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)

Шоман Данило Володимирович _____
(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

Шоману Данилу Володимировичу
(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) Предпроєктне обстеження предметної області: постановка завдання дипломного проєктування, аналіз предметної області, аналіз існуючих рішень, аналіз та моделювання бізнес-процесів.
2) Інформаційне забезпечення: варіанти використання програмного забезпечення, розроблення функціональних вимог, розроблення нефункціональних вимог, аналіз системних вимог, аналіз економічних показників програмного забезпечення, постановка завдання на розробку програмного забезпечення.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, 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 архітектури програмного забезпечення
2) Схема сутностей бази даних
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 _____ Данило ШОМАН
(підпис) (ініціали, прізвище)

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

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 39 таблиць, 32 рисунка та 20 джерел – загалом 86 сторінок.

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

Мета – розвиток 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.

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, СЕРВЕР, DJANGO, БАЗА ДАНИХ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 39 tables, 32 figures and 20 sources – in total 86 pages.

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

The goal is to develop the strategic and logical skills of the player by passing and building the levels of the game, as well as the development of socialization by spreading their own levels and passing the levels of other people.

In the section «Preliminary examination of the subject area» the statement of the task of diploma design is considered, the analysis of the subject area is carried out, an overview of existing software and technical solutions is carried out, and the corresponding business processes are modeled.

The section «Information Support» is devoted to the definition of software use cases, the development of functional and non-functional requirements, the analysis of system and economic indicators, as well as the formulation of the task for software development.

The section «Software Design and Development» is devoted to describing the architecture of the client and server parts, justifying the choice of the technology stack, implementing key software components and analyzing data security.

The section «Quality Analysis and Software Testing» is devoted to assessing the quality of the software product, describing the tests performed and the control example.

The «Software Deployment and Maintenance» section describes the process of deploying software and organizing its maintenance after implementation.

Software implemented for personal computers on Windows Platform

KEYWORDS: MOBILE APP, WEARABLE DEVICES, ANDROID, ANDROID STUDIO, GOOGLE FIREBASE, DATABASE, STATISTICS.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1230.045480.01.91	Технічне завдання	18	
3	А4	КПІ.ІП-1230.045480.02.81	Пояснювальна записка	86	
4	А4	КПІ.ІП-1230.045480.03.12	Текст програми	29	
5	А4	КПІ.ІП-1230.045480.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1230.045480.05.34	Керівництво користувача	12	
7	А3	КПІ.ІП-1230.045480.06.99	Графічний матеріал	3	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1230.045480.00.90		
Розроб.		Шоман Д. В.			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:

_____ Данило ШОМАН

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 і креативного мислення користувача за рахунок створення власних рівнів і обміну ними та пропозиціями до рівнів інших.

Метою розробки є розширення функціональності гри платформеру за рахунок підтримки спільної творчості гравців та можливості редагувати власні рівні.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– Меню входу, що містить кнопки із наступним функціоналом: увійти, зареєструватися; поля для введення логіну і паролю (рис. 4.1);

Рисунок 4.1 – Меню входу

– Меню реєстрації, що містить кнопки із наступним функціоналом: назад, зареєструватися; поля для введення логіну і паролю (рис. 4.2);

Рисунок 4.2 – Меню реєстрації

– Меню сервера, що містить інтерактивні списки з публічними та локальними рівнями користувача, при натисненні на які, відбувається перехід на меню відповідного публічного чи локального рівня; кнопки із наступним функціоналом: перейти до публічних рівнів інших гравців, перейти до пропозицій інших гравців, видалити свої дані (рис. 4.3);

Рисунок 4.3 – Меню сервера

– Меню локального рівня, що містить кнопки із наступним функціоналом: редагувати рівень, викласти рівень у публічний доступ; поля для введення назви та опису рівня (рис. 4.4);

Рисунок 4.4 – Меню локального рівня

– Меню публічного рівня, що містить кнопки із наступним функціоналом: редагувати рівень, зберегти рівень, видалити рівень; поля для введення назви та опису рівня (рис. 4.5);

Рисунок 4.5 – Меню публічного рівня

– Меню публічних рівнів інших гравців, що містить інтерактивний список з рівнями інших гравців, при натисненні на які, відбувається перехід на меню відповідного рівня іншого гравця (рис. 4.6);

Рисунок 4.6 – Меню публічних рівнів інших гравців

– Меню пропозицій інших гравців, що містить інтерактивний список з пропозиціями змін до власних рівнів від інших гравців, при натисненні на які, відбувається перехід на меню відповідної пропозиції (рис. 4.7);

Рисунок 4.7 – Меню пропозицій інших гравців

– Меню видалення власних даних, що містить дисклеймер про видалення даних; кнопки із наступним функціоналом: видалити дані, відмінити (рис. 4.8);

Рисунок 4.8 – Меню видалення власних даних

– Меню рівня іншого гравця, що містить кнопки із наступним функціоналом: спробувати, запропонувати зміни; назву рівня і його опис (рис. 4.9);

Рисунок 4.9 – Меню рівня іншого гравця

– Меню створення пропозиції, що містить кнопки із наступним функціоналом: редагувати рівень, запропонувати зміни; назву рівня; поле для введення коментаря до змін (рис. 4.10);

Рисунок 4.10 – Меню створення пропозиції

– Меню пропозиції від іншого гравця, що містить кнопки із наступним функціоналом: спробувати рівень, погодитись з пропозицією, скасувати пропозицію (рис. 4.11);

Рисунок 4.11 – Меню пропозиції від іншого гравця

– Редактор рівнів, що містить сітку для розташування елементів, інтерактивні об’єкти ігрового персонажа та горизонту на полотні, кнопки для вибору елементів для розташування однієї із категорій: ландшафт, скарби, вороги, цегла, пальми, інформація (рис. 4.12);

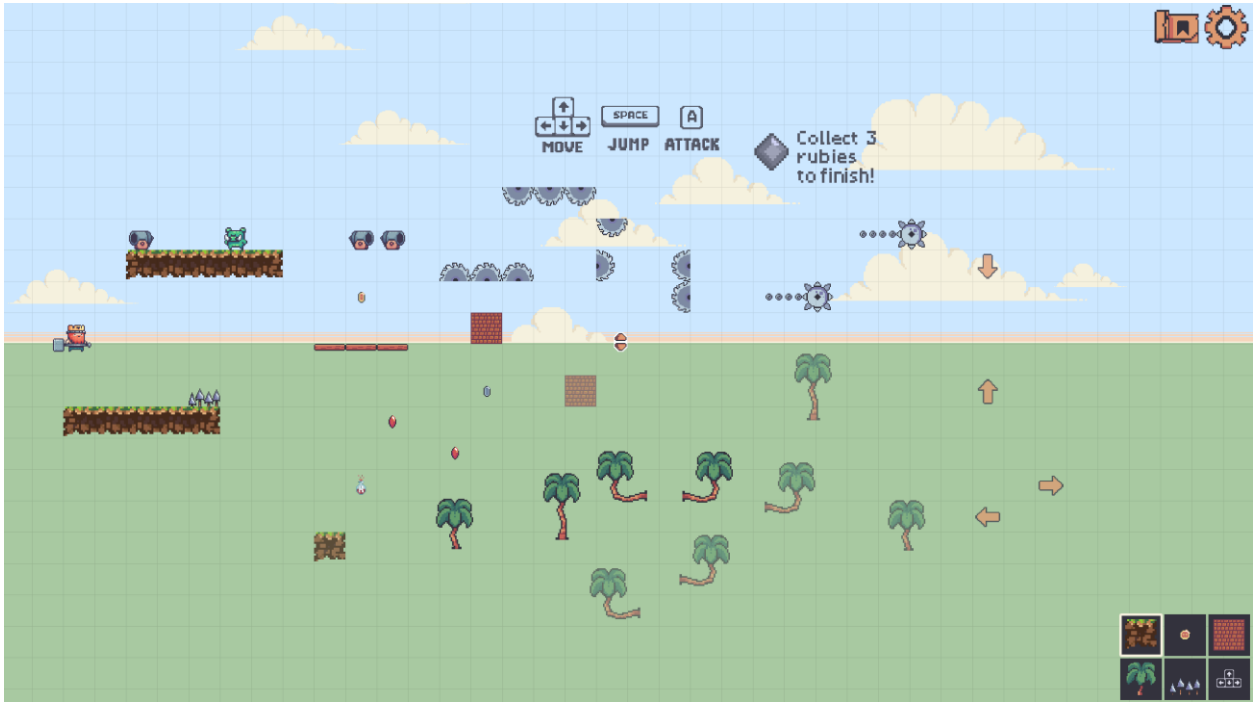


Рисунок 4.12 – Редактор рівнів

4.1.2 Облік користувачів:

4.1.2.1 Для авторизованого користувача:

– функція редагування рівня;

– функція викладення власних рівнів на сервер, з подальшою можливістю їх видалити, змінити або зберегти;

– функція перегляду та випробування рівнів інших гравців

– функція видалення власних даних;

– функція пропонування змін до рівнів інших;

– функція перегляду та прийняття або відхилення запитів на зміну від інших гравців;

4.1.2.2 Для неавторизованого користувача:

– функція редагування рівня;

– функція входу у власний аккаунт

– функція реєстрації нового аккаунту

4.2 Вимоги до надійності

Передбачити контроль введення даних та обмеження прав доступу для запобігання несанкціонованим діям користувачів. Реалізувати авторизацію за допомогою JWT та захищене зберігання паролів із хешуванням (PBKDF2 з сіллю). Усунути потенційні вразливості за допомогою системи дозволів Django REST Framework, ORM для захисту від SQL-ін’єкцій та обробки CSRF-захисту. Контейнеризувати серверну частину через Docker з використанням безпечних офіційних образів.

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 власних рівнів у середині.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: файли формату JSON, зі структурою власних рівнів 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 бізнес процесів
– схема структурна варіантів використань
– схема архітектури програмного забезпечення
– схема сутностей бази даних
– креслення вигляду звітних 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-1230.045480.02.81

Київ – 2025

ЗМІСТ

Перелік умовних позначень

IDE	– Integrated Development Environment – інтегроване середовище розробки.
API	– Application programming interface, прикладний програмний Інтерфейс.
IT	– Інформаційні технології.
ER	– Entity-Relation diagram.
ОС	– Операційна система.
БД	– База даних.
ПКМ	– Права кнопка миші
ЛКМ	– Ліва 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.

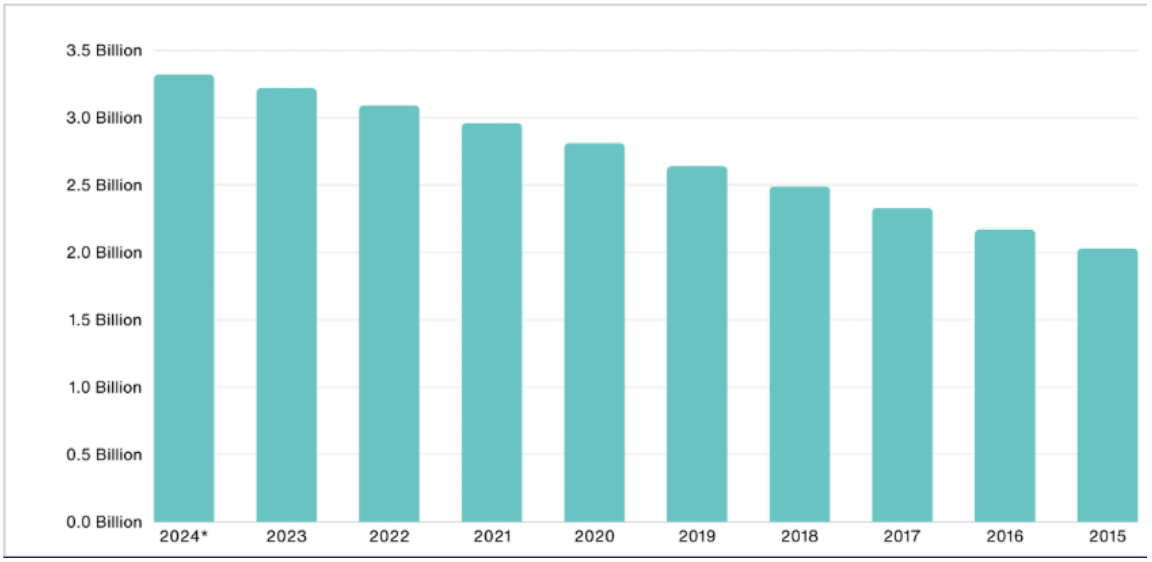


Рисунок 1.1 – Графік відношення кількості гравців до року[5]

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

Гра 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.

