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: Класична гра у жанрі платформер з розробкою редактора рівнів(комплексна тема)

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

Логвиненко Владислав Олексійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст.викл, д-р філософії, Дифучин А.Ю _____
(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 ас. кафедри, д-р філософії, Нікітін В. А. _____
(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

Логвиненку Владиславу Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Класична гра у жанрі платформер з розробкою редактора рівнів(комплексна тема)

керівник проєкту Дифучин Антон Юрійович, д-р філософії
(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) Передпроектне обстеження предметної області: постановка завдання дипломного проєктування, аналіз предметної області, аналіз існуючих технічних рішень, аналіз та моделювання бізнес-процесів.
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, розроблення функціональних вимог, розроблення нефункціональних вимог, аналіз системних вимог, аналіз економічних показників.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забеспечення, архітектурні рішення та обґрунтування вибору засобів розробки, конструювання програмного забезпечення, аналіз безпеки даних

4) Аналіз якості та тестування програмного забезпечення: аналіз якості програмного забезпечення, опис процесів тестування, опис контрольного прикладу.

5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Діаграми класів програмного забеспечення

3) Креслення вигляду екранних форм

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	21.04.2025	
3	Постановка та формалізація задачі	22.04.2025	
4	Розробка інформаційного забезпечення	23.04.2025	
5	Алгоритмізація задачі	28.04.2025	
6	Обґрунтування вибору використаних технічних засобів	02.05.2025	
7	Розробка програмного забезпечення	10.05.2025	
8	Налагодження програми	13.05.2025	
9	Виконання графічних документів	27.05.2025	
10	Оформлення пояснювальної записки	01.06.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Владислав ЛОГВИНЕНКО
(підпис) (ініціали, прізвище)

Керівник _____ Антон ДИФУЧИН
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 37 таблиць, 29 рисунків та 20 джерел – загалом 84 сторінок.

Дипломний проєкт присвячений розробці класичної гри у жанрі платформер з розробкою редактора рівнів.

Мета - розвиток логічного та вдосконалення стратегічного мислення гравця

У розділі «Предпроєктне обстеження предметної області» розглянуто постановку завдання дипломного проєктування, виконано аналіз предметної області, проведено огляд існуючих програмних та технічних рішень, а також змодельовано відповідні бізнес-процеси.

Розділ «Інформаційне забезпечення» присвячений визначенню варіантів використання програмного забезпечення, розробленню функціональних і нефункціональних вимог, аналізу системних та економічних показників, а також формулюванню завдання на розробку програмного забезпечення.

Розділ «Конструювання та розроблення програмного забезпечення» присвячений опису архітектури клієнтської та серверної частин, обґрунтуванню вибору технологічного стеку, реалізації ключових компонентів програмного забезпечення та аналізу безпеки даних.

Розділ «Аналіз якості та тестування програмного забезпечення» присвячений оцінці якості програмного продукту, опису проведених тестувань і контрольного прикладу.

Розділ «Розгортання та супровід програмного забезпечення» присвячений опису процесу розгортання програмного забезпечення та організації його супроводу після впровадження.

Програмне забезпечення впроваджено для персональних комп’ютерів на платформі Windows

КЛЮЧОВІ СЛОВА: КОМП’ЮТЕРНА ГРА, РЕДАКТОР РІВНІВ, ПЛАТФОРМЕР, ІГРОВИЙ ІНТЕРФЕЙС, ІГРОВИЙ ВСЕСВІТ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 37 tables, 29 figures and 20 sources – in total 84 pages.

The purpose of the diploma project is the development of a classic platformer game along with the creation of a level editor.

The goal is to enhance the player's logical and strategic thinking.

The section «Pre-project analysis of the subject area» presents the formulation of the diploma design task, an analysis of the subject area, a review of existing software and technical solutions, and the modeling of relevant business processes.

The section «Information support» focuses on defining software use cases, developing functional and non-functional requirements, analyzing system and economic indicators, and formulating the software development task.

The section «Design and development of software» is devoted to the architecture of both the client and server sides, the rationale for choosing the technology stack, the implementation of key software components, and the analysis of data security.

The section «Quality analysis and software testing» is dedicated to assessing the quality of the software product, describing the testing process, and presenting a validation example.

The section «Deployment and maintenance of software» describes the process of deploying the software and organizing its post-deployment maintenance.

The software has been deployed for personal computers on the Windows platform.

KEYWORDS: COMPUTER GAME, LEVEL EDITOR, PLATFORMER, GAME INTERFACE, GAME UNIVERSE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1217.045480.01.91	Технічне завдання	17	
3	А4	КПІ.ІП-1217.045480.02.81	Пояснювальна записка	84	
4	А4	КПІ.ІП-1217.045480.03.12	Текст програми	39	
5	А4	КПІ.ІП-1217.045480.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1217.045480.05.34	Керівництво користувача	13	
7	А3	КПІ.ІП-1217.045480.06.99	Графічний матеріал	3	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1217.045480.00.90		
Розроб.		Логвиненко В.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Дифучин А.Ю.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-12	
Н.контр.		Головченко М.М.					
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

Класична гра у жанрі платформер, з розробкою редактора рівнів

(комплексна тема)

Технічне завдання

КПІ.ІП-1217.045480.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Антон ДИФУЧИН

Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Владислав ЛОГВИНЕНКО

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Класична гра у жанрі платформер, з розробкою редактора рівнів (комплексна тема).

Галузь застосування:

Наведене технічне завдання поширюється на розробку комп'ютерної гри «Level King» [045480], котра використовується для використання у галузі розваг, потенційними користувачами якої є любителі комп’ютерних ігор.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки «Level King» 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 розваг і тренування стратегічного й логічного мислення користувача за рахунок проходження рівнів гри.

Метою розробки є сприяння розвитку логічного та стратегічного мислення гравця.

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, що містить кнопки із наступним функціоналом: перейти до меню готових рівнів, перейти до редактора рівнів, перейти до меню власних рівнів, перейти до меню обміну рівнями, вийти з гри (рис. 4.1);

Рисунок 4.1 – Головне меню

– меню готових рівнів, що містить кнопку для повернення до головного меню, та кнопки з готовими рівнями гри, при натисканні на які почнеться проходження обраного рівня(рис. 4.2);

Рисунок 4.2 – Меню готових рівнів

– меню власних рівнів, що містить комірки для власних рівнів користувача, що містять наступний функціонал: якщо натиснути лівою кнопкою миші на зайняту комірку - починається проходження обраного власного рівня; якщо натиснути правою кнопкою миші на зайняту комірку - рівень видалиться з комірки; якщо перехід до меню власних рівнів був здійснений з редактора рівнів, то натискання лівою кнопкою миші по вільній комірці призведе до збереження поточного власного, що був створений у редакторів рівнів; якщо перехід до меню власних рівнів був здійснений з редактора рівнів, то натискання лівою кнопкою миші по зайнятій комірці призведе до перезапису поточного власного рівня, що був створений у редакторів рівнів, замість старого власного рівня (рис. 4.3);

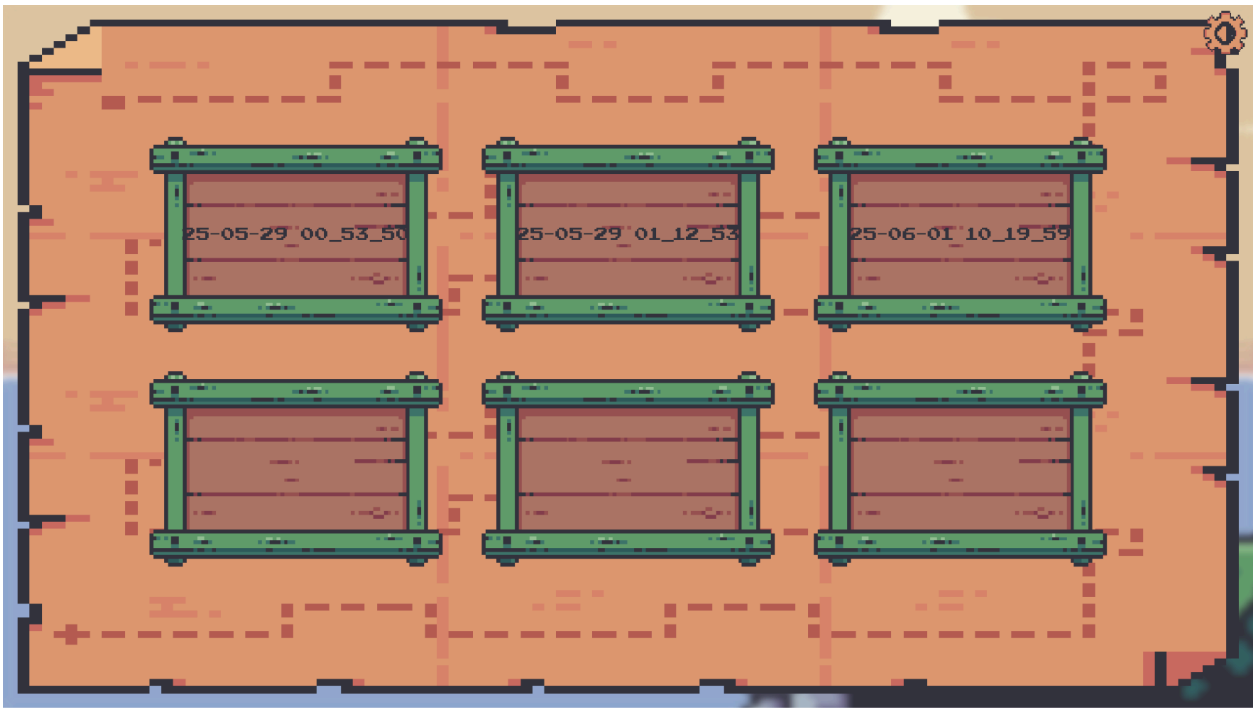


Рисунок 4.3 – Меню власних рівнів

– під час гри, ігровий інтерфейс, що містить: індикатор здоров’я , рахунок персонажа, кнопку паузи, кнопку повернення до останньої сцени (рис. 4.4);

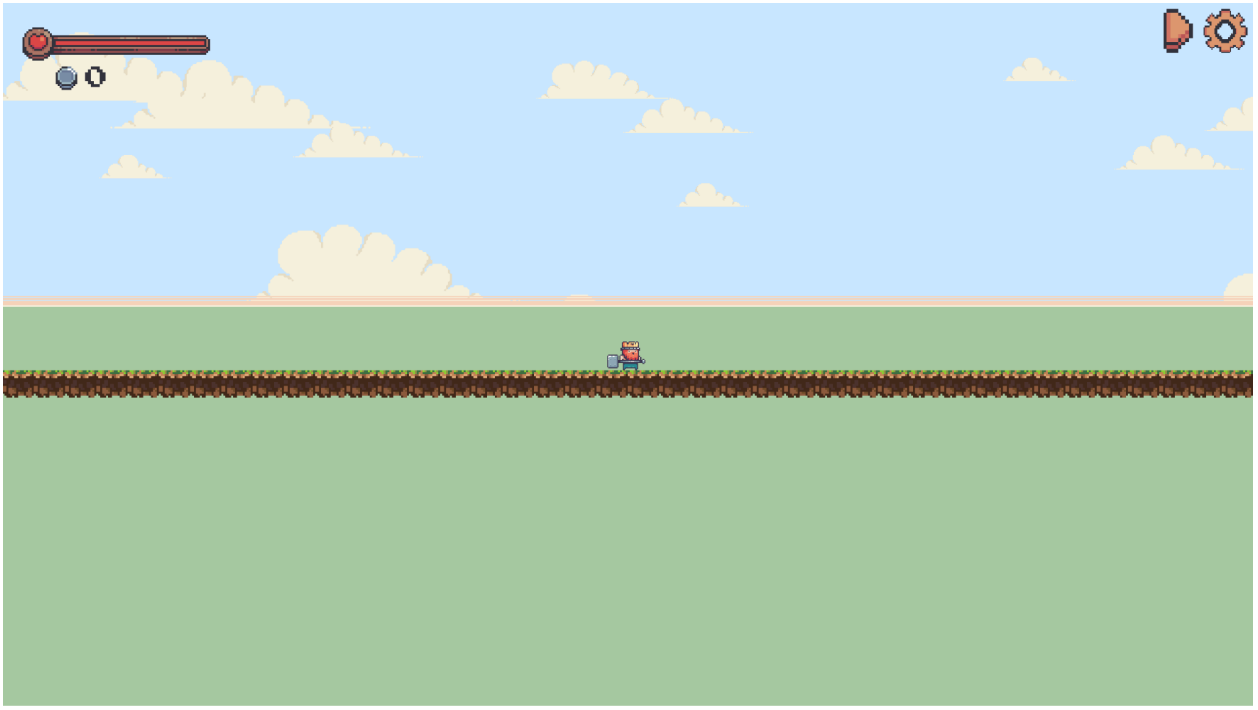


Рисунок 4.4 –Ігровий інтерфейс

– під час гри, меню паузи, при визові якого гра призупиняється і що містить наступний функціонал: поточний прогрес персонажа по рівню,

поточна кількість очок персонажа, та містить кнопки перезапуску рівня та продовження ігрового процесу(рис. 4.5);

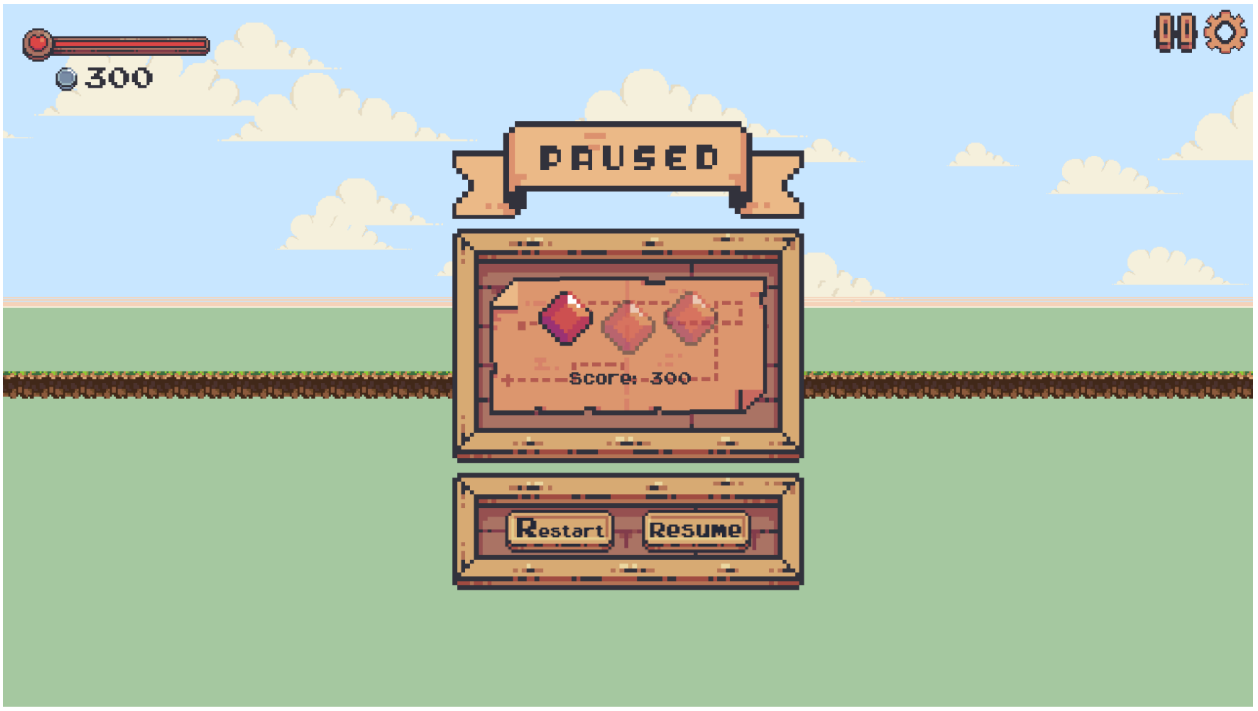


Рисунок 4.5 –Меню паузи

– меню програшу, що викликається після смерті персонажа, що містить наступний функціонал: поточний прогрес персонажа по рівню, поточна кількість очок персонажа, та містить кнопки перезапуску рівня та повернення до меню готових рівнів(рис. 4.6);

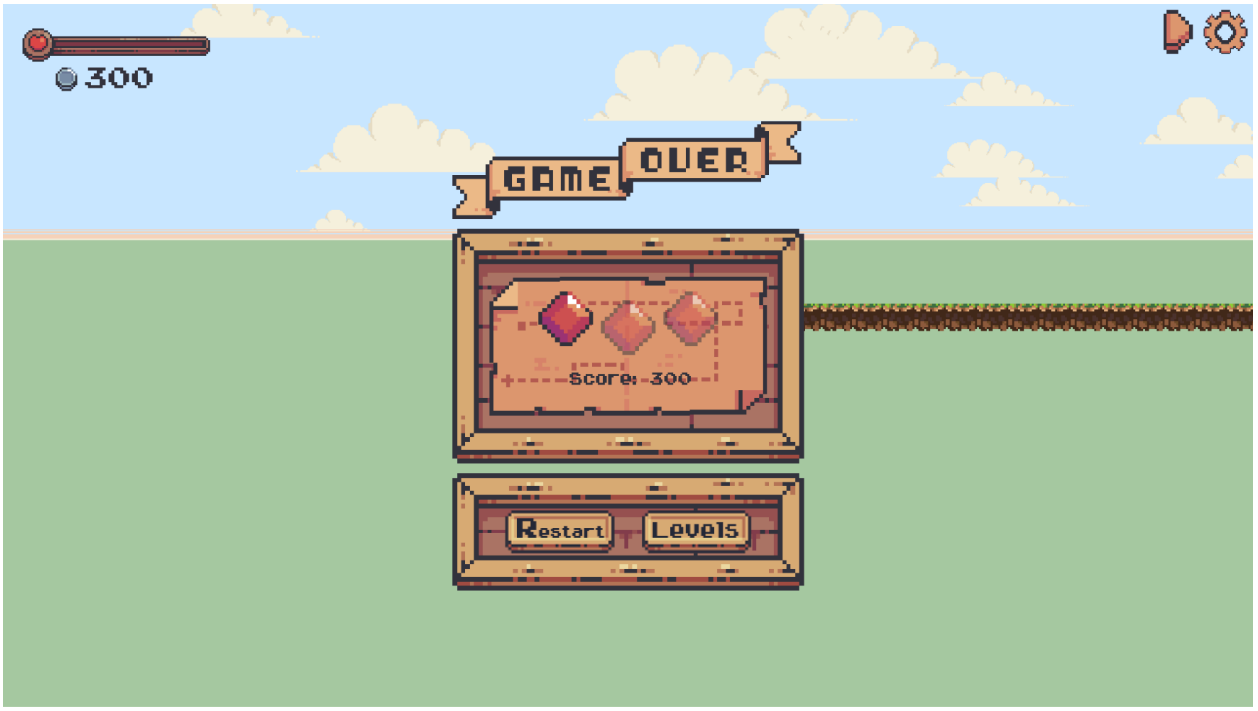


Рисунок 4.6 –Меню програшу

– меню перемоги, що викликається після виконання умов перемоги персонажем, що містить наступний функціонал: поточний прогрес персонажа по рівню, поточна кількість очок персонажа, та містить кнопки перезапуску рівня та повернення до меню готових рівнів(рис. 4.7);

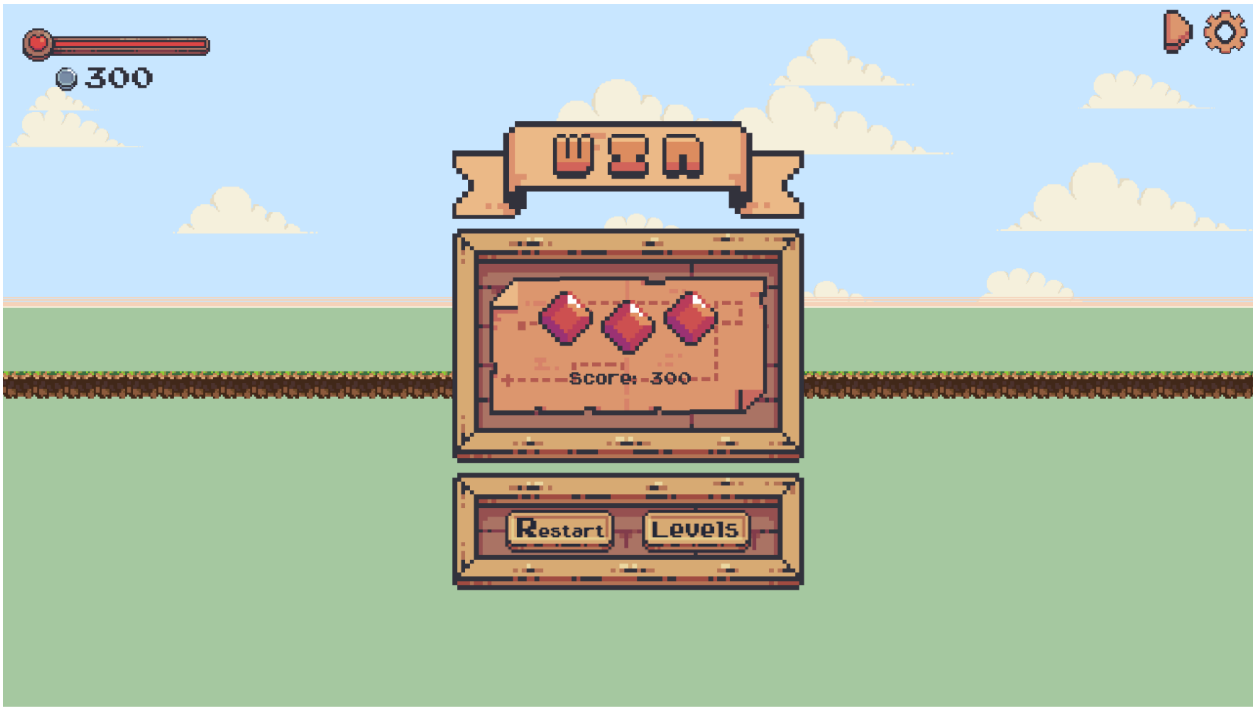


Рисунок 4.7 –Меню перемоги

4.1.2 Для користувача

4.1.2.1 Керування персонажем

– функція переміщення персонажа;

– функція наявності системи здоров’я персонажа;

– функція смерті персонаж;

– функція атаки персонажа;

4.1.2.2 Зміна очок персонажа

– функція взаємодії з об’єктами очок;

4.1.2.3 Проходження рівня

– функція взаємодії об’єктами що потрібні для проходження рівня

– функція паузи;

4.1.2.4 Вороги

– функція статичних ворогів;
– функція динамічних ворогів;
– функція пасток;
– функція вбивства ворогів;

4.1.2.5 Інтерфейс

– функція ігрового інтерфейсу;
– функція головного меню;
– функція виходу з гри;
– функція меню готових рівнів;
– функція меню власних рівнів;
– функція комірок в меню власних рівнів;
– функція меню паузи;
– функція меню перемоги/програшу;
– функція меню обміну рівнями;

4.1.2.6 Збереження власних рівнів

– функція запису власних рівнів;
– функція видалення власних рівнів;
– функція перезапису власних рівнів;

4.1.2.7 Сканування файлу рівня на шкідливий вміст.

– функція перевірки файлу рівня на шкідливий змість перед завантаженням на сервер;

4.1.3 Додаткові вимоги

– наявність готових рівнів гри, що ознайомлять користувача з базовими правилами гри;
– інтерфейс на англійській мові;
– зберігання назви власних рівнів відповідно до дати і часу їх створення;

– перехід між компонентами меню повинен бути виконаний через анімацію.

– музика та відповідні звукові ефекти під час гри

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити перевірку файлу рівня на шкідливий вміст перед завантаженням на сервер для обміну рівнями.

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 функціонуватиме на пристроях, що працюють на ОС Windows 10.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3;

– об‘єм ОЗП: 4 Гб;

– об’єм вільної пам’яті накопичувача: 300 Мб.

– швидкість інтернету 5 мб/с

Рекомендована конфігурація технічних засобів :

– тип процесору: Intel Core i5;

– об‘єм ОЗП: 16 Гб;

– об’єм вільної пам’яті накопичувача: 300 Мб.

– швидкість інтернету 10 мб/с

Мінімальні системні вимоги відповідають тим пристроям, котрі здатні підтримувати дану версію ОС Windows.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем Windows версії 10 або пізніше.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: вхіднми даними є натиски клавіш на клавіатурі та зміна позиції і натиски клавіш комп’ютерної миші а також файли формату JSON, зі структурою власного рівня у середині.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: файли формату JSON, зі структурою власного рівня у середині.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Python.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Pygame у середовищі PyCharm.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію на 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 варіантів використання;
– архітектура програмного забезпечення;
– схема структурна класів програмного забезпечення;
– креслення вигляду екранних форм;

5.2 Спеціальні вимоги до програмної документації

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

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	15.03	
2.	Розробка технічного завдання	19.04	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	21.04	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	23.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	10.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	13.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	19.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проєкту	21.05	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	01.06	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

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

Пояснювальна записка
до дипломного проєкту

на тему: **Класична гра у жанрі платформер, з розробкою редактора рівнів (комплексна тема)**

КПІ.ІП-1217.045480.02.81

Київ – 2025

ЗМІСТ

Перелік умовних позначень

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс.
ОС	–	Операційна система.
ПК	–	Персональний комп’ютер.
ЛКМ	–	Ліва кнопка миші
ПКМ	–	Права кнопка миші

Вступ

На сьогодні, технології тісно пов’язані з кожним аспектом нашого життя. Це значно спростило повсякденність, але в той же час має свої негативні наслідки. Люди у значно більшій степені почали відчувати інформаційну перенавантаженість, стрес та соціальну ізоляцію. В цій ситуації на допомогу приходять ігри, які виступають як технологічний аналог медитації та морального розвантаження. Вони не тільки допомагають послабити негативні наслідки діджиталізації, але й сприяють розвитку інтелектуальних здібностей гравця.

Внаслідок, популярність ігрової індустрії зростає і досі. Люди шукають можливості грати усюди: як вдома, використовуючи ПК та приставки, так і на вулиці завантаживши ігри на телефон. Кожна з цих платформ має як свої переваги, так і недоліки. Ігри на телефоні дозволяють насолодитися ігровим процесом майже у будь-якому місці, але в той же час технічно обмежені через маленьку потужність пристрою і банальний розмір дисплею. В свою чергу, ігри на приставках мають свої ексклюзивні способи взаємодії з гравцем, через що вимагають потужного обладнання і багато коштують. Саме тому у пошуках найякіснішого геймплейного досвіду, більшість людей звертається до комп’ютерних ігор, що зберігають баланс між технічними можливостями та ціною.

Крім різноманіття у платформах, ігри мають величезну кількість жанрів, що дозволяють користувачам знайти те що їм подобається найбільше. Одним із найпопулярніших з них є Platformer. Ігри цього жанру вирізняються захоплюючим геймплеєм і водночас високим рівнем виклику до стратегічних навичок гравця. Також, варто зазначити що зазвичай ці ігри є дуже компактними у технічному плані, що дозволяє грати у них навіть на слабких ПК.

Знайшови гарну гру, зазвичай, пристрасні геймери проходять її за пару десятків годин, після чого їм залишається лише мріяти про її наступну частину. Тут у нагоді стає редактор рівнів. Він дозволяє гравцю не лише

проходити базові рівні гри заготовлені розробником, а і самостійно створювати цікаві геймплейні рішення, тим самим отримуючи нові емоції від вже пройденої гри. Хоча проходити свої власноруч стоврені рівні захопливо, набагато цікавішим буде проходити рівні створені іншими гравцями, а також ділитися своїми власними напрацюваннями та рішеннями. Це сприятиме розвитку спільноти гри, підтримуючи інтерес до неї.

В рамках цієї дипломної роботи буде розглянуто створення однокористувацької гри на ПК у жанрі Platformer , з розробкою редактора рівнів та надання користувачу можливості обмінюватися рівнями. Метою даної роботи є сприяння розвитку стратегічних та логічних навичок гравця за рахунок проходження та побудови рівнів гри, а також розвиток соціалізації завдяки поширенню власних рівнів та проходженню рівнів інших людей.

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 обраного завдання розробки у рамках ДП;
- аналіз та моделювання бізнес-процесів;
- розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;
- аналіз економічних показників програмного забезпечення ДП;
- постановка завдання на розробку програмного забезпечення ДП;
- розроблення архітектури програмного забезпечення;
- розроблення архітектурних рішень та обґрунтування вибору засобів розробки програмного забезпечення;
- конструювання та розроблення програмного забезпечення;
- аналіз безпеки даних програмного забезпечення;
- аналіз якості та тестування програмного забезпечення;
- розгортання та супровід програмного забезпечення;
- створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

Комп’ютерний застосунок - це програмне забезпечення, що встановлюється і використовується на ПК з метою виконання конкретних завдань і надання користувачу певного арсеналу функціональних можливостей [1]. На відміну від програмного забезпечення, яке підтримує роботу комп’ютера загалом, застосунки призначені для безпосередньої взаємодії з користувачем і можуть включати текстові редактори, графічні редактори, браузери, ігри, тощо.

Комп’ютерна гра - це різновид комп’ютерного застосунку, призначенням якого є розвага користувача[2]. Залежно від жанру, комп’ютерна гра також може бути спрямована на розвиток корисних навичок у користувача(реакцію, логіку, тощо) або мати повчальний характер. Комп’ютерна гра базується на активній взаємодії з гравцем, який керує процесом за допомогою пристроїв введення, таких як клавіатура, миша чи геймпад.

Жанр Platformer - є одним із найстаріших і найвідоміших жанрів комп’ютерних ігор для геймерів. Його особливість полягає у взаємодії гравця з рухомими платформами шляхом стрибків, бігу або лазіння. Головним завданням гравця є подолання перешкод та пасток на шляху до кінця рівня. Жанр набув своєї популярності у 1980-х роках із виходом таких іконічних ігор, як Donkey Kong та Super Mario Bros., які заклали основи його механік[3]. Відтоді платформери постійно видозмінюються, але зберігають свою основну суть - динамічний та захоплюючий ігровий процес, що вимагає точності та реакції від гравця.

Редактор рівнів - це додатковий або вбудований інструмент у складі відеоігор або ігрових рушіїв. Він дозволяє користувачу створювати свої власні ігрові рішення обмежені функціоналом початкової гри. Як правило, редактори дозволяють вставляти об’єкти, визначаючи поведінку елементів і змінювати умови перемоги чи поразки, не вимагаючи навичок програмування. Такі можливості позитивно впливають на реіграбельність і на розвиток суспільства гравців цієї гри.

Windows - це одна з найвідоміших операційних систем, розроблена компанією Microsoft та випущена в 1985 році під назвою Windows 1.0 для ПК[4]. Вона забезпечує інтерфейс між користувачем і апаратною частиною комп’ютера, дозволяючи запускати програми, працювати з різноманітними файлами, переглядати мультимедіа й взаємодіяти з інтернетом. Наразі, Windows є однією з найпопулярніших ОС для ігрових ПК.

Варто зазначити, що ігрова індустрія демонструє не лише стабільне зростання, а й гарні перспективи на майбутнє. Завдяки стрімкому розвитку технологій, зі звичайного хобі ,комп'ютерні ігри перетворюються на універсальний формат розваги та пізнання навколишнього світу. Враховуючи це очікується ,що станом на 2027 рік прибуток індустрії комп’ютерних ігор сягне 298.2 мільярдів доларів[5]. Такі позитивні передбачення базуються на тому факті, що у 2023 році майже кожна третя людина у світі є гравцем. Лише за один рік, з 2023 по 2024, до спільноти геймерів приєдналося понад 100 мільйонів людей.

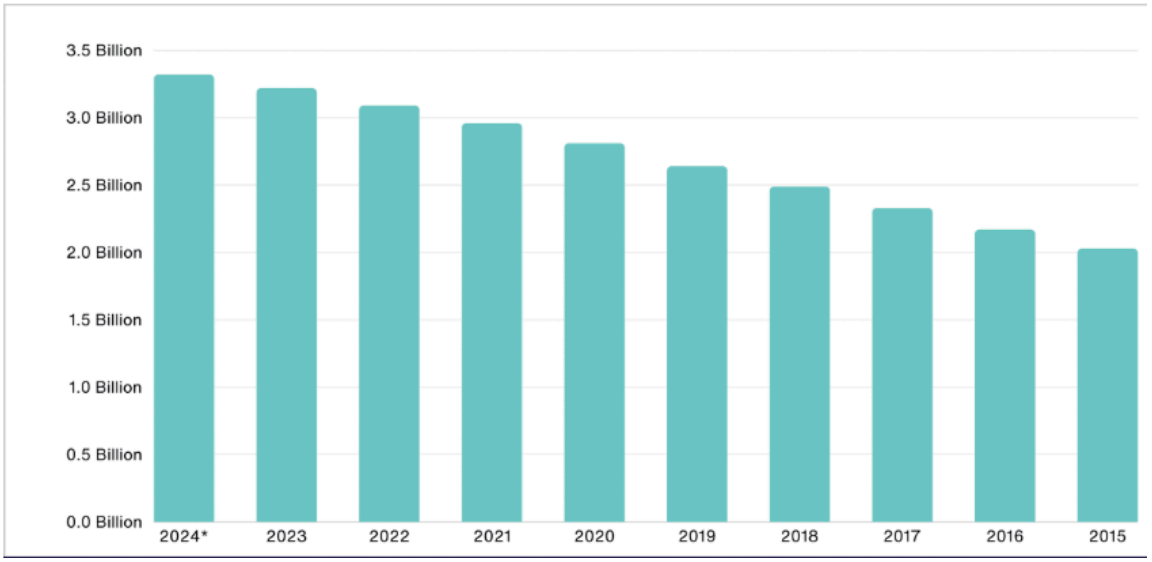


Рисунок 1.1 – Графік відношення кількості гравців до року[5]

На жаль, у цього успіху є і свої мінуси. Ігри починають вимагати все більшого фінансового залучення користувача. Гра може початково коштувати велику суму, або навіть позиціонувати себе як безкоштовна , але при цьому мати платні додатки з основною частиною вмісту. По справжньому безкоштовними зачасту виявляються ігри що перебувають на тестуванні або просто є прототипами.

Саме тому, задача цієї дипломної роботи полягає в створені безкоштовного комп’ютерного ігрового застосунку у жанрі Platformer на основі ОС Windows. Також передбачається розробка редактора рівнів і системи для обміну рівнями між гравцями задля зацікавленості аудиторії.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації комп'ютерної гри у жанрі Platformer , з розробкою редактора рівнів. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

Geometry Dash лаконічно поєднує в собі 2D-платформер і ритм-гру. Вона була розроблена студією RobTop Games у 2013 році[6]. Першопочатково, вона була доступна тільки на мобільних пристроях iOS/Android, але пізніше гру додали на ПК. Гра має на перший погляд просту систему управління, але після декількох рівнів стає зрозумілим чому ця гра вимагає підвищену концентрацію і гарне відчуття такту. Геймплей полягає у тому що гравець керує геометричною фігурою (зазвичай кубом), яка автоматично рухається вперед. Його завданням є вчасно стрибати та уникати перешкод у ритмі енергійної музики, що є на кожному рівні. Вона містить як базові рівні підготовлені розробником, так і зручний редактор рівнів для створення власних рівнів.

Geometry Dash отримала численні позитивні відгуки за свою динамічність, стиль і високу реіграбельність.

Рисунок 1.2 – Вигляд гри Geometry Dash[6]

Super Meat Boy - продукт студії Team Meat що вийшов у світ у 2010 році для ПК, а згодом і на інші платформи[7]. Це платформер, у якому гравець керує маленьким кубиком м’яса на ім’я «Міт Бой», що намагається врятувати свою подругу «Бендидж Герл» від злого доктора «Фетуса». Гра запам’ятовується через свою складність та миттєвими перезапусками після чергової смерті, що дозволяє зберегти швидкий темп гри попри численні невдачі. Гравець поступово звикає до фізики гри під час самого геймплею, через що неминуче стає краще грати. В свою чергу, гра пропонує безліч варіацій пасток. За ітогом, кожен рівень перетворюється на випробування точності та реакції, де одна помилка може коштувати життя персонажа. Для ПК-версії гри був доступний вбудований редактор рівнів, однак можливості передачі рівнів між гравцями не було передбачено.

Незважаючи на це, Super Meat Boy здобув культовий статус серед геймерів завдяки своєму динамічному геймплею, ретро-стилістиці та точному управлінню.

