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: Ігровий застосунок у жанрі Roguelike

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

Сімчук Андрій Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник професор, д.т.н., засл. діяч, Павлов О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст. викл., д-р філософії, Головченко М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент асистент кафедри ОТ Нечай Д.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

Сімчук Андрію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Ігровий застосунок у жанрі Roguelike
керівник проєкту Павлов Олександр Анатолійович, д.т.н., проф., засл. діяч
(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: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, алгоритмічних та технічних рішень, опис бізнес-процесів.
2) Розроблення вимог до програмного забезпечення: аналіз варіантів використання ПЗ, розробка функціональних та нефункціональних вимог, аналіз системних вимог та економічних показників ПЗ, постановка завдання.
3) Конструювання та розроблення ПЗ: побудова архітектури ПЗ, обґрунтування вибору засобів розробки, конструювання ПЗ.
4) Аналіз якості та тестування ПЗ: статичний аналіз якості ПЗ, опис процесів тестування та наведення контрольного прикладу.
5) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна класів програмного забезпечення

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	20.03.2025	
3	Постановка та формалізація задачі	02.04.2025	
4	Розробка інформаційного забезпечення	10.04.2025	
5	Алгоритмізація задачі	15.04.2025	
6	Обґрунтування вибору використаних технічних засобів	23.04.2025	
7	Розробка програмного забезпечення	10.05.2025	
8	Налагодження програми	13.05.2025	
9	Виконання графічних документів	16.05.2025	
10	Оформлення пояснювальної записки	26.05.2025	
11	Подання ДП на попередній захист	04.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Андрій СІМЧУК
(підпис) (ініціали, прізвище)

Керівник _____ Олександр ПАВЛОВ
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 31 таблицю, 37 рисунків та 24 джерела – загалом 76 сторінок.

Дипломний проєкт присвячено розробці 2D-ігрового застосунку «One Man Army» у середовищі Unity, який поєднує динамічний ігровий процес з інтелектуальною логікою керування персонажем, противниками та внутрішніми ігровими механіками. Актуальність теми зумовлена зростаючим попитом на ігрові продукти з глибокою механікою та розширеною функціональністю, а також можливістю розширення знань у сфері розробки ігрових систем з акцентом на ефективну архітектуру та підтримку.

Метою проєкту є створення функціонального ігрового продукту, що дозволяє користувачу взаємодіяти з багатьма ігровими елементами у динамічному (процедурно згенерованому) середовищі, із можливістю перемикання персонажів, застосуванням інвентарю, складною логікою ворогів, системами атак, ефектів, підсилень тощо.

У першому розділі було проаналізовано особливості розробки 2D-ігор, специфіку роботи з Unity, а також виконано огляд подібних ігрових проєктів. Це дозволило визначити основні функціональні вимоги до системи та сформулювати концепцію проєкту.

Другий розділ присвячено формалізації вимог до програмного забезпечення, побудові діаграм варіантів використання, розробці функціональних та нефункціональних вимог, а також формулюванню технічного завдання на реалізацію гри.

У третьому розділі описано архітектуру системи, побудовано діаграми класів, визначено логіку взаємодії між компонентами гри. Також розглянуто алгоритм перемикання персонажів, взаємодії з ворогами та застосування ефектів у бою, що дозволяє створити глибоку ігрову механіку з можливістю масштабування.

Четвертий розділ містить аналіз якості коду, визначення метрик підтримуваності, надійності, повторюваності коду, а також опис результатів

мануального тестування гри. Наведено приклад проходження ігрового сценарію та оцінено правильність реалізації основної логіки.

У п’ятому розділі представлено процес розгортання проєкту за допомогою Unity, а також описано механізм подальшого супроводу та оновлення програмного продукту. Враховано можливість використання Unity Test Framework для контролю покриття коду та моніторингу стабільності версій.

КЛЮЧОВІ СЛОВА: ІГРОВИЙ ЗАСТОСУНОК, UNITY, ШТУЧНИЙ ІНТЕЛЕКТ, ROGUELIKE, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, МОДУЛЬНІСТЬ, ІГРОВИЙ РУШІЙ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 31 tables, 37 figures and 24 sources – in total 76 pages.

The diploma project is dedicated to the development of a 2D game application “One Man Army” using the Unity engine, combining dynamic gameplay with intelligent logic for controlling characters, enemies, and internal game mechanics. The relevance of the topic is driven by the growing demand for games with deep mechanics and extended functionality, as well as the opportunity to enhance expertise in designing game systems with a focus on efficient architecture and maintainability.

The goal of the project is to create a functional game product that allows users to interact with multiple gameplay elements in a dynamic (procedurally generated) environment, featuring character switching, inventory usage, complex enemy behavior, attack systems, status effects, power-ups, and more.

The first chapter analyzes the features of 2D game development, specifics of working with Unity, and provides a review of similar projects. This analysis helped define the system’s core functional requirements and shaped the overall project concept.

The second chapter formalizes the software requirements, builds use case diagrams, develops functional and non-functional requirements, and defines the technical task for implementing the game.

The third chapter describes the system architecture, provides class diagrams, and explains the logic of interaction between the game’s components. It also discusses the character-switching algorithm, enemy interactions, and effect application in combat, enabling rich game mechanics with scalability potential.

The fourth chapter includes a code quality analysis, evaluation of maintainability, reliability, and duplication metrics, as well as the results of manual testing. An example gameplay scenario is provided, demonstrating the correctness of the main logic implementation.

The fifth chapter presents the project deployment process using Unity and outlines the mechanism for further support and updates of the software product. It also

considers the use of the Unity Test Framework to monitor code coverage and ensure version stability.

KEYWORDS: GAME APPLICATION, UNITY, ARTIFICIAL INTELLIGENCE, ROGUELIKE, PROCEDURAL GENERATION, MODULARITY, GAME ENGINE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1228.045480.01.91	Технічне завдання	14	
3	А4	КПІ.ІП-1228.045480.02.81	Пояснювальна записка	76	
4	А4	КПІ.ІП-1228.045480.03.12	Текст програми	47	
5	А4	КПІ.ІП-1228.045480.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1228.045480.05.34	Керівництво користувача	11	
7	А3	КПІ.ІП-1228.045480.06.99	Графічний матеріал	3	

КПІ.ІП-1228.045480.00.90

Змін.	Арк.	№ докум.	Підп.	Дата
Розроб.		Сімчук А.В.		
Перевір.		Павлов О.А.		
Н. контр.		Головченко М.М.		
Затв.		Жаріков Е.В.		

Відомість дипломного проєкту

Літ.	Аркуш	Аркушів
	1	

КПІ ім. Ігоря Сікорського
ФІОТ каф. ІПІ гр. ІП-12

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ROGUELIKE

Технічне завдання

КПІ.ІП-1228.045480.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____Максим ГОЛОВЧЕНКО

Виконавець:
_____ Андрій СІМЧУК

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Ігровий застосунок в жанрі Roguelike.

Галузь застосування:

Наведене технічне завдання поширюється на розробку ігрового застосунку “One Man Army”, котрий використовується для реалізації динамічного 2D-ігрового процесу з елементами процедурної генерації, стратегічного управління персонажами, боїв із супротивниками та збору ігрових ресурсів та призначений для користувачів, зацікавлених у roguelike-іграх, інді-геймерів, а також для використання в освітньо-дослідницьких цілях студентами, що вивчають розробку ігор у середовищі Unity.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки ігрового застосунку в жанрі 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 використання гри на персональних комп’ютерах з операційною системою macOS кінцевими користувачами в розважальних або навчальних цілях.

Метою розробки є підвищення ефективності генерації ігрового контенту та поведінки ігрових об’єктів шляхом впровадження алгоритмів процедурної генерації рівнів і базових засобів штучного інтелекту. Очікується, що це дозволить автоматизувати створення ігрового середовища, забезпечити варіативність сценаріїв, адаптивну поведінку супротивників, а також загальне покращення якості та реіграбельності ігрового процесу. Покращення процедурної генерації досягається шляхом комбінування двох підходів — шуму Перліна та алгоритму пошуку в ширину (BFS), що дозволяє формувати структуровані, логічно пов’язані ігрові локації.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– головне меню (меню перемоги) та меню налаштувань гри (рис. 4.1 - 4.3);

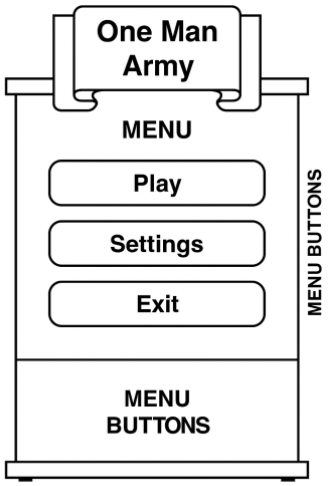


Рисунок 4.1 – Головне меню

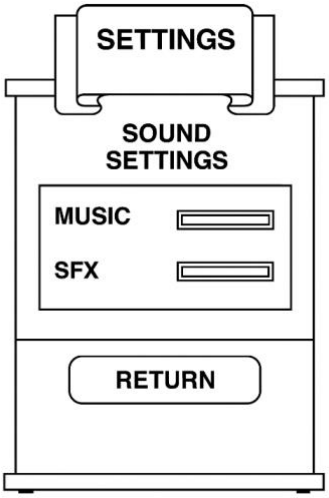


Рисунок 4.2 – Меню налаштувань

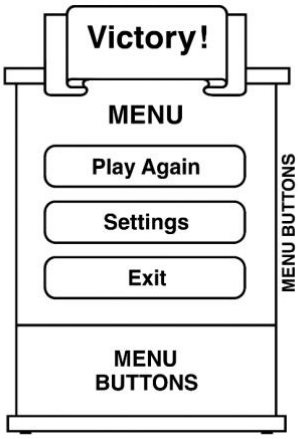


Рисунок 4.3 – Меню перемоги

– основне UI гравця (рис. 4.4);

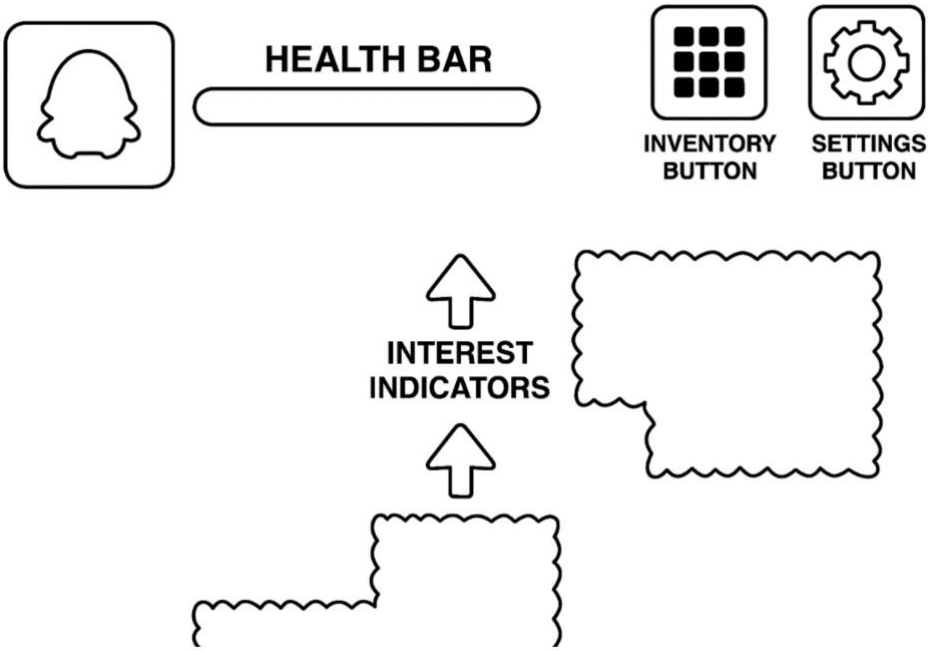


Рисунок 4.4 – Основний інтерфейс гравця

– UI інвентарю де можна побачити наявні предмети і їх опис (рис. 4.5);

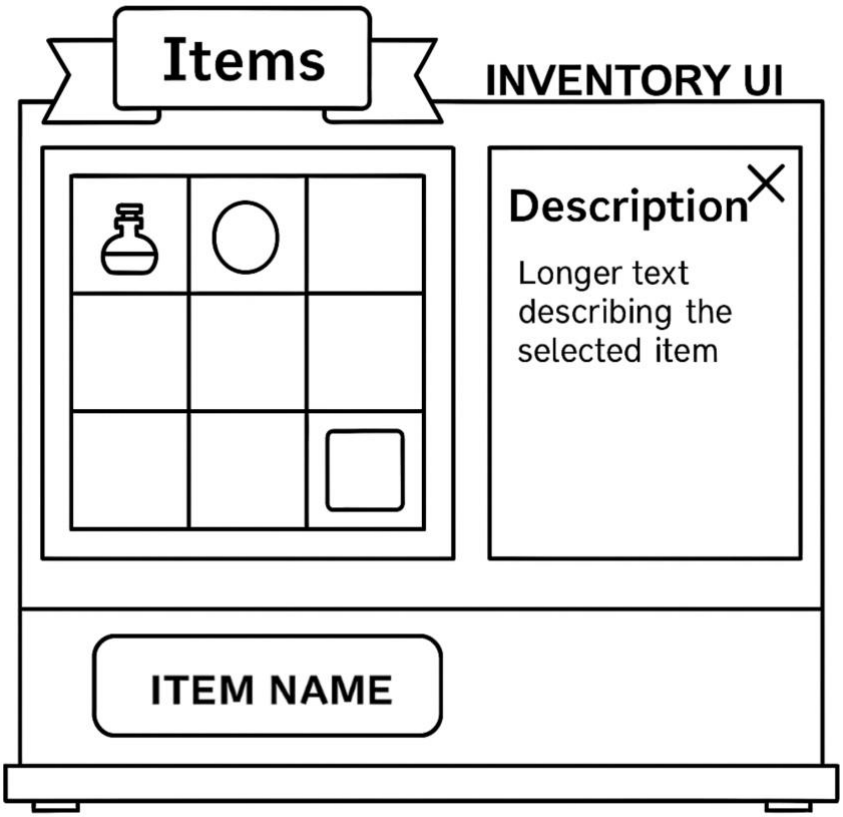


Рисунок 4.5 – Інтерфейс інвентарю

4.1.2 Реалізація функціонування ключових механік гри

4.1.2.1 Для користувача

– переміщення персонажа у межах ігрової зони (за допомогою клавіш або джойстика);

– перемикання між доступними героями (в межах сесії гри);

– атака ворогів ближнього або дальнього бою;

– використання активних та пасивних здібностей;

– отримання зворотного зв’язку про здоров’я, статуси (отруєння, спалення тощо) і також отримані предмети через UI;

– гравець має змогу орієнтуватися в просторі гри завдяки елементам інтерфейсу, зокрема – стрілкам інтересу.

4.1.2.2 Додаткові вимоги

– програмне забезпечення повинно мати можливість розширення та додавання нового функціоналу без необхідності значної переробки існуючого коду;

– гра повинна залишатися стабільною під час тривалого ігрового процесу та не містити критичних помилок, які призводять до аварійного завершення;

– реалізація має забезпечувати підтримку різних роздільностей екрану;

– повинна бути можливість виявлення та логування помилок під час гри для подальшого аналізу;

– архітектура проєкту повинна дозволяти ефективне командне розширення функціональності в майбутньому.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Передбачити низку механізмів, спрямованих на контроль введення інформації користувачем та захист від некоректних дій, які можуть призвести до нестабільної роботи програми або порушення ігрової логіки.

Крім того, у більшості взаємодій із інтерфейсом застосувати валідацію станів – кнопки або елементи неактивні до настання відповідної умови, а переходи між сценами або станами гри можливі лише за правильної послідовності дій.

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 «One Man Army», розроблене на основі ігрового рушія Unity, повинно функціонувати на IBM-сумісних персональних комп’ютерах під керуванням операційної системи Windows, macOS або Linux.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5 або Apple M1;

– об‘єм ОЗП: 4 Гб;

– графічна карта: вбудована, з підтримкою Metal (на macOS) з підтримкою DirectX 11 / OpenGL 4.1;

– вільне місце на диску: не менше 0.3 ГБ.

Рекомендована конфігурація технічних засобів

– тип процесору: Apple Silicon M2 (MacBook Pro 2022+);

– об‘єм ОЗП: 16 Гб;

– графічна карта: вбудована Apple GPU (M2);

– вільне місце на диску: не менше 0.5 ГБ.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно бути сумісним із сучасними операційними системами, які підтримуються ігровим рушієм Unity. Забезпечується робота під управлінням наступних ОС:

- Windows: Windows 10 / 11 (64-bit);
- macOS: macOS 12 Monterey або новіші версії (зокрема підтримка чипів Apple Silicon – M1/M2);
- Linux: Дистрибутиви, сумісні з Unity (наприклад, Ubuntu 20.04+).

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# , яка є основною мовою для створення скриптів у середовищі Unity.

4.5.4 Вимоги до середовища розробки

Розробку Розробку виконано за допомогою ігрового рушія Unity (версія 2022.3 LTS), який надає потужні інструменти для створення 2D-ігор, а також підтримку візуального редактора, фізики, анімації та роботи зі звуком. Для написання, налагодження та аналізу коду використовувалось середовище Visual Studio, яке має інтеграцію з Unity та інструментами статичного аналізу коду.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний Вихідний код програми має бути представлений у вигляді готового проєкту, завантаженого в репозиторій на сайті GitHub.

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 класів програмного забезпечення;
– креслення вигляду екранних форм;
– схема структурна варіантів використань.

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 етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	20.03.2025	
3	Постановка та формалізація задачі	02.04.2025	
4	Розробка інформаційного забезпечення	10.04.2025	
5	Алгоритмізація задачі	15.04.2025	
6	Обґрунтування вибору використаних технічних засобів	23.04.2025	
7	Розробка програмного забезпечення	10.05.2025	
8	Налагодження програми	13.05.2025	
9	Виконання графічних документів	16.05.2025	
10	Оформлення пояснювальної записки	26.05.2025	
11	Подання ДП на попередній захист	04.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

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

Пояснювальна записка
до дипломного проєкту

на тему: **Ігровий застосунок у жанрі Roguelike**

КПІ.ІП-1228.045480.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Roguelike	–	Жанр рольових відеоігор, визначними особливостями якого є випадкове, процедурне створення рівнів, перманентна смерть персонажа у випадку поразки.
NPC	–	Non Player Character, персонаж гри, який контролюється комп’ютером.
RPG	–	Role Play Game, жанр ігор, в якому гравець розвиває свого персонажа унікальним способом (з багатьох доступних).
UI	–	User Interface, засіб зручної взаємодії користувача з інформаційною системою.
FPS	–	Frames Per Second, кількість кадрів на секунду.
ПК	–	Персональний комп’ютер.

ВСТУП

Розробка програмного забезпечення у сфері створення ігрових застосунків залишається актуальною та динамічно розвивається, зокрема завдяки зростанню інтересу користувачів до інноваційних та глибоких геймплейних механік. Одним із напрямів, що привертає увагу як гравців, так і розробників, є ігри в жанрі roguelike, які вирізняються процедурною генерацією рівнів, перманентною смертю персонажа, високим рівнем складності та значною реіграбельністю.

Провідні тенденції у розробці ігор цього типу передбачають використання сучасних ігрових рушіїв, таких як Unity та Unreal Engine, впровадження елементів штучного інтелекту, оптимізацію алгоритмів генерації контенту та вдосконалення системи прогресії персонажа. Успішні проєкти на кшталт Hades, Dead Cells або The Binding of Isaac продемонстрували великий попит на roguelike-ігри серед широкої аудиторії.

Сучасні наукові та дослідницькі установи приділяють увагу вивченню процедурної генерації, поведінкових моделей гравців та побудові балансу в ігрових системах. Вчені та фахівці, зокрема у сфері штучного інтелекту та розробки ігрових механік, активно працюють над підвищенням якості й різноманіття досвіду користувача.

У рамках даного дипломного проєкту буде реалізовано ігровий застосунок у стилі roguelike із базовим набором геймплейних функцій: процедурною генерацією рівнів, системою бою, інвентарем, розвитком персонажа та системою збереження прогресу. Застосунок розроблятиметься з урахуванням принципів оптимізації та можливістю подальшого масштабування.

Призначенням проєкту є створення базової платформи для roguelike-ігрового застосунку, яку можна використовувати як шаблон для майбутніх розробок або як самостійний ігровий продукт. Метою є опанування та застосування сучасних технологій у розробці ігрового ПЗ, вдосконалення

навичок програмування, роботи з ігровими рушіями та застосування алгоритмів процедурної генерації.

Можливими сферами застосування результатів цього проєкту є індустрія комп’ютерних і мобільних ігор, навчальні платформи з елементами гейміфікації, а також розробка прототипів для інді-проєктів.

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;

– 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 якості та тестування програмного забезпечення;

– створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

Індустрія розробки комп’ютерних ігор є одним із найдинамічніших і найприбутковіших напрямів сучасних інформаційних технологій. Ігрове програмне забезпечення охоплює широкий спектр жанрів, що обумовлюється розвитком як апаратних засобів, так і методів програмування, дизайну, генерації контенту та користувацького досвіду. Одним із напрямів, що активно розвивається у межах інді-розробки, є створення ігор у жанрі roguelike.

Жанр *roguelike* (від назви гри *Rogue*, 1980) вирізняється такими ознаками: процедурна генерація рівнів, покроковий або реальний геймплей,

перманентна смерть персонажа (перманентність програшу), варіативність контенту та складність, яка змушує гравця адаптувати стратегію під умови середовища [1].

Сучасні розробки у цьому жанрі активно застосовують:

– процедурну генерацію контенту (рівнів, ворогів, предметів), часто із використанням стохастичних алгоритмів або шумів Перліна;

– елементи RPG (розвиток персонажа, інвентар, навички);

– просту, але функціональну бойову систему;

– піксельну або стилізовану графіку, що дозволяє зменшити витрати на виробництво [2].

На сьогодні використання знань предметної області у сфері IT реалізується через ігрові рушії (Unity, Unreal Engine, Godot), що дають змогу швидко реалізовувати ітеративний підхід до створення ігор, від прототипу до готового продукту. Існує значна кількість відкритих бібліотек, фреймворків та туторіалів, що дозволяють автоматизувати процес генерації контенту та пришвидшити реалізацію типових механік [3].

Недоліки сучасного стану розробки в цій галузі включають:

– складність налаштування збалансованої генерації без втрати ігрового балансу;

– проблеми з ігреабельністю – через випадкову генерацію можуть виникати ситуації, коли гравець отримує занадто легкі або, навпаки, неможливі для проходження рівні, що знижує інтерес до гри;

– необхідність великих обсягів тестування через варіативність сценаріїв;

– високі вимоги до оптимізації коду при процедурному підході;

– обмежена комерційна життєздатність інді-продуктів без сильного візуального чи маркетингового компонента [4].

Можливі шляхи покращення включають використання шаблонів проєктування (Design Patterns), модульної архітектури, застосування

ScriptableObjects у Unity для конфігурації контенту, а також поєднання процедурної генерації з ручною модерацією для покращення якості рівнів.

У рамках даного дипломного проєкту реалізується базова гра в жанрі *roguelike* з акцентом на:

– побудову системи процедурної генерації рівнів;

– створення інтерфейсу користувача;

– реалізацію основного геймплею (пересування, бій, взаємодія з об’єктами);

– застосування структурованої архітектури з поділом логіки на окремі модулі;

– реалізацію трьох ігрових персонажів, на яких однакові предмети впливають по-різному – що створює додаткову тактичну глибину та значно підвищує реіграбельність гри;

– створення основи для подальшого масштабування або використання як шаблону в інших проєктах.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації One Man Army. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

У галузі інді-розробки ігор існує велика кількість представників жанру *roguelike*, які реалізують різні підходи до процедурної генерації, бойової системи, взаємодії з інвентарем та розвитком персонажа. Для порівняння обрано три популярні приклади: Dead Cells, The Binding of Isaac: Rebirth та Hades, які мають спільні риси з розроблюваним дипломним проєктом, проте відрізняються за складністю реалізації, масштабом та візуальним стилем. Для

дослідження досвіду інді-розробників було використано результати опитування GDC [6].

Dead Cells – динамічна 2D roguelike-гра з процедурною генерацією, складними боями та RPG-елементами. Виділяється швидким темпом і великою варіативністю зброї. Також наявний розвиток гілки - інтелект, сила, швидкість (рис. 1.1).

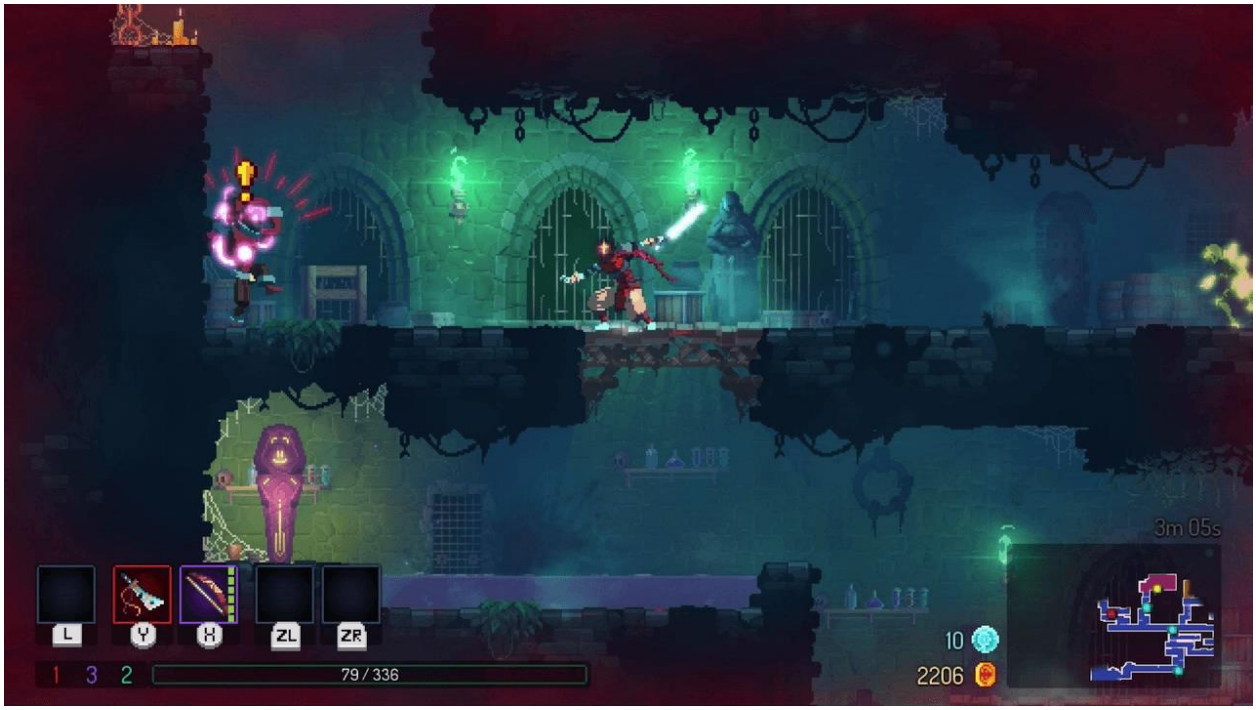


Рисунок 1.1 – Скріншот ігрового процесу Dead Cells

Це ізометрична roguelike-гра, в якій величезна кількість предметів створює нові комбінації ефектів щозапуску. Комбінація абсолютно різних предметів і різних персонажів створює різний досвід при кожному запуску. Генерація рівнів та випадковість – ключові елементи (рис. 2.1).

Рисунок 1.2 – Скріншот ігрового процесу The Binding of Isaac: Rebirth

Hades – roguelike-екшн з ізометричним видом та сильною наративною складовою. У грі реалізовано систему прогресу навіть після поразок, глибоку бойову систему та велику кількість можливостей взаємодії з персонажами (рис 1.3).

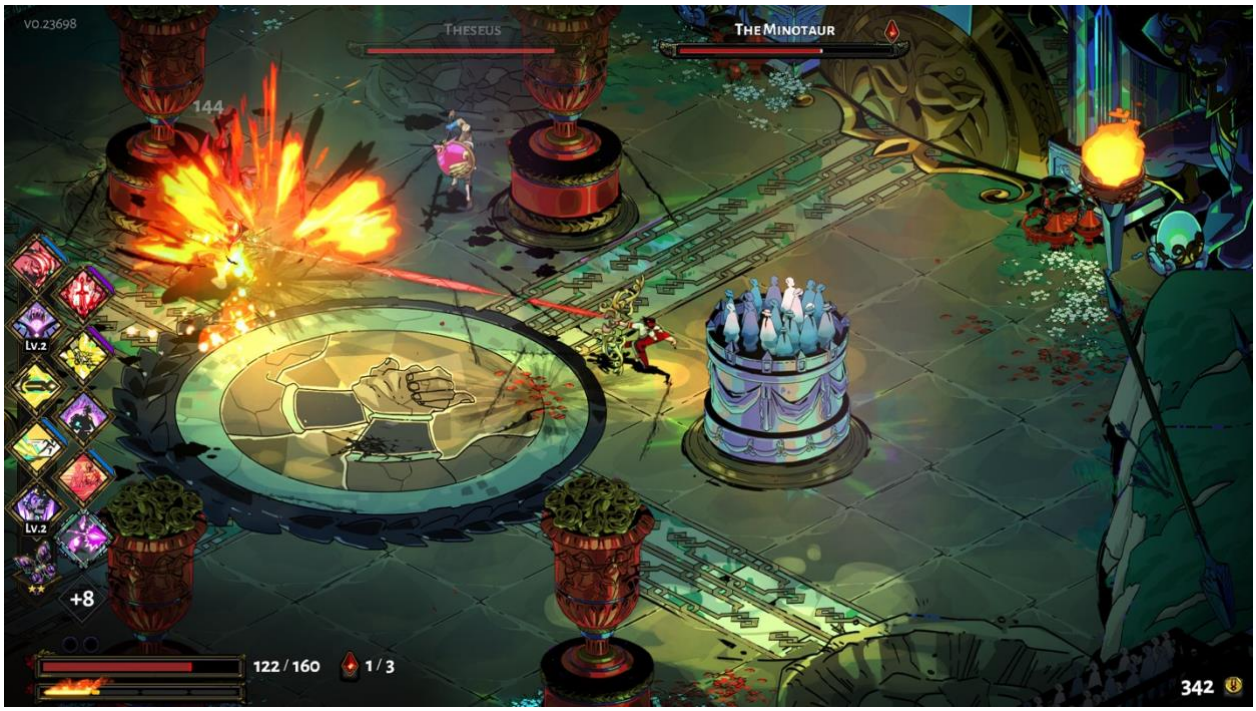


Рисунок 1.3 – Скріншот ігрового процесу Hades

Порівняння проєкту з аналогами можна побачити в таблиці 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	One Man Army	Dead Cells	The Binding of Isaac	Hades	Пояснення
Процедурна генерація рівнів	Так	Так	Так	Так	Усі проєкти використовують генерацію; у One Man Army реалізовано варіант перлін-шуму.
Перманентна смерть	Так	Так	Так	Частково	У Hades реалізовано мета-прогрес після смерті.
Три ігрові персонажі, між якими можна переключатись у процесі гри	Так	Ні	Ні	Ні	Унікальна механіка One Man Army.
Взаємодія з предметами	Так	Ні	Так	Так	One Man Army має унікуальну особливість предметів – вони по-різному впливають на кожного персонажа.