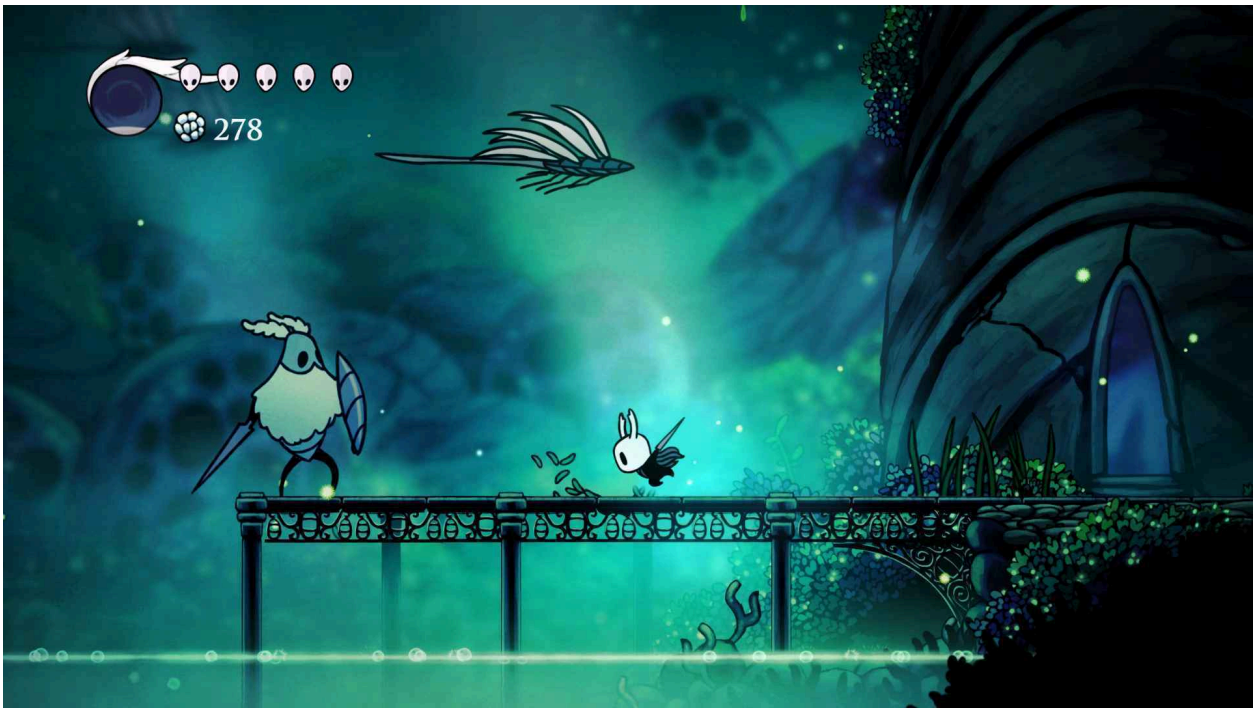


Рисунок 1.4 – Вигляд гри Hollow Knight[8]

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 користування редактором рівнів використовується BPMN модель (рисунок 1.5)

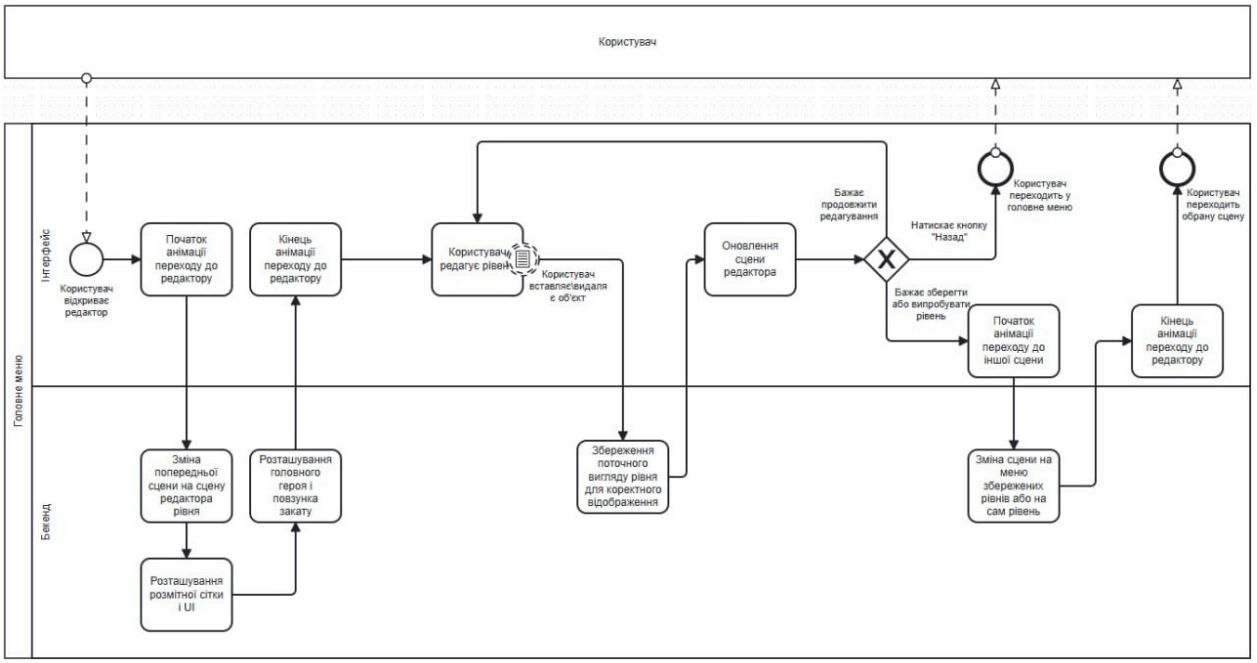


Рисунок 1.5 – Модель бізнес-процесу користування редактором рівнів

Опис моделі бізнес-процесу редагування рівня:

− користувач хоче перейти до редактора рівнів та натискає на кнопку у головному меню;

− анімація переходу розпочинається у головному меню,після чого сцена змінюється на редактор рівнів і анімація переходу закінчується ;

− користувач починає редагування рівнів та розміщує об’єкт, бекенд оновлює склад рівню і виводить назад на екран.

− користувач може обрати: натиснути на кнопку «Назад» і повернутися до головного меню; продовжити редагування; зберегти або спробувати рівень, після чого він перейде до відповідної сцени.

Для моделювання створення облікового запису користувача використовується BPMN модель (рисунок 1.6)

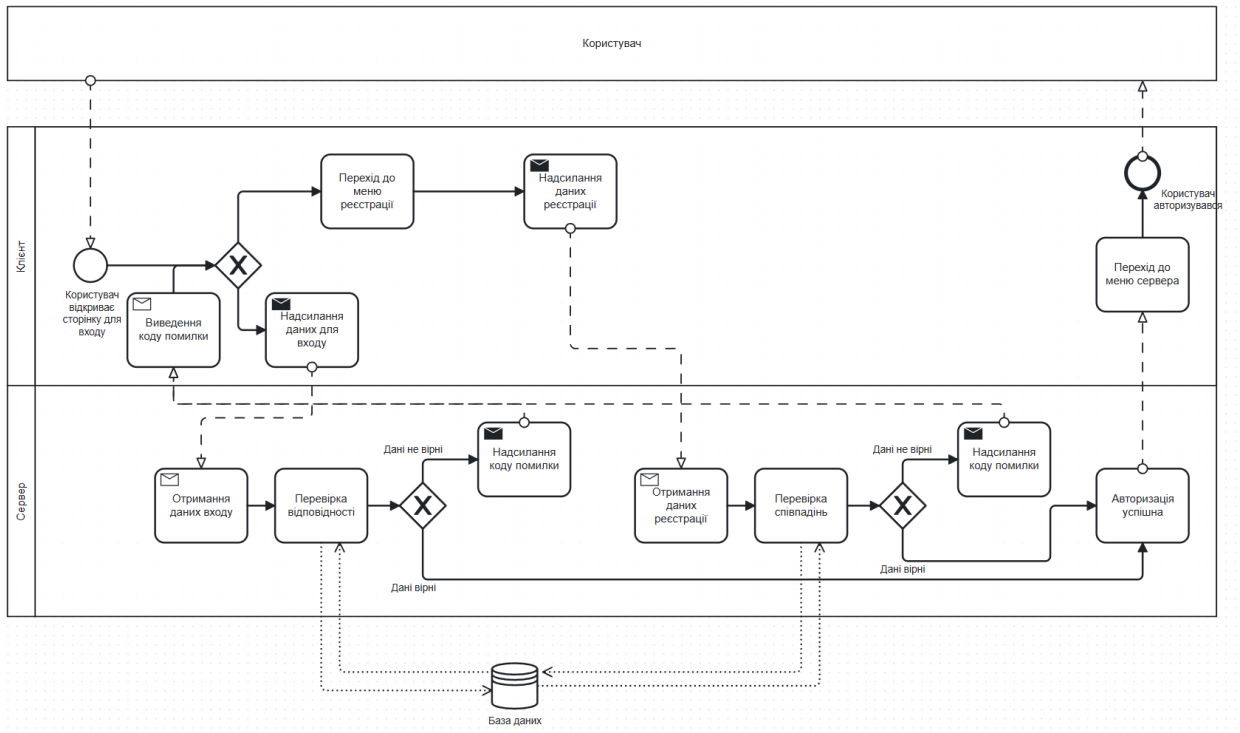


Рисунок 1.6 – Модель бізнес-процесу створення облікового запису користувача

Опис моделі бізнес-процесу створення облікового запису користувача:

– користувач відкриває сторінку для входу або реєстрації в системі.

– вводить необхідні дані (логін і пароль для входу або нові дані для реєстрації).

– дані перевіряються на клієнтській частині: якщо виявлено помилки - користувач отримує повідомлення про некоректні дані й повертається до поля введення.

– якщо дані коректні, вони надсилаються на сервер.

– Сервер отримує запит і перевіряє, чи існує користувач з такими даними:якщо це новий користувач - система створює нового користувача в базі даних;якщо дані входу неправильні - система надсилає повідомлення про помилку;якщо дані коректні - відбувається вхід у систему.

– у разі успішного входу або реєстрації користувач отримує підтвердження і переходить до головного меню сервера.

Для моделювання запиту на зміни в рівні використовується BPMN модель (рисунок 1.7)

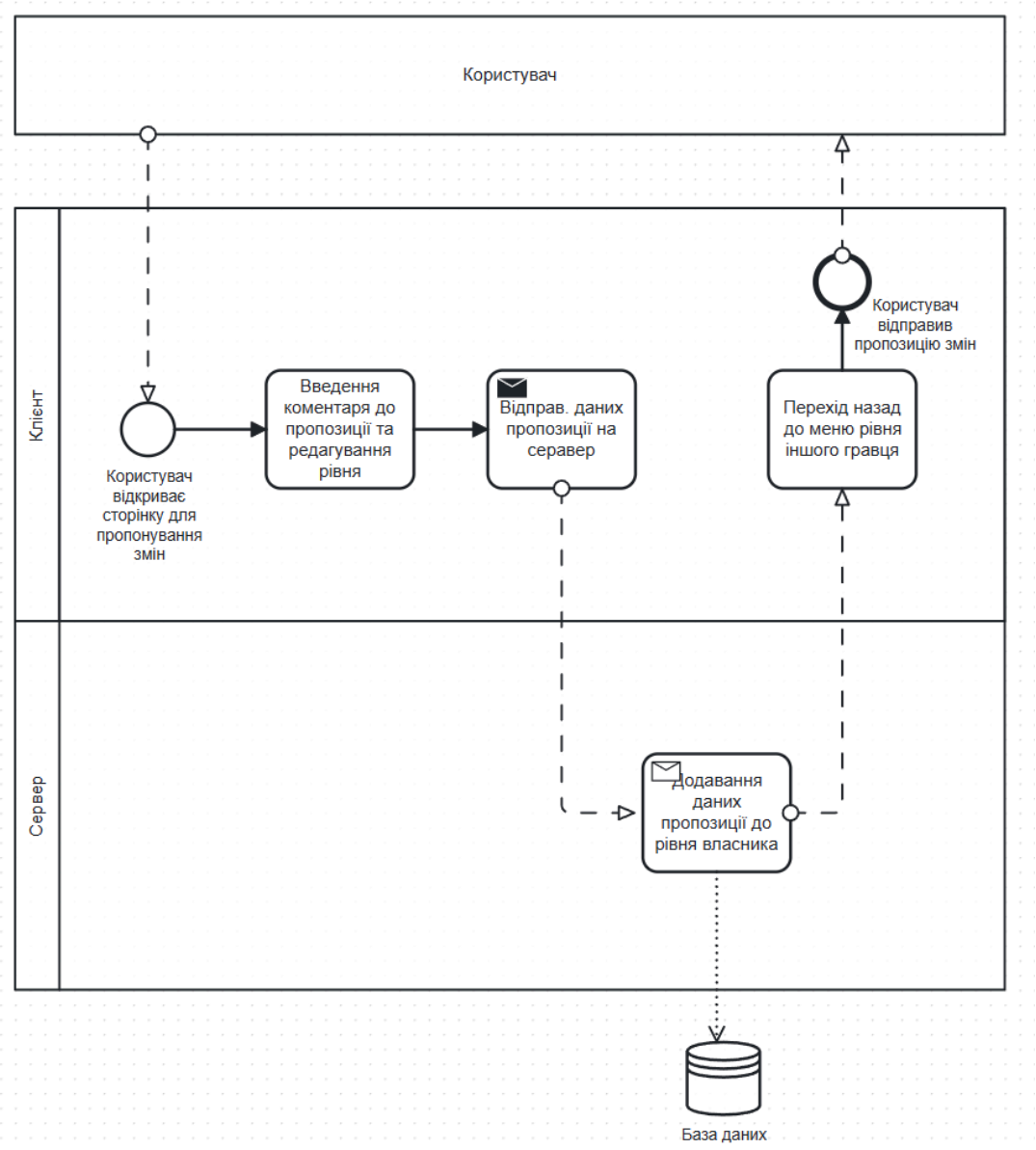


Рисунок 1.7 – Модель бізнес-процесу створення пропозиції

Опис моделі бізнес-процесу створення пропозиції:

- користувач відкриває сторінку для пропонування змін до існуючого рівня.
- вводить коментар із поясненням змін, які хоче запропонувати.
- введені дані надсилаються на сервер.
- на серверній частині відбувається обробка отриманої інформації:

– дані пропозиції додаються до рівня, що належить відповідному користувачу, і зберігаються в базі даних.