Рисунок 1.3 – Вигляд гри Super Meat Boy[7]

Гра Hollow Knight була розроблена незалежною студією Team Cherry та видана у 2017 році для платформи ПК[8]. На відміну від минулих проектів, ця гра має більш глибоку історію за якою цікаво спостерігати. Гравець, у ролі безіменного мандрівника, опиняється у підземному королівстві Галонест де він бореться з ворогами поступово досліджуючи нові території та таємниці ігрового світу. На своєму шляху гравцю доведеться посилювати свого персонажа вивчаючи нові здібності задля перемоги над більш сильними ворогами та босами. Гра не має редактора рівнів, натомість зосереджена на більш якісному авторському дизайні світу.

Hollow Knight отримала численні позитивні відгуки за художній стиль, геймплейну глибину та музичний супровід, і стала однією з найуспішніших інді-ігор свого покоління.

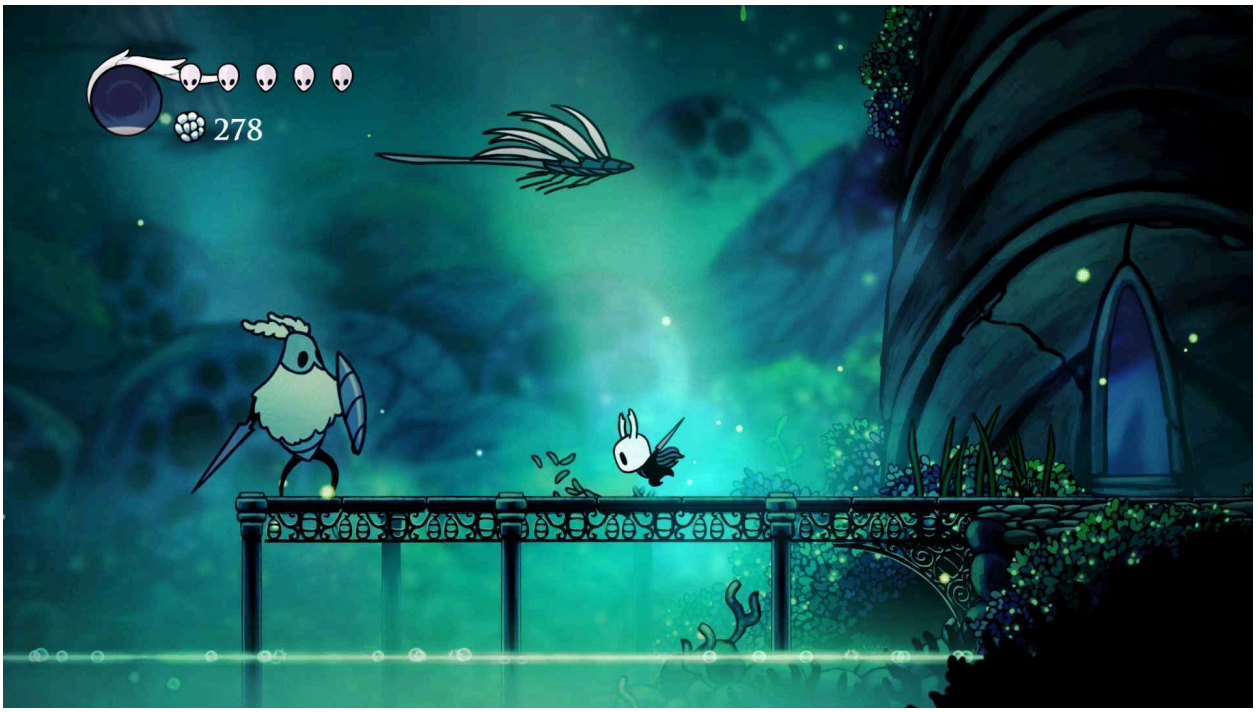


Рисунок 1.4 – Вигляд гри Hollow Knight[8]

Для порівняння проєкту з аналогом можна скористатись таблицею 1.3.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	Дипломний проєкт	Geometry Dash	Super Meat Boy	Hollow Knight	Пояснення
Безкоштовність	+	-	-	-	Можливість безкоштовно пограти у гру
Редактор рівнів	+	+	+	-	Наявність редактора рівнів у самій грі
Передача рівнів	+	+	-	-	Наявність системи обміну рівнями між гравцями

Продовження таблиці 1.1

Функціонал	Дипломний проєкт	Geometry Dash	Super Meat Boy	Hollow Knight	Пояснення
Вороги	+	-	-	+	Наявність ворогів на рівнях
Пастки	+	+	+	+	Наявність пасток на рівнях
Система здоров’я	+	-	-	+	Можливість втрачати\поповнювати здоров’я персонажа
Система очок	+	-	-	-	Наявність системи очок що відображає вдалість пройденого рівня
Прокачка персонажа	-	-	-	+	Можливість посилювати головного персонажа
Прогрес у відсотках	-	+	-	-	Наявність системи що відображає прогрес на рівні, навіть якщо він не був закінчений

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

Одним з важливих моментів у процесі створення гри є вибір рушія. Саме він забезпечує функціонування усіх ключових процесів: виведення зображення і звуку, обробка дій гравця, логіку поведінки об’єктів і їх фізичні властивості. Зазвичай, вибір рушія залежить саме від цілей проекту та технологічних вимог до нього. У сучасному ігровому світі існує велике кількість актуальних рушіїв, кожен з яких має свої переваги та особливості. Ігровий рушій Pygame[9] - є бібліотекою для мови програмування Python[10], що призначення для створення 2D-ігор. Він надає розробнику безпосередній контроль над усіма частинами ігрового циклу - графічним рендерингом, обробкою введення, подій, анімацій та звуку. Все це відбувається без зайвих абстракцій або жорсткої структури, як це зачасту буває у складних рушіях. На відміну від великовагових рішень по типу Unity[11] або Unreal Engine[12], Pygame не нав'язує готову архітектуру чи робочий конвеєр, що дозволяє гнучко будувати власну логіку проєкту «з нуля». Завдяки використанню Pygame у зв’язці з Django (на Python), стало можливим реалізувати систему обміну рівнями між користувачами без залучення додаткових важковагових технологій або сторонніх бекенд-сервісів. Усі компоненти - і гра, і сервер, і мережеві запити - реалізовані на єдиній мові (Python), що значно спрощує підтримку та інтеграцію. Такий підхід дозволив легко реалізувати завантаження рівнів на сервер, пропозиції змін, затвердження, автентифікацію, без потреби створювати окремі API-шари, що часто необхідно при використанні рушіїв типу Unity або Unreal Engine.

Крім того, в межах цієї архітектури реалізована класична модель розділення клієнта і сервера, де Pygame-гра виступає у ролі клієнта, а серверна частина, побудована на Django + Django REST Framework, слугує серверною частиною, що містить базу даних і API. Через використання

REST-інтерфейсу й передачу даних у форматі JSON, взаємодія між компонентами системи є зрозумілою, прозорою та легко масштабованою.

Наступним важливим етапом розробки був вибір IDE. Так як основною мовою програмування цього проекту є Python, було розглянуто два найпопулярніші інструменти для цієї мови - PyCharm[13] та Visual Studio Code[14]. VS Code - це гнучкий безкоштовний редактор що підтримує Python через офіційне розширення. Це середовище має велику та активну спільноту, яка постійно вдосконалює цей продукт у вигляді корисних плагінів та розширень, що дозволяють налаштовувати IDE під конкретні потреби.
Натомість, PyCharm - це повноцінне інтегроване середовище розробки, яке одразу містить усе необхідне для професійної роботи з Python і не потребує такої кількості додаткових модулів. Середовище включає у себе: інтелектуальне автодоповнення, вбудований дебагер, генерацію діаграм, систему керування залежностями, а також зручну інтеграцію з системами контролю версій. Окремою перевагою цього IDE в контексті цього комплексного диплому є підтримка функції Code With Me[15] - інструменту для спільної розробки, що дає змогу кільком розробникам паралельно працювати над одним проєктом, редагувати код у реальному часі, запускати налагодження та обговорювати зміни без потреби використовувати сторонні сервіси. Зважаючи на такі переваги, вибір було зроблено на користь PyCharm як найбільш збалансованого та ефективного середовища розробки для цього проекту.

Другорядною задачею є розробка власної графіки для гри. Подібний запит можна реалізувати завдяки редакторам 2D-графіки. Програми цього спектру дозволяють попіксельно малювати власні спрайти для внутрішньо ігрових об’єктів. В межах цього проєкту було використано два популярні редактори - Aseprite[16] та Tilesetter[17]. Aseprite є професійним інструментом, що підтримує роботу з шарами, таймлайном, прозорістю та має зручну систему попереднього перегляду анімації, через що він ідеально підходить для малювання персонажів та об’єктів. В свою чергу Tilesetter

здебільшого слугує для розробки плиткових текстур(платформ). Його головною перевагою є автоматичне генерування варіацій текстур за допомогою алгоритмів, що значно пришвидшує та спрощує роботу над графікою.

1.4 Аналіз та моделювання бізнес-процесів

Для моделювання бізнес-процесу використовується BPMN модель (рисунок 1.5) (рисунок слугує ілюстрацією BPMN моделі).

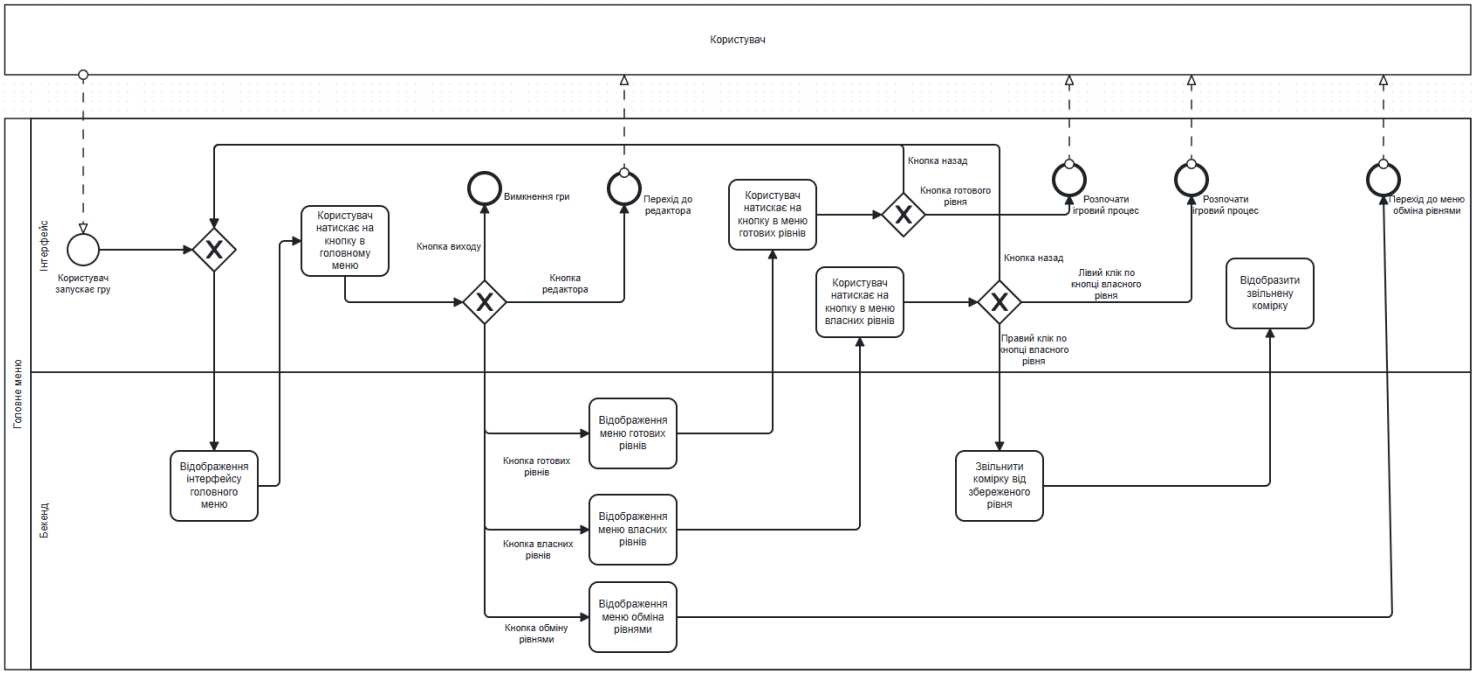


Рисунок 1.5 – Схема послідовного користування головного меню

Опис моделі бізнес-процесу послідовного користування головного меню:

– користувач запускає гру;

– бекенд відображає головне меню;

– користувач натискає на кнопку в головному меню: якщо це кнопка вихід, то гра закривається; якщо це кнопка редактора, то користувач потрапляє до редактора рівнів; якщо це кнопка готових рівнів, то користувач переходить до меню готових рівнів ;якщо це кнопка власних рівнів, то користувач переходить до меню власних рівнів ;якщо це кнопка меню обміна рівнями, то користувач переходить до меню обміна рівнями;

– потрапивши в меню готових рівнів, користувач натискає на кнопку в цьому меню: якщо це кнопка назад, то користувач переходить до головного меню; якщо це кнопка одного з рівнів, то користувач починає проходити відповідний готовий рівень;

– потрапивши в меню власних рівнів, користувач натискає на кнопку в цьому меню: якщо це кнопка назад, то користувач переходить до головного меню; якщо натискання відбувається лівою кнопкою миші по кнопці одного з рівнів, то користувач починає проходити відповідний власний рівень ;якщо натискання відбувається правою кнопкою миші по кнопці одного з рівнів, то користувач видаляє відповідний власний рівень і комірка очищується.

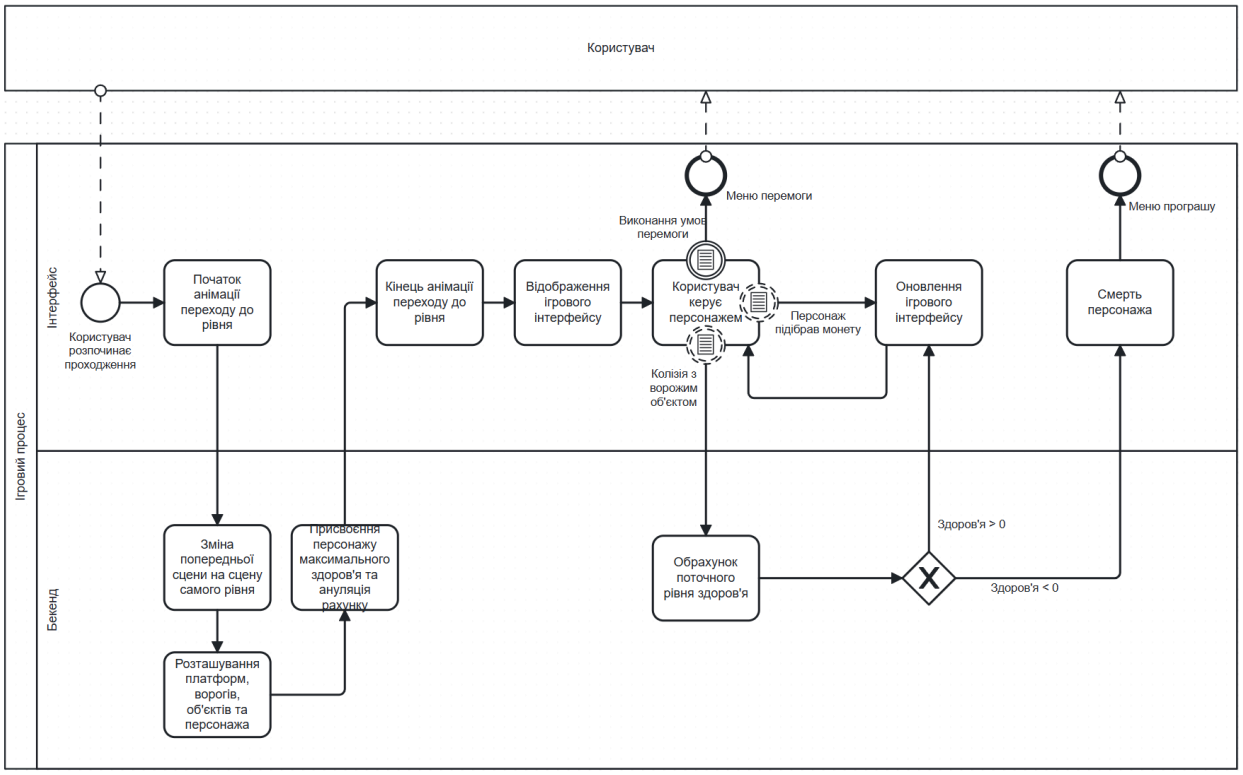


Рисунок 1.6 – Схема ігрового процесу

Опис моделі ігрового процесу:

– користувач розпочинає проходження рівня(з меню власних або готових рівнів);

– починається анімація переходу між сценами;

- бекенд підгружає рівень;
- закінчується анімація переходу між сценами;
- користувач бачить ігровий інтерфейс і отримує можливість керувати персонажем, проходження рівня розпочинається;
- під час час проходження рівня користувач може виконати декілька умов: підібрати монету, через що ігрова статистика оновиться і користувач побачить це на ігровому інтерфейсі; зіштовхнутися з ворогом, через що здоров’я персонажа зменшиться і користувач побачить це на ігровому інтерфейсі;виконати умови перемоги, через що проходження рівня закінчиться і користувач побачить меню перемоги;
- якщо після колізії з ворогом здоров’я персонажа стає меншим за 0, персонаж вважається мертвим, виводиться меню програшу.

Висновки до розділу

Підсумовуючи, у розділі «Передпроєктне обстеження предметної області» були розглянуті основні аспекти розробки комп’ютерного ігрового застосунку. На основі проведеного аналізу була сформульована постановка завдання дипломного проєктування, а також зазначені конкретні завдання, які треба розв’язати у рамках дипломної роботи.

Було надано визначення основним поняттям що будуть використовуватись у рамках даної роботи. Крім того, було виконано аналіз предметної області і надана статистика щодо актуальності цієї теми на сьогодні.

Був створений порівняльний аналіз цього проекту та вже існуючих аналогів у жанрі 2D-платформер і надана інформація про їх сильні та слабкі сторони. Критерії аналізу стосуються як самого геймплею так і можливостей користувача поза межами ігрового процесу. Одними з основних критеріїв були наявний редактор рівнів, що дозволяє гравцям створювати свої власні

ігрові рішення , а також реалізована система що дозволяла б гравцям обмінюватись рівнями.

Крім цього, були проаналізовані допоміжні програмні засоби такі як: ігровий рішуй, середовище розробки, редактор графіки тощо. Для кожного з вказаних типів програм було наведено кілька популярних рішень, після чого здійснено обґрунтований вибір найбільш доцільного варіанту з урахуванням особливостей проєкту.

Також в цьому розділі були створені BPMN моделі що описують окремі ключові бізнес-процеси, пов’язані з функціонуванням гри.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання з ключовими акторами та основними функціями зображена у графічному матеріалі, креслення 1.

В таблицях 2.1-2.12 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Керування персонажем
Use case ID	UC-01
Goals	Дати можливість керувати ігровим персонажем у середині гри
Actors	Користувач
Trigger	Користувач бажає керувати персонажем
Pre-conditions	Користувач зайшов на рівень. Гра не на паузі. Персонаж не мертвий. Рівень не вважається пройденим
Flow of Events	Користувач натискає на відповідні кнопки на клавіатурі що відповідають за рух персонажа.
Extension	Персонаж змінює своє положення
Post-Condition	-

Таблиця 2.2 – Варіант використання UC-02

Use case name	Система здоров’я персонажа
Use case ID	UC-02

Продовження таблиці 2.2

Goals	Дати можливість змінювати запас здоров’я ігрового персонажа. Передбачити смерть ігрового персонажа(програш), у випадку якщо його запас здоров’я закінчиться.
Actors	Користувач
Trigger	Під час керування персонажем, користувач отримує шкоду або наштовхується на об’єкт що поповнює запас здоров’я.
Pre-conditions	Користувач зайшов на рівень. Гра не на паузі. Персонаж не мертвий. Рівень не вважається пройденим
Flow of Events	Запас здоров’я персонажа змінюється
Extension	Якщо у персонажа закінчується запас здоров’я, він вмирає, а рівень вважається програним
Post-Condition	-

Таблиця 2.3 – Варіант використання UC-03

Use case name	Система очок персонажа
Use case ID	UC-03
Goals	Дати можливість змінювати кількість очок ігрового персонажа.
Actors	Користувач
Trigger	Під час керування персонажем, користувач наштовхується на об’єкти що збільшують його кількість очок.
Pre-conditions	Користувач зайшов на рівень. Гра не на паузі. Персонаж не мертвий. Рівень не вважається пройденим

Продовження таблиці 2.3

Flow of Events	Кількість очок персонажа змінюється
Extension	-
Post-Condition	Кількість очок персонажа збільшена

Таблиця 2.4 – Варіант використання UC-04

Use case name	Проходження рівнів
Use case ID	UC-04
Goals	Дати можливість користувачу завершувати рівні під час ігрового процесу
Actors	Користувач
Trigger	Ігровий персонаж з’явився на початку рівня
Pre-conditions	Користувач зайшов на рівень. Гра не на паузі. Персонаж не мертвий. Рівень не вважається пройденим
Flow of Events	Користувач рухається персонажем по рівню, збирає діаманти для закінчення рівня. Рівень вважається закінченим коли користувач збере 3 діаманти.
Extension	Якщо персонаж помирає під час проходження рівня, користувач програє і процес закінчується.
Post-Condition	Меню перемоги з’являється у разі виконання умов, у разі смерті - з’являється меню програшу

Таблиця 2.5 – Варіант використання UC-05

Use case name	Взаємодія з ворогами
Use case ID	UC-05

Продовження таблиці 2.5

Goals	Дати можливість користувачу взаємодіяти з ворогами; вбивати та отримувати від них шкоду
Actors	Користувач
Trigger	Ігровий персонаж з’явився на початку рівня, користувач зустрів ворога на рівні
Pre-conditions	Користувач зайшов на рівень. Гра не на паузі. Персонаж не мертвий. Рівень не вважається пройденим
Flow of Events	Користувач рухається персонажем по рівню, зустрічає: динамічного ворога; статичного ворога; пастку. У разі якщо атака ворога попала по персонажу або персонаж потрапив у пастку, персонаж отримує шкоду і його запас здоров’я зменшується
Extension	Якщо персонаж помирає під час проходження рівня, користувач програє і процес закінчується.
Post-Condition	Меню перемоги з’являється у разі виконання умов, у разі смерті - з’являється меню програшу

Таблиця 2.6 – Варіант використання UC-06

Use case name	Навігація в головному меню
Use case ID	UC-06
Goals	Дати можливість користувачу користуватись головним меню
Actors	Користувач
Trigger	Користувач заходить у головне меню
Pre-conditions	Головне меню відкрите.

Продовження таблиці 2.6

Flow of Events	Користувач має змогу натиснути на одну з кнопок головного меню: «Рівні» переходить до меню готових рівнів; «Редактор рівнів» переходить до редактора рівнів; «Збережені рівні» переходить до меню збережених рівнів; «Вихід» виходить з гри.
Extension	-
Post-Condition	Відбувається обрана дія

Таблиця 2.7 – Варіант використання UC-07

Use case name	Зберігати власні рівні
Use case ID	UC-07
Goals	Дати можливість користувачу зберігати власні рівні розроблені в редактору рівнів
Actors	Користувач
Trigger	Користувач бажає зберегти створений рівень
Pre-conditions	Користувач зайшов у редактор рівнів. Розташував об'єкти бажаними чином.
Flow of Events	Користувач натискає на кнопку збереження рівня, після чого його переносить до меню власних рівнів, де він може вибрати комірку в яку хоче зберегти поточний рівень.
Extension	При бажанні, користувач може зберегти поточний рівень навіть у зайняту комірку. При цьому, старий рівень видалиться, а новий запишеться на його місце.
Post-Condition	Меню перемоги з’являється у разі виконання умов, у разі смерті - з’являється меню програшу

Таблиця 2.8 – Варіант використання UC-08

Use case name	Навігація в меню власних рівнів
Use case ID	UC-08
Goals	Дати можливість користувачу користуватись меню власних рівнів
Actors	Користувач
Trigger	Користувач заходить у меню власних рівнів
Pre-conditions	Меню власних рівнів відкрите.
Flow of Events	Користувач має змогу натиснути на одну з кнопок меню власних рівнів: «Власний рівень» користувач може натиснути ЛКМ по рівню, тоді розпочнеться проходження бажаного власного рівня ; «Власний рівень» користувач може натиснути ПКМ по рівню, тоді обраний власний рівень видалиться з комірки; «Назад» користувач повернеться до головного меню.
Extension	-
Post-Condition	Відбувається обрана дія

Таблиця 2.9 – Варіант використання UC-09

Use case name	Навігація в меню готових рівнів
Use case ID	UC-09
Goals	Дати можливість користувачу користуватись меню готових рівнів
Actors	Користувач
Trigger	Користувач заходить у меню готових рівнів

Продовження таблиці 2.9

Pre-conditions	Меню готових рівнів відкрите.
Flow of Events	Користувач має змогу натиснути на одну з кнопок меню готових рівнів:«Готовий рівень» користувач може натиснути ЛКМ по рівню, тоді розпочнеться проходження бажаного готового рівня ;«Назад» користувач повернеться до головного меню/
Extension	-
Post-Condition	Відбувається обрана дія

Таблиця 2.10 – Варіант використання UC-10

Use case name	Навігація в меню паузи
Use case ID	UC-10
Goals	Дати можливість користувачу користуватись меню паузи
Actors	Користувач
Trigger	Користувач натискає на кнопку паузи під час проходження рівня
Pre-conditions	Користувач зайшов на рівень. Персонаж не мертвий. Рівень не вважається пройденим
Flow of Events	Користувач бачить свій поточний результат проходження рівня у меню паузи. Користувач має змогу натиснути на одну з кнопок меню паузи: «Продовжити» користувач виходить із меню паузи і повертається до гри ; «Перезапуск» рівень перезапускається з початку і користувач починає його проходження; «Вихід» користувач повертається до головного меню.

Продовження таблиці 2.10

Extension	-
Post-Condition	Відбувається обрана дія

Таблиця 2.11 – Варіант використання UC-11

Use case name	Навігація в меню обміну рівнями
Use case ID	UC-11
Goals	Дати можливість користувачу користуватись меню обміну рівнями
Actors	Користувач
Trigger	Користувач заходить у меню обміну рівнями
Pre-conditions	Меню обміну рівнями відкрите.
Flow of Events	
Extension	-
Post-Condition	Відбувається обрана дія

Таблиця 2.12 – Варіант використання UC-12

Use case name	Вийти з гри
Use case ID	UC-12
Goals	Дати можливість користувачу вийти з гри
Actors	Користувач
Trigger	Користувач натискає на кнопку вийти з гри у головному меню

Продовження таблиці 2.12

Pre-conditions	Користувач знаходиться у головному меню
Flow of Events	Користувач натискає на кнопку виходу гри, гра закривається
Extension	-
Post-Condition	Користувач закрив гру

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.13 наведено загальну модель вимог, а в таблиці 2.14 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.13 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Керування персонажем	FR-1	Високий	Високий
1.1	Переміщення персонажа	FR-2	Високий	Середній
1.2	Система здоров’я персонажа	FR-3	Високий	Середній
1.2.1	Смерть персонажу	FR-4	Високий	Високий
1.3	Атака персонажа	FR-5	Високий	Середній
2	Зміна очок персонажа	FR-6	Середній	Низький
2.1	Взаємодія з об’єктами очок	FR-7	Середній	Низький
3	Проходження рівня	FR-8	Високий	Низький
3.1	Взаємодія з об’єктами що потрібні для проходження рівня	FR-9	Високий	Низький
3.2	Пауза	FR-10	Середній	Середній
4	Вороги	FR-11	Високий	Високий
4.1	Статичні вороги	FR-12	Середній	Середній
4.2	Динамічні вороги	FR-13	Середній	Середній
4.3	Пастки	FR-14	Середній	Середній
4.4	Вбивство ворогів	FR-15	Середній	Середній
5	Інтерфейс	FR-16	Високий	Високий
5.1	Ігровий інтерфейс	FR-17	Середній	Низький
5.2	Головне меню	FR-18	Високий	Середній

Продовження таблиці 2.13

5.3	Вийти з гри	FR-19	Низький	Низький
5.4	Меню готових рівнів	FR-20	Середній	Середній
5.5.1	Готові рівні	FR-21	Низький	Низький
5.5	Меню власних рівнів	FR-22	Середній	Середній
5.5.1	Комірки власних рівнів	FR-23	Середній	Середній
5.6	Почати гру	FR-24	Середній	Високий
5.7	Меню паузи	FR-25	Середній	Низький
5.8	Меню перемоги/програшу	FR-26	Середній	Низький
5.9	Меню обміну рівнями	FR-27	Високий	Середній
5.9.1	Меню авторизації	FR-28	Середній	Середній
5.9.2	Меню видалення особистої інформації	FR-29	Середній	Високий
5.9.3	Меню локальних рівнів	FR-30	Середній	Середній
5.9.4	Меню публічних рівнів	FR-31	Середній	Середній
5.9.5	Меню пропозицій до змін	FR-32	Середній	Середній
6	Збереження власних рівнів	FR-33	Високий	Високий
6.1	Видалення власних рівнів	FR-34	Середній	Низький
6.2	Перезапис власних рівнів	FR-35	Середній	Низький
7	Сканування файлу рівня на шкідливий вміст.	FR-36	Середній	Високий

Таблиця 2.14 – Перелік функціональних вимог

Назва	Опис
FR-1	Керування персонажем. Користувач має змогу керувати ігровим персонажем у мажах рівня.
FR-2	Переміщення персонажа. Користувач має змогу переміщати ігрового персонажа у межах рівня. При цьому, персонаж не може проходити крізь платформи або стіни. Персонаж може як ходити вліво та вправо, якщо йому нічого не заважає, так і стрибати та падати.
FR-3	Система здоров’я персонажа. Користувач має змогу змінювати запас здоров’я персонажа у процесі проходження рівня.
FR-4	Смерть персонажу. Користувач має змогу знизити запас здоров’я персонажа до 0, тим самим вбити його. При смерті персонажа гра вважається програною.
FR-5	Атака персонажа. Користувач має змогу атакувати певних ворогів за ігрового персонажа, тим самим впливаючі на них.

Продовження таблиці 2.14

FR-6	Зміна очок персонажа. Користувач має змогу змінювати кількість очок персонажа під час проходження рівня. Очки будуть підраховані і виведені у меню перемоги/програшу.
FR-7	Взаємодія з об’єктами очок. Користувач має змогу взаємодіяти з об’єктами що будуть впливати на кількість очок.
FR-8	Проходження рівня. Користувач має змогу пройти рівень, у разі виконання умов для його проходження. Умовою проходження рівня є збір 3 діамантів розташованих по рівню.
FR-9	Взаємодія з об’єктами що потрібні для проходження рівня. Користувач має змогу взаємодіяти з діамантами, підбираючи їх. При цьому оновлюється інформація щодо прогресу користувача на рівні. Вона буде виведена у разі перемоги/програшу.
FR-10	Пауза. Користувач має змогу поставити проходження рівня на паузу за потреби. При цьому проходження рівня «заморожується» і прогрес зберігається. У разі потреби, персонаж може «розморозити» рівень знявши його з паузи і продовжити проходження.
FR-11	Вороги. Користувач має змогу взаємодіяти з ворогами розташованими на рівнях. Вони можуть завдавати шкоду ігровому персонажу. Вороги виступають певним ускладненням рівня на шляху до перемоги.
FR-12	Статичні вороги. Користувач має змогу взаємодіяти зі статичними ворогами. Статичні вороги не мають змоги пересуватись по платформам, але мають змогу атакувати ігрового персонажа у разі виконання умов для атаки.
FR-13	Динамічні вороги. Користувач має змогу взаємодіяти з динамічними ворогами. Динамічні вороги мають змогу пересуватись по платформам, а також атакувати ігрового персонажа у разі виконання умов атаки.
FR-14	Пастки. Користувач має змогу взаємодіяти з пастками. Пастки виступають частиною інтер’єру рівня при контакті з якою користувач отримує шкоду.
FR-15	Вбивство ворогів. Користувач має змогу вбивати певних ворогів. Для вбивства ворога треба виконати умови що залежать від виду ворога. При вбивстві ворога він пропадає з рівня і більше не становить загрози.

Продовження таблиці 2.14

FR-16	Інтерфейс. Користувач має змогу взаємодіяти з інтерфейсом гри для зручної навігації, виконання бажаних дій і отримання ігрової інформації.
FR-17	Ігровий інтерфейс. Користувач має змогу взаємодіяти з ігровим інтерфейсом. Ігровий інтерфейс - це інтерфейс що користувач бачить безпосередньо під час проходження рівня. Він допомагає користувачу орієнтуватися в поточному прогресі проходження рівня і відображає корисну ігрову інформацію.
FR-18	Головне меню. Користувач має змогу взаємодіяти з головним меню що виступає першою частиною гри що побачить користувач при запуску. Воно містить: логотип або назву гри; кнопку для переходу до меню готових рівнів; кнопку для переходу у редактор рівнів; кнопку для переходу у меню збережених рівнів; кнопку для виходу з гри.
FR-19	Вийти з гри. Користувач має змогу закрити ігровий застосунок натиснувши на кнопку «вийти» у головному меню гри.
FR-20	Меню готових рівнів. Користувач має змогу взаємодіяти з меню готових рівнів обираючи бажаний рівень для проходження. Також, користувач має можливість натиснути на кнопку «назад» і повернутися до головного меню.
FR-21	Готові рівні. Користувач має змогу проходити готові рівні, що йдуть у базовій комплектації гри. Ці рівні потрібні гравцю для ознайомлення з базовими правилами гри.
FR-22	Меню власних рівнів. Користувач має змогу взаємодіяти з меню власних рівнів обираючи бажаний рівень для проходження. Також, користувач має можливість натиснути на кнопку «назад» і повернутися до головного меню.
FR-23	Комірки власних рівнів. Користувач має змогу взаємодіяти з комірками власних рівнів. Користувач має змогу самостійно зберігати, видаляти, перезаписувати власні рівні.
FR-24	Почати гру. Користувач має змогу почати проходження рівня, обравши його у меню готових рівнів або у меню власних рівнів.

Продовження таблиці 2.14

FR-25	Меню паузи. Користувач має змогу взаємодіяти з меню паузи. В меню паузи користувач може побачити свій поточний прогрес по рівню. Також користувач має змогу натиснути на кнопки: «продовжити» для продовження проходження; «перезапуск» для того щоб почати рівень заново; «вихід» щоб повернутись у головне меню.
FR-26	Меню перемоги/програшу. Користувач має змогу взаємодіяти з меню перемоги/програшу. Ці меню з'являються при закінченні рівню (шляхом смерті ігрового персонажа або перемоги у рівні) і відображають особисту статистику користувача на цьому рівні.
FR-27	Меню обміну рівнями Користувач має змогу взаємодіяти з меню обміну рівнями. Це меню включає в себе: меню авторизації; меню видалення особистої інформації; меню локальних рівнів; меню публічних рівнів; меню пропозицій до змін.
FR-28	Меню авторизації Це меню надає інтерфейс для авторизації користувачів.
FR-29	Меню видалення особистої інформації Це меню надає інтерфейс для видалення особистої інформації.
FR-30	Меню локальних рівнів Це меню надає інтерфейс для взаємодії з власними рівнями
FR-31	Меню публічних рівнів Це меню надає інтерфейс для взаємодії з публічними рівнями
FR-32	Меню пропозицій до змін Це меню надає інтерфейс для взаємодії з системою змін
FR-33	Збереження власних рівнів. Користувач має змогу зберігати власні рівні, розроблені у редакторі рівні. Він має змогу помістити їх у комірки меню власних рівнів.
FR-34	Видалення власних рівнів. Користувач має змогу видалити власні рівні, перебуваючи у меню власних рівнів і натиснувши ПКМ по зайнятій комірці з рівнем.
FR-35	Перезапис власних рівнів. Користувач має змогу перезаписати власні рівні, перейшовши з редактору рівнів до меню власних рівнів і натиснувши ЛКМ по зайнятій комірці.
FR-36	Сканування файлу рівня на шкідливий вміст. Користувач має змогу завантажити файл з рівнем і бути впевненим у його вмісті. Для цього потрібно сканувати файл рівня перед тим як будь-який користувач хоче завантажити його на сервер

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20	FR-21	FR-22	FR-23	FR-24	FR-25	FR-26	FR-27	FR-28	FR-29	FR-30	FR-31	FR-32	FR-33	FR-34	FR-35	FR-36
UC-1	×	×			×		×		×						×		×																			
UC-2			×	×																																
UC-3						×																														
UC-4								×		×							×								×	×										
UC-5											×	×	×	×	×																					
UC-6																×		×	×																	
UC-7																																	×	×	×	×
UC-8																×						×	×	×										×		
UC-9																×				×	×			×												
UC-10																×																				
UC-11																											×	×	×	×	×	×				
UC-12																			×																	

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Користувач не повинен відчувати в картинці маленьку кількість кадрів, тож необхідно щоб ПК користувача не був перевантаженим та міг стабільно відображати кадри щоб користувач не відчував дискомфорту.