Продовження таблиці 1.1

Функціонал	One Man Army	Dead Cells	The Binding of Isaac	Hades	Пояснення
Розвиток персонажа	Обмежено	Так	Ні	Так	У One Man Army персонаж розвивається двома способами: предмети, які покращують персонажів, збільшення здоров'я після переходу на наступний острів.
Інтерфейс користувача	Так	Так	Так	Так	One Man Army має базовий UI.
2D / Ізометрична графіка	2D	2D	2D	Ізометрія	Усі в 2D окрім Hades.
Система бою	Так (ближній або дальній бій)	Так (динамічна)	Так (шутерна)	Так (екшн-комбо)	У One Man Army – доступні як ближній, так і дальній бій.

Аналіз відомих алгоритмічних та технічних рішень

Процедурна генерація – це метод автоматичного створення контенту за допомогою алгоритмів. У жанрі *roguelike* вона відіграє ключову роль у формуванні непередбачуваного середовища та підвищенні реіграбельності. Особливості побудови roguelike-систем описані у працях Kim A. [17], що стали основою для реалізації внутрішньої логіки гри. Під час аналізу підходів до процедурної генерації контенту було враховано методики, викладені у роботах [3], [18], [23].

Серед найпоширеніших алгоритмів, що використовуються для генерації рівнів:

– Шум Перліна (Perlin Noise) – дозволяє створювати плавні, природні переходи між зонами, використовується в генерації ландшафтів;

– BSP (Binary Space Partitioning) – розділення простору на кімнати, ефективне для побудови коридорних систем;

– Wave Function Collapse (WFC) – сучасний підхід, який використовує правила сусідства для створення шаблоноподібних структур;

– Клітинні автомати (Cellular Automata) – підходять для створення лабіринтів або печер з природною геометрією;

– Flood Fill / BFS (Breadth-First Search) – пошук зв’язаних областей, застосовується для виявлення зон на мапі.

Порівняння алгоритмів процедурної генерації можна побачити в таблиці 1.2.

Таблиця 1.2 – Порівняння з аналогами

Алгоритм	Переваги	Недоліки	Придатність до проєкту
Perlin Noise	Плавний результат, контроль щільності, реалізується швидко.	Складно створювати чіткі кімнати або маршрути.	Використовується як основа.

Продовження таблиці 1.2

BSP	Структурованість, добре підходить для dungeon-like систем.	Важко уникнути повторень, потрібна ручна адаптація.	Можна адаптувати.
WFC	Створює цікаві патерни, підтримує шаблони.	Важка реалізація, складність налагодження.	Непридатний у поточному проєкті.
Клітинні автомати	Натуральні структури, простота реалізації.	Малоконтрольована геометрія, потребує фільтрації результатів.	Обмежена придатність.
Flood Fill / BFS	Швидкий пошук регіонів.	Не формує структуру.	Використовується для зон.

У дипломному проєкті реалізовано модифіковану комбінацію шуму Перліна та кластерного підходу:

– використовується Perlin Noise з пониженням значень на відстані від центру для формування “острова”;

– застосовується Flood Fill (BFS) для виявлення окремих зон (кластерів тайлів);

– реалізовано логіку з’єднання цих кластерів за допомогою мостів через найкоротші можливі шляхи;

– додатково створюються тіні, стіни та об’єкти середовища через постобробку тайлів.

Цей підхід є авторською адаптацією, яка поєднує переваги стандартних методів і оптимізована для потреб 2D-ігрового середовища з Tilemap.

Unity Engine обрано як основну платформу розробки, оскільки:

– підтримує 2D Tilemap, RuleTile, ScriptableObject;

– забезпечує інтеграцію з C#;

– має зручне середовище для візуального редагування та налагодження.

Також наявний A*-алгоритм для ворогів для пошуку шляху по тайловій карті Unity з урахуванням трьох рівнів ландшафту: пісок, трава, гори. В залежності від обраного рівня (EdgeBlocker.Level), навігація можлива лише по відповідній тайлмапі, де тайли не перекриті “вищими” типами рельєфу. Також враховуються “блокуючі” карти (наприклад, перешкоди), які унеможливлюють рух через певні тайли.

Алгоритм працює за стандартною схемою A*: створює відкриту множину вузлів (open), обраховує g (вартість шляху) і h (евристичну відстань до цілі), вибирає вузол із найменшим F = g + h, ітеративно розширює сусідів, перевіряючи, чи вони прохідні. Якщо знаходить шлях до цільової клітинки, то реконструює його у вигляді списку світових координат.

Архітектура ПЗ базується на компонентному підході (MonoBehaviour + ScriptableObject):

– Ігрова логіка поділена на модулі (рух, генерація, UI, інвентар).

– Використано ScriptableObject для гнучкої конфігурації предметів і персонажів.

– Обробка колізій, взаємодії, збереження параметрів – через окремі менеджери.

1.4 Аналіз та моделювання бізнес-процесів

Для моделювання бізнес-процесу використовується BPMN модель (рисунок 1.4).

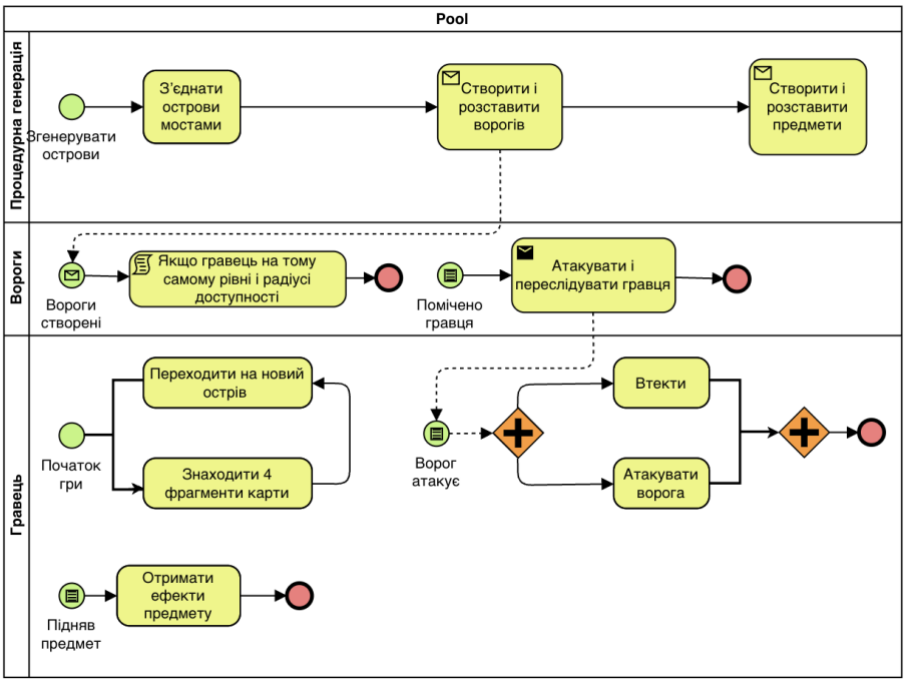


Рисунок 1.4 – Модель бізнес-процесу ігрового процесу One Man Army

Опис моделі бізнес-процесу процедурної генерації:

– система генерує острови за допомогою процедурної генерації;
– система об’єднує острови між собою за допомогою мостів;
– система створює ворогів і розставляє їх по карті;
– система створює та розміщує предмети на ігровій карті.

Опис моделі бізнес-процесу ворогів:

– після створення ворог ставить системою на його місце;
– якщо гравець знаходиться на тому ж рівні та в радіусі атаки виявлення, ворог «помічає» гравця;
– ворог переслідує гравця і намагається атакувати.
– в процесі атаки ворог передає сигнал про атаку гравцеві.

Опис моделі бізнес-процесу гравця:

– гравець починає гру, отримує ціль – знайти 4 фрагменти карти;
– після збору усіх фрагментів, гравець переходить на новий острів;
– при підборі предмета активуються його ефекти;

– якщо відбувається напад ворога, гравець або атакує ворога, або втікає від нього.

Висновки до розділу

У межах даного розділу було здійснено повноцінне теоретичне та технічне обґрунтування ключових аспектів реалізації дипломного проєкту, присвяченого створенню 2D-ігрового застосунку в жанрі *roguelike*.

Насамперед проаналізовано предметну область, окреслено провідні тенденції у розробці ігор цього типу та обґрунтовано доцільність вибору тематики. Було детально описано бізнес-процеси, притаманні даному жанру: процедурна генерація рівнів, система бою, інвентаризація, взаємодія з оточенням тощо. Визначено недоліки сучасного стану розробок у цій галузі, зокрема проблеми з балансом, реіграбельністю та оптимізацією, а також запропоновано шляхи їх подолання.

Проведено порівняльний аналіз трьох відомих ігрових продуктів: *Dead Cells*, *The Binding of Isaac* та *Hades*, а також побудовано таблицю порівняння з дипломним проєктом, що дало змогу визначити унікальні риси власної реалізації (зокрема, три персонажі з різним впливом предметів).

Окремо розглянуто та проаналізовано алгоритми, які застосовуються в ігровій розробці, з фокусом на процедурну генерацію. Порівняно такі методи як шум Перліна, BSP, клітинні автомати, Wave Function Collapse. В обґрунтуваннях зроблено висновок про доцільність використання комбінації алгоритму Перліна та BFS для створення карти з мостами та кластерними зонами. В результаті – реалізовано власний варіант генерації, адаптований до Tilemap-системи Unity.

Також здійснено аналіз архітектурних і технічних рішень. Обрано Unity як рушій для реалізації, C# як мову розробки, та компонентно-модульну архітектуру, яка дозволяє розділити ігрову логіку на незалежні блоки (рух, інвентар, бої, UI). Підтверджено ефективність цього підходу при масштабуванні проєкту.

На завершення створено BPMN-модель основних геймплейних процесів (генерація світу, поведінка ворогів, дії гравця), яка наочно ілюструє логіку взаємодії у грі.

Таким чином, розділ заклав теоретичну та технічну основу для реалізації програмного забезпечення, забезпечивши чітке розуміння методів, інструментів та рішень, що будуть використані у подальшій практичній реалізації.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання з ключовими акторами та основними функціями наведена в графічних матеріалах, креслення 1.

В таблицях 2.1-2.9 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Рух персонажем
Use case ID	UC-01
Goals	Персонаж повинен рухатись.
Actors	Гравець
Trigger	Гравець хоче рухатись в заданому напрямку.
Pre-conditions	Гра активна.
Flow of Events	Гравець натискає кнопки WASD для управління персонажем.
Extension	Персонаж може рухатись лише своїм поверхом.
Post-Condition	Персонаж рухається відповідно до натиснутих кнопок.

Таблиця 2.2 – Варіант використання UC-02

Use case name	Змінити поверх
Use case ID	UC-02
Goals	Персонаж повинен змінити поверх.
Actors	Гравець
Trigger	Гравець хоче змінити поверх.
Pre-conditions	Гра активна, гравець близько драбини.
Flow of Events	Гравець натискає правою кнопкою миші кнопку в грі.
Extension	-
Post-Condition	Персонаж змінює поверх.

Таблиця 2.3 – Варіант використання UC-03

Use case name	Атакувати / Стріляти
Use case ID	UC-03

Продовження таблиці 2.3

Goals	Персонаж повинен атакувати / стріляти.
Actors	Гравець
Trigger	Гравець хоче атакувати / стріляти.
Pre-conditions	Гра активна, вибраний клас персонажа.
Flow of Events	Гравець натискає кнопку F для атаки / стрільби.
Extension	Якщо cooldown атаки / пострілу не закінчився, то атака / постріл не відбудеться.
Post-Condition	Персонаж атакує / стріляє.

Таблиця 2.4 – Варіант використання UC-04

Use case name	Підняти предмет
Use case ID	UC-04
Goals	Персонаж повинен підняти предмет.
Actors	Гравець
Trigger	Гравець хоче підняти предмет.
Pre-conditions	Гра активна, персонаж підходить до предмета.
Flow of Events	Гравець підходить персонажем до предмета.
Extension	Якщо в інвентарі немає місця, предмет не підніметься.
Post-Condition	Персонаж піднімає предмет.

Таблиця 2.5 – Варіант використання UC-05

Use case name	Підняти фрагмент карти
Use case ID	UC-05
Goals	Персонаж повинен підняти фрагмент карти.
Actors	Гравець
Trigger	Гравець хоче підняти фрагмент карти.
Pre-conditions	Гра активна, персонаж підходить до фрагмента карти.
Flow of Events	Гравець підходить персонажем до фрагмента карти.
Extension	Якщо в інвентарі немає місця, фрагмент карти не підніметься.
Post-Condition	Персонаж піднімає фрагмент карти.

Таблиця 2.6 – Варіант використання UC-06

Use case name	Відкрити інвентар
Use case ID	UC-06

Продовження таблиці 2.6

Goals	Персонаж повинен відкрити інвентар.
Actors	Гравець
Trigger	Гравець хоче відкрити інвентар.
Pre-conditions	Гра активна.
Flow of Events	Гравець натискає на кнопку інвентаря, інвентар відривається.
Extension	-
Post-Condition	Персонаж відкриває інвентар.

Таблиця 2.7 – Варіант використання UC-07

Use case name	Перехід на наступний острів
Use case ID	UC-07
Goals	Персонаж повинен перейти на наступний острів.
Actors	Гравець
Trigger	Гравець зібрав 4 фрагменти карти.
Pre-conditions	Гра активна, 4 фрагменти карти зібрані.
Flow of Events	Гравець збирає 4 фрагменти карти і переходить на наступний острів.
Extension	-
Post-Condition	Персонаж переходить на наступний острів.

Таблиця 2.8 – Варіант використання UC-08

Use case name	Пауза
Use case ID	UC-08
Goals	Поставити гру на паузу.
Actors	Гравець
Trigger	Гравець хоче поставити гру на паузу.
Pre-conditions	Гра активна.
Flow of Events	Користувач натискає кнопку паузи.
Extension	-
Post-Condition	Гра ставиться на паузу (неактивний стан), з’являється меню паузи.

Таблиця 2.9 – Варіант використання UC-09

Use case name	Вихід з гри

Продовження таблиці 2.9

Use case ID	UC-09
Goals	Вийти з гри.
Actors	Гравець
Trigger	Гравець хоче завершити гру.
Pre-conditions	Гра знаходить в меню паузи.
Flow of Events	Користувач натискає кнопку паузи і вибирає кнопку виходу.
Extension	-
Post-Condition	Гра переходить в головне меню.

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.10 наведено загальну модель вимог, а в таблиці 2.11 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.10 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Процедурна генерація островів.	FR-1	Високий	Середній
2	З’єднання островів мостами.	FR-2	Високий	Високий
3.1	Система руху персонажа.	FR-3	Високий	Низький
3.2	Система зміни поверху.	FR-4	Високий	Низький
4	Слідування камери.	FR-5	Високий	Низький
5	Атака / стрільба.	FR-6	Високий	Низький
6	Система отримання пошкоджень.	FR-7	Середній	Низький
7	Смерть персонажа.	FR-8	Середній	Низький
8	Піднімання предмета.	FR-9	Високий	Низький
9	Піднімання фрагменту камери.	FR-10	Високий	Низький
10	Отримання ефекту предмету.	FR-11	Високий	Високий
11	Перехід на наступний острів.	FR-12	Високий	Низький
12	Загальний віджет інвентаря.	FR-13	Високий	Низький

Продовження таблиці 2.10

№	Назва	ID Вимоги	Пріоритети	Ризики
13	Загальний віджет паузи.	FR-14	Високий	Низький
14	Штучний інтелект супротивників.	FR-15	Високий	Високий
15	Розкидання супротивників по карті.	FR-16	Високий	Високий
16	Розкидання предметів по карті.	FR-17	Високий	Високий

Таблиця 2.11 – Перелік функціональних вимог

Назва	Опис
FR-1	Процедурна генерація острова. Ігровий рівень автоматично генерується з використанням шуму Перліна, з урахуванням радіуса острівної області. Формуються різні типи поверхні: вода, пісок, трава, гори.
FR-2	З’єднання островів мостами. Алгоритм визначає окремі зони (кластеризовані області) та з’єднує їх за допомогою мостів по найкоротшому доступному шляху.
FR-3	Система руху персонажа. Гравець може керувати персонажем для переміщення по рівню. Реалізовано перевірку колізій, зручну реакцію на натискання клавіш.
FR-4	Система зміни поверху. Гравець може переходити між різними поверхами карти (наприклад, драбиною з нижнього рівня на верхній), з використанням перевірки умов переходу.
FR-5	Слідування камери. Камера динамічно слідкує за гравцем, зберігаючи фокус і плавно коригуючи позицію відповідно до руху.
FR-6	Атака / стрільба. Персонаж має змогу атакувати або стріляти, залежно від класу. Атака викликає анімацію та взаємодіє з ворогами в зоні досяжності.

Продовження таблиці 2.11

Назва	Опис
FR-7	Система отримання пошкоджень. Коли персонаж отримує шкоду від ворога або іншого джерела, зменшується рівень здоров’я. Відображається в UI
FR-8	Смерть персонажа. Після досягнення нуля здоров’я персонаж вважається мертвим, відтворюється відповідна подія (екран поразки).
FR-9	Піднімання предмета. При зіткненні з предметом гравець може його підібрати. Об’єкт зникає з карти, а його ефекти додаються в інвентар або застосовуються.
FR-10	Піднімання фрагменту камери. Фрагменти карти розкидані по рівню. Гравець має зібрати 4 фрагменти для переходу до нового острова. Кожен збір фіксується.
FR-11	Отримання ефекту предмету. Після підбору предмета автоматично активується його ефект (збільшення здоров’я, посилення атаки, тощо). Ефекти залежать від типу персонажа.
FR-12	Перехід на наступний острів. Після збору всіх фрагментів гравець може перейти на новий згенерований рівень з новими ворогами й предметами.
FR-13	Загальний віджет інвентаря. Інвентар представлений у вигляді інтерфейсного віджета, який дозволяє переглядати, застосовувати або видаляти предмети.
FR-14	Загальний віджет паузи. Під час гри доступне меню паузи, в якому можна призупинити гру, переглянути налаштування або вийти в головне меню

Продовження таблиці 2.11

Назва	Опис
FR-15	Штучний інтелект супротивників. Вороги мають простий AI: виявлення гравця, переслідування та атака, при наявності в полі зору, також під час кулдауну атаки відступ.
FR-16	Розкидання супротивників по карті. Після генерації рівня вороги розміщуються на карті в зонах, визначених системою спавну. Враховується кількість та типи ворогів.
FR-17	Розкидання предметів по карті. Предмети (предмети, фрагменти карти) розміщуються випадковим чином у межах доступних тайлів, з урахуванням ймовірності та категорій.

FR/UC	1	2	3	4	5	6	7	8	9
1	+	+							
2	+	+							
3	+	+							
4	+	+							
5	+	+							
6			+						
7			+						
8									
9				+					
10					+		+		
11				+					
12					+				
13						+	+		
14								+	+
15			+						
16	+								
17	+			+					

Рисунок 2.2 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Список нефункціональних умов:

– мінімум 30 FPS в ігровому процесі, якщо ПК відповідає мінімальним системним вимогам;

– відсутність критичних для геймплею багів;

– input-латентність, а сам менше 100 мс;

– використання CPU, менше 20%.

Під час формування функціональних і нефункціональних вимог було враховано стандарти якості програмного забезпечення згідно ISO/IEC 9126 [24].

2.4 Аналіз системних вимог

Мінімальна конфігурація технічних засобів:

– процесор: Intel Core i3 8100 / AMD Ryzen 3 3200G;

– оперативна пам’ять: 4 ГБ;

– відеокарта: Nvidia GT 1030 / AMD Radeon Vega 8 (інтегрована);

– накопичувач: 1 ГБ вільного місця;

– операційна система: Windows 10 (64-bit).

Рекомендована конфігурація технічних засобів:

– процесор: Intel Core i5 10400 / AMD Ryzen 5 3600;

– оперативна пам’ять: 8 ГБ;

– відеокарта: Nvidia GTX 1650 / AMD Radeon RX 580;

– накопичувач: SSD, 1 ГБ вільного місця;

– операційна система: Windows 10 або 11 (64-bit).

2.5 Аналіз економічних показників програмного забезпечення

Так як кількість строк коду не є корисним показником у випадку ігрових проєктів – тут буде використовувати метод функціональних точок.

Список функціональних точок наведено у таблиці 2.12.

Таблиця 2.12 – Список функціональних точок.

№	Назва	ID Вимоги	Complexity
1	Процедурна генерація острова.	FR-1	Complex
2	З’єднання островів мостами.	FR-2	Complex
3	Система руху персонажа.	FR-3	Simple
4	Система зміни поверху.	FR-4	Simple
5	Слідування камери.	FR-5	Simple
6	Атака / стрільба.	FR-6	Average
7	Система отримання пошкоджень.	FR-7	Average
8	Смерть персонажа.	FR-8	Simple
9	Піднімання предмета.	FR-9	Simple
10	Піднімання фрагменту камери.	FR-10	Simple
11	Отримання ефекту предмету.	FR-11	Complex
12	Перехід на наступний острів.	FR-12	Average
13	Загальний віджет інвентаря.	FR-13	Average
14	Загальний віджет паузи.	FR-14	Simple
15	Штучний інтелект супротивників.	FR-15	Complex
16	Розкидання супротивників по карті.	FR-16	Complex
17	Розкидання предметів по карті.	FR-17	Complex

Загальна кількість Simple = 7.

Загальна кількість Average = 4.

Загальна кількість Complex = 6.

UPF (Unadjusted Functional Points) = $7 \cdot 3 + 4 \cdot 4 + 6 \cdot 6 = 73$

Таблиця 2.13 – Technical Complexity Factor

Factor	Weight	Assessment	Impact
Distributed system	2	0	0
Performance objectives	2	4	8
End-user efficiency	1	4	4
Complex processing	1	3	3
Reusable code	1	4	4
Easy to install	0.5	5	2.5

Продовження таблиці 2.13

Easy to use	0.5	4	2
Portable	2	4	8
Easy to change	1	4	4
Parallel processing	1	1	1
Security	1	1	1
Access for third parties	1	2	2
Training needs	1	1	1

TFactor = 8 + 4 + 3 + 4 + 2.5 + 2 + 8 + 4 + 1 + 1 + 2 + 1 = 40.5

TCF = 0.6 + (0.01 · 40.5) = 1.005

Таблиця 2.14 – Environment Complexity Factor

Factor	Weight	Assessment	Impact
Familiar with the development process	1.5	4	6
Application experience	0.5	3	1.5
Object-oriented experience	1	5	6
Lead analyst capability	0.5	2	1
Motivation	1	4	5
Stable requirements	2	4	7
Part-time staff	-1	0	0
Difficult programming language	-1	1	-1

EFactor = 6 + 1.5 + 6 + 1 + 5 + 7 + 0 – 1 = 25.5

EF = 1.4 + (-0.03 · 25.5) = 0.635

Рахуємо Functional Points:

FP = UFP · TCF · EF = 73 · 1.005 · 0.635 = 46.59

Середнє значення годин / FP для інді-розробки – 20.

46.59 · 20 = 931.8 люд/год

931.8 / 40 = 23.3 люд / тиж

23.3 / 4.5 = 5.18 люд / міс

Середня зарплата Junior-розробника 700$:

Вартість розробки = 5.18 · 700$ = 3626$ = 148666 грн.

2.6 Постановка завдання на розробку програмного забезпечення

Метою розробки є створення ігрового застосунку в жанрі *roguelike*, реалізованого на основі технологій Unity, що включає процедурну генерацію

рівнів, просту бойову систему, підтримку декількох ігрових персонажів з індивідуальними характеристиками, інвентаризацію предметів.

Основна ідея проєкту полягає у створенні базової, але функціонально завершеної гри з можливістю проходження рівнів, що кожного разу генеруються випадково, забезпечуючи реіграбельність і варіативність сценаріїв. У межах гри користувач керує одним із трьох доступних персонажів, кожен з яких по-різному реагує на підбір предметів, що також впливає на геймплей. Завданням гравця є дослідження островів, пошук чотирьох фрагментів карти та перехід на наступні рівні.

У рамках дипломного проєкту планується реалізувати:

– систему процедурної генерації ізольованих рівнів-островів із можливістю їх з’єднання;

– логіку спавну та переміщення гравця, включно з атакою та отриманням пошкоджень;

– штучний інтелект супротивників для автоматизованої поведінки у межах рівня;

– систему інвентарю та предметів, що мають різний ефект залежно від обраного персонажа;

– візуальні елементи інтерфейсу: віджет інвентаря, віджет паузи, шкалу здоров’я;

– базову систему переходу між рівнями.

Кінцевою метою розробки є отримання гнучкої програмної основи для подальшого масштабування проєкту або повторного використання окремих компонентів (генерація, інвентар, AI) в інших застосунках ігрового спрямування. Загальні принципи проєктування ігрових систем описані у [10], [11], [21].

Таким чином, розробка ПЗ у межах дипломного проєкту передбачає реалізацію повного циклу функціонального ядра гри, що містить ключові елементи жанру *roguelike* та дозволяє оцінити ефективність обраних архітектурних і алгоритмічних рішень.

Висновки до розділу

У даному підрозділі було сформульовано мету, основну ідею та завдання, що реалізуються в межах дипломного проєкту. Основною метою розробки є створення базової 2D-ігри в жанрі *roguelike*, яка демонструє використання процедурної генерації рівнів, системи персонажів з унікальними реакціями на предмети, елементів бою, інвентаря та інтерактивного інтерфейсу користувача.

У процесі постановки завдання визначено, що програмне забезпечення має забезпечувати повторювану геймплейну цінність за рахунок рандомізації середовища, створення унікальних сценаріїв на кожному проходженні гри та варіативність взаємодії гравця з оточенням. Визначено основні компоненти, які реалізуються: генератор островів з логікою з'єднання, бойова система, штучний інтелект ворогів, система предметів та їхнього впливу, а також UI-елементи.

Окремо сформульовано завдання, які вирішуються у рамках розробки: реалізація управління персонажем, створення різних типів ландшафту, генерація ворогів і предметів, обробка зіткнень, взаємодія з предметами, перехід між рівнями та виведення ігрової інформації через віджети інтерфейсу.

Підсумовуючи, постановка завдання дозволила чітко визначити обсяг та межі розробки, а також сформувати логічну структуру майбутнього програмного продукту, яка буде реалізована у наступних етапах дипломного проєкту.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Загальна архітектура runtime Unity описана на рисунку 3.1.

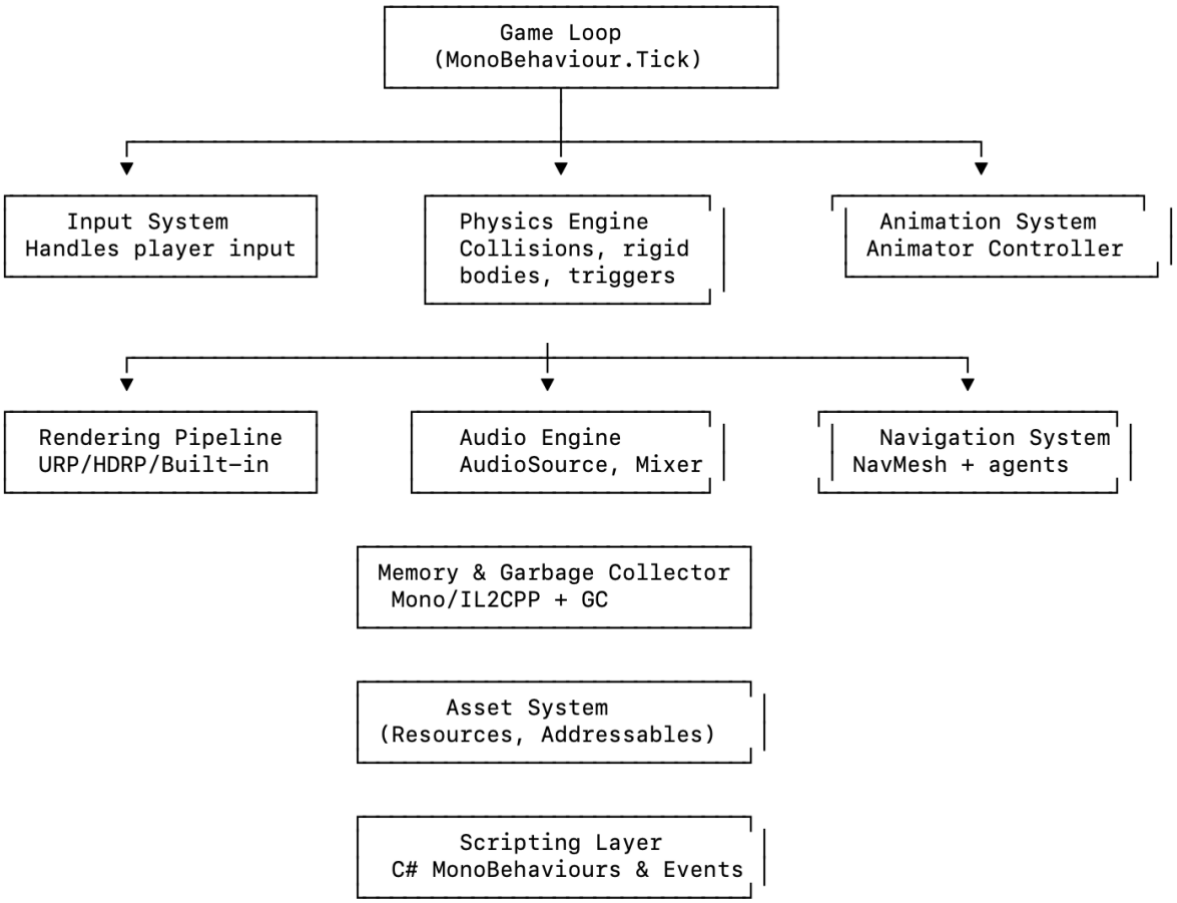


Рисунок 3.1 – Архітектурна схема runtime Unity

Побудову архітектури здійснено з урахуванням шаблонів проєктування з Game Programming Patterns [8] та Game Engine Architecture [15].