– після успішної обробки даних сервер надсилає підтвердження про завершення операції.

Для моделювання запиту на зміни в рівні використовується BPMN модель (рисунок 1.8)

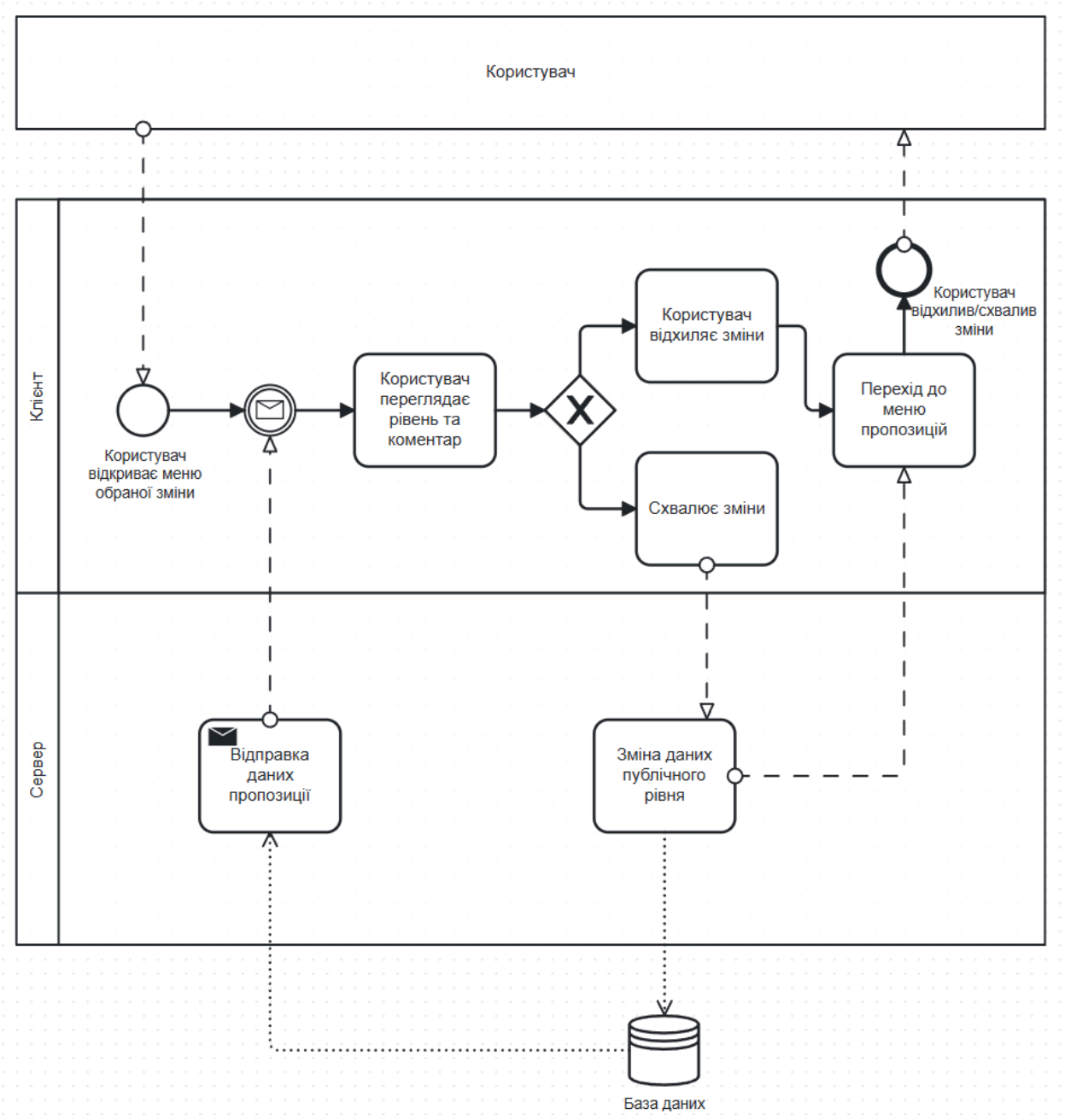


Рисунок 1.8 – Модель бізнес-процесу схвалення-відхилення пропозиції

Опис моделі бізнес-процесу схвалення-відхилення пропозиції:

- користувач відкриває сторінку з переліком пропозицій змін до свого рівня.
- він обирає одну з доступних пропозицій.
- після цього користувач приймає рішення: погодитись із запропонованими змінами або відхилити їх.
- якщо пропозиція відхиляється, сервер не вносить змін до рівня, оновлює статус пропозиції як declined, після чого користувач отримує повідомлення про відхилення змін.
- якщо користувач погоджується із запропонованими змінами, сервер перезаписує дані публічного рівня, відповідно до отриманої пропозиції, оновлює статус пропозиції як approved, і надсилає підтвердження про успішне застосування змін.

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 задачі, які необхідно вирішити в межах дипломного завдання.

Було надано визначення основним поняттям що будуть використовуватись у рамках даної роботи. Також був проведений 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.

Особливу прискіпливість було приділено саме системі обміну рівнями, який реалізовано у проекті завдяки Python-бібліотам Pygame та Django, запобігаючи використанню сторонніх великовагових рушіїв чи хмарних сервісів. Даний підхід сприяє простоті реалізації, зниженню системних технічних вимог, а також поліпшує контрольованість серверної логіки для розробника.

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 проекту.

Додатково були проаналізовані можливості застосування Python бібліотек в якості єдиного технологічного середовища для побудови ігрового клієнта і серверної частини. Це дозволило на подальших стадіях розробки проекту легко запровадити функції збереження, обміну рівнями а також аутентифікації, з використанням лише Python-інструментів без залучення більш обтяжуючих платформ.

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 показана на рисунку 2.1

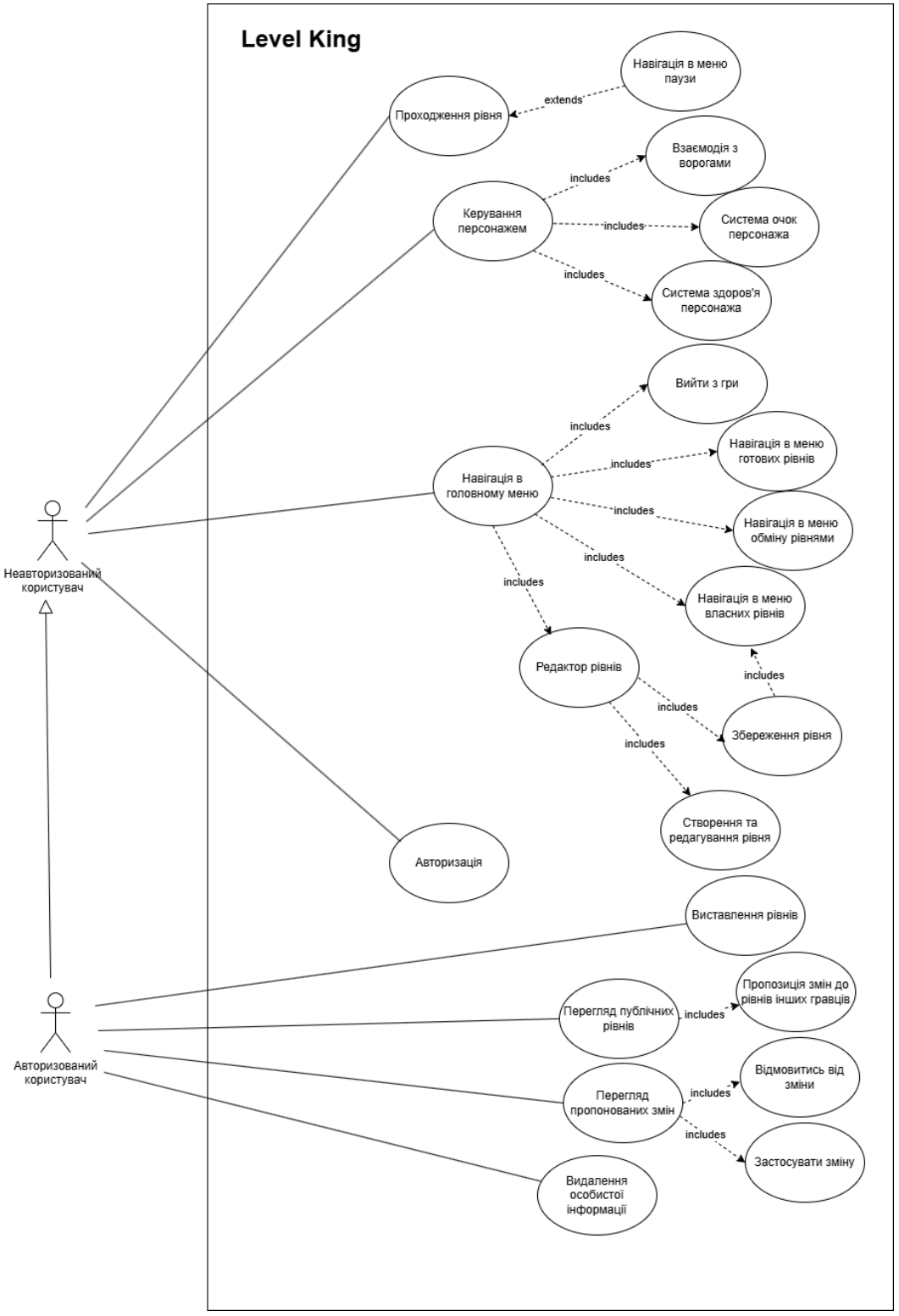


Рисунок 2.1 – Діаграма варіантів використання

В таблицях 2.1-2.10 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Редактор рівнів
Use case ID	UC-01
Goals	Дати можливість користувачу користуватись редактором рівнів
Actors	Користувач
Trigger	Користувач бажає відкрити редактор рівнів
Pre-conditions	Запущена гра.
Flow of Events	Користувач натискає на кнопку редактора у головному меню, або у меню обміну рівнями.
Extension	-
Post-Condition	Користувач відкриває редактор рівнів.

Таблиця 2.2 – Варіант використання UC-02

Use case name	Створення та редагування рівня
Use case ID	UC-02
Goals	Дати можливість користувачу створювати та редагувати рівень у редакторів рівнів.
Actors	Користувач
Trigger	Користувач бажає створити або редагувати рівень.
Pre-conditions	Запущена гра. Запущений редактор рівнів.

Продовження таблиці 2.2

Flow of Events	Користувач може виконувати наступні дії: обирати об’єкт який бажає вставити у меню об’єктів; натиснувши ЛКМ по сітці редактора, користувач вставляє обраний об’єкт; натиснувши ПКМ по комірці сітки з об’єктом, користувач видаляє об’єкт; зажавши ЛКМ на об’єкті в сітці редактора, навколо об’єкта з’являються імітовані рамки, користувач може перетягувати цей об’єкт рухом миші.
Extension	Користувач може як створювати рівень з нуля, так і обрати готовий рівень для його редагування.
Post-Condition	Користувач змінив рівень.

Таблиця 2.3 – Варіант використання UC-03

Use case name	Авторизація користувача
Use case ID	UC-03
Goals	Надати доступ до серверних функцій авторизованому користувачу.
Actors	Користувач
Trigger	Користувач бажає увійти в аккаунт.
Pre-conditions	Користувач має обліковий запис.
Flow of Events	Користувач надсилає POST-запит з логіном і паролем. Сервер повертає JWT.
Extension	У випадку неправильних даних – помилка авторизації.
Post-Condition	Користувач отримує access token для подальших запитів до API.

Таблиця 2.4 – Варіант використання UC-04

Use case name	Виставлення рівня

Продовження таблиці 2.4

Use case ID	UC-04
Goals	Зберегти створений рівень на сервері.
Actors	Авторизований користувач
Trigger	Користувач бажає виставити рівень у мережу.
Pre-conditions	Користувача авторизовано, рівень містить валідні дані.
Flow of Events	Користувач обирає рівень. Рівень перетворюється на JSON. Клієнт надсилає POST-запит.
Extension	Якщо токен недійсний - повертається помилка.
Post-Condition	Рівень збережено у базі даних, він стає доступним через REST API для інших.

Таблиця 2.5 – Варіант використання UC-05

Use case name	Перегляд публічних рівнів інших користувачів
Use case ID	UC-05
Goals	Отримати список рівнів, доступних для перегляду з серверу.
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути рівні інших гравців.
Pre-conditions	Користувач авторизований, у базі є хоча б один публічний рівень.
Flow of Events	Клієнт виконує GET-запит. Сервер повертає список JSON. Користувач може далі обирати, редагувати рівні інших.
Extension	Якщо відсутні рівні - повертається порожній список.
Post-Condition	Користувач отримує список публічних рівнів.

