWeaponSpreadService _weaponSpreadService;
       private LayersConfiguration _layersConfiguration;

       private int _bulletsInMagazine;
       private float _lastShotTime;
       private float _reloadingTimer;

       [Inject]
       private void InjectDependencies(ProjectilesObjectPool
projectilePoolService,
                                       IWeaponSpreadService
weaponSpreadService,
                                       IWeaponConfigurationService
weaponConfigurationService,
                                       LayersConfiguration
layersConfiguration) {
          _projectilesPool = projectilePoolService;
          _weaponSpreadService = weaponSpreadService;
          _weaponConfigurationService = weaponConfigurationService;
          _layersConfiguration = layersConfiguration;
       }

       private void Awake() {
          _weaponConfiguration =
_weaponConfigurationService.GetConfiguration(WEAPON_TYPE);
          RefillMagazine();
       }

       private void Update() {
          if (!IsReloading)
             return;
```

```
            _reloadingTimer -= Time.deltaTime;
            if (_reloadingTimer <= 0)
               RefillMagazine();
         }

         [Button]
         public void Shoot(Vector2 direction) {
            if (IsReloading || Time.time - _lastShotTime <
_weaponConfiguration.ShootDelay)
               return;

            IProjectile projectile = _projectilesPool.Get(PROJECTILE_TYPE);
            projectile.GetBehaviour<IRaycastBulletBehaviour>()
                      .SetStartPosition(_firePoint.position)
                      .SetDirection(_weaponSpreadService.ApplySpread(direction,
_weaponConfiguration.Spread))
                      .SetDamage(_weaponConfiguration.Damage)
                      .SetSpeed(_weaponConfiguration.Speed)
                      .SetMaxDistance(_weaponConfiguration.MaxDistance)
                      .SetEnemyLayerMask(_layersConfiguration.EnemyLayerMask)

.SetTrailRendererConfiguration(_weaponConfiguration.TrailRendererConfiguratio
n);

            projectile.Launch();

            _lastShotTime = Time.time;
            if (--_bulletsInMagazine == 0)
               StartReloading();
         }

         public bool IsReloading => _reloadingTimer > 0;

         public void StartReloading() =>
            _reloadingTimer = _weaponConfiguration.ReloadTime;

         public void RotateInDirection(Vector2 direction)
         {
            float angle = Mathf.Atan2(direction.y, direction.x) *
Mathf.Rad2Deg;

            if (direction.x < 0)
            {
               transform.localScale = new Vector3(-1, 1, 1);
               angle -= 180;
            }
            else
               transform.localScale = new Vector3(1, 1, 1);

            transform.rotation = Quaternion.Euler(0f, 0f, angle);
         }

         private void RefillMagazine() =>
            _bulletsInMagazine = _weaponConfiguration.MagazineSize;

         public IStatService<WeaponStats> Stats { get; private set; }
         public void InitWeaponStats(IStatService<WeaponStats> weaponStats) {
            Stats = weaponStats;
         }

         private Vector3 GetAttackDirection() =>
            transform.rotation * (_attackDirection * transform.localScale.x);
```

```
        public void Shoot() {
            Shoot(GetAttackDirection());
        }
    }
}
```

Файл MonoPooledProjectile.cs
Реалізація функціональної вимоги «Механіки стрільби»

```
using System.Linq;
using _Project.Features.ObjectPoolModule;
using
_Project.Features.WeaponsModule.Scripts.Projectiles.ProjectilesCoreModule;
using UnityEngine;
using Zenject;

namespace _Project.Features.WeaponsModule.Scripts.Projectiles.ProjectilesPool
{
    public class MonoPooledProjectile : MonoBehaviour,
IPooledObject<ProjectileType>, IProjectile {
        private ProjectilesObjectPool _projectilesPool;
        private IProjectileBehaviour[] _projectileBehaviours;

        [Inject]
        public void InjectDependencies(ProjectilesObjectPool projectilesPool)
=>
            _projectilesPool = projectilesPool;

        public ProjectileType Type { get; set; }

        private void Awake() =>
            _projectileBehaviours = GetComponents<IProjectileBehaviour>();

        private void OnEnable() {
            foreach (IProjectileBehaviour projectileBehaviour in
_projectileBehaviours)
                projectileBehaviour.OnDestroy += ReturnToPool;
        }

        private void OnDisable() {
            foreach (IProjectileBehaviour projectileBehaviour in
_projectileBehaviours)
                projectileBehaviour.OnDestroy -= ReturnToPool;
        }

        private void ReturnToPool() =>
            _projectilesPool.Release(this);

        public void OnEnabled() =>
            gameObject.SetActive(true);

        public void OnDisabled() =>
            gameObject.SetActive(false);

        public void OnDestroyed() {
            if (gameObject == null)
                return;
            Destroy(gameObject);
        }

        public void Launch() {
            foreach (IProjectileBehaviour projectileBehaviour in
_projectileBehaviours)
```

```
            projectileBehaviour.Activate();
        }

        public TProjectileBehaviour GetBehaviour<TProjectileBehaviour>()
           where TProjectileBehaviour : IProjectileBehaviour =>
           (TProjectileBehaviour)_projectileBehaviours.First(behaviours =>
behaviours is TProjectileBehaviour);
    }
}
```

Файл RaycastHitDetector.cs
Реалізація функціональної вимоги «Механіки стрільби»

```
using _Project.Features.DamageModule;
using Features.WeaponsModule.Scripts.Weapons.WeaponsInstances;
using Global.Helpers.Scripts;
using UnityEngine;
using System.Collections.Generic;

namespace _Project.Features.WeaponsModule.Scripts.Weapons.HitDetectorModule
{
    public class RaycastHitDetector : IRaycastHitDetector
    {
        private readonly LayersConfiguration _layersConfiguration;
        private readonly int _initialBufferSize = 16;
        private RaycastHit2D[] _enemyHitResults;
        private RaycastHit2D[] _environmentHitResults;
        private readonly List<HitInfo> _validHits = new();

        public RaycastHitDetector(LayersConfiguration layersConfiguration)
        {
            _layersConfiguration = layersConfiguration;
            _enemyHitResults = new RaycastHit2D[_initialBufferSize];
            _environmentHitResults = new RaycastHit2D[_initialBufferSize];
        }

        public Hit DetectHit(Vector2 startPosition, Vector2 direction, float
maxDistance, LayerMask enemyLayerMask)
        {
            _validHits.Clear();

            PerformEnemyRaycast(startPosition, direction, maxDistance,
enemyLayerMask);
            PerformEnvironmentRaycast(startPosition, direction, maxDistance);

            _validHits.Sort((a, b) =>
a.Hit.distance.CompareTo(b.Hit.distance));

            foreach (HitInfo hitInfo in _validHits)
                if (hitInfo.IsEnemy)
                {
                    HurtBox hurtBox =
hitInfo.Hit.collider.GetComponent<HurtBox>();
                    if (hurtBox != null && hurtBox.Damageable != null)
                        return new TargetHit()
                            .With(targetHit => targetHit.Point =
hitInfo.Hit.point)
                            .With(targetHit => targetHit.Distance =
hitInfo.Hit.distance)
                            .With(targetHit => targetHit.Damageable =
hurtBox.Damageable);
                }
                else
```

```
                    return new EnvironmentHit()
                        .With(environmentHit => environmentHit.Point =
hitInfo.Hit.point)
                        .With(environmentHit => environmentHit.Distance =
hitInfo.Hit.distance);

            return new NullHit()
                .With(nullHit => nullHit.Point = startPosition + direction *
maxDistance)
                .With(nullHit => nullHit.Distance = maxDistance);
        }

        private void PerformEnvironmentRaycast(Vector2 startPosition, Vector2
direction, float maxDistance) {
            int environmentHitCount = Physics2D.RaycastNonAlloc(
                startPosition,
                direction,
                _environmentHitResults,
                maxDistance,
                _layersConfiguration.EnvironmentLayerMask
            );

            if (environmentHitCount > _environmentHitResults.Length) {
                int newSize = Mathf.NextPowerOfTwo(environmentHitCount);
                _environmentHitResults = new RaycastHit2D[newSize];
                environmentHitCount = Physics2D.RaycastNonAlloc(
                    startPosition,
                    direction,
                    _environmentHitResults,
                    maxDistance,
                    _layersConfiguration.EnvironmentLayerMask
                );
            }

            for (int i = 0; i < environmentHitCount; i++)
            {
                RaycastHit2D hit = _environmentHitResults[i];

                if (hit.collider != null)
                {
                    _validHits.Add(new HitInfo(hit, false));
                }
            }
        }

        private void PerformEnemyRaycast(Vector2 startPosition, Vector2
direction, float maxDistance, LayerMask enemyLayerMask) {
            int enemyHitCount = Physics2D.RaycastNonAlloc(
                startPosition,
                direction,
                _enemyHitResults,
                maxDistance,
                enemyLayerMask
            );

            if (enemyHitCount > _enemyHitResults.Length)
            {
                int newSize = Mathf.NextPowerOfTwo(enemyHitCount);
                _enemyHitResults = new RaycastHit2D[newSize];
                enemyHitCount = Physics2D.RaycastNonAlloc(
                    startPosition,
                    direction,
                    _enemyHitResults,
                    maxDistance,
```

```
                    enemyLayerMask
                );
            }

            for (int i = 0; i < enemyHitCount; i++)
            {
                RaycastHit2D hit = _enemyHitResults[i];

                if (hit.collider != null &&
hit.collider.gameObject.activeSelf)
                    _validHits.Add(new HitInfo(hit, true));
            }
        }

        private class HitInfo
        {
            public RaycastHit2D Hit;
            public bool IsEnemy;

            public HitInfo(RaycastHit2D hit, bool isEnemy)
            {
                Hit = hit;
                IsEnemy = isEnemy;
            }
        }
    }
}
```

Файл RaycastBulletProjectileBehaviour.cs
Реалізація функціональної вимоги «Механіки стрільби»

```
using System;
using System.Collections;
using _Project.Features.WeaponsModule.Scripts.Weapons.HitDetectorModule;
using _Project.Features.WeaponsModule.Scripts.Weapons.WeaponConfigurations;
using Global.Helpers.Scripts;
using UnityEngine;
using Zenject;

namespace _Project.Features.WeaponsModule.Scripts.Weapons.WeaponsInstances {
    public class RaycastBulletProjectileBehaviour : MonoBehaviour,
IRaycastBulletBehaviour, IHitVisitor {
       [SerializeField] private TrailRenderer _trailRenderer;
       private LayerMask _enemyLayerMask;

       private float _damage;
       private Vector2 _direction;
       private float _maxDistance;
       private float _speed;
       private Vector2 _startPosition;

       private IRaycastHitDetector _raycastHitDetector;

       public event Action<Vector2> OnHit;
       public event Action OnDestroy;