Перехід між компонентами меню повинен бути виконаний через анімацію, щоб всі компоненти меню плавно з’являлися перед гравцем.

Користувацький інтерфейс повинен бути зрозумілим на інтуітивному рівні, що повинно досягатись завдяки вдалому розташуванню компонентів та використанні відповідних графічних компонентів. Тобто максимальна глибина в ієрархії меню не повинна перевищувати 3.

Користувач повинен мати змогу ознайомитись з основними правилами гри пройшовши базові рівні. Вони нададуть користувачу розуміння роботи переміщення головного персонажа, роботи ворогів та пасток, а також умов що необхідно виконати для успішного проходження рівня.

Користувач повинен мати змогу чути музику та відповідні звукові ефекти під час гри. Це допоможе як у зануренні в атмосферу гри, так і більш вдало реагувати на ігрові події.

Також, користувач повинен мати змогу взаємодіяти з різноманітними ворогами та перешкодами, що мають різні патерни дій. Через це користувачу потрібно буде адаптуватися до кожного нового рівня, що не дозволить гри стати занадто простою для проходження.

2.4 Аналіз системних вимог

Програмне забезпечення функціонуватиме на пристроях, що працюють на ОС Windows 10.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3;

– об‘єм ОЗП: 4 Гб;

– об’єм вільної пам’яті накопичувача: 300 Мб.

– швидкість інтернету 5 мб/с

Рекомендована конфігурація технічних засобів :

– тип процесору: Intel Core i5;

– об‘єм ОЗП: 16 Гб;

– об’єм вільної пам’яті накопичувача: 300 Мб.

– швидкість інтернету 10 мб/с

Мінімальні системні вимоги відповідають тим пристроям, котрі здатні підтримувати дану версію ОС Windows.

2.5 Аналіз економічних показників програмного забезпечення

Для аналізу економічних показників програмного забезпечення буде розрахована кількість «UCP». На основі діаграми використання, складаємо перелік акторів («UAW»), повний перелік Use Cases («UUCW»), Unadjusted Use-case Points («UUCP»), Technical Complexity Factor («TCF»), Environmental Factors(«EF») та обрахуємо остаточну кількість Use-case points(«UCP»).

2.5.1 Актори («UAW»)

Таблиця 2.15 – Актори(«UAW»):

Тип	Опис	Коефіцієнт	Кількість
Простий	Локальні файли рівнів (Pickle, JSON) та серверна база даних.	1	1

Продовження таблиці 2.15

Середній	Програмний модуль/бібліотека, що надсилає та отримує HTTP-запити до/від сервера (PHP, Django чи будь-який клієнт, який працює з REST).	2	1
Складний	Людина, що керує грою/редактором, вводить дані, приймає рішення, натискає кнопки, переглядає списки, запускає рівень тощо.	3	1

Для підрахунку, складемо коефіцієнти акторів, домноживши на їх кількість:

$$UAW = 3 + 2 + 1 = 6 \quad (2.1)$$

2.5.2 Повний перелік Use Cases («UUCW»)

Таблиця 2.16 – Повний перелік Use Cases («UUCW»):

Назва Use Case	Категорія	Вага (UCP)
Проходження рівня	Складний	15
Навігація в меню паузи	Простий	5
Керування персонажем	Складний	15
Взаємодія з ворогами	Складний	15
Система очок персонажа	Простий	5
Система здоров’я персонажа	Простий	5
Вийти з гри	Простий	5
Навігація в головному меню	Простий	5
Навігація в меню готових рівнів	Простий	5
Навігація в меню обміну рівнями	Простий	5
Навігація в меню власних рівнів	Простий	5
Збереження рівня (у редакторі)	Середній	10
Редактор рівнів	Середній	10
Створення та редагування рівня	Середній	10

Продовження таблиці 2.16

Авторизація (реєстрація/вхід)	Простий	5
Перегляд публічних рівнів	Простий	5
Виставлення рівня (Publish Level)	Складний	15
Пропозиція змін до рівнів інших гравців	Середній	10
Перегляд пропонованих змін	Середній	10
Відмовитися від зміни	Середній	10
Застосувати зміну	Простий	5
Видалити особисту інформацію	Середній	10

Підрахуємо загальну вагу:

$$UUCW = 55 + 70 + 60 = 185 \quad (2.2)$$

2.5.3 Unadjusted Use-case Points («UUCP»)

Для отримання «UUCP», складемо раніше отримані «UAW» та «UUCW»:

$$UUCP = UAW + UUCW = 6 + 185 = 191 \quad (2.3)$$

2.5.4 Technical Complexity Factor («TCF»)

Таблиця 2.17 – Technical Complexity Factor («TCF»):

Фактор	Wi	Fi	Wi×Fi
Розподілена система	2.0	4	8.0
Вимоги до продуктивності	1.0	2	2.0
Зручність для кінцевого користувача	1.0	4	4.0
Складна внутрішня обробка	1.0	3	3.0
Повторне використання коду	1.0	2	2.0
Легкість встановлення	0.5	1	0.5

Продовження таблиці 2.17

Зручність використання (UI)	0.5	3	1.5
Портативність	2.0	2	4.0
Легкість внесення змін	1.0	3	3.0
Паралельне використання	1.0	3	3.0
Спеціальні вимоги до безпеки	1.0	3	3.0
Наявність навчальних матеріалів	0.5	1	0.5
Легкість конфігурації	0.5	2	1.0

Підрахуємо коефіцієнт технічної складності використовуючи наступну формулу:

$$TCF = 0.6 + 0.01 \times \sum(Wi * Fi) = 0.955 \quad (2.4)$$

2.5.5 Environmental Factors («EF»)

Таблиця 2.18 – Environmental Factors («EF»):

Фактор	Wi	Fi	Wi×Fi
Знання UML/моделювань	1.5	3	4.5
Досвід із подібними додатками	0.5	2	1.0
Досвід об’єктно-орієнтованого програмування	1.0	3	3.0
Здібності ведучого аналітика	0.5	3	1.5
Мотивація	1.0	4	4.0
Стабільність вимог	2.0	2	4.0
Часткова зайнятість персоналу	1.0	1	1.0

Продовження таблиці 2.18

Складність мови програмування	1.0	2	2.0

Також підрахуємо фактори навколишнього середовища по формулі:

$$EF = 1.4 + (-0.03 \times \sum(Wi * Fi)) = 0.77 \quad (2.5)$$

2.5.6 Остаточний розрахунок «UCP»

У результаті, отримаємо наступну кількість «UCP»:

$$UCP = UUCP \times TCF \times EF \approx 191 \times 0.73535 \approx 140 \quad (2.6)$$

2.5.7 Переведення «UCP» у трудомісткість і вартість

Розрахуємо скільки знадобиться часу на його реалізацію у одного розробника, при тому що він буде працювати по 160 годин у місяць:

$$140 \text{ UCP} * \frac{20 \text{ год}}{UCP} = 2800 \text{ годин} \quad (2.7)$$

$$\frac{2800 \text{ год}}{160} = 17.5 \text{ PM} \quad (2.8)$$

Так як цей проєкт комплексний, ми отримуємо що кожен з 2 розробників повинен витратити по 8.75 місяці на реалізацію.

Для підрахунку загальної вартості проекту можна взяти середню зарплату Junior розробника на мові Python, що складає близько 700$, або приблизно 29000 гривень.

Тож можна сказати що загальна вартість цього проекту складає біля 507500 гривень.

2.6 Постановка завдання на розробку програмного забезпечення

Призначення розробки - безкоштовна комп’ютерна гра у жанрі Platformer, з розробкою редактора рівнів і системою обміну рівнями. Цей проект призначений для дозвілля та розвитку реакції користувача. Метою розробки є сприяння розвитку логічного та стратегічного мислення гравця .

Дана мета досягається завдяки вирішенню задач зі списку:

- створення ігрового світу заповнюючі його різноманітними інтерактивними об’єктами, що дозволять персонажу пересуватись, отримувати очки, прогресувати у проходженні рівня ,змінювати власне здоров’я;
- керування ігровим персонажем та розробка основних законів фізики рухів, що надасть можливість користувачу взаємодіяти із ігровим світом і з об’єктами у ньому;
- різноманітні вороги на рівнях, що будуть вимагати унікального підходу для проходження крізь них або перемоги над ними;
- реалізація користувацького інтерфейсу як для коректного відображення ігрової інформації, так і для навігації по застосунку;
- розробка системи збереження рівнів, створених у редакторі рівнів, для подальшого перепроходження та системи безпечного поширення цих рівнів.

Висновки до розділу

У цьому розділі «Розроблення вимог до ПЗ» було виконано аналіз вимог до програмного забезпечення, потрібних для розробки проєкту - комп’ютерна гра у жанрі Platformer, з розробкою редактора рівнів і системою обміну рівнями.

Було створено діаграму варіантів використання програмного забезпечення, яка відображає сценарії взаємодії користувача з різними елементами гри, зокрема з ігровими меню, виходом із гри, початком проходження рівня тощо. Це дало змогу визначити ключові сценарії, що мають вирішальне значення для формування зручного та ефективного застосунку.

Також були розроблені функціональні вимоги до застосунку, які окреслюють основні функції гри, такі як : система здоров'я та очок; керування персонажем; різновиди ворогів тощо. Ці вимоги визначають умови для стабільної роботи застосунку та реалізації усього запланованого функціоналу. У відповідності до функціональних вимог та варіантів використань була побудована матриця трасування вимог.

Крім того, були сформульовані нефункціональні вимоги, які переважно стосуються продуктивності, стабільності роботи, зручності користування, а також цілісності та послідовності дизайну застосунку. Дотримання цих вимог відіграє важливу роль у забезпеченні позитивного користувацького досвіду.

Був проведений аналіз системних вимог для визначення мінімальних та рекомендованих параметрів систем для запуску цього застосунку. А також був проведений аналіз економічних показників для визначення основних метрик що впливали на розробку цього застосунку.

Так у кінці, було визначено призначення та мету цієї розробки. Цей безкоштовний комп’ютерний застосунок призначений для покращення дозвілля, а також реакції користувача. А його метою є сприяння розвитку логічного та стратегічного мислення гравця .

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Розроблене програмне забезпечення побудоване за принципами клієнт-серверної архітектури, що дозволяє розділити систему на два незалежних, але взаємодіючих компоненти - клієнтську частину (ігровий застосунок) та серверну частину (вебсервер). Такий підхід забезпечує масштабованість, підтримку взаємодії між користувачами, централізоване зберігання даних, а також спрощує оновлення та обслуговування окремих частин системи.

3.1.1 Клієнт

Клієнтська частина реалізована на мові програмування Python з використанням бібліотеки Pygame. Вона відповідає за наступний функціонал:

- запуск самої гри;
- запуск редактора рівнів;
- інтерфейс для створення, редагування та тестування рівнів;
- взаємодія із сервером через HTTP-запити для завантаження та надсилання рівнів;
- авторизація користувача.

Дані рівнів, створені в редакторі, зберігаються у вигляді вкладених словників (JSON-подібна структура), які пізніше надсилаються на сервер для централізованого збереження та обміну.

3.1.2 Сервер

Серверна частина реалізована на базі Django з використанням Django REST Framework. Вона забезпечує:

- обробку HTTP-запитів від клієнтів;
- авторизацію користувачів (на основі JWT);

- збереження рівнів у базі даних;
- доступ до рівнів інших користувачів;
- обробку запитів на зміну чужих рівнів (механізм «pull request» у контексті рівнів);
- схвалення або відхилення запропонованих змін автором рівня.

Комунікація клієнта з сервером відбувається через REST API, що дозволяє передавати дані у форматі JSON. Така взаємодія є простою у реалізації, зручною для дебагу та масштабованою.

3.1.3 Архітектурний патерн MVC

Серверна частина побудована з урахуванням архітектурного патерну MVC (Model–View–Controller)[18]:

Model - відповідає за структуру даних у базі (ORM-моделі): користувачі, рівні, запити на зміни. Наприклад, модель Level зберігає назву рівня, автора, вміст у форматі JSON, дату створення тощо.

View - реалізує представлення даних у форматі REST API. Це серіалізатори та представлення у Django REST Framework, які конвертують об’єкти моделей у формат JSON та навпаки, не містячи при цьому логіки.

Controller - містить бізнес-логіку, яка обробляє запити: перевірка автентичності користувача, логіка оновлення рівнів, зміна статусу запиту на редагування, авторизаційні обмеження тощо.

Такий розподіл дозволяє чітко структурувати логіку серверної частини, полегшити її тестування, оновлення та масштабування.

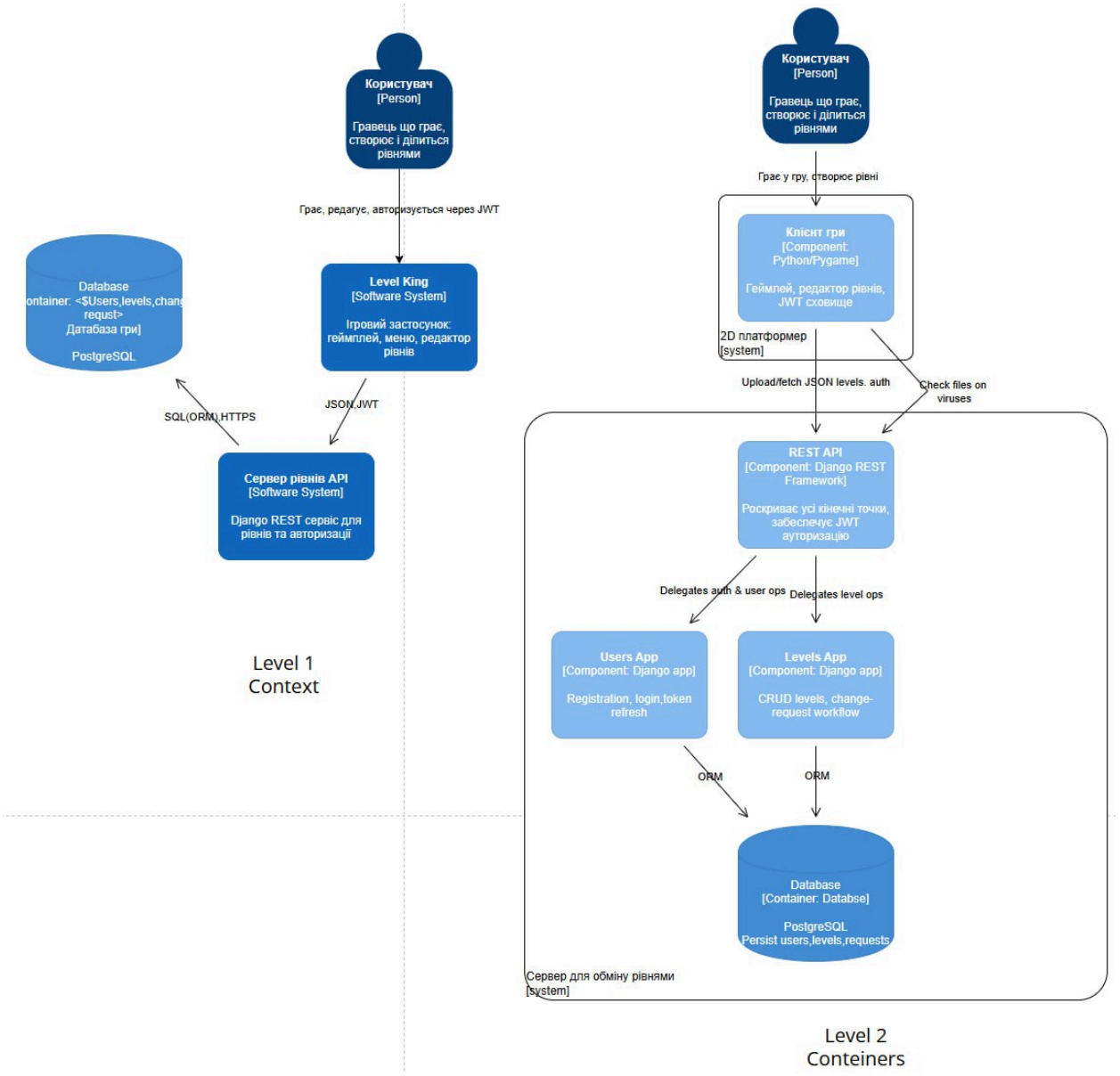


Рисунок 3.1 – Діаграма архітектури

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

Для розробки клієнтської частини гри було обрано низку ключових інструментів, зокрема Pygame як основу ігрового рушія та PyCharm як основне середовище розробки. Вибір цих засобів базувався на їхній ефективності, повній відповідності технічним потребам проєкту, а також на зручності у повсякденній розробці.

Pygame був обраний як головна бібліотека для реалізації ігрової логіки та графіки. Основними аргументами на користь Pygame стали:

– підтримка 2D-графіки: Pygame спеціалізується на створенні 2D-ігор і забезпечує всі базові інструменти для рендерингу спрайтів, обробки подій, анімацій та керування звуком, що ідеально підходить для платформера;

– гнучкість: у Pygame немає заздалегідь заданих шаблонів, що дозволяє повністю контролювати архітектуру гри, побудову редактора рівнів та взаємодію з сервером;

– Python-орієнтованість: оскільки серверна частина реалізована на Django, використання Pygame забезпечує однорідність технологій і спрощує інтеграцію клієнт-серверної взаємодії через REST API;

– простота освоєння: у порівнянні з важкими рушіями, такими як Unity чи Unreal Engine, Pygame дозволяє зосередитися безпосередньо на логіці гри, не витрачаючи час на вивчення складних редакторів або графічних інтерфейсів;

– активна спільнота: незважаючи на свою простоту, Pygame має велику базу прикладів, документації та підтримку від спільноти, що пришвидшує розробку.

Подібні можливості мають інші рушії, такі як Godot, однак він має власну логіку побудови сцен, що не дозволяє досягти такої гнучкості при створенні кастомного редактора рівнів і прямої інтеграції з Python-базованим сервером, як це можливо з Pygame.

PyCharm був обраний як основне середовище розробки. Це IDE від JetBrains спеціально оптимізоване для Python і містить усі необхідні інструменти для зручної, стабільної роботи над складними багатомодульними проєктами.

Функціональні можливості PyCharm:

– ефективна робота з Pygame: налаштування конфігурацій запуску, дебаг графіки та інтеграція з середовищем виконання Python відбуваються без труднощів;

– вбудована система налагодження (debug): дозволяє ефективно знаходити помилки як у клієнтській, так і серверній частині;

– Code With Me: функція спільної розробки в реальному часі, що дозволяє декільком учасникам команди працювати в одному проєкті паралельно, редагуючи й переглядаючи код без потреби передавати проєкт вручну;

– підтримка Git ,що дозволяє використовувати систему контролю версій, що в свою чергу дозволяє легко керувати історією змін, комітами та злиттями.

Visual Studio Code також міг би бути використаний як альтернатива, однак він більше орієнтований на легку роботу з кодом і потребує встановлення та налаштування численних розширень для досягнення подібної функціональності. PyCharm ж забезпечує повну готовність до роботи з Python-проєктами одразу після встановлення, що дозволяє зосередитися на самій розробці, а не на конфігурації середовища.

3.3 Конструювання програмного забезпечення

Архітектура гри складається з кількох сцен, котрі всі запускаються з екземпляру основного класу Main і перемикаються між собою в залежності від дій користувача. Діаграма класів програмного забезпечення зображена у графічному матеріалі, креслення 2.

Якщо говорити більш детально, то в грі ми маємо п’ять головних сцен, в них реалізуються різні вікна програмного забезпечення, поділ логіки. Main_menu реалізовує головне меню програмного меню і також має перехід до меню рівнів, меню завантаження, редактора рівнів та меню обміну рівнями. Меню рівнів, меню завантаження та редактор рівнів реалізують в свою чергу перехід до сцени рівня, але кожен з використанням власної сітки розташування елементів. Меню обміну рівнями надає зручний інтерфейс для завантаження власних рівнів, перегляду публічних рівнів, надання та отримання рекомендацій щодо зміни рівнів. Реалізує перехід між компонентами клас Passage, а за користувацький інтерфейс на рівні

відповідає клас UI. Більш детальний розбір класів та їх наслідувань можна побачити на рисунку.

Таблиця 3.1 – Опис призначення основних класів:

Назва класу	Опис класу
EntityBase	Базовий спадкоємець pygame.sprite.Sprite, надає «пасивні» спрайти: зберігає зображення, Z-шар і бере участь у групах колізій чи відмалювання. Від нього походить більшість нерухомих об’єктів рівня.
AnimatedEntity	Розширює EntityBase: додає масив кадрів, лічильник кадру та швидкість прокрутки, тож підкласові достатньо заповнити словник анімацій, аби отримати циклічний рух.
FXParticle	Візуальний ефект: відтворює послідовність кадрів і самознищується, коли анімація закінчилась, тому підходить для вибухів та об’єктів.
Treasure	Колекційний предмет. Обертається за допомогою AnimatedEntity, після торкання підвищує лічильник і зникає з рівня.
Player	Центральний керований об’єкт: обробляє введення, гравітацію, багатоступеневу анімацію та взаємодію з довкіллям (стрибки, ковзання, тощо). Має власні таймери безсмертя й атаки.
CameraRig	Перевизначений pygame.sprite.Group. Рахує зсув камери відносно позиції гравця та малює спрайти у глибину (сортування по Y). Додає «parallax»-зміщення для фону.
UI	Графічний інтерфейс поверх рівня: містить лічильники життів, монет, алмазів і короткі повідомлення. Окремий метод update(), щоб логіка UI не змішувалась із логікою рівня.
GameLevel	«Серце» геймплею: завантажує карту, ініціалізує всі спрайти, керує циклами подій, колізіями, перемогою/поразкою та паузою. Спілкується з UI, Timer і менеджером переходів.
Main	Дирижер для всіх глобальних станів: запускає головне меню, передає керування рівню, редактору чи таблиці результатів і стежить за плавними переходами між ними.
Timer	Універсальний лічильник (зворотний або «просто відмітка часу»). Використовується для анімацій ворогів, респавну частинок та ефектів безсмертя.

Продовження таблиці 3.1

Passage	Анімація між станами. Малює чорний прямокутник з прозрачним колом посередині. Коли коло зводиться до центру зображення, змінює стан на обраний та повторює анімацію в обратному напрямку.

3.3.1 Алгоритм роботи динамічного ворога(Pig)

Для роботи рухомого ворога спочатку випадковим чином обирається напрямок його руху, при цьому ключовою умовою є те, що ворог не може бути створений у повітрі ,під час ініціалізації екземпляра класу створюється rect зі стандартного класу спрайтів Pygame, який перевіряє дотик до поверхні, і якщо такий відсутній, екземпляр видаляється. Анімації руху відтворюються відповідно до поточного напрямку: рух вліво супроводжується анімацією бігу вліво, а рух вправо анімацією бігу вправо. Логіка переміщення реалізована за аналогічним принципом: два rect перевіряють дотик зліва та справа для обробки зіткнень зі стінами, а ще два rect, розташовані знизу зліва та знизу справа, відповідають за виявлення кінців платформ, що забезпечує коректну поведінку ворога при русі та взаємодії з ігровим середовищем(рис. 3.2).

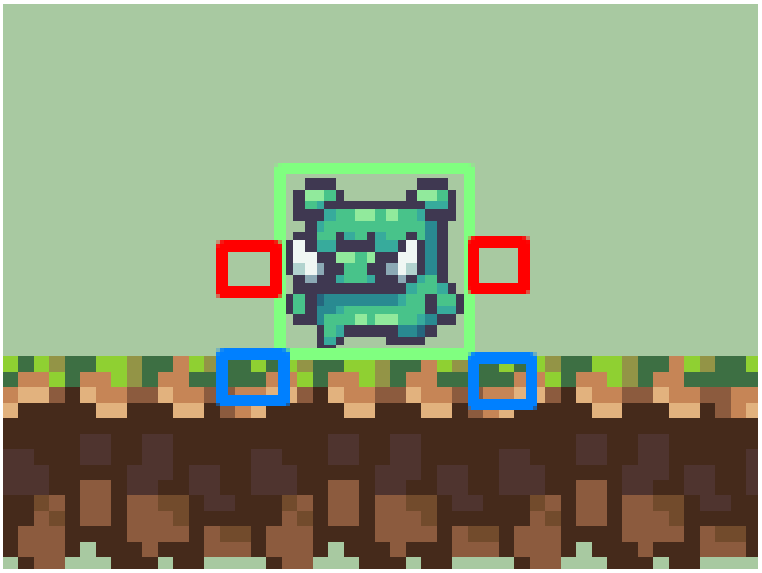


Рисунок 3.2 – Розташування зон перевірки дотику

Механіка смерті ворога реалізована через аналіз взаємного розташування точок моделей: система порівнює положення верхньої точки

моделі ворога, його середньої точки та нижньої точки гравця. При зіткненні з гравцем, якщо його нижня межа знаходиться між серединою та верхньою частиною ворога, і одночасно активний напрямок руху "вниз", екземпляр ворога зупиняється, його спрайт миттєво змінює колір на пів секунди, після чого об'єкт повністю видаляється з ігрового простору.

3.3.2 Алгоритм роботи статичного ворога(Cannon)

Механіка атаки нерухомого ворога Cannon працює за принципом дистанційної активації, якщо гравець знаходиться на відстані менше 700 пікселів, ворог переходить у режим атаки, при більшій відстані залишається у стані спокою. Процес атаки полягає у створенні снаряду (екземпляра класу Cannonball), який рухається у напрямку, визначеному поточним положенням Cannon, з заданою швидкістю. Важливою умовою для активації атаки є статус таймера атаки (екземпляр класу Timer) ,якщо таймер активний, ворог переходить у стан «вже атакував» і очікує його завершення, у іншому випадку ініціюється атака. Створення снаряду відбувається виключно під час другого кадру анімації атаки (моменту пострілу з гармати)(рис. 3.3). Снаряд має обмежений час існування (3.5 секунди) і зникає при одній з трьох умов: при дотику з гравцем (викликається процедура нанесення шкоди), при зіткненні з твердою поверхнею або після закінчення встановленого часу життя.

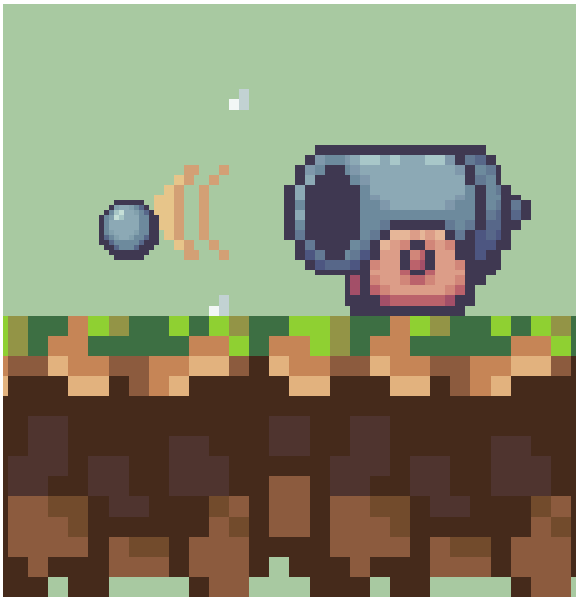


Рисунок 3.3 – Створення снаряду нерухомого ворога

3.3.3 Алгоритм роботи обертової пастки (Mace)

Під час ініціалізації системи визначається центр обертання, а саме геометричний центр тайлу, до якого прив’язана пастка. Задається радіус траєкторії руху та початковий кут обертання, після чого за допомогою тригонометричних функцій sin і cos обчислюються стартові координати спрайта на колі вказаного радіуса. При кожному оновленні кадру поточний кут збільшується на величину, пропорційну кутовій швидкості та дельті часу (для забезпечення плавності незалежно від частоти кадрів). На основі оновленого кута перераховуються нові координати спрайта відносно центру обертання, після чого позиція його прямокутника оновлюється, забезпечуючи переміщення. В результаті пастка рівномірно обертається по колу навколо фіксованої точки з постійною швидкістю та незмінним радіусом, створюючи ефект циклічного кругового руху без змін у висоті чи віддалі від центру. (рис. 3.4).

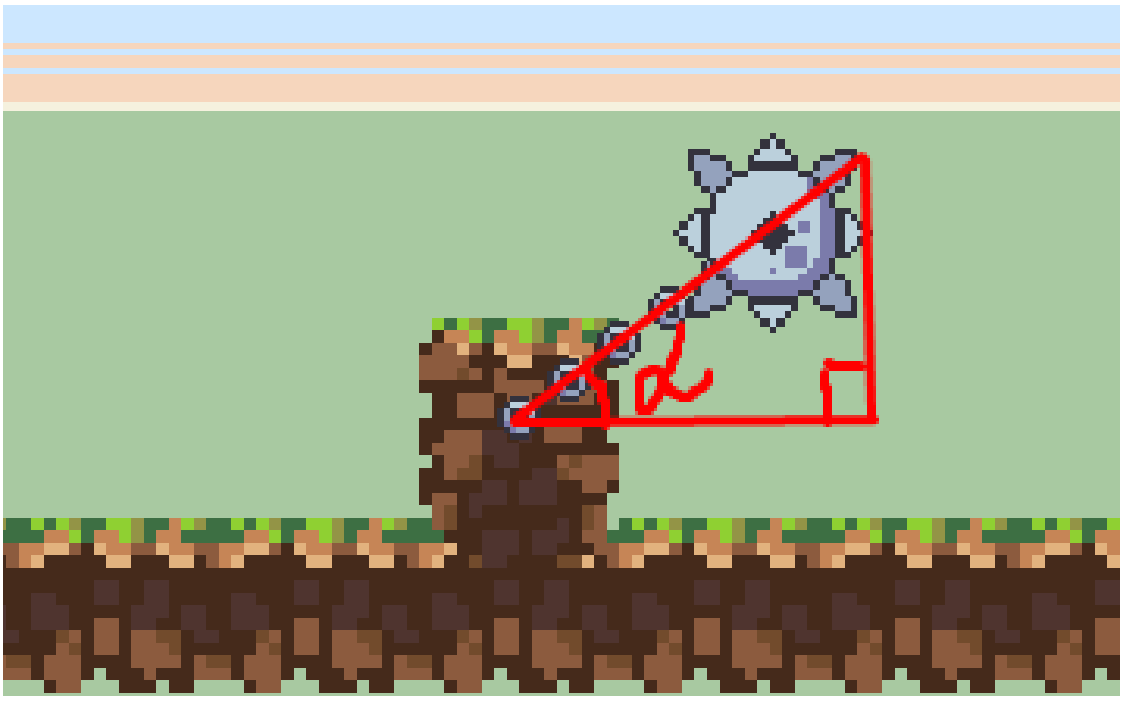


Рисунок 3.4 – Розрахунок кута для правильної роботи обертової пастки

3.3.4 Алгоритм роботи гравця (Player)

Для роботи алгоритму гравця (персонаж, яким маніпулює гравець) передбачено декілька його станів: «мертвий», «стрибок», «падіння», «біг»,

«атака», «ковзання по стіні» та «в спокої». Від них залежить анімація: для кожного зі станів своя. Якщо стан героя «мертвий» (кількість його здоров'я досягнула 0), його швидкість зменшується до 0 та програється анімація смерті, характеристики (здоров'я) в цьому стані також перестає змінюватись, можливість отримувати шкоду блокується. Зміна ж інших станів відбувається завдяки маніпуляціям користувача (кнопки руху), а також взаємодіям гравітації та дотику. Горизонтальний дотик реалізовано в залежності від напрямку руху гравця: якщо він рухається ліворуч і його хітбокс стикається з перешкодою, відбувається переміщення крайнього лівого положення хітбоксу гравця, до крайнього правого положення моделі об'єкту, з яким гравець зіткнувася, і навпаки для руху праворуч. Для колізій вертикальних використовується така ж логіка, але на вертикальний рух також діє гравітація, тобто перевіряється, чи гравець знаходиться на твердому блоці, якщо ж ні – виконується додавання певного значення, що залежить від dt (змінна для позначення часового кроку в pygame), таким чином, гравця якби «тягне» вниз. Для стрибка використовується зміна значення напрямку по осі «у». «Ковзання по стіні» активується, коли гравець у стрибку або падінні торкається вертикальної поверхні боком і продовжує утримувати клавішу руху у бік стіни. У цьому стані вертикальна швидкість обмежується до фіксованого «повільного» значення (приблизно 30% від максимальної швидкості падіння), а гравітація частково компенсується, тому герой повільно ковзає вниз замість вільного падіння. Після переходу в «ковзання по стіні» таймер стрибків скидається, що дає змогу одразу виконати «стрибок зі стіни». «Стрибок зі стіни» спрацьовує, якщо під час «ковзання по стіні» гравець натискає кнопку стрибка. Алгоритм змінює вертикальну складову швидкості на звичний початковий імпульс стрибка, а горизонтальну - на значення, обернене напрямку стіни (тобто відштовхування назад). Одночасно змінюється стан на «стрибок»; при цьому активується звичайний таймер стрибка. Анімаційно «стрибок зі стіни» супроводжується окремою фазою відштовхування та звуковим ефектом. Для опрацювання шкоди,

використовується таймер (екземпляр класу Timer) для активації періоду невразливості (супроводжується зміною кольору моделі на білий). При отриманні шкоди (окрім отримання шкоди від води та деяких пасток), гравець виконує невеликий стрибок, активується таймер невразливості, здоров’я зменшується на кількість, що залежить від джерела шкоди. У випадку ж, якщо модель гравця торкається моделі води, йому надається статус «задихається», при якому він отримує періодичну шкоду, його швидкість та гравітація зменшується вдвічі. Отримання шкоди, стрибок супроводжуються звуковим супроводом.

3.3.5 Алгоритм роботи зміни сцени (Passage)

Механізм зміни сцен в застосунку реалізований через клас Passage та методи класу Main: manage_routes, switch і create_level. Плавний перехід досягається завдяки методу display класу Passage: під час його роботи створюється круг із радіусом, рівним половині діагоналі вікна, у якого поступово збільшується ширина контуру. У момент, коли контур повністю заповнює весь видимий простір, викликаються функції manage_routes та create_level, після чого контур починає звужуватися до повного зникнення (рис. 3.5). Функція manage_routes відповідає за керування станом активності сцен: вона визначає, яку сцену потрібно деактивувати, а яку - активувати. Функція create_level виконується лише в тому випадку, якщо у запиті на перехід було передано «сітку» (список із даними про об’єкти на рівні) для створення рівня. Про необхідність цього їй «сповіщає» функція switch, яка сама викликається з різних сцен. Крім того, у функцію switch передається параметр у вигляді списку з інформацією про те, які сцени потрібно змінити, а вона вже «передає» ці дані до manage_routes, забезпечуючи коректний перехід між станами.