Таблиця 2.6 – Варіант використання UC-06

Use case name	Пропозиція змін до чужого рівня
Use case ID	UC-06
Goals	Надіслати автору рівня пропозицію змін для розгляду
Actors	Авторизований користувач
Trigger	Користувач бажає запропонувати зміни до рівня іншого гравця
Pre-conditions	Користувач авторизований, рівень викладено в публічний доступ.
Flow of Events	Клієнт надсилає POST-запит з JSON-структурою зміненого рівня.
Extension	Якщо рівень належить користувачу або поле data пошкоджено - вспливає помилка 400, якщо ж рівня з подібним id нема - вспливає помилка 404
Post-Condition	Запит на зміну збережено в базі зі статусом pending

Таблиця 2.7 – Варіант використання UC-07

Use case name	Застосувати зміну
Use case ID	UC-07
Goals	Дозволити автору оновити свій рівень відповідно до запропонованих змін.
Actors	Автор рівня
Trigger	Автор «погоджується» з запропонованими змінами.
Pre-conditions	Існує активний запит на зміну до рівня, який належить даному автору.
Flow of Events	Автор переглядає список запитів на зміну та натискає кнопку «Схвалити». Надсилається POST-запит.

Продовження таблиці 2.7

Extension	Якщо автор не є власником рівня - повертається помилка 403. Якщо ж рівень або запит не знайдено - повертається помилка 404.
Post-Condition	Зміни застосовані, статус запиту оновлено на accepted, рівень оновлено.

Таблиця 2.8 – Варіант використання UC-08

Use case name	Відхилення від зміни
Use case ID	UC-08
Goals	Дозволити автору відхилити запропоновані зміни.
Actors	Автор рівня
Trigger	Автор «відхиляє» запропоновані зміни.
Pre-conditions	Існує активний запит на зміну до рівня, який належить даному автору.
Flow of Events	Автор переглядає список запитів на зміну та натискає кнопку «Відхилити» . Надсилається POST-запит.
Extension	Якщо автор не є власником рівня - повертається помилка 403. Якщо ж рівень або запит не знайдено - повертається помилка 404.
Post-Condition	Запит змінено на статус rejected, рівень залишається без змін.

Таблиця 2.9 – Варіант використання UC-09

Use case name	Перегляд запропонованих змін
Use case ID	UC-09
Goals	Дати можливість автору рівня переглядати запропоновані зміни іншими користувачами.
Actors	Авторизований користувач.

Продовження таблиці 2.9

Trigger	Користувач бажає переглянути запропоновані зміни.
Pre-conditions	Гра запущена. Користувач авторизований. Користувач зайшов у меню пропозицій.
Flow of Events	Користувач бачить всі запропоновані зміни до свого рівня.
Extension	Користувач може спробувати пройти змінений рівень перед тим як відхилити чи прийняти пропозицію про зміну.
Post-Condition	Користувач переглянув запропоновані зміни.

Таблиця 2.10 – Варіант використання UC-10

Use case name	Видалення особистої інформації
Use case ID	UC-10
Goals	Дати можливість користувачу видалити всі дані, пов’язані з його акаунтом
Actors	Авторизований користувач.
Trigger	Користувач бажає видалити свої дані з бази.
Pre-conditions	Гра запущена. Користувач авторизований. Користувач зайшов у меню видалення акаунту.
Flow of Events	Користувач натискає ‘Delete’, меню змінюється на головне, дані користувача видаляються з бази
Extension	Дію може виконати тільки авторизований користувач
Post-Condition	Користувач видалив всю інформацію про свій акаунт з бази

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.11 наведено загальну модель вимог, а в таблиці 2.12 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити в таблиці 2.13.

Таблиця 2.11 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритет	Ризики
1	Система керування обліковими записами	FR-1	Високий	Високий
1.1	Реєстрація користувачів	FR-2	Високий	Високий
1.2	Вхід в обліковий запис користувача	FR-3	Високий	Високий
2	Редактор рівнів з підтримкою створення та редагування	FR-4	Високий	Середній
2.1	Поведінка розміщеного в залежності від типу об’єкта	FR-5	Низький	Низький
2.1.1	Правильне розміщення елементів за системою шарів	FR-6	Низький	Низький
2.2	Захист від випадкових натискань	FR-7	Середній	Низький
3	Завантаження рівнів на сервер та збереження у БД	FR-8	Високий	Середній
3.1	Редагування інформації про рівень	FR-9	Низький	Низький
4	Перегляд публічних рівнів інших користувачів	FR-10	Середній	Середній

Продовження таблиці 2.11

№	Назва	ID Вимоги	Пріоритет	Ризики
5	Механізм пропозицій змін	FR-11	Високий	Середній
5.1	Запропонувати зміну	FR-12	Високий	Середній
5.1.1	Надати коментар до зміни	FR-13	Низький	Низький
5.2	Перегляд пропонованих змін	FR-14	Високий	Середній
5.2.1	Схвалення змін автором рівня	FR-15	Високий	Середній
5.2.2	Відхилення змін автором рівня	FR-16	Високий	Середній
6	Взаємодія між клієнтом і сервером через REST API	FR-17	Високий	Низький
6.1	Перевірка прав доступу	FR-18	Високий	Високий
6.2	Перевірка авторизовності користувача	FR-19	Високий	Високий
6.3	Перевірка правильності використовуваних даних	FR-20	Високий	Середній
7	Видалення особистої інформації	FR-21	Високий	Високий

Таблиця 2.12 – Перелік функціональних вимог

Назва	Опис
FR-1	Наявність візуального інтерфейсу системи керування обліковими записами.
FR-2	Користувач повинен мати змогу зареєструвати обліковий запис.

Продовження таблиці 2.12

Назва	Опис
FR-3	Користувач повинен мати змогу увійти в обліковий запис.
FR-4	Наявність візуального інтерфейсу редактора рівнів.
FR-5	Розміщення, пересування, збереження об’єкта має відповідати його типу.
FR-6	Відображення елементів має відповідати логіці шарів (ігровий персонаж поверх інших елементів, тощо).
FR-7	При натисканні у зони, не передбачені для цього у даний час редагування, нічого не повинно змінюватись.
FR-8	Завантаження рівнів на сервер. Авторизований користувач повинен мати змогу надсилати створений рівень на сервер для збереження в базі даних.
FR-9	Користувач повинен мати змогу додати опис до викладаємого рівня
FR-10	Перегляд публічних рівнів. Користувач повинен мати можливість переглядати рівні інших користувачів, які були позначені як публічні.
FR-11	Наявність візуального інтерфейсу пропозицій змін до рівня.
FR-12	Користувач повинен мати змогу запропонувати альтернативну версію чужого рівня для подальшого розгляду автором.
FR-13	Користувач повинен мати змогу додати коментар до пропонованої зміни.
FR-14	Автор рівня повинен мати змогу переглядати запити на зміну.
FR-15	Автор рівня повинен мати змогу схвалити запит на зміну.

Продовження таблиці 2.12

Назва	Опис
FR-16	Автор рівня повинен мати змогу відхилити запит на зміну.
FR-17	Взаємодія між клієнтом і сервером повинна проходити через REST API REST API взаємодія. Вся комунікація між клієнтом та сервером повинна відбуватись через REST API із використанням формату JSON і токенів доступу.
FR-18	Всі дії повинні виконуватись тільки тими акторами, для яких призначена ця дія.
FR-19	Всі дії повинні виконуватись тільки після перевірки авторизованості по токенам доступу.
FR-20	Всі дії повинні виконуватись тільки за правильних даних.
FR-21	Користувач повинен мати змогу видалити особисту інформації при бажанні.

Таблиця 2.13 – Матриця трасування вимог

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10
FR-1			+							+
FR-2			+							
FR-3			+							
FR-4	+	+								
FR-5		+								
FR-6		+								
FR-7		+								
FR-8				+						
FR-9				+						

Продовження таблиці 2.13

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10
FR-10					+					
FR-11						+			+	
FR-12						+				
FR-13						+				
FR-14									+	
FR-15							+			
FR-16								+		
FR-17			+	+	+	+	+	+	+	+
FR-18				+	+	+	+	+	+	+
FR-19			+	+	+	+	+	+	+	+
FR-20			+	+	+	+	+	+	+	+
FR-21										+

2.3 Розроблення нефункціональних вимог

Серверна частина програмного забезпечення повинна функціонувати стабільно при одночасному надходженні кількох запитів від різних користувачів. Важливо, щоб обробка типових запитів, таких як завантаження рівня або пропонування змін, відбувалося без помітних затримок, задля того щоб користувач не відчував дискомфорту під час взаємодії з системою.

З метою забезпечення безпеки, запити до маршрутів REST API повинні супроводжуватись валідацією токенів авторизації. Сервер повинен відхиляти будь-які неавторизовані або потенційно шкідливі запити, забезпечуючи збереження персональних даних та цілісність рівнів.

Користувач не має стикатися з ситуацією, коли він не може завантажити рівень через технічні несправності. Усі важливі операції з сервером повинні

супроводжуватися чіткою обробкою помилок з відповідними повідомленнями, що дають зрозуміти користувачу суть проблеми.

Редактор рівнів повинен запускатися швидко і працювати без затримок при переміщенні додаванні чи видаленні елементів. Всі дії користувача мають відображатись у реальному часі без візуальних затримок, фризів чи приторможенні в оновленні графіки.

Графічний інтерфейс редактора рівнів повинен бути логічним та зрозумілим для будь-якого користувача, забезпечуючи інтуїтивне керування. Важливо, що елементи повинні бути впорядковані таким чином, щоб уникати зайвого навантаження на увагу користувача.

Користувач повинен мати змогу легко завантажити створений рівень у придатному форматі. Така операція має виконуватись швидко, без додаткової необхідності підтверджень або переписування налаштувань, особливо для повторного збереження.

Основна частина гри і серверна мають бути синхронізовані таким чином, щоб забезпечити єдиний користувацький досвід - без необхідності відкривати браузер чи інші сторонні сервіси.

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 2.14 – Актори(«UAW»):

Тип	Опис	Коефіцієнт	Кількість
Простий	Локальні файли рівнів (Pickle, JSON) та серверна база даних.	1	1
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 2.15 – 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
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 2.16 – 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
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 2.17 – 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
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 місяців 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

Призначенням даної частини проєкту є створення інструментів для створення, збереження та обміну рівнями між користувачами. Редактор рівнів дозволяє гравцям візуально створювати та змінювати власні ігрові середовища, тоді як серверна частина забезпечує централізоване зберігання, автентифікацію користувачів і синхронізацію даних між клієнтами.

Основною метою є розширення функціональності гри платформеру за рахунок підтримки спільної творчості гравців та можливості редагувати власні рівні.

Поставлена мета реалізується завдяки вирішенню наступних задач:

- розробка редактора рівнів, що дозволяє користувачу додавати, переміщувати та видаляти об’єкти, змінювати типи тайлів, створювати ігрові ситуації без потреби програмування.
- забезпечення інтуїтивного користувацького інтерфейсу редактора, де всі інструменти розташовані логічно.
- збереження створеного рівня у форматі, придатному для подальшого завантаження на сервер. Формат повинен бути структурованим (наприклад, словник Python/JSON).
- розробка серверної частини на базі Django та Django REST Framework, яка приймає, зберігає, повертає та оновлює рівні на запити клієнта.
- реалізація авторизації користувачів через JWT, що дозволяє обмежити доступ до ресурсів лише авторизованим користувачам.
- забезпечення механізму пропозицій змін до чужих рівнів (аналог «pull request») та можливість їх подальшого схвалення або відхилення автором.
- надання користувачу можливості переглядати публічні рівні інших користувачів, тестувати їх та пропонувати модифікації.
- забезпечення стабільної взаємодії між клієнтом і сервером через REST API, з передачею даних у форматі JSON і відповідною обробкою помилок.

Висновки до розділу

У даному розділі було сформовано повний набір функціональних, нефункціональних та системних вимог до програмного забезпечення, яке реалізує можливість створення, обміну та спільного удосконалення рівнів у грі жанру платформер.

Проведено детальний аналіз сценаріїв взаємодії користувача з системою, який викладено у формі варіантів використання (use case). Було охоплено як локальні дії в редакторі рівнів (додавання, видалення,

переміщення об’єктів), так і взаємодію з сервером (авторизація, завантаження рівнів, схвалення змін тощо). Для кожного варіанту було описано повну послідовність подій, очікувану поведінку системи, а також можливі виняткові ситуації.

Сформовано узагальнену модель функціональних вимог (FR-1 – FR-20), кожна з яких описує глобальну частину функціональності: від керування обліковими записами до повної реалізації REST API-взаємодії між клієнтом та сервером. Матриця трасування вимог дозволяє встановити зв'язок між сценаріями використання та відповідними функціональними вимогами, що підтверджує повноту покриття системної логіки.

Було сформульовано нефункціональні вимоги до системи, зосереджені на продуктивності, безпеці, зручності використання та стабільності роботи як редактора рівнів, так і серверної частини. Вказано вимоги до обробки помилок, валідації даних, швидкодії, а також вимоги до архітектурної синхронізації між клієнтом і сервером.

Окрім цього, визначено системні вимоги до технічного середовища, у якому передбачено експлуатацію програмного забезпечення, включно з мінімальними та рекомендованими конфігураціями пристроїв.

Таким чином, розділ забезпечив чітке формування вимог до майбутньої реалізації проєкту, що дозволяє перейти до етапу проєктування архітектури програмного забезпечення з чітким розумінням очікуваної функціональності.

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.

Діаграма архітектури зображена у графічному матеріалі, креслення 1.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

У ході розробки проєкту було прийнято низку архітектурних рішень, що забезпечують ефективну реалізацію функціональності, стабільність системи, зручність масштабування, та відповідність поставленим задачам.