       [Inject]
       private void InjectDependencies(IRaycastHitDetector
raycastHitDetector) =>
          _raycastHitDetector = raycastHitDetector;

       public void Visit(Hit hit) =>
          StartCoroutine(SpawnTrail(_trailRenderer, hit.Point));
```

```
        public void Visit(TargetHit hit) {
            StartCoroutine(InvokeOnHitAfterTime(hit.Point, hit.Distance /
_speed));
            hit.Damageable.TakeDamage(_damage);
        }

        public void Visit(EnvironmentHit hit) =>
            StartCoroutine(InvokeOnHitAfterTime(hit.Point, hit.Distance /
_speed));

        public void Visit(NullHit hit) { }

        public void Activate() {
            StartCoroutine(ResetTrailRenderer());

            transform.position = _startPosition;

            _raycastHitDetector.DetectHit(_startPosition, _direction,
_maxDistance, _enemyLayerMask).Accept(this);
        }

        private IEnumerator InvokeOnHitAfterTime(Vector2 hitPoint, float
timeToReach) {
            yield return new WaitForSeconds(timeToReach);
            OnHit?.Invoke(hitPoint);
        }

        private IEnumerator SpawnTrail(TrailRenderer trailRenderer, Vector2
finalPosition) {
            float time = 0;
            Vector2 startPosition = transform.position;

            float totalDistance = Vector2.Distance(startPosition,
finalPosition);
            float timeToReach = totalDistance / _speed;

            while (time < timeToReach) {
                transform.position = Vector2.Lerp(startPosition, finalPosition,
time / timeToReach);
                time += Time.deltaTime;

                yield return null;
            }

            transform.position = finalPosition;

            Invoke(nameof(InvokeOnDestroy), trailRenderer.time);
        }

        private IEnumerator ResetTrailRenderer() {
            _trailRenderer.Clear();
            _trailRenderer.emitting = false;
            yield return null;
            _trailRenderer.emitting = true;
        }

        private void InvokeOnDestroy() =>
            OnDestroy?.Invoke();

        #region Builder

        public IRaycastBulletBehaviour SetDirection(Vector2 direction) =>
            this.With(() => _direction = direction.normalized);
```

```
        public IRaycastBulletBehaviour SetStartPosition(Vector2 startPosition)
=>
            this.With(() => _startPosition = startPosition);

        public IRaycastBulletBehaviour SetDamage(float damage) =>
            this.With(() => _damage = damage);

        public IRaycastBulletBehaviour SetSpeed(float speed) =>
            this.With(() => _speed = speed);

        public IRaycastBulletBehaviour SetMaxDistance(float maxDistance) =>
            this.With(() => _maxDistance = maxDistance);

        public IRaycastBulletBehaviour SetTrailRendererConfiguration(
            TrailRendererConfiguration trailRendererConfiguration) =>
            this.With(() =>
trailRendererConfiguration.ApplyTo(_trailRenderer));

        public IRaycastBulletBehaviour SetEnemyLayerMask(LayerMask
enemyLayerMask) =>
            this.With(() => _enemyLayerMask = enemyLayerMask);

        #endregion
    }
}
```

Файл BehaviourTree.cs

Реалізація функціональної вимоги «Штучний інтелект ворогів»

```
using System.Text;
using UnityEngine;

namespace _Project.Features.EnemyModule.BehaviourTrees {
    public class BehaviourTree : Node {
        private readonly IBehaviourTreeStrategy _behaviourTreeStrategy;

        public BehaviourTree(string name, IBehaviourTreeStrategy
behaviourTreeStrategy = null) : base(name) =>
            _behaviourTreeStrategy = behaviourTreeStrategy ??
BehaviourTreeStrategies.RunForever;

        public override Status Process() {
            Status status = Children[CurrentChild].Process();
            if (_behaviourTreeStrategy.ShouldReturn(status))
                return status;

            CurrentChild = (CurrentChild + 1) % Children.Count;
            return Status.Running;
        }

        public void DebugLogTree() {
            StringBuilder sb = new StringBuilder();
            PrintNode(this, 0, sb);
            Debug.Log(sb.ToString());
        }

        private static void PrintNode(Node node, int indentLevel,
StringBuilder sb) {
            sb.Append(' ', indentLevel * 2).AppendLine(node.Name);
            foreach (Node child in node.Children)
                PrintNode(child, indentLevel + 1, sb);
        }
```

```
    }
}
```

Файл Leaf.cs
Реалізація функціональної вимоги «Штучний інтелект ворогів»

```
namespace _Project.Features.EnemyModule.BehaviourTrees {
    public class Leaf : Node {
        private readonly IStrategy _strategy;

        public Leaf(string name, IStrategy strategy, int priority = 0) :
base(name, priority) =>
            _strategy = strategy;

        public override Status Process() => _strategy.Process();

        protected override void Reset() => _strategy.Reset();
    }
}
```

Файл Node.cs
Реалізація функціональної вимоги «Штучний інтелект ворогів»

```
using System.Collections.Generic;

namespace _Project.Features.EnemyModule.BehaviourTrees {
    public class Node {
        public enum Status { Success, Failure, Running }

        public readonly string Name;
        public readonly int Priority;

        public readonly List<Node> Children = new();
        protected int CurrentChild;

        protected Node(string name = "Node", int priority = 0) {
            Name = name;
            Priority = priority;
        }

        public void AddChild(Node child) => Children.Add(child);

        public virtual Status Process() => Children[CurrentChild].Process();

        protected virtual void Reset() {
            CurrentChild = 0;
            foreach (Node child in Children)
                child.Reset();
        }
    }
}
```

Файл Selector.cs
Реалізація функціональної вимоги «Штучний інтелект ворогів»

```
namespace _Project.Features.EnemyModule.BehaviourTrees {
    public class Selector : Node {
        public Selector(string name, int priority = 0) : base(name, priority)
{ }
```

```
        public override Status Process() {
            if (CurrentChild < Children.Count) {
                if (Children[CurrentChild].Process() == Status.Running)
                    return Status.Running;

                if (Children[CurrentChild].Process() == Status.Success) {
                    Reset();
                    return Status.Success;
                }

                CurrentChild++;
                return Status.Running;
            }

            Reset();
            return Status.Failure;
        }
    }
}
```

Файл Sequence.cs
Реалізація функціональної вимоги «Штучний інтелект ворогів»

```
namespace _Project.Features.EnemyModule.BehaviourTrees {
    public class Sequence : Node {
        public Sequence(string name, int priority = 0) : base(name, priority)
{ }

        public override Status Process() {
            if (CurrentChild < Children.Count) {
                if (Children[CurrentChild].Process() == Status.Running)
                    return Status.Running;

                if (Children[CurrentChild].Process() == Status.Failure) {
                    CurrentChild = 0;
                    return Status.Failure;
                }

                CurrentChild++;
                return CurrentChild == Children.Count ? Status.Success :
Status.Running;
            }

            Reset();
            return Status.Success;
        }
    }
}
```

Файл Blackboard.cs
Реалізація функціональної вимоги «Штучний інтелект ворогів»

```
using System;
using System.Collections.Generic;
using System.Reflection;

namespace _Project.Features.EnemyModule.Blackboard {
    [Serializable]
    public readonly struct BlackboardKey : IEquatable<BlackboardKey> {
        private readonly string _name;
```

```
        private readonly int _hashedKey;

        public BlackboardKey(string name) {
            _name = name;
            _hashedKey = name.ComputeFNV1aHash();
        }

        public bool Equals(BlackboardKey other) =>
            _hashedKey == other._hashedKey;

        public override bool Equals(object obj) =>
            obj is BlackboardKey other && Equals(other);

        public override int GetHashCode() =>
            _hashedKey;

        public override string ToString() =>
            _name;

        public static bool operator ==(BlackboardKey lhs, BlackboardKey rhs)
=>
            lhs._hashedKey == rhs._hashedKey;

        public static bool operator !=(BlackboardKey lhs, BlackboardKey rhs)
=>
            !(lhs == rhs);
    }

    [Serializable]
    public class BlackboardEntry<T> {
        public BlackboardKey Key { get; }
        public T Value { get; }
        public Type ValueType { get; }

        public BlackboardEntry(BlackboardKey key, T value) {
            Key = key;
            Value = value;
            ValueType = typeof(T);
        }

        public override bool Equals(object obj) =>
            obj is BlackboardEntry<T> other && other.Key == Key;

        public override int GetHashCode() =>
            Key.GetHashCode();
    }

    [Serializable]
    public class Blackboard {
        private Dictionary<string, BlackboardKey> _keys = new();
        private Dictionary<BlackboardKey, object> _entries = new();

        public bool TryGetValue<T>(BlackboardKey key, out T value) {
            if (_entries.TryGetValue(key, out object entry) && entry is
BlackboardEntry<T> castedEntry) {
                value = castedEntry.Value;
                return true;
            }

            value = default;
            return false;
        }

        public void SetValue<T>(BlackboardKey key, T value) =>
```

```
            _entries[key] = new BlackboardEntry<T>(key, value);

        public void Debug() {
            foreach (KeyValuePair<BlackboardKey, object> entry in _entries) {
                Type entryType = entry.Value.GetType();

                if (entryType.IsGenericType &&
entryType.GetGenericTypeDefinition() == typeof(BlackboardEntry<>)) {
                    PropertyInfo valueProperty =
entryType.GetProperty("Value");
                    if (valueProperty == null)
                        continue;

                    object value = valueProperty.GetValue(entry.Value);
                    UnityEngine.Debug.LogError($"[Blackboard.Debug] Key:
{entry.Key} Value: {value}");
                }
            }
        }

        public BlackboardKey GetOrRegisterKey(string name) {
            if (_keys.TryGetValue(name, out BlackboardKey registerKey))
                return registerKey;

            BlackboardKey key = new(name);
            _keys.Add(name, key);
            return key;
        }

        public bool ContainsKey(BlackboardKey key) =>
            _entries.ContainsKey(key);

        public void Remove(BlackboardKey key) =>
            _entries.Remove(key);
    }
}
```

Файл XPOrb.cs

Реалізація функціональної вимоги «Збір предметів та покращення»

```
using DG.Tweening;
using UnityEngine;

namespace _Project.Features.ExperienceModule.Orbs {
    public class XPOrb : MonoBehaviour
    {
        [Header("Components")]
        [SerializeField] private SpriteRenderer orbSprite;
        [SerializeField] private Collider2D orbCollider;
        [SerializeField] private ParticleSystem pickupFX;