Опис компонентів:

– Game Loop – є головним циклом Unity (Update, FixedUpdate, LateUpdate);

– Input System – старий Input або новий Input System (Event-based);

– Physics Engine – Box2D або PhysX (залежно від 2D/3D);

– Animation System – Animator Controller, Mecanim;

– Rendering pipeline – URP, HDRP або Built-in;

– Audio Engine – Відтворення звуків, мікшування;

– Garbage Collector – Звільнення пам’яті Mono/IL2CPP;
– Asset System – Addressables, Resources, Asset Bundles.;
– Scripting Layer – Логіка проєкту, написана на C#.

Діаграма класів проєкту наведено в графічних матеріалах, креслення 2. Класи описано в таблиці 3.1.

Таблиця 3.1 – Опис класів

Назва класу	Опис класу
CharacterSwitcher	Керує перемиканням між ігровими персонажами. Поля: – characters: GameObject[] – масив доступних персонажів; – cameraFollo: CameraFollow – компонент, який слідкує за активним персонажем; – followUI: FollowPlayerUI – інтерфейс, що відображає інформацію про активного персонажа; – selector: CharacterSelector – UI-елемент вибору персонажа; – currentPlayer: GameObject – поточний активний персонаж. Методи: – Start() – ініціалізація;; – Update() – обробка вводу для перемикання; – SwitchTo() – логіка перемикання персонажа; ActivateOnly() – активує лише одного персонажа.

Продовження таблиці 3.1

Назва класу	Опис класу
CameraFollow	Відповідає за стеження камери за об’єктом (наприклад, гравцем). Поля: – target: Transform – посилання на об’єкт, за яким слідкує камера. Методи: – Update() – оновлення позиції камери щокадрово.
FollowPlayerUI	UI-компонент, який вказує на поточного гравця або об’єкт (наприклад, на драбину). Поля: – target: Transform – об’єкт, за яким слідкує індикатор; – ladderLayer: LayerMask – шар, у якому перевіряється наявність драбин; – buttonLabe: TextMeshProUGUI – текстова мітка кнопки; – currentLadder: Collider2D – активна драбина, якщо така є. Методи: – Update() – оновлення UI відповідно до позиції гравця; – TriggerLadderAction() – виклик функції підйому по драбині.

Продовження таблиці 3.1

Назва класу	Опис класу
CharacterSelector	Відповідає за візуальне відображення вибраного персонажа. Поля: – characterImage: GameObject – зображення персонажа; – animator: Animator – анімація вибору. Методи: – SelectCharacter() – вибір персонажа та оновлення UI.
PauseManager	Призначення: Керує паузою, меню, поверненням у гру. Поля: – pauseMenus, pauseButtons: елементи UI; – currentMenu: GameObject – активне меню. Методи: – Pause(), Resume() – зупинка/відновлення; – ChangeMenu(), ExitGame() – логіка перемикання.
Healthbar	Призначення: Відображає здоров’я персонажа. Поля: – healthSlider, effectHealthSlider бари HP; – health: float – значення. Методи: – TakeDamage() – зміна при втраті HP.

Продовження таблиці 3.1

Назва класу	Опис класу
InventorySlotUI	Призначення: Відображає здоров’я персонажа. Поля: – healthSlider, effectHealthSlider: бари HP; – health : float – значення. Методи: – TakeDamage() – зміна при втраті HP.
CharacterStatsBase	Призначення: Характеристики персонажа. Поля: – HP, швидкість, атака, статус-ефекти. Методи: – TakeDamage(), Heal(), Die(); – ApplyBurn/Poison(), HasEffect().
Arrow	Призначення: Снаряд від героя. Поля: – owner: CharacterStatsBase, startPosition. Методи: – OnTriggerEnter2D() – ураження.
CharacterActions	Призначення: Обробка атак, комбо, стрільби. Поля: – stats, anim, attackPoint, enemyLayers. Методи: – TryAttack(), TryShoot(), ComboStrike().

Продовження таблиці 3.1

Назва класу	Опис класу
PlayerSpawner	Призначення: Розміщення гравця на карті. Поля: – playerPrefab, islandGenerator. Методи: – SpawnPlayer(), RestartLevel().
EdgeBlocker	Призначення: Обмеження руху поза карту. Поля: – edgeLevel, edgeCollider. Методи: – TryUpdateCollision(), OnCollisionStay2D().
PlayerMovement	Призначення: Рух гравця і логіка переходів по рівнях. Поля: – stats, movement, newLevel. Методи: – Update(), SetHeightLevel(), Flip().
ArcherStats KnightStats SlaveStats	Призначення: Наслідують CharacterStatsBase, задають унікальні характеристики кожного героя. Методи: – TriggerTransmutationEffect(), Start(), Update().

Продовження таблиці 3.1

Назва класу	Опис класу
EnemySpawner	Призначення: Спавн ворогів на острові за зонами. Поля: – sandEnemies, grassEnemies, mountainEnemies. Методи: DelayedSpawn(), SpawnEnemiesOnTilemap().
IslandGenerator	Призначення: Процедурна генерація островів з різними зонами та об’єктами. Методи: – GenerateIsland(), GenerateBridges(), PlaceEdgeColliders().
ItemSpawner	Призначення: Процедурна генерація островів з різними зонами та об’єктами. Методи: – GenerateIsland(), GenerateBridges(), PlaceEdgeColliders().
ItemEffect	Призначення: Містить ефекти від предметів (наприклад, лікування, підсилення). Методи: – ApplyEffect().
PickupItem	Призначення: Підбір предмета і зникнення з мапи. Методи: – OnTriggerEnter2D(), DestroyAfterDelay().

Продовження таблиці 3.1

Назва класу	Опис класу
InventoryManager	Призначення: Глобальний менеджер інвентаря для UI та ефектів. Поля: – itemNameText, descriptionText, fullMap. Методи: – AddItem(), ReapplyTo(), UpdateInventoryUI().
EnemyAI	Призначення: Поведінка ворогів: рух, атака, чекання. Поля: – allowedLevel, player, rb, stats, animator. Методи: – Update(), IdleOrWalk(), AttackSequence().
EnemyStats	Призначення: Логіка поведінки ворога: пересування, патрулювання, атаки, відступи. Поля: – player EnemyStats, curPath, allowedLevel, retreatTimer. Методи: – Start(), Update(), Attack(), Repath(), FollowPath().

Продовження таблиці 3.1

Назва класу	Опис класу
SimpleGridPathfinder	Призначення: A* пошук шляху по тайловій карті. Враховує рівень і перешкоди. Поля: – level. Методи: – FindPath(), IsTopmostTile(), IsBlocked(), Reconstruct().

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

Основною частиною технологічного стеку при розробці гри є ігровій рушій. Сьогодні для цього є три популярних варіанти: Unity, Godot, Unreal Engine 5. Порівняння цих ігрових рушіїв можна побачити в таблиці 3.2.

Таблиця 3.2 – Порівняння ігрових рушіїв

Рушій	Unity	Godot	Unreal Engine 5
Мова програмування	C#	GDScript, C#, C++ (лише чатково)	C++
Ліцензія	Безкоштовна для некомерційних проектів	Безкоштовна, MIT	Безкоштовна для некомерційних проектів
Вихідний код	Closed Source	Open Source	Open Source
Спільнота	Велика	Маленька	Велика
Якість 2D рендеру	Висока	Висока	Висока

Продовження таблиці 3.2

Рушій	Unity	Godot	Unreal Engine 5
Головний напрям розробки	2D/3D середні за обсягом ігри	2D, середні та малі за обсягом ігри	3D, великі за обсягом ігри

На сьогодні Unity є одним із найпопулярніших рушіїв у сфері розробки інтерактивного програмного забезпечення, зокрема відеоігор, мобільних застосунків та VR/AR-додатків. Його універсальність, кросплатформеність та потужна спільнота розробників роблять його логічним вибором для реалізації проєкту в рамках дипломної роботи.

Основною причиною вибору саме Unity є поєднання зручності, низького порогу входу, широкого набору вбудованих інструментів і гнучкості у використанні C#. Завдяки великій кількості документації та активній спільноті, платформа дозволяє ефективно реалізувати навіть складні концепції, як-от процедурна генерація, система інвентаря чи штучний інтелект супротивників.

Для розробки використовується середовище Unity Editor, яке є офіційною IDE для цього рушія. Написання коду здійснюється в Visual Studio, що інтегрується з Unity за замовчуванням, забезпечуючи підсвітку синтаксису, автодоповнення та відлагодження (debugging).

Для роботи з 2D-графікою, текстурами та візуальними ресурсами використовувались інструменти, сумісні з Unity – наприклад, Blender (для створення або редагування 3D-об’єктів у перспективі подальшого масштабування) або paint.net для роботи з текстурами. Unity підтримує широкий спектр форматів файлів, зокрема .png, .tga, .fbx, що значно спрощує імпорт контенту до сцени.

У рамках даного проєкту рушій Unity надає необхідну технічну базу для реалізації гри в жанрі roguelike, включаючи механіку процедурної генерації, модульну архітектуру, а також інструменти для створення UI, систем бою та управління ресурсами.

3.3 Конструювання програмного забезпечення

3.3.1 Розробка алгоритмів

3.3.1.1 Розробка алгоритму процедурної генерації карти

У процесі розробки було реалізовано алгоритм процедурної генерації карти, що дозволяє створювати унікальний ігровий простір кожного разу при запуску гри. Основою для генерації служить шум Перліна, до якого додається залежність від відстані до центру сцени, що забезпечує природне формування острівної геометрії. Це дозволяє отримати плавні переходи між типами поверхні: вода, пісок, трава та гори. Генерація відбувається відносно поточної позиції камери, що дозволяє масштабувати карту динамічно навколо гравця.

Після формування базового рельєфу карта автоматично розділяється на зони, які можуть бути фізично ізольованими одна від одної. Щоб забезпечити ігрову прохідність, було реалізовано механізм з’єднання таких зон мостами. Для цього використовується пошук найближчих точок між регіонами з урахуванням допустимих напрямків (по горизонталі або вертикалі) та перевіркою колізій, що дозволяє уникнути перетину мостів або некоректного розміщення на воді.

На заключному етапі генерації відбувається побудова додаткових компонентів карти – зокрема, стін, тіней, драбин та фізичних колайдерів для кожного рівня поверхні. Також додаються тіньові ефекти для мостів та меж зон, що підвищує візуальну глибину. Усі об’єкти генеруються через окремі Tilemap-шари, що дозволяє легко управляти зовнішнім виглядом і логікою сцени. Такий підхід забезпечує не лише повторне використання логіки, а й легке масштабування та модифікацію структури ігрового світу.

Окрім генерації самої геометрії острова, алгоритм також передбачає автоматичне розміщення ігрових об’єктів – зокрема, ворогів та предметів. Після побудови основної карти запускається окремий процес розкидання сутностей відповідно до типу місцевості (пісок, трава або гори). Кожен тип території має власний набір ворогів і предметів, які випадково вибираються з

попередньо визначеного списку та розміщуються на доступних тайлах. Розміщення відбувається з урахуванням відсутності перешкод, колізій і близькості до меж карти, що забезпечує баланс гри й уникнення некоректного розташування об’єктів. Такий підхід дозволяє формувати новий розподіл ворогів і луту кожного разу при генерації карти, що підвищує реіграбельність проєкту.

3.3.1.2 Розробка алгоритму руху і дій ворогів

Для забезпечення базової ігрової взаємодії було реалізовано алгоритм руху та дій ворогів, що включає просту модель штучного інтелекту. Кожен ворог має певний тип поведінки, який визначає його активність у межах дозволеного шару (поверхні) острова. У разі виявлення гравця в межах досяжності ворог починає його переслідувати, орієнтуючись на відстань та перешкоди.

У пасивному стані ворог патрулює локальну область, переміщаючись між випадковими точками в межах свого рівня. Після виявлення гравця (в радіусі зору) переходить у стан переслідування, розраховуючи маршрут до цілі за допомогою A*-алгоритму (SimpleGridPathfinder). При цьому враховується не лише прямий напрямок, а й прохідність клітинок, висотний рівень і наявність блокуючих об’єктів.

Всі дії ворога синхронізовані з анімаційними подіями та обробкою шкоди. Під час атаки гравець отримує пошкодження, яке відображається через систему здоров’я. Також враховано застосуван ня ефектів, таких як отруєння, уповільнення та підпал, якщо ворог володіє відповідними здібностями. Такий алгоритм забезпечує базову, але функціональну бойову систему, яку легко масштабувати або ускладнювати у майбутньому.

3.3.2 Опис структури таблиць характеристик

У проєкті як засіб зберігання даних використовуються вбудовані таблиці даних. Опис цих таблиць наведено у таблицях 3.3 – 3.4.

Таблиця 3.3 – Опис таблиці CharacterStatsBase

Назва поля	Тип даних	Опис
level	int	Поточний рівень персонажа
maxHealth	float	Максимальне значення здоров’я
_currentHealth	float	Поточне значення здоров’я
damageResistance	float	Відсоток зниження вхідної шкоди
damageIgnoreChance	float	Ймовірність повного ігнорування вхідної атаки
baseDamage	float	Базова шкода при звичайній атаці
attackRate	float	Кількість атак на секунду
attackRange	float	Дальність атаки
critChance	float	Шанс нанесення критичного удару
critMultiplier	float	Множник шкоди при критичному ударі
isRanged	bool	Чи є атака дистанційною
arrowSpeedMultiplier	float	Множник швидкості стріли
moveSpeed	float	Поточна швидкість персонажа
originalSpeed	float	Початкова базова швидкість
bonusSpeedTimer	float	Таймер дії бонусної швидкості
bonusSpeedAmount	float	Значення бонусу до швидкості
hasAutoSpeedBoost	bool	Чи має автоматичний бонус до швидкості

Продовження таблиці 3.3

Назва поля	Тип даних	Опис
nextAutoBoostTime	float	Час наступного спрацьовування авто-бонусу
isDead	bool	Статус – чи мертвий персонаж
isBurning	bool	Статус – ефект підпалу активний
burnTimer	float	Час до закінчення підпалу
burnDamage	float	Шкода від підпалу
isPoisoned	bool	Статус – ефект отруєння активний
poisonTimer	float	Час до завершення отруєння
poisonDamage	float	Шкода від отруєння
burnOnCrit	bool	Чи застосовується підпал при критичному ударі
burnOnHit	bool	Чи застосовується підпал при звичайному ударі
bonusVsPoisoned	bool	Чи має бонус проти отруєних ворогів
hasToxicBoostWhenLow	bool	Отримує бонус, якщо отруєний і має мало здоров’я
hasBloodRefund	bool	Отримує частину шкоди назад як лікування
hasBeastSpeedBoost	bool	Має шанс на прискорення при пораненні

Продовження таблиці 3.3

Назва поля	Тип даних	Опис
doubleShotChance	float	Ймовірність подвійного пострілу
attacksMarkEnemies	bool	Чи залишає мітки на ворогах
hasSniperBonus	bool	Чи отримує бонус при стрільбі з далекої відстані
sniperBonusApplied	bool	Чи був уже застосований снайперський бонус
sniperBonusRange	float	Мінімальна відстань для бонусу снайпера
sniperBonusDamage	float	Бонусна шкода при великій відстані
attacksSlow	bool	Чи уповільнює атаками
slowDuration	float	Тривалість ефекту уповільнення
slowChance	float	Ймовірність викликати уповільнення
canAutoHeal	bool	Чи може самостійно лікуватись
autoHealCooldown	float	Час кулдауну для самолікування
autoHealPercent	float	Відсоток здоров’я, що відновлюється
lastTimeHealed	float	Час останнього відновлення

Продовження таблиці 3.3

Назва поля	Тип даних	Опис
onKillHealFactor	float	Коефіцієнт лікування при вбивстві
lastDamageTaken	float	Значення останньої отриманої шкоди
lastDamageTakenTime	float	Час останнього отримання шкоди
lastEnemyHitDamage	float	Скільки шкоди було завдано ворогу
lastPosition	Vector3	Координати попередньої позиції персонажа
standingTimer	float	Час бездіяльності (стоячи на місці)
isStandingStill	bool	Чи персонаж зараз нерухомий
retaliatePoisonOnHit	bool	Чи отруює атакуючого у відповідь
comboStrikeEnabled	bool	Чи активовано режим комбо-атаки
hasShieldFortify	bool	Має захист після отриманої шкоди
fortifyDuration	float	Тривалість захисного стану
fortifyTimer	float	Таймер активності захисту
hasPiercingShot	bool	Чи має персонаж проникаючу атаку

Продовження таблиці 3.3

Назва поля	Тип даних	Опис
transmuteReady	bool	Чи доступна трансмутація
transmuteCooldown	float	Кулдаун для трансмутації
transmuteTimer	float	Таймер до повторного використання трансмутації

Таблиця 3.4 – Опис таблиці EnemyStats

Назва поля	Тип даних	Опис
loadSceneOnDeath	bool	Визначає, чи слід завантажити сцену після смерті ворога
sceneToLoad	string	Назва сцени, яку слід завантажити після смерті
maxHealth	float	Максимальне значення здоров’я ворога
_currentHealth	float	Поточний рівень здоров’я ворога
baseDamage	float	Базова шкода, яку завдає ворог під час атаки
isSlowed	bool	Статус – чи ворог сповільнений

Продовження таблиці 3.4

Назва поля	Тип даних	Опис
slowTimer	float	Таймер ефекту уповільнення
speed	float	Поточна швидкість руху ворога
originalSpeed	float	Базова швидкість ворога без ефектів
isDead	bool	Статус – чи мертвий ворог
isPoisoned	bool	Статус – чи ворог отруєний
poisonTimer	float	Таймер ефекту отруєння
isBurning	bool	Статус – чи ворог у вогні
burnTimer	float	Таймер ефекту підпалу
isMarked	bool	Чи відмічений ворог (наприклад, для бонусної шкоди)
originalColor	Color	Базовий колір спрайту ворога
healthbar	Healthbar	Посилання на компонент відображення здоров’я

Продовження таблиці 3.4

Назва поля	Тип даних	Опис
Start()	void	Ініціалізація параметрів при запуску
Update()	void	Оновлення таймерів ефектів, статусів та стану
TakeDamage(float)	void	Обробка отримання шкоди
ApplyPoison(float)	void	Застосування ефекту отруєння
ApplyBurn(float)	void	Застосування ефекту підпалу
ApplySlow(float)	void	Застосування ефекту уповільнення
HandlePoison()	void	Щокадрова обробка отруєння
HandleSlow()	void	Щокадрова обробка уповільнення
HandleBurn()	void	Щокадрова обробка підпалу
IsDead()	bool	Повертає статус смерті
Die()	void	Викликає смерть ворога

Продовження таблиці 3.4

Назва поля	Тип даних	Опис
FlashColor(Color)	void	Тимчасова зміна кольору (наприклад, при отриманні шкоди)
ResetColor()	void	Повернення кольору до початкового
AttackTarget(CharacterStatsBase)	void	Завдає шкоди цілі під час атаки
DisableScripts()	void	Вимикає компоненти після смерті або в особливих станах

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.5.

Таблиця 3.5 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Unity	Основне середовище розробки, у якому реалізовано ігрову логіку, генерацію рівнів, інтерфейс користувача та керування об’єктами сцени.
2	Visual Studio	Інтегроване середовище розробки, що використовувалося для написання коду на C#, його налагодження та інтеграції з Unity.
3	Krita	Графічний редактор, який застосовувався для створення та редагування 2D-ресурсів, зокрема іконок інтерфейсу та текстур.

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

3.4 Аналіз безпеки даних

One Man Army є однокористувацьким ігровим застосунком, який поширюється за принципом «as-is» – тобто в поточному вигляді, без гарантій з боку розробника щодо технічної підтримки, стабільності або оновлень. Оскільки гра не передбачає мережевої взаємодії, реєстрації облікових записів чи обміну даними з віддаленими серверами, потенційні ризики безпеки зведені до мінімуму. Усі дії в грі виконуються локально на пристрої користувача, тому відповідальність за збереження даних, захист пристрою та системні обмеження повністю покладається на самого користувача. Розробка не включає систем збору чи обробки персональної інформації, що додатково зменшує потребу у впровадженні окремих засобів кіберзахисту.

Висновки до розділу

У даному розділі було розглянуто процес реалізації ключових функціональних компонентів ігрового проєкту «One Man Army», створеного на рушії Unity. Основну увагу приділено процедурній генерації ігрової карти, поведінці ворогів, логіці дій гравця, а також системам інтерфейсу, бою, інвентаря та обробки статусних ефектів. Розробка охоплює як геймплейну частину, так і технічні аспекти організації архітектури, коду та внутрішньої взаємодії об’єктів.

Алгоритм процедурної генерації забезпечує створення унікального острова з різними біомами (пісок, трава, гори), автоматичним додаванням мостів між зонами, генерацією стін, драбин, колайдерів і тіней. Після створення середовища, окремі модулі відповідають за розміщення ворогів і предметів на карті залежно від типу поверхні. Це дозволяє досягти високої варіативності й реіграбельності.

У грі реалізовано повноцінну систему управління персонажем: рух між рівнями висоти, бойові дії (атака, стрільба, комбо), отримання шкоди та смерті. Інвентар дозволяє підбирати предмети, які впливають на характеристики кожного з трьох персонажів індивідуально. Вороги мають базовий ШІ, що передбачає патрулювання, переслідування гравця та атаку з подальшим відступом під час кулдауну. Для UI реалізовані елементи здоров’я, інвентаря, паузи, та підказки взаємодії з об’єктами. Усі системи реалізовано у вигляді окремих модулів із можливістю масштабування або повторного використання в інших проєктах.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Метриками для оцінки якості ПЗ обрано наступні:

– надійність (reliablility) – відсутність неточностей у роботі застосунку, ПЗ завжди видає очікувані (передбачувані результати);

– підтримуваність (maintainability) – зрозумілість та модульність коду, легкість внесення змін до нього та його супроводу;

– повторюваність коду (duplications) – відсутність повторювань великих фрагментів коду (винесення такої логіки в окремі методи).

Метрики якості коду оцінювалися згідно з критеріями, наведеними у [24]. Unity Manual [5] та Script Reference [12] використовувалися для перевірки відповідності стандартам рушія.

Код проєкту реалізовано з дотриманням базових принципів стійкості:

– ключові методи (Start, Update, TakeDamage, Die) містять перевірки на null, обробку помилок (try-catch) або логування винятків;

– в обробці зіткнень (OnTriggerEnter2D, OnCollisionStay2D) передбачено перевірку відповідності об’єктів та статусів;

– більшість компонентів (наприклад, CharacterStatsBase, EnemyStats) мають внутрішні обмеження й умови для уникнення некоректної логіки (наприклад, уникнення атаки після смерті).

Код проєкту добре структурований:

– використовуються окремі класи для ролей (CharacterActions, PlayerMovement, InventoryManager, EnemyAI тощо);

– є приклади модульної архітектури – більшість класів мають вузьку відповідальність;

– назви змінних і методів описові (наприклад, ActivateBonusSpeed, TriggerTransmutationEffect);

– основна логіка рознесена по методах середньої довжини – рідко перевищує 50 рядків, що спрощує читання й редагування.

У проєкті OneManArmy загалом спостерігається добре структурований код, однак окремі повторювані патерни – такі як ініціалізація компонентів, виклики методів Start, TriggerTransmutationEffect, Die – повторюються у схожих класах. Це пояснюється уніфікованим підходом до реалізації поведінки персонажів, що робить логіку послідовною та передбачуваною.

Такий рівень повторення є прийнятним для Unity-проєкту з подібною архітектурою та навіть покращує читабельність для нових розробників, оскільки структура методів однакова в кожному персонажі.

У перспективі, при зростанні проєкту, частину спільної поведінки можна вивести в базовий клас або утиліти, що дозволить ще більше скоротити дублювання без втрати ясності коду.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.11.

Таблиця 4.1 – Тест 1.1 Початок гри користувача

Початковий стан системи	Користувач знаходиться в початковому меню.
Вхідні дані	Натиснення кнопки “Play”.
Опис проведення тесту	Користувач знаходиться в головному меню. Користувач натискає кнопку “Play”.
Очікуваний результат	Гравця переносить на перший рівень.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Рух гравця

Початковий стан системи	Користувач знаходиться на ігровому рівні.
Вхідні дані	Натиснення кнопок руху.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і натискає кнопки руху.
Очікуваний результат	Гравець рухається.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Зміна поверху гравця

Початковий стан системи	Користувач знаходиться на ігровому рівні.
Вхідні дані	Натиснення кнопки на драбині.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і натискає кнопку на драбині.
Очікуваний результат	Гравець переміщується на інший поверх.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Зміна персонажів

Початковий стан системи	Користувач знаходиться на ігровому рівні.
Вхідні дані	Натиснення кнопки зміни персонажа.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і натискає кнопку зміни персонажа.

Продовження таблиці 4.4

Очікуваний результат	Персонаж змінюється.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Атака на ворога

Початковий стан системи	Користувач знаходиться на ігровому рівні і атакує ворога.
Вхідні дані	Натиснення кнопки атаки.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і натискає кнопку атаки поруч з ворогом.
Очікуваний результат	Ворог отримує шкоду.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Отримування шкоди

Початковий стан системи	Користувач знаходиться на ігровому рівні і отримує шкоду.
Вхідні дані	Знаходження близько супротивника.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і близько супротивника.
Очікуваний результат	Ворог атакує і наносить шкоду гравцеві.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Піднімання предмету

Початковий стан системи	Користувач знаходиться на ігровому рівні і підходить до предмету.
Вхідні дані	Користувач заходить персонажем на предмет.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і підходить до предмету.
Очікуваний результат	Предмет підбирається в інвентар.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Програш гравця

Початковий стан системи	Користувач знаходиться на ігровому рівні і отримає достатньо шкоди для програшу.
Вхідні дані	Гравець отримує достатньо шкоди для програшу.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і отримує достатньо шкоди для програшу.
Очікуваний результат	Відкривається меню програшу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Знищення ворога

Початковий стан системи	Користувач знаходиться на ігровому рівні і наносить достатню кількість шкоди для знищення ворога.
Вхідні дані	Користувач наносить достатню кількість шкоди для знищення ворога.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і наносить достатню кількість шкоди для знищення ворога.

Продовження таблиці 4.9

Очікуваний результат	Ворог знищується.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.10 Перехід на новий рівень

Початковий стан системи	Користувач знаходиться на ігровому рівні і збирає 4 фрагменти карти.
Вхідні дані	Користувач збирає 4 фрагменти карти.
Опис проведення тесту	Користувач знаходиться на ігровому рівні і збирає 4 фрагменти карти.
Очікуваний результат	Гравця переносить на наступний острів (рівень).
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.11 Перемога гравця

Початковий стан системи	Користувач знаходиться на останньому ігровому рівні і перемагає боса.
Вхідні дані	Користувач перемагає боса.
Опис проведення тесту	Користувач знаходиться на останньому ігровому рівні і перемагає боса.
Очікуваний результат	Гравця переносить на панель перемоги.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

У контрольному прикладі буде описано процес гри гравця також можливість його програшу або виграшу. При роботі з графікою орієнтувався на Krita Manual [9].

Відкривається ігровий застосунок з стартового меню (рис. 4.1).

Рисунок 4.1 – Стартове меню

Після натиснення кнопки «Play» гравець потрапляє на перший острів (рис. 4.2).