3.2.1 Управління користувачами

Система реалізує авторизацію користувачів на стороні сервера, без використання сторонніх сервісів типу Single Sign-On. Кожен користувач створює обліковий запис безпосередньо у системі, що спрощує реалізацію та забезпечує повний контроль над механізмом автентифікації.

Для автентифікації обрано технологію JWT (JSON Web Token). При успішному вході користувач отримує пару токенів - access token і refresh token, які зберігаються на боці клієнта. Усі запити до API відправляються з access token у заголовку Authorization. Це дозволяє ефективно і безпечно контролювати доступ до ресурсів на сервері.

Переваги JWT:

- не потребує сесій на сервері (масштабованість);
- зручна інтеграція з мобільними чи десктопними клієнтами;
- проста реалізація у Django через бібліотеку djangorestframework-simplejwt.

3.2.2 Інтеграція з зовнішніми системами

Система наразі не інтегрується з жодними сторонніми сервісами, що забезпечує повну автономність. Однак вона легко може бути розширена, наприклад, через інтеграцію з e-mail сервером для повідомлень про схвалення/відхилення змін у рівнях.

Протокол взаємодії між клієнтом і сервером - HTTP із використанням REST API. Передача даних здійснюється у форматі JSON, що забезпечує легкість інтеграції та універсальність обробки.

3.2.3 Тип бази даних

Для збереження даних використано реляційну базу даних (тип - таблична модель). Основна причина вибору - чітка структура даних (користувачі, рівн), наявність зв'язків між сутностями та підтримка транзакцій.

Для зберігання вмісту рівнів застосовано поле типу JSONField, яке дозволяє поєднати переваги реляційної структури (ідентифікація власника, фільтрація, контроль прав) з гнучкістю JSON для внутрішнього представлення складної структури рівня.

3.2.4 Нефункціональні вимоги та їх реалізація

Безпека: всі дії, які змінюють або переглядають персональні дані, потребують автентифікації (JWT). Права доступу контролюються через permissions у DRF.

- Масштабованість: завдяки REST API клієнтів може бути скільки завгодно, без впливу на інтерфейс чи структуру бази.
- Збереження даних: дані зберігаються у PostgreSQL.
- Надійність: серверна частина легко розгортається на будь-якому сервері через Docker, що знижує ризики залежностей і проблем з оточенням.

3.2.5 Технологічний стек

Таблиця 3.1 – Технологічний стек:

Компонент	Інструмент	Обґрунтування
Клієнт	Python + Pygame	Зручна бібліотека для 2D-ігор, підтримка кастомного UI, зручна робота з іншими бібліотечними Python ресурсами
Сервер	Django + Django REST Framework	Швидка розробка REST API, інтегрована автентифікація, ORM

Продовження таблиці 3.1

Компонент	Інструмент	Обґрунтування
Авторизація	JWT (SimpleJWT)	JWT (SimpleJWT) безпечна, масштабована, не потребує серверних сесій
База даних	PostgreSQL	Стабільна реляційна БД з підтримкою JSON
Передача даних	REST API + JSON	Сумісність, прозорість, простота інтеграції
Розгортання	Docker	Однакове середовище для тестування і продакшну
IDE	PyCharm	Зручний для Python-розробки, підтримка підсвічування, автодоповнення, інтеграція з Git

3.2.6 Порівняльний аналіз альтернативних засобів

Таблиця 3.2 – Flask vs Django:

Flask	Django
Простіший, але потребує ручної реалізації багатьох функцій (авторизація, адміністрування).	Забезпечує «повний стек» із коробки: ORM, адмін-панель, маршрутизація, підтримка REST через DRF.

Таблиця 3.3 – SQLite vs PostgreSQL:

SQLite	PostgreSQL
Простий, підходить для локальної розробки, але не підтримує паралелізм і великі обсяги.	Краще масштабований, підтримує JSONField, надійний для продакшн-середовища.

3.3 Конструювання програмного забезпечення

3.3.1 Алгоритм підстановки правильних спрайтів для блоків землі

Для наочного прикладу принципу роботи цього алгоритму, спочатку ми встановимо перший блок землі. Задля більшого розуміння у подальшому, довільно привласнимо йому координати відносно початкової точки рівня, наприклад (3;3).

Рисунок 3.2 – Перший блок землі(3;3)

Якщо користувач захоче встановити ще один блок, котрий буде знаходитися на одну клітинку вище нашого початкового блоку (3;3), візуально правильно буде зробити наш початковий блок (3;3) нижнім відносно нового блоку. Тобто прибрати з верху початкового блоку рослинність і додати її на новий блок. Можемо побачити макет бажаного результату на рисунку 2.5 .

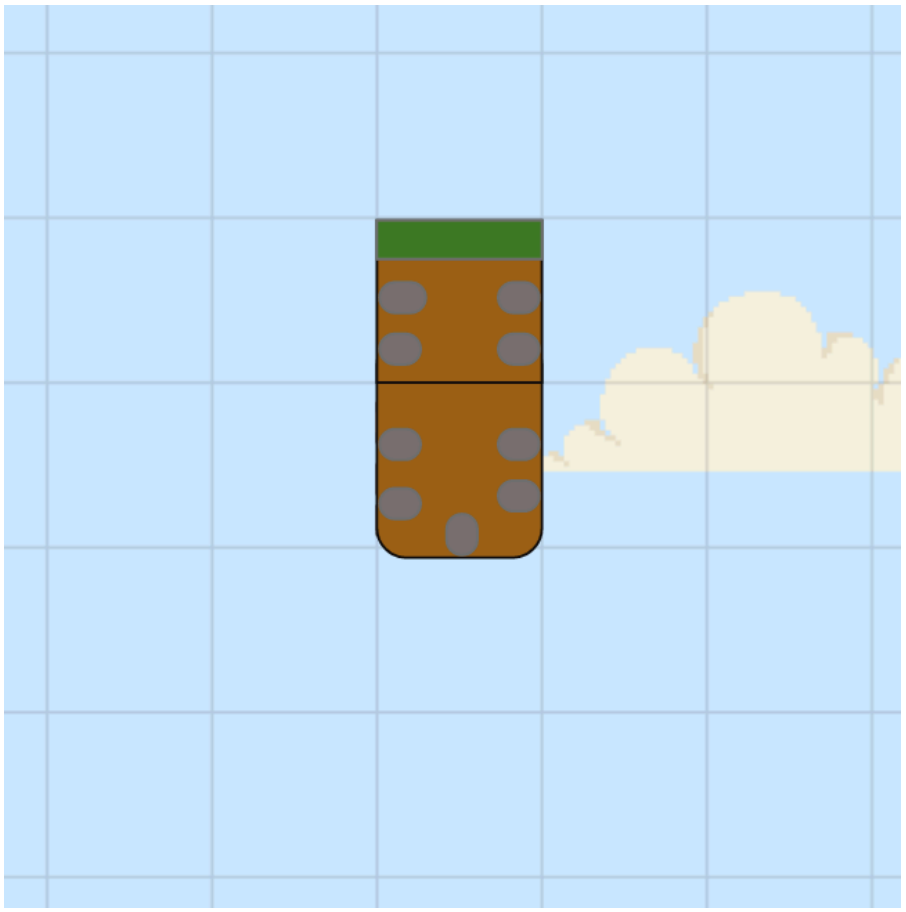


Рисунок 3.3 – Макет бажаного результату

Для отримання подібного результату, при встановленні кожного нового блоку землі, нам потрібно перевіряти наявність та розташування сусідніх блоків землі і в залежності від цього, привласнювати новим блокам їх спрайти, та оновлювати спрайти сусідніх блоків. На рисунку 2.6, стрілочками показано які саме блоки алгоритм повинен перевірити для присвоєння коректного спрайту блоку, котрий ми хочемо встановити на «Бажане місце».

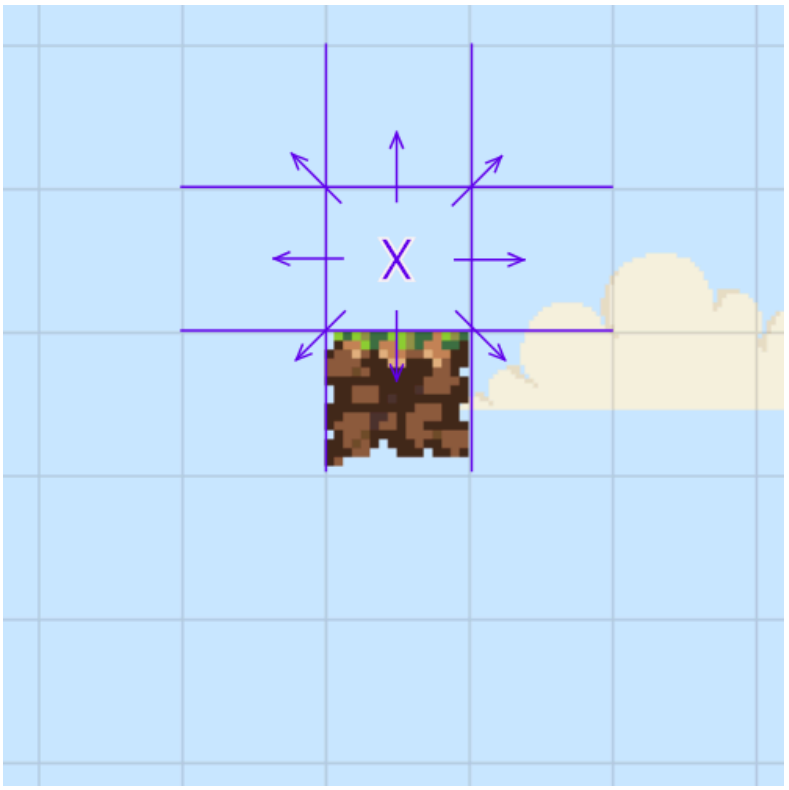


Рисунок 3.4 – Перевірка сусідів відносно «Бажаного місця»

Але також, не потрібно забувати що нам потрібно оновити спрайти у вже створених блоків, наприклад нашого початкового блоку (3;3). Для цього, нам також потрібно перевірити сусідів усіх сусідніх клітинок, відносно «Бажаного місця». Розберемо на прикладі початкового блоку (3;3).

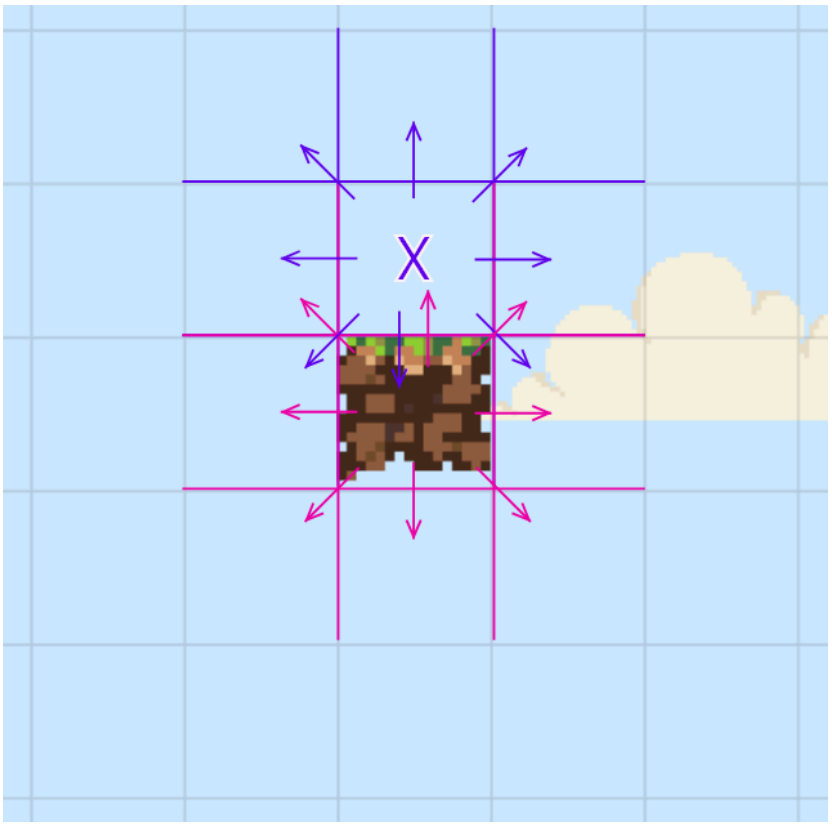


Рисунок 3.5 – Перевірка сусідів початкового блоку (3;3)

Для того щоб алгоритм розумів який саме спрайт підставляти, була розроблена наступна система:

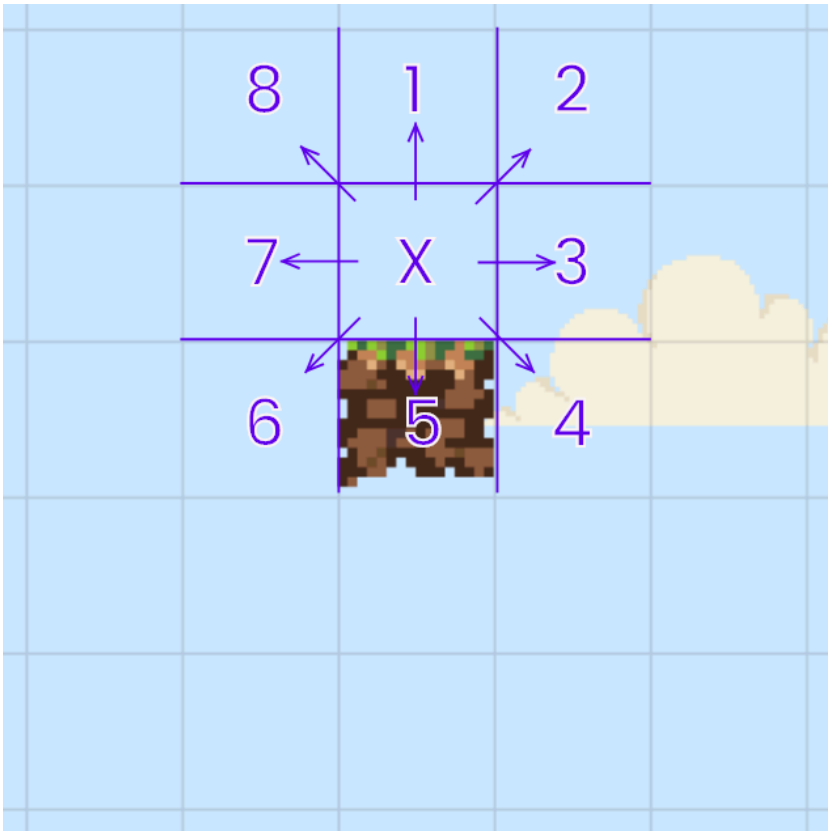


Рисунок 3.6 – Система підставляння спрайтів

Рисунок 3.7 – Вміст файлу зі спрайтами

Кожній можливій позиції сусіда, було надано своє власне ім’я у вигляді латинської літери. Починаючі з позиції «1» , алгоритм перевіряє наявність кожного сусіда по годинниковій стрілці. У випадку, якщо сусід наявний, алгоритм додає назву позиції до імені бажаного спрайту. У нашому випадку, наявний тільки сусід «5», тож назва спрайту котрий потрібно встановити на бажане місце – «5.png». Але також, алгоритму потрібно оновити спрайт першого блоку (3;3). Для цього, він проходить по сусідам цього блоку. Так як у нього наявний тільки верхній сусід «1», назва спрайту для початкового блоку буде «1.png».

Рисунок 3.8 – «5.png»

Рисунок 3.9 –«1.png»

На практиці, ми отримуємо наступний результат:

Рисунок 3.10 – Результат роботи алгоритму

По принципу цього прикладу, алгоритм працює з кожним спрайтом кожної позиції сусідів та видає бажані результати.

3.3.2 Опис структури бази даних

У якості системи управління базами даних (СУБД) для серверної частини програмного забезпечення обрано PostgreSQL. PostgreSQL - це потужна об’єктно-реляційна СУБД з відкритим кодом, яка підтримує широкий спектр типів даних, включаючи структуровані, неструктуровані (JSON/JSONB), а також реляційні зв’язки між сутностями.

СУБД виконує всі необхідні функції для збереження, обробки та захисту даних: забезпечує зручний механізм доступу до даних, обробляє

SQL-запити, підтримує транзакції, цілісність і узгодженість даних, контролює права доступу до об’єктів бази.

База даних, що обслуговується цією СУБД, являє собою впорядковану сукупність структурованих даних, організованих у вигляді взаємопов'язаних таблиць. У межах проєкту ця база використовується для зберігання всіх ключових об'єктів, необхідних для роботи платформи:

− Інформація про зареєстрованих користувачів - включає унікальні ідентифікатори, облікові дані, дати реєстрації та ознаки ролі (звичайний користувач або адміністратор).

− Ігрові рівні, і апити на зміну рівнів. Кожен рівень містить мета-інформацію (назву, власника, дату створення) та основний вміст, який зберігається у форматі JSONB - вкладена структура з елементами і координатами, отримана з редактора рівнів. Запити ж на зміну рівнів - дозволяють одному користувачу запропонувати зміни до рівня, який належить іншому. Автор рівня може перевірити запропоновану зміну та схвалити або відхилити її. Кожен запит має статус (pending, accepted, rejected), а також зберігає альтернативну версію рівня у JSON-форматі.

Таким чином, СУБД PostgreSQL забезпечує всю логіку збереження, доступу та модифікації цих даних, тоді як сама база даних містить таблиці з фактичними записами, які використовуються під час взаємодії клієнта з сервером.

ER-діаграма сутностей User, Level, LevelChangeRequest зображена у графічному матеріалі, креслення 2.

Таблиця 3.4 - Опис таблиці user

Назва поля	Тип даних	Опис
id	serial	Ідентифікатор користувача
username	varchar	Ім’я користувача
email	varchar	Електронна пошта користувача

Продовження таблиці 3.4

Назва поля	Тип даних	Опис
password	varchar	Хеш пароля користувача
is_active	boolean	Чи активний обліковий запис
is_staff	boolean	Ознака адміністратора
date_joined	timestamp	Дата створення облікового запису

Таблиця 3.5 - Опис таблиці level

Назва поля	Тип даних	Опис
level_id	serial	Ідентифікатор рівня
title	varchar	Назва рівня
description	varchar	Опис рівня
data	jsonb	Структура рівня, створена у редакторі гри
owner_id	int (FK)	Посилання на користувача (автора рівня)
is_public	boolean	Ознака публічності рівня
created_at	timestamp	Дата створення

Таблиця 3.6 - Опис таблиці level_change_request

Назва поля	Тип даних	Опис
id	serial	Ідентифікатор запиту
level_id	int (FK)	Рівень, до якого запропоновано зміну

Продовження таблиці 3.6

Назва поля	Тип даних	Опис
proposer_id	int (FK)	Користувач, що пропонує зміну
new_data	jsonb	Нова версія рівня, що пропонується
status	varchar	Статус запиту: pending, accepted, rejected
comment	text	Коментар до запиту
created_at	timestamp	Дата створення запиту

Таблиця 3.7 - Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	PyCharm	Основне середовище розробки гри та серверної частини (з підтримкою Django)
2	Postman	Тестування REST API-запитів, перевірка авторизації, перевірка CRUD-операцій
3	pgAdmin / DBeaver	Графічний інтерфейс для перегляду таблиць бази даних PostgreSQL
4	Docker	Створення ізольованого середовища для серверу з базою даних
5	GitHub	Зберігання коду, співпраця, контроль версій

Таблиця 3.8 – Опис бібліотек

№ п/п	Назва бібліотек	Опис застосування
1	pygame	Бібліотека для створення ігор та графічних додатків на мові програмування Python. Вона надає інструменти для роботи з графікою, звуком, введенням користувача та іншими аспектами гри.
2	pygame.math: Vector2	Модуль pygame.math містить реалізацію класу Vector2, який представляє математичний вектор у двовимірному просторі. Використовувався для виконання операцій з векторами, такими як додавання, віднімання, множення на скаляр

3.4 Аналіз безпеки даних

Програмне забезпечення, що розробляється, містить серверну частину з централізованим зберіганням даних користувачів, їхніх рівнів та взаємодій між ними. У зв’язку з цим питання безпеки даних є критично важливими на всіх етапах розробки та впровадження системи.

3.4.1 Аналіз вразливостей (Vulnerability Management)

Передача облікових даних здійснюється через авторизаційний механізм JWT, що запобігає передачі паролів у відкритому вигляді.

Хешування паролів реалізовано з використанням алгоритму, вбудованого у Django (PBKDF2 з сіллю), що унеможливлює пряме зчитування паролів із бази.

Вразливості на рівні доступу (наприклад, зміна чужого рівня) усуваються через систему permissions у Django REST Framework, де кожна дія перевіряється на відповідність правам користувача.

Масштабованість перевірок дозволяє впровадити сканери вразливостей типу Bandit або інтегрувати CI-аналіз безпеки через GitHub Actions у майбутньому.

3.4.2 Тестування безпеки (Security Testing)

Ручне тестування проводилося для перевірки авторизації, доступу до рівнів інших користувачів, захисту від несанкціонованих запитів (без токена).

Типові загрози, такі як SQL-ін’єкції, XSS та CSRF, були враховані:

SQL-ін’єкції унеможливлюються завдяки використанню ORM;

CSRF автоматично обробляється у Django для POST-запитів із браузера (хоча у даному випадку API переважно використовується з гри, а не з браузера).

3.4.3 Безпека у середовищі контейнерів і хмари (Security for Container and Cloud)

Контейнеризація через Docker дозволяє ізолювати середовище виконання, зменшуючи ризики, пов’язані з залежностями.

Образи, створені для розгортання серверу, базуються на офіційних мінімалістичних образах (python:slim), що знижує кількість вразливих компонентів.

У майбутньому можливе розгортання в хмарному середовищі (наприклад, Heroku або Railway) з підтримкою шифрування даних у транспортному каналі (HTTPS) та з резервним копіюванням.

3.4.4 Висновки

Отже, програмне забезпечення враховує ключові аспекти захисту даних користувачів та ігрового контенту. Використання сучасних інструментів (ORM, JWT, Docker) забезпечує базовий рівень безпеки навіть без складної інфраструктури. У разі подальшої розробки доцільно реалізувати автоматизоване сканування вразливостей, журналювання дій користувачів та шифрування рівнів.

Висновки до розділу

У цьому розділі було розглянуто основні аспекти конструювання та розроблення програмного забезпечення, включаючи архітектуру системи, вибір технологічного стеку, структуру бази даних, алгоритми взаємодії між компонентами та механізми захисту даних.

Програмне забезпечення побудоване за клієнт-серверною архітектурою, що забезпечує масштабованість, централізоване управління ігровими рівнями та зручну багатокористувацьку взаємодію. Сервер реалізовано за принципами патерну MVC із використанням фреймворку Django та Django REST Framework, що дозволило чітко розділити логіку, представлення і структуру даних. Авторизація реалізована за допомогою JWT, що забезпечує надійний захист даних при передачі через REST API.

В якості СУБД обрано PostgreSQL, яка забезпечує реляційну модель даних з підтримкою гнучких типів, таких як JSONB. У базі зберігаються користувачі, рівні, та запити на зміну рівнів з можливістю схвалення змін.

Ретельно проаналізовано та впроваджено основні механізми захисту інформації, включаючи контроль прав доступу, хешування паролів, захист від SQL-ін’єкцій, а також контейнеризацію серверного оточення для ізоляції та

безпеки. Враховано можливості подальшого масштабування та інтеграції з хмарними середовищами.

Таким чином, розроблене рішення поєднує в собі надійну архітектурну основу, сучасні технології, безпечну обробку користувацьких даних та гнучкість до подальшого розширення функціоналу.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Метою тестування є:

− перевірка коректного функціонування програмного забезпечення відповідно до заданих функціональних вимог;

− оцінювання сумісності застосунку з різними операційними системами, зокрема Windows та Linux;

− виявлення можливих помилок, недоліків або вразливостей у коді з метою їх подальшого усунення;

− аналіз зручності користувацького інтерфейсу та перевірка його стабільної роботи.

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].

Результати тестування подано на рисунку 4.1.

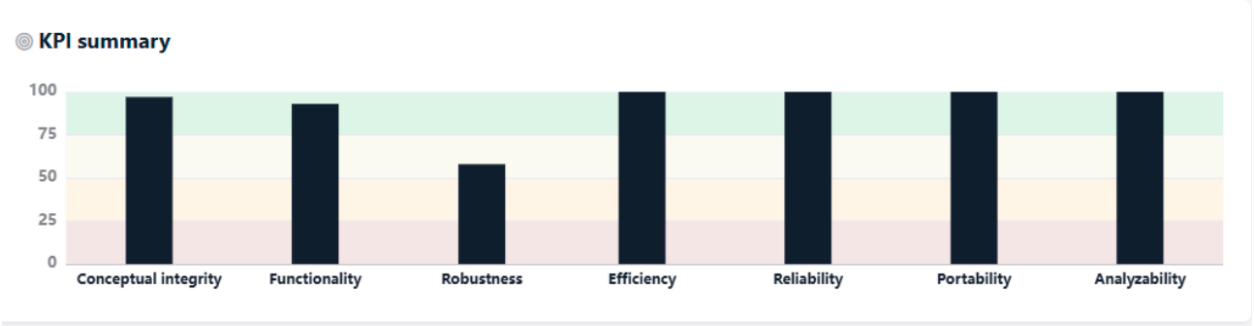


Рисунок 4.1 – Результати обчислень метрик

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 Реєстрація користувача

Початковий стан системи	Користувач знаходиться на меню реєстрації
Вхідні дані	Логін, пароль
Опис проведення тесту	У відповідні поля вводяться: логін, пароль який має мінімум 8 символів, не є схожим на ім’я користувача, не є поширеним (типу password, 12345678), не є лише з цифр. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і переходить до меню сервера.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Вхід в акаунт користувача

Початковий стан системи	Користувач знаходиться на меню входу, користувач має зареєстрований акаунт
Вхідні дані	Логін, пароль
Опис проведення тесту	У відповідні поля вводяться: логін, пароль. Після цього натискається кнопка входу.

Продовження таблиці 4.1

Очікуваний результат	Вхід проходить успішно, користувач переходить до меню сервера.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Перегляд власних локальних рівнів

Початковий стан системи	Користувач знаходиться на меню серверу, користувач увійшов в акаунт. Користувач має збережені локальні рівні.
Вхідні дані	-
Опис проведення тесту	Користувач бачить список власних локальних рівнів. При натисканні на них відкривається меню локального рівня з інформацією про нього, кнопками для публікації та редагування.
Очікуваний результат	Збережені рівні видно в списку. Відображається коректна інформація про рівень, наявні опції працюють справно.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Публікація власного рівня

Початковий стан системи	Користувач має збережений локальний рівень. Користувач знаходиться на меню обраного локального рівня, користувач увійшов в акаунт.
Вхідні дані	Назва, опис
Опис проведення тесту	Користувач вказує назву рівня та його опис, натискає «Опублікувати»
Очікуваний результат	Обраний рівень додається до списку публічних, з відповідними назвою та описом.