        [Header("Settings")]
        [SerializeField] private float initialBounceHeight = 0.5f;
        [SerializeField] private float initialBounceDuration = 0.5f;

        private float xpValue;
        private bool isMoving;
        private XPOrbData orbData;
        private Sequence moveSequence;

        public float XPValue => xpValue;

        public void Initialize(float xp, XPOrbData data)
        {
```

```
            xpValue = xp;
            orbData = data;
            orbSprite.color = data.orbColor;

            // Spawn animation
            transform.localScale = Vector3.zero;
            transform.DOScale(1f, 0.3f).SetEase(Ease.OutBack);

            // Initial bounce
            Vector3 initialPos = transform.position;
            transform.DOMoveY(initialPos.y + initialBounceHeight,
initialBounceDuration)
                .SetEase(Ease.OutQuad)
                .OnComplete(() =>
                {
                    transform.DOMoveY(initialPos.y, initialBounceDuration *
0.5f)
                        .SetEase(Ease.InQuad);
                });
        }

        public void MoveToTarget(Transform target)
        {
            if (isMoving) return;

            isMoving = true;
            orbCollider.enabled = false;

            Transform targetTransform = target;

            float moveTime = 1f / orbData.moveSpeed;

            moveSequence = DOTween.Sequence();

            Vector3 startPos = transform.position;
            moveSequence.Append(DOTween.To(() => 0f, x =>
            {
                float t = orbData.moveCurve.Evaluate(x);
                transform.position = Vector3.Lerp(startPos,
targetTransform.position, t);
            }, 1f, moveTime));

            moveSequence.AppendCallback(() =>
            {
                if (pickupFX != null)
                {
                    pickupFX.Play();
                }

                transform.DOScale(0f, orbData.dissolveDuration)
                    .SetEase(Ease.InBack);

                orbSprite.DOFade(0f, orbData.dissolveDuration)
                    .OnComplete(() =>
                    {
                        Destroy(gameObject);
                    });
            });
        }

        public void CancelMovement()
        {
            if (moveSequence != null && moveSequence.IsActive())
            {
```

```
                moveSequence.Kill();
            }
            isMoving = false;
            orbCollider.enabled = true;
        }

        private void OnDestroy()
        {
            if (moveSequence != null && moveSequence.IsActive())
            {
                moveSequence.Kill();
            }
        }
    }
}
```

Файл ExperienceModel.cs
Реалізація функціональної вимоги «Збір предметів та покращення»

```
using System;
using _Project.Scripts.Infrastructure;
using UnityEngine;

namespace _Project.Features.ExperienceModule {
    public class ExperienceModel : IModel
    {
        public int CurrentLevel { get; private set; } = 1;
        public int CurrentXP { get; private set; }
        public int MaxXP { get; private set; }

        public float XPPercentage => MaxXP > 0 ? (float)CurrentXP / MaxXP :
0f;

        public event Action<int> OnLevelChanged;
        public event Action OnLevelUp;
        public event Action<int> OnCurrentXPChanged;
        public event Action<int> OnMaxXPChanged;

        public void SetCurrentXP(int xp)
        {
            CurrentXP = xp;
            OnCurrentXPChanged?.Invoke(xp);
        }

        public void SetLevel(int level)
        {
            CurrentLevel = level;
            OnLevelChanged?.Invoke(level);
        }

        public void SetMaxXP(int xp) {
            MaxXP = xp;
            OnMaxXPChanged?.Invoke(xp);
        }

        public void SetLevelSilently(int level) =>
            CurrentLevel = level;

        public void IncrementLevel() {
            CurrentLevel++;
            OnLevelUp?.Invoke();
            OnLevelChanged?.Invoke(CurrentLevel);
        }
```

```
    }
}
```

Файл ExperienceService.cs
Реалізація функціональної вимоги «Збір предметів та покращення»

```
namespace _Project.Features.ExperienceModule {
    public class ExperienceService : IExperienceService {
        private readonly ExperienceModel _model;
        private readonly XPLevelConfiguration _levelConfig;

        public ExperienceService(ExperienceModel model, XPLevelConfiguration
levelConfig) {
            _model = model;
            _levelConfig = levelConfig;
        }

        public void AddXP(int amount) {
            int currentXP = _model.CurrentXP;
            int remainingXP = currentXP + amount;

            while (remainingXP >= _model.MaxXP && _model.CurrentLevel <
_levelConfig.MaxLevel) {
                remainingXP -= _levelConfig.GetXPForLevel(_model.CurrentLevel
+ 1);
                _model.IncrementLevel();

_model.SetMaxXP(_levelConfig.GetXPForLevel(_model.CurrentLevel + 1));
            }

            _model.SetCurrentXP(remainingXP);
        }
    }
}
```

Файл LevelConfiguration.cs
Реалізація функціональної вимоги «Збір предметів та покращення»

```
using UnityEngine;

namespace _Project.Features.ExperienceModule {
    [CreateAssetMenu(fileName = nameof(XPLevelConfiguration) + "_Default",
menuName = "Configurations/ExperienceModule/" +
nameof(XPLevelConfiguration))]
    public class XPLevelConfiguration : ScriptableObject
    {
        [Header("Level Settings")]
        [SerializeField] private int firstLevelXP = 100;
        [SerializeField] private float xpMultiplier = 1.2f;
        [SerializeField] private int maxLevel = 100;

        public int FirstLevelXP => firstLevelXP;
        public float XPMultiplier => xpMultiplier;
        public int MaxLevel => maxLevel;

        public int GetXPForLevel(int level)
        {
            if (level <= 0) return 0;
            if (level > maxLevel) level = maxLevel;

            return Mathf.RoundToInt(firstLevelXP * Mathf.Pow(xpMultiplier,
```

```
level - 1));
        }
    }
}
```

Файл PlayerXPPresenter.cs
Реалізація функціональної вимоги «Збір предметів та покращення»

```
using _Project.Features.ExperienceModule;
using _Project.Infrastructure.MVP.Core;
using JetBrains.Annotations;

namespace _Project.Features.UIModule.PlayerLevelUI {
    [PublicAPI]
    public class PlayerXPPresenter : PresenterBehaviour<PlayerXPViewBase> {
        private readonly ExperienceModel _experienceModel;

        public PlayerXPPresenter(ExperienceModel experienceModel) =>
            _experienceModel = experienceModel;

        public override void OnViewSet() {
            _experienceModel.OnCurrentXPChanged += UpdateXPPercent;
            _experienceModel.OnMaxXPChanged += UpdateXPPercent;
            _experienceModel.OnLevelChanged += UpdateLevel;

            UpdateXPPercent();
            UpdateLevel();
        }

        public override void OnDisposed() {
            _experienceModel.OnCurrentXPChanged -= UpdateXPPercent;
            _experienceModel.OnMaxXPChanged -= UpdateXPPercent;
            _experienceModel.OnLevelChanged -= UpdateLevel;
        }

        private void UpdateLevel() =>
            View.SetLevel(_experienceModel.CurrentLevel);

        private void UpdateXPPercent() {
            float percent = _experienceModel.XPPercentage;
            View.SetXPPercent(percent);
        }

        private void UpdateLevel(int _) =>
            UpdateLevel();

        private void UpdateXPPercent(int _) =>
            UpdateXPPercent();
    }
}
```

Файл PlayerXPView.cs
Реалізація функціональної вимоги «Збір предметів та покращення»

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace _Project.Features.UIModule.PlayerLevelUI {
    internal class PlayerXPView : PlayerXPViewBase {
        [SerializeField] private Slider _xpSlider;
```

```
        [SerializeField] private TMP_Text _levelText;

        public override void SetXPPercent(float percent) =>
            _xpSlider.value = percent;

        public override void SetLevel(int level) =>
            _levelText.text = level.ToString();
    }
}
```

Файл UpgradeApplicationService.cs
Реалізація функціональної вимоги «Збір предметів та покращення»

```
using _Project.Features.StatsModule;
using _Project.Features.PlayerSpawnerModule;

namespace _Project.Features.UpgradesModule.API {
    public class UpgradeApplicationService : IUpgradeApplicationService {
        private readonly IPlayerWeaponService _playerWeaponService;
        private readonly PlayerStatsModel _playerStatsModel;

        public UpgradeApplicationService(IPlayerWeaponService
playerWeaponService, PlayerStatsModel playerStatsModel) {
            _playerWeaponService = playerWeaponService;
            _playerStatsModel = playerStatsModel;
        }

        public void ApplyUpgrade(UpgradeData upgrade, int currentLevel) {
            switch (upgrade.upgradeType) {
                case UpgradeType.Stat: ApplyStatUpgrade(upgrade.statData,
currentLevel + 1); break;
                case UpgradeType.Weapon:
ApplyWeaponUpgrade(upgrade.weaponData, currentLevel + 1); break;
                case UpgradeType.WeaponUnlock:
ApplyWeaponUnlock(upgrade.unlockData); break;
            }
        }

        private void ApplyStatUpgrade(StatUpgradeData statData, int level) {
            float value = statData.baseValue + (statData.valuePerLevel *
(level - 1));
            if (statData.isPercentage)

_playerStatsModel.Stats.ModifyStatPercentage(statData.statType, value);
            else
                _playerStatsModel.Stats.ModifyStat(statData.statType, value);
        }

        private void ApplyWeaponUpgrade(WeaponUpgradeData weaponData, int
level) {
            foreach (WeaponModifier modifier in weaponData.modifiers) {
                float value = modifier.baseValue + (modifier.valuePerLevel *
(level - 1));
                _playerWeaponService.ModifyWeapon(weaponData.weaponType,
modifier.statToModify, value, modifier.isPercentage);
            }
        }

        private void ApplyWeaponUnlock(WeaponUnlockData unlockData) =>
            _playerWeaponService.UnlockWeapon(unlockData.weaponType);

        public string GetUpgradeDescription(UpgradeData upgrade, int level) {
            switch (upgrade.upgradeType) {
```

```
                case UpgradeType.Stat: return
GetStatUpgradeDescription(upgrade.statData, level);
                case UpgradeType.Weapon: return
GetWeaponUpgradeDescription(upgrade.weaponData, level);
                case UpgradeType.WeaponUnlock: return
GetWeaponUnlockDescription(upgrade.unlockData, level);
                default: return upgrade.description;
            }
        }

        private string GetStatUpgradeDescription(StatUpgradeData statData,
int level) {
            float value = statData.baseValue + (statData.valuePerLevel *
(level - 1));
            string suffix = statData.isPercentage ? "%" : "";
            return $"{statData.statType}: +{value}{suffix}";
        }

        private string GetWeaponUpgradeDescription(WeaponUpgradeData
weaponData, int level) {
            var description = "";
            foreach (var modifier in weaponData.modifiers) {
                float value = modifier.baseValue + (modifier.valuePerLevel *
(level - 1));
                string suffix = modifier.isPercentage ? "%" : "";
                description += $"{modifier.statToModify.ToString()}:
+{value}{suffix}\n";
            }
            return description.TrimEnd('\n');
        }

        private string GetWeaponUnlockDescription(WeaponUnlockData
unlockData, int level) {
            if (level >= 1)
                return "Already Unlocked";
            return $"Unlock: {unlockData.weaponType.ToString()}";
        }
    }
}
```