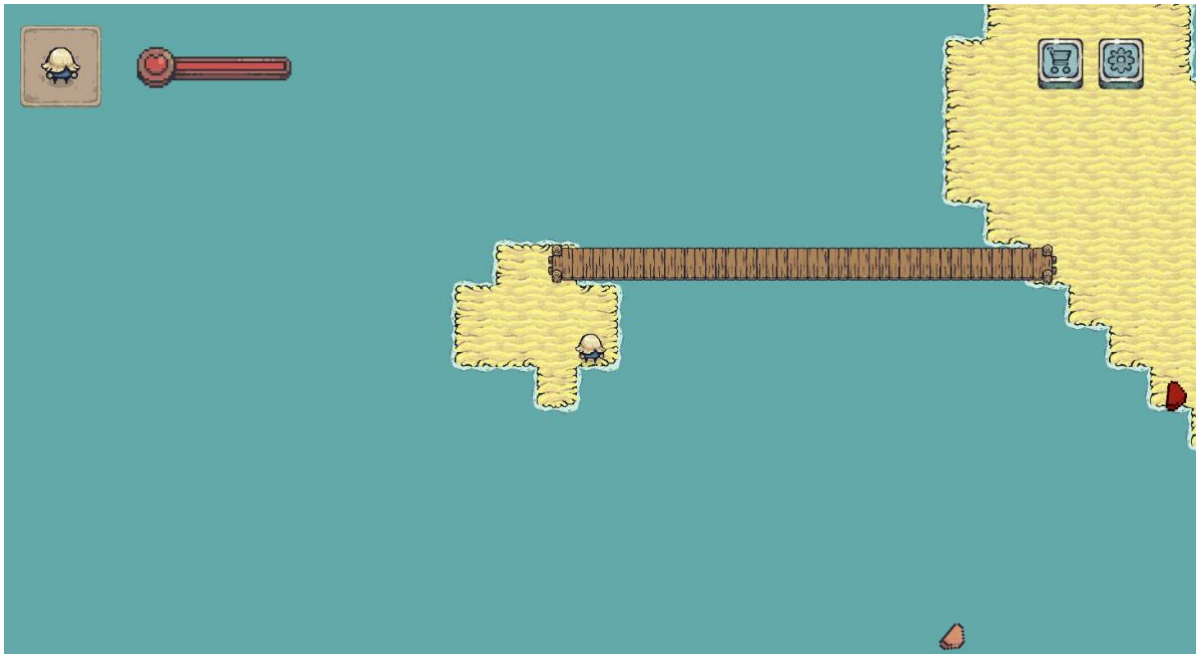


Рисунок 4.2 – Гравець заспавнений на першому рівні

При натисканні клавіш руху персонаж починає рухатись (рис. 4.3).

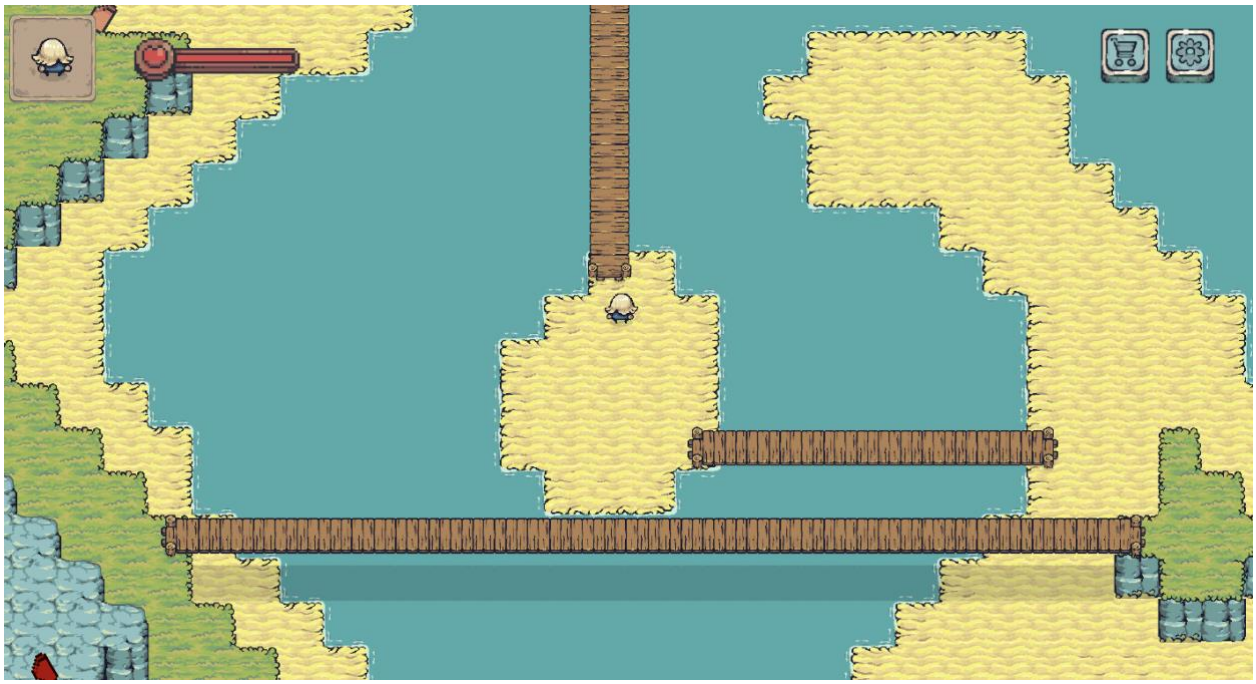


Рисунок 4.3 – Рух гравця

При натисненні кнопки зміни поверху на драбині гравець переміщується на інший поверх (рис. 4.4).

Рисунок 4.4 – Зміна поверху гравця

При натисненні клавіш зміни персонажів персонаж змінюється (рис. 4.5).

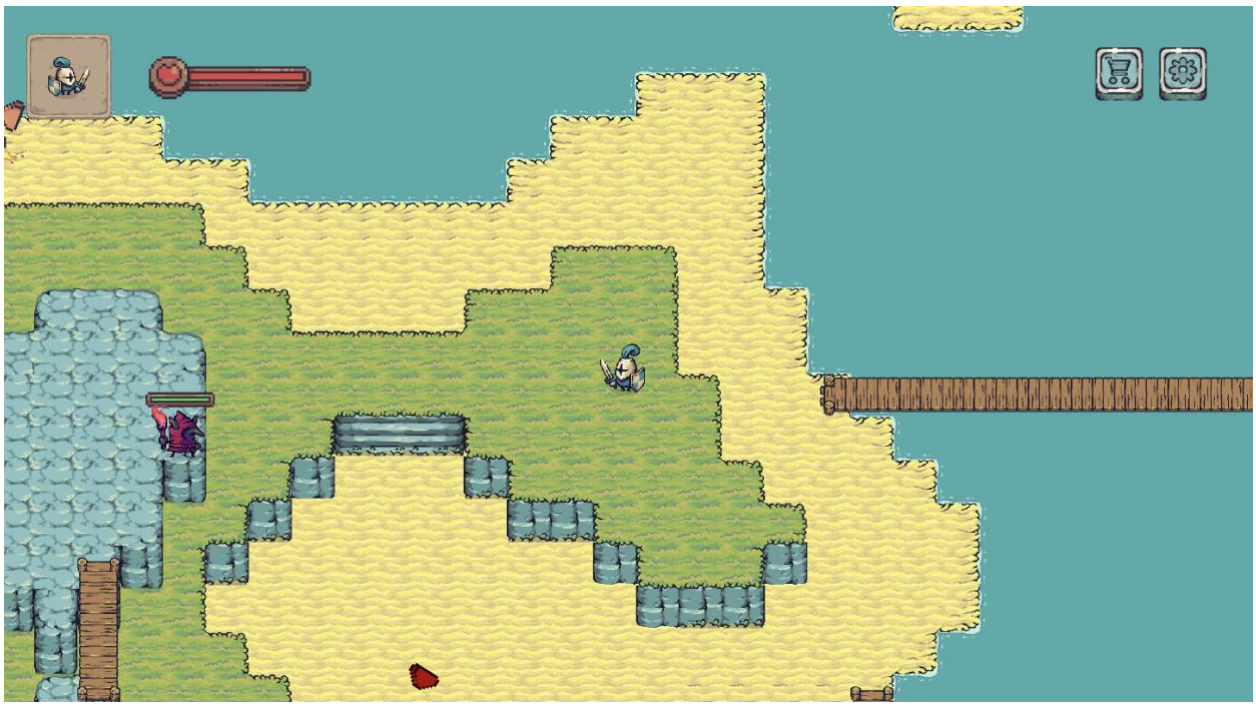


Рисунок 4.5 – Зміна персонажа

Якщо ворог знаходиться в діапазоні атаки то він отримує шкоду (рис. 4.6).

Рисунок 4.6 – Атака на ворога

Коли ворог помічає гравця, то він атакує і персонаж отримуємо шкоду (рис. 4.7).

Рисунок 4.7 – Ворог атакує

Якщо до предмету підійти, то він автоматично перенесеться нам в інвентар (рис. 4.8).

Рисунок 4.8 – Піднімання предмету

При отриманні достатньої кількості шкоди гравець програє і відкривається меню програшу (рис. 4.9).

Рисунок 4.9 – Меню програшу

При нанесенні достатньої кількості шкоди, персонаж знищує ворога (рис. 4.10).

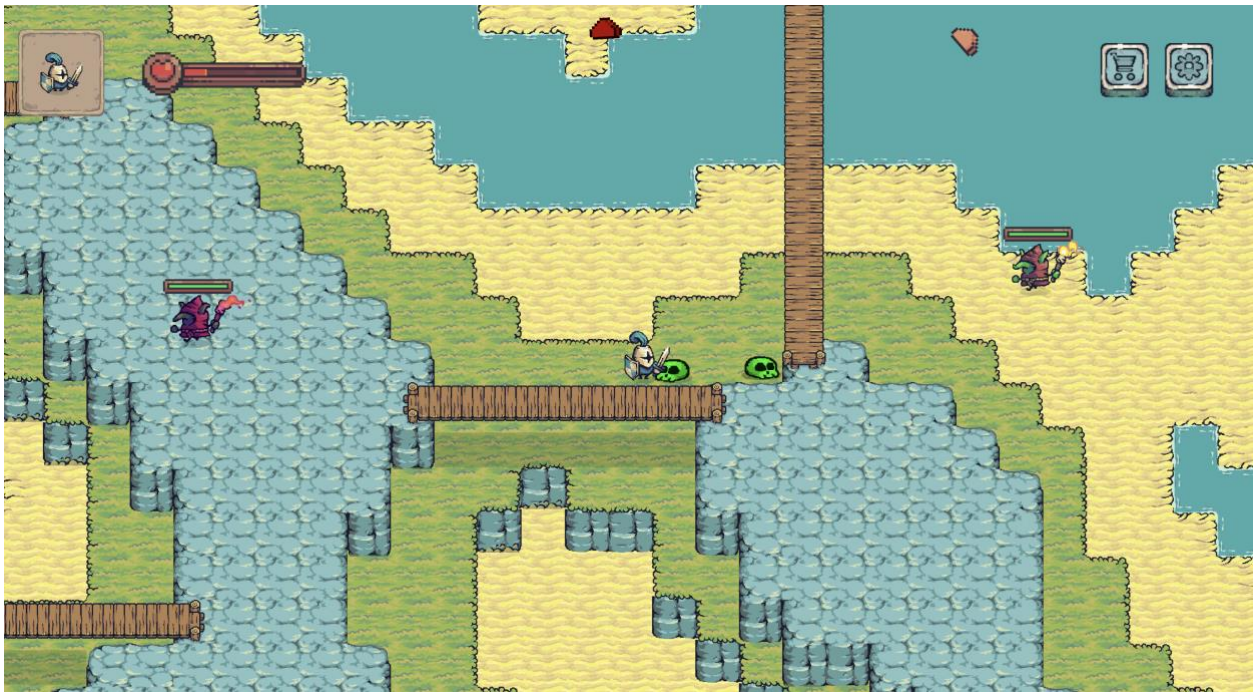


Рисунок 4.10 – Знищення ворога

Після перемоги над фінальним босом гравця переносить в меню перемоги (рис. 4.11).

Рисунок 4.11 – Меню перемоги

Висновки до розділу

У цьому розділі було здійснено оцінку якості розробленого програмного забезпечення та перевірено його відповідність функціональним вимогам, визначеним у другому розділі.

Для аналізу якості було вибрано такі метрики: надійність, підтримуваність, повторюваність коду. Після їхнього детального аналізу можна зробити висновок, що надійність і підтримуваність у проєкту на високому рівні. Щодо повторюваності коду, через специфіку Unity, воно буде на середньому рівні, проте по мірі масштабування даний параметр буде покращуватись.

Додатково було розроблено тест-кейси, за якими проведено ручне тестування ключових функцій. Перевірка засвідчила, що застосунок повністю відповідає функціональним вимогам і стабільно реалізує всі заплановані сценарії.

Завершальним етапом став контрольний приклад використання (end-to-end), який продемонстрував повноцінне проходження ігрового циклу – від запуску рівня до взаємодії з основними ігровими механіками, що дозволяє впевнено стверджувати про готовність ПЗ до подальшої інтеграції та використання.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання розробленого програмного забезпечення було обрано Unity як основне середовище розробки та побудови проєкту. Unity надає вбудовані інструменти для експорту проєктів на різні платформи, включаючи Windows, Android, WebGL та інші. Основною цільовою платформою обрано MacOS (Система розроблялася з урахуванням принципів кросплатформеності, описаних у [13], [22].) у вигляді standalone-застосунку, що дозволяє запускати гру без встановлення додаткових інструментів. Хмарне розгортання надихалося прикладами, опублікованими у спільнотах [14], [20].

Розгортання проєкту передбачає наступні кроки:

Підготовка середовища:

– встановити Unity Hub;

– завантажити та встановити відповідну версію Unity (використану у проєкті);

– переконатися, що встановлено модулі для платформи Windows Build Support.

Завантаження проєкту:

– клонувати або завантажити репозиторій проєкту з GitHub;

– відкрити папку проєкту в Unity Hub.

Відкриття проєкту:

– відкрити проєкт у Unity через Unity Hub;

– дочекатися повного завантаження всіх ресурсів (може зайняти декілька хвилин) (рис. 5.1).

Налаштування параметрів збірки:

– у Unity перейти до меню File → Build Settings (рис. 5.2);

– обрати цільову платформу (наприклад, PC, Mac & Linux Standalone → Windows);

– обрати стартову сцену для збірки;

– натиснути Add Open Scenes, щоб додати її до списку.

Створення збірки:

– натиснути Build;

– вказати директорію для збереження збірки;

– дочекатися завершення процесу компіляції.

Запуск застосунку:

– перейти до директорії, в яку було збережено проєкт;

– запустити гру подвійним кліком (запущений застосунок на рисунку 5.3).

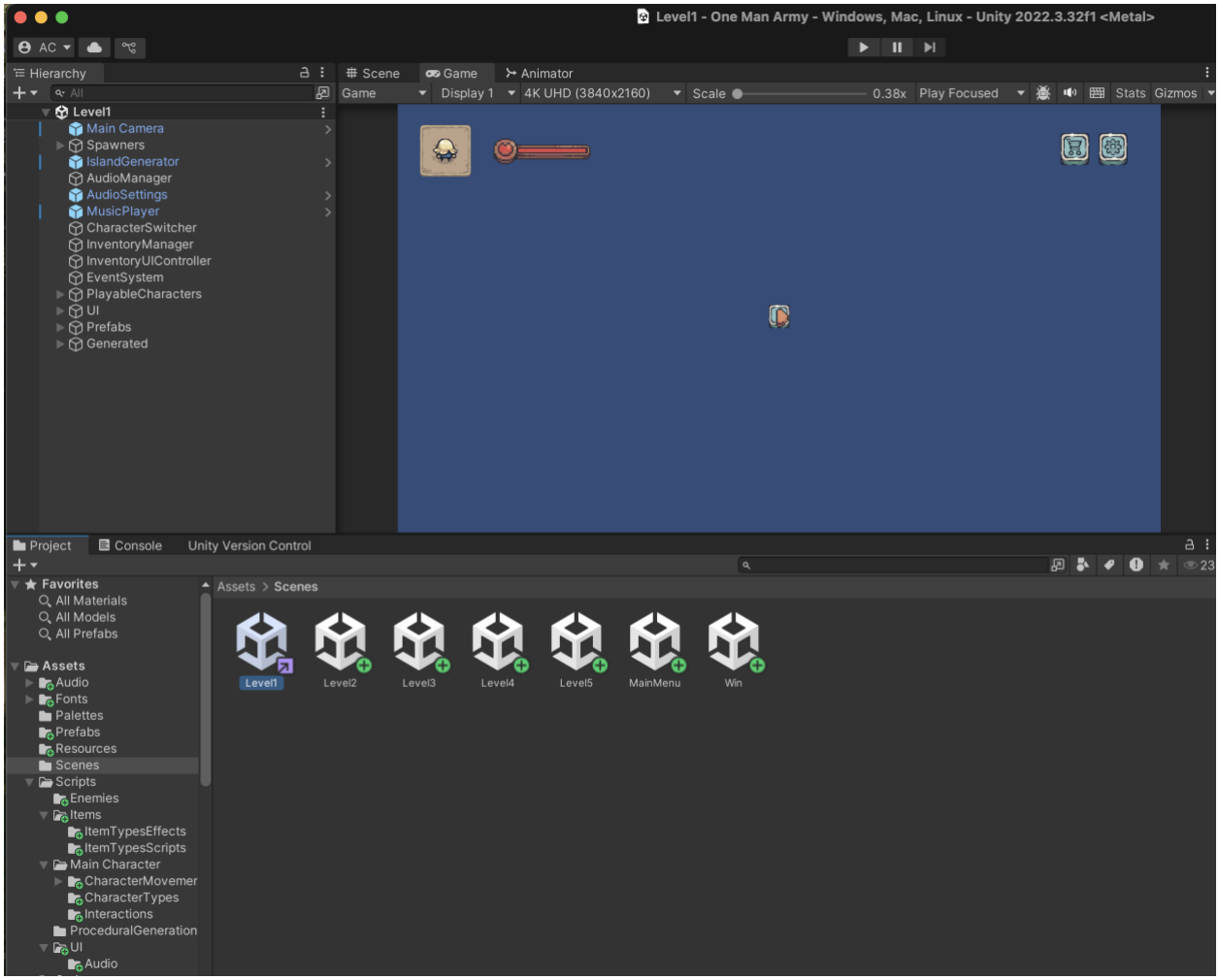


Рисунок 5.1 – Проєкт в Unity

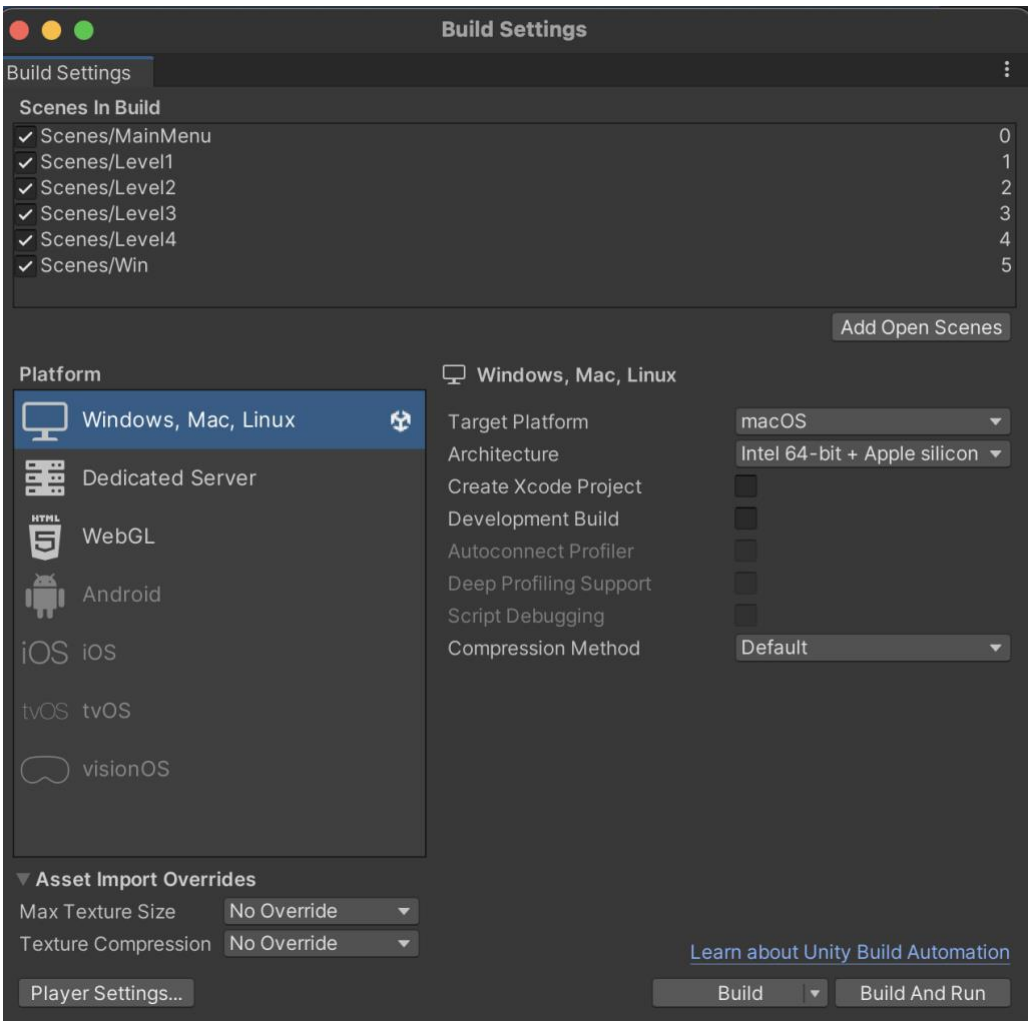


Рисунок 5.2 – Налаштування білду

Рисунок 5.3 – Запущений застосунок

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі «Керівництво користувача».

Для забезпечення довготривалої підтримки та оновлення проєкту One Man Army було передбачено використання системи контролю версій Git у зв’язці з GitHub як основним сховищем коду, а також GitHub Actions як засобу CI/CD для автоматизації процесів розгортання, тестування та аналізу якості коду.

Створення нового випуску гри починається в момент, коли нова стабільна версія проєкту потрапляє у гілку main репозиторію GitHub, з відповідним тегом версії у форматі v*.*.*, де замість * вказуються номера основної, другорядної та патч-версії. Після цього автоматично запускається GitHub Actions.

Висновки до розділу

У межах даного розділу було детально описано процес розгортання розробленого програмного забезпечення, реалізованого на базі ігрового рушія Unity. Основною платформою для розгортання проєкту виступає локальна система розробника та система безперервної інтеграції GitHub Actions.

Збірка гри виконується локально за допомогою Unity Editor, де налаштовуються параметри експорту для цільової платформи (наприклад, Windows або WebGL). Для автоматизації розгортання було використано CI/CD-підхід, зокрема, GitHub Actions, який дозволяє збирати гру при кожному оновленні головної гілки репозиторію (main).

Після успішної збірки виконувані файли автоматично архівуються та прикріплюються до нового релізу на GitHub, що спрощує доставку нової версії гри до кінцевого користувача. Для демонстраційної версії або публічного доступу також можливе розгортання збірки WebGL, яку можна завантажити на хмарну платформу (наприклад, Itch.io або GitHub Pages).

На відміну від типових вебзастосунків, у даному випадку не використовувалась контейнеризація через Docker, оскільки архітектура Unity-проєкту не передбачає серверного середовища чи бази даних. Проте застосування автоматизованої збірки забезпечує високу повторюваність та контроль над якістю випусків.

ВИСНОВКИ

У ході виконання даного дипломного проєкту було спроєктовано та реалізовано ігровий застосунок, розроблений на базі рушія Unity, що поєднує в собі жанрові елементи екшену, rogue-like та процедурної генерації рівнів. Метою розробки було створення функціонального, технічно якісного та масштабованого ігрового продукту з динамічним геймплеєм, системами розвитку персонажа та адаптивною поведінкою ворогів. Також передбачалась можливість подальшого розширення ігрової логіки та оптимізація під різні платформи. Дана мета була досягнута у повному обсязі.

Перед початком технічної реалізації було проаналізовано ключові тенденції та особливості жанру, зокрема механіки бою, штучного інтелекту противників, процедурної генерації та підходів до побудови UI. На основі аналізу було сформовано функціональні та нефункціональні вимоги до гри, виділено ключові компоненти архітектури та взаємозв’язки між ними. Для підвищення керованості та повторного використання логіки було застосовано принципи модульного та компонентного програмування, типові для Unity-екосистеми.

Було реалізовано низку ключових систем:

- механіку перемикання персонажів та індивідуальні характеристики кожного;
- бойову систему з підтримкою ближніх і дальніх атак, критичних ударів та статус-ефектів;
- алгоритм поведінки ворогів з патрулюванням, переслідуванням, відступом та аналізом рівня поверхні;
- систему статусів і шкоди (отруєння, підпал, уповільнення);
- модуль генерації рівнів з різними типами ландшафтів (пісок, трава, гори) та з’єднанням між ними мостами;
- інвентар, підбір предметів та їх вплив на статистику персонажа;
- UI-інтерфейси для здоров’я, інвентарю, вибору персонажа та паузи.

У процесі розробки велика увага приділялася якості коду. Було створено діаграми класів і залежностей, виконано статичний аналіз за допомогою SonarCloud, а також впроваджено автоматизовані CI-процеси за допомогою GitHub Actions. Основними метриками аналізу якості стали надійність, підтримуваність та повторюваність коду. Проєкт показав високий рівень структурованості та стабільності, а дублювання коду – помірне й обґрунтоване архітектурою Unity.

Функціональність застосунку було перевірено за допомогою ручного тестування відповідно до розроблених тест-кейсів. У результаті підтверджено, що основні сценарії гри виконуються коректно, ігрова логіка працює стабільно, а поведінка ворогів відповідає очікуванням.

Також було описано процес локального запуску та налагодження гри, а також інструменти, які можуть бути використані для майбутнього автоматизованого деплою, зокрема Unity Cloud Build або GitHub CI для WebGL-збірок. Передбачено можливості оновлення гри та її підтримки через систему контролю версій Git.

У підсумку можна зробити висновок, що всі поставлені завдання виконано повністю, а мета проєкту досягнута. Ігровий застосунок є стабільним, функціональним, має логічну структуру та добрі перспективи для подальшого розвитку. Серед потенційних напрямів майбутньої роботи можна виділити мультиплеєрну підтримку, розширення сюжетної частини, вдосконалення AI ([7], [19]), локалізацію та публікацію гри на ігрових платформах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Оформлення бібліографії ДСТУ 7.1:2006
2. Оформлення бібліографії ДСТУ 8302-2015 з прикладами
3. Berlin E., Santamaria J. Procedural Content Generation in Games: AI Pro. URL: https://www.gameaipro.com/GameAIPro2/GameAIPro2_Chapter40_Procedural_Content_Generation_An_Overview.pdf (date of access: 25.05.2025).
4. Dungeons of Dredmor Developer Blog. URL: https://www.gaslampgames.com (date of access: 25.05.2025).
5. Unity Technologies. Unity Manual. URL: https://docs.unity3d.com/Manual/ (date of access: 25.05.2025).
6. Game Developers Conference. Indie Game Developer Survey Report 2022. URL: https://gdconf.com/survey2022 (date of access: 25.05.2025).
7. Rabin S. Game AI Pro 3: Collected Wisdom of Game AI Professionals. URL: https://arxiv.org/abs/1904.08972 (date of access: 25.05.2025).
8. Nystrom R. Game Programming Patterns. URL: https://github.com/munificent/game-programming-patterns (date of access: 25.05.2025).
9. Krita Manual. URL: https://docs.krita.org/en/ (date of access: 25.05.2025).
10. Rollings A., Adams E. Fundamentals of Game Design. URL: https://www.academia.edu/81247135/Fundamentals_of_game_design (date of access: 25.05.2025).
11. Jesse Schell. The Art of Game Design: A Book of Lenses. URL: https://www.inventoridigiochi.it/wp-content/uploads/2020/07/art-of-game-design.pdf (date of access: 25.05.2025).
12. Unity Technologies. Script Reference. URL: https://docs.unity3d.com/ScriptReference/ (date of access: 25.05.2025).
13. McShaffry M., Graham S. Game Coding Complete. URL: https://theswissbay.ch/pdf/Gentoomen Library/The Actually Useful Programming

Library/Game Design/Game Coding Complete 3e - McShaffry - Course Tech %282009%29/Game Coding Complete 3e - McShaffry - Course Tech %282009%29.pdf (date of access: 25.05.2025).

14. GameDev.net. Game Development Community. URL: https://www.gamedev.net/ (date of access: 25.05.2025).

15. Gregory J. Game Engine Architecture. URL: https://studio.eecs.umich.edu/confluence/download/attachments/17827062/Game_Engine_Architecture_-_Jason_Gregory.pdf (date of access: 25.05.2025).

16. Unity Learn. URL: https://learn.unity.com/ (date of access: 25.05.2025).

17. Kim A. Roguelike Game Design. URL: https://github.com/Apress/roguelike-development-javascript (date of access: 25.05.2025).

18. Gellel A., Sweetser P. A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels. URL: https://openresearch-repository.anu.edu.au/bitstream/1885/205015/5/FDG20_PCG-submitted.pdf (date of access: 25.05.2025).

19. Millington I., Funge J. Artificial Intelligence for Games (2nd ed.). URL: https://archive.org/details/artificial-intelligence-for-games-2nd-edition (date of access: 25.05.2025).

20. IndieDB – Indie Game Developer Database. URL: https://www.indiedb.com/ (date of access: 25.05.2025).

21. Fullerton T. Game Design Workshop. URL: https://archive.org/details/gamedesignworksh0000full_g4g0 (date of access: 25.05.2025).

22. GitHub. Unity Projects. URL: https://github.com/search?q=unity (date of access: 25.05.2025).

23. Red Blob Games. Procedural Generation. URL: https://www.redblobgames.com/ (date of access: 25.05.2025).

24. ISO/IEC 9126. Software engineering. Product quality. URL: https://people.scs.carleton.ca/~jeanpier/sharedF14/T1/17-%20ISO%209126.pdf (date of access: 25.05.2025).

ДОДАТКИ

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

StrikePlagiarism.com

Дата звіту 6/15/2025
Дата редагування 6/15/2025

Документ прийнятий

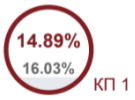
Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-12_Сімчук_ПЗ
Автор Науковий керівник / Експерт
ІП-12_СімчукПавлов 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.



10	9007	68969
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.

Ігровий застосунок у жанрі Roguelike

Текст програми

КПІ.ІП-1228.045480.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр Павлов

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Андрій СІМЧУК

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/simatiz/OneManArmy/

Файл IslandGenerator.cs
Реалізація функціональної задачі процедурної генерації островів та з'єднання цих островів мостами.