Продовження таблиці 4.4

Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Перегляд власних публічних рівнів

Початковий стан системи	Користувач знаходиться на меню серверу, користувач увійшов в акаунт. Користувач має викладені публічні рівні.
Вхідні дані	-
Опис проведення тесту	Користувач бачить список власних публічних рівнів. При натисканні на них відкривається меню публічного рівня з інформацією про нього, кнопками для збереження та редагування.
Очікуваний результат	Виставлені публічні рівні видно в списку. Відображається коректна інформація про рівень, наявні опції працюють справно.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Перегляд публічних рівнів

Початковий стан системи	Користувач знаходиться на меню публічних рівнів, користувач увійшов в акаунт. Наявні публічні рівні, виставлені іншими користувачами.
Вхідні дані	-
Опис проведення тесту	Користувач бачить список публічних рівнів. При натисканні на них відкривається меню публічного рівня з інформацією про нього, кнопками для гри та пропонування змін.
Очікуваний результат	Публічні рівні видно в списку. Відображається коректна інформація про рівень, наявні опції працюють справно.

Продовження таблиці 4.6

Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Запропонувати зміну для публічного рівня

Початковий стан системи	Користувач знаходиться на меню обраного публічного рівня, користувач увійшов в акаунт.
Вхідні дані	Опис пропозиції
Опис проведення тесту	Користувач натискає на кнопку «Запропонувати», переходить до меню пропонування, де може ввести пропозицію, редагувати рівень через редактор та відправити запит на зміну.
Очікуваний результат	Перехід, редагування рівня та додавання тексту пропозиції працює справно.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Перегляд наявних пропозицій до власних рівнів

Початковий стан системи	Користувач знаходиться на меню пропозицій, користувач увійшов в акаунт, наявні пропозиції змін до публічних рівнів користувача.
Вхідні дані	-
Опис проведення тесту	Користувач бачить пропозиції до власних публічних рівнів. Бачить інформацію про рівень, до якого зроблено пропозицію. Натискає на обрану пропозицію і переходить в її меню, де може спробувати її, погодитися з нею або відмовитись.
Очікуваний результат	Пропозиції відображаються коректно. Переходи, опції в меню пропозиції працюють справно.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Погодження пропозиції

Початковий стан системи	Користувач знаходиться на меню обраної пропозиції, користувач увійшов в акаунт.
Вхідні дані	-
Опис проведення тесту	Користувач натискає кнопку «Застосувати» в меню обраної пропозиції.
Очікуваний результат	Перехід на меню сервера, відповідні зміни внесено до власного публічного рівня.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Відхилення пропозиції

Початковий стан системи	Користувач знаходиться на меню обраної пропозиції, користувач увійшов в акаунт.
Вхідні дані	-
Опис проведення тесту	Користувач натискає кнопку «Відмовити» в меню обраної пропозиції.
Очікуваний результат	Перехід на меню сервера, відповідні зміни не було застосовано до власного публічного рівня.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Видалення даних про акаунт

Початковий стан системи	Користувач знаходиться на меню видалення акаунту, користувач увійшов в акаунт
Вхідні дані	-
Опис проведення тесту	Користувач читає дисклеймер про видалення даних, натискає кнопку «Видалити» в меню видалення акаунту.

Продовження таблиці 4.9

Очікуваний результат	Перехід на головне меню. Всі дані користувача успішно видалено.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.10 Додавання бажаних елементів до рівня

Початковий стан системи	Користувач знаходиться в редакторі рівнів
Вхідні дані	-
Опис проведення тесту	Користувач обирає бажаний елемент (через ЛКМ) в блоці елементів, його стиль (через колесико миші) і розміщає його на полотні, натискаючи ЛКМ.
Очікуваний результат	Елемент додається на полотно. Для відповідних елементів програються анімації, правильно відображаються шари. Відповідні елементи прикріплюються до сітки полотна (крім об’єктів)
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.11 Видалення небажаних елементів з рівня

Початковий стан системи	Користувач знаходиться в редакторі рівнів. Користувач розмістив певні елементи на полотні.
Вхідні дані	-
Опис проведення тесту	Користувач обирає бажаний елемент на полотні і натискає на ньому ПКМ.
Очікуваний результат	Елемент видаляється з полотна.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.12 Перетягнути об’єкт в редакторі рівня

Початковий стан системи	Користувач знаходиться в редакторі рівнів. Користувач розмістив певні об’єкти на полотні.
Вхідні дані	-
Опис проведення тесту	Користувач обирає бажаний об’єкт (в тому числі ігрового персонажа або лінію горизонту) на полотні затискає на ньому ПКМ і перетягує на бажане місце.
Очікуваний результат	Навколо об’єкта з’являються рамки, об’єкт перетягнуто на бажане місце.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.13 Рух сітки редактора

Початковий стан системи	Користувач знаходиться в редакторі рівнів. Користувач розмістив певні елементи на полотні.
Вхідні дані	-
Опис проведення тесту	Користувач затискає ПКМ на вільному місці сітки редактора та перетягує мишу. Або ж прокручує колесико миші з клавішею Ctrl чи без неї
Очікуваний результат	Всі елементи рухаються разом з сіткою.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

У якості контрольного прикладу було обрано сценарій публікації рівня одним користувачем та подальше подання пропозиції змін до цього рівня іншим користувачем.

Першим етапом є реєстрація нового користувача (рис. 4.2). У меню реєстрації вводиться логін та надійний пароль, після чого натискається

кнопка підтвердження. Після успішної реєстрації користувач автоматично переходить у меню сервера.

Рисунок 4.2 – Меню реєстрації

Після цього користувач переглядає список своїх локальних рівнів (рис. 4.3), обирає один з них (рис. 4.4), заповнює назву та опис, натискає «Опублікувати». Рівень з’являється у загальному списку публічних рівнів.

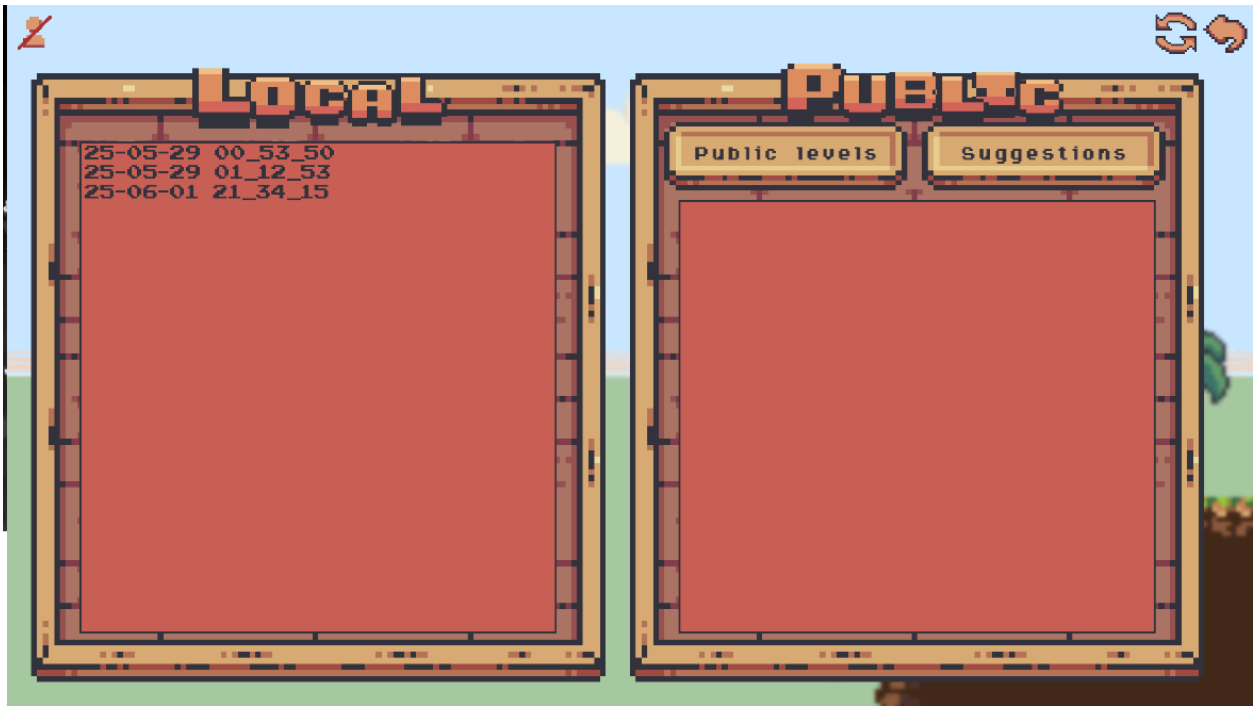


Рисунок 4.3 – Список локальних рівнів

Рисунок 4.4 – Меню локального рівня

Наступним кроком є вхід до системи іншим користувачем (рис. 4.5), який переглядає доступні публічні рівні (рис. 4.6). У списку видно щойно опублікований рівень іншого користувача, що підтверджує його доступність у публічному просторі.

Рисунок 4.5 – Вхід під іншим користувачем

Рисунок 4.6 – Меню публічних рівнів

Інший користувач обирає цей рівень, натискає кнопку «Запропонувати зміни» та переходить до відповідного меню. Тут він редагує рівень за допомогою вбудованого редактора (наприклад, додає нові об’єкти) (рис. 4.7) та вводить опис змін, після чого натискає кнопку підтвердження (рис. 4.8).

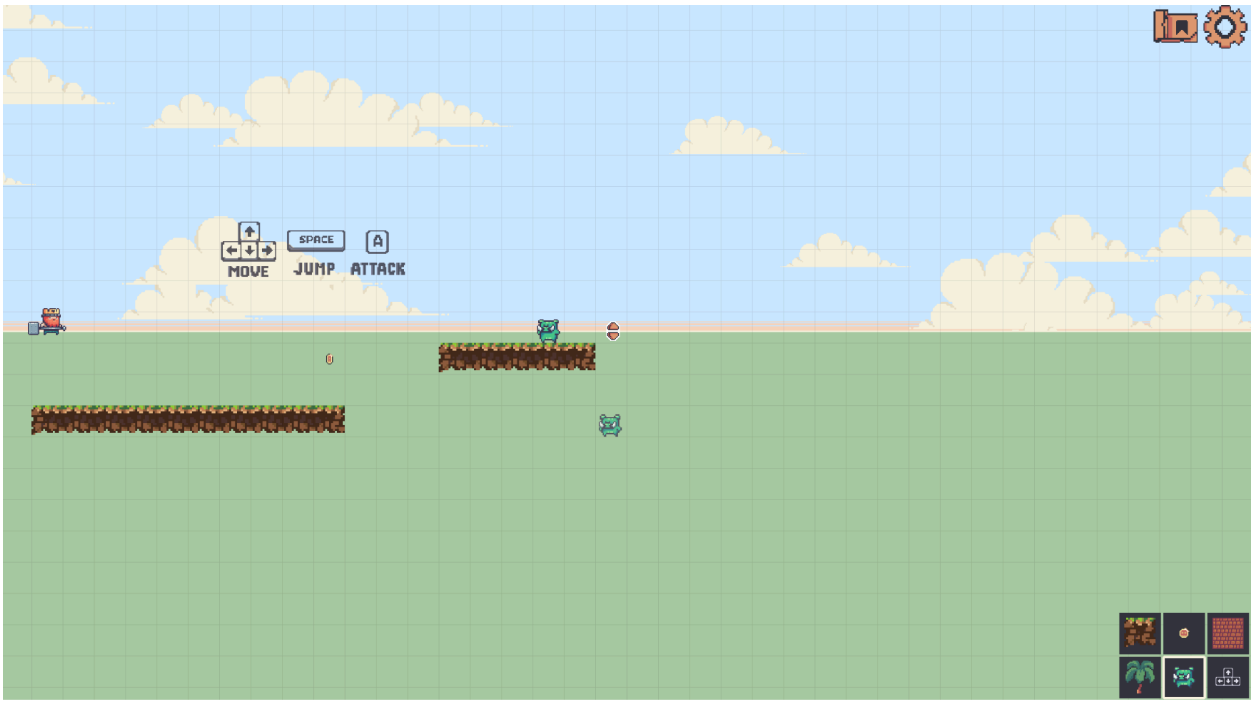


Рисунок 4.7 – Редактор рівня

Рисунок 4.8 – Вікно пропонування змін

Після цього власник бачить нову пропозицію (рис. 4.9). Він заходить у меню пропозицій, переглядає запропоновану версію рівня, тестує її та приймає зміни, натискаючи кнопку «Застосувати» (рис. 4.10).

Рисунок 4.9 – Меню пропозицій

Рисунок 4.10 – Меню пропозиції

У результаті, зміни збережені у публічній версії рівня, що підтверджується повторним переглядом рівня у публічному списку (рис. 4.11). Усі дії відбулись згідно очікуваного сценарію.