Файл HitBox.cs

Реалізація функціональної вимоги «Система здоров’я та шкоди»

```
using System.Collections.Generic;
using _Project.Features.WeaponsModule.Scripts.Weapons.DamagablesModule;
using UnityEngine;

namespace _Project.Features.DamageModule {
    public class HitBox : MonoBehaviour {
        [SerializeField] private List<Collider2D> _hitColliders;

        private readonly List<Collider2D> _cache = new List<Collider2D>();

        public List<IDamageable> GetOverlappingDamageables(LayerMask
layerMask) {
            List<IDamageable> damageables = new List<IDamageable>();
            ContactFilter2D contactFilter = new ContactFilter2D {
                useTriggers = true,
                useLayerMask = true,
                layerMask = layerMask
            };
            foreach (Collider2D hitCollider in _hitColliders) {
                int collidersOverlapped =
Physics2D.OverlapCollider(hitCollider, contactFilter, _cache);
```

```
                for (int i = 0; i < collidersOverlapped; i++) {
                    if (!_cache[i].TryGetComponent(out HurtBox hurtBox))
                        continue;

                    if (damageables.Contains(hurtBox.Damageable))
                        continue;

                    damageables.Add(hurtBox.Damageable);
                }
            }

            return damageables;
        }
    }
}
```

Файл HurtBox.cs
Реалізація функціональної вимоги «Система здоров’я та шкоди»

```
using _Project.Features.WeaponsModule.Scripts.Weapons.DamagablesModule;
using UnityEngine;

namespace _Project.Features.DamageModule {
    public class HurtBox : MonoBehaviour {
        public IDamageable Damageable { get; set; }
    }
}
```

Файл Attacker.cs
Реалізація функціональної вимоги «Система здоров’я та шкоди»

```
using System.Collections.Generic;
using System.Linq;
using _Project.Features.WeaponsModule.Scripts.Weapons.DamagablesModule;
using Sirenix.OdinInspector;
using UnityEngine;

namespace _Project.Features.DamageModule {
    public class Attacker : MonoBehaviour, IDamageDealer {
        [SerializeField] private HitBox[] _hitBoxes;
        [SerializeField] private LayerMask _enemyLayerMask;

        [Button]
        public void DealDamage(float damage) {
            IEnumerable<IDamageable> damageables = _hitBoxes.SelectMany(el =>
el.GetOverlappingDamageables(_enemyLayerMask));
            foreach (IDamageable damageable in damageables.Distinct())
                damageable.TakeDamage(damage);
        }
    }
}
```

Файл SimpleMonoDamageable.cs
Реалізація функціональної вимоги «Система здоров’я та шкоди»

```
using System;
using _Project.Features.StatsModule;
using _Project.Features.WeaponsModule.Scripts.Weapons.DamagablesModule;
using Sirenix.OdinInspector;
```

```
using UnityEngine;
using Zenject;

namespace _Project.Features.DamageModule {
    public class SimpleMonoDamageable : MonoDamageable {
        [SerializeField] private HurtBox[] _hurtBoxes;
        private IStatService<EntityStats> _stats;

        [Inject]
        private void InjectDependencies(IStatService<EntityStats> stats) =>
            _stats = stats;

        private float Health { get => _stats[EntityStats.CurrentHealth]; set
=> _stats[EntityStats.CurrentHealth] = value; }

        private void OnEnable() {
            _stats[EntityStats.CurrentHealth] =
_stats[EntityStats.MaxHealth];
            foreach (HurtBox hurtBox in _hurtBoxes)
                hurtBox.Damageable = this;
        }

        public override event Action<float> OnTakeDamage;
        public override event Action OnDeath;
        public override event Action OnAfterDeath;

        public override void TakeDamage(float damage) {
            if (Health <= 0)
                return;

            Health -= damage;
            OnTakeDamage?.Invoke(damage);

            if (Health <= 0) {
                Health = 0;
                OnDeath?.Invoke();
                OnAfterDeath?.Invoke();
            }
        }

        [Button]
        private void InstantDeath() {
            TakeDamage(float.MaxValue);
        }
    }
}
```

Файл PlayerHealthPresenter.cs

Реалізація функціональної вимоги «Система здоров’я та шкоди»

```
using _Project.Features.PlayerSpawnerModule;
using _Project.Features.StatsModule;
using _Project.Infrastructure.MVP.Core;
using JetBrains.Annotations;

namespace _Project.Features.UIModule.PlayerHealthUI {
    [PublicAPI]
    public class PlayerHealthPresenter :
PresenterBehaviour<PlayerHealthViewBase> {
        private readonly PlayerStatsModel _playerStatsModel;

        public PlayerHealthPresenter(PlayerStatsModel playerStatsModel) =>
            _playerStatsModel = playerStatsModel;
```

```
        public override void OnViewSet() =>
            _playerStatsModel.Stats.OnStatChanged += OnCurrentHealthChanged;

        public override void OnDisposed() =>
            _playerStatsModel.Stats.OnStatChanged -= OnCurrentHealthChanged;

        private void OnCurrentHealthChanged(EntityStats stat, float newValue)
{
            switch (stat) {
                case EntityStats.MaxHealth:
                case EntityStats.CurrentHealth:
                    UpdateCurrentHealth();
                    break;
            }
        }

        private void UpdateCurrentHealth() {

View.SetCurrentHealth((int)_playerStatsModel.Stats[EntityStats.CurrentHealth]
);

View.SetHPBarFill(_playerStatsModel.Stats[EntityStats.CurrentHealth] /
(float)_playerStatsModel.Stats[EntityStats.MaxHealth]);
        }

        private void OnCurrentHealthChanged(int currentHealth) =>
            UpdateCurrentHealth();

        private void OnMaxHealthChanged(int maxHealth) {
            UpdateCurrentHealth();
        }
    }
}
```

Файл PlayerHealthView.cs
Реалізація функціональної вимоги «Система здоров’я та шкоди»

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace _Project.Features.UIModule.PlayerHealthUI {
    internal class PlayerHealthView : PlayerHealthViewBase {
        [SerializeField] private Slider _healthBar;
        [SerializeField] private TMP_Text _currentHealthText;

        public override void SetCurrentHealth(int currentHealth) {
            if (_currentHealthText)
                _currentHealthText.text = currentHealth.ToString();
        }

        public override void SetHPBarFill(float fillAmount) =>
            _healthBar.value = fillAmount;
    }
}
```

Файл GamePauseService.cs
Реалізація функціональної вимоги «Пауза та налаштування»

```
namespace _Project.Features.GameTimeModule {
    public class GamePauseService : IGamePauseService
    {
        private readonly GamePauseModel _model;

        public bool IsPaused => _model.IsPaused;
        public int PauseCounter => _model.PauseCounter;

        public GamePauseService(GamePauseModel model) =>
            _model = model;

        public void PauseTime() =>
            _model.IncrementPause();

        public void ResumeTime() =>
            _model.DecrementPause();

        public void ForceResumeTime()
        {
            _model.ForceResume();
        }
    }
}
```

Файл GamePausePresenter.cs
Реалізація функціональної вимоги «Пауза та налаштування»

```
using _Project.Infrastructure.MVP.Core;
using _Project.Infrastructure.StateMachines.Scripts.GameplayStates;
using _Project.Scripts.Infrastructure.StateMachines;
using _Project.Scripts.Infrastructure.StateMachines.GlobalStates;
using JetBrains.Annotations;

namespace _Project.Features.UIModule.GamePauseUI {
    [PublicAPI]
    public class GamePausePresenter : PresenterBehaviour<GamePauseViewBase> {
        private readonly GameplayStateMachine _gameplayStateMachine;
        private readonly GlobalStateMachine _globalStateMachine;

        public GamePausePresenter(GlobalStateMachine globalStateMachine,
GameplayStateMachine gameplayStateMachine) {
            _globalStateMachine = globalStateMachine;
            _gameplayStateMachine = gameplayStateMachine;
        }

        public override void OnViewSet() {
            View.OnResumeButtonClicked += OnResumeButtonClicked;
            View.OnBackToMainMenuButtonClicked +=
OnBackToMainMenuButtonClicked;
            View.OnExitButtonClicked += OnExitButtonClicked;
        }

        public override void OnDisposed() {
            View.OnResumeButtonClicked -= OnResumeButtonClicked;
            View.OnBackToMainMenuButtonClicked -=
OnBackToMainMenuButtonClicked;
            View.OnExitButtonClicked -= OnExitButtonClicked;
        }

        private void OnResumeButtonClicked() =>
            _gameplayStateMachine.Enter<ExploreLevelState>();

        private void OnExitButtonClicked() =>
```

```
            _globalStateMachine.Enter<QuitGameState>();

        private void OnBackToMainMenuButtonClicked() =>
            _globalStateMachine.Enter<MainMenuState>();
    }
}
```

Файл GamePauseView.cs

Реалізація функціональної вимоги «Пауза та налаштування»

```
using UnityEngine;
using UnityEngine.UI;

namespace _Project.Features.UIModule.GamePauseUI {
    internal class GamePauseView : GamePauseViewBase {
        [SerializeField] private Button _resumeButton;
        [SerializeField] private Button _backToMainMenuButton;
        [SerializeField] private Button _exitButton;

        private void OnEnable() {
            _resumeButton.onClick.AddListener(InvokeOnResumeButtonClicked);

_backToMainMenuButton.onClick.AddListener(InvokeOnBackToMainMenuButtonClicked
);
            _exitButton.onClick.AddListener(InvokeOnExitButtonClicked);
        }

        private void OnDisable() {

_resumeButton.onClick.RemoveListener(InvokeOnResumeButtonClicked);

_backToMainMenuButton.onClick.RemoveListener(InvokeOnBackToMainMenuButtonClic
ked);
            _exitButton.onClick.RemoveListener(InvokeOnExitButtonClicked);
        }
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

МОБІЛЬНА ГРА В ЖАНРІ SHOOTER З ЕЛЕМЕНТАМИ ROGUELIKE

Програма та методика тестування

КПІ.ІП-1214.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ЗАРІЧКОВИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Валерій КОРНІЄНКО

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є гра «Chaos Corridor» в жанрі Shooter з елементами Roguelike. Дане програмне забезпечення призначене для мобільних девайсів на платформах Android та iOS.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка сумісності застосунку з різними операційними системами (Android, iOS);
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу;

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- системне тестування – перевіряється усе програмне забезпечення в цілому;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;
- тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування буде проводитися мануально, без використання сторонніх утиліт.