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

public class IslandGenerator : MonoBehaviour
{
    public int width = 50;
    public int height = 50;
    public float scale = 10f;
    public float islandRadius = 20f;
    public int minZoneSize = 5; // Мінімальна кількість тайлів, щоб зона
могла бути поєднана мостом
    public int maxBridgeAttempts = 150; // Обмеження на кількість спроб
знайти міст

    public bool spawnGrass = true;
    public bool spawnMountain = true;

    public Tilemap waterTilemap;
    public Tilemap foamTilemap;

    public Tilemap sandTilemap;
    public Tilemap grassTilemap;
    public Tilemap mountainTilemap;

    public Tilemap wallTilemap;
    public Tilemap higherWallTilemap;
    public Tilemap grassLadderTilemap;
    public Tilemap mountainLadderTilemap;

    public Tilemap grassWallShadowTilemap;
    public Tilemap sandWallShadowTilemap;
    public Tilemap bridgeShadowTilemap;

    public Tilemap grassDirtTilemap;
    public Tilemap sandDirtTilemap;

    public Tilemap sandBridgeTilemap;
    public Tilemap grassBridgeTilemap;
    public Tilemap mountainBridgeTilemap;

    public RuleTile waterTile;
    public RuleTile sandTile;
    public RuleTile grassTile;
    public RuleTile mountainTile;
    public RuleTile wallTile;
    public RuleTile bridgeTile;
    public RuleTile ladderTile;

    public TileBase shadowTile;
    public TileBase bridgeShadowTile;
    public TileBase grassDirtTile;
    public TileBase sandDirtTile;
```

```
    public AnimatedTile foamTile;

    public GameObject sandEdgeColliderPrefab;
    public GameObject grassEdgeColliderPrefab;
    public GameObject mountainEdgeColliderPrefab;

    public Transform sandColliderParent;
    public Transform grassColliderParent;
    public Transform mountainColliderParent;

    public Transform sandBridgeColliderParent;
    public Transform grassBridgeColliderParent;
    public Transform mountainBridgeColliderParent;

    public System.Action OnGenerationComplete;

    private float offsetX;
    private float offsetY;

    public bool generationCompleted = false;

    private HashSet<Vector3Int> placedBridges = new HashSet<Vector3Int>(); //
Щоб уникнути дублювання мостів
    private HashSet<Vector3Int> placedWalls = new HashSet<Vector3Int>();
    private List<HashSet<Vector3Int>> sandIslands = new
List<HashSet<Vector3Int>>();

    void Start()
    {
        // Генеруємо випадковий зсув, щоб кожен раз був унікальний острів
        offsetX = Random.Range(0f, 10000f);
        offsetY = Random.Range(0f, 10000f);

        if (!spawnGrass) spawnMountain = false;

        GenerateIsland();
    }

    void GenerateIsland()
    {
        // Отримуємо поточну позицію камери
        Vector3 cameraPosition = Camera.main.transform.position;

        // Обчислюємо центр острова відносно камери
        int centerX = Mathf.RoundToInt(cameraPosition.x);
        int centerY = Mathf.RoundToInt(cameraPosition.y);

        // Центруємо острів відносно камери
        Vector2 center = new Vector2(centerX, centerY);

        for (int x = -width / 2; x < width / 2; x++)
        {
            for (int y = -height / 2; y < height / 2; y++)
            {
                Vector3Int tilePosition = new Vector3Int(centerX + x, centerY
+ y, 0);
                Vector3Int aboveTilePosition = new Vector3Int(centerX + x,
centerY + y + 1, 0);

                float distance = Vector2.Distance(new Vector2(tilePosition.x,
tilePosition.y), center) / islandRadius;
                float noiseValue = Mathf.PerlinNoise((tilePosition.x +
offsetX) / scale, (tilePosition.y + offsetY) / scale) - distance;
```

```
                // Очищаємо старі тайли перед встановленням нових
                waterTilemap.SetTile(tilePosition, null);
                sandTilemap.SetTile(tilePosition, null);
                grassTilemap.SetTile(tilePosition, null);
                mountainTilemap.SetTile(tilePosition, null);
                wallTilemap.SetTile(tilePosition, null);
                higherWallTilemap.SetTile(tilePosition, null);

                // Встановлення тайлів на відповідні рівні
                if (noiseValue <= 0.002f)
                {
                    waterTilemap.SetTile(tilePosition, waterTile);
                }
                else if (noiseValue <= 0.2f)
                {
                    waterTilemap.SetTile(tilePosition, waterTile);
                    sandTilemap.SetTile(tilePosition, sandTile);
                    foamTilemap.SetTile(tilePosition, foamTile);
                }
                else if (noiseValue <= 0.4f)
                {
                    waterTilemap.SetTile(tilePosition, waterTile);
                    sandTilemap.SetTile(tilePosition, sandTile);
                    if (spawnGrass)
                        grassTilemap.SetTile(tilePosition, grassTile);
                }
                else
                {
                    waterTilemap.SetTile(tilePosition, waterTile);
                    sandTilemap.SetTile(tilePosition, sandTile);
                    if (spawnGrass)
                        grassTilemap.SetTile(tilePosition, grassTile);
                    if (spawnMountain)
                        mountainTilemap.SetTile(tilePosition, mountainTile);
                }
            }
        }

        GenerateBridges();
        GenerateWalls();

        PlaceEdgeColliders(sandTilemap, sandBridgeTilemap,
sandEdgeColliderPrefab, sandColliderParent);
        PlaceEdgeColliders(grassTilemap, grassBridgeTilemap,
grassEdgeColliderPrefab, grassColliderParent);
        PlaceEdgeColliders(mountainTilemap, mountainBridgeTilemap,
mountainEdgeColliderPrefab, mountainColliderParent);

        PlaceSideBridgeColliders(sandTilemap, sandBridgeTilemap,
sandBridgeColliderParent, sandEdgeColliderPrefab);
        PlaceSideBridgeColliders(grassTilemap, grassBridgeTilemap,
grassBridgeColliderParent, grassEdgeColliderPrefab);
        PlaceSideBridgeColliders(mountainTilemap, mountainBridgeTilemap,
mountainBridgeColliderParent, mountainEdgeColliderPrefab);

        generationCompleted = true;
        OnGenerationComplete?.Invoke();
    }

    void GenerateWalls()
    {
        for (int x = -width / 2; x < width / 2; x++)
        {
            for (int y = -height / 2; y < height / 2; y++)
```

```
            {
                Vector3Int tilePosition = new Vector3Int(
                    Mathf.RoundToInt(Camera.main.transform.position.x) + x,
                    Mathf.RoundToInt(Camera.main.transform.position.y) + y,
                    0);

                Vector3Int belowTilePosition = new Vector3Int(tilePosition.x,
tilePosition.y - 1, 0);
                Vector3Int secondBelowTilePosition = new
Vector3Int(tilePosition.x, tilePosition.y - 2, 0);
                Vector3Int leftTilePosition = new Vector3Int(tilePosition.x -
1, tilePosition.y, 0);
                Vector3Int rightTilePosition = new Vector3Int(tilePosition.x
+ 1, tilePosition.y, 0);

                bool isGrassLeft = grassTilemap.GetTile(leftTilePosition) !=
null;
                bool isGrassRight = grassTilemap.GetTile(rightTilePosition)
!= null;

                bool isSandBelow =
sandTilemap.GetTile(secondBelowTilePosition) != null;
                bool isGrassBelow =
grassTilemap.GetTile(secondBelowTilePosition) != null;

                // Якщо в цій позиції є гора (mountainTile), ставимо звичайну
стіну
                if (mountainTilemap.GetTile(tilePosition) != null &&
mountainTilemap.GetTile(belowTilePosition) == null)
                {
                    wallTilemap.SetTile(belowTilePosition, wallTile);
                    placedWalls.Add(belowTilePosition);

                    grassWallShadowTilemap.SetTile(belowTilePosition,
shadowTile);
                    if (isSandBelow || isGrassBelow)
                    {
                        if (Random.value < 0.6f)
                        {
                            grassDirtTilemap.SetTile(belowTilePosition,
grassDirtTile);
                        }
                    }
                }
                // Якщо в цій позиції є земля (grassTile), ставимо зміщену
стіну
                else if (grassTilemap.GetTile(tilePosition) != null &&
grassTilemap.GetTile(belowTilePosition) == null)
                {
                    higherWallTilemap.SetTile(belowTilePosition, wallTile);
                    placedWalls.Add(belowTilePosition);

                    // Якщо зліва є трава, додаємо стіну ліворуч
                    if (isGrassLeft)
                    {
                        Vector3Int leftWallPosition = new
Vector3Int(belowTilePosition.x - 1, belowTilePosition.y, 0);
                        higherWallTilemap.SetTile(leftWallPosition,
wallTile);
                    }

                    // Якщо справа є трава, додаємо стіну праворуч
                    if (isGrassRight)
                    {
```

```
                        Vector3Int rightWallPosition = new
Vector3Int(belowTilePosition.x + 1, belowTilePosition.y, 0);
                        higherWallTilemap.SetTile(rightWallPosition,
wallTile);
                    }

                    sandWallShadowTilemap.SetTile(belowTilePosition,
shadowTile);

                    if (isSandBelow || isGrassBelow)
                    {
                        if (Random.value < 0.6f)
                        {
                            sandDirtTilemap.SetTile(belowTilePosition,
sandDirtTile);
                        }
                    }
                    else
                    {
                        sandTilemap.SetTile(belowTilePosition, null);
                    }
                }
            }
        }

        foreach (Vector3Int pos in placedWalls)
        {
            Vector3Int belowTilePosition = new Vector3Int(pos.x, pos.y - 1,
0);
            Vector3Int secondBelowTilePosition = new Vector3Int(pos.x, pos.y
- 2, 0);
            Vector3Int aboveTilePosition = new Vector3Int(pos.x, pos.y + 1,
0);

            bool leftSupport = wallTilemap.GetTile(new Vector3Int(pos.x - 1,
pos.y, 0)) != null ||
                higherWallTilemap.GetTile(new Vector3Int(pos.x - 1, pos.y,
0)) != null ||
                grassLadderTilemap.GetTile(new Vector3Int(pos.x - 1, pos.y,
0)) != null ||
                mountainLadderTilemap.GetTile(new Vector3Int(pos.x - 1,
pos.y, 0)) != null;

            bool rightSupport = wallTilemap.GetTile(new Vector3Int(pos.x + 1,
pos.y, 0)) != null ||
                higherWallTilemap.GetTile(new Vector3Int(pos.x + 1, pos.y,
0)) != null ||
                grassLadderTilemap.GetTile(new Vector3Int(pos.x + 1, pos.y,
0)) != null ||
                mountainLadderTilemap.GetTile(new Vector3Int(pos.x + 1,
pos.y, 0)) != null;

            bool isCorrectPlaceLadder =
                sandTilemap.GetTile(secondBelowTilePosition) != null &&
                sandTilemap.GetTile(belowTilePosition) != null &&
                higherWallTilemap.GetTile(belowTilePosition) == null &&
                higherWallTilemap.GetTile(secondBelowTilePosition) == null &&
                grassLadderTilemap.GetTile(belowTilePosition) == null &&
                grassLadderTilemap.GetTile(secondBelowTilePosition) == null
&&
                sandBridgeTilemap.GetTile(aboveTilePosition) == null &&
                sandBridgeTilemap.GetTile(pos) == null &&
                grassBridgeTilemap.GetTile(aboveTilePosition) == null &&
                grassBridgeTilemap.GetTile(pos) == null &&
```

```
                mountainBridgeTilemap.GetTile(aboveTilePosition) == null &&
                mountainBridgeTilemap.GetTile(pos) == null &&
                leftSupport && rightSupport;

            if (Random.value < 0.5 && isCorrectPlaceLadder)
            {
                if (wallTilemap.HasTile(pos))
                {
                    wallTilemap.SetTile(pos, null);
                    mountainLadderTilemap.SetTile(pos, ladderTile);
                }
                else
                {
                    higherWallTilemap.SetTile(pos, null);
                    grassLadderTilemap.SetTile(pos, ladderTile);
                }
            }
        }

        List<Vector3Int> toRelocate = new List<Vector3Int>();

        for (int i = 1; i <= 3; i++) {
            toRelocate = new List<Vector3Int>();

            // Знаходимо всі стіни без опори знизу
            foreach (Vector3Int pos in placedWalls)
            {
                Vector3Int below = pos + Vector3Int.down;

                Vector3Int down = pos + Vector3Int.down;
                Vector3Int downLeft = pos + new Vector3Int(-1, -1, 0);
                Vector3Int downRight = pos + new Vector3Int(1, -1, 0);

                bool hasSupportBelow =
                    HasGroundTile(down) &&
                    HasGroundTile(downLeft) &&
                    HasGroundTile(downRight);

                if (!hasSupportBelow)
                {
                    toRelocate.Add(pos);
                }
            }

            // Переносимо стіни нижче, якщо це дозволено
            foreach (var wallPos in toRelocate)
            {
                Vector3Int above = wallPos + Vector3Int.up;
                Vector3Int newWallPos = wallPos + Vector3Int.down;

                // Пропускаємо, якщо над стіною – пісок (його чіпати не
можна)
                if (sandTilemap.HasTile(above))
                    continue;

                // Перевіряємо, що зверху трава або гора
                bool isGrass = grassTilemap.HasTile(above);
                bool isMountain = mountainTilemap.HasTile(above);

                if (!isGrass && !isMountain)
                    continue;

                // Видаляємо стару стіну
```

```
                if (wallTilemap.HasTile(wallPos))
wallTilemap.SetTile(wallPos, null);
                if (higherWallTilemap.HasTile(wallPos))
higherWallTilemap.SetTile(wallPos, null);

                // Видаляємо блок над нею
                if (isGrass) grassTilemap.SetTile(above, null);
                if (isMountain) mountainTilemap.SetTile(above, null);

                // Ставимо нову стіну нижче
                if (!wallTilemap.HasTile(newWallPos) &&
!higherWallTilemap.HasTile(newWallPos))
                {
                    wallTilemap.SetTile(newWallPos, wallTile);
                    placedWalls.Add(newWallPos);
                }
            }
        }

        EnsureSandUnderGrassWalls();

        // Зміщуємо Tilemap нижчих стін вверх
        higherWallTilemap.transform.position += new Vector3(0, +0.05f, 0);
        grassLadderTilemap.transform.position += new Vector3(0, +0.05f, 0);
        sandDirtTilemap.transform.position += new Vector3(0, +0.05f, 0);
        grassWallShadowTilemap.transform.position += new Vector3(0, -0.02f,
0);
        sandWallShadowTilemap.transform.position += new Vector3(0, -0.02f,
0);
    }

    bool HasGroundTile(Vector3Int position)
    {
        // Якщо поза межами – не опора
        if (!waterTilemap.cellBounds.Contains(position)) return false;

        if (waterTilemap.HasTile(position) ||
higherWallTilemap.HasTile(position) || grassLadderTilemap.HasTile(position))
return false;

        return sandTilemap.HasTile(position) ||
               grassTilemap.HasTile(position) ||
               mountainTilemap.HasTile(position);
    }

    void EnsureSandUnderGrassWalls()
    {
        foreach (var pos in placedWalls)
        {
            Vector3Int above = pos + Vector3Int.up;

            if (grassTilemap.HasTile(above))
            {
                Vector3Int below1 = pos + Vector3Int.down;
                Vector3Int below2 = below1 + Vector3Int.down;
                Vector3Int downLeft = pos + new Vector3Int(-1, -1, 0);
                Vector3Int downRight = pos + new Vector3Int(1, -1, 0);

                PlaceSand(pos);
                PlaceSand(below1);
                PlaceSand(below2);
                PlaceSand(downLeft);
                PlaceSand(downRight);
            }
```

```
        }
    }

    void PlaceSand(Vector3Int pos)
    {
        if (!sandTilemap.HasTile(pos))
        {
            sandTilemap.SetTile(pos, sandTile);
            foamTilemap.SetTile(pos, foamTile);
        }
    }

    void GenerateBridges()
    {
        ConnectZones(grassTilemap);
        sandIslands = ConnectZones(sandTilemap);
        ConnectZones(mountainTilemap);
    }

    List<HashSet<Vector3Int>> ConnectZones(Tilemap terrainTilemap)
    {
        List<HashSet<Vector3Int>> regions =
FindDisconnectedRegions(terrainTilemap);

        List<HashSet<Vector3Int>> islands = new List<HashSet<Vector3Int>>();
        foreach (var hashSet in regions)
        {
            islands.Add(new HashSet<Vector3Int>(hashSet));
        }

        // Видаляємо маленькі зони, які не потрібно поєднувати
        regions.RemoveAll(region => region.Count < minZoneSize);
        if (regions.Count <= 1) return new List<HashSet<Vector3Int>>(); //
Всі зони вже пов'язані

        int attempts = 0;

        while (regions.Count > 1 && attempts < maxBridgeAttempts)
        {
            attempts++;

            HashSet<Vector3Int> regionA = regions[0];
            HashSet<Vector3Int> closestRegion = null;
            Vector3Int bestStart = Vector3Int.zero, bestEnd =
Vector3Int.zero;
            int bestDistance = int.MaxValue;

            foreach (var regionB in regions)
            {
                if (regionA == regionB) continue;

                foreach (var posA in regionA)
                {
                    foreach (var posB in regionB)
                    {
                        int distanceX = Mathf.Abs(posA.x - posB.x);
                        int distanceY = Mathf.Abs(posA.y - posB.y);

                        // Міст тільки горизонтальний або вертикальний
                        if (distanceX > 0 && distanceY > 0) continue;

                        int distance = distanceX + distanceY;
```

```
                        if (distance < bestDistance && CanPlaceBridge(posA,
posB))
                        {
                            bestDistance = distance;
                            bestStart = posA;
                            bestEnd = posB;
                            closestRegion = regionB;
                        }
                    }
                }
            }

            if (closestRegion != null)
            {
                PlaceBridge(bestStart, bestEnd);
                regionA.UnionWith(closestRegion);
                regions.Remove(closestRegion);
            }
            else
            {
                // Якщо після кількох спроб не знайшлося хорошого мосту –
виходимо
                break;
            }
        }
        return islands;
    }

    List<HashSet<Vector3Int>> FindDisconnectedRegions(Tilemap terrainTilemap)
    {
        HashSet<Vector3Int> visited = new HashSet<Vector3Int>();
        List<HashSet<Vector3Int>> regions = new List<HashSet<Vector3Int>>();

        for (int x = -width / 2; x < width / 2; x++)
        {
            for (int y = -height / 2; y < height / 2; y++)
            {
                Vector3Int pos = new Vector3Int(
                    Mathf.RoundToInt(Camera.main.transform.position.x) + x,
                    Mathf.RoundToInt(Camera.main.transform.position.y) + y,
                    0
                );

                if (terrainTilemap.GetTile(pos) != null &&
!visited.Contains(pos))
                {
                    HashSet<Vector3Int> region = new HashSet<Vector3Int>();
                    Queue<Vector3Int> queue = new Queue<Vector3Int>();
                    queue.Enqueue(pos);
                    visited.Add(pos);

                    while (queue.Count > 0)
                    {
                        Vector3Int current = queue.Dequeue();
                        region.Add(current);

                        foreach (Vector3Int neighbor in
GetNeighbors(current))
                        {
                            if (!visited.Contains(neighbor) &&
terrainTilemap.GetTile(neighbor) != null)
                            {
                                queue.Enqueue(neighbor);
                                visited.Add(neighbor);
```

```
                        }
                    }
                }

                regions.Add(region);
            }
        }
    }

    return regions;
}

bool CanPlaceBridge(Vector3Int start, Vector3Int end)
{
    if (placedBridges.Contains(start) || placedBridges.Contains(end))
return false;

    Vector3Int direction = new Vector3Int(
        Mathf.Clamp(end.x - start.x, -1, 1),
        Mathf.Clamp(end.y - start.y, -1, 1),
        0
    );

    Vector3Int pos = start;

    // Перевіряємо, чи поруч із початком або кінцем мосту є інший міст
    if (IsBridgeNearStartOrEnd(start) || IsBridgeNearStartOrEnd(end))
return false;

    while (pos != end)
    {
        pos += direction;
        // Перевірка чи вже є міст у цьому місці
        if (sandBridgeTilemap.GetTile(pos) != null) return false;
        if (grassBridgeTilemap.GetTile(pos) != null) return false;
        if (mountainBridgeTilemap.GetTile(pos) != null) return false;
        if (IsBridgeTooClose(pos, direction)) return false;
        if (waterTilemap.GetTile(pos) == null) return false;
    }

    return true;
}

bool IsBridgeTooClose(Vector3Int pos, Vector3Int direction)
{
    // Перевіряємо сусідні тайли в напрямку, перпендикулярному мосту
    Vector3Int checkLeft = pos + new Vector3Int(-direction.y,
direction.x, 0);
    Vector3Int checkRight = pos + new Vector3Int(direction.y, -
direction.x, 0);

    return (sandBridgeTilemap.GetTile(checkLeft) != null ||
sandBridgeTilemap.GetTile(checkRight) != null ||
        grassBridgeTilemap.GetTile(checkLeft) != null ||
grassBridgeTilemap.GetTile(checkRight) != null ||
        mountainBridgeTilemap.GetTile(checkLeft) != null ||
mountainBridgeTilemap.GetTile(checkRight) != null);
}

bool IsBridgeNearStartOrEnd(Vector3Int pos)
{
    List<Vector3Int> neighbors = new List<Vector3Int>
{
    pos + Vector3Int.up,
```

```
            pos + Vector3Int.down,
            pos + Vector3Int.left,
            pos + Vector3Int.right
        };

            foreach (var neighbor in neighbors)
            {
                if (sandBridgeTilemap.GetTile(neighbor) != null ||
grassBridgeTilemap.GetTile(neighbor) != null ||
mountainBridgeTilemap.GetTile(neighbor) != null)
                {
                    return true; // Є міст поруч із початком або кінцем
                }
            }

            return false;
        }

        void PlaceBridge(Vector3Int start, Vector3Int end)
        {
            Vector3Int direction = new Vector3Int(
                Mathf.Clamp(end.x - start.x, -1, 1),
                Mathf.Clamp(end.y - start.y, -1, 1), 0);

            Vector3Int pos = start;

            bool connectsHigh = IsGrassTile(start) && IsGrassTile(end);
            bool connectsHighest = IsMountainTile(start) && IsMountainTile(end);

            if (!placedBridges.Contains(pos))
            {
                Tilemap targetBridgeMap = GetBridgeTilemapForLevel(start, end);
                if (targetBridgeMap != null)
                {
                    targetBridgeMap.SetTile(pos, bridgeTile);
                    placedBridges.Add(pos);
                }
            }

            while (pos != end)
            {
                pos += direction;
                if (!placedBridges.Contains(pos))
                {
                    Tilemap targetBridgeMap = GetBridgeTilemapForLevel(start,
end);
                    if (targetBridgeMap != null)
                    {
                        targetBridgeMap.SetTile(pos, bridgeTile);
                        placedBridges.Add(pos);
                    }
                    if (connectsHigh && direction.y == 0 && pos != start && pos
!= end)
                        AddBridgeShadow(pos);
                }
            }

            Tilemap GetBridgeTilemapForLevel(Vector3Int start, Vector3Int end)
            {
                bool isSand = sandTilemap.GetTile(start) != null &&
sandTilemap.GetTile(end) != null;
                bool isGrass = grassTilemap.GetTile(start) != null &&
grassTilemap.GetTile(end) != null;
```

```
            bool isMountain = mountainTilemap.GetTile(start) != null &&
mountainTilemap.GetTile(end) != null;

            if (isMountain) return mountainBridgeTilemap;
            if (isGrass) return grassBridgeTilemap;
            if (isSand) return sandBridgeTilemap;

            return null;
        }

        void AddBridgeShadow(Vector3Int bridgePos)
        {
            Vector3Int shadowPos = bridgePos + Vector3Int.down; // Тінь
завжди нижче моста

            // Якщо тінь ще не існує, додаємо її
            if (bridgeShadowTilemap.GetTile(shadowPos) == null)
            {
                if (connectsHigh && !connectsHighest)
                {
                    if (!IsGrassTile(shadowPos) &&
!IsMountainTile(shadowPos))
                    {
                        bridgeShadowTilemap.SetTile(shadowPos,
bridgeShadowTile);
                    }
                }
                else
                {
                    if (!IsMountainTile(shadowPos))
                    {
                        if (IsGrassTile(shadowPos))
                            bridgeShadowTilemap.SetTile(shadowPos,
bridgeShadowTile);
                        else
                            if (!IsGrassTile(shadowPos + Vector3Int.down))
                            bridgeShadowTilemap.SetTile(shadowPos +
Vector3Int.down, bridgeShadowTile);
                    }
                }
            }
        }
    }

    void PlaceEdgeColliders(Tilemap tilemap, Tilemap bridgeTilemap,
GameObject colliderPrefab, Transform parent)
    {
        HashSet<Vector3Int> placed = new HashSet<Vector3Int>(); // Щоб
уникнути дублювання

        for (int x = -width / 2; x < width / 2; x++)
        {
            for (int y = -height / 2; y < height / 2; y++)
            {
                Vector3Int pos = new Vector3Int(
                    Mathf.RoundToInt(Camera.main.transform.position.x) + x,
                    Mathf.RoundToInt(Camera.main.transform.position.y) + y,
                    0);

                if (tilemap.GetTile(pos) != null)
                {
                    foreach (var neighbor in GetNeighbors(pos))
                    {
                        if (
```

```
                            tilemap.GetTile(neighbor) == null &&
                            grassLadderTilemap.GetTile(neighbor) == null &&
                            mountainLadderTilemap.GetTile(neighbor) == null
&&
                            !placed.Contains(neighbor))
                        {
                            Vector3 worldPos = tilemap.CellToWorld(neighbor)
+ new Vector3(0.3f, 0.3f, 0);
                            Instantiate(colliderPrefab, worldPos,
Quaternion.identity, parent);
                            placed.Add(neighbor);
                        }
                    }
                }
            }
        }
    }

    void PlaceSideBridgeColliders(Tilemap groundTilemap, Tilemap bridgeMap,
Transform colliderParent, GameObject colliderPrefab)
    {
        HashSet<Vector3Int> placed = new HashSet<Vector3Int>();

        foreach (Vector3Int pos in bridgeMap.cellBounds.allPositionsWithin)
        {
            if (!bridgeMap.HasTile(pos)) continue;

            bool hasLeft = bridgeMap.HasTile(pos + Vector3Int.left);
            bool hasRight = bridgeMap.HasTile(pos + Vector3Int.right);
            bool hasUp = bridgeMap.HasTile(pos + Vector3Int.up);
            bool hasDown = bridgeMap.HasTile(pos + Vector3Int.down);

            // Якщо горизонтальний міст
            if (hasLeft && hasRight && !(hasUp || hasDown))
            {
                Vector3Int up = pos + Vector3Int.up;
                Vector3Int down = pos + Vector3Int.down;

                if (!placed.Contains(up))
                {
                    Vector3 worldUp = bridgeMap.CellToWorld(up) + new
Vector3(0.3f, 0.3f, 0);
                    Instantiate(colliderPrefab, worldUp, Quaternion.identity,
colliderParent);
                    placed.Add(up);
                }

                if (!placed.Contains(down))
                {
                    Vector3 worldDown = bridgeMap.CellToWorld(down) + new
Vector3(0.3f, 0.3f, 0);
                    Instantiate(colliderPrefab, worldDown,
Quaternion.identity, colliderParent);
                    placed.Add(down);
                }
            }

            if ((hasLeft || hasRight) && !(hasUp || hasDown))
            {
                Vector3Int up = pos + Vector3Int.up;
                Vector3Int down = pos + Vector3Int.down;

                if (!placed.Contains(up) && !groundTilemap.HasTile(up))
                {
```

```
                    Vector3 worldUp = bridgeMap.CellToWorld(up) + new
Vector3(0.3f, 0.3f, 0);
                    Instantiate(colliderPrefab, worldUp, Quaternion.identity,
colliderParent);
                    placed.Add(up);
                }

                if (!placed.Contains(down) && !groundTilemap.HasTile(down))
                {
                    Vector3 worldDown = bridgeMap.CellToWorld(down) + new
Vector3(0.3f, 0.3f, 0);
                    Instantiate(colliderPrefab, worldDown,
Quaternion.identity, colliderParent);
                    placed.Add(down);
                }
            }

            // Якщо вертикальний міст
            if (hasUp && hasDown && !(hasLeft || hasRight))
            {
                Vector3Int left = pos + Vector3Int.left;
                Vector3Int right = pos + Vector3Int.right;

                if (!placed.Contains(left))
                {
                    Vector3 worldLeft = bridgeMap.CellToWorld(left) + new
Vector3(0.3f, 0.3f, 0);
                    Instantiate(colliderPrefab, worldLeft,
Quaternion.identity, colliderParent);
                    placed.Add(left);
                }

                if (!placed.Contains(right))
                {
                    Vector3 worldRight = bridgeMap.CellToWorld(right) + new
Vector3(0.3f, 0.3f, 0);
                    Instantiate(colliderPrefab, worldRight,
Quaternion.identity, colliderParent);
                    placed.Add(right);
                }
            }

            if ((hasUp || hasDown) && !(hasLeft || hasRight))
            {
                Vector3Int left = pos + Vector3Int.left;
                Vector3Int right = pos + Vector3Int.right;

                if (!placed.Contains(left) && !groundTilemap.HasTile(left))
                {
                    Vector3 worldLeft = bridgeMap.CellToWorld(left) + new
Vector3(0.3f, 0.3f, 0);
                    Instantiate(colliderPrefab, worldLeft,
Quaternion.identity, colliderParent);
                    placed.Add(left);
                }

                if (!placed.Contains(right) && !groundTilemap.HasTile(right))
                {
                    Vector3 worldRight = bridgeMap.CellToWorld(right) + new
Vector3(0.3f, 0.3f, 0);
                    Instantiate(colliderPrefab, worldRight,
Quaternion.identity, colliderParent);
                    placed.Add(right);
                }
```

```
            }
        }
    }

    bool IsGrassTile(Vector3Int pos)
    {
        return grassTilemap.GetTile(pos) != null;
    }

    bool IsMountainTile(Vector3Int pos)
    {
        return mountainTilemap.GetTile(pos) != null;
    }

    public List<Vector3Int> GetNeighbors(Vector3Int pos)
    {
        return new List<Vector3Int>
        {
            pos + Vector3Int.up,
            pos + Vector3Int.down,
            pos + Vector3Int.left,
            pos + Vector3Int.right
        };
    }

    public List<HashSet<Vector3Int>> getSandIslands()
    {
        return sandIslands;
    }
}
```

Файл PlayerMovement.cs

Реалізація функціональної задачі системи руху гравця та системи зміни поверху (рівня).