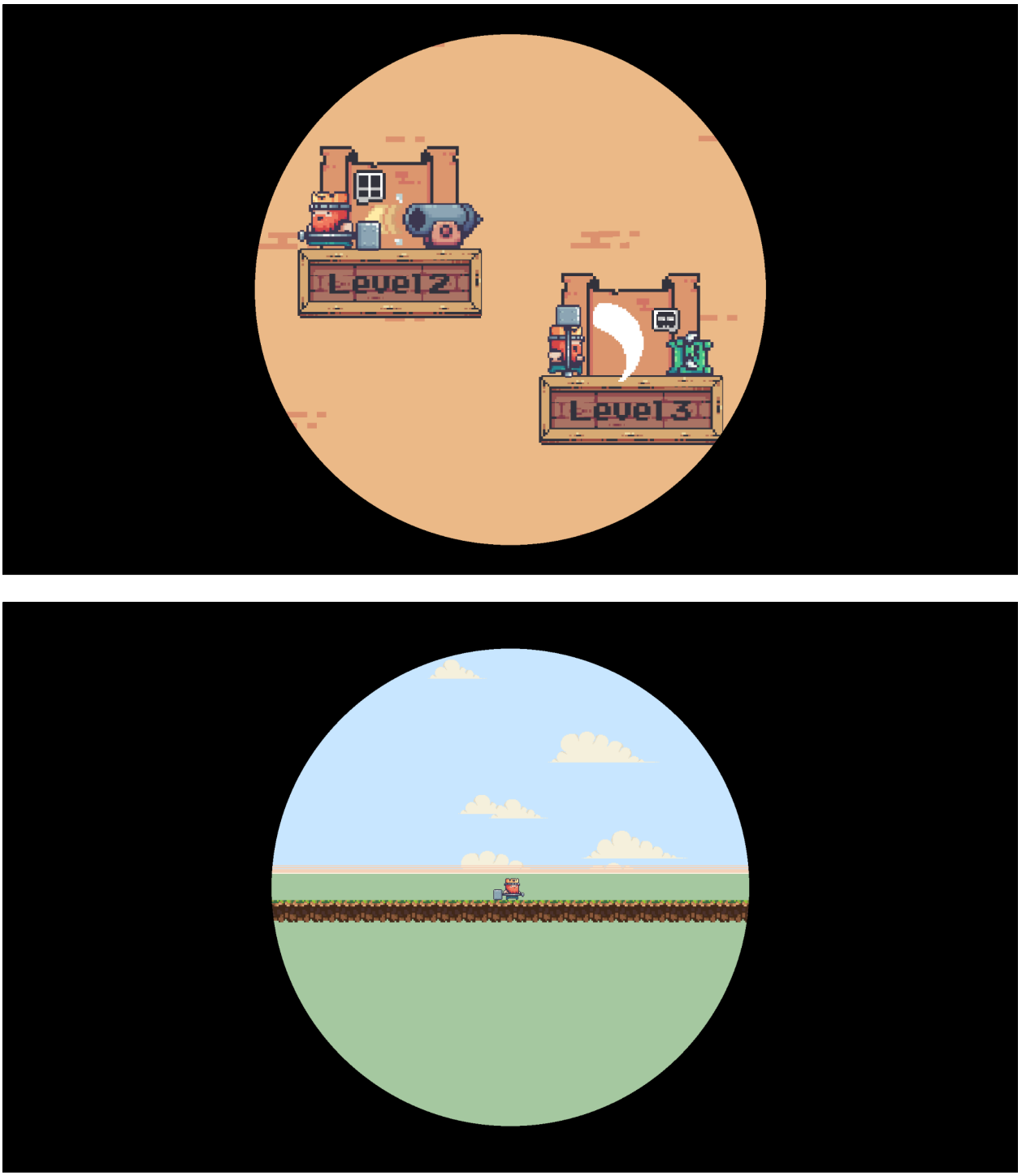


Рисунок 3.5 – Процес переходу з меню рівнів на рівень

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.1.

Таблиця 3.2 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування

Продовження таблиці 3.2

1	PyCharm	Інтегроване середовище розробки (IDE) для Python від JetBrains, яке використовувалось як основний інструмент розробки. Воно надає розширені можливості для роботи з Python кодом: розумне автодоповнення, аналіз синтаксису, інструменти налагодження, інтеграцію з Git та підтримку віртуальних середовищ. IDE також підтримує популярні фреймворки (Django, Flask). У цьому проекті PyCharm довів свою ефективність під час роботи з великою кількістю коду.
2	Github	GitHub вебплатформа для керування Git репозиторіями, яка використовувалася для спільної розробки. Вона надає інструменти для зберігання коду, контролю версій, створення гілок та обговорення змін через пул-реквести. Платформа спростила колаборацію, дозволяючи синхронізувати роботу над кодом та оперативно вносити правки.
3	Draw.io	Онлайн-інструмент для побудови діаграм.
4	Bpmn.io	Онлайн-інструмент для побудови моделей за нотацією BPMN.

Таблиця 3.3 – Опис бібліотек

№ п/п	Назва бібліотек	Опис застосування
1	pygame	Бібліотека для розробки ігор та мультимедійних додатків на Python. Надає низку інструментів для роботи з 2D-об’єктами, обробки звуку, керування вводом користувача (клавіатура, миша, джойстик) та реалізації ігрової логіки. Використовувалась як основний інструмент для створення ігрового рушія, обробки зіткнень, рендерингу спрайтів та реалізації ігрових механік. Бібліотека дозволила ефективно працювати з анімаціями, фізикою об'єктів та ігровим циклом.

Продовження таблиці 3.3

2	pygame.math: Vector2	Спеціалізований клас для роботи з векторами, що використовувався для математичних розрахунків у проекті. Надає базові операції: додавання/віднімання векторів, множення на скаляр, нормалізацію, обчислення довжини та скалярного добутку. Особливо корисний для реалізації фізики руху, траєкторій снарядів, напрямків анімації та інших ігрових механік, де потрібні точні векторні обчислення. Оптимізована реалізація класу дозволила ефективно працювати з просторовими координатами без необхідності створювати власні математичні структури.
3	pickle	Забезпечує засоби для серіалізації й десеріалізації об’єктів Python. Використовувався для збереження об'єктів у файлі
4	os	Забезпечує можливість взаємодії з операційною системою. Містить функції для роботи з файловою системою, обробки змінних середовища та виконання інших системних дій. Застосовувався для вказування абсолютних шляхів до графічних ресурсів.
5	random	Містить функціонал для створення випадкових чисел і виконання випадкового вибору з набору елементів.
6	datetime	Надає можливості для опрацювання дати й часу в Python. Містить класи та функції для маніпуляцій з датами, часовими інтервалами та іншими складовими часового виміру.
7	pygame.mouse: get_pos	Функція get_pos з модуля pygame.mouse повертає актуальні координати положення курсора миші на екрані.
8	pygame.image: load	Функція load з модуля pygame.image використовується для завантаження графічного зображення з файлу у формат, придатний для подальшого використання в грі.

Продовження таблиці 3.3

9	sys	Забезпечує доступ до системних параметрів і функцій. Використовувався для виконання операцій з видалення файлів.
10	pygame.mouse: get_pressed	Функція get_pressed з модуля pygame.mouse повертає кортеж, який відображає стан кнопок миші — зокрема, чи натиснуті ліва, права або середня кнопка.

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

3.4 Аналіз безпеки даних

Однією з задач у межах проєкту було забезпечення попередньої перевірки рівнів, які надсилаються користувачами, перед їх збереженням у базі даних. Оскільки рівень у грі представляється у вигляді складного списку, що містить словник з декількома шарами об’єктів та координатами їх розміщення, важливо гарантувати коректність, безпечність та відповідність структури очікуваному формату.

Для цього було реалізовано функцію validate_level_data, яка проводить детальну перевірку кожного шару, координат та значень, щоб уникнути навмисно некоректних або небезпечних даних. Зокрема, вона перевіряє, що структура відповідає очікуваному типу - список з двох елементів, де перший є словником відомих шарів, а другий - список із двох цілих чисел, що позначає зсув. Усі ключі всередині кожного шару перевіряються на відповідність формату координат, а значення - на тип (наприклад, int або str) згідно з очікуваннями для відповідного шару.

Функція також реалізує захист від потенційно шкідливих або експлуатаційних конструкцій. Вона відкидає значення, що містять небезпечні ключові слова (<script>, __class__, lambda), або мають надмірну довжину чи надто великі числові значення, які можуть спричинити помилки або

призвести до відмови в обслуговуванні. Крім того, встановлені жорсткі обмеження на кількість об'єктів у кожному шарі та загальну кількість об'єктів на рівні, що запобігає перевантаженню системи зловмисними запитами.

Таким чином, ще до етапу збереження даних у базі або обробки на стороні гри, перевірка дозволяє ефективно виявити та заблокувати шкідливі або некоректно сформовані рівні. Це рішення покращує стабільність, забезпечує відповідність даних внутрішньому протоколу гри та підвищує безпеку взаємодії між клієнтами і сервером без ускладнення логіки програми.

Висновки до розділу

У цьому розділі було детально проаналізовано архітектуру та етапи розробки програмного забезпечення комп’ютерної гри жанру Platformer, з розробкою редактора рівнів і систему обміну рівнями.

Було наведено обґрунтування вибору ігрового рушія Pygame, що надає широку платформну підтримку, розширений набір інструментів для реалізації ігрових механік і графіки, а також дозволяє повністю контролювати архітектуру проєкту. Крім того, вибір Pygame зумовлений активною спільнотою розробників, що полегшує розв’язання технічних труднощів у процесі розробки.

Для реалізації збереження рівнів було обрано бібліотеку Pickle, оскільки вона дозволяє швидко серіалізувати складні Python-об’єкти, включно зі словниками, списками та класами, без необхідності ручного перетворення у формат JSON. Це забезпечує простоту збереження та завантаження даних, а також високу сумісність з внутрішніми структурами редактора рівнів.

Процес побудови програмного забезпечення охоплював створення авторських класів і нестандартних підходів, призначених для впровадження характерних ігрових функцій, таких як динамічні вороги, статичні вороги та пастки. Були наведені алгоритми їх роботи з детальним поясненням більшості функціоналу. До переліку основних інструментів, задіяних у

процесі реалізації проєкту, належать такі засоби: Pygame - для проєктування рівнів та ігрових об’єктів, а також Pycharm - для написання програмного коду. Крім того, було описано алгоритм перевірки файлів рівнів перед їхнім завантаженням на сервер з метою подальшого обміну.

У результаті застосування обраних інструментів та підходів до розробки вдалося реалізувати повноцінну гру з оригінальними ігровими механіками та різнопланових ворогами на рівнях, що гарантує гравцям захопливий і динамічний ігровий процес.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Метою тестування є наступне:

− перевірка коректності функціонування програмного забезпечення відповідно до встановлених функціональних вимог;

− виявлення проблем, помилок і недоліків з подальшим їх усуненням;

− оцінювання зручності та ергономічності графічного інтерфейсу.

Метриками для оцінки якості ПЗ обрано наступні:

Ефективність (Efficiency) – відображає ступінь оптимізації програмного забезпечення. Висока ефективність досягається завдяки дотриманню стандартів програмування, уникненню дублювання коду та усуненню надлишкових елементів. Отриманий бал 100 свідчить про чудову оптимізацію та відповідність найкращим практикам написання коду.

Портативність (Portability) – показує, наскільки легко програмне забезпечення може бути перенесене між різними операційними системами або середовищами. Оцінка 100 означає, що застосунок повністю готовий до роботи на кількох платформах без необхідності внесення змін у код.

Надійність (Reliability) – оцінює здатність програмного забезпечення продовжувати роботу в умовах відмов, а також його стійкість до збоїв і можливість відновлення після них. Показник 100 вказує на високу стійкість та безвідмовну роботу системи.

Аналізованість (Analyzability) – визначає легкість виявлення причин помилок, аналізу проблем у коді та ідентифікації фрагментів, які потребують доопрацювання. Максимальна оцінка 100 демонструє зручність у супроводі та діагностиці.

Функціональність (Functionality) – характеризує здатність програмного забезпечення відповідати функціональним вимогам і коректно виконувати

покладені на нього задачі. Бал 93 свідчить про високу якість реалізації функцій, однак наявність 9 проблем вказує на потенційні недоліки, які варто усунути.

Концептуальна цілісність (Conceptual Integrity) – відображає єдність і логічну узгодженість архітектури програмного забезпечення, зокрема стилю програмування, організації модулів та іменування. Значення 97 підтверджує високу структурну якість і послідовність у реалізації.

Стійкість до помилок (Robustness) – описує здатність системи працювати коректно навіть у випадках неочікуваних або помилкових вхідних даних. Оцінка 58 вказує на допустимий рівень, однак велика кількість виявлених проблем (44) свідчить про потребу в підвищенні стабільності при нестандартних ситуаціях.

Ці метрики були отримані у результаті використання сервісу embold.io[19].

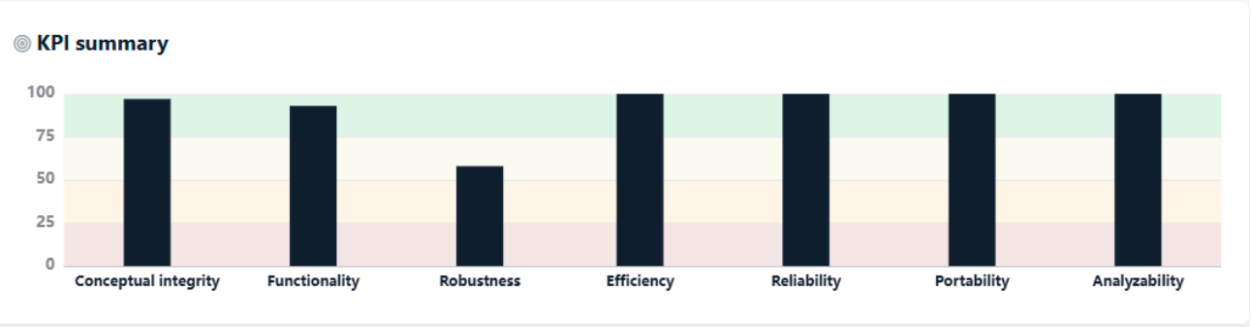


Рисунок 4.1 – Результати обчислень метрик

Метрики демонструють високу загальну якість програмного забезпечення, зокрема в частині ефективності, переносимості, надійності та легкості в обслуговуванні. Основним напрямом для покращення є робастність, оскільки наявність значної кількості пов’язаних проблем може впливати на стабільність застосунку в умовах помилок або нестандартних сценаріїв використання.

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 – 4.15.

Таблиця 4.1 – Тест 1.1 Перехід у меню готових рівнів

Початковий стан системи	Користувач знаходиться у головному меню
Вхідні дані	-
Опис проведення тесту	Користувач здійснює натискання на кнопку «Play» у головному меню гри.
Очікуваний результат	Відображається меню з готовими рівнями.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Перехід у редактор рівнів

Початковий стан системи	Користувач знаходиться у головному меню
Вхідні дані	-
Опис проведення тесту	Користувач здійснює натискання на кнопку «Editor» у головному меню гри.
Очікуваний результат	Відкривається редактор рівнів.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3– Тест 1.3 Перехід у меню власних рівнів

Початковий стан системи	Користувач знаходиться у головному меню
Вхідні дані	-

Продовження таблиці 4.3

Опис проведення тесту	Користувач здійснює натискання на кнопку «Load» у головному меню гри.
Очікуваний результат	Відображається меню з власними рівнями.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4– Тест 1.4 Вихід з гри

Початковий стан системи	Користувач знаходиться у головному меню
Вхідні дані	-
Опис проведення тесту	Користувач здійснює натискання на кнопку «Quit» у головному меню гри.
Очікуваний результат	Гра закривається.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5– Тест 1.5 Запуск готового рівня

Початковий стан системи	Користувач знаходиться у меню готових рівнів
Вхідні дані	-
Опис проведення тесту	Користувач здійснює натискання на обраний готовий рівень.

Продовження таблиці 4.5

Очікуваний результат	Програється анімація переходу між меню готових рівнів та самим рівнем. Меню готових рівнів змінюється на сцену рівня. На екрані з’являється головний персонаж, об’єкти рівню і ігровий інтерфейс. Користувач отримує можливість керувати персонажем.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6– Тест 1.6 Встановлення паузи, меню паузи

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Користувач здійснює натискання на клавішу «P», або натискає ЛКМ по кнопці паузи в ігровому інтерфейсі.
Очікуваний результат	Ігровий процес припиняється. На екрані з’являється меню паузи.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7– Тест 1.7 Керування персонажем

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Користувач здійснює натискання на одну з клавіш: «←»;«→»;«↓»;«Space»;«A».

Продовження таблиці 4.7

Очікуваний результат	Ігровий персонаж виконує закріплену за клавішею дію:«←» - рух у ліво; «→» - рух у право; «↓» - рух вниз; «Space» - стрибок; «A» - атака.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8– Тест 1.8 Зниження ворожого снаряду

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Ворожий снаряд знаходиться у межах досягнення атаки ігрового персонажа. Користувач натискає на кнопку «A», що викликає атаку головного персонажа.
Очікуваний результат	Ворожий снаряд знищується не завдавши шкоди ігровому персонажу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9– Тест 1.9 Зміна напрямку ворога-свині

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Ворог-свиня знаходиться у межах досягнення атаки ігрового персонажа і йде у його напрямку. Користувач натискає на кнопку «A», що викликає атаку головного персонажа.

Продовження таблиці 4.9

Очікуваний результат	Ворог-свиня розвертається у протилежному напрямку не завдавши шкоди ігровому персонажу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10– Тест 1.10 Вбивство ворога-свині

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Ігровий персонаж виконує «наскок» на ворога-свиню, що призводить до того що нижня частина модельки ігрового персонажа зіштовується з верхньою частиною модельки ворога-свині.
Очікуваний результат	Ігрового персонажа підкидає вгору від ворога-свині Ворог-свиня програє анімацію смерті, після чого знищується. Ігровий персонаж не отримує шкоду за цю дію.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11– Тест 1.11 Збір монети

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Ігровий персонаж зіштовхується з монетою.

Продовження таблиці 4.11

Очікуваний результат	Монета програє анімацію підбору, після чого знищується. Якщо монета була золотою, рахунок персонажа збільшується на 50, срібною - на 10. Інформація про поточний рахунок персонажа оновлюється і відображається на інтерфейсі.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12– Тест 1.12 Взаємодія з водою

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Ігровий персонаж опиняється у воді.
Очікуваний результат	Ігровий персонаж сповільнюється і отримує переодичний урон під час знаходження у воді. У випадки якщо здоров’я персонажа падає до 0, ігровий персонаж вмирає і виводиться меню програшу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13– Тест 1.13 Вихід за межі рівня

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-

Продовження таблиці 4.13

Опис проведення тесту	Ігровий персонаж опиняється знизу найнижчого блоку землі на рівні.
Очікуваний результат	Здоров’я ігрового персонажа стає рівним 0, через що він вмирає. Виводиться меню програшу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14– Тест 1.14 Збір зілля здоров’я

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Зілля здоров’я програє анімацію підбору, після чого знищується. Поточний запас здоров’я персонажа зростає на 20. Запас здоров’я не може бути більше за 100.
Очікуваний результат	Здоров’я ігрового персонажа стає рівним 0, через що він вмирає. Виводиться меню програшу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.15– Тест 1.15 Проходження рівня

Початковий стан системи	Користувач знаходиться у процесі проходження рівня. Ігровий персонаж не мертвий.
Вхідні дані	-
Опис проведення тесту	Персонаж зібрав 3 кристали у процесі проходження рівня.

Продовження таблиці 4.15

Очікуваний результат	Рівень вважається пройденим. Виводиться меню перемоги.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

У якості контрольного прикладу було обрано проходження заготовленого рівня.

Після запуску застосунку, натискаємо на кнопку готових рівнів в головному меню (рис. 4.2).

Рисунок 4.2 – Головне меню

Після чого користувач опиняється у меню готових рівнів. Тут йому треба обрати рівень що він бажає пройти. Нехай це буде перший рівень (рис. 4.3).

Рисунок 4.3 – Меню готових рівнів

Після того як закінчилась анімація переходу, на екрані з’являється перший рівень гри, наповнений ворогами і інтерактивними об’єктами. Користувачу надається можливість керувати персонажем. Розпочинається проходження рівня (рис. 4.4).

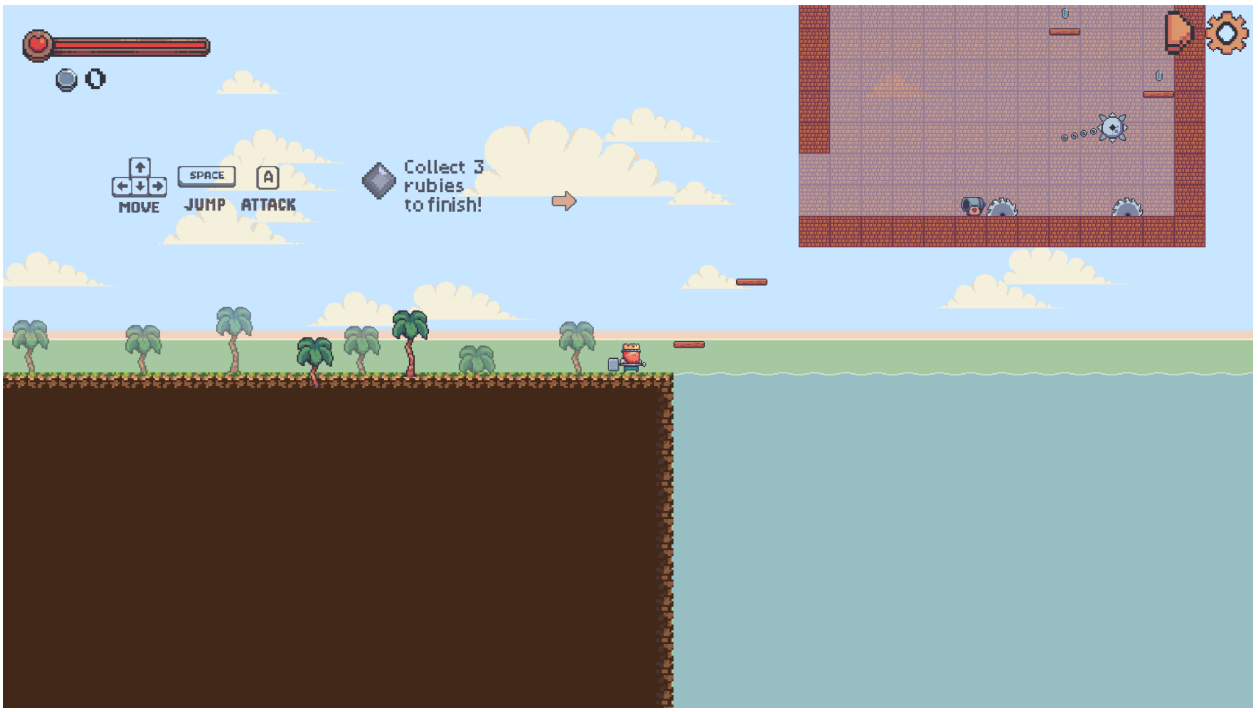


Рисунок 4.4 – Перший рівень

Під час проходження рівня користувачу будуть зустрічатись різні вороги, що можуть завдати йому шкоди у разі якщо ігровий персонаж попаде під їх атаку(рис. 4.5).

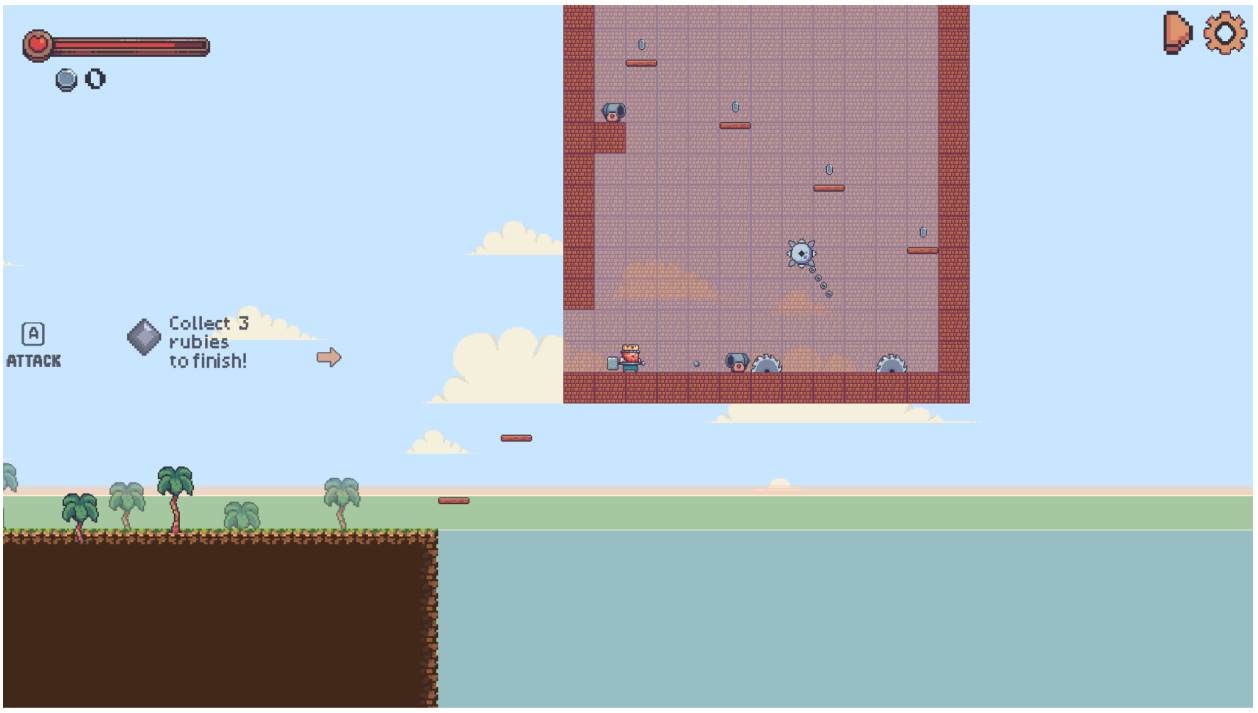


Рисунок 4.5 – Атака ворогів

В будь-який момент проходження, користувач може викликати меню паузи, натиснувши клавішу «P», або іконку паузи на ігровому інтерфейсі, що призупинить проходження і виведе основну інформацію (кількість зібраних діамантів, score – зібрану кількість монет) з можливістю продовжити або перезапустити рівень (рис. 4.6). Також користувач може повернутися до головного меню натиснувши на іконку шестерні у правому верхньому куті.

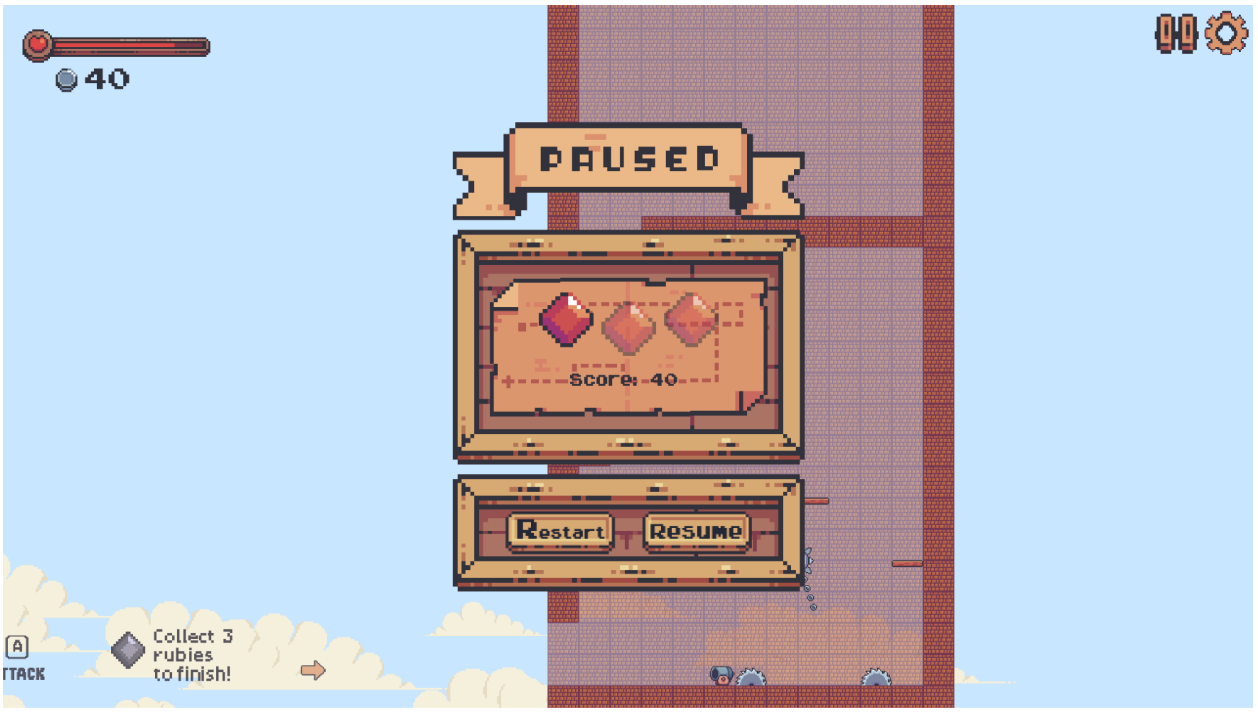


Рисунок 4.6 – Меню паузи

Користувач може керувати ігровим персонажем натискаючи на клавіші. Ігровий персонаж виконує закріплену за клавішею дію:«←» - рух у ліво; «→» - рух у право; «↓» - рух вниз; «Space» - стрибок; «A» - атака.

Користувач має змогу протистояти ворогам, атакуючи їх, знищуючи їх самих або їх снаряди. Наприклад, користувач може виконати «наскок» на ворога-свиню тим самим знищуючи його(рис. 4.7), або знищити снаряд гармати атакою ігрового персонажа(рис. 4.8).

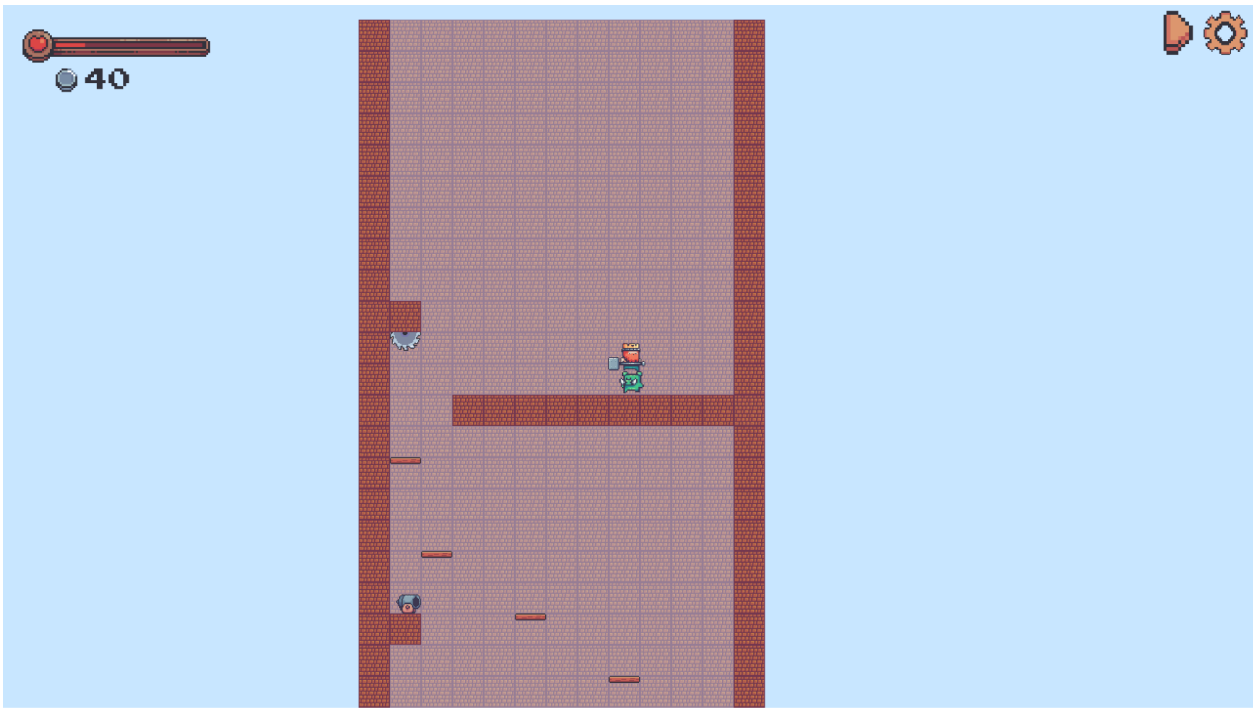


Рисунок 4.7 – Знищення ворога-свині

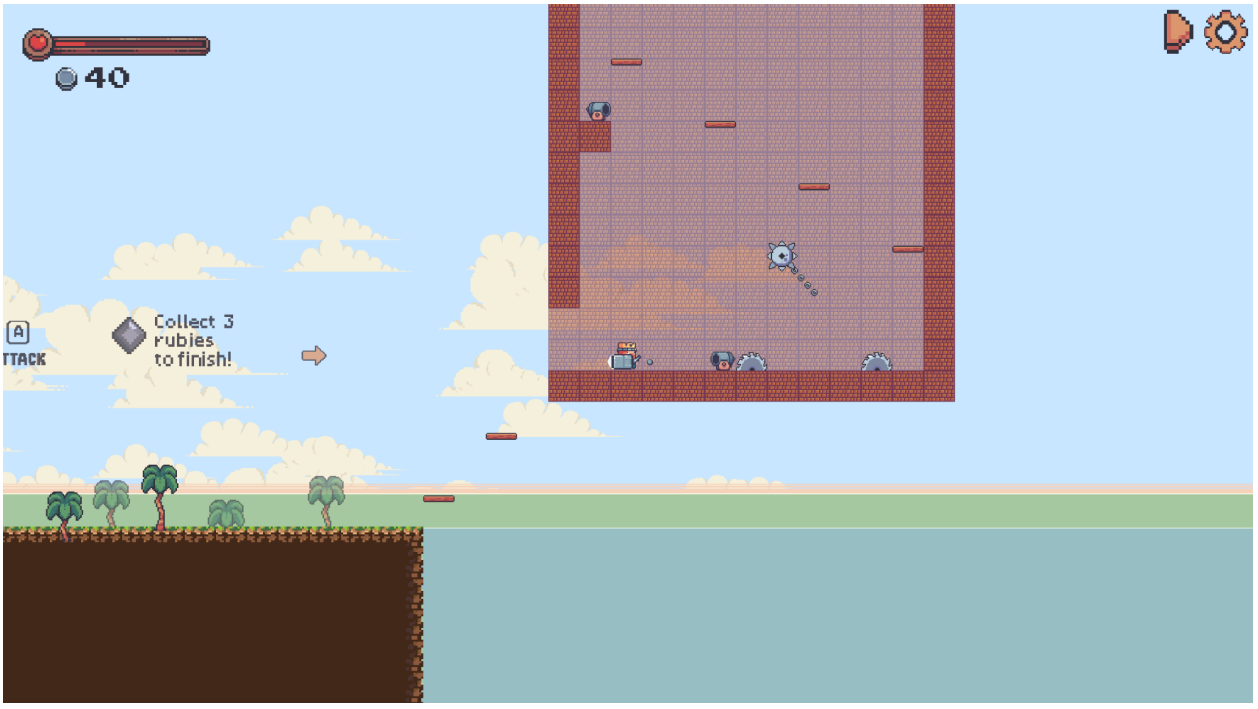


Рисунок 4.8 – Знищення снаряду гармати

Якщо користувач буде не обережним, він може також отримати шкоду від пасток розташованих на рівні. Якщо здоров'я ігрового персонажа впаде нижче нуля, він помре і користувач побачить меню програшу(рис. 4.9).

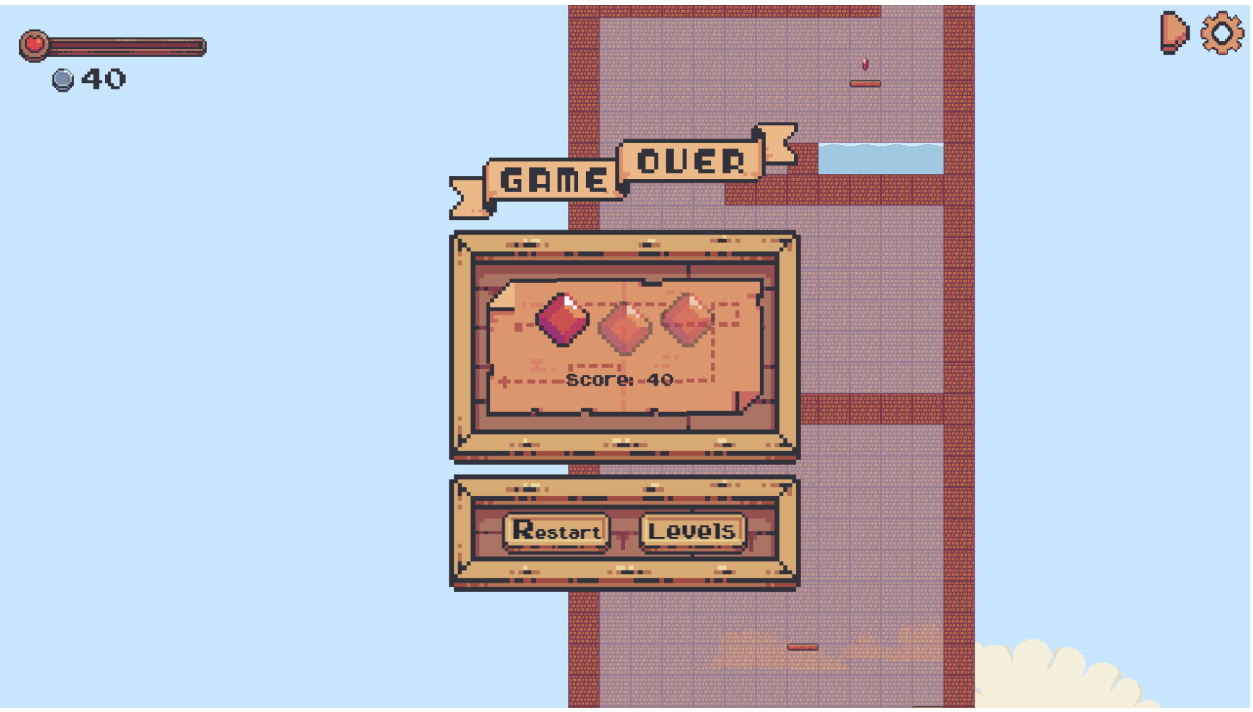


Рисунок 4.9 – Меню програшу

За різновид пастики також можна вважати воду розташовану на рівні. Якщо ігровий персонаж потрапляє у неї, це уповільнити його рухи, а також персонаж буде отримувати періодичну шкоду під час знаходження у воді(рис. 4.10).