Порядок проведення тестування буде наступним:

- динамічне тестування на відповідність функціональним вимогам;
- тестування на мобільних пристроях з різною роздільною здатністю екрана;
- тестування на мобільних пристроях з різною версією операційної системи;
- тестування зручності використання;
- тестування роботи застосунку за відсутності з’єднання з мережею.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

МОБІЛЬНА ГРА В ЖАНРІ SHOOTER З ЕЛЕМЕНТАМИ ROGUELIKE

Керівництво користувача

КПІ.ІП-1214.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ЗАРІЧКОВИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Валерій КОРНІЄНКО

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Chaos Corridor» - це мобільна гра в жанрі Shooter з елементами Roguelike. Гра має просунуту процедурну генерацію рівнів та адаптивний ШІ для ворогів. Дана програмне забезпечення доступне для Android та iOS.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність мобільного пристрою з операційною системою на базі Android з версією 6.0 або вище чи iOS з версією 12.0 або вище;

– наявність доступу до Інтернету;

– для встановлення додатку на мобільному пристрої повинно бути не менше 100 МБ вільної пам’яті.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи відповідний apk-файл або ipa-файл для платформи Android та iOS відповідно. Для цього спершу необхідно завантажити його на мобільний пристрій, а потім, використовуючи який-небудь інсталятор, виконати встановлення даного додатку.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку (Рисунок 2.1). У разі, якщо дана іконка не з’явилась, то встановлення відбулось не успішно. Інакше користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись початкова сторінка застосунку.

Рисунок 2.1 – Іконка гри «Chaos Corridor»

3 ВИКОНАННЯ ПРОГРАМИ

Після запуску гри відображається головне меню (Рисунок 3.1). У головному меню присутні дві основні опції. Перша опція «Почати гру» дозволяє завантажити новий рівень. Друга опція «Вийти» завершує роботу застосунку. При натисканні на кнопку «Почати гру» відбувається завантаження ігрового рівня.

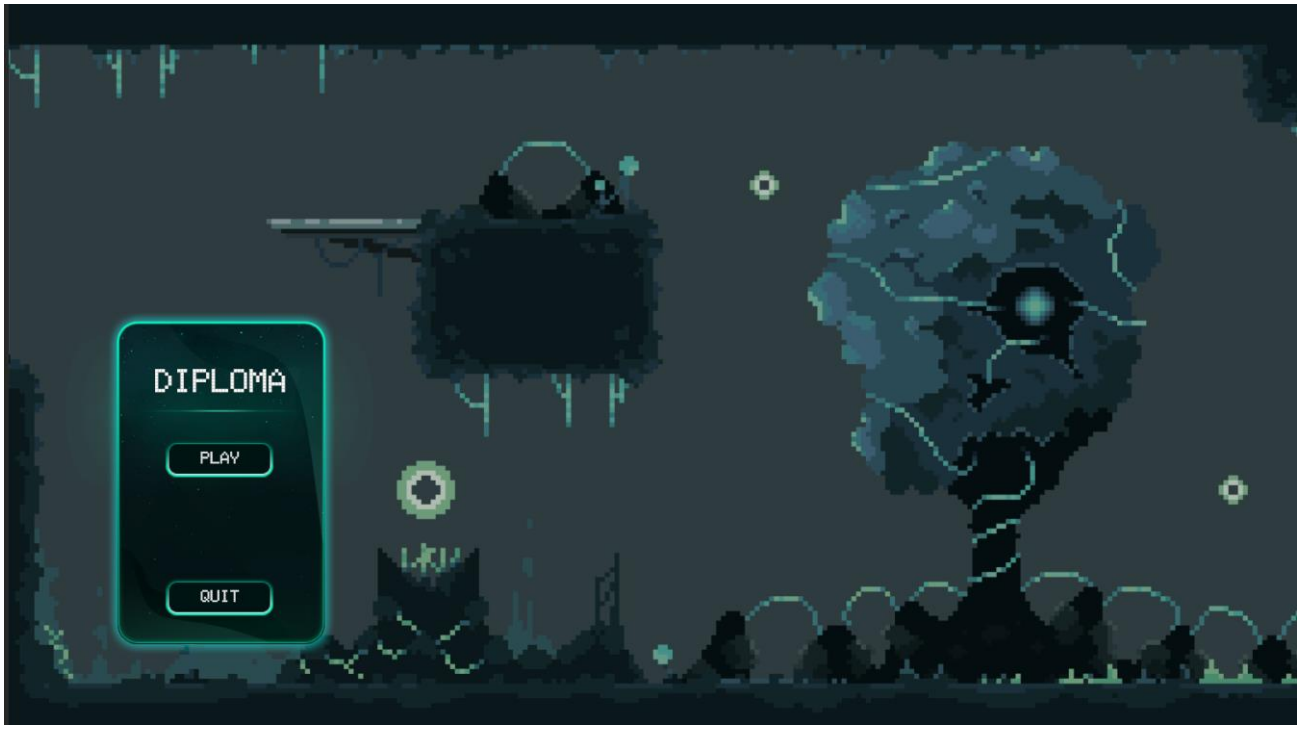


Рисунок 3.1 – Головне меню гри

Після завантаження рівня на екрані відображається головний герой у центрі. Рівень містить згенеровані платформи, декорації, ворогів та елементи графічного інтерфейсу. Зовнішній вигляд гри після завантаження зображений на Рисунку 3.2.

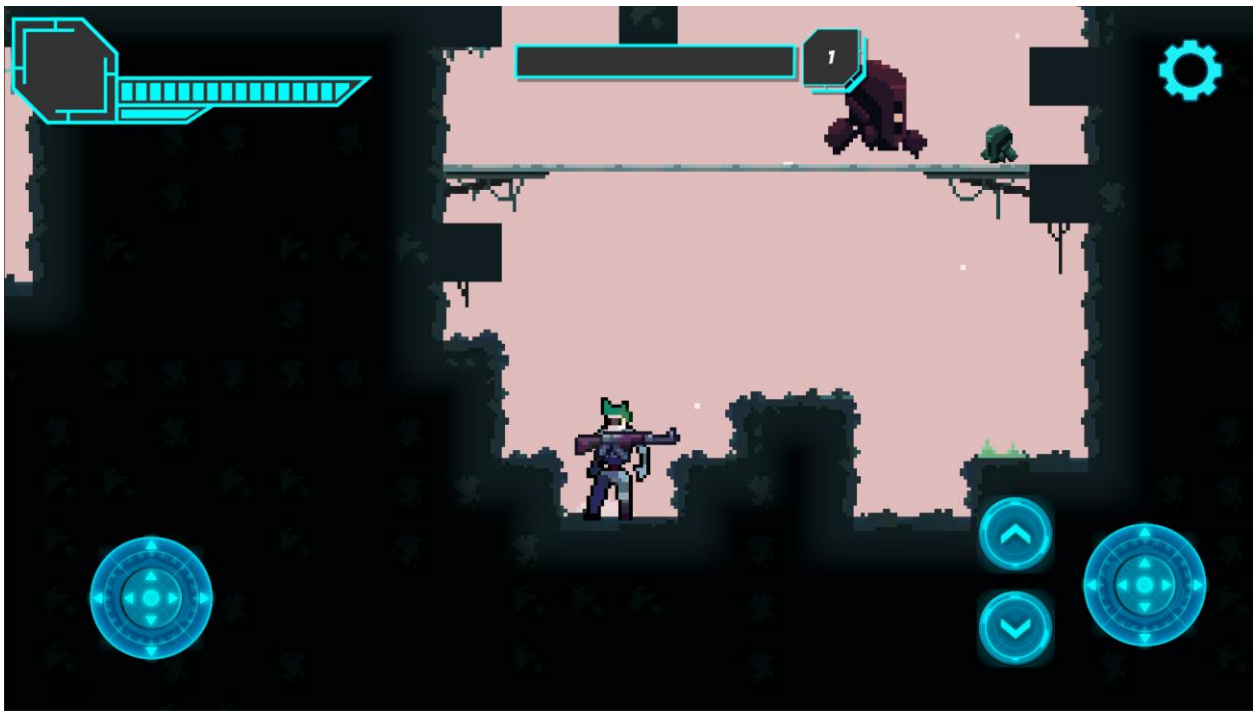


Рисунок 3.2 – Ігровий рівень

Для керування персонажем користувач може використовувати екранні контроли (віртуальні кнопки на екрані). Серед них є 2 екранних джойстика (Рисунок 3.3) та 2 кнопки (Рисунок 3.4). Лівий та правий джойстики відповідають за напрямок руху та напрямок атаки відповідно. Дві кнопки з символами вниз і вгору відповідають за стрибок персонажа та провалювання крізь платформу. Також на дисплеї гравця знаходиться кнопка відкриття внутрішньоігрового меню (Рисунок 3.5).

Рисунок 3.3 – Екранний джойстик

Рисунок 3.4 – Кнопки стрибка та провалювання

Рисунок 3.5 – Кнопка павзи

Також на графічному інтерфейсі зображена шкала здоров’я (Рисунок 3.6) та шкала досвіду (Рисунок 3.7). Шкала здоров’я зменшується при зміні здоров’я персонажа. Шкала досвіду заповнюється при отриманні персонажем досвіду та оновлюється при збільшенні рівня головного героя.