```
using UnityEngine;
using UnityEngine.Tilemaps;

public class PlayerMovement : MonoBehaviour
{
    public float ladderSpeed = 2f;

    private Rigidbody2D rb;
    private Animator animator;
    private CharacterStatsBase stats;
    private SpriteRenderer spR;
    private Vector2 movement;
    private bool isFacingRight = true;
    public bool canMove = true;

    public Tilemap sandTilemap, grassTilemap, mountainTilemap;
    public Tilemap sandBridgeTilemap, grassBridgeTilemap,
mountainBridgeTilemap;
    public Tilemap wallTilemap, higherWallTilemap;
    public Tilemap grassLadderTilemap, mountainLadderTilemap;
    public Transform sandColliderParent, grassColliderParent,
mountainColliderParent;
    public Transform sandBridgeColliderParent, grassBridgeColliderParent,
mountainBridgeColliderParent;

    private TilemapCollider2D sandCollider, grassCollider, mountainCollider;
    private TilemapCollider2D sandBridgeCollider, grassBridgeCollider,
mountainBridgeCollider;
```

```
    private enum HeightLevel { Sand, Grass, Mountain }
    private HeightLevel currentLevel = HeightLevel.Sand;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        stats = GetComponent<CharacterStatsBase>();
        spR = GetComponent<SpriteRenderer>();

        sandCollider = sandTilemap.GetComponent<TilemapCollider2D>();
        grassCollider = grassTilemap.GetComponent<TilemapCollider2D>();
        mountainCollider = mountainTilemap.GetComponent<TilemapCollider2D>();

        sandBridgeCollider =
sandBridgeTilemap.GetComponent<TilemapCollider2D>();
        grassBridgeCollider =
grassBridgeTilemap.GetComponent<TilemapCollider2D>();
        mountainBridgeCollider =
mountainBridgeTilemap.GetComponent<TilemapCollider2D>();
    }

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

        if (movement.magnitude > 1)
            movement.Normalize();

        if (movement.x > 0 && !isFacingRight)
            Flip();
        else if (movement.x < 0 && isFacingRight)
            Flip();
    }

    void FixedUpdate()
    {
        if (!canMove)
        {
            rb.velocity = Vector2.zero;
            animator.SetFloat("Speed", 0f);
            return;
        }

        rb.velocity = new Vector2(movement.x * stats.moveSpeed, movement.y *
stats.moveSpeed);
        animator.SetFloat("Speed", rb.velocity.magnitude);
    }

    void Flip()
    {
        isFacingRight = !isFacingRight;
        var scale = transform.localScale;
        scale.x *= -1;
        transform.localScale = scale;
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer ==
LayerMask.NameToLayer("SandBridge"))
        {
```

```
            EnableColliders(sandColliderParent, false);
            EnableColliders(sandBridgeColliderParent, true);
        }
        else if (collision.gameObject.layer ==
LayerMask.NameToLayer("GrassBridge"))
        {
            Debug.Log("Enter Grass Bridge");
            EnableColliders(grassColliderParent, false);
            EnableColliders(grassBridgeColliderParent, true);
        }
        else if (collision.gameObject.layer ==
LayerMask.NameToLayer("MountainBridge"))
        {
            EnableColliders(mountainColliderParent, false);
            EnableColliders(mountainBridgeColliderParent, true);
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.gameObject.layer ==
LayerMask.NameToLayer("SandBridge"))
        {
            EnableColliders(sandColliderParent, true);
            EnableColliders(sandBridgeColliderParent, false);
        }
        else if (collision.gameObject.layer ==
LayerMask.NameToLayer("GrassBridge"))
        {
            EnableColliders(grassColliderParent, true);
            EnableColliders(grassBridgeColliderParent, false);
        }
        else if (collision.gameObject.layer ==
LayerMask.NameToLayer("MountainBridge"))
        {
            EnableColliders(mountainColliderParent, true);
            EnableColliders(mountainBridgeColliderParent, false);
        }
    }

    private void SetHeightLevel(HeightLevel level)
    {
        if (currentLevel != level)
        {
            currentLevel = level;
            UpdateColliders();
            Debug.Log("LEVEL → " + currentLevel);
        }
    }

    public EdgeBlocker.Level GetCurrentLevel()
    {
        Debug.Log(currentLevel);
        switch (currentLevel)
        {
            case HeightLevel.Sand: return EdgeBlocker.Level.Sand;
            case HeightLevel.Grass: return EdgeBlocker.Level.Grass;
            case HeightLevel.Mountain: return EdgeBlocker.Level.Mountain;
            default: return EdgeBlocker.Level.Sand;
        }
    }

    void UpdateColliders()
    {
```

```
        EnableColliders(sandColliderParent, currentLevel ==
HeightLevel.Sand);
        EnableColliders(sandBridgeColliderParent, false);
        EnableColliders(grassColliderParent, currentLevel ==
HeightLevel.Grass);
        EnableColliders(grassBridgeColliderParent, false);
        EnableColliders(mountainColliderParent, currentLevel ==
HeightLevel.Mountain);
        EnableColliders(mountainBridgeColliderParent, false);

        UpdateWallVisibility();
        UpdateLevels();
        UpdatePlayerLayer();
    }

    void UpdateWallVisibility()
    {
        if (wallTilemap != null)
        {
            bool enableMountainWalls = currentLevel == HeightLevel.Grass;
            wallTilemap.GetComponent<TilemapCollider2D>().enabled =
enableMountainWalls;
        }

        if (higherWallTilemap != null)
        {
            bool enableGrassWalls = currentLevel == HeightLevel.Sand;
            higherWallTilemap.GetComponent<TilemapCollider2D>().enabled =
enableGrassWalls;
        }
    }

    void UpdateLevels()
    {
        if (currentLevel == HeightLevel.Sand)
        {
            grassCollider.enabled = true;
            mountainCollider.enabled = true;
            grassBridgeCollider.isTrigger = false;
            grassBridgeCollider.enabled = false;
            mountainBridgeCollider.isTrigger = false;
            mountainBridgeCollider.enabled = false;
        }

        else if(currentLevel == HeightLevel.Grass)
        {
            grassCollider.enabled = false;
            mountainCollider.enabled = true;
            grassBridgeCollider.isTrigger = true;
            grassBridgeCollider.enabled = true;
            mountainBridgeCollider.isTrigger = false;
            mountainBridgeCollider.enabled = false;
        }
        else if (currentLevel == HeightLevel.Mountain)
        {
            grassCollider.enabled = false;
            mountainCollider.enabled = false;
            grassBridgeCollider.isTrigger = false;
            grassBridgeCollider.enabled = false;
            mountainBridgeCollider.isTrigger = true;
            mountainBridgeCollider.enabled = true;
        }
    }
```

```
    void EnableColliders(Transform parent, bool enable)
    {
        foreach (Transform child in parent)
        {
            Collider2D col = child.GetComponent<Collider2D>();
            if (col != null)
            {
                col.enabled = enable;
            }
        }
    }

    void UpdatePlayerLayer()
    {
        if (currentLevel == HeightLevel.Sand) spR.sortingOrder = 6;
        else if (currentLevel == HeightLevel.Grass) spR.sortingOrder = 8;
        else spR.sortingOrder = 10;
    }

    public void SetHeightLevelToGrass()
    {
        SetHeightLevel(HeightLevel.Grass);
    }

    public void SetHeightLevelToMountain()
    {
        SetHeightLevel(HeightLevel.Mountain);
    }

    public void SetHeightLevelToSand()
    {
        SetHeightLevel(HeightLevel.Sand);
    }

    public void SetNewHeightLevel(EdgeBlocker.Level level)
    {
        HeightLevel newLevel;
        switch (level)
        {
            case EdgeBlocker.Level.Sand: newLevel = HeightLevel.Sand;
                break;
            case EdgeBlocker.Level.Grass: newLevel = HeightLevel.Grass;
                break;
            case EdgeBlocker.Level.Mountain: newLevel = HeightLevel.Mountain;
                break;
            default: newLevel = HeightLevel.Sand;
                break;
        }
        SetHeightLevel(newLevel);
    }
}
```

Файл CameraFollow.cs

Реалізація функціональної задачі слідування камери за гравцем.

```
using UnityEngine;

public class CameraFollow : MonoBehaviour
{
    public float FollowSpeed = 2f;
    public float yOffset = 1f;
    public Transform target;

    void Update()
    {
```

```
        Vector3 newPos = new Vector3(target.position.x, target.position.y +
yOffset, -10f);
        transform.position = Vector3.Slerp(transform.position, newPos,
FollowSpeed * Time.deltaTime);
    }
}
```

Файл CharacterStatsBase.cs

Реалізація функціональних задач атаки(стрільби), системи отримання пошкоджень або ефектів, смерті персонажа.

```
using UnityEngine;
using System.Collections;
using UnityEngine.Audio;

public abstract class CharacterStatsBase : MonoBehaviour
{
    [Header("Level")]
    public static int level = 1;

    [Header("Audio SFX")]
    public AudioClip attackClip;
    public AudioClip hurtClip;
    public AudioClip deathClip;

    AudioSource sfx;
    public AudioMixerGroup sfxGroup;

    [Header("Health")]
    public static float maxHealth = 100f;
    public static float _currentHealth = 100f;
    public static float CurrentHealth
    {
        get => _currentHealth;
        set
        {
            _currentHealth = Mathf.Clamp(value, 0, maxHealth);
        }
    }
    public float damageResistance = 1f;
    public float damageIgnoreChance = 0f;
    public PauseManager UI;
    public Healthbar healthbar;
    private float deathDelay = 2f;
    private Animator animator;
    private static bool healthInitialized = false;

    [Header("Combat")]
    public float baseDamage;
    public float attackRate;
    public float attackRange;
    public float critChance;
    public float critMultiplier;
    public bool isRanged;
    public float arrowSpeedMultiplier = 1f;

    [Header("Movement")]
    public float moveSpeed;
    private float originalSpeed;
    private float bonusSpeedTimer;
    private float bonusSpeedAmount;
    public bool hasAutoSpeedBoost = false;
    public float nextAutoBoostTime = 0f;
```

```
[Header("Status Effects")]
public bool isDead = false;

public bool isBurning;
private float burnTimer;
private float burnDamage = 3f;

public bool isPoisoned;
private float poisonTimer;
private float poisonDamage = 2f;

public bool burnOnCrit = false;
public bool burnOnHit = false;
public bool bonusVsPoisoned = false;
public bool hasToxicBoostWhenLow = false;
public bool hasBloodRefund = false;
public bool hasBeastSpeedBoost = false;
public float doubleShotChance = 0f;
public bool attacksMarkEnemies = false;
public bool hasSniperBonus = false;
private bool sniperBonusApplied = false;
public float sniperBonusRange = 2f;
public float sniperBonusDamage = 0.15f;

[Header("Slow Effects")]
public bool attacksSlow = false;
public float slowDuration = 0f;
public float slowChance = 1f;

[Header("Healing / Buffs")]
public bool canAutoHeal = false;
public float autoHealCooldown = 30f;
public float autoHealPercent = 0.2f;
private float lastTimeHealed;

public float onKillHealFactor = 0f;
public float lastDamageTaken;
public float lastDamageTakenTime;
public float lastEnemyHitDamage;

[Header("Standing")]
private Vector3 lastPosition;
public float standingTimer;
public bool isStandingStill;

[Header("Toxic Retaliation")]
public bool retaliatePoisonOnHit = false;

[Header("Ultimates")]
public bool comboStrikeEnabled = false;
public bool hasShieldFortify = false;
public float fortifyDuration = 10f;
public float fortifyTimer = 0f;
public bool hasPiercingShot = false;

[Header("Transmutation Skull Ability")]
public bool transmuteReady = true;
public float transmuteCooldown = 60f;
private float transmuteTimer = 0f;

protected virtual void Start()
{
    animator = GetComponent<Animator>();
    if (!healthInitialized)
```

```
        {
            _currentHealth = maxHealth;
            healthInitialized = true;
        }
        lastPosition = transform.position;
        originalSpeed = moveSpeed;

        if (healthbar != null)
        {
            healthbar.SetMaxHealth(maxHealth);
            healthbar.UpdateHealth(_currentHealth);
        }

        sfx = gameObject.AddComponent<AudioSource>();
        sfx.playOnAwake = false;
        sfx.spatialBlend = 1f;
        sfx.rolloffMode = AudioRolloffMode.Linear;
        sfx.maxDistance = 20f;
        sfx.outputAudioMixerGroup = sfxGroup;
    }

    protected virtual void Update()
    {
        TrackStanding();
        HandleStatusEffects();

        if(hasAutoSpeedBoost && Time.time >= nextAutoBoostTime)
        {
            ActivateBonusSpeed(3f, 5f);
            nextAutoBoostTime = Time.time + 25f;
        }

        if (canAutoHeal && Time.time >= lastTimeHealed + autoHealCooldown)
        {
            Heal(maxHealth * autoHealPercent);
            lastTimeHealed = Time.time;
        }

        if (hasSniperBonus && isStandingStill && standingTimer >= 2f)
        {
            if (!sniperBonusApplied)
            {
                baseDamage *= (1f + sniperBonusDamage);
                attackRange += sniperBonusRange;
                sniperBonusApplied = true;
            }
        }
        else if (sniperBonusApplied)
        {
            baseDamage /= (1f + sniperBonusDamage);
            attackRange -= sniperBonusRange;
            sniperBonusApplied = false;
        }

        if (!transmuteReady)
        {
            transmuteTimer -= Time.deltaTime;
            if (transmuteTimer <= 0f)
            {
                transmuteReady = true;
                Debug.Log("Transmutation Ability Ready Again!");
            }
        }
```

```
        if (Input.GetKeyDown(KeyCode.E) && transmuteReady)
        {
            UseTransmutationAbility();
        }
    }

    protected void PlaySFX(AudioClip clip, float volume = 1f)
    {
        if (clip == null) return;
        sfx.pitch = Random.Range(0.9f, 1.1f); // невеличка варіація, щоб не
монотонно
        sfx.PlayOneShot(clip, volume);
    }

    public void PlayClip(AudioClip clip, float volume = 1f) => PlaySFX(clip,
volume);

    void TrackStanding()
    {
        if (Vector3.Distance(transform.position, lastPosition) < 0.01f)
        {
            standingTimer += Time.deltaTime;
            isStandingStill = true;
        }
        else
        {
            standingTimer = 0f;
            isStandingStill = false;
        }

        lastPosition = transform.position;
    }

    void HandleStatusEffects()
    {
        if (isBurning)
        {
            burnTimer -= Time.deltaTime;
            if (burnTimer <= 0)
            {
                isBurning = false;
            }
            else if (Time.frameCount % 30 == 0)
            {
                TakeDamage(burnDamage, true);
            }
        }

        if (isPoisoned)
        {
            poisonTimer -= Time.deltaTime;
            if (poisonTimer <= 0)
            {
                isPoisoned = false;
            }
            else if (Time.frameCount % 30 == 0)
            {
                TakeDamage(poisonDamage, true);
            }
        }
    }

    public virtual void TakeDamage(float amount, bool isStatusEffect = false,
EnemyStats attacker = null)
```

```
{
    if (isDead) return;

    if (!isStatusEffect && Random.value < damageIgnoreChance) return;
    if (!isStatusEffect)
        PlaySFX(hurtClip, 0.9f);

    float finalDamage = amount * damageResistance;

    if (hasShieldFortify && Time.time <= fortifyTimer)
    {
        finalDamage *= 0.5f;
    }

    _currentHealth -= finalDamage;
    _currentHealth = Mathf.Clamp(_currentHealth, 0, maxHealth);
    lastDamageTaken = finalDamage;
    lastDamageTakenTime = Time.time;

    healthbar?.UpdateHealth(_currentHealth);

    if (!isStatusEffect && retaliatePoisonOnHit && attacker != null)
    {
        attacker.ApplyPoison(2f);
    }

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

public virtual void DealDamage(float amount, bool killed = false)
{
    lastEnemyHitDamage = amount;

    if (killed && onKillHealFactor > 0f)
    {
        Heal(amount * onKillHealFactor);
    }

    if (hasBloodRefund && Time.time - lastDamageTakenTime <= 1f)
    {
        Heal(lastDamageTaken * 0.5f);
    }

    if (burnOnCrit && Random.value < critChance)
    {
        ApplyBurn(2f);
    }
}

public virtual void Heal(float amount)
{
    if (isDead || amount <= 0) return;
    _currentHealth = Mathf.Min(_currentHealth + amount, maxHealth);
    healthbar?.UpdateHealth(_currentHealth);
    lastTimeHealed = Time.time;
}

public virtual void Die()
{
    isDead = true;
    PlaySFX(deathClip, 1f);
```

```
        if (animator != null)
        {
            animator.SetBool("Dead", true);
            DisableScripts();
        }

        StartCoroutine(HandleDeathAfterDelay());
    }

    IEnumerator HandleDeathAfterDelay()
    {
        yield return new WaitForSeconds(deathDelay);

        UI.ChangeMenu(3);
        UI.Pause();
    }

    void DisableScripts()
    {
        MonoBehaviour[] scripts =
GetComponentsInChildren<MonoBehaviour>(true);

        foreach (MonoBehaviour script in scripts)
        {
            if (!(script is Animator  || script == this))
            {
                script.enabled = false;
            }
        }

        // Легкий зсув вгору перед заморозкою
        transform.position += new Vector3(0f, 0.3f, 0f);

        Rigidbody2D rb = GetComponent<Rigidbody2D>();
        if (rb != null)
        {
            rb.velocity = Vector2.zero;
            rb.angularVelocity = 0f;
            rb.bodyType = RigidbodyType2D.Static; // повністю зупиняє об’єкт
        }
    }

    public void ApplyBurn(float duration)
    {
        isBurning = true;
        burnTimer = duration;
    }

    public void ApplyPoison(float duration)
    {
        isPoisoned = true;
        poisonTimer = duration;
    }

    public void ActivateBonusSpeed(float amount, float duration)
    {
        moveSpeed = originalSpeed + amount;
        bonusSpeedAmount = amount;
        bonusSpeedTimer = duration;
    }

    public virtual bool HasEffect(ItemType type)
    {
```

```
        return InventoryManager.Instance != null &&
InventoryManager.Instance.HasItem(type);
    }


    public void UseTransmutationAbility()
    {
        if (!transmuteReady || !HasEffect(ItemType.TransmutationSkull))
return;

        transmuteReady = false;
        transmuteTimer = transmuteCooldown;

        Debug.Log($"{gameObject.name} used Transmutation Skull Ability!");

        TriggerTransmutationEffect();
    }

    protected virtual void TriggerTransmutationEffect()
    {
        // Empty – перевизначується у підкласах
    }
}
```

Файл InventoryManager.cs

Реалізація функціональних задач піднімання предмету, фрагменту карти, отримання ефекту предмету і також переходу на наступний острів, взаємодія з меню інвентаря.

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

public class InventoryManager : MonoBehaviour
{
    public static InventoryManager Instance;

    public List<ItemEffect> items = new List<ItemEffect>();
    public List<InventorySlotUI> slots = new List<InventorySlotUI>();
    public int maxItems = 9;
    public TextMeshProUGUI itemNameText;
    public TextMeshProUGUI descriptionText;
    public int mapFragmentSlotIndex = 8; // 9-та ячейка (індекс 8)
    public ItemEffect fullMap;

    void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
            return;
        }

        Instance = this;
        itemNameText.text = "Empty";
        descriptionText.text = "What to describe?";
    }

    public bool AddItem(ItemEffect item)
    {
        if (item.type == ItemType.MapFragment)
        {
```

```
            int currentFragments = items.Count(i => i.type ==
ItemType.MapFragment);
            if (currentFragments >= 4) return false; // вже максимум

            items.Add(item);
            UpdateInventoryUI(items);

            if (currentFragments + 1 == 4)
            {
                // Замінюємо фрагменти на карту
                items.RemoveAll(i => i.type == ItemType.MapFragment);

                items.Add(fullMap);
                UpdateInventoryUI(items);
                if (fullMap == null)
                {
                    Debug.LogError("FullMap is null!");
                }
                else if (!(fullMap is FullMapEffect))
                {
                    Debug.LogError($"FullMap is not of type FullMapEffect!
It's actually {fullMap.GetType()}");
                }
                if (fullMap is FullMapEffect fullMapEffect)
                {

fullMapEffect.ApplyEffect(FindObjectOfType<CharacterStatsBase>());
                    items.RemoveAll(i => i.type == ItemType.FullMap);
                }
                else
                {
                    Debug.LogError("FullMap is not of type FullMapEffect!");
                }
            }

            return true;
        }

        if (items.Count(i => i.type != ItemType.MapFragment) >= maxItems - 1)
return false;

        items.Add(item);
        ApplyToAllCharacters();
        UpdateInventoryUI(items);
        return true;
    }

    public bool HasItem(ItemType type) => items.Any(i => i.type == type);
    public int CountOf(ItemType type) => items.Count(i => i.type == type);

    public void ApplyToAllCharacters()
    {
        foreach (var stats in FindObjectsOfType<CharacterStatsBase>())
        {
            foreach (var item in items)
            {
                item.ApplyEffect(stats: stats);
            }
        }
    }

    public void ReapplyTo(CharacterStatsBase stats)
    {
        foreach (var item in items)
```

```
        {
            item.ApplyEffect(stats: stats);
        }
    }

    public void UpdateInventoryUI(List<ItemEffect> items)
    {
        foreach (var slot in slots)
            slot.Clear();

        var mapFragments = items.Where(i => i.type == ItemType.MapFragment ||
i.type == ItemType.FullMap).ToList();
        var otherItems = items.Where(i => i.type != ItemType.MapFragment &&
i.type != ItemType.FullMap).ToList();

        if (mapFragments.Count > 0)
            slots[mapFragmentSlotIndex].SetItem(mapFragments[0]); //
показуємо лише один

        int index = 0;
        for (int i = 0; i < otherItems.Count && index < slots.Count; i++)
        {
            if (index == mapFragmentSlotIndex) index++; // пропустити слот
для фрагментів
            if (index >= slots.Count) break;

            slots[index].SetItem(otherItems[i]);
            index++;
        }
    }

    public void ShowItemNameAndDescription(string name, string description)
    {
        itemNameText.text = name;
        descriptionText.text = description;
    }

    public void HideItemName()
    {
        itemNameText.text = "Empty";
        descriptionText.text = "What to describe?";
    }

    public void AssignUISlots(List<InventorySlotUI> newSlots)
    {
        slots = newSlots;
        UpdateInventoryUI(items);
    }

    public void AssignTexts(TextMeshProUGUI itemName, TextMeshProUGUI
description)
    {
        itemNameText = itemName;
        descriptionText = description;
    }
}
```

Файл PauseManager.cs

Реалізація функціональних задач взаємодії різних панелей: паузи, інвентаря, налаштувань, головного меню.

```
using UnityEngine;

public class PauseManager : MonoBehaviour
{
    [Header("Panels / Buttons")]
    public GameObject[] pauseMenus;
    public GameObject[] pauseButtons;

    GameObject currentMenu;

    void Awake()
    {
        if (pauseMenus != null && pauseMenus.Length > 0)
            currentMenu = pauseMenus[0];
    }

    public void Pause()
    {
        Time.timeScale = 0f;
        currentMenu?.SetActive(true);

        if (currentMenu != null && currentMenu.name == "InventoryPanel")

InventoryManager.Instance?.UpdateInventoryUI(InventoryManager.Instance.items)
;

        foreach (var b in pauseButtons) if (b) b.SetActive(false);
    }

    public void Resume()
    {
        Time.timeScale = 1f;
        currentMenu?.SetActive(false);
        foreach (var b in pauseButtons) if (b) b.SetActive(true);
    }

    public void ChangeMenu(int index)
    {
        if (index < 0 || index >= pauseMenus.Length) return;
        currentMenu = pauseMenus[index];
    }

    public void ChangeToOtherMenu(int index)
    {
        if (index < 0 || index >= pauseMenus.Length) return;
        currentMenu.SetActive(false);
        currentMenu = pauseMenus[index];
        currentMenu.SetActive(true);
    }

    public void ExitGame() => Application.Quit();
}
```

Файл EnemyAI.cs

Реалізація функціональної задачі наявності штучного інтелекту ворогів, а саме взаємодії з гравцем і основна поведінка ворогів.

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

[RequireComponent(typeof(EnemyStats))]
public class EnemyAI : MonoBehaviour
{
    [Header("Behaviour")]
    public float detectionRadius = 5f;
    public float attackRadius = 2.5f;
    public float attackCooldown = 1f;
    public float retreatTime = 0.5f;
    public float moveSpeed = 2f;
    public float pathRefresh = 1.5f;

    [Header("Level Binding")]
    public EdgeBlocker.Level allowedLevel;
    public IslandGenerator islandGenerator;

    [Header("Visual")]
    public GameObject healthbar;

    Transform player;
    EnemyStats stats;
    Animator anim;
    Renderer rend;
    SimpleGridPathfinder path;

    List<Vector3> curPath;
    int pathIdx;
    float pathTimer;
    bool isRetreat;
    float retreatTimer;
    Vector3 spawnPos;
    static readonly Vector3 VISUAL_OFFSET = Vector3.down * 0.9f;
    Vector3 logicalSpawn;

    void Start()
    {
        player = CharacterSwitcher.currentPlayer?.transform;
        stats = GetComponent<EnemyStats>();
        anim = GetComponent<Animator>();
        rend = GetComponent<Renderer>();
        spawnPos = transform.position;
        logicalSpawn = spawnPos - VISUAL_OFFSET;

        path = gameObject.AddComponent<SimpleGridPathfinder>();
        path.Initialize(
            islandGenerator.sandTilemap,
            islandGenerator.grassTilemap,
            islandGenerator.mountainTilemap,
            new[] {
                islandGenerator.wallTilemap,
                islandGenerator.higherWallTilemap,
                islandGenerator.grassLadderTilemap,
                islandGenerator.mountainLadderTilemap
            },
            allowedLevel);
    }
```

```
    void Update()
    {
        if (!rend.isVisible) return;
        if (stats.isDead) return;
        if (CharacterSwitcher.currentPlayer != null)
            player = CharacterSwitcher.currentPlayer.transform;
        if (player == null) return;

        PlayerMovement pm = player.GetComponent<PlayerMovement>();
        bool sameLvl = pm != null && pm.GetCurrentLevel() == allowedLevel;
        float distToPlayer = Vector3.Distance(transform.position,
player.position);

        if (isRetreat)
        {
            retreatTimer -= Time.deltaTime;
            if (retreatTimer <= 0f) isRetreat = false;
            FollowPath(moveSpeed);
            return;
        }

        if (sameLvl && distToPlayer <= attackRadius)
        {
            Face(player.position - transform.position);
            if ((attackCooldown -= Time.deltaTime) <= 0f)
            {
                if (player.TryGetComponent(out CharacterStatsBase target))
                    StartCoroutine(Attack(target));
                attackCooldown = 1f;
            }
            return;
        }

        pathTimer -= Time.deltaTime;

        if (sameLvl && distToPlayer <= detectionRadius)
        {
            if (NeedsPath()) Repath(player.position);
        }
        else
        {
            if (NeedsPath()) Repath(GetRandomPatrolPoint());
        }

        FollowPath(moveSpeed);
    }

    bool NeedsPath() => curPath == null || pathIdx >= curPath.Count ||
pathTimer <= 0f;

    void Repath(Vector3 logicalTarget)
    {
        Vector3 logicalSelf = transform.position - VISUAL_OFFSET;

        curPath = path.FindPath(logicalSelf, logicalTarget);
        pathIdx = 0;
        pathTimer = pathRefresh;
    }

    void FollowPath(float speed)
    {
        if (curPath == null || pathIdx >= curPath.Count)
        { anim.SetFloat("Speed", 0); return; }
```

```
        Vector3 dst = curPath[pathIdx] + VISUAL_OFFSET;
        Vector3 dir = dst - transform.position;
        float step = speed * Time.deltaTime;

        if (dir.magnitude <= step) { transform.position = dst; pathIdx++; }
        else { transform.position += dir.normalized * step; }

        Face(dir);
        anim.SetFloat("Speed", speed);
    }

    void Face(Vector3 dir)
    {
        if (Mathf.Abs(dir.x) < 0.01f) return;
        bool right = dir.x > 0;
        transform.localScale = new Vector3(right ? 1 : -1, 1, 1);
        healthbar.transform.localScale = new Vector3(right ? 0.05f : -0.05f,
0.05f, 0.05f);
    }

    IEnumerator Attack(CharacterStatsBase target)
    {
        anim.SetBool("Attack", true);
        yield return new WaitForSeconds(0.2f);
        stats.AttackTarget(target);
        yield return new WaitForSeconds(0.3f);
        anim.SetBool("Attack", false);

        isRetreat = true;
        retreatTimer = retreatTime;
        Repath(logicalSpawn);
    }

    Vector3 GetRandomPatrolPoint()
    {
        Tilemap map = GetLevelMap();
        map.CompressBounds();
        List<Vector3Int> cells = new();

        foreach (var c in map.cellBounds.allPositionsWithin)
            if (IsCellWalkable(c)) cells.Add(c);

        if (cells.Count == 0) return transform.position;
        return map.GetCellCenterWorld(cells[Random.Range(0, cells.Count)]);
    }

    bool IsCellWalkable(Vector3Int c)
    {
        if (!path.IsTopmostTile(c)) return false;
        if (islandGenerator.wallTilemap.HasTile(c) ||
            islandGenerator.higherWallTilemap.HasTile(c) ||
            islandGenerator.grassLadderTilemap.HasTile(c) ||
            islandGenerator.mountainLadderTilemap.HasTile(c))
            return false;
        return true;
    }