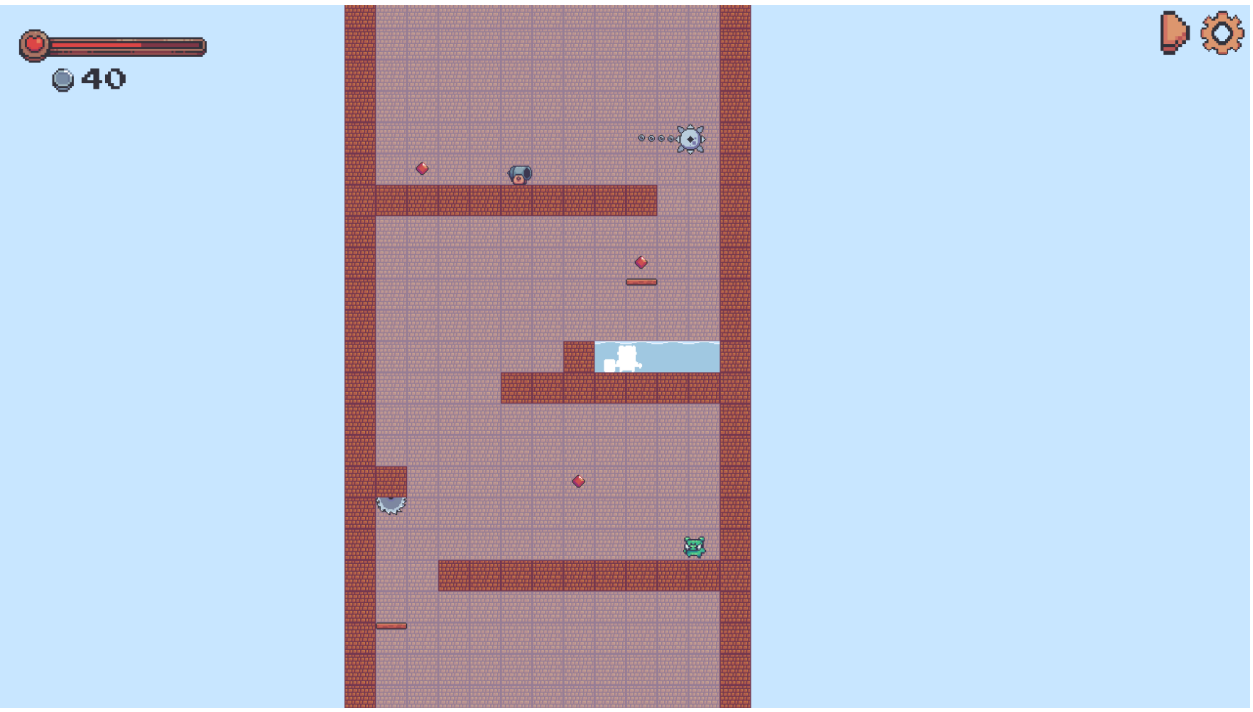


Рисунок 4.10 – Взаємодія з водою

Також під час проходження користувач може збирати ігрові об’єкти. Підбір монет збільшує рахунок персонажа. Золота монета варта 50 очок, а срібна 10. Збір зілля здоров’я відновить певну кількість втраченого здоров’я. Збір кристалів буде наближати користувача до повного проходження рівня. Задля перемоги, користувач повинен зібрати 3 кристали. При виконанні цієї умови, на екран вивдеться меню перемоги, де користувач побачить свій рахунок і може перепройти рівень(«Restart»), або повернутися до меню рівнів («Levels»)(рис. 4.11).

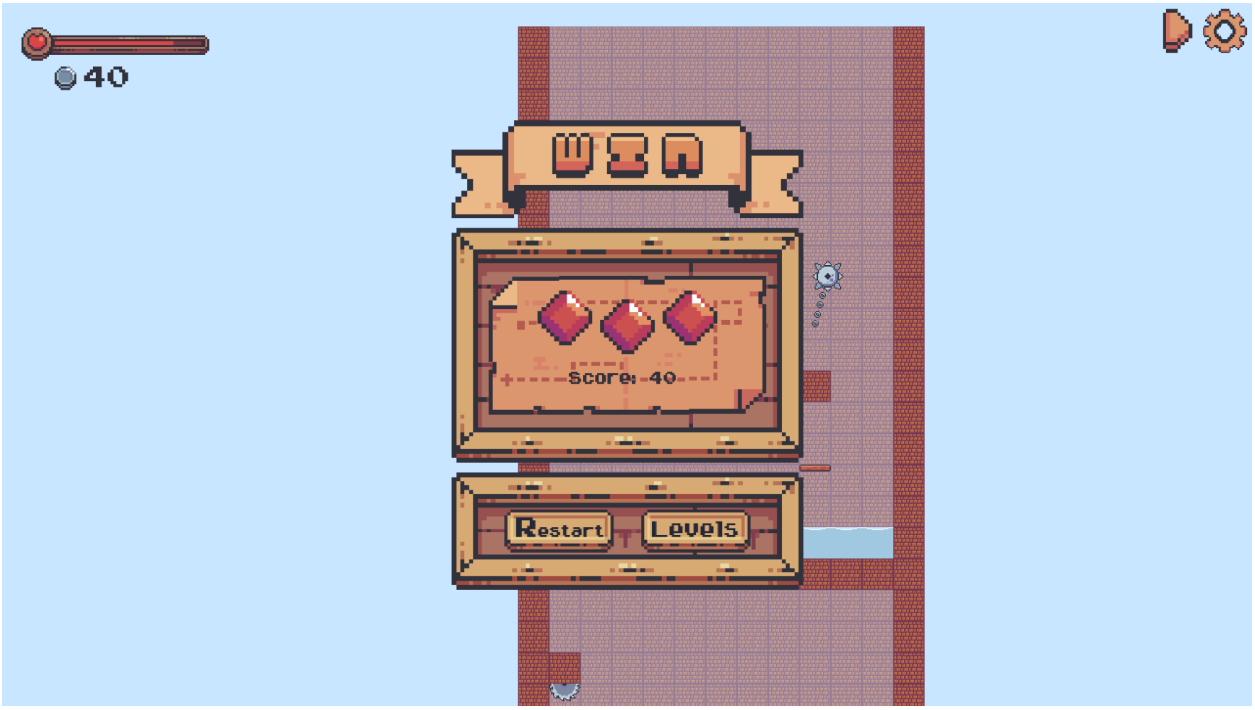


Рисунок 4.11 – Перемога

Висновки до розділу

У цьому розділі було виконано тестування класичної гри у жанрі платформер, з розробкою редактора рівнів і системою обміну рівнями й проведено аналіз її якості.

Були проаналізовані наступні метрики якості ПЗ: ефективність, портативність, надійність, аналізованість, функціональність, концептуальна цілісність та стійкість до помилок. Отримані оцінки демонструють високу загальну якість програмного забезпечення.

Крім того, було здійснено й детально описано мануальне тестування такого функціоналу: ігрові меню, встановлення пузи, система керування персонажем, взаємодія з ворожими об’єктами, взаємодія з ігровими об’єктами та проходження рівня.

І нарешті був описаний контрольний приклад проходження заготовленого рівня.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання комп’ютерної гри, розробленої на бібліотеці Pygame, використовувалася утиліта PyInstaller, яка дозволяє зібрати Python-застосунок у вигляді окремого виконуваного файлу.

Команда для створення виконуваного файлу мала такий вигляд:

client/main.py --name LevelKing --icon=LevelKing.ico --noconfirm --noconsole --onefile --add-data "client/graphics;graphics" --add-data "client/audio;audio" --add-data "client/levels;levels" --add-data "client/saved_levels;saved_levels"

У результаті виконання цієї команди формується єдиний .exe файл, який можна запускати на ОС Windows без встановлення інтерпретатора Python.

Для створення інсталятора гри було використано Inno Setup Compiler[20], який дозволяє згенерувати інсталяційний .exe файл, що встановлює гру на комп’ютер користувача (рис. 5.1).

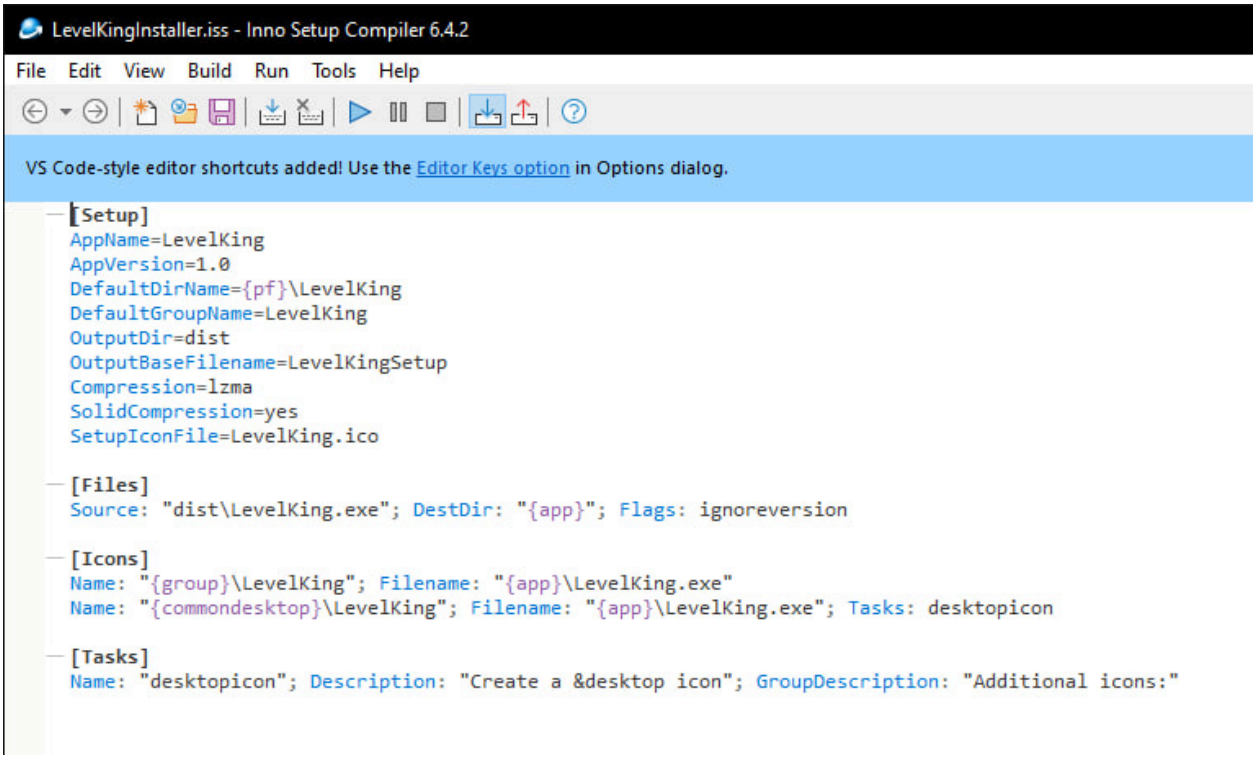


Рисунок 5.1 – Файл конфігурації для Inno Setup

У конфігураційному файлі було вказано основні параметри, зокрема:

- назву гри (AppName);
- виконуваний файл (OutputBaseFilename);
- шлях до виконуваного файлу (dist/LevelKing.exe);
- створення ярлика на робочому столі.

У результаті генерується інсталятор LevelKingSetup.exe, який встановлює гру в директорію Program Files\LevelKing з відповідним значком і ярликом (рис. 5.2).

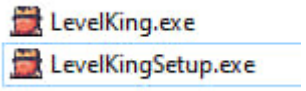


Рисунок 5.2 – Виконуваний файл та інсталятор гри

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі «Керівництво користувача»[КПІ.ІП-1217.045480.05.34].

Публікація оновлень гри виконується через сервіс GitHub Releases, який дозволяє зручно розповсюджувати нові версії гри серед користувачів.

Для створення випуску необхідно:

- задати тег версії (наприклад, v1.0.0);
- написати опис оновлення;
- додати інсталяційний файл LevelKingSetup.exe як вкладення.

Приклад оформлення релізу наведено на рисунку 5.3.

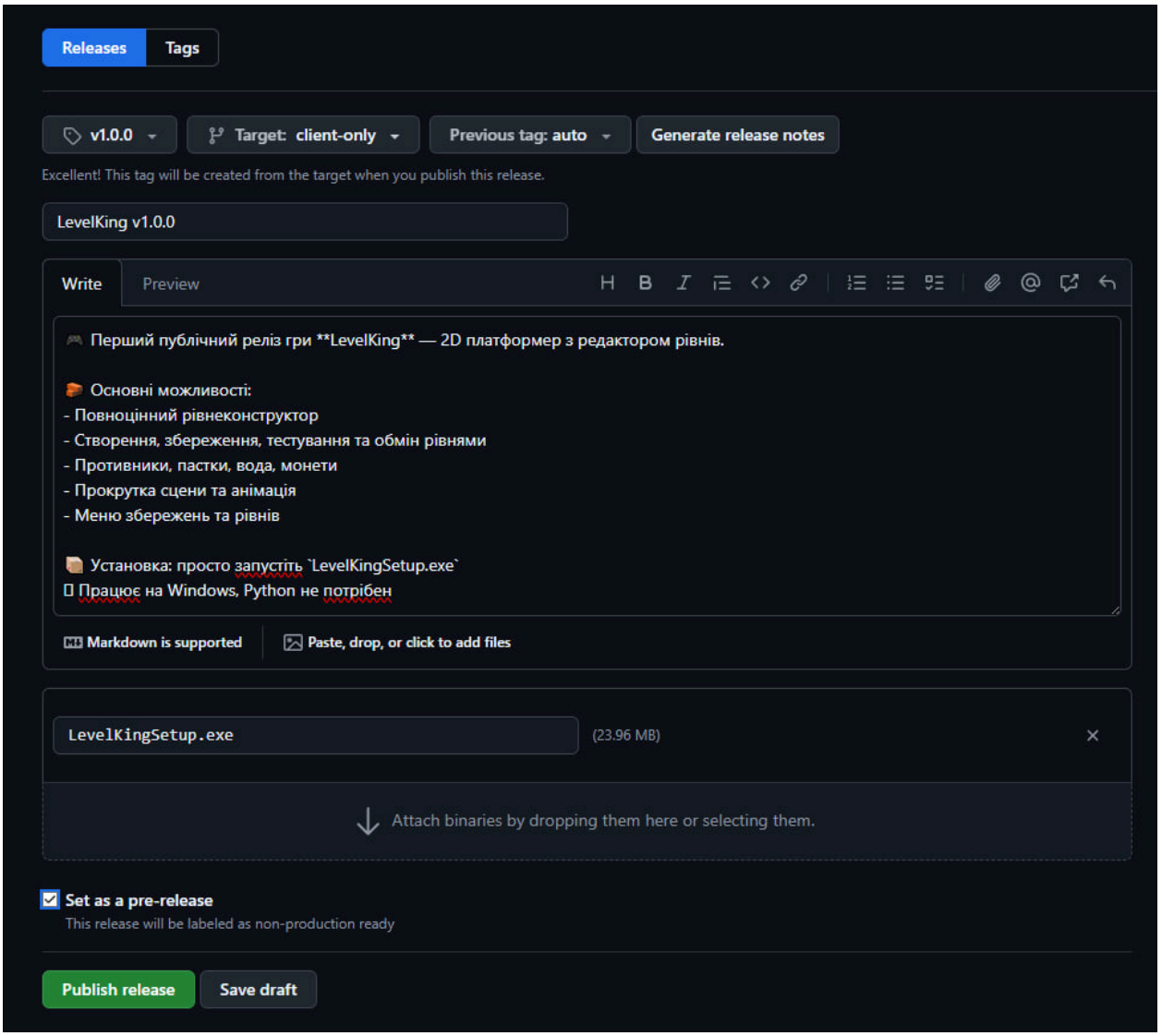


Рисунок 5.3 – Інтерфейс створення релізу на GitHub

Після натиснення кнопки «Publish release», реліз стає доступним для завантаження всіма користувачами.

Висновки до розділу

У цьому розділі було описано процес розгортання та супроводу гри LevelKing, реалізованої з використанням Pygame.

Розгортання здійснювалося за допомогою утиліти PyInstaller, що дозволило створити самостійний .exe файл для Windows.

Для зручного встановлення гри користувачами було створено інсталятор за допомогою Inno Setup.

Публікація нових версій гри здійснюється через GitHub Releases, що забезпечує доступність оновлень та легке поширення програмного забезпечення серед користувачів.

ВИСНОВКИ

В ході виконання дипломного проєкту було розроблено класичну гру у жанрі Platformer з редактором рівнів та системою обміну рівнями між користувачами.

Таким чином, головна мета цього проєкту, а саме сприяння розвитку логічного та стратегічного мислення гравця, була повністю досягнута завдяки реалізації наступного функціоналу:

- була реалізована система керування ігровим персонажем шляхом натискання закріплених за певними діями клавіш, а також були розроблені основні закони фізики рухів у грі, що дозволяють вдало маневрувати ігровим персонажем і взаємодіяти з ігровим світом;
- був створений ігровий світ, наповнений різноманітними інтерактивними об’єктами такими як: різні типи ворогів, що вимагають унікального підходу для проходження крізь них або перемоги над ними; пастки, що вимагають стратегічного мислення користувача для їх обходу; об’єкти-платформи, що перевіряють спритність користувача; монети та кристали, що збільшують рахунок персонажа і наближають його до перемоги; зілля здоров’я, що можуть врятувати персонажа у складній ситуації; об’єкти заднього плану, що дозволяють наповнити рівень, зробиши його більш «живим»;
- був розроблений зручний користувацький інтерфейс що включає в себе: головне меню; меню готових рівнів; меню власних рівнів; меню для обміну рівнями;меню паузи; меню програшу; меню перемоги; ігровий інтерфейс, що відображає корисну поточну інформацію під час проходження рівня;
- була розроблена система збереження рівнів, створених у редакторі рівнів, та система безпечного поширення цих рівнів. Ці системи дозволяють перепроходити власні рівні, а також без опаски завантажувати рівнів інших користувачів, для подальшого проходження;

Також варто зазначити, що у цьому проєкти наявна доцільність продовження його розробки, так як у результаті було отримано універсальний застосунк, в який з легкістю можна додавати нові аспекти і внутрішньо-ігрові механіки. Тим самим, гра буде постійно вдосконалюватися, розширювати свій функціонал та ігровий простір, що сприятиме підвищенню інтересу гравців та забезпечить її актуальність у довгостроковій перспективі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What are computer applications? *Wrexham University*. URL: https://online.wrexham.ac.uk/what-are-computer-applications/ (дата звернення 10.05.2025).

2. Video Game. *PCMag*. URL: https://www.pcmag.com/encyclopedia/term/video-game (дата звернення 10.05.2025).

3. What is a Platform Game? *Lifewire*. URL: https://www.lifewire.com/what-is-a-platform-game-812371 (дата звернення 10.05.2025).

4. The Definition of Windows Operating System: History, Functions, and Features. *Telkom University*. URL: https://it.telkomuniversity.ac.id/en/windows-operating-system/ (дата звернення 10.05.2025).

5. How Many Gamers Are There in 2025? (Worldwide Statistics). *Demandsage*. URL: https://www.demandsage.com/gamers-statistics/ (дата звернення 10.05.2025).

6. Geometry Dash. *Steam*. URL: https://store.steampowered.com/app/322170/Geometry_Dash/ (дата звернення 10.05.2025).

7. Super Meat Boy. *Steam*. URL: https://store.steampowered.com/app/40800/Super_Meat_Boy/ (дата звернення 10.05.2025).

8. Hollow Knight. *Steam*. URL: https://store.steampowered.com/app/367520/Hollow_Knight/ (дата звернення 10.05.2025).

9. Pygame. *Pygame*. URL: https://www.pygame.org/wiki/ (дата звернення 10.05.2025).

10. Python. *Python*. URL: https://www.python.org/about/ (дата звернення 10.05.2025).

11. Unity. *Unity*. URL: https://unity.com/grow (дата звернення 10.05.2025).

12. Unreal Engine. *Unreal Engine*. URL: https://www.unrealengine.com/en-US (дата звернення 10.05.2025).

13. PyCharm. *JetBrains*. URL: https://www.jetbrains.com/pycharm/ (дата звернення 10.05.2025).

14. Visual Studio Code. *Visual Studio Code*. URL: https://code.visualstudio.com/ (дата звернення 10.05.2025).

15. Code With Me. *JetBrains*. URL: https://www.jetbrains.com/code-with-me/ (дата звернення 10.05.2025).

16. Aseprite. *Steam*. URL: https://store.steampowered.com/app/431730/Aseprite/ (дата звернення 10.05.2025).

17. Tilesetter. *Steam*. URL: https://store.steampowered.com/app/1105890/Tilesetter/ (дата звернення 10.05.2025).

18. MVC: Model, View, Controller. *Codecademy*. URL: https://www.codecademy.com/article/mvc (дата звернення 10.05.2025).

19. Embold. *Embold.io*. URL: https://embold.io/company/ (дата звернення 10.05.2025).

20. Inno Setup. *Jrsoftware.org*. URL: https://jrsoftware.org/isinfo.php (дата звернення 10.05.2025).

ДОДАТОК А

StrikePlagiarism.com

Дата звіту 6/2/2025
Дата редагування 6/2/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-12_Логвиненко_ПЗ
Автор Науковий керівник / Експерт
ІП-12_ЛогвиненкоДифучин А.Ю.
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.

13.15%
13.15%
КП 1

10	12023	87937
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Рисунок А.1 – Звіт подібності

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

Класична гра у жанрі платформер, з розробкою редактора рівнів
(комплексна тема)
Текст програми
КПІ.ІП-1217.045480.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав ЛОГВИНЕНКО

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/ShomanDanyloIP-12/LevelKing

Файл suggest_menu.py
Реалізація функціональної задачі користувацького інтерфейсу

```
class Suggest_menu:
    def __init__(self, switch, suggest, update_level_builder_grid_str):
        self.display_surface = pygame.display.get_surface()
                        self.image_background = load(path.join(script_directory, 'graphics', 'menus', 'main_menu',
'background.png')).convert_alpha()
        window_size = self.display_surface.get_size()
        self.image_background = pygame.transform.scale(self.image_background, window_size)
        self.switch = switch
        self.update_level_builder_grid_str = update_level_builder_grid_str
        self.buttons = Buttons_sstm()
        self.suggest = suggest
        self.input_fields = InputField()
        self.switch_locker = True
        self.labels = Label('123')

    def event_loop(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            self.menu_click(event)
            self.input_fields.handle_event(event)

    def run(self, dt):
        self.event_loop()
        self.display_surface.blit(self.image_background, (0, 0))
        self.buttons.display()
        self.labels.draw()
        self.input_fields.draw()


    def menu_click(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN and self.buttons.b_rect.collidepoint(
                mouse_pos()) and self.switch_locker == True:
            self.switch_locker = False
            self.switch({'from': 'suggest_menu', 'to': 'stranger_level_menu'})
        if event.type == pygame.MOUSEBUTTONDOWN and self.buttons.suggest_button_rect.collidepoint(
                mouse_pos()) and self.switch_locker == True:
            self.switch_locker = False
            self.suggest()
            self.switch({'from': 'suggest_menu', 'to': 'public_levels_menu'})
        if event.type == pygame.MOUSEBUTTONDOWN and self.buttons.redact_button_rect.collidepoint(
                mouse_pos()) and self.switch_locker == True:
            self.switch_locker = False
            self.update_level_builder_grid_str()
            self.switch({'from': 'suggest_menu', 'to': 'level_builder'})


class Label:
    def __init__(self, level_name):
        self.display_surface = pygame.display.get_surface()
        self.window_size = self.display_surface.get_size()
        self.multiplayer_x = self.window_size[0] / 1280
```

```
        self.multiplayer_y = self.window_size[1] / 720

        self.level_name = level_name
        self.color_bg = pygame.Color('#c86259')
        self.color_border = pygame.Color('#33323d')
        self.font = pygame.font.Font(path.join(script_directory, 'graphics', 'ui', 'ARCADEPI.ttf'),
                                     int(18 * self.multiplayer_x))

        # wrapped lines
        def sx(value):
            return int(value * self.multiplayer_x)
        def sy(value):
            return int(value * self.multiplayer_y)

        b_size = sx(45)
        margin = sx(25)
        topleft = (margin, margin + b_size)
        self.level_rect = pygame.Rect(topleft[0] + sx(250), topleft[1] + sy(80), sx(6) + sx(360), sy(30))


    def draw(self):
        pygame.draw.rect(self.display_surface, self.color_bg, self.level_rect)
        pygame.draw.rect(self.display_surface, self.color_border, self.level_rect, 5)
            self.display_surface.blit(self.font.render(self.level_name, True, self.color_border), (self.level_rect.x + 5,
self.level_rect.y + 5))

    def update_info(self, level):
        self.level_name = level['title']
        self.display_surface.blit(self.font.render(self.level_name, True, self.color_border),
                                  (self.level_rect.x + 5, self.level_rect.y + 5))


class InputField:
    def __init__(self):
        max_length = 25
        self.display_surface = pygame.display.get_surface()
        self.window_size = self.display_surface.get_size()
        self.multiplayer_x = self.window_size[0] / 1280
        self.multiplayer_y = self.window_size[1] / 720

        self.color_inactive = pygame.Color('#c86259')
        self.color_active = pygame.Color('#de9970')
        self.color_comment = self.color_inactive
        self.comment_text = ''
        self.comment_wrapped_lines = []
                self.font = pygame.font.Font(path.join(script_directory, 'graphics', 'ui', 'ARCADEPI.ttf'), int(18 *
self.multiplayer_x))
        self.comment_txt_surface = self.font.render(self.comment_text, True, self.color_comment)
        self.comment_active = False
        self.max_length = max_length
        self.create_fields()

    def create_fields(self):
        def sx(value):
            return int(value * self.multiplayer_x)
        def sy(value):
            return int(value * self.multiplayer_y)

        b_size = sx(45)
        margin = sx(25)
        topleft = (margin, margin + b_size)

        self.comment_rect = pygame.Rect(topleft[0] + sx(250), topleft[1] + sy(145), sx(6) + sx(360), sy(210))
```

```
    def handle_event(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN:
            if self.comment_rect.collidepoint(event.pos):
                self.comment_active = not self.comment_active
            else:
                self.comment_active = False
            self.color_comment = self.color_active if self.comment_active else self.color_inactive

        elif event.type == pygame.KEYDOWN:
            if self.comment_active:
                if event.key == pygame.K_BACKSPACE:
                    self.comment_text = self.comment_text[:-1]
                elif len(self.comment_text) < self.max_length * 10:
                    self.comment_text += event.unicode
                self.comment_txt_surface = self.font.render(self.comment_text, True, '#33323d')
                self.comment_wrapped_lines = self.wrap_text(self.comment_text, self.comment_rect.width)

    def wrap_text(self, text, max_width):
        lines = []
        current_line = ''

        for char in text:
            test_line = current_line + char
            width, _ = self.font.size(test_line)
            if width <= max_width - 10:  # -10 — для внутрішнього відступу
                current_line = test_line
            else:
                lines.append(current_line)
                current_line = char
        if current_line:
            lines.append(current_line)
        return lines

    def draw(self):
        pygame.draw.rect(self.display_surface, self.color_comment, self.comment_rect)
        pygame.draw.rect(self.display_surface, '#33323d', self.comment_rect, 5)
        line_height = self.font.get_height()
        for i, line in enumerate(self.comment_wrapped_lines):
            line_surface = self.font.render(line, True, '#33323d')
            self.display_surface.blit(line_surface,
                                      (self.comment_rect.x + 5, self.comment_rect.y + 5 + i * line_height))

    def get_comment(self):
        return self.comment_text


class Buttons_sstm:
    def __init__(self):
        self.display_surface = pygame.display.get_surface()
        self.window_size = self.display_surface.get_size()
        self.multiplayer_x = self.window_size[0] / 1280
        self.multiplayer_y = self.window_size[1] / 720
        self.font = pygame.font.Font(path.join(script_directory, 'graphics', 'ui', 'ARCADEPI.ttf'),
                                     int(18 * self.multiplayer_x))
        self.create_buttons()

    def create_buttons(self):
        self.display_surface = pygame.display.get_surface()

        def sx(value):
            return int(value * self.multiplayer_x)
```

```
        def sy(value):
            return int(value * self.multiplayer_y)

        # back to main menu
        b_size = sx(45)
        b_margin = sx(6)
        b_topleft = (self.window_size[0] - b_size - b_margin, b_margin)
        self.b_rect = pygame.Rect(b_topleft, (b_size, b_size))
        self.image = load(path.join(script_directory, 'graphics', 'menus', 'back_button.png')).convert_alpha()
        self.image = pygame.transform.scale(self.image, (b_size, b_size))

        # menu area

        margin = sx(25)
        width = self.window_size[0] - (margin * 2)
        height = self.window_size[1] - (margin * 2) - b_size
        topleft = (margin, margin + b_size)
        self.local_level_image = load(
            path.join(script_directory, 'graphics', 'menus', 'main_menu', 'buttons_back.png')).convert_alpha()
        self.local_level_image = pygame.transform.scale(self.local_level_image, (width, height))
        self.local_level_rect = pygame.Rect(topleft, (width, height))

        # labels area

        self.level_label_offset = topleft[0] + sx(100), topleft[1] + sy(85), sx(6) + sx(360) / 2, sy(30)
        self.comment_label_offset = topleft[0] + sx(100), topleft[1] + sy(150), sx(6) + sx(360) / 2, sy(30)

        # button areas
        self.image_redact_button = load(
            path.join(script_directory, 'graphics', 'menus', 'level_buttons', 'redact_button.png')).convert_alpha()
        self.image_redact_button = pygame.transform.scale(self.image_redact_button, (sx(180), sy(70)))
        self.redact_button_rect = pygame.Rect((topleft[0] + sx(250), topleft[1] + sy(500)), (sx(180), sy(70)))
        self.image_suggest_button = load(
            path.join(script_directory, 'graphics', 'menus', 'level_buttons',
                      'suggest_button.png')).convert_alpha()
        self.image_suggest_button = pygame.transform.scale(self.image_suggest_button, (sx(180), sy(70)))
        self.suggest_button_rect = pygame.Rect(
            (topleft[0] + sx(436), topleft[1] + sy(500)),
            (sx(180), sy(70)))

    def display(self):
        self.display_surface.blit(self.image, self.b_rect.topleft)
        self.display_surface.blit(self.local_level_image, self.local_level_rect.topleft)
        self.display_surface.blit(self.font.render('Level name:', True, '#33323d'), self.level_label_offset)
        self.display_surface.blit(self.font.render('Comment:', True, '#33323d'), self.comment_label_offset)
        self.display_surface.blit(self.image_redact_button, self.redact_button_rect.topleft)
        self.display_surface.blit(self.image_suggest_button, self.suggest_button_rect.topleft)
```

Файл check_file.py

Реалізація функціональної задачі перевірки файлу перед завантаженням на сервер

```
# Допустимі шари
ALLOWED_LAYERS = {
    'bricks bg', 'bg palms', 'water', 'interract_info', 'finish_info',
    'sign bottom', 'sign top', 'sign left', 'sign right',
```

```
    'terrain', 'bricks fg', 'platforms', 'enemies',
    'coins', 'fg objects'
}

# Максимальні межі координат
X_LIMIT = 64000
Y_LIMIT = 32000

# Максимальна кількість об'єктів у шарі
MAX_OBJECTS_PER_LAYER = 1000
MAX_TOTAL_OBJECTS = 10000

# Допустимі типи значень для кожного шару
LAYER_VALUE_TYPES = {
    'terrain': str,
    'coins': int,
    'platforms': str,
    'enemies': int,
    'bricks fg': str,
    'bricks bg': str,
    'bg palms': int,
    'fg objects': int,
    'water': str,
    'interract_info': str,
    'finish_info': str,
    'sign bottom': str,
    'sign top': str,
    'sign left': str,
    'sign right': str
}

def validate_level_data(data: list) -> bool:
    try:
        # Перевірка основної структури
        if not isinstance(data, list) or len(data) != 2:
            return False
        layers, ser_offset = data

        if not isinstance(layers, dict) or not isinstance(ser_offset, list) or len(ser_offset) != 2:
            return False
        if not all(isinstance(n, int) for n in ser_offset):
            return False
```

```
total_objects = 0

for layer_name, layer_data in layers.items():
    # Назва шару — тільки з whitelist
    if layer_name not in ALLOWED_LAYERS:
        return False

    if not isinstance(layer_data, dict):
        return False

    if len(layer_data) > MAX_OBJECTS_PER_LAYER:
        return False

    expected_type = LAYER_VALUE_TYPES.get(layer_name)
    if expected_type is None:
        return False

    for coords, value in layer_data.items():
        # Ключ має бути кортежем з 2 цілих чисел
        if not (isinstance(coords, tuple) and len(coords) == 2 and all(isinstance(c, int) for c in coords)):
            return False
        x, y = coords
        if abs(x) > X_LIMIT or abs(y) > Y_LIMIT:
            return False

        # Значення має бути потрібного типу
        if not isinstance(value, expected_type):
            return False

        # Захист від вбудованих класів / функцій
        if isinstance(value, str):
            if any(bad in value.lower() for bad in ['<script>', '__class__', '__globals__', 'lambda']):
                return False
            if len(value) > 100:  # довгі рядки не допускаються
                return False
        if isinstance(value, int) and abs(value) > 1_000_000:
            return False

    total_objects += len(layer_data)

if total_objects > MAX_TOTAL_OBJECTS:
```

```
            return False

        return True

    except Exception:
        return False
```