Рисунок 3.6 – Шкала досвіду персонажа

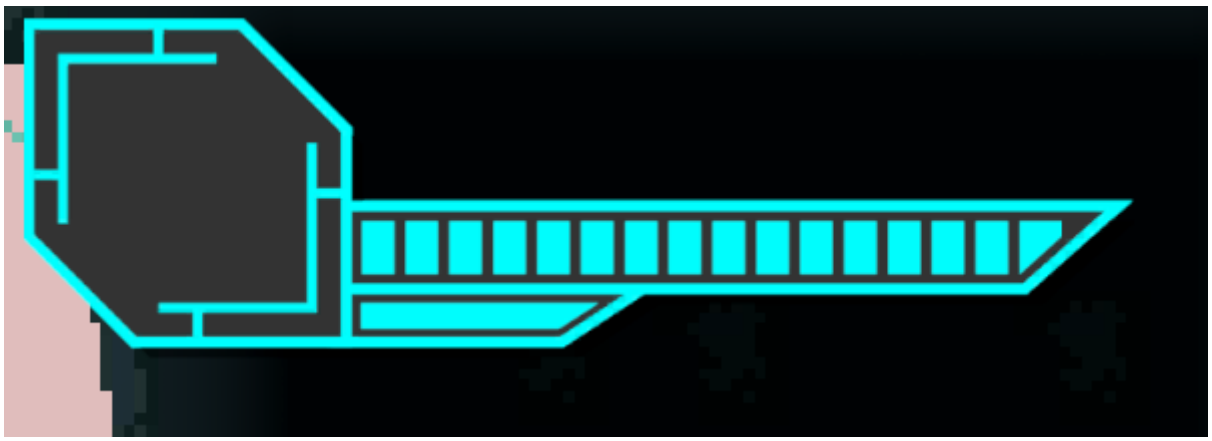


Рисунок 3.7 – Шкала здоров’я персонажа

На рівні користувач зустрічає різноманітних ворогів, кожен з яких має власну поведінку. Приклади ворогів зображені на рисунках 3.8 – 3.10. Кожен ворог має свій особливий паттерн поведінки, але загалом кожен ворог може атакувати гравця. При влученні ворожої атаки персонаж гравця отримує пошкодження, яке відображається на шкалі здоров'я. У випадку, коли атака не влучає у персонажа, гра відображає спеціальний візуальний ефект.

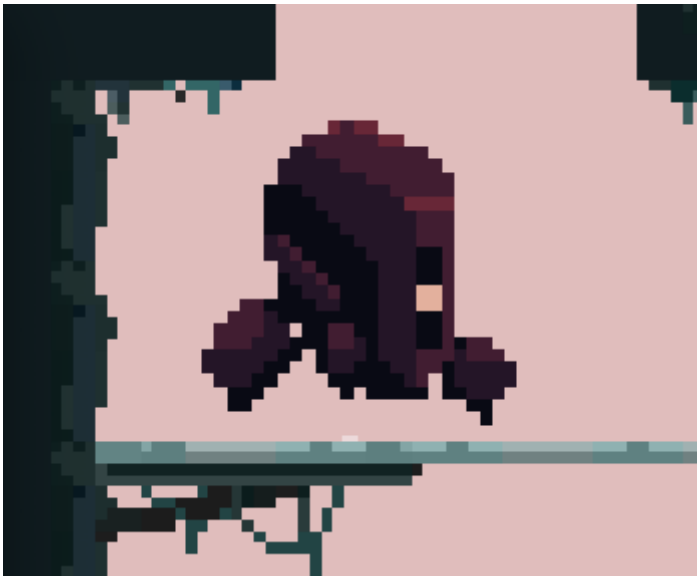


Рисунок 3.8 – Перший приклад ворогу на рівні

Рисунок 3.9 – Другий приклад ворогу на рівні

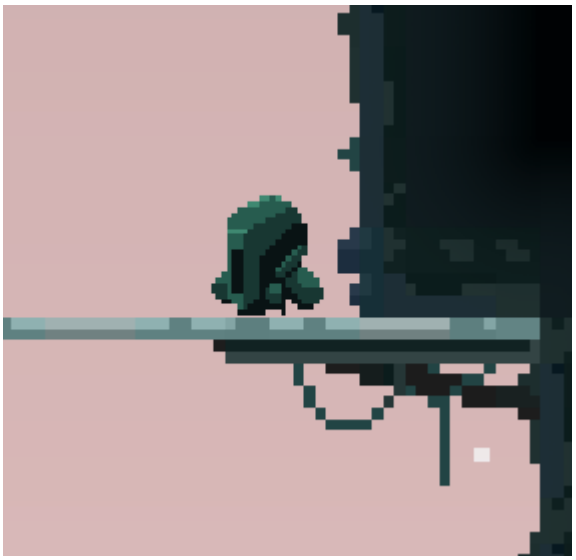


Рисунок 3.10 – Третій приклад ворогу на рівні

Якщо після отримання серії пошкоджень від ворогів здоров'я персонажа знижується до нуля, гра відображає вікно програшу (Рисунок 3.11). У цьому вікні користувачу доступні дві опції: повернутись до головного меню або перезапустити рівень для нової спроби проходження.

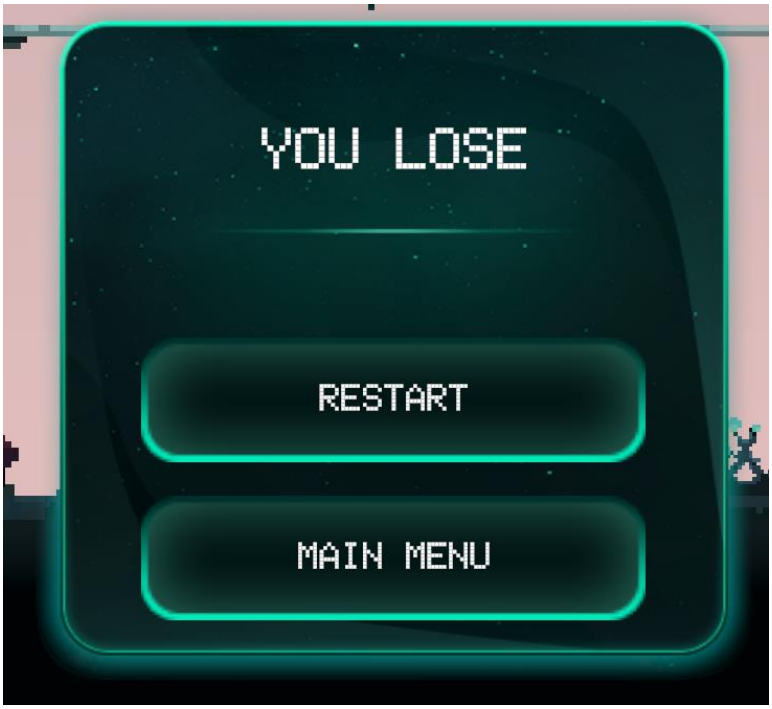


Рисунок 3.11 – Вікно програшу

Після нанесення ворогу критичної шкоди від атак гравця, з переможених ворогів можуть випасти спеціальні об'єкти досвіду. Після підбирання цих об'єктів, шкала досвіду персонажа поступово заповнюється. Заповнена шкала досвіду відображається в інтерфейсі гри (Рисунок 3.12).

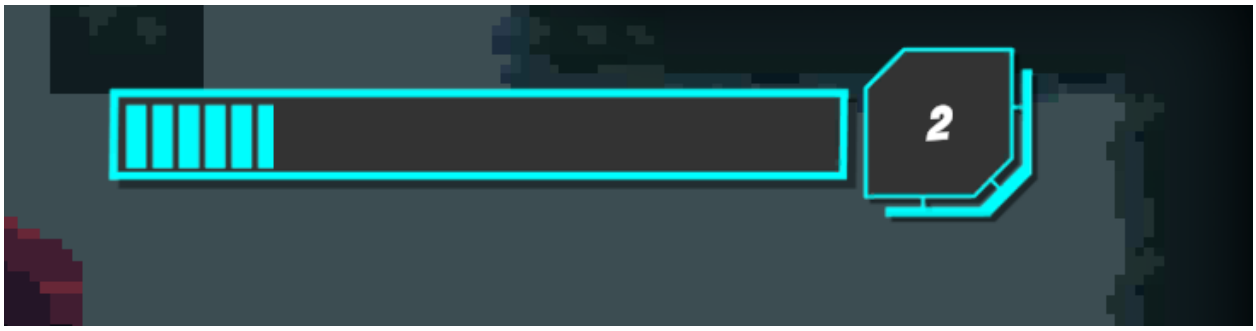


Рисунок 3.12 – Частково заповнена шкала досвіду

При накопиченні достатньої кількості досвіду користувачу відкривається спеціальне вікно з трьома варіантами покращень для персонажа (Рисунок 3.13). Якщо користувач обирає одне з цих покращень, вікно закривається автоматично. Персонаж отримує обране покращення, а показник рівня персонажа збільшується на одну одиницю. Після цього користувач може продовжити ігровий процес.