    Tilemap GetLevelMap() => allowedLevel switch
    {
        EdgeBlocker.Level.Sand => islandGenerator.sandTilemap,
        EdgeBlocker.Level.Grass => islandGenerator.grassTilemap,
        EdgeBlocker.Level.Mountain => islandGenerator.mountainTilemap,
        _ => null
```

```
    };
}
```

Файл SimpleGridPathfinder.cs

Реалізація функціональної задачі наявності штучного інтелекту ворогів, а саме пошуку шляхів для патрулювання або переслідування гравця.

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

public class SimpleGridPathfinder : MonoBehaviour
{
    [SerializeField] private Tilemap sandTilemap;
    [SerializeField] private Tilemap grassTilemap;
    [SerializeField] private Tilemap mountainTilemap;

    [SerializeField] private Tilemap[] blockers;

    public EdgeBlocker.Level level;

    public void Initialize(
        Tilemap sand, Tilemap grass, Tilemap mountain,
        Tilemap[] blockingMaps, EdgeBlocker.Level allowed)
    {
        sandTilemap = sand;
        grassTilemap = grass;
        mountainTilemap = mountain;
        blockers = blockingMaps;
        level = allowed;
    }

    public List<Vector3> FindPath(Vector3 startWorld, Vector3 targetWorld)
    {
        Tilemap map = GetLevelMap(level);
        if (map == null) return null;

        Vector3Int start = map.WorldToCell(startWorld);
        Vector3Int goal = map.WorldToCell(targetWorld);

        if (!IsTopmostTile(start) || !IsTopmostTile(goal)) return null;

        /* A* */
        List<Node> open = new();                          // відкрита множина
        HashSet<Vector3Int> closed = new();               // закрита множина
        Dictionary<Vector3Int, Node> all = new();

        Node startNode = new Node(start, 0, Heuristic(start, goal));
        open.Add(startNode); all[start] = startNode;

        Vector3Int[] dirs = { Vector3Int.up, Vector3Int.down,
                              Vector3Int.left, Vector3Int.right };

        while (open.Count > 0)
        {
            Node cur = PopLowestF(open);
            if (cur.pos == goal) return Reconstruct(cur, map);

            open.Remove(cur); closed.Add(cur.pos);

            foreach (Vector3Int d in dirs)
            {
                Vector3Int nb = cur.pos + d;
                if (closed.Contains(nb)) continue;
```

```
            if (!IsTopmostTile(nb)) continue;
            if (IsBlocked(nb)) continue;

            int g = cur.g + 1;
            if (!all.TryGetValue(nb, out Node n))
            {
                n = new Node(nb, g, Heuristic(nb, goal));
                n.parent = cur;
                all[nb] = n;
                open.Add(n);
            }
            else if (g < n.g)
            {
                n.g = g;
                n.parent = cur;
            }
        }
    }
    return null;    // шлях не знайдено
}

public bool IsTopmostTile(Vector3Int c)
{
    switch (level)
    {
        case EdgeBlocker.Level.Sand:
            return sandTilemap.HasTile(c) &&
                   !grassTilemap.HasTile(c) &&
                   !mountainTilemap.HasTile(c);

        case EdgeBlocker.Level.Grass:
            return grassTilemap.HasTile(c) &&
                   !mountainTilemap.HasTile(c);

        case EdgeBlocker.Level.Mountain:
            return mountainTilemap.HasTile(c);
    }
    return false;
}

Tilemap GetLevelMap(EdgeBlocker.Level l) => l switch
{
    EdgeBlocker.Level.Sand => sandTilemap,
    EdgeBlocker.Level.Grass => grassTilemap,
    EdgeBlocker.Level.Mountain => mountainTilemap,
    _ => null
};

bool IsBlocked(Vector3Int c)
{
    foreach (var b in blockers)
        if (b != null && b.HasTile(c)) return true;
    return false;
}

int Heuristic(Vector3Int a, Vector3Int b) =>
    Mathf.Abs(a.x - b.x) + Mathf.Abs(a.y - b.y);

Node PopLowestF(List<Node> list)
{
    Node best = list[0];
    foreach (var n in list)
        if (n.F < best.F || (n.F == best.F && n.h < best.h)) best = n;
    return best;
```

```
    }

    List<Vector3> Reconstruct(Node n, Tilemap map)
    {
        List<Vector3> p = new();
        while (n.parent != null)
        {
            p.Add(map.GetCellCenterWorld(n.pos));
            n = n.parent;
        }
        p.Reverse();
        return p;
    }

    class Node
    {
        public Vector3Int pos;
        public int g, h;              // g-cost, h-cost
        public int F => g + h;
        public Node parent;
        public Node(Vector3Int p, int g, int h)
        { pos = p; this.g = g; this.h = h; }
    }
}
```

Файл EnemySpawner.cs

Реалізація функціональної задачі розкидання супротивників по карті.

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

public class EnemySpawner : MonoBehaviour
{
    [Header("Links")]
    public IslandGenerator islandGenerator;
    public Transform enemyParent;

    [Header("Prefabs")]
    public GameObject[] sandEnemies;
    public GameObject[] grassEnemies;
    public GameObject[] mountainEnemies;

    [Header("Counts")]
    public int sandEnemyCount = 3;
    public int grassEnemyCount = 4;
    public int mountainEnemyCount = 2;
    static readonly Vector3 VISUAL_OFFSET = Vector3.down * 0.9f;

    void Start()
    {
        if (islandGenerator == null)
        {
            Debug.LogError("[EnemySpawner] IslandGenerator reference is
missing");
            return;
        }

        islandGenerator.OnGenerationComplete += () =>
StartCoroutine(DelayedSpawn());
        if (islandGenerator.generationCompleted)
StartCoroutine(DelayedSpawn());
    }
```

```
    IEnumerator DelayedSpawn()
    {
        yield return new WaitForSeconds(0.4f);

        SpawnOnTilemap(
            islandGenerator.sandTilemap,
            sandEnemies,
            sandEnemyCount,
            EdgeBlocker.Level.Sand,
            new[] { islandGenerator.grassTilemap,
islandGenerator.mountainTilemap });

        SpawnOnTilemap(
            islandGenerator.grassTilemap,
            grassEnemies,
            grassEnemyCount,
            EdgeBlocker.Level.Grass,
            new[] { islandGenerator.mountainTilemap });

        SpawnOnTilemap(
            islandGenerator.mountainTilemap,
            mountainEnemies,
            mountainEnemyCount,
            EdgeBlocker.Level.Mountain,
            null);
    }

    void SpawnOnTilemap(
        Tilemap levelMap,
        GameObject[] prefabs,
        int count,
        EdgeBlocker.Level level,
        Tilemap[] higherMaps)
    {
        if (prefabs == null || prefabs.Length == 0) return;

        List<Vector3Int> candidates = CollectValidTiles(levelMap, level,
higherMaps);
        if (candidates.Count == 0) return;

        HashSet<Vector3Int> used = new HashSet<Vector3Int>();

        for (int i = 0; i < count; i++)
        {
            Vector3Int cell;
            int safety = 50;
            do { cell = candidates[Random.Range(0, candidates.Count)]; }
while (used.Contains(cell) && --safety > 0);
            used.Add(cell);

            Vector3 spawnPos = levelMap.GetCellCenterWorld(cell) +
VISUAL_OFFSET;
            GameObject enemyGO = Instantiate(
                prefabs[Random.Range(0, prefabs.Length)],
                spawnPos,
                Quaternion.identity,
                enemyParent);

            if (enemyGO.TryGetComponent(out EnemyAI ai))
                ai.allowedLevel = level;
        }
    }

    List<Vector3Int> CollectValidTiles(
```

```
        Tilemap levelMap,
        EdgeBlocker.Level level,
        Tilemap[] higherMaps)
    {
        List<Vector3Int> list = new List<Vector3Int>();
        levelMap.CompressBounds();

        foreach (Vector3Int pos in levelMap.cellBounds.allPositionsWithin)
        {
            if (!IsTopmostForLevel(pos, level, higherMaps)) continue;

            // блокери (стіни/драбини)
            if (islandGenerator.wallTilemap.HasTile(pos) ||
                islandGenerator.higherWallTilemap.HasTile(pos) ||
                islandGenerator.grassLadderTilemap.HasTile(pos) ||
                islandGenerator.mountainLadderTilemap.HasTile(pos))
                continue;

            list.Add(pos);
        }

        return list;
    }

    bool IsTopmostForLevel(Vector3Int cell, EdgeBlocker.Level level,
Tilemap[] higherMaps)
    {
        // На клітинці ОБОВ'ЯЗКОВО має бути тайл потрібного рівня
        switch (level)
        {
            case EdgeBlocker.Level.Sand: if
(!islandGenerator.sandTilemap.HasTile(cell)) return false; break;
            case EdgeBlocker.Level.Grass: if
(!islandGenerator.grassTilemap.HasTile(cell)) return false; break;
            case EdgeBlocker.Level.Mountain: if
(!islandGenerator.mountainTilemap.HasTile(cell)) return false; break;
        }

        // Переконатися, що над нею немає тайлів вищих шарів
        if (higherMaps != null)
            foreach (Tilemap t in higherMaps)
                if (t != null && t.HasTile(cell)) return false;

        return true;
    }
}
```

Файл ItemSpawner.cs

Реалізація функціональної задачі розкидання предметів по карті.

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

public class ItemSpawner : MonoBehaviour
{
    public IslandGenerator islandGenerator;

    public List<ItemSpawnData> sandItems;
    public List<ItemSpawnData> grassItems;
    public List<ItemSpawnData> mountainItems;

    public Transform itemParent;
```

```
    void Start()
    {
        if (islandGenerator == null)
        {
            Debug.LogError("IslandGenerator не встановлено!");
            return;
        }

        islandGenerator.OnGenerationComplete += () =>
        {
            StartCoroutine(DelayedSpawn());
        };
    }

    IEnumerator DelayedSpawn()
    {
        yield return new WaitForSeconds(0.1f);

        SpawnItemsOnTilemap(islandGenerator.sandTilemap, sandItems, new
Tilemap[] { islandGenerator.grassTilemap, islandGenerator.mountainTilemap },
islandGenerator.wallTilemap, islandGenerator.higherWallTilemap);
        SpawnItemsOnTilemap(islandGenerator.grassTilemap, grassItems, new
Tilemap[] { islandGenerator.mountainTilemap }, islandGenerator.wallTilemap,
islandGenerator.higherWallTilemap);
        SpawnItemsOnTilemap(islandGenerator.mountainTilemap, mountainItems,
null, islandGenerator.wallTilemap, islandGenerator.higherWallTilemap);
    }

    void SpawnItemsOnTilemap(Tilemap tilemap, List<ItemSpawnData>
itemsToSpawn, Tilemap[] blockedAbove, Tilemap wallTilemap, Tilemap
ladderTilemap)
    {
        List<Vector3Int> validTiles = new List<Vector3Int>();

        tilemap.CompressBounds();
        foreach (var pos in tilemap.cellBounds.allPositionsWithin)
        {
            if (!tilemap.HasTile(pos)) continue;

            // Перевірка: не має бути стіни чи драбини
            if ((wallTilemap != null && wallTilemap.HasTile(pos)) ||
(ladderTilemap != null && ladderTilemap.HasTile(pos)))
                continue;

            // Блок над рівнем
            bool blocked = false;
            if (blockedAbove != null)
            {
                foreach (var above in blockedAbove)
                {
                    if (above.HasTile(pos))
                    {
                        blocked = true;
                        break;
                    }
                }
            }

            if (!blocked)
                validTiles.Add(pos);
        }

        foreach (var itemData in itemsToSpawn)
        {
```

```
            for (int i = 0; i < itemData.count; i++)
            {
                if (validTiles.Count == 0) return;

                Vector3Int spawnTile = validTiles[Random.Range(0,
validTiles.Count)];
                Vector3 worldPos = tilemap.CellToWorld(spawnTile) + new
Vector3(0.5f, 0.5f, 0);

                Instantiate(itemData.prefab, worldPos, Quaternion.identity,
itemParent);
            }
        }
    }
}

[System.Serializable]
public class ItemSpawnData
{
    public GameObject prefab;
    public int count;
}
```