Файл save_menu.py

Реалізація функціональної задачі локального зберігання рівнів

```
script_directory = path.dirname(path.realpath(__file__))
class Save_menu:
    def __init__(self, switch, save_file):
        self.display_surface = pygame.display.get_surface()
        self.image_background = load(path.join(script_directory, 'graphics', 'menus', 'save_menu',
'background.png')).convert_alpha()
        window_size = self.display_surface.get_size()
        self.image_background = pygame.transform.scale(self.image_background, window_size)
        self.switch = switch
        self.buttons = Buttons_sm()
        self.switch_locker = True
        self.from_where = ''
        self.save_file = save_file

        self.read_button_states()

    def event_loop(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            self.menu_click(event)

    def run(self, dt):
        self.event_loop()
        self.display_surface.blit(self.image_background, (0, 0))
        self.buttons.display()

    def read_level(self, name):
        file_path = path.join(script_directory, 'saved_levels', f'{name}.json')
        with open(file_path, 'rb') as file:
            data = pickle.load(file)
```

```
    return data[0]

def save_button_states(self):
    saved = {'state1': self.buttons.saved1,
             'state2': self.buttons.saved2,
             'state3': self.buttons.saved3,
             'state4': self.buttons.saved4,
             'state5': self.buttons.saved5,
             'state6': self.buttons.saved6,
             'name1': self.buttons.save1,
             'name2': self.buttons.save2,
             'name3': self.buttons.save3,
             'name4': self.buttons.save4,
             'name5': self.buttons.save5,
             'name6': self.buttons.save6}
    file_path = path.join(script_directory, 'saved_levels', f'states.json')
    with open(file_path, 'wb') as file:
        pickle.dump(saved, file)

def read_button_states(self):
    file_path = path.join(script_directory, 'saved_levels', f'states.json')
    with open(file_path, 'rb') as file:
        saves = pickle.load(file)
        self.buttons.saved1 = saves['state1']
        self.buttons.saved2 = saves['state2']
        self.buttons.saved3 = saves['state3']
        self.buttons.saved4 = saves['state4']
        self.buttons.saved5 = saves['state5']
        self.buttons.saved6 = saves['state6']
        self.buttons.save1 = saves['name1']
        self.buttons.save2 = saves['name2']
        self.buttons.save3 = saves['name3']
        self.buttons.save4 = saves['name4']
        self.buttons.save5 = saves['name5']
        self.buttons.save6 = saves['name6']

def menu_click(self, event):
    if event.type == pygame.MOUSEBUTTONDOWN and self.buttons.mm_rect.collidepoint(
            mouse_pos()) and self.switch_locker == True:
        self.switch_locker = False
        self.save_button_states()
        self.switch({'from': 'save_menu', 'to': 'main_menu'})
```

```
    if event.type == pygame.MOUSEBUTTONDOWN and (self.buttons.level1_save_button_rect.collidepoint(
            mouse_pos()) or self.buttons.level2_save_button_rect.collidepoint(
            mouse_pos()) or self.buttons.level3_save_button_rect.collidepoint(
            mouse_pos()) or self.buttons.level4_save_button_rect.collidepoint(
            mouse_pos()) or self.buttons.level5_save_button_rect.collidepoint(
            mouse_pos()) or self.buttons.level6_save_button_rect.collidepoint(
            mouse_pos())) and self.switch_locker == True:
        self.click(mouse_pos(), mouse_buttons())

def click(self, mouse_pos, mouse_button):
    if self.from_where == 'level_builder':
        if self.buttons.level1_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0]:  # left mouse on_click
                self.buttons.save1 = datetime.now().strftime("%y-%m-%d %H_%M_%S")
                self.buttons.saved1 = True
                self.save_file(self.buttons.save1)
            if mouse_button[2]:  # right on_click
                self.buttons.saved1 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save1 + '.json'))
        elif self.buttons.level2_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0]:  # left mouse on_click
                self.buttons.save2 = datetime.now().strftime("%y-%m-%d %H_%M_%S")
                self.buttons.saved2 = True
                self.save_file(self.buttons.save2)
            if mouse_button[2]:  # right on_click
                self.buttons.saved2 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save2 + '.json'))
        elif self.buttons.level3_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0]:  # left mouse on_click
                self.buttons.save3 = datetime.now().strftime("%y-%m-%d %H_%M_%S")
                self.buttons.saved3 = True
                self.save_file(self.buttons.save3)
            if mouse_button[2]:  # right on_click
                self.buttons.saved3 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save3 + '.json'))
        elif self.buttons.level4_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0]:  # left mouse on_click
                self.buttons.save4 = datetime.now().strftime("%y-%m-%d %H_%M_%S")
                self.buttons.saved4 = True
                self.save_file(self.buttons.save4)
            if mouse_button[2]:  # right on_click
                self.buttons.saved4 = False
```

```
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save4 + '.json'))
        elif self.buttons.level5_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0]:  # left mouse on_click
                self.buttons.save5 = datetime.now().strftime("%y-%m-%d %H_%M_%S")
                self.buttons.saved5 = True
                self.save_file(self.buttons.save5)
            if mouse_button[2]:  # right on_click
                self.buttons.saved5 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save5 + '.json'))
        elif self.buttons.level6_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0]:  # left mouse on_click
                self.buttons.save6 = datetime.now().strftime("%y-%m-%d %H_%M_%S")
                self.buttons.saved6 = True
                self.save_file(self.buttons.save6)
            if mouse_button[2]:  # right on_click
                self.buttons.saved6 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save6 + '.json'))
    if self.from_where == 'main_menu':
        if self.buttons.level1_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0] and self.buttons.saved1:  # left mouse on_click
                self.switch_locker = False
                self.switch({'from': 'save_menu', 'to': 'level'}, self.read_level(self.buttons.save1))
            if mouse_button[2]:  # right on_click
                self.buttons.saved1 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save1 + '.json'))
        elif self.buttons.level2_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0] and self.buttons.saved2:  # left mouse on_click
                self.switch_locker = False
                self.switch({'from': 'save_menu', 'to': 'level'}, self.read_level(self.buttons.save2))
            if mouse_button[2]:  # right on_click
                self.buttons.saved2 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save2 + '.json'))
        elif self.buttons.level3_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0] and self.buttons.saved3:  # left mouse on_click
                self.switch_locker = False
                self.switch({'from': 'save_menu', 'to': 'level'}, self.read_level(self.buttons.save3))
            if mouse_button[2]:  # right on_click
                self.buttons.saved3 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save3 + '.json'))
        elif self.buttons.level4_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0] and self.buttons.saved4:  # left mouse on_click
                self.switch_locker = False
```

```
                self.switch({'from': 'save_menu', 'to': 'level'}, self.read_level(self.buttons.save4))
            if mouse_button[2]:  # right on_click
                self.buttons.saved4 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save4 + '.json'))
        elif self.buttons.level5_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0] and self.buttons.saved5:  # left mouse on_click
                self.switch_locker = False
                self.switch({'from': 'save_menu', 'to': 'level'}, self.read_level(self.buttons.save5))
            if mouse_button[2]:  # right on_click
                self.buttons.saved5 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save5 + '.json'))
        elif self.buttons.level6_save_button_rect.collidepoint(mouse_pos):
            if mouse_button[0] and self.buttons.saved6:  # left mouse on_click
                self.switch_locker = False
                self.switch({'from': 'save_menu', 'to': 'level'}, self.read_level(self.buttons.save6))
            if mouse_button[2]:  # right on_click
                self.buttons.saved6 = False
                os.remove(path.join(script_directory, 'saved_levels', self.buttons.save6 + '.json'))
        self.save_button_states()

class Buttons_sm:
    def __init__(self):
        self.display_surface = pygame.display.get_surface()
        self.window_size = self.display_surface.get_size()
        self.multiplayer_x = self.window_size[0] / 1280
        self.multiplayer_y = self.window_size[1] / 720
        self.create_buttons()

    def create_buttons(self):
        self.display_surface = pygame.display.get_surface()

        def sx(value):
            return int(value * self.multiplayer_x)
        def sy(value):
            return int(value * self.multiplayer_y)
        # back to main menu
        mm_size = sx(45)
        mm_margin = sx(6)
        mm_topleft = (self.window_size[0] - mm_size - mm_margin, mm_margin)
        self.mm_rect = pygame.Rect(mm_topleft, (mm_size, mm_size))
        self.image = load(path.join(script_directory, 'graphics', 'menus', 'main_menu_button.png')).convert_alpha()
        self.image = pygame.transform.scale(self.image, (mm_size, mm_size))
```

```
        # button areas
        width = sx(300)
        height = sy(200)
        self.level1_save_button_rect = pygame.Rect((sx(150), sy(135)), (width, height))
        self.level2_save_button_rect = pygame.Rect((width + sx(190), sy(135)), (width, height))
        self.level3_save_button_rect = pygame.Rect((width * 2 + sx(230), sy(135)), (width, height))

        self.level4_save_button_rect = pygame.Rect((sx(150), height + sy(175)), (width, height))
        self.level5_save_button_rect = pygame.Rect((width + sx(190), height + sy(175)), (width, height))
        self.level6_save_button_rect = pygame.Rect((width * 2 + sx(230), height + sy(175)), (width, height))

        self.image_save_button = load(path.join(script_directory, 'graphics', 'menus', 'save_menu',
'save_button.png')).convert_alpha()
        self.image_save_button = pygame.transform.scale(self.image_save_button, (sx(300), sy(200)))

        # saves
        self.font = pygame.font.Font(path.join(script_directory, 'graphics', 'ui', 'ARCADEPI.ttf'), sx(18))
        self.save1 = None
        self.save2 = None
        self.save3 = None
        self.save4 = None
        self.save5 = None
        self.save6 = None

        self.saved1 = False
        self.saved2 = False
        self.saved3 = False
        self.saved4 = False
        self.saved5 = False
        self.saved6 = False

    def display(self):
        self.display_surface.blit(self.image, self.mm_rect.topleft)
        self.display_surface.blit(self.image_save_button, self.level1_save_button_rect.topleft)
        self.display_surface.blit(self.image_save_button, self.level2_save_button_rect.topleft)
        self.display_surface.blit(self.image_save_button, self.level3_save_button_rect.topleft)
        self.display_surface.blit(self.image_save_button, self.level4_save_button_rect.topleft)
        self.display_surface.blit(self.image_save_button, self.level5_save_button_rect.topleft)
        self.display_surface.blit(self.image_save_button, self.level6_save_button_rect.topleft)

        if self.saved1:
```

```
            save1_text_surf = self.font.render(str(self.save1), False, '#33323d')
            save1_text_rect = save1_text_surf.get_rect(center=(self.level1_save_button_rect.centerx,
self.level1_save_button_rect.centery))
            self.display_surface.blit(save1_text_surf, save1_text_rect)
        if self.saved2:
            save2_text_surf = self.font.render(str(self.save2), False, '#33323d')
            save2_text_rect = save2_text_surf.get_rect(center=(self.level2_save_button_rect.centerx,
self.level2_save_button_rect.centery))
            self.display_surface.blit(save2_text_surf, save2_text_rect)
        if self.saved3:
            save3_text_surf = self.font.render(str(self.save3), False, '#33323d')
            save3_text_rect = save3_text_surf.get_rect(center=(self.level3_save_button_rect.centerx,
self.level3_save_button_rect.centery))
            self.display_surface.blit(save3_text_surf, save3_text_rect)
        if self.saved4:
            save4_text_surf = self.font.render(str(self.save4), False, '#33323d')
            save4_text_rect = save4_text_surf.get_rect(center=(self.level4_save_button_rect.centerx,
self.level4_save_button_rect.centery))
            self.display_surface.blit(save4_text_surf, save4_text_rect)
        if self.saved5:
            save5_text_surf = self.font.render(str(self.save5), False, '#33323d')
            save5_text_rect = save5_text_surf.get_rect(center=(self.level5_save_button_rect.centerx,
self.level5_save_button_rect.centery))
            self.display_surface.blit(save5_text_surf, save5_text_rect)
        if self.saved6:
            save6_text_surf = self.font.render(str(self.save6), False, '#33323d')
            save6_text_rect = save6_text_surf.get_rect(center=(self.level6_save_button_rect.centerx,
self.level6_save_button_rect.centery))
            self.display_surface.blit(save6_text_surf, save6_text_rect)



        # pygame.draw.rect(self.display_surface, 'red', self.level1_save_button_rect)
        # pygame.draw.rect(self.display_surface, 'red', self.level2_save_button_rect)
        # pygame.draw.rect(self.display_surface, 'red', self.level3_save_button_rect)
        # pygame.draw.rect(self.display_surface, 'red', self.level4_save_button_rect)
        # pygame.draw.rect(self.display_surface, 'red', self.level5_save_button_rect)
        # pygame.draw.rect(self.display_surface, 'red', self.level6_save_button_rect)
```

Файл entities.py

Реалізація функціональної задачі створення світу і різноманітних ворогів

```
class Treasure(AnimatedEntity):
            def __init__(self, treasure_type, assets, pos, group):
```

```
                super().__init__(assets, pos, group)
                self.rect = self.image.get_rect(center = pos)
                self.treasure_type = treasure_type


class FXParticle(AnimatedEntity):
        def __init__(self, assets, pos, group):
                super().__init__(assets, pos, group)
                self.rect = self.image.get_rect(center=pos)

        def animate(self, dt):
                self._advance_frame(dt)

        def _advance_frame(self, dt):
                self.current_frame += 8 * dt
                index = int(self.current_frame)

                if index < len(self.frame_assets):
                        self.image = self.frame_assets[index]
                else:
                        self.kill()


class SolidBlock(EntityBase):
        def __init__(self, pos, size, group):
                surf = pygame.Surface(size)
                super().__init__(pos, surf, group)


class SkyCloud(EntityBase):
        def __init__(self, pos, surf, group, deletion_limit):
                super().__init__(pos, surf, group, 1)
                self.deletion_limit = deletion_limit
                self._init_position()
                self._init_speed()

        def _init_position(self):
                self.pos = vector(self.rect.topleft)

        def _init_speed(self):
                self.speed = randint(20, 30)

        def update(self, dt):
                self._move_left(dt)
                self._destroy_if_out_of_bounds()

        def _move_left(self, dt):
                self.pos.x -= self.speed * dt
                self.rect.x = round(self.pos.x)

        def _destroy_if_out_of_bounds(self):
                if self.rect.right < self.deletion_limit:
                        self.kill()


class SpikeTrap(EntityBase):
        def __init__(self, surf, pos, group):
                super().__init__(pos, surf, group)
                self.mask = pygame.mask.from_surface(self.image)


class Mace(EntityBase):
        def __init__(self, assets, pos, group, radius):
```

```
                self.center = (pos[0] + 64 / 2,pos[1] + 64 / 2)
                self.radius = radius
                self.speed = 120
                self.angle = 0

                y = self.center[1] + sin(radians(self.angle)) * self.radius
                x = self.center[0] + cos(radians(self.angle)) * self.radius

                super().__init__((x, y), assets, group)
                self.mask = pygame.mask.from_surface(self.image)

        def update(self, dt):
                self.angle += self.speed * dt
                y = self.center[1] + sin(radians(self.angle)) * self.radius
                x = self.center[0] + cos(radians(self.angle)) * self.radius
                self.rect.center = (x, y)


class Pig(EntityBase):
        def __init__(self, assets, pos, group, collision_entities):
                self.current_frame = 0
                self.frame_assets = assets
                self.orientation = 'right'
                self.direction = vector(choice((1, -1)), 0)
                self.orientation = 'left' if self.direction.x < 0 else 'right'

                surf = self.frame_assets[f'run_{self.orientation}'][self.current_frame]
                super().__init__(pos, surf, group)

                self.rect.bottom = self.rect.top + 64
                self.mask = pygame.mask.from_surface(self.image)

                self.speed = 150
                self.collision_entities = collision_entities

                self.pos = vector(self.rect.topleft)

                self.dead = False
                self.death_timer = Timer(100)

                if not self._is_on_ground():
                        self.kill()

        def _is_on_ground(self):
                check_pos = self.rect.midbottom + vector(0, 10)
                return any(entity.rect.collidepoint(check_pos) for entity in self.collision_entities)

        def update(self, dt):
                self.animate(dt)
                self.move(dt)
                self.death_timer.update()
                self._handle_death()

        def animate(self, dt):
                frames = self.frame_assets[f'run_{self.orientation}']
                self.current_frame = (self.current_frame + 8 * dt) % len(frames)
                self.image = frames[int(self.current_frame)]
                self.mask = pygame.mask.from_surface(self.image)

                if self.death_timer.is_active():
                        temp = self.mask.to_surface()
                        temp.set_colorkey('black')
                        self.image = temp
```

```
        def _handle_death(self):
                if self.dead:
                        self.speed = 0
                        if not self.death_timer.is_active():
                                self.kill()

        def move(self, dt):
                if self.direction.x > 0:
                        if self._wall_hit("right") or not self._has_floor("right"):
                                self._turn_around("left")

                if self.direction.x < 0:
                        if self._wall_hit("left") or not self._has_floor("left"):
                                self._turn_around("right")

                self.pos.x += self.direction.x * self.speed * dt
                self.rect.x = round(self.pos.x)

        def _wall_hit(self, side):
                offset = vector(1, 0) if side == "right" else vector(-1, 0)
                check = self.rect.midright if side == "right" else self.rect.midleft
                return any(entity.rect.collidepoint(check + offset) for entity in self.collision_entities)

        def _has_floor(self, side):
                offset = vector(1, 1) if side == "right" else vector(-1, 1)
                check = self.rect.bottomright if side == "right" else self.rect.bottomleft
                return any(entity.rect.collidepoint(check + offset) for entity in self.collision_entities)

        def _turn_around(self, new_orientation):
                self.direction.x *= -1
                self.orientation = new_orientation

class Platform(EntityBase):
        def __init__(self, pos, surf, group):
                super().__init__(pos, surf, group)
                self.mask = pygame.mask.from_surface(self.image)

                mask_rects = self.mask.get_bounding_rects()

                if mask_rects:
                        mask_rect = mask_rects[0].copy()
                        for r in mask_rects[1:]:
                                mask_rect.union_ip(r)
                else:
                        mask_rect = self.image.get_rect()  # fallback

                self.rect = mask_rect.move(self.rect.topleft)

class Cannon(EntityBase):
        def __init__(self, orientation, assets, pos, group, cannonball_surf, damage_entities):
                self.frame_index = 0
                self.orientation = orientation
                self.frame_assets = self._prepare_assets(assets)

                self.animation_name = 'idle'

                initial_frame = self.frame_assets[self.animation_name][self.frame_index]
                super().__init__(pos, initial_frame, group)

                self.rect.bottom = self.rect.top + 64

                self.made_shot = False
```

```
                self.cannonball_surf = cannonball_surf

                self.attack_cooldown = Timer(3500)

                self.all_entities = group[0]
                self.damage_entities = damage_entities

        def _prepare_assets(self, assets):
                if self.orientation == 'right':
                        return {
                                key: [pygame.transform.flip(surf, True, False) for surf in frames]
                                for key, frames in assets.items()
                        }
                return assets.copy()

        def update(self, dt):
                self._update_status()
                self._animate(dt)
                self.attack_cooldown.update()

        def _update_status(self):
                distance = vector(self.player.rect.center).distance_to(vector(self.rect.center))
                if distance < 700 and not self.attack_cooldown.is_active():
                        self.animation_name = 'attack'
                else:
                        self.animation_name = 'idle'

        def _animate(self, dt):
                frames = self.frame_assets[self.animation_name]
                self.frame_index += 8 * dt

                if self.frame_index >= len(frames):
                        self.frame_index = 0
                        if self.made_shot:
                                self.attack_cooldown.activate()
                                self.made_shot = False

                if int(self.frame_index) == 2 and self.animation_name == 'attack' and not self.made_shot:
                        self._shoot()

                self.image = frames[int(self.frame_index)]

        def _shoot(self):
                direction = vector(1, 0) if self.orientation == 'right' else vector(-1, 0)
                offset = direction * (20 if self.orientation == 'right' else 50) + vector(0, -10)
                position = self.rect.center + offset

                Cannonball(position, direction, self.cannonball_surf, [self.all_entities, self.damage_entities])
                self.made_shot = True

class Saw(EntityBase):
        def __init__(self, orientation, assets, pos, group):
                self.orientation = orientation
                self.frame_assets = assets.copy()
                if orientation == 'top':
                        flipped_frames = []
                        for surf in self.frame_assets:
                                flipped = pygame.transform.flip(surf, True, True)
                                flipped_frames.append(flipped)
                        self.frame_assets = flipped_frames

                if orientation == 'left':
                        flipped_frames = []
```

```
                        for surf in self.frame_assets:
                                flipped = pygame.transform.rotate(surf, -90)
                                flipped_frames.append(flipped)
                        self.frame_assets = flipped_frames

                if orientation == 'right':
                        flipped_frames = []
                        for surf in self.frame_assets:
                                flipped = pygame.transform.rotate(surf, 90)
                                flipped_frames.append(flipped)
                        self.frame_assets = flipped_frames

                self.frame_index = 0
                super().__init__(pos, self.frame_assets[self.frame_index], group)
                self.rect.bottom = self.rect.top + 64

        def animate(self, dt):
                current_animation = self.frame_assets
                self.frame_index += 8 * dt
                if self.frame_index >= len(current_animation):
                        self.frame_index = 0
                self.image = current_animation[int(self.frame_index)]

        def update(self, dt):
                self.animate(dt)


class Cannonball(EntityBase):
        def __init__(self, pos, direction, surf, group):
                super().__init__(pos, surf, group)
                self.direction = direction.normalize()
                self.disappear_timer = Timer(3500)
                self.speed = 300
                self.disappear_timer.activate()

                self.mask = pygame.mask.from_surface(self.image)
                self._set_initial_position()

        def _set_initial_position(self):
                self.pos = vector(self.rect.topleft)

        def _move(self, dt):
                self.pos += self.direction * self.speed * dt
                self.rect.topleft = self.pos

        def _check_lifetime(self):
                self.disappear_timer.update()
                if not self.disappear_timer.is_active():
                        self.kill()

        def update(self, dt):
                self._move(dt)
                self._check_lifetime()


class Player(EntityBase):
        def __init__(self, pos, assets, group, collision_entities, sound, player_dead, semi_colidable_entities,
knockback_entities):

                # animation
                self.frames = assets
                self.frame_index = 0
                self.attack_frame_index = 0
```

```
                self.semi_colidable_entities = semi_colidable_entities
                self.knockback_entities = knockback_entities
                self.animation_name = 'idle'
                self.orientation = 'right'
                self.player_dead = player_dead
                self.player_attack = False
                surf = self.frames[f'{self.animation_name}_{self.orientation}'][self.frame_index]
                super().__init__(pos, surf, group)
                self.mask = pygame.mask.from_surface(self.image)
                self.speed = 400
                self.direction = vector()
                self.pos = vector(self.rect.center)

                self.gravity = 4
                self.wall_jump_x_impulse = 0
                self.wall_jump_decay = 5
                self.on_right_wall = False
                self.on_left_wall = False
                self.on_floor = False

                self.collision_entities = collision_entities
                self.hitbox = self.rect.inflate(-50, 0)
                self.is_drowning = False

                self.immortality_timer = Timer(500)
                self.platform_skip_timer = Timer(150)
                self.attack_cooldown_timer = Timer(1000)

                self.jmp_sound = sound
                self.jmp_sound.set_volume(0.1)

        def drown(self, flipper):
                if flipper == True:
                        self.speed = 150
                        self.gravity = 2
                elif flipper == False:
                        self.speed = 300
                        self.gravity = 4

        def set_drowning(self, flipper):
                if flipper == True:
                        self.is_drowning = True
                else:
                        self.is_drowning = False

        def update_animation(self):
                if self.player_dead():
                        self.animation_name = 'dead'
                        return

                if self.player_attack:
                        self.animation_name = 'attack'
                        return

                if self.direction.y < 0:
                        self.animation_name = 'jump'
                        return

                if not self.on_floor and self.direction.y > 0:
                        if (self.on_left_wall and self.direction.x < 0) or \
                                                (self.on_right_wall and self.direction.x > 0):
                                self.animation_name = 'wall_slide'
                        else:
```

```
                                        self.animation_name = 'fall'
                                return

                        self.animation_name = 'run' if self.direction.x != 0 else 'idle'

        def get_hit(self):
                        if not self.immortality_timer.is_active():
                                self.immortality_timer.activate()
                                self.direction.y -= 1.5

        def animate(self, dt):
                        if self.animation_name == 'dead':
                                current_animation = self.frames[f'{self.animation_name}_{self.orientation}']
                                self.speed = 0
                                self.frame_index += 8 * dt
                                self.frame_index = 3 if self.frame_index >= len(current_animation) else
self.frame_index
                                self.image = current_animation[int(self.frame_index)]
                                self.mask = pygame.mask.from_surface(self.image)
                        elif self.animation_name == 'attack':
                                current_animation = self.frames[f'{self.animation_name}_{self.orientation}']
                                self.attack_frame_index += 8 * dt
                                if self.attack_frame_index >= len(current_animation):
                                        self.player_attack = False
                                        self.attack_frame_index = 0
                                else:
                                        self.image = current_animation[int(self.attack_frame_index)]

                        else:
                                current_animation = self.frames[f'{self.animation_name}_{self.orientation}']
                                self.frame_index += 8 * dt
                                self.frame_index = 0 if self.frame_index >= len(current_animation) else
self.frame_index
                                self.image = current_animation[int(self.frame_index)]
                                self.mask = pygame.mask.from_surface(self.image)

                        if self.immortality_timer.is_active():
                                surf = self.mask.to_surface()
                                surf.set_colorkey('black')
                                self.image = surf
                        if self.is_drowning == True and self.animation_name != 'dead':
                                self.drown(True)
                        if self.is_drowning == False and self.animation_name != 'dead':
                                self.drown(False)

        def check_input(self):
                        keys = pygame.key.get_pressed()

                        if keys[pygame.K_RIGHT]:
                                self.direction.x = 1
                                self.orientation = 'right'
                        elif keys[pygame.K_LEFT]:
                                self.direction.x = -1
                                self.orientation = 'left'
                        else:
                                self.direction.x = 0

                        if keys[pygame.K_a] and not self.attack_cooldown_timer.is_active():
                                self.player_attack = True
                                self.attack_cooldown_timer.activate()

                        if keys[pygame.K_SPACE]:
                                if self.on_floor:
```

```
                                        self.direction.y = -2
                                        self.jmp_sound.play()
                                elif self.animation_name == 'wall_slide':
                                        self.direction.y = -2
                                        self.wall_jump_x_impulse = 2 if self.on_left_wall else -2
                                        self.jmp_sound.play()

                        if keys[pygame.K_DOWN]:
                                self.platform_skip_timer.activate()

        def decay_wall_jump(self, dt):
                if self.wall_jump_x_impulse != 0:
                        decay = self.wall_jump_decay * dt
                        if abs(self.wall_jump_x_impulse) <= decay:
                                self.wall_jump_x_impulse = 0
                        else:
                                self.wall_jump_x_impulse -= decay if self.wall_jump_x_impulse > 0
else -decay

        def move(self, dt):

                # horizontal movement
                total_x = self.direction.x + self.wall_jump_x_impulse
                self.pos.x += total_x * self.speed * dt
                self.hitbox.centerx = round(self.pos.x)
                self.rect.centerx = self.hitbox.centerx
                self.collide('horizontal')

                self.pos.y += self.direction.y * self.speed * dt
                self.hitbox.centery = round(self.pos.y)
                self.rect.centery = self.hitbox.centery
                self.collide('vertical')
                self.semi_collide()

        def add_gravity_y(self, dt):
                self.direction.y += self.gravity * dt
                if self.animation_name == 'wall_slide':
                        self.direction.y = min(self.direction.y, 0.5)
                self.rect.y += self.direction.y

        def if_collides_floor(self):
                floor_rect = pygame.Rect(self.hitbox.bottomleft, (self.hitbox.width, 2))
                floor_entities = []
                for entity in self.collision_entities:
                        if entity.rect.colliderect(floor_rect):
                                floor_entities = entity
                for entity in self.semi_colidable_entities:
                        if entity.rect.colliderect(floor_rect):
                                floor_entities = entity
                self.on_floor = True if floor_entities else False

        def check_knockback(self):
                left_edge = pygame.Rect(self.hitbox.left - 4, self.hitbox.top, 4, self.hitbox.height)
                right_edge = pygame.Rect(self.hitbox.right, self.hitbox.top, 4, self.hitbox.height)
                for entity in self.knockback_entities:
                        if entity.rect.colliderect(left_edge) and self.orientation == 'left' and
self.animation_name == 'attack':
                                entity.direction.x = - 1
                                entity.orientation = 'left'
                        elif entity.rect.colliderect(right_edge) and self.orientation == 'right' and
self.animation_name == 'attack':
                                entity.direction.x = 1
                                entity.orientation = 'right'
```

```
        def if_collides_wall(self):
                left_rect = pygame.Rect(self.hitbox.topleft + vector(-2, self.hitbox.height / 4), (2, self.hitbox.height / 2))
                right_rect = pygame.Rect(self.hitbox.topright + vector(0, self.hitbox.height / 4), (2, self.hitbox.height / 2))
                right_wall_entities = [entity for entity in self.collision_entities if entity.rect.colliderect(right_rect)]
                left_wall_entities = [entity for entity in self.collision_entities if entity.rect.colliderect(left_rect)]
                self.on_right_wall = True if right_wall_entities else False
                self.on_left_wall = True if left_wall_entities else False

        def collide(self, direction):
                for entity in self.collision_entities:
                        if entity.rect.colliderect(self.hitbox):
                                if direction == 'horizontal':
                                        if self.direction.x > 0:
                                                self.hitbox.right = entity.rect.left
                                        elif self.direction.x < 0:
                                                self.hitbox.left = entity.rect.right
                                        self.rect.centerx = self.hitbox.centerx
                                        self.pos.x = self.hitbox.centerx
                                else:
                                        if self.direction.y < 0:
                                                self.hitbox.top = entity.rect.bottom
                                        elif self.direction.y > 0:
                                                self.hitbox.bottom = entity.rect.top
                                        self.rect.centery = self.hitbox.centery
                                        self.pos.y = self.hitbox.centery
                                        self.direction.y = 0

        def semi_collide(self):
                if not self.platform_skip_timer.is_active():
                        for entity in self.semi_colidable_entities:
                                bottom_edge = pygame.Rect(self.hitbox.left, self.hitbox.bottom - 1, self.hitbox.width, 2)
                                if entity.rect.colliderect(bottom_edge):
                                        if self.direction.y > 0:
                                                self.hitbox.bottom = entity.rect.top
                                                self.rect.centery, self.pos.y = self.hitbox.centery, self.hitbox.centery
                                                self.direction.y = 0


        def update(self, dt):
                self.check_input()
                self.add_gravity_y(dt)
                self.move(dt)
                self.if_collides_floor()
                self.if_collides_wall()
                self.immortality_timer.update()
                self.platform_skip_timer.update()
                self.attack_cooldown_timer.update()
                self.decay_wall_jump(dt)
                self.check_knockback()

                self.update_animation()
                self.animate(dt)
```

Файл ui.py
Реалізація функціональної задачі користувацького інтерфейсу

```
class UI:
    def __init__(self, surface):
        script_dir = path.dirname(path.realpath(__file__))
        self.display_surface = surface

        self._base_width, self._base_height = 1280, 720
        self._update_scale()

        self.health_bar = pygame.image.load(path.join(script_dir, 'graphics', 'ui', 'health_bar.png')).convert_alpha()
        self.health_bar = pygame.transform.scale(self.health_bar, self._scale(192, 64))
        self.health_bar_pos = self._scale_pos(54, 39)
        self.bar_max_width = self._scale_value(152, axis='x')
        self.bar_height = self._scale_value(4, axis='y')

        self.coin = pygame.image.load(path.join(script_dir, 'graphics', 'ui', 'coin.png')).convert_alpha()
        self.coin = pygame.transform.scale(self.coin, self._scale(32, 32))
        self.coin_rect = self.coin.get_rect(topleft=self._scale_pos(50, 61))

        self.font = pygame.font.Font(path.join(script_dir, 'graphics', 'ui', 'ARCADEPI.ttf'), self._scale_value(30,
axis='x'))

    def _update_scale(self):
        width, height = self.display_surface.get_size()
        self._mx = width / self._base_width
        self._my = height / self._base_height

    def _scale_value(self, val, axis='x'):
        return int(val * (self._mx if axis == 'x' else self._my))

    def _scale(self, w, h):
        return (self._scale_value(w, 'x'), self._scale_value(h, 'y'))

    def _scale_pos(self, x, y):
        return self._scale_value(x, 'x'), self._scale_value(y, 'y')

    def show_health(self, current, full):
        self._update_scale()
        self.display_surface.blit(self.health_bar, self._scale_pos(20, 10))
        ratio = current / full
        current_width = self.bar_max_width * ratio
```

```
    bar_rect = pygame.Rect(self.health_bar_pos, (current_width, self.bar_height))
    pygame.draw.rect(self.display_surface, '#dc4949', bar_rect)

def show_coins(self, amount):
    self.display_surface.blit(self.coin, self.coin_rect)
    text_surf = self.font.render(str(amount), False, '#33323d')
    text_rect = text_surf.get_rect(midleft=(self.coin_rect.right + 4, self.coin_rect.centery))
    self.display_surface.blit(text_surf, text_rect)
```

Файл main.py

Реалізація функціональної задачі оркестра гри

```
def change_coins(self, amount, nulify=None):
    self.coins += amount
    if nulify:
        self.coins = 0

def change_health(self, amount, nulify=None):
    if amount < 0 and self.cur_health - amount > self.max_health:
        self.cur_health = self.max_health
    else:
        self.cur_health -= amount

    if nulify:
        self.cur_health = 100

def change_diamonds(self, amount, nulify=None):
    self.diamonds += amount
    if nulify:
        self.diamonds = 0

def level_complete(self):
    if self.diamonds == 3:
        self.level.complete = True

def get_score(self):
    return self.coins

def get_diamonds(self):
    return self.diamonds

def death(self):
```

```
    if self.cur_health <= 0:
        self.player_dead = True

def player_dead_get(self):
    return self.player_dead

def change_player_dead(self):
    self.player_dead = False

def save_level(self, name):
    file_path = path.join(script_directory, 'saved_levels', f'{name}.json')
    with open(file_path, 'wb') as file:
        pickle.dump(self.level_builder.generate_grid(), file)

def change_routes(self, routes):
    self.replacement = routes

def manage_routes(self):
    if self.replacement['from'] == 'level_builder':
        self.level_builder_active = False
    elif self.replacement['from'] == 'level':
        self.coins = 0
        self.cur_health = 100
        self.diamonds = 0
        self.player_dead = False
        self.level.background_music.stop()
        self.music.play(loops=-1)
        self.level_active = False
    elif self.replacement['from'] == 'main_menu':
        self.main_menu_active = False
    elif self.replacement['from'] == 'level_menu':
        self.level_menu_active = False
    elif self.replacement['from'] == 'save_menu':
        self.save_menu_active = False
    elif self.replacement['from'] == 'authorisation_menu':
        self.authorisation_menu_active = False
    elif self.replacement['from'] == 'authorisation_s_menu':
        self.authorisation_s_menu_active = False
    elif self.replacement['from'] == 'server_menu':
        self.server_menu_active = False
    elif self.replacement['from'] == 'delete_user_menu':
        self.delete_user_menu_active = False
```

```
elif self.replacement['from'] == 'public_level_menu':
    self.public_level_menu_active = False
elif self.replacement['from'] == 'local_level_menu':
    self.local_level_menu_active = False
elif self.replacement['from'] == 'suggestions_menu':
    self.suggestions_menu_active = False
elif self.replacement['from'] == 'public_levels_menu':
    self.public_levels_menu_active = False
elif self.replacement['from'] == 'suggestion_menu':
    self.suggestion_menu_active = False
elif self.replacement['from'] == 'stranger_level_menu':
    self.stranger_level_menu_active = False
elif self.replacement['from'] == 'suggest_menu':
    self.suggest_menu_active = False

if self.replacement['to'] == 'level_builder':
    if self.replacement['from'] != 'level':
        self.level_builder.from_where = self.replacement['from']
    self.level_builder_active = True
elif self.replacement['to'] == 'level':
    if self.replacement['from'] != 'level':
        self.set_level_from_where(self.replacement['from'])
    self.music.stop()
    self.level.background_music.play(loops=-1)
    self.level_active = True
elif self.replacement['to'] == 'main_menu':
    self.main_menu_active = True
elif self.replacement['to'] == 'level_menu':
    self.level_menu_active = True
elif self.replacement['to'] == 'save_menu':
    self.save_menu.from_where = self.replacement['from']
    self.save_menu_active = True
elif self.replacement['to'] == 'authorisation_menu':
    self.authorisation_menu_active = True
elif self.replacement['to'] == 'authorisation_s_menu':
    self.authorisation_s_menu_active = True
elif self.replacement['to'] == 'server_menu':
    self.server_menu_active = True
elif self.replacement['to'] == 'delete_user_menu':
    self.delete_user_menu_active = True
elif self.replacement['to'] == 'public_level_menu':
    self.public_level_menu_active = True
```

```
elif self.replacement['to'] == 'local_level_menu':
    self.local_level_menu_active = True
elif self.replacement['to'] == 'suggestions_menu':
    self.suggestions_menu_active = True
elif self.replacement['to'] == 'public_levels_menu':
    self.public_levels_menu_active = True
elif self.replacement['to'] == 'suggestion_menu':
    self.suggestion_menu_active = True
elif self.replacement['to'] == 'stranger_level_menu':
    self.stranger_level_menu_active = True
elif self.replacement['to'] == 'suggest_menu':
    self.suggest_menu_active = True

if self.level_builder_active:
    self.level_builder.switch_locker = True
if self.level_active:
    self.level.switch_locker = True
if self.main_menu_active:
    self.main_menu.switch_locker = True
if self.level_menu_active:
    self.level_menu.switch_locker = True
if self.save_menu_active:
    self.save_menu.switch_locker = True
if self.authorisation_menu_active:
    self.authorisation_menu.switch_locker = True
if self.authorisation_s_menu_active:
    self.authorisation_s_menu.switch_locker = True
if self.server_menu_active:
    self.server_menu.switch_locker = True
if self.delete_user_menu_active:
    self.delete_user_menu.switch_locker = True
if self.public_level_menu_active:
    self.public_level_menu.switch_locker = True
if self.local_level_menu_active:
    self.local_level_menu.switch_locker = True
if self.suggestions_menu_active:
    self.suggestions_menu.switch_locker = True
if self.public_levels_menu_active:
    self.public_levels_menu.switch_locker = True
if self.suggestion_menu_active:
    self.suggestion_menu.switch_locker = True
if self.stranger_level_menu_active:
```

```
            self.stranger_level_menu.switch_locker = True
        if self.suggest_menu_active:
            self.suggest_menu.switch_locker = True

        if not self.level_active:
            self.coins = 0
            self.cur_health = 100
            self.diamonds = 0
            self.player_dead = False

    def switch(self, routes, grid=None):
        self.change_routes(routes)
        self.passage.active = True
        if grid:
            self.passage.set_grid(grid)

    def run(self):
        while True:
            dt = self.time.tick() / 1000

            if self.level_builder_active:
                self.level_builder.run(dt)
            elif self.level_active:
                self.level.run(dt)
                self.ui.show_health(self.cur_health, self.max_health)
                self.ui.show_coins(self.coins)
                self.level_complete()
                self.death()
            elif self.main_menu_active:
                self.main_menu.run(dt)
            elif self.level_menu_active:
                self.level_menu.run(dt)
            elif self.save_menu_active:
                self.save_menu.run(dt)
            elif self.authorisation_menu_active:
                self.authorisation_menu.run(dt)
            elif self.authorisation_s_menu_active:
                self.authorisation_s_menu.run(dt)
            elif self.server_menu_active:
                self.server_menu.run(dt)
            elif self.delete_user_menu_active:
                self.delete_user_menu.run(dt)
```

```
            elif self.public_level_menu_active:
                self.public_level_menu.run(dt)
            elif self.local_level_menu_active:
                self.local_level_menu.run(dt)
            elif self.suggestions_menu_active:
                self.suggestions_menu.run(dt)
            elif self.public_levels_menu_active:
                self.public_levels_menu.run(dt)
            elif self.suggestion_menu_active:
                self.suggestion_menu.run(dt)
            elif self.stranger_level_menu_active:
                self.stranger_level_menu.run(dt)
            elif self.suggest_menu_active:
                self.suggest_menu.run(dt)
            self.passage.display(dt)
            pygame.display.update()


class Passage:
    def __init__(self, manage_routes, create_level):
        self.display_surface = pygame.display.get_surface()
        self.window_size = self.display_surface.get_size()
        self.manage_routes = manage_routes
        self.active = False
        self.grid = None
        self.create_level = create_level

        self.diaphragm = 0
        self.dir = 1
        self.center = (self.window_size[0] / 2, self.window_size[1] / 2)
        self.circle = vector(self.center).magnitude()
        self.limit = self.circle + 150

    def set_grid(self, grid):
        self.grid = grid

    def display(self, dt):
        if self.active:
            self.diaphragm += 2000 * dt * self.dir
            if self.diaphragm >= self.limit:
                self.dir = -1
                self.diaphragm += -10
```

```
            if self.grid:
                self.create_level(self.grid)
            self.manage_routes()

        if self.diaphragm < 0:
            self.active = False
            self.diaphragm = 0
            self.dir = 1
        pygame.draw.circle(self.display_surface, 'black', self.center, self.circle, int(self.diaphragm))
```