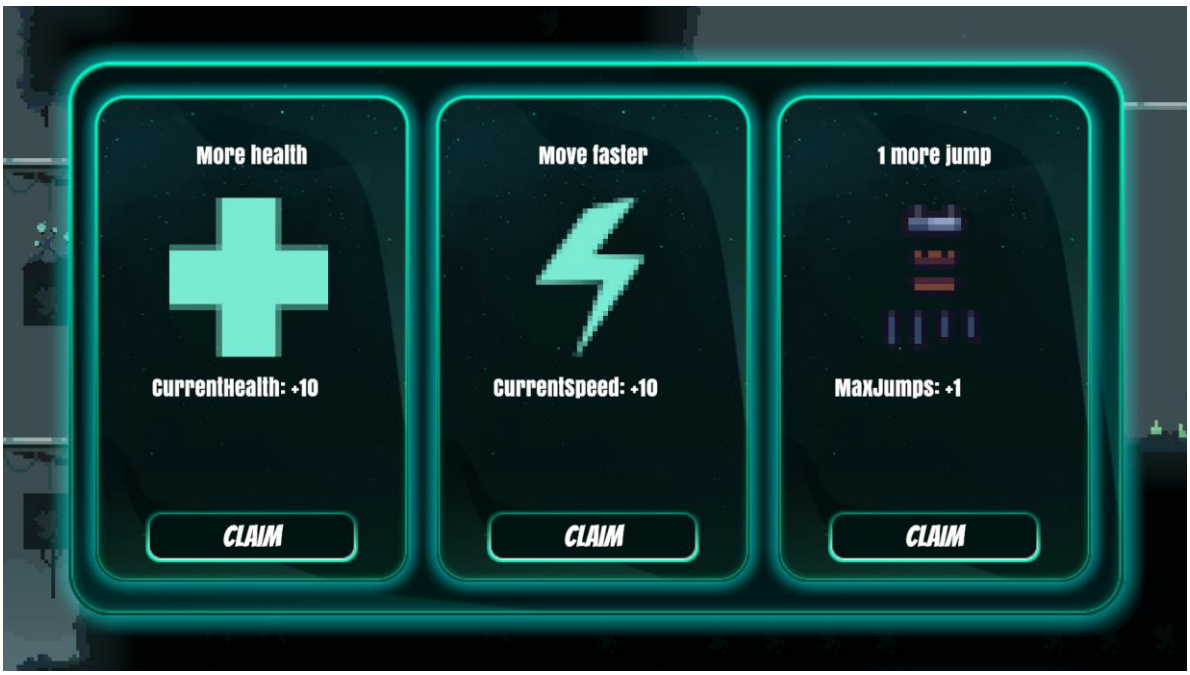
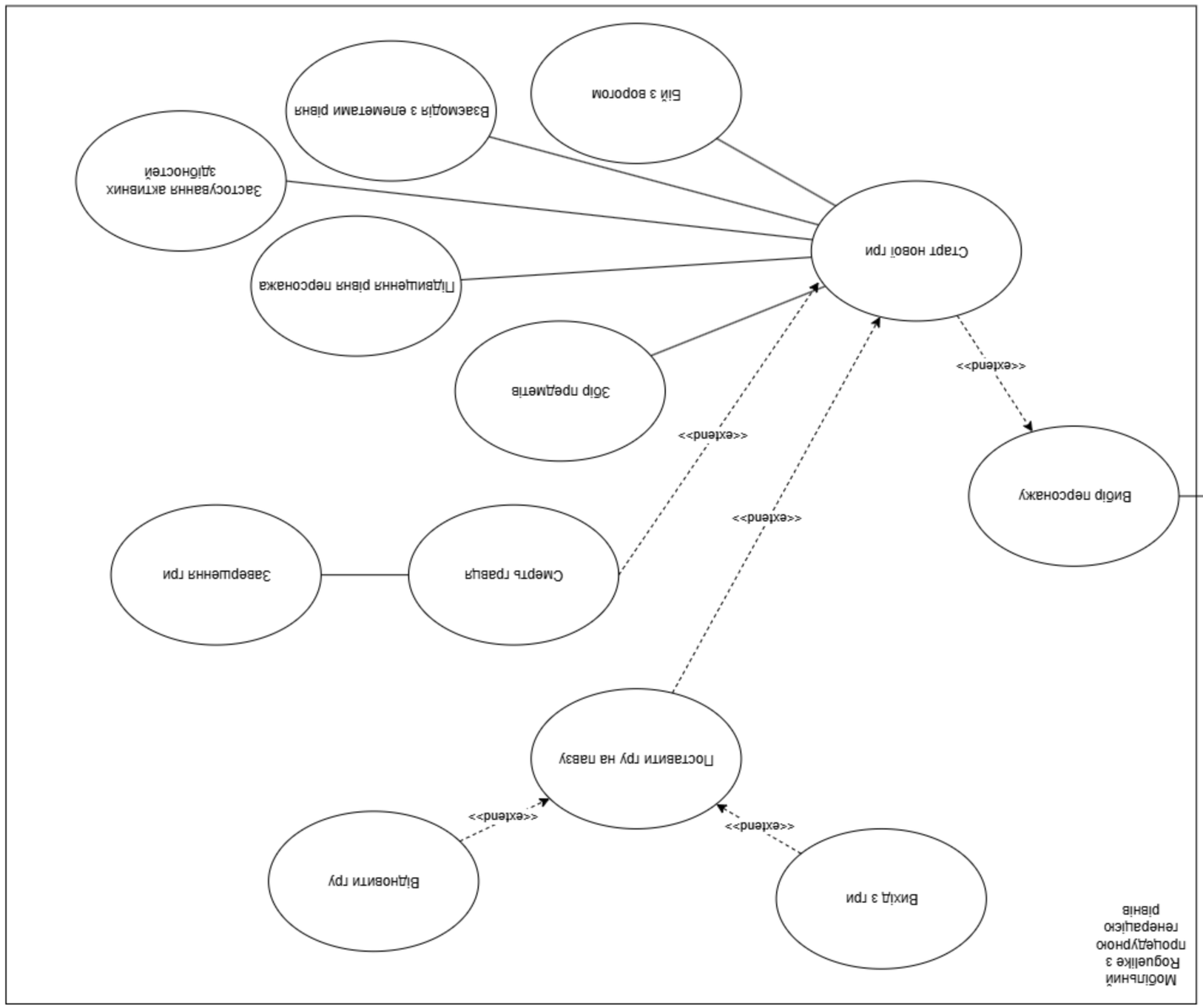


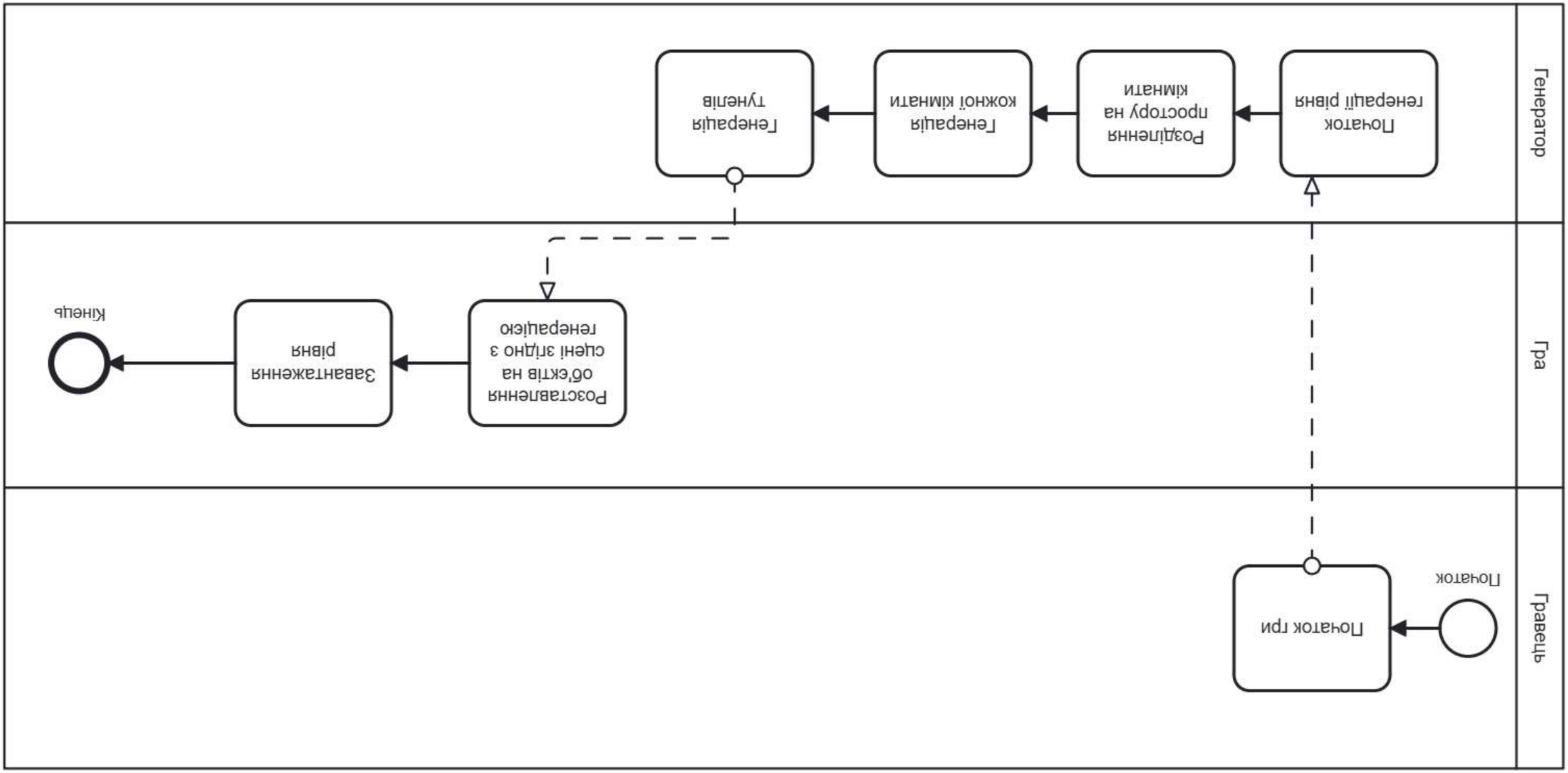
Рисунок 3.13 – Вікно покращень персонажу

Під час проходження гри користувач має можливість відкрити внутрішньо-ігрове меню (Рисунок 3.14), натиснувши відповідну кнопку, яка знаходиться в кутку екрана. У цьому меню доступні три основні опції. Перша опція «Продовжити гру» закриває меню та повертає до поточної ігрової сесії. Друга опція «Вихід у головне меню» завершує поточну гру та повертає до головного меню. Третя опція «Завершити гру» повністю закриває застосунок.