Файл EnemyStats.cs

Реалізація взаємодії ворогів з гравцем: отримання або нанесення шкоди, смерть, також отримання або нанесення ефектів.

```
using UnityEngine;
using System.Collections;
using UnityEngine.Audio;
using UnityEngine.SceneManagement;

public class EnemyStats : MonoBehaviour
{
    [Header("Scene change on death")]
    public bool loadSceneOnDeath = false;
    public string sceneToLoad = "Win";

    [Header("Sound FX")]
    public AudioClip hitClip;
    public AudioClip hurtClip;
    public AudioClip deathClip;
    AudioSource sfx;
    public AudioMixerGroup sfxGroup;

    public float maxHealth = 100f;
    public float _currentHealth = 100f;
    public float CurrentHealth
    {
        get => _currentHealth;
        set
        {
            _currentHealth = Mathf.Clamp(value, 0, maxHealth);
        }
    }
    public float baseDamage = 10f;

    public bool isSlowed = false;
    private float slowTimer = 0f;
    public float speed = 2f;
    private float originalSpeed;

    public bool isDead = false;
    public bool isPoisoned = false;
```

```
    public bool isBurning = false;
    public bool isMarked = false;

    private float poisonTimer = 0f;
    private float burnTimer = 0f;

    private SpriteRenderer spriteRenderer;
    private Color originalColor;
    private Animator animator;

    public Healthbar healthbar;

    private void Start()
    {
        _currentHealth = maxHealth;
        originalSpeed = speed;
        spriteRenderer = GetComponent<SpriteRenderer>();
        animator = GetComponent<Animator>();

        if (spriteRenderer != null)
            originalColor = spriteRenderer.color;

        healthbar.SetMaxHealth(maxHealth);

        sfx = gameObject.AddComponent<AudioSource>();
        sfx.playOnAwake = false;
        sfx.spatialBlend = 1f;
        sfx.rolloffMode = AudioRolloffMode.Linear;
        sfx.maxDistance = 20f;
        sfx.outputAudioMixerGroup = sfxGroup;
    }

    private void Update()
    {
        HandlePoison();
        HandleBurn();
        HandleSlow();
    }

    protected void PlaySFX(AudioClip clip, float volume = 1f)
    {
        if (clip == null) return;
        sfx.pitch = Random.Range(0.9f, 1.1f); // невеличка варіація, щоб не
монотонно
        sfx.PlayOneShot(clip, volume);
    }

    public void TakeDamage(float amount)
    {
        if (isDead) return;

        float finalDamage = amount;

        if (isMarked)
            finalDamage *= 1.4f;

        _currentHealth -= finalDamage;
        healthbar.UpdateHealth(_currentHealth);
        PlaySFX(hurtClip, 0.9f);

        if (_currentHealth <= 0)
        {
            Die();
        }
```

```
}

public void ApplyPoison(float duration)
{
    isPoisoned = true;
    poisonTimer = duration;
    FlashColor(Color.green);
}

public void ApplyBurn(float duration)
{
    isBurning = true;
    burnTimer = duration;
    FlashColor(Color.red);
}

public void ApplySlow(float duration)
{
    isSlowed = true;
    slowTimer = duration;
    speed = originalSpeed * 0.5f;
}

private void HandlePoison()
{
    if (!isPoisoned) return;

    poisonTimer -= Time.deltaTime;
    if (poisonTimer <= 0)
    {
        isPoisoned = false;
        ResetColor();
    }
    else if (Time.frameCount % 30 == 0)
    {
        TakeDamage(2f);
    }
}

private void HandleSlow()
{
    if (!isSlowed) return;

    slowTimer -= Time.deltaTime;
    if (slowTimer <= 0)
    {
        isSlowed = false;
        speed = originalSpeed;
    }
}

private void HandleBurn()
{
    if (!isBurning) return;

    burnTimer -= Time.deltaTime;
    if (burnTimer <= 0)
    {
        isBurning = false;
        ResetColor();
    }
    else if (Time.frameCount % 30 == 0)
    {
        TakeDamage(3f);
```

```
        }
    }

    public bool IsDead()
    {
        return isDead;
    }

    private void Die()
    {
        isDead = true;
        PlaySFX(deathClip, 0.9f);
        if (healthbar != null)
        {
            Destroy(healthbar.gameObject);
        }
        if (animator != null)
        {
            animator.SetBool("Dead", true);
            DisableScripts();
        }

        StartCoroutine(HandleDeathAfterDelay(0.5f));
    }

    private void FlashColor(Color color)
    {
        if (spriteRenderer != null)
            spriteRenderer.color = color;
    }

    private void ResetColor()
    {
        if (spriteRenderer != null)
            spriteRenderer.color = originalColor;
    }

    public void AttackTarget(CharacterStatsBase target)
    {
        float actualDamage = baseDamage;

        if (isBurning && target is KnightStats &&
target.HasEffect(ItemType.FlameCrystal))
        {
            actualDamage *= 0.75f; // -25% шкоди
        }
        PlaySFX(hitClip, 0.9f);
        target.TakeDamage(actualDamage);
    }

    void DisableScripts()
    {
        MonoBehaviour[] scripts =
GetComponentsInChildren<MonoBehaviour>(true);

        foreach (MonoBehaviour script in scripts)
        {
            if (!(script is Animator || script == this))
            {
                script.enabled = false;
            }
        }

        // Легкий зсув вгору перед заморозкою
```

```
        transform.position += new Vector3(0f, 0.3f, 0f);

        Rigidbody2D rb = GetComponent<Rigidbody2D>();
        if (rb != null)
        {
            rb.velocity = Vector2.zero;
            rb.angularVelocity = 0f;
            rb.bodyType = RigidbodyType2D.Static; // повністю зупиняє об’єкт
        }
    }

    IEnumerator HandleDeathAfterDelay(float deathDelay)
    {
        yield return new WaitForSeconds(deathDelay);

        if (loadSceneOnDeath && !string.IsNullOrEmpty(sceneToLoad))
        {
            SceneManager.LoadScene(sceneToLoad);
        }

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

Файл PlayerSpawner.cs

Реалізація функціональної задачі створення гравця на карті в правильній точці.

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

public class PlayerSpawner : MonoBehaviour
{
    public IslandGenerator islandGenerator;
    public GameObject playerPrefab;

    void Start()
    {
        if (islandGenerator == null)
        {
            Debug.LogError("IslandGenerator не встановлено! Перетягніть
об'єкт у PlayerSpawner в інспекторі.");
            RestartLevel();
            return;
        }
        try
        {
            StartCoroutine(WaitForIslandGeneration());
        }
        catch (System.Exception ex)
        {
            Debug.LogError("Помилка під час запуску генерації острова: " +
ex.Message);
            RestartLevel();
        }
    }

    IEnumerator WaitForIslandGeneration()
    {
        yield return new WaitUntil(() =>
GetTileCount(islandGenerator.sandTilemap) > 0);

        Debug.Log("Острів згенеровано! Запускаємо пошук піщаних островів.");
```

```
        List<HashSet<Vector3Int>> sandIslands =
islandGenerator.getSandIslands();

        if (sandIslands.Count > 0)
        {
            List<Vector3Int> islandTiles = FindChosen(sandIslands);
            // Фільтруємо лише внутрішні плитки (відкидаємо межі)
            List<Vector3Int> safeTiles = new List<Vector3Int>();
            foreach (Vector3Int tile in islandTiles)
            {
                bool isEdge = false;
                foreach (Vector3Int dir in new Vector3Int[] {
                new Vector3Int(1, 0, 0), new Vector3Int(-1, 0, 0),
                new Vector3Int(0, 1, 0), new Vector3Int(0, -1, 0) })
                {
                    Vector3Int neighbor = tile + dir;
                    if (!islandGenerator.sandTilemap.HasTile(neighbor)) //
Якщо хоча б один сусід не є піском, значить це край острова
                    {
                        isEdge = true;
                        break;
                    }
                }
                if (!isEdge)
                    safeTiles.Add(tile);
            }

            if (safeTiles.Count == 0)
            {
                Debug.LogWarning("Не знайдено внутрішніх тайлів для
спавну!");
                RestartLevel();
                yield break;
            }

            SpawnPlayer(safeTiles[Random.Range(0, safeTiles.Count)]);
        }
        else
        {
            Debug.LogWarning("Не знайдено жодного піщаного острова для спавну
гравця!");
            RestartLevel();
            yield break;
        }
    }

    int GetTileCount(Tilemap tilemap)
    {
        tilemap.CompressBounds();
        BoundsInt bounds = tilemap.cellBounds;
        TileBase[] allTiles = tilemap.GetTilesBlock(bounds);
        int count = 0;

        foreach (TileBase tile in allTiles)
        {
            if (tile != null) count++;
        }
        return count;
    }

    List<Vector3Int> FindChosen(List<HashSet<Vector3Int>> islands)
    {
        List<HashSet<Vector3Int>> validIslands = new
List<HashSet<Vector3Int>>();
```

```
        foreach (var island in islands)
        {
            if (island.Count < 10 || island.Count > 30)
                continue;

            bool bridgeTouching = false;

            foreach (Vector3Int tile in island)
            {
                for (int dx = -1; dx <= 1; dx++)
                {
                    for (int dy = -1; dy <= 1; dy++)
                    {
                        Vector3Int neighbor = tile + new Vector3Int(dx, dy,
0);
                        if
(islandGenerator.sandBridgeTilemap.HasTile(neighbor))
                        {
                            bridgeTouching = true;
                            break;
                        }
                    }
                    if (bridgeTouching) break;
                }

                if (bridgeTouching)
                    break;
            }

            if (bridgeTouching)
            {
                validIslands.Add(island);
            }
        }

        if (validIslands.Count == 0)
        {
            Debug.LogWarning("Не знайдено жодного острова, що має міст.");
            RestartLevel();
            return new List<Vector3Int> { Vector3Int.zero }; // Запобігає
крашу
        }

        HashSet<Vector3Int> chosenIsland = validIslands[Random.Range(0,
validIslands.Count)];
        return new List<Vector3Int>(chosenIsland);
    }

    void SpawnPlayer(Vector3Int spawnTile)
    {
        if (spawnTile == Vector3Int.zero)
        {
            Debug.LogWarning("Спавн не відбувся, координати невірні!");
            RestartLevel();
            return;
        }

        Vector3 worldPosition =
islandGenerator.sandTilemap.CellToWorld(spawnTile);
        playerPrefab.transform.position = worldPosition + new Vector3(0.8f, -
0.5f, 0);
        Debug.Log($"Гравець заспавнений на {worldPosition}");
    }
```

```
    void RestartLevel()
    {
        Debug.LogWarning("Рестарт рівня через помилку спавну!");
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ІГРОВИЙ ЗАСТОСУНОК У ЖАНТІ ROGUELIKE

Програма та методика тестування

КПІ.ІП-1228.045480.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Андрій СІМЧУК

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є ігровий застосунок у стилі Roguelike, розроббений на мові програмування C# з використання ігрового двигуна Unity.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка сумісності застосунку з різними операційними системами (Windows, Linux, MacOS;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- вбудована перевірка коду від Visual Studio;
- вбудована перевірка від Unity Editor.

Порядок проведення тестування буде наступним:

- тестування на комп’ютерних пристроях з різною роздільною здатністю екрана;
- тестування на комп’ютерних пристроях з різною версією операційної системи;
- тестування зручності використання;
- мануальне тестування на відповідність функціональним вимогам;
- тестування на коректність обробки помилки.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ROGUELIKE

Керівництво користувача

КПІ.ІП-1228.045480.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Андрій СІМЧУК

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«One Man Army» – це ігровий застосунок у стилі Roguelike, з відповідними для цього жанру особливостями (рандомізація і пермасмерть), який забезпечує можливість виконання усіх необхідних функцій серед яких:

- можливість руху персонажа по рівню;
- можливість змінити рівень;
- нанесення шкоди ворогові;
- отримання шкоди від ворога;
- збирання предметів, які надають ефекти персонажам;
- при зборі достатньої кількості фрагментів карти – переніс наступний рівень.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення «One Man Army», розроблене на основі ігрового рушія Unity, повинно функціонувати на IBM-сумісних персональних комп’ютерах під керуванням операційної системи Windows, macOS або Linux.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5 або Apple M1;

– об‘єм ОЗП: 4 Гб;

– графічна карта: вбудована, з підтримкою Metal (на macOS) з підтримкою DirectX 11 / OpenGL 4.1;

– вільне місце на диску: не менше 0.3 ГБ.

Рекомендована конфігурація технічних засобів

– тип процесору: Apple Silicon M2 (MacBook Pro 2022+);

– об‘єм ОЗП: 16 Гб;

– графічна карта: вбудована Apple GPU (M2);

– вільне місце на диску: не менше 0.5 ГБ.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи відповідні exe-файл (для Windows) або app-файл (для MacOS). Для цього спершу необхідно завантажити його на пристрій, а потім, використовуючи який-небудь інсталятор, виконати встановлення даного додатку.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі пристрою повинна відобразитись іконка даного застосунку. 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.

3 ВИКОНАННЯ ПРОГРАМИ

Одразу після запуску ігрового застосунку відкривається головне меню гри (рис. 3.1).

Рисунок 3.1 – Головне меню

Після натиснення кнопки «Play» гравець потрапляє на перший острів (рис. 3.2).

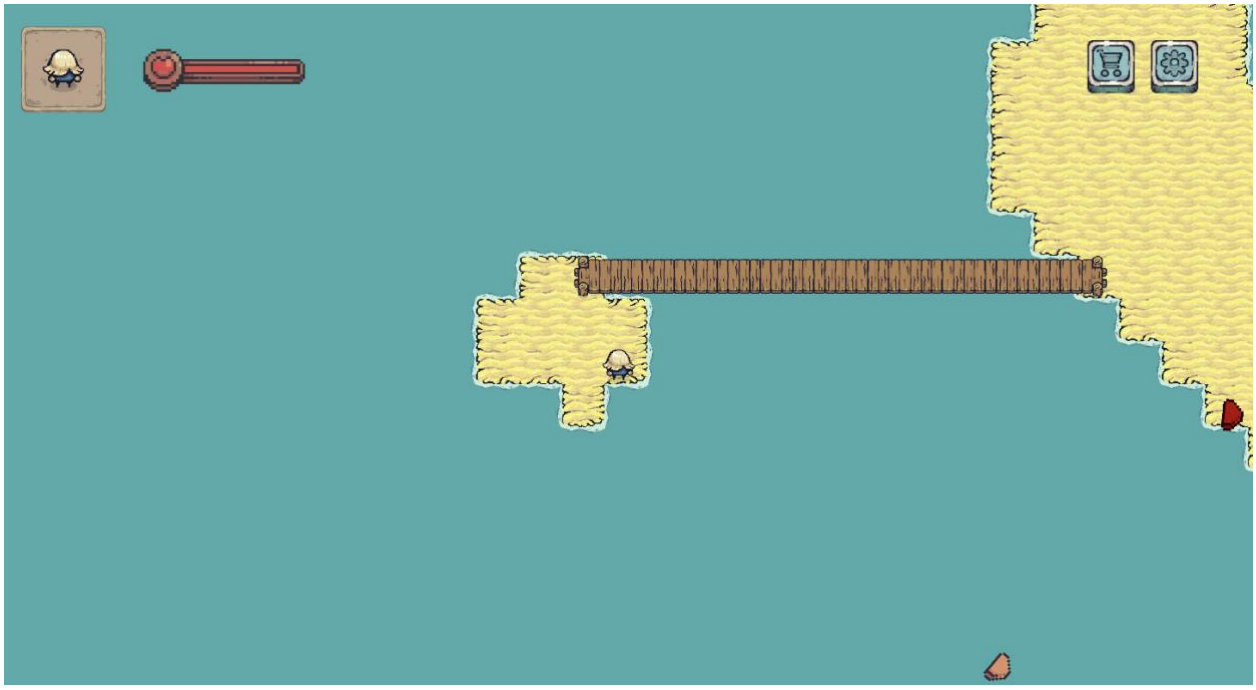


Рисунок 3.2 – Гравець заспавнений на першому рівні

При натисканні клавіш руху персонаж починає рухатись (рис. 3.3).

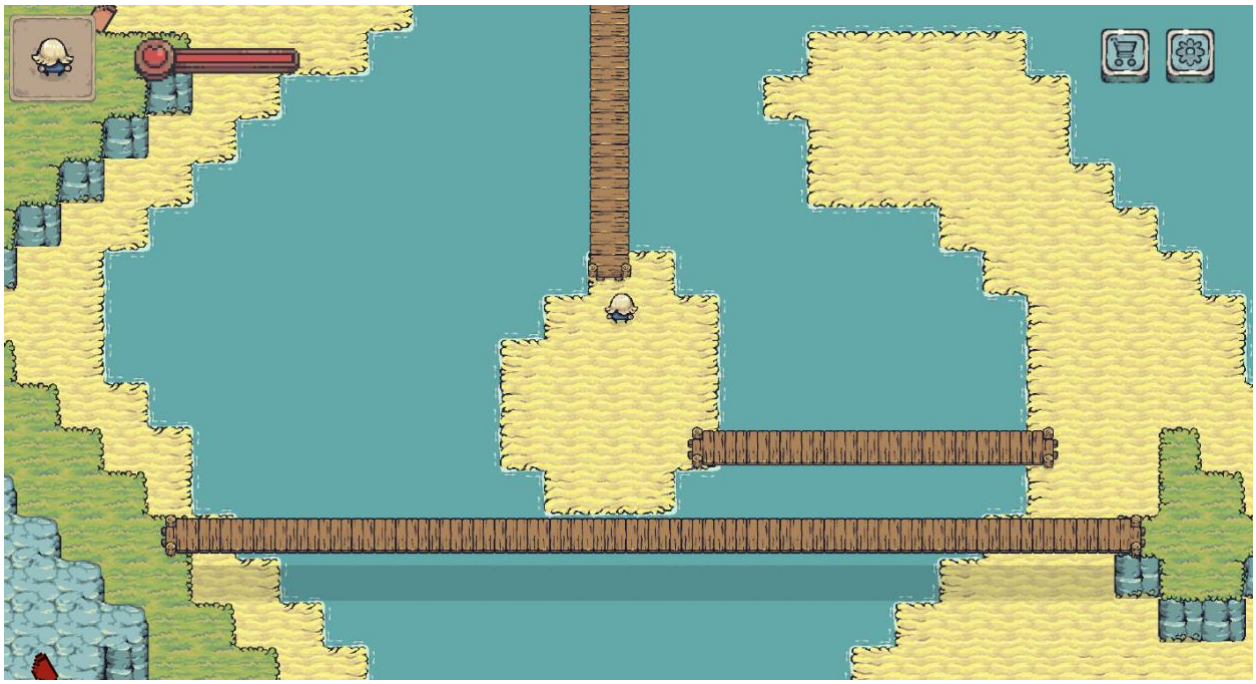


Рисунок 3.3 – Рух гравця

При натисненні кнопки зміни поверху на драбині гравець переміщується на інший поверх (рис. 3.4).

Рисунок 3.4 – Зміна поверху гравця

При натисненні клавіш зміни персонажів персонаж змінюється (рис. 3.5).

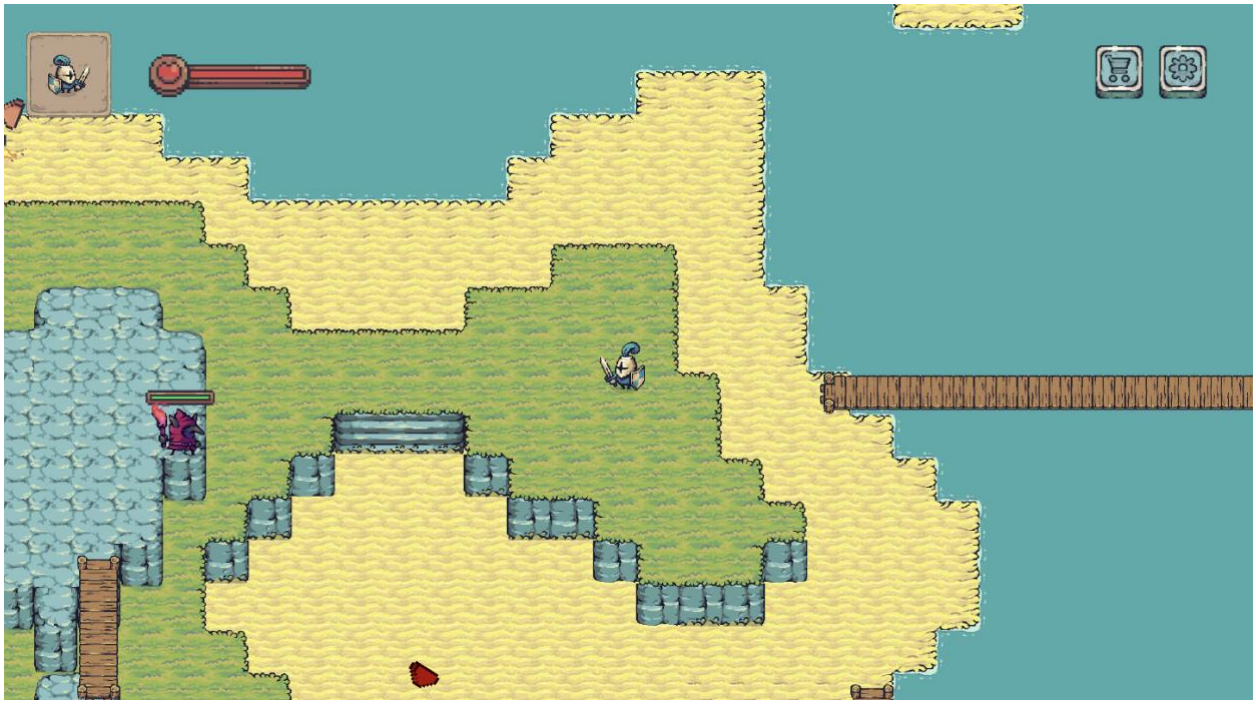


Рисунок 3.5 – Зміна персонажа

Якщо ворог знаходиться в діапазоні атаки то він отримує шкоду (рис. 3.6).

Рисунок 3.6 – Атака на ворога

Коли ворог помічає гравця, то він атакує і персонаж отримуємо шкоду (рис. 3.7).

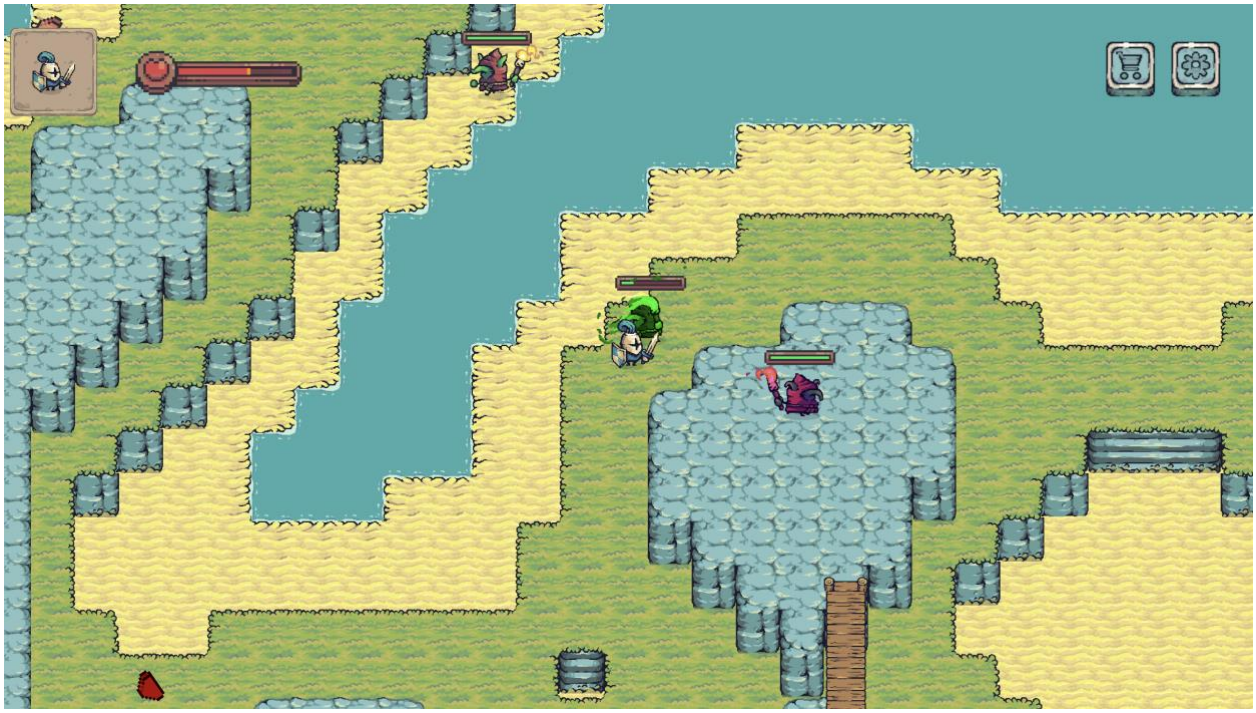


Рисунок 3.7 – Ворог атакує

Якщо до предмету підійти, то він автоматично перенесеться нам в інвентар (рис. 3.8).

Рисунок 3.8 – Піднімання предмету

При отриманні достатньої кількості шкоди гравець програє і відкривається меню програшу (рис. 3.9).

Рисунок 3.9 – Меню програшу

При нанесенні достатньої кількості шкоди, персонаж знищує ворога (рис. 3.10).

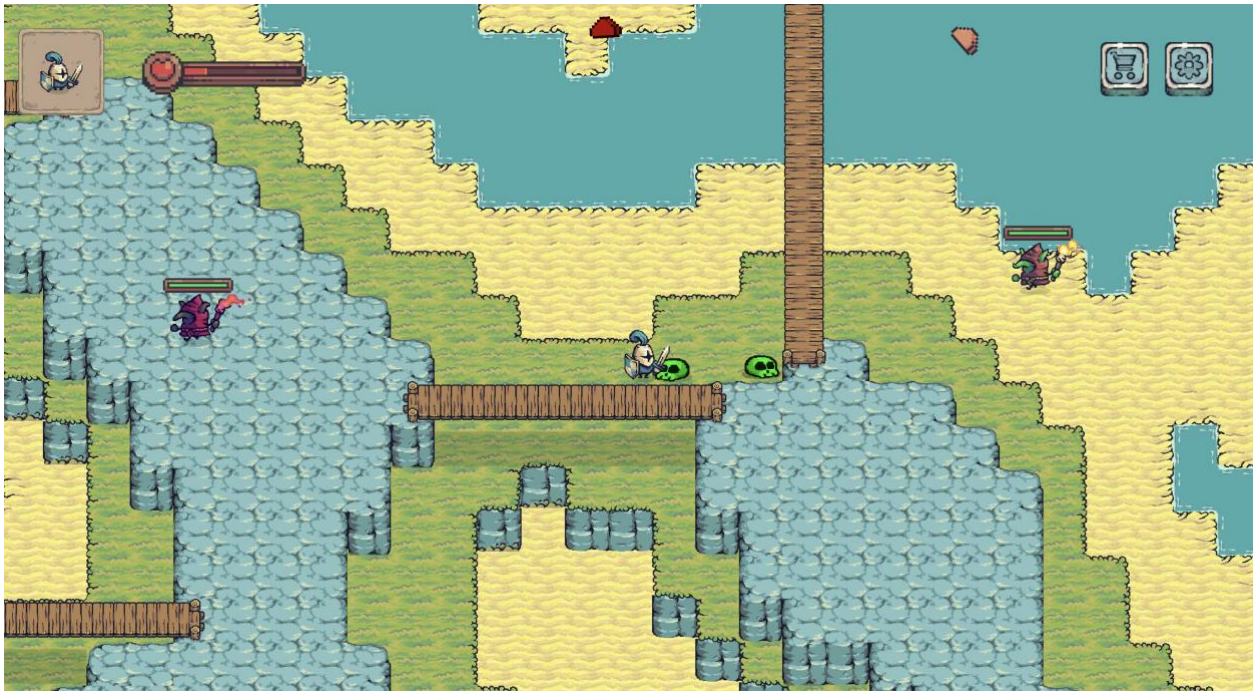


Рисунок 3.10 – Знищення ворога

Після перемоги над фінальним босом гравця переносить в меню перемоги (рис. 3.11).

Рисунок 3.11 – Меню перемоги

Також після натиснення кнопки інвентарю є можливість подивитись що персонаж має у інвентарі і які предмети мають які ефекти (рис. 3.12).

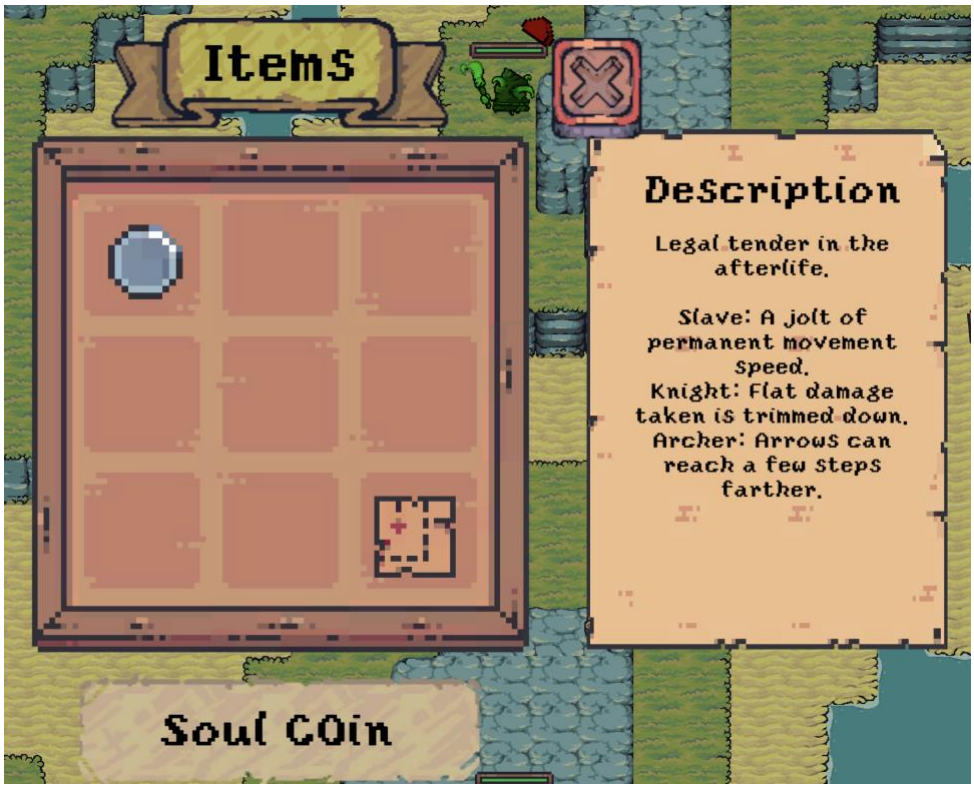


Рисунок 3.12 – Меню інвентаря гравця

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ROGUELIKE

Графічний матеріал

КПІ.ІП-1228.045480.06.99

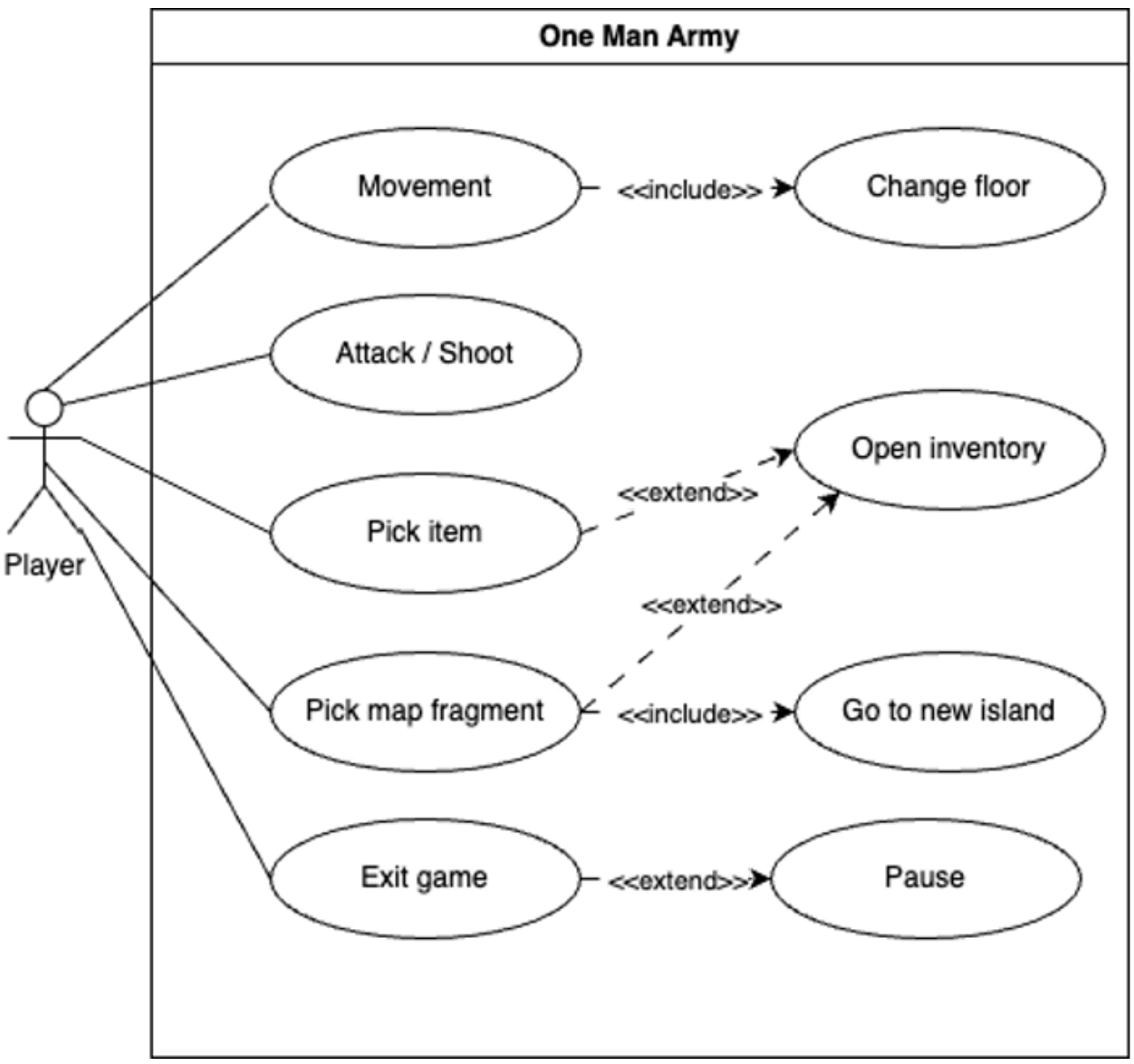
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____Максим ГОЛОВЧЕНКО

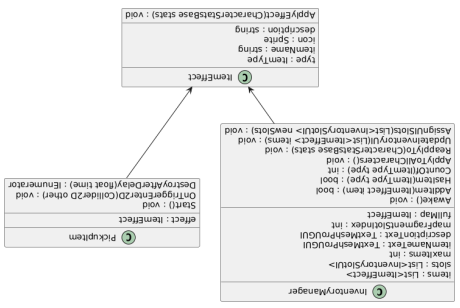
Виконавець:
_____ Андрій СІМЧУК

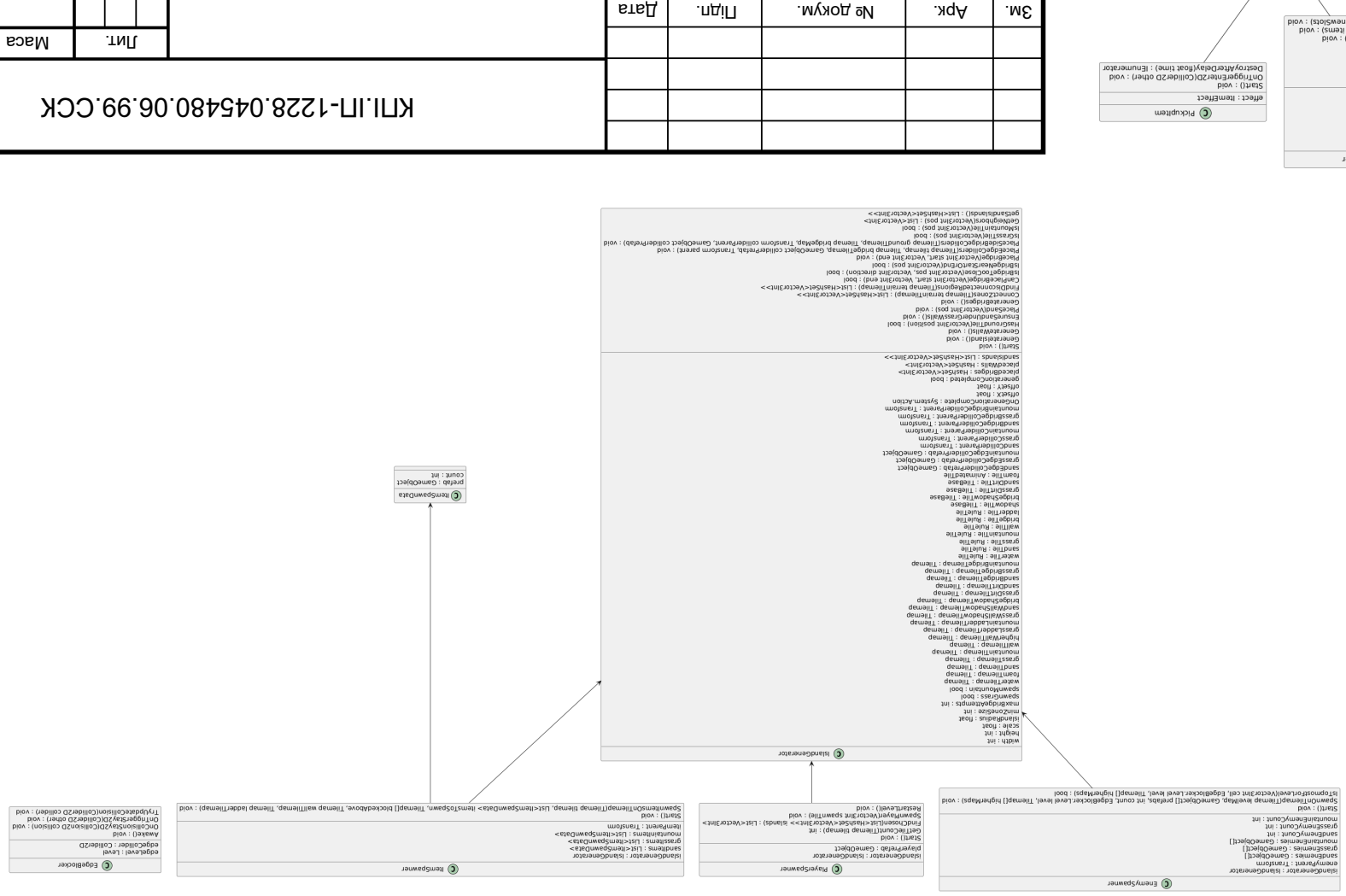
Київ – 2025

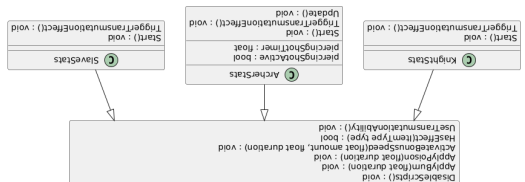
КПІ.ІП-1228.045480.06.99.ССВ

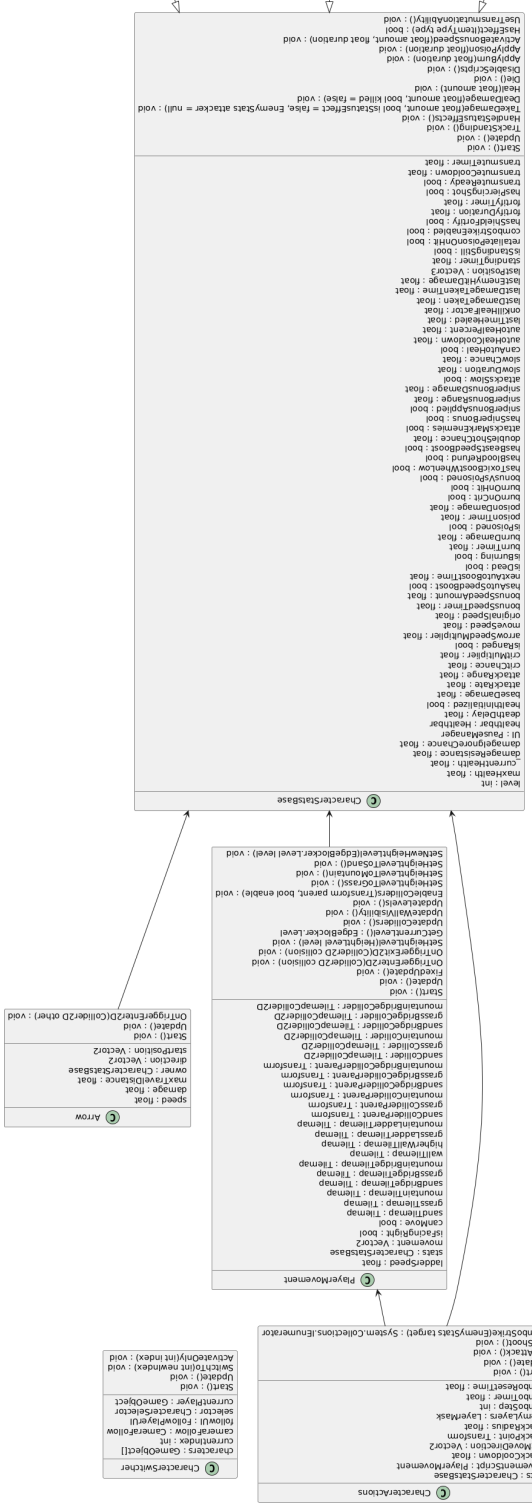


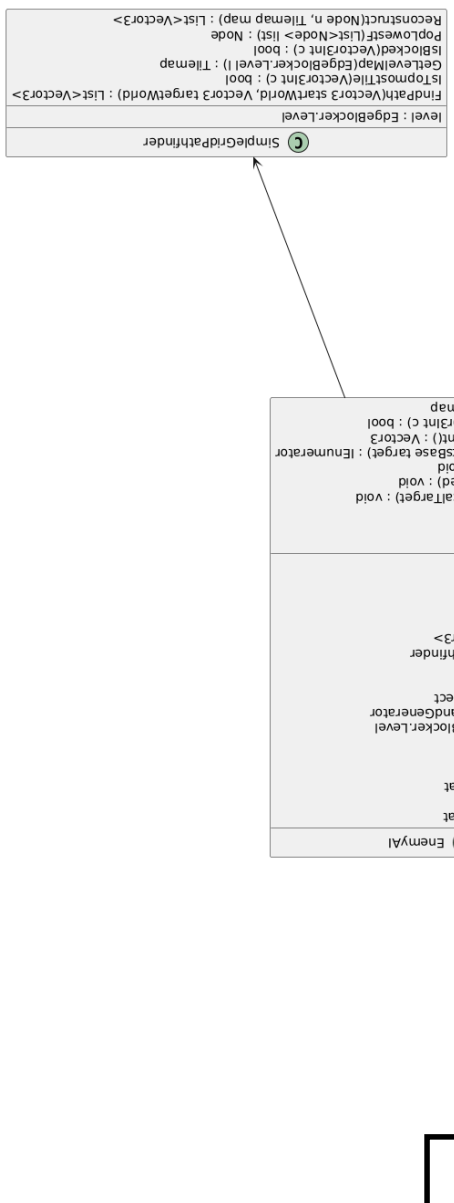
					КПІ.ІП-1228.045480.06.99.ССВ	
					Схема структурна варіантів використань	Лит. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підп.	Дата		
Розробив		Сімчук А.В.				Аркуш / Аркушів
Перевірив		Павлов О.А.				
Т. контр.						
					Ігровий застосунок у жанрі Roguelike	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12
Н. контр.		Головченко М.М.				
Затвердив		Жаріков Е.В.				



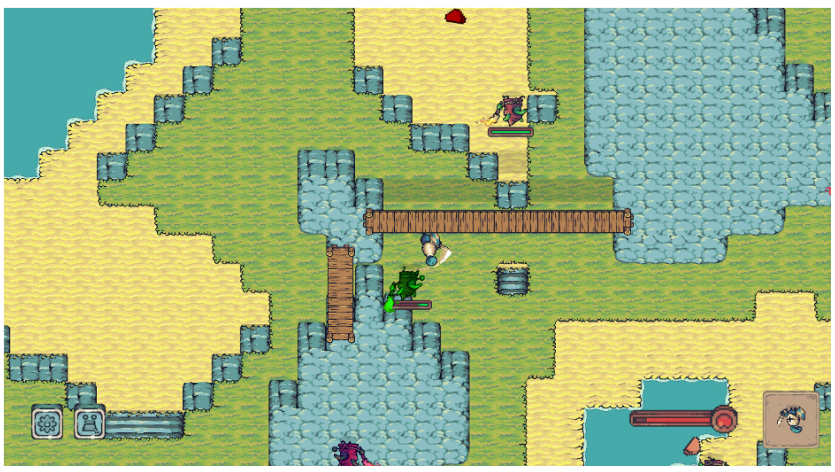


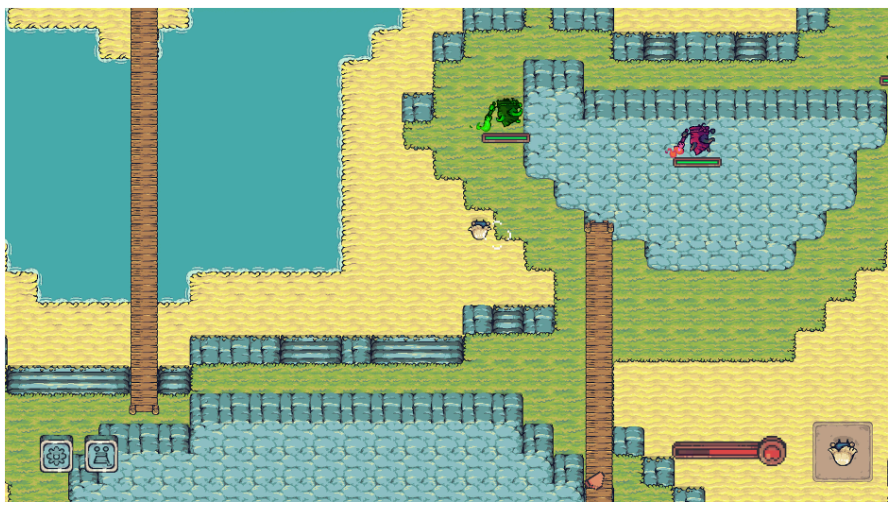






Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1228.045480.06.99.ССК
Розробив		Сімчук А.В.			
Перевірив		Лавров О.А.			
Т. контр.					Схема структура класів програмного забезпечення
Н. контр.		Голованенко М.М.			
Затвердив		Жарікова Е.В.			
Лист		Маса		Масштаб	
Арк.1					КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12





КПІ.ІП-1228.045480.06.99.КЕ

Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1228.045480.06.99.КЕ		
Розробив		Сімчук А.В.			Креслення вигляду екранних форм	Літ.	Маса	Масштаб
Перевірив		Лавров О.А.						
Т. контр.						Аркуш	Аркушів	
Н. контр.		Голованенко М.М.			Ігровий застосунок у жанрі Roguelike	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12	
Затвердив		Жарікова Е.В.					