Файл game_level.py

Реалізація функціональної задачі створення ігрового рівня

```
script_directory = path.dirname(path.realpath(__file__))


class GameLevel:
    def __init__(self, grid, switch, assets, audio, change_coins, change_health, player_dead, change_diamonds, get_score, get_diamonds, player_alive, get_level_from_where):
        self.display_surface = pygame.display.get_surface()

        self.get_level_from_where = get_level_from_where
        self.switch = switch
        self.switch_locker = True

        self.player_dead = player_dead
        self.player_attack = False
        self.player_alive = player_alive
        self.window_size = self.display_surface.get_size()

        self.all_entities = CameraRig()
        self.treasure_entities = pygame.sprite.Group()
        self.collision_entities = pygame.sprite.Group()
        self.semi_colidable_entities = pygame.sprite.Group()
        self.cannon_entities = pygame.sprite.Group()
        self.cannonball_destroy = pygame.sprite.Group()
        self.mortal_enemy_collisions = pygame.sprite.Group()
        self.damage_entities = pygame.sprite.Group()
        self.damage_entities_cannonball = pygame.sprite.Group()
        self.water_damage = pygame.sprite.Group()
        self.knockback_entities = pygame.sprite.Group()

        self.startup_level(grid, assets, audio['jump'])

        self.change_coins = change_coins
        self.change_health = change_health
        self.change_diamonds = change_diamonds
        self.buttons = Buttons_lvl()
        self.get_score = get_score
        self.get_diamonds = get_diamonds
        self.additional_surf = pygame.Surface(self.window_size)
        self.additional_surf.fill('green')
        self.additional_surf.set_colorkey('green')
        self.alpha = 0
        self.additional_surf.set_alpha(self.alpha)
```

```
                self.score_menu = Score_menu(self.get_score, self.additional_surf)
                self.paused = False
                self.after_pause_timer = Timer(700)

                self.level_borders = {
                'left': -self.window_size[0],
                'right': sorted(list(grid['terrain'].keys()), key = lambda pos: pos[0])[-1][0] + 1000
                }
                self.red_line = sorted(list(grid['terrain'].keys()), key=lambda pos: pos[1])[-1][1] + 100

                self.cloud_frames = assets['clouds']
                self.particle_frames = assets['particle']
                self.cloud_spam_timer = pygame.USEREVENT + 2
                pygame.time.set_timer(self.cloud_spam_timer, 2000)
                self.update_clouds()
                self.complete = False
                self.grid = grid

                self.background_music = audio['music']
                self.background_music.set_volume(0.1)

                self.treasure_sound = audio['treasure']
                self.treasure_sound.set_volume(0.1)

                self.damage_sound = audio['damage']
                self.damage_sound.set_volume(0.1)

        def startup_level(self, grid, assets, jump_sound):
                layer_entity_map = {
                        'terrain': lambda pos, data: EntityBase(pos, assets['soil'][data],

                        [self.all_entities, self.collision_entities, self.cannonball_destroy]),
                        'water': lambda pos, data: AnimatedEntity(assets['water top'], pos,

                          [self.all_entities, self.water_damage], 4)
                        if data == 'top' else EntityBase(pos, assets['water bottom'], [self.all_entities,
self.water_damage], 4),
                        'platforms': lambda pos, _: Platform(pos, assets['platforms'],

                 [self.all_entities, self.semi_colidable_entities]),
                        'bricks fg': lambda pos, _: EntityBase(pos, assets['bricks fg'],

                  [self.all_entities, self.collision_entities, self.cannonball_destroy]),
                        'bricks bg': lambda pos, _: EntityBase(pos, assets['bricks bg'], [self.all_entities]),
                        'interract_info': lambda pos, _: EntityBase(pos, assets['interract_info'],
[self.all_entities], 3),
                        'finish_info': lambda pos, _: EntityBase(pos, assets['finish_info'],
[self.all_entities], 3),
                        'sign bottom': lambda pos, _: EntityBase(pos, assets['sign bottom'],
[self.all_entities], 3),
                        'sign top': lambda pos, _: EntityBase(pos, assets['sign top'], [self.all_entities], 3),
                        'sign left': lambda pos, _: EntityBase(pos, assets['sign left'], [self.all_entities], 3),
                        'sign right': lambda pos, _: EntityBase(pos, assets['sign right'], [self.all_entities],
3),
                }

                for layer_name, layer in grid.items():
                        for pos, data in layer.items():
                                if layer_name in layer_entity_map:
                                        layer_entity_map[layer_name](pos, data)

                                match data:
                                        case 0:
```

```
                self.player = Player(pos, assets['player'],
self.all_entities, self.collision_entities,
                    jump_sound, self.player_dead, self.semi_colidable_entities,
                    self.knockback_entities)
            case 1:
                self.horizon_y = pos[1]
                self.all_entities.horizon_y = pos[1]
            case 4:
                Treasure('gold', assets['golden coin'], pos,
[self.all_entities, self.treasure_entities])
            case 5:
                Treasure('silver', assets['silver coin'], pos,
[self.all_entities, self.treasure_entities])
            case 6:
                Treasure('diamond', assets['diamond'], pos,
[self.all_entities, self.treasure_entities])
            case 7:
                SpikeTrap(assets['spikes'], pos,
[self.all_entities, self.damage_entities])
            case 8:
                Pig(assets['pig'], pos,
                    [self.all_entities,
self.mortal_enemy_collisions, self.knockback_entities],
                    self.collision_entities)
            case 9 | 10:
                Cannon(
                    orientation='left' if data == 9 else
'right',
                    assets=assets['cannon'],
                    pos=pos,
                    group=[self.all_entities,
self.collision_entities, self.cannon_entities],
                    cannonball_surf=assets['cannonball'],
                    damage_entities=self.damage_entities_cannonball
                )
            case 11 | 12 | 13 | 14:
                key = ['small_fg', 'large_fg', 'left_fg',
'right_fg'][data - 11]
                AnimatedEntity(assets['palms'][key], pos,
self.all_entities)
                offset = vector(50, 0) if data == 14 else vector()
                SolidBlock(pos + offset, (76, 50),
self.collision_entities)
            case 15 | 16 | 17 | 18:
                key = ['small_bg', 'large_bg', 'left_bg',
'right_bg'][data - 15]
                AnimatedEntity(assets['palms'][key], pos,
self.all_entities, 3)
            case 21 | 22 | 23 | 24:
                directions = ['bottom', 'top', 'left', 'right']
                Saw(orientation=directions[data - 21],
assets=assets['saw'], pos=pos,
                    group=[self.all_entities,
self.damage_entities])
            case 32:
                radius = 100
                for rad in range(0, radius, 20):
                    Mace(assets=assets['mace_chain'],
pos=pos, group=[self.all_entities], radius=rad)
```

```
                                        Mace(assets=assets['mace'], pos=pos, group=[self.all_entities, self.damage_entities],
                                             radius=radius)
                                case 33:
                                        Treasure('heal_potion', assets['heal_potion'], pos, [self.all_entities, self.treasure_entities])

                for entity in self.cannon_entities:
                        entity.player = self.player

        def player_under_red_line(self):
                if self.player.pos[1] >= self.red_line and not self.player.immortality_timer.is_active() and self.player.player_dead() == False:
                        self.damage_sound.play()
                        self.player.get_hit()
                        self.change_health(100)

        def update_player_hit(self):
                collision_entities = pygame.sprite.spritecollide(self.player, self.damage_entities, False, pygame.sprite.collide_mask)
                if collision_entities and not self.player.immortality_timer.is_active() and self.player.player_dead() == False:
                        self.damage_sound.play()
                        self.player.get_hit()
                        self.change_health(20)

        def update_treasures(self):
                collided_treasures = pygame.sprite.spritecollide(self.player, self.treasure_entities, True)
                for entity in collided_treasures:
                        self.treasure_sound.play()
                        FXParticle(self.particle_frames, entity.rect.center, self.all_entities)
                        if entity.treasure_type == "gold":
                                self.change_coins(50)
                        elif entity.treasure_type == "silver":
                                self.change_coins(10)
                        elif entity.treasure_type == "diamond":
                                self.change_diamonds(1)
                        elif entity.treasure_type == 'heal_potion':
                                self.change_health(-20)

        def update_hit_cannonball(self):
                collision_entities = pygame.sprite.spritecollide(self.player, self.damage_entities_cannonball, True, pygame.sprite.collide_mask)
                pygame.sprite.groupcollide(self.cannonball_destroy, self.damage_entities_cannonball, False, True)
                if collision_entities and not self.player.immortality_timer.is_active() and self.player.player_dead() == False and self.player.animation_name != "attack":
                        self.damage_sound.play()
                        self.player.get_hit()
                        self.change_health(20)

        def update_water_hit(self):
                collision_entities = pygame.sprite.spritecollide(self.player, self.water_damage, False, pygame.sprite.collide_mask)
                if collision_entities and not self.player.immortality_timer.is_active() and self.player.player_dead() == False:
                        self.damage_sound.play()
                        self.player.set_drowning(True)
                        self.change_health(20)
                        self.player.immortality_timer.activate()
                elif collision_entities and self.player.player_dead() == False:
                        self.player.set_drowning(True)
                else:
```

```
                self.player.set_drowning(False)

        def menu_click(self, event):
                if event.type == pygame.MOUSEBUTTONDOWN and self.buttons.mm_rect.collidepoint(mouse_pos()) and self.switch_locker == True:
                        self.switch_locker = False
                        self.switch({'from': 'level', 'to': f'{self.get_level_from_where()}'})
                if event.type == pygame.MOUSEBUTTONDOWN and self.buttons.pause_rect.collidepoint(mouse_pos()) and self.switch_locker == True and self.paused:
                        self.paused = False
                        self.after_pause_timer.activate()
                elif event.type == pygame.MOUSEBUTTONDOWN and self.buttons.pause_rect.collidepoint(mouse_pos()) and self.switch_locker == True and self.paused == False:
                        self.paused = True

        def score_menu_click(self, event):
                if event.type == pygame.MOUSEBUTTONDOWN and self.score_menu.restart_rect.collidepoint(mouse_pos()) and self.switch_locker == True:
                        self.switch_locker = False
                        self.switch({'from': 'level', 'to': 'level'}, self.grid)
                if event.type == pygame.MOUSEBUTTONDOWN and self.score_menu.levels_rect.collidepoint(mouse_pos()) and self.switch_locker == True and self.paused == False:
                        self.switch_locker = False
                        self.switch({'from': 'level', 'to': 'level_menu'})
                if event.type == pygame.MOUSEBUTTONDOWN and self.score_menu.levels_rect.collidepoint(mouse_pos()) and self.switch_locker == True and self.paused:
                        self.paused = False
                        self.after_pause_timer.activate()

        def mortal_enemy_collision(self):
                mortal_collided = pygame.sprite.spritecollide(self.player, self.mortal_enemy_collisions, False, pygame.sprite.collide_mask)
                if mortal_collided:
                        for enemy in mortal_collided:
                                enemy_center = enemy.rect.centery
                                enemy_top = enemy.rect.top
                                player_bottom = self.player.rect.bottom
                                if enemy_top < player_bottom < enemy_center and self.player.animation_name == 'fall' :
                                        self.player.direction.y -= 3.5
                                        enemy.death_timer.activate()
                                        enemy.dead = True
                                elif not enemy_top < player_bottom < enemy_center and not self.player.immortality_timer.is_active():
                                        self.damage_sound.play()
                                        self.player.get_hit()
                                        self.change_health(20)

        def event_loop(self):
                for event in pygame.event.get():
                        if event.type == pygame.QUIT:
                                pygame.quit()
                                sys.exit()
                        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE and self.switch_locker == True:
                                self.switch_locker = False
                                self.switch({'from': 'level', 'to': f'{self.get_level_from_where()}'})
                        if event.type == pygame.KEYDOWN and event.key == pygame.K_p:
                                if self.paused:
                                        self.paused = not self.paused
                                        self.after_pause_timer.activate()
                                else:
                                        self.paused = not self.paused
```

```
                if event.type == self.cloud_spam_timer:
                        cloud_surf = choice(self.cloud_frames)
                        if randint(0, 5) > 3:
                                cloud_surf = pygame.transform.scale2x(cloud_surf)

                        spawn_x = self.level_borders['right'] + randint(200, 500)
                        spawn_y = self.horizon_y - randint(-50, 800)

                        SkyCloud(
                                pos=(spawn_x, spawn_y),
                                surf=cloud_surf,
                                group=self.all_entities,
                                deletion_limit=self.level_borders['left']
                        )

                self.menu_click(event)
                if self.complete or self.player_dead() or self.paused:
                        self.score_menu_click(event)

    def update_clouds(self):
            for _ in range(40):
                    cloud_surf = choice(self.cloud_frames)
                    if randint(0, 5) > 3:
                            cloud_surf = pygame.transform.scale2x(cloud_surf)

                    spawn_x = randint(self.level_borders['left'], self.level_borders['right'])
                    spawn_y = self.horizon_y - randint(-50, 600)

                    SkyCloud(
                            pos=(spawn_x, spawn_y),
                            surf=cloud_surf,
                            group=self.all_entities,
                            deletion_limit=self.level_borders['left']
                    )

    def transition(self):
            self.alpha = max(0, min(self.alpha, 255))
            self.alpha += 10
            self.additional_surf.set_alpha(self.alpha)

    def transition_out(self):
            self.alpha = max(0, min(self.alpha, 255))
            self.alpha -= 10
            self.additional_surf.set_alpha(self.alpha)

    def run(self, dt):
            self.event_loop()
            if self.paused:
                    self.transition()
                    self.score_menu.display(False, self.get_diamonds, True)
            else:
                    self.all_entities.update(dt)
                    self.update_player_hit()
                    self.update_hit_cannonball()
                    self.update_water_hit()
            if self.after_pause_timer.is_active():
                    self.transition_out()
            self.update_treasures()
            self.mortal_enemy_collision()
            self.player_under_red_line()

            self.display_surface.fill('#cce7ff')
            self.all_entities.manage_layers(self.player)
```

```
                if self.paused:
                        self.buttons.display(True)
                else:
                        self.buttons.display(False)
                self.display_surface.blit(self.additional_surf, (0, 0))
                self.after_pause_timer.update()
                if self.player_dead():
                        self.transition()
                        self.score_menu.display(False, self.get_diamonds)
                if self.complete:
                        self.transition()
                        self.score_menu.display(True)
                        self.player.speed = 0

class CameraRig(pygame.sprite.Group):
        def __init__(self):
                super().__init__()
                self.display_surface = pygame.display.get_surface()
                self.window_size = self.display_surface.get_size()
                self.offset = vector()

        def render_scenery(self):
                horizon_pos = self.horizon_y - self.offset.y
                if horizon_pos < 0:
                        self.display_surface.fill('#a8c9a1')
                elif horizon_pos < self.window_size[1]:
                        sea_rect = pygame.Rect(0, horizon_pos, self.window_size[0],
self.window_size[1] - horizon_pos)
                        pygame.draw.rect(self.display_surface, '#a8c9a1', sea_rect)
                        pygame.draw.rect(self.display_surface, '#f6d6bd', (0, horizon_pos - 10,
self.window_size[0], 10))