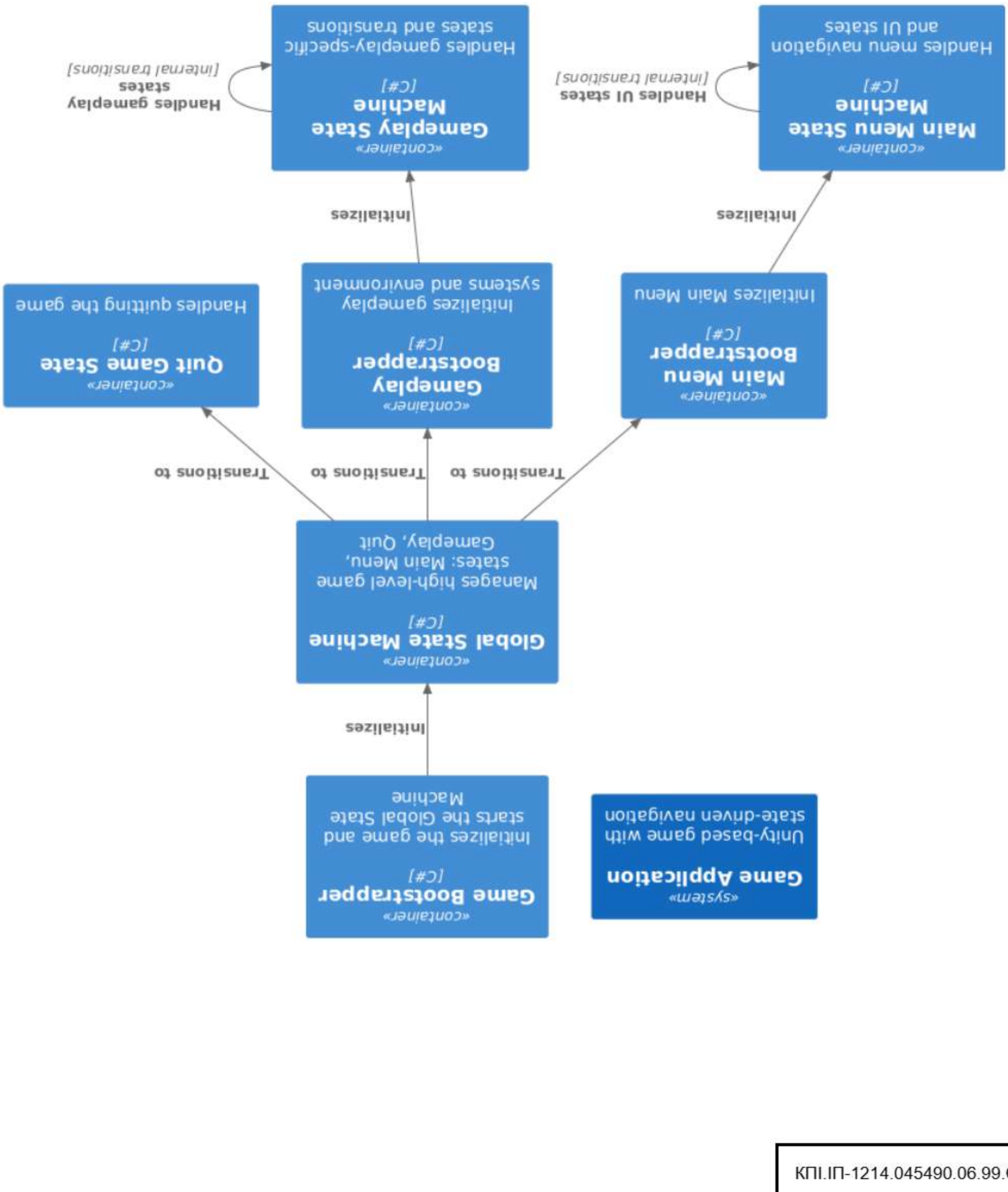
Рисунок 3.14 – Внутрішньоігрове меню



											КПІ.ІП-1214.045490.06.99.ССВ
Зм.	Арк.	№ документа	Підпис	Дата							
Розробив		Корнієнко В.С.			Схема структура варіантів використання				Літера	Маса	Масштаб
Перевірив		Зарічковий О.А.									
Т. контр.									Аркуш 1	Аркушів 1	
Н. контр.		Голованенко М.М.			Мобільна гра в жанрі Shooter з елементами Roguelike				КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12		
Затвердив		Жарікова Е.В.									



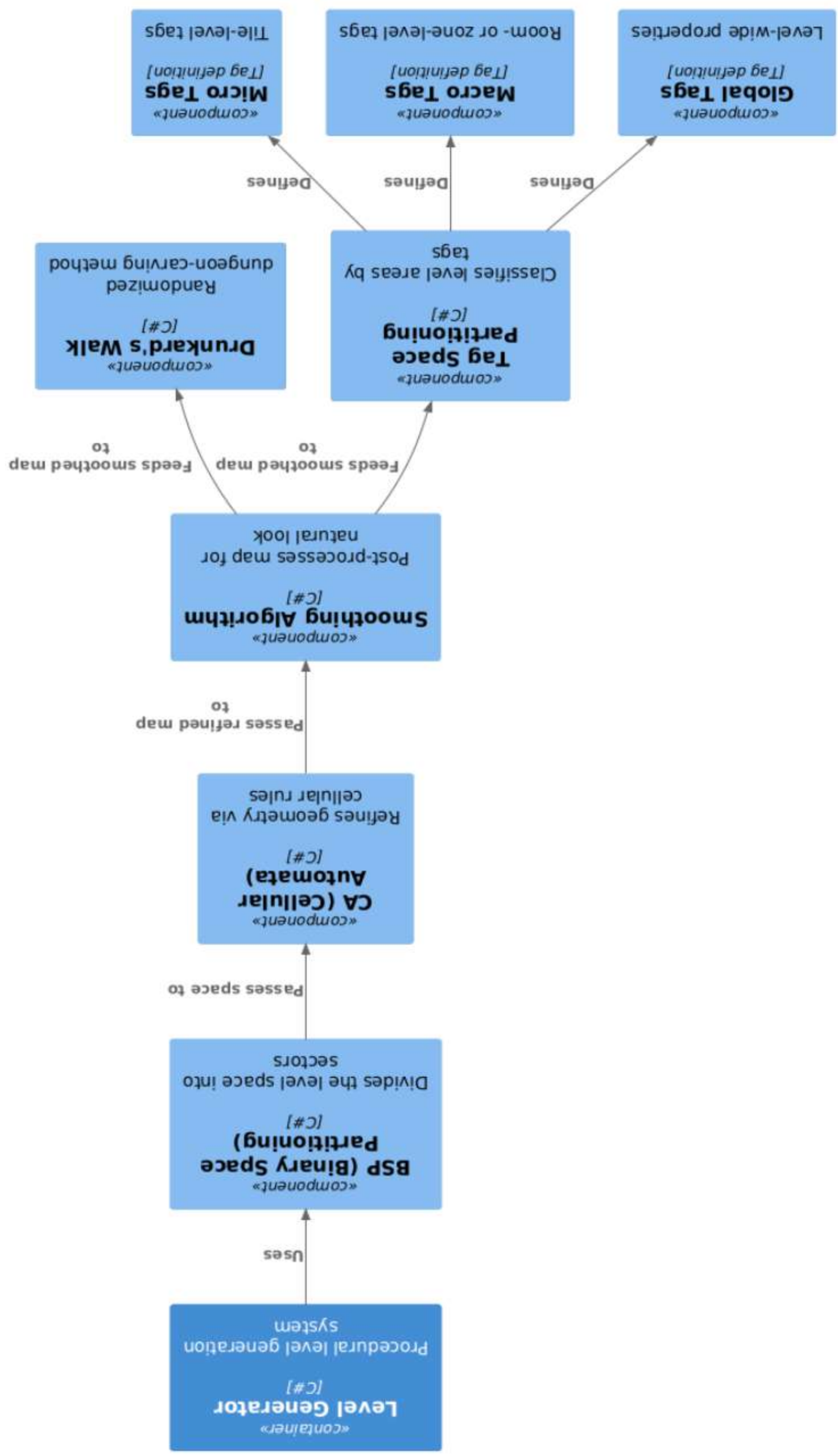
										КПІ.ІП-1214.045490.06.99.ССД	
Літера	Маса	Масштаб									
Аркуш 1		Аркушів 1									
3м.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності						
Розробив		Корнієнко В.С.									
Перевірив		Зарічковий О.А.									
Т. контр.											
Н. контр.		Голованенко М.М.									
Затвердив		Жарікова Е.В.									
					Мобільна гра в жанрі Shooter з елементами Roguelike						
										КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12	

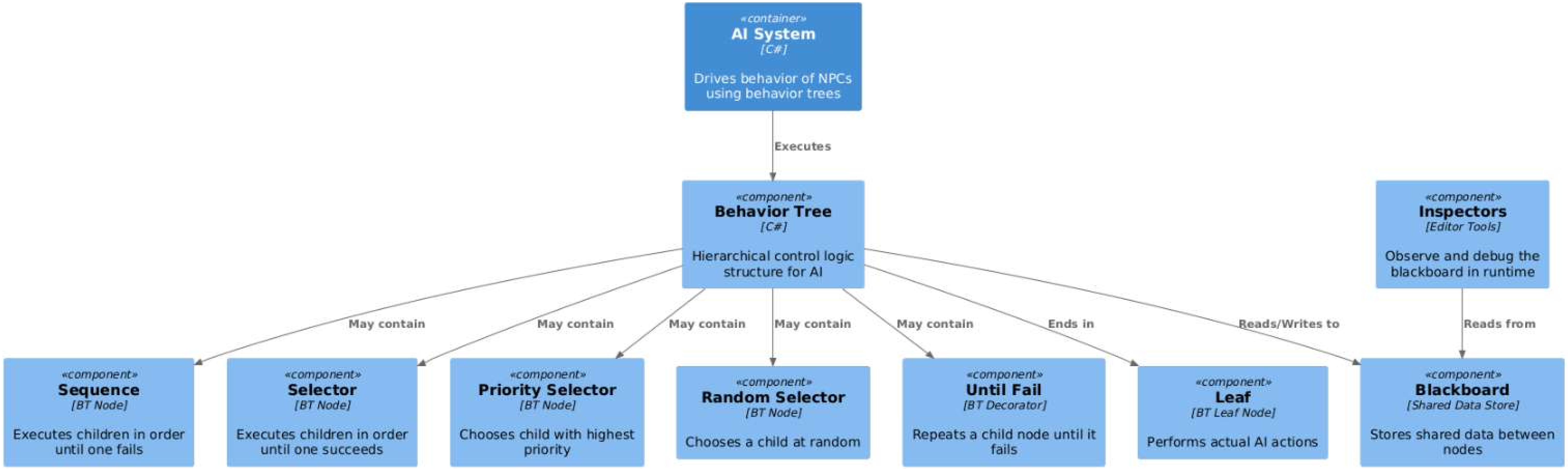


					КПІ.ІП-1214.045490.06.99.ССМ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Корнієнко В.С.			Схема структура компонентів архітектури станів гри
Перевірив		Зарічковий О.А.			
Т. контр.					
Н. контр.		Голованенко М.М.			
Затвердив		Жарікова Е.В.			
					Мобільна гра в жанрі Shooter з елементами Roguelike
					КПІ ім.Ігоря Сікорського кафедра ІПІ гр. ІП-12
Лист		Маса		Масштаб	
					Аркуш 1
					Аркушів 1

					КПІ.ІП-1214.045490.06.99.ССМ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Корнієнко В.С.			Схема структура компонентів процесу генерації рівня
Перевірив		Зарічковий О.А.			
Т. контр.					
Н. контр.		Голованенко М.М.			Мобільна гра в жанрі Shooter з елементами Roguelike
Затвердив		Жарікова Е.В.			КПІ ім.Ігоря Сікорського кафедра ІПІ гр. ІП-12

Літ.	Маса	Масштаб
Аркуш 1	Аркушів 1	





Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІП-1214.045490.06.99.ССМ		
Розробив		Корнієнко В.С.			Схема структурна компонентів архітектури ШІ ворогів	Літера	Маса	Масштаб
Перевірив		Зарічковий О.А.				Аркуш 1	Аркушів 1
Т. контр.					Мобільна гра в жанрі Shooter з елементами RogueLike	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12	
Н. контр.		Головченко М.М.					
Затвердив		Жаріков Е.В.					