                        pygame.draw.rect(self.display_surface, '#f6d6bd', (0, horizon_pos - 16,
self.window_size[0], 4))
                        pygame.draw.rect(self.display_surface, '#f6d6bd', (0, horizon_pos - 20,
self.window_size[0], 2))
                        pygame.draw.line(self.display_surface, '#f5f1de', (0, horizon_pos),
(self.window_size[0], horizon_pos), 3)

        def manage_layers(self, player):
                self.offset.x = player.rect.centerx - self.window_size[0] / 2
                self.offset.y = player.rect.centery - self.window_size[1] / 2

                for entity in self:
                        if entity.z == 1:
                                offset_rect = entity.rect.copy()
                                offset_rect.center -= self.offset
                                self.display_surface.blit(entity.image, offset_rect)

                self.render_scenery()

                for layer in range(2, 6):
                        for entity in self:
                                if entity.z == layer:
                                        offset_rect = entity.rect.copy()
                                        offset_rect.center -= self.offset
                                        self.display_surface.blit(entity.image, offset_rect)

class Buttons_lvl:
        def __init__(self):
                self.display_surface = pygame.display.get_surface()
                self.window_size = self.display_surface.get_size()
                self.multiplayer_x = self.window_size[0] / 1280
                self.multiplayer_y = self.window_size[1] / 720
```

```
                self.create_buttons()


        def create_buttons(self):
                def sx(value):
                        return int(value * self.multiplayer_x)
                def sy(value):
                        return int(value * self.multiplayer_y)
                mm_size = sx(45)
                mm_margin = sx(6)
                mm_topleft = (self.window_size[0] - mm_size - mm_margin, mm_margin)
                self.mm_rect = pygame.Rect(mm_topleft, (mm_size, mm_size))
                self.image = load(path.join(script_directory, 'graphics', 'menus',
'main_menu_button.png')).convert_alpha()
                self.image = pygame.transform.scale(self.image, (mm_size, mm_size))
                self.pause_rect = pygame.Rect(vector(mm_topleft) - (mm_size + mm_margin, 0), (mm_size,
mm_size))
                self.p_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu',
'pause_button.png')).convert_alpha()
                self.p_image = pygame.transform.scale(self.p_image, (mm_size, mm_size))
                self.r_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu',
'release_pause_button.png')).convert_alpha()
                self.r_image = pygame.transform.scale(self.r_image, (mm_size, mm_size))


        def display(self, paused = False):
                self.display_surface.blit(self.image, self.mm_rect.topleft)
                if paused:
                        self.display_surface.blit(self.r_image, self.pause_rect.topleft)
                else:
                        self.display_surface.blit(self.p_image, self.pause_rect.topleft)

class Score_menu:
        def __init__(self, get_score, surf):
                self.display_surface = surf
                self.get_score = get_score
                self.window_size = self.display_surface.get_size()
                self.multiplayer_x = self.window_size[0] / 1280
                self.multiplayer_y = self.window_size[1] / 720
                self.create_buttons()



        def create_buttons(self):
                def sx(value):
                        return int(value * self.multiplayer_x)
                def sy(value):
                        return int(value * self.multiplayer_y)
                topleft = (self.window_size[0] / 2 - sx(360) / 2, sy(120))
                self.tittle_rect = pygame.Rect(topleft, (sx(360), sy(100)))
                self.image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu',
'win.png')).convert_alpha()
                self.image = pygame.transform.scale(self.image, (sx(360), sy(100)))
                self.image_go = load(path.join(script_directory, 'graphics', 'menus', 'score_menu',
'game_over.png')).convert_alpha()
                self.image_go = pygame.transform.scale(self.image_go, (sx(360), sy(100)))
                self.image_paused = load(path.join(script_directory, 'graphics', 'menus', 'score_menu',
'paused.png')).convert_alpha()
                self.image_paused = pygame.transform.scale(self.image_paused, (sx(360), sy(100)))
                self.score_rect = pygame.Rect(vector(topleft) + (sx(0), sy(110)), (sx(360), sy(240)))
                self.score_rect_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu',
'score_back.png')).convert_alpha()
                self.score_rect_image = pygame.transform.scale(self.score_rect_image, (sx(360), sy(240)))
```

```
                self.diamonds0_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu', '0diamond_score.png')).convert_alpha()
                self.diamonds0_image = pygame.transform.scale(self.diamonds0_image, (sx(360), sy(240)))
                self.diamonds1_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu', '1diamond_score.png')).convert_alpha()
                self.diamonds1_image = pygame.transform.scale(self.diamonds1_image, (sx(360), sy(240)))
                self.diamonds2_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu', '2diamond_score.png')).convert_alpha()
                self.diamonds2_image = pygame.transform.scale(self.diamonds2_image, (sx(360), sy(240)))
                self.buttons_rect = pygame.Rect(vector(topleft) + (sx(0), sy(360)), (sx(360), sy(120)))
                self.buttons_rect_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu', 'buttons_back.png')).convert_alpha()
                self.buttons_rect_image = pygame.transform.scale(self.buttons_rect_image, (sx(360), sy(120)))
                self.restart_rect = pygame.Rect(vector(topleft) + (sx(50), sy(396)), (sx(120), sy(46)))
                self.restart_rect_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu', 'restart.png')).convert_alpha()
                self.restart_rect_image = pygame.transform.scale(self.restart_rect_image, (sx(120), sy(46)))
                self.levels_rect = pygame.Rect(vector(topleft) + (sx(190), sy(396)), (sx(120), sy(46)))
                self.levels_rect_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu', 'levels.png')).convert_alpha()
                self.levels_rect_image = pygame.transform.scale(self.levels_rect_image, (sx(120), sy(46)))
                self.resume_rect_image = load(path.join(script_directory, 'graphics', 'menus', 'score_menu', 'resume.png')).convert_alpha()
                self.resume_rect_image = pygame.transform.scale(self.resume_rect_image, (sx(120), sy(46)))
                self.font = pygame.font.Font(path.join(script_directory, 'graphics', 'ui', 'ARCADEPI.ttf'), sx(18))



        def display(self, level_complete, get_diamonds = None, paused = False):
                def sx(value):
                        return int(value * self.multiplayer_x)
                def sy(value):
                        return int(value * self.multiplayer_y)
                self.score_surf = self.font.render(f"Score: {self.get_score()}", False, '#33323d')
                self.score_surf_rect = self.score_surf.get_rect(topleft=(vector(self.window_size[0] / 2 - sx(360) / 2, sy(120)) + (sx(120), sy(255))))
                self.display_surface.blit(self.buttons_rect_image, self.buttons_rect.topleft)
                if level_complete:
                        self.display_surface.blit(self.image, self.tittle_rect.topleft)
                        self.display_surface.blit(self.score_rect_image, self.score_rect.topleft)
                else:
                        if paused:
                                self.display_surface.blit(self.image_paused, self.tittle_rect.topleft)
                        else:
                                self.display_surface.blit(self.image_go, self.tittle_rect.topleft)
                        if get_diamonds() == 0:
                                self.display_surface.blit(self.diamonds0_image, self.score_rect.topleft)
                        elif get_diamonds() == 1:
                                self.display_surface.blit(self.diamonds1_image, self.score_rect.topleft)
                        if get_diamonds() == 2:
                                self.display_surface.blit(self.diamonds2_image, self.score_rect.topleft)
                self.display_surface.blit(self.restart_rect_image, self.restart_rect.topleft)
                if paused:
                        self.display_surface.blit(self.resume_rect_image, self.levels_rect.topleft)
                else:
                        self.display_surface.blit(self.levels_rect_image, self.levels_rect.topleft)
                self.display_surface.blit(self.score_surf, self.score_surf_rect)
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

Класична гра у жанрі платформер, з розробкою редактора рівнів
(комплексна тема)
Програма та методика тестування
КПІ.ІП-1217.045480.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав ЛОГВИНЕНКО

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є класична гра у жанрі Platformer, з розробкою редактора рівнів на рушії Pygame та її поведінка на платформі Windows.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка коректності функціонування програмного забезпечення відповідно до встановлених функціональних вимог;
- виявлення проблем, помилок і недоліків з подальшим їх усуненням;
- оцінювання зручності та ергономічності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- тестування на відповідність функціональним вимогам;
- тестування на персональних комп’ютерах з різною роздільною здатністю екрану;
- тестування інтерфейсу користувача;
- тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

Класична гра у жанрі платформер, з розробкою редактора рівнів
(комплексна тема)
Керівництво користувача
КПІ.ІП-ІП-1217.045480.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав ЛОГВИНЕНКО

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«LevelKing» – це комп’ютерна гра у жанрі класичного платформера, створена для операційної системи Windows, що поєднує динамічний геймплей із можливістю творчого самовираження. Ви керуєте персонажем, долаєте ворогів і пастки, створюєте власні рівні у вбудованому редакторі, а також можете проходити рівні, створені іншими гравцями. Крім того, гра дозволяє надсилати пропозиції змін до чужих рівнів - автор рівня отримає вашу версію з коментарем і зможе або прийняти зміни, або відхилити їх, що сприяє спільній творчості та постійному вдосконаленню контенту.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3;

– об‘єм ОЗП: 4 Гб;

– об’єм вільної пам’яті накопичувача: 300 Мб.

– швидкість інтернету 5 мб/с

Рекомендована конфігурація технічних засобів (та на якій виконувалась розробка) :

– тип процесору: Intel Core i5;

– об‘єм ОЗП: 16 Гб;

– об’єм вільної пам’яті накопичувача: 300 Мб.

– швидкість інтернету 10 мб/с

2.2 Завантаження застосунку

Для використання комп’ютерної гри LevelKing необхідно завантажити інсталяційний файл LevelKingSetup.exe, доступний на сторінці релізів проєкту на платформі GitHub (рис. 2.1).

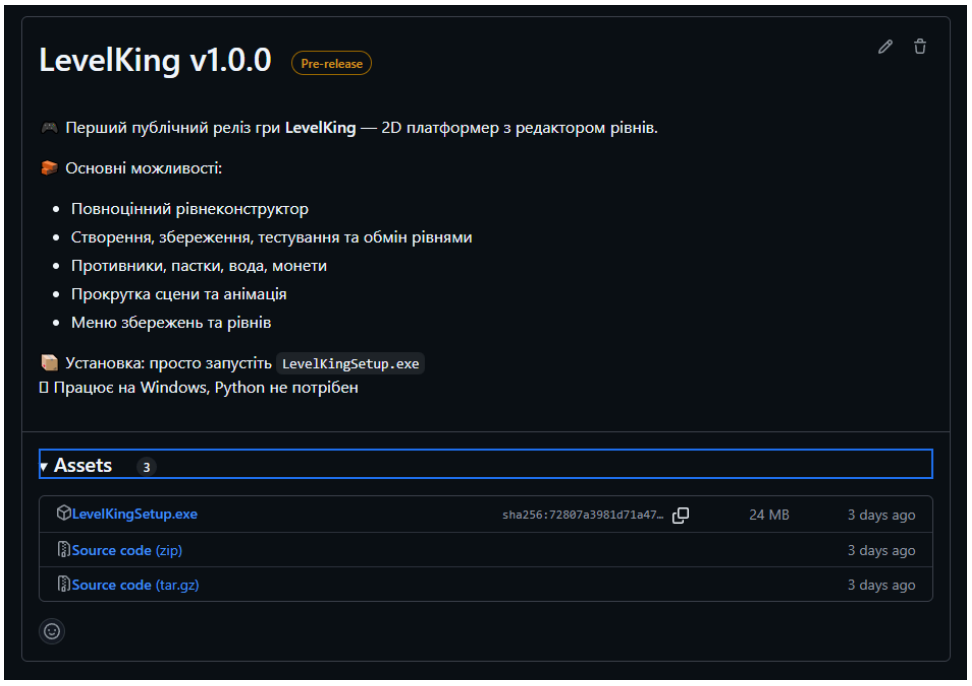


Рисунок 2.1 – Сторінка релізу

2.3 Перевірка коректної роботи

Після завершення встановлення гри, у вказаній директорії має з’явитися виконуваний файл «Level King.exe», запуск якого відкриє головне меню гри (рис. 2.2). У разі відсутності ярлика або файлу, рекомендується перевстановити гру або перевірити налаштування антивірусу, який міг помилково заблокувати виконуваний файл.

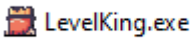


Рисунок 2.2 – Вміст встановленої директорії гри LevelKing у Windows

3 ВИКОНАННЯ ПРОГРАМИ

При запуску комп’ютерної гри перед користувачем повстає головне меню, з якого можна вийти з гри (кнопка «Exit»), перейти до меню готових рівнів(кнопка «Play»), перейти до редактора рівнів(кнопка «Editor»), перейти до меню власних рівнів(кнопка «Load») та перейти до меню обміну рівнями (кнопка «Globe»)(рис. 3.1)..

Рисунок 3.1 – Головне меню

У меню готових рівнів користувач може обрати один з 4 заготовлених рівнів для проходження (рис. 3.2). Кожен з цих рівнів слугує «посібником» по певній ігровій механіці. Розраховано, що пройшовши всі 4 заготовлені рівня. користувач ознайомиться з основними правилами гри і буде готовий створювати власні рівні у редакторі рівнів, або проходити рівні від інших користувачів, завантаживши їх з меню обміну рівнями.

Рисунок 3.2 – Меню готових рівнів

У меню власних рівнів користувач може побачити свої локальні створені рівнів(рис. 3.3). Імена цих рівнів - дати та час їх створення користувачем. Користувач може проходити їх(натиснувши ліву кнопку миші по комірці), видаляти (натиснувши праву кнопку миші по комірці), та перезаписувати (натиснувши ліву кнопку миші по зайнятій комірці, перейшовши у це меню з редактора рівнів). У разі якщо користувач ще не користувався редактором рівнів і не зберігав власні рівні, комірки в цьому меню будуть пустими.

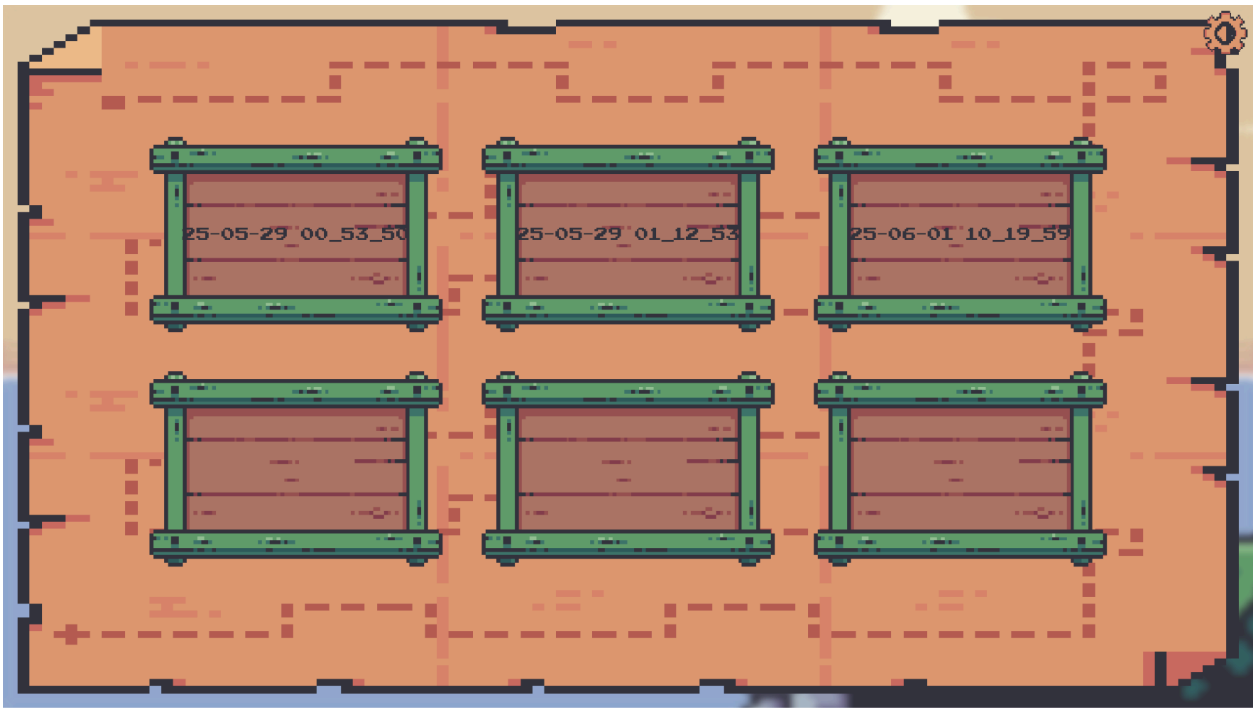


Рисунок 3.3 – Меню готових рівнів

Користувач може почати проходження рівня, обравши його у одному з вищевказаних меню. Опинившись на рівні, користувачу надається можливість управляти ігровим персонажем. Також відображається інтерфейс ігрового персонажа, в який входить індикатор здоров’я та індикатор рахунку персонажа. Також користувач може натиснути на кнопку повернення до останнього меню («Gear»), або на кнопку паузи («Play/Pause»)(рис. 3.4).

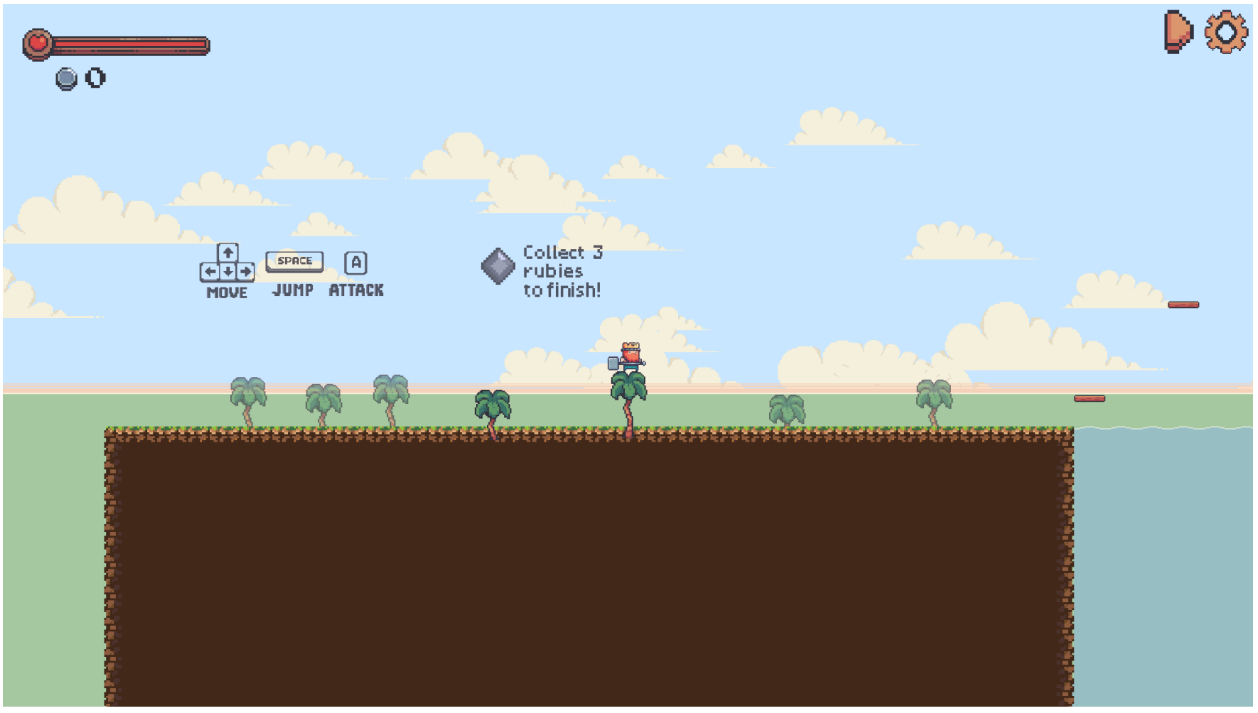


Рисунок 3.4 – Меню готових рівнів

На рівні, користувач може зустріти різноманітних ворогів та потрапити у пастки. У разі якщо персонаж потрапить під атаку ворога або у пастку, він втратить очки здоров’я. Користувач має змогу протистояти ворогам, атакуючи їх, знищуючи їх самих або їх снаряди. Розглянемо ворогів більш детально.

Ворог-свиня є динамічним ворогом, що переміщується по платформам. Якщо користувач буде не обережним і зіткнеться з ворогом свинею, то персонаж втратить певну кількість здоров’я. В той же час, ворога свиню можна доволі легко подолати або обійти. Персонаж може перенаправити ворога свиню у іншу сторону, у разі якщо користувач попаде своєю атакою по ньому. Також, ворога свиню можна легко знищити виконавши «наскок» на нього(рис. 3.5).

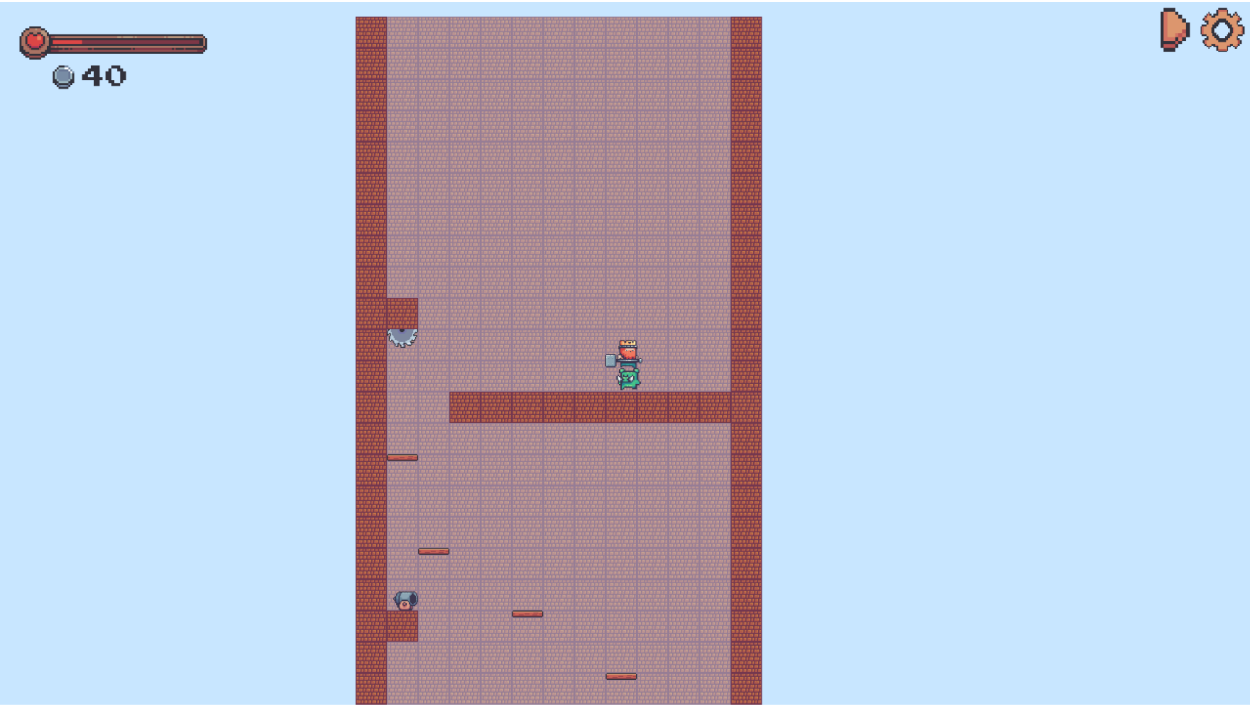


Рисунок 3.5 – Знищення ворога-свині

Ворог-гармата вистрілює снарядами у ту сторону в яку він розташований. Ігровий персонаж провокує ворога-гармату на атаку, підійшовши на певну відстань. Хоч ворога-гармату не можна вбити, персонаж може легко уникнути шкоди від нього перепрыгнув снаряд, або знищити снаряд своєю атакою(рис. 3.6).

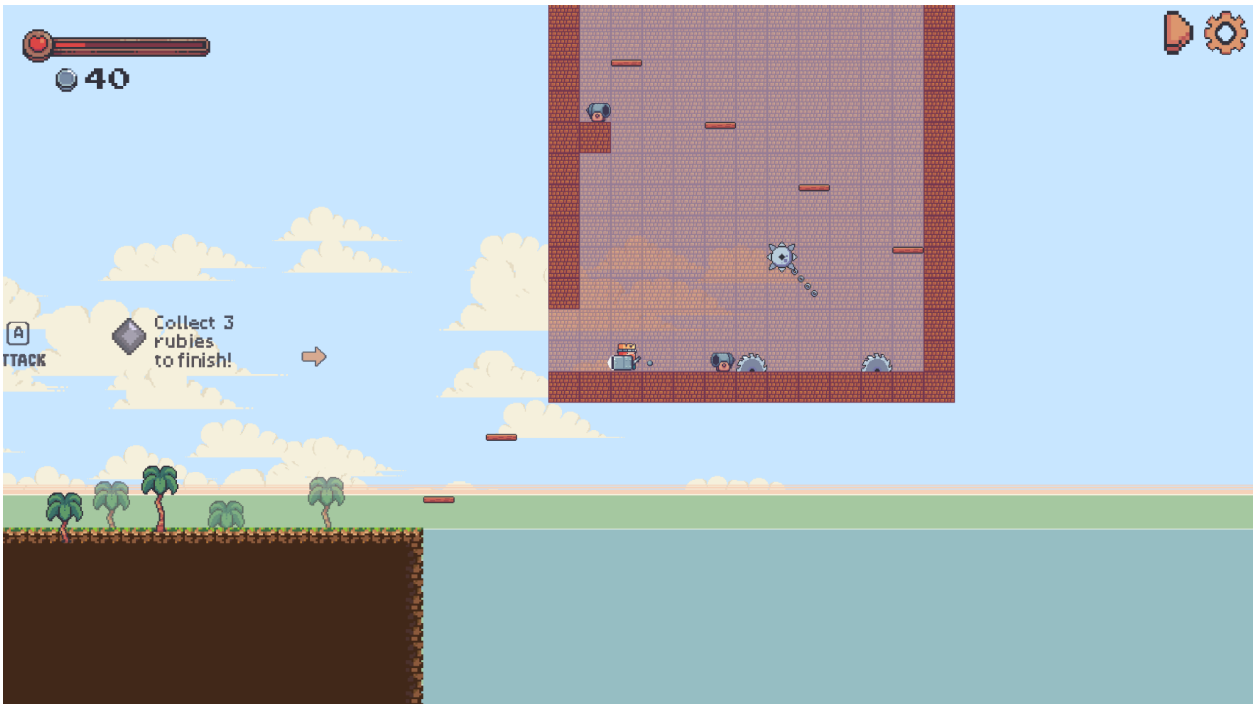


Рисунок 3.6 – Знищення снаряду гармати

Також на рівні розташовані пастки, що працюють за незрозумілою логікою: попав у пастку - отримав шкоду. Найнебезпечнішою пасткою для персонажа є обертова пастка «Масе». Ця пастка обертає навколо себе велику булаву, під яку дуже легко потрапити. (рис. 3.7).

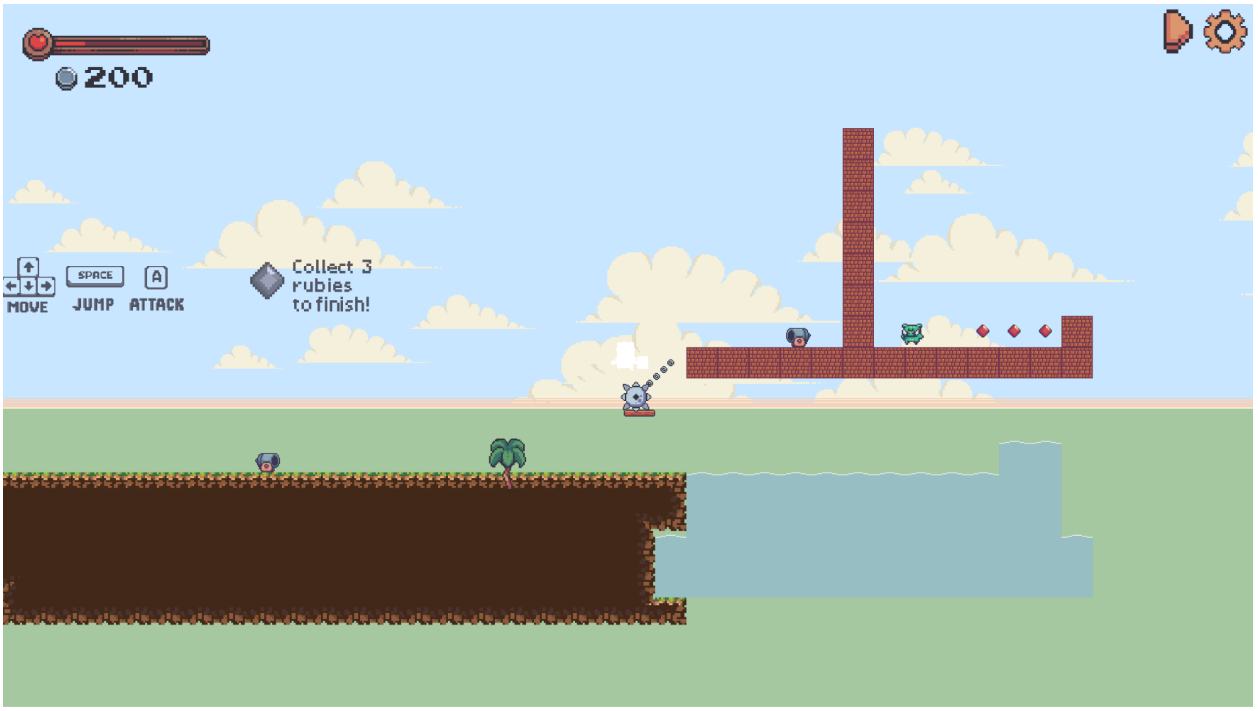


Рисунок 3.7– Обертова пастка «Масе»

За різновид пастики також можна вважати воду розташовану на рівні. Якщо ігровий персонаж потрапляє у неї, це уповільнити його рухи, а також персонаж буде отримувати періодичну шкоду під час знаходження у воді(рис. 3.8).

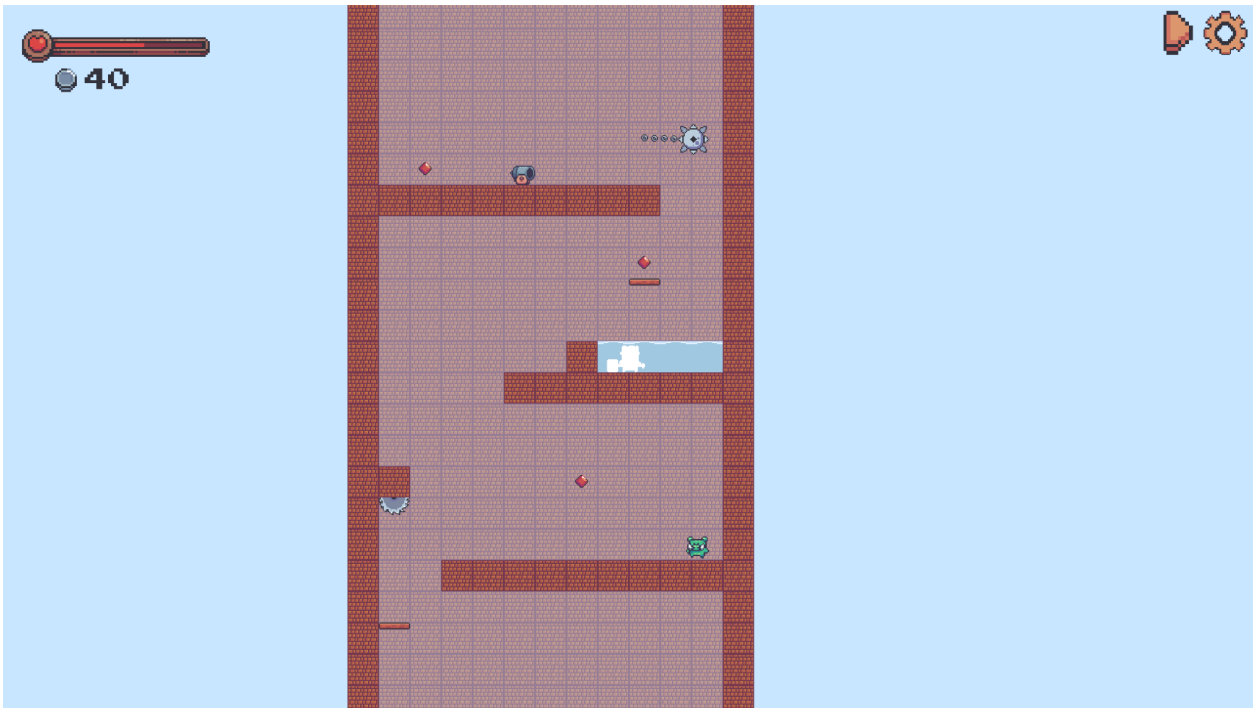


Рисунок 3.8 – Взаємодія з водою

Якщо здоров'я ігрового персонажа впаде нижче нуля, він помре і користувач побачить меню програшу(рис. 3.9).

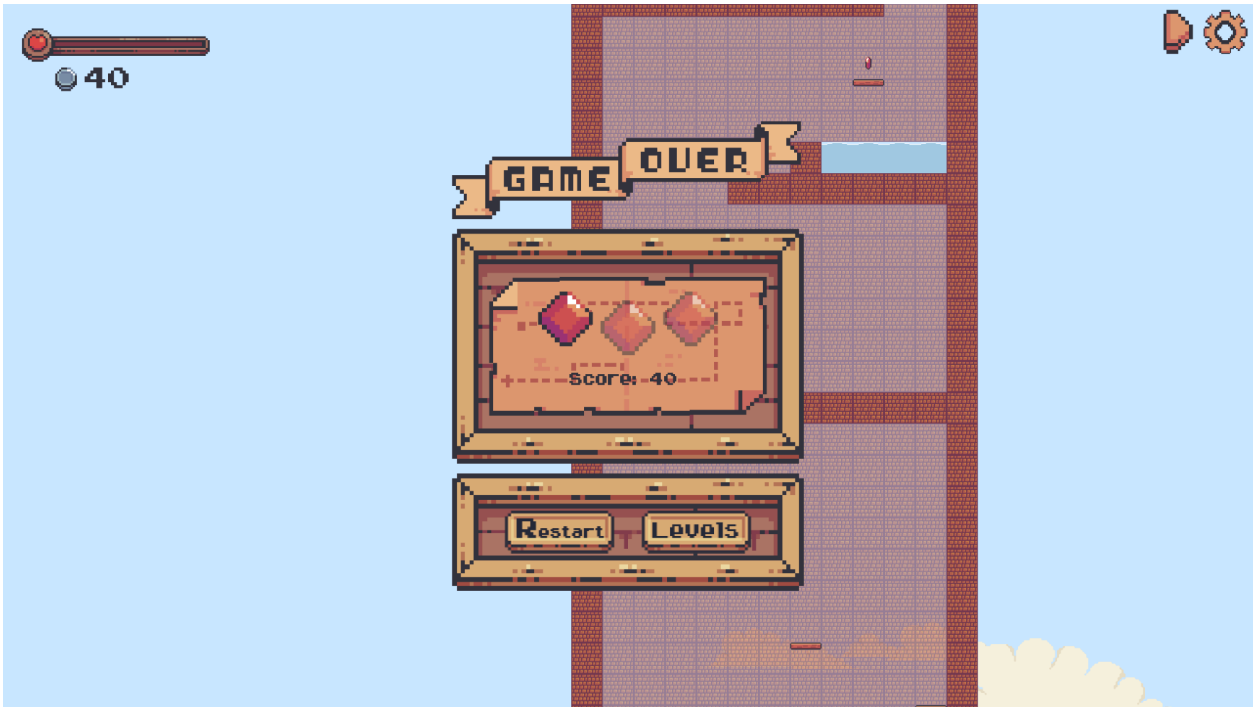


Рисунок 3.9 – Меню програшу

У будь-який момент гри користувач може викликати меню паузи, натиснувши клавішу «P» або іконку паузи на екрані. Це відкриє меню паузи, де буде показано основну інформацію - кількість зібраних діамантів і монет (score), а також з’явиться можливість продовжити гру або почати рівень заново (рис. 3.10). Крім того, натиснувши на іконку шестерні у верхньому правому куті, користувач може повернутися до головного меню.

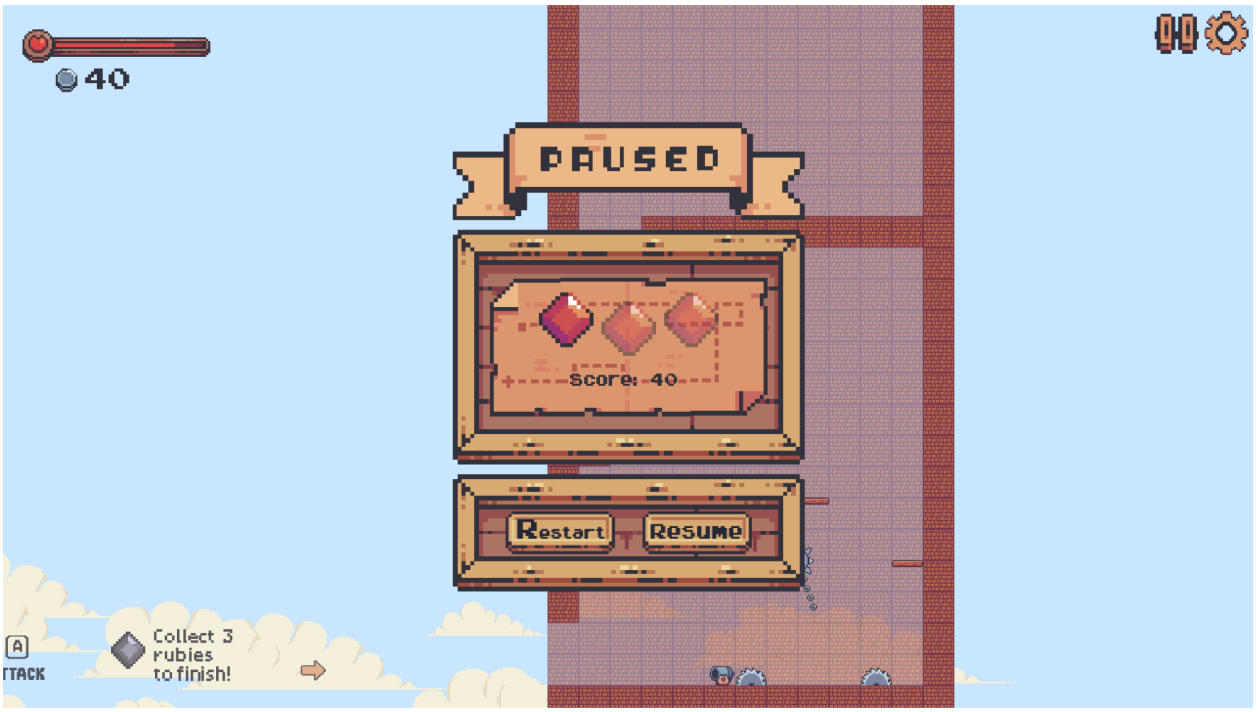


Рисунок 3.10 – Меню паузи

Також під час проходження користувач може збирати ігрові об’єкти. Підбір монет збільшує рахунок персонажа. Золота монета варта 50 очок, а срібна 10. Збір зілля здоров’я відновить певну кількість втраченого здоров’я. Збір кристалів буде наближати користувача до повного проходження рівня. Задля перемоги, користувач повинен зібрати 3 кристали. При виконанні цієї умови, на екран вивдеться меню перемоги, де користувач побачить свій рахунок і може перепройти рівень(«Restart»), або повернутися до меню рівнів («Levels»)(рис. 3.11).

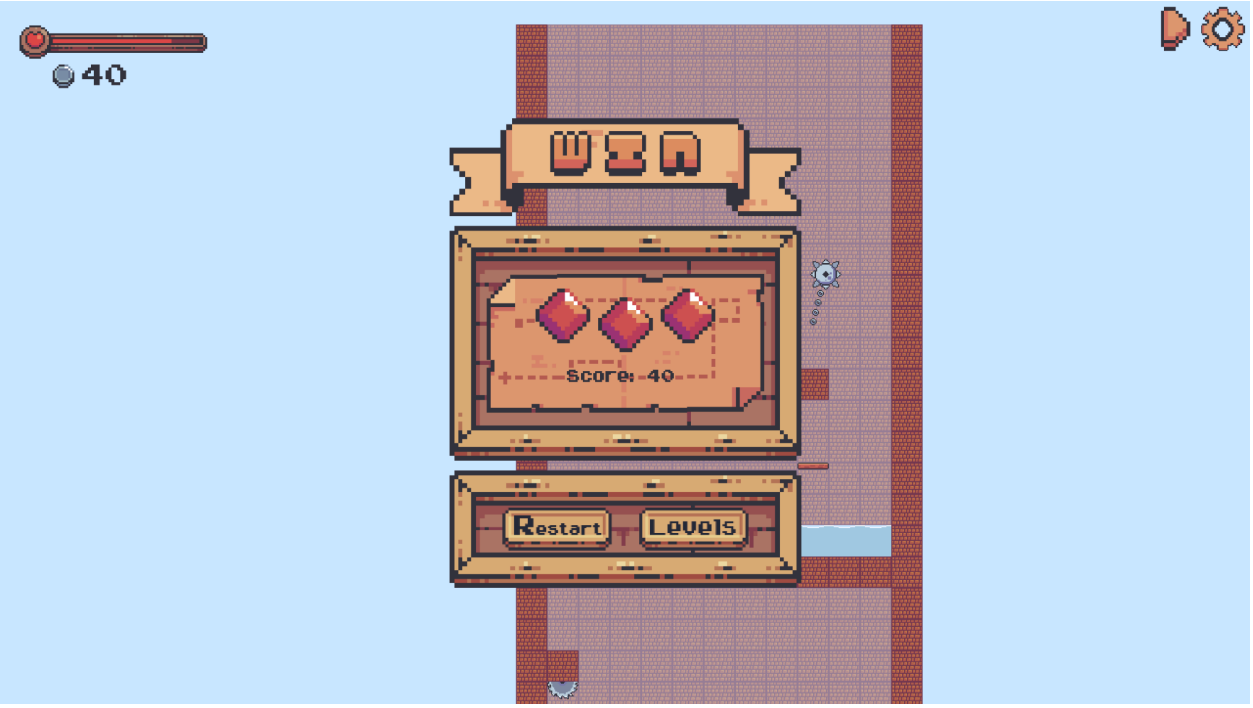


Рисунок 3.11 – Перемога

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

Класична гра у жанрі платформер, з розробкою редактора рівнів
(комплексна тема)
Графічний матеріал
КПІ.ІП-1217.045480.06.99

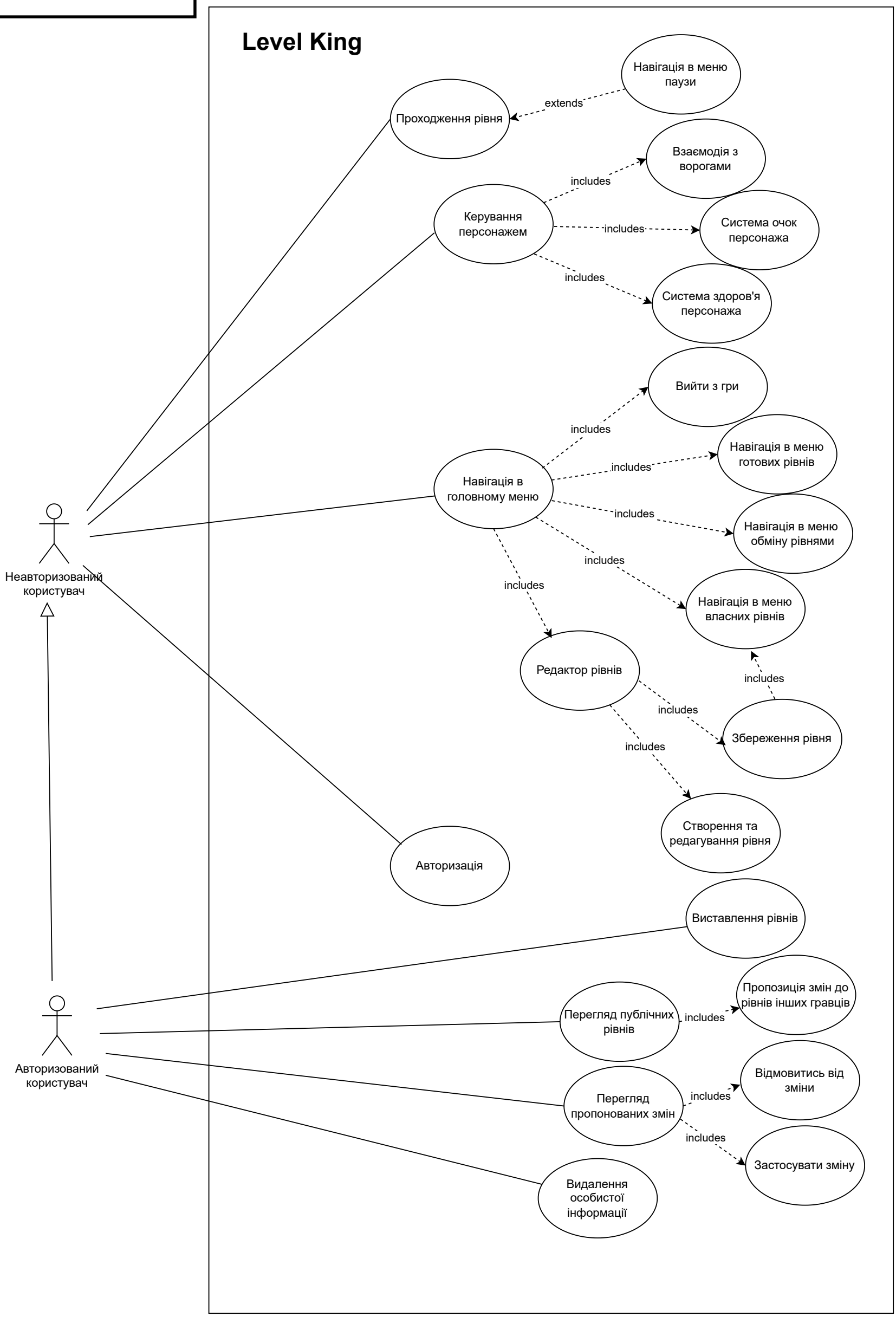
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав ЛОГВИНЕНКО

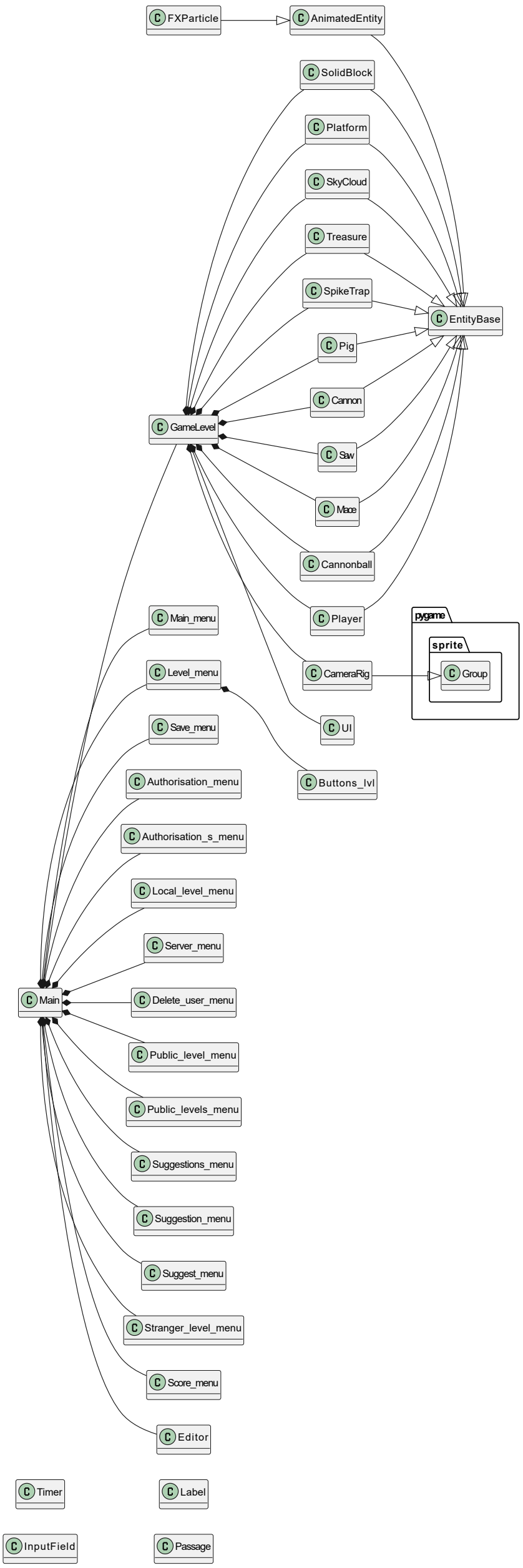
Київ – 2025

КПІ.ІП-1217.045480.06.99.ССВ



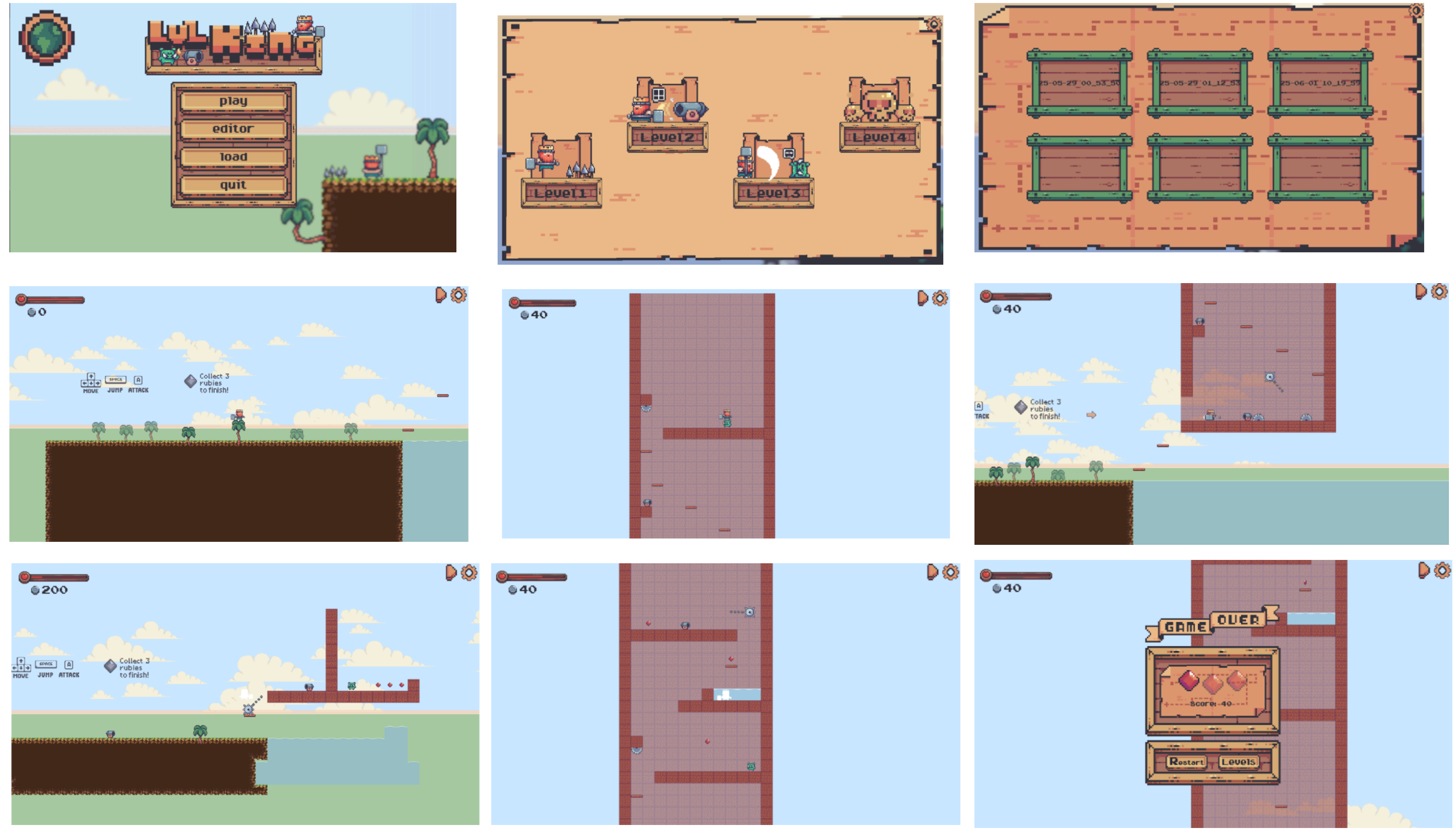
					КПІ.ІП-1217.045480.06.99.ССВ		
						Лит.	Маса	Масштаб
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна варіантів використань			
Розробив		Логвиненко В.О.				Аркуш 1	Аркушів 1	
Перевірив		Дифучин А.Ю.						
Т. контр.								
					Класична гра у жанрі платформер, з розробкою редактора рівнів(комплексна тема)	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12		
Н. контр.		Головченко М.М.						
Затвердив		Жаріков Е.В.						

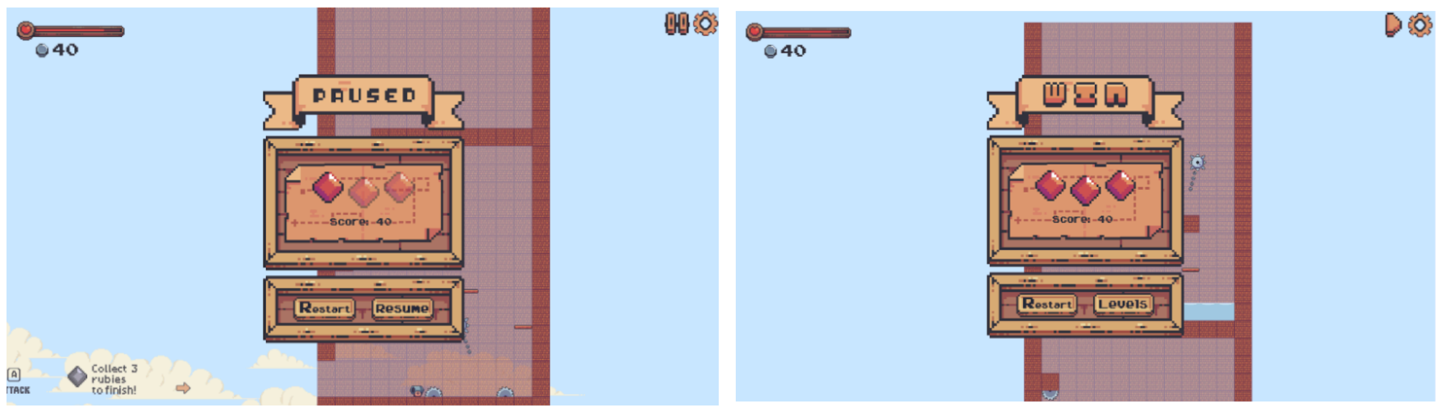
КПІ.ІП-1217.045480.06.99.ССК



					КПІ.ІП-1217.045480.06.99.ССК
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна класів програмного забезпечення
Розробив		Логвиненко В.О.			Лит. / Маса / Масштаб
Перевірив		Дифучин А.Ю.			Аркуш 1 / Аркушів 1
Т. контр.					Класична гра у жанрі платформер, з розробкою редактора рівнів(комплексна тема)
					КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12
Н. контр.		Головченко М.М.			
Затвердив		Жаріков Е.В.			

КПІ.ІП-1217.045480.06.99.КЕ





					КПІ.ІП-1217.045480.06.99.КЕ
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Логвиненко В.О.			Креслення вигляду екранних форм
Перевірив		Дифучин А.Ю.			
Т. контр.					
Н. контр.		Головченко М.М.			Класична гра у жанрі платформер, з розробкою редактора рівнів(комплексна тема)
Затвердив		Жаріков Е.В.			

Літера	Маса	Масштаб
Аркуш 1		Аркушів 1

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-12