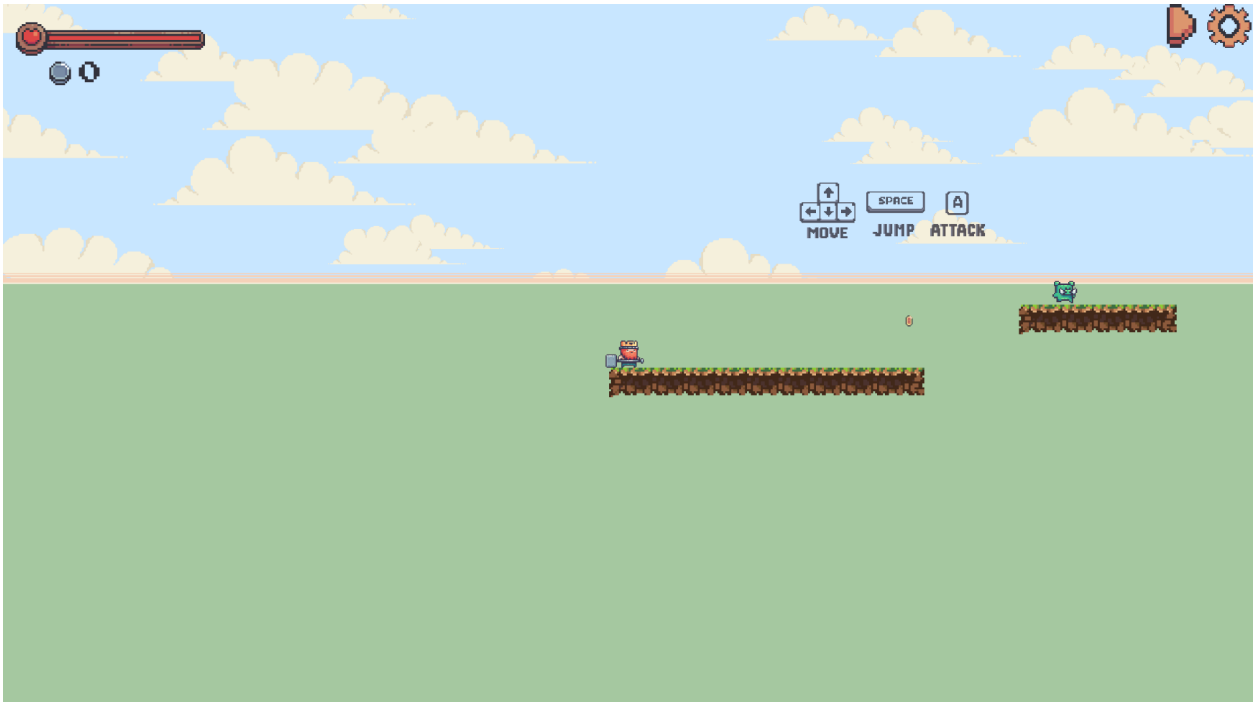


Рисунок 4.11 – Змінений рівень

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: реєстрація користувача, вхід до акаунту, перегляд локальних рівнів, публікація рівнів, перегляд власних і сторонніх публічних рівнів, подання пропозицій змін, перегляд пропозицій, погодження та відхилення пропозицій, видалення акаунту, додавання елементів у редакторі, їх видалення, перетягування об’єктів, рух сітки редактора.

Також був описаний контрольний приклад виставлення рівня в публічний доступ і пропозиція змін до нього іншим користувачем.

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).

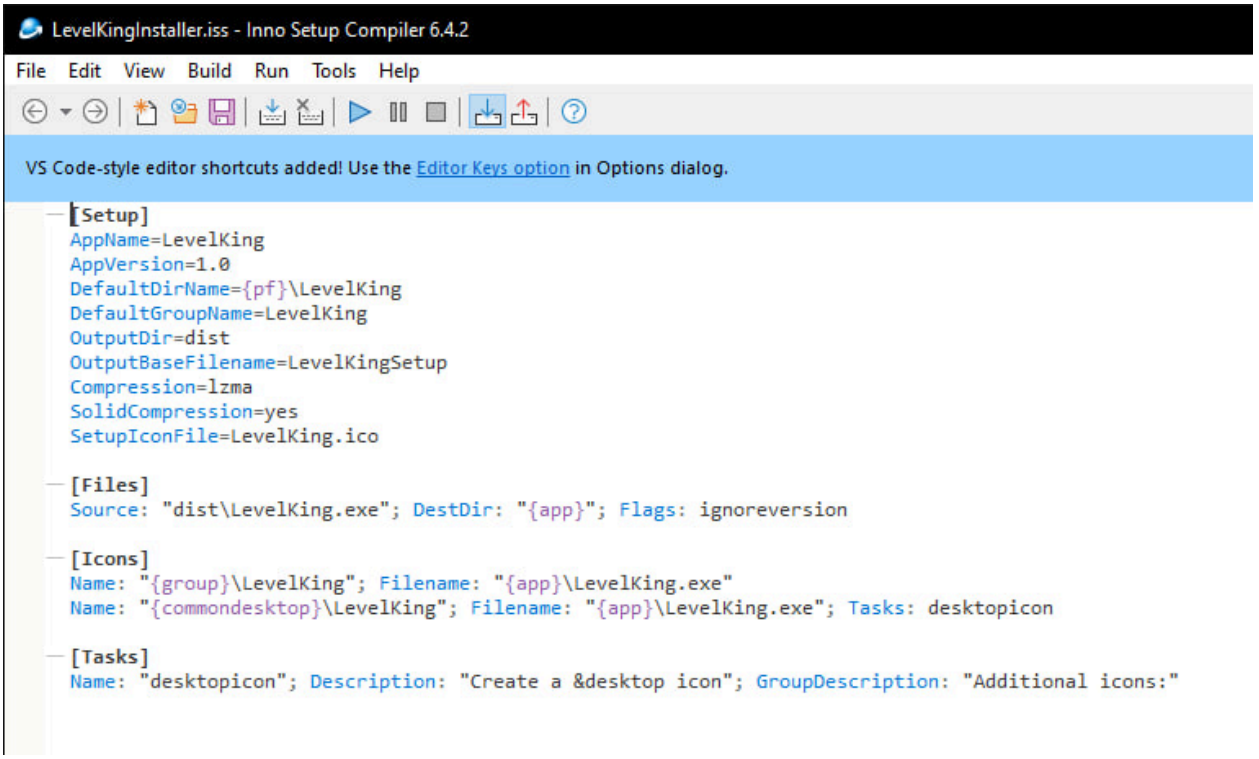


Рисунок 5.1 – Файл конфігурації для Inno Setup

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).

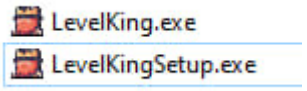


Рисунок 5.2 – Виконуваний файл та інсталятор гри

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі «Керівництво користувача» [КПІ.ІП-1230.045480.05.34].

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.

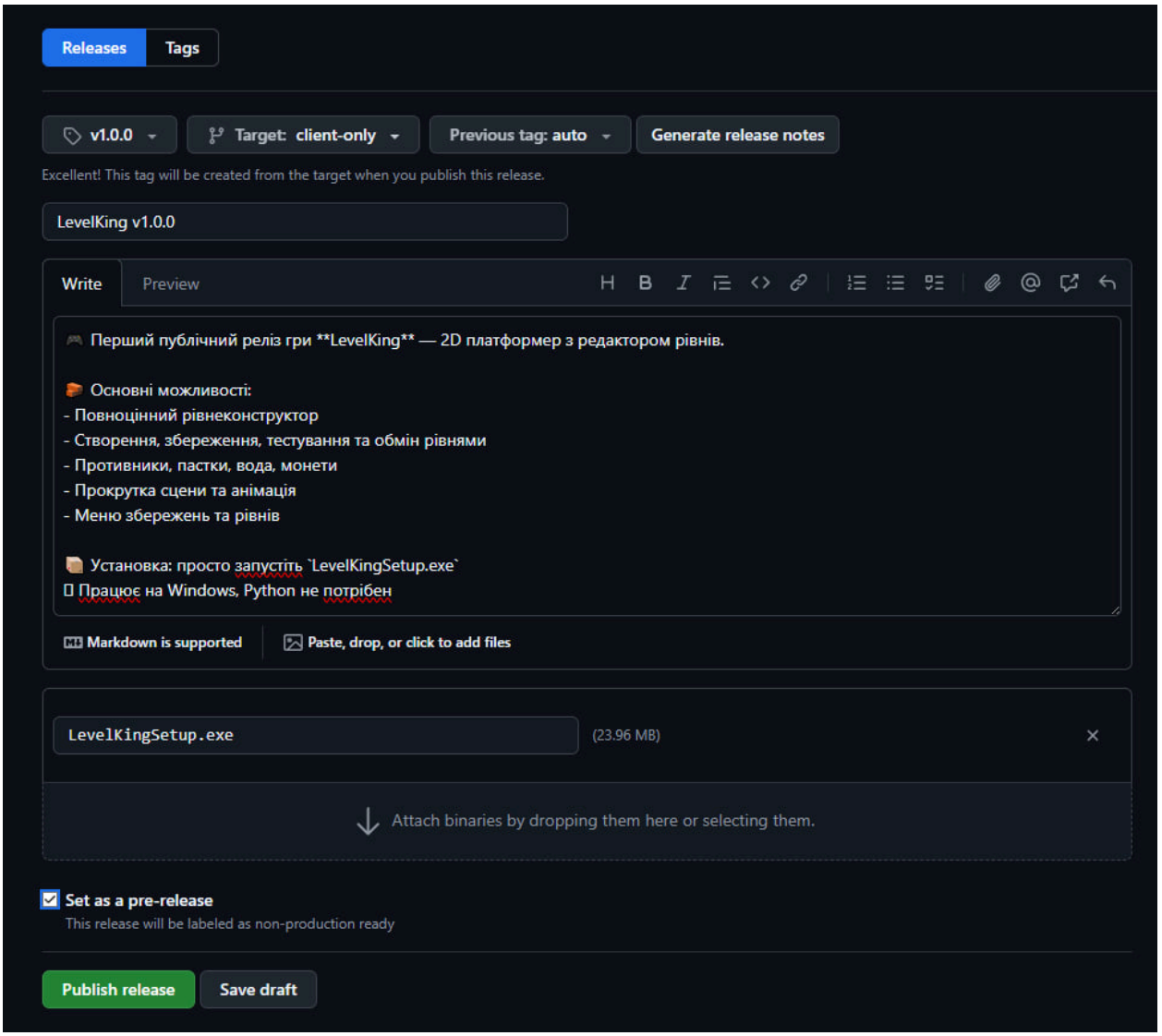


Рисунок 5.3 – Інтерфейс створення релізу на GitHub

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 розширення функціональності гри платформеру за рахунок підтримки спільної творчості гравців та можливості редагувати власні рівні, була повністю досягнута завдяки реалізації наступного функціоналу:

Серверна частина побудована на фреймворку Django та виконує роль централізованого сховища і координатора взаємодії між користувачами гри. Вона дозволяє: зберігати публічні рівні в базі даних у форматі JSON з метаданими (назва, опис, автор, дата створення тощо), отримувати список доступних рівнів, як для власника, так і для сторонніх користувачів, надавати API для взаємодії клієнтської частини з сервером (публікація, редагування, отримання рівнів тощо), забезпечити ізольований обмін рівнями, завдяки чому користувачі можуть ділитися результатами своєї творчості без прямого доступу до файлів, вести облік авторства рівнів та контроль версій через пропозиції змін.

Це дозволяє реалізувати спільну творчість, зберігаючи цілісність і безпеку даних.

Для керування доступом і відстеження дій користувачів реалізовано систему авторизації з JWT (JSON Web Token) за допомогою пакету SimpleJWT. Вона надає: реєстрацію нового користувача з перевіркою складності пароля, вхід у систему із збереженням токена доступу (access) і оновлення (refresh), ідентифікацію користувачів у запитах до серверу, що дозволяє, визначати, хто створив певний рівень; обмежити функціонал для неавторизованих користувачів (наприклад, публікація рівнів, подання пропозицій); забезпечити безпеку та персоналізацію (лише власник бачить свої локальні/публічні рівні, свої пропозиції тощо).

Це дає змогу гарантувати, що дії у грі виконуються лише тими користувачами, які мають на це право.

Редактор рівнів — це потужний інструмент у клієнтській частині гри, який дозволяє, створювати власні ігрові рівні за допомогою інтерактивного інтерфейсу, що включає: розміщення блоків террейну; додавання ворогів, пасток, об'єктів, монет, лінії горизонту тощо; керування шарами, стилями, зміщенням сітки; редагувати раніше створені рівні як локально, так і в контексті пропозицій змін. зберігати рівень у файл або відправляти на сервер, у форматі, зручному для подальшої обробки.

Завдяки редактору гра перетворюється з фіксованого набору рівнів на відкриту платформу для створення контенту.

Функціонал пропонування змін у рівнях з можливістю схвалення або відхилення їх автором. Цей функціонал реалізує механізм спільної творчості, де один користувач може: переглянути публічний рівень іншого користувача; увійти в режим пропозиції, редагуючи копію рівня у своєму редакторі; надіслати автору змінений варіант разом з описом. власник рівня бачить пропозицію і може: ознайомитися з описом змін, переглянути рівень; спробувати змінену версію, переконатися у її доцільності; прийняти або відхилити пропозицію.

Це дозволяє підтримувати контроль над власним контентом, водночас відкриваючи простір для колективного покращення рівнів.

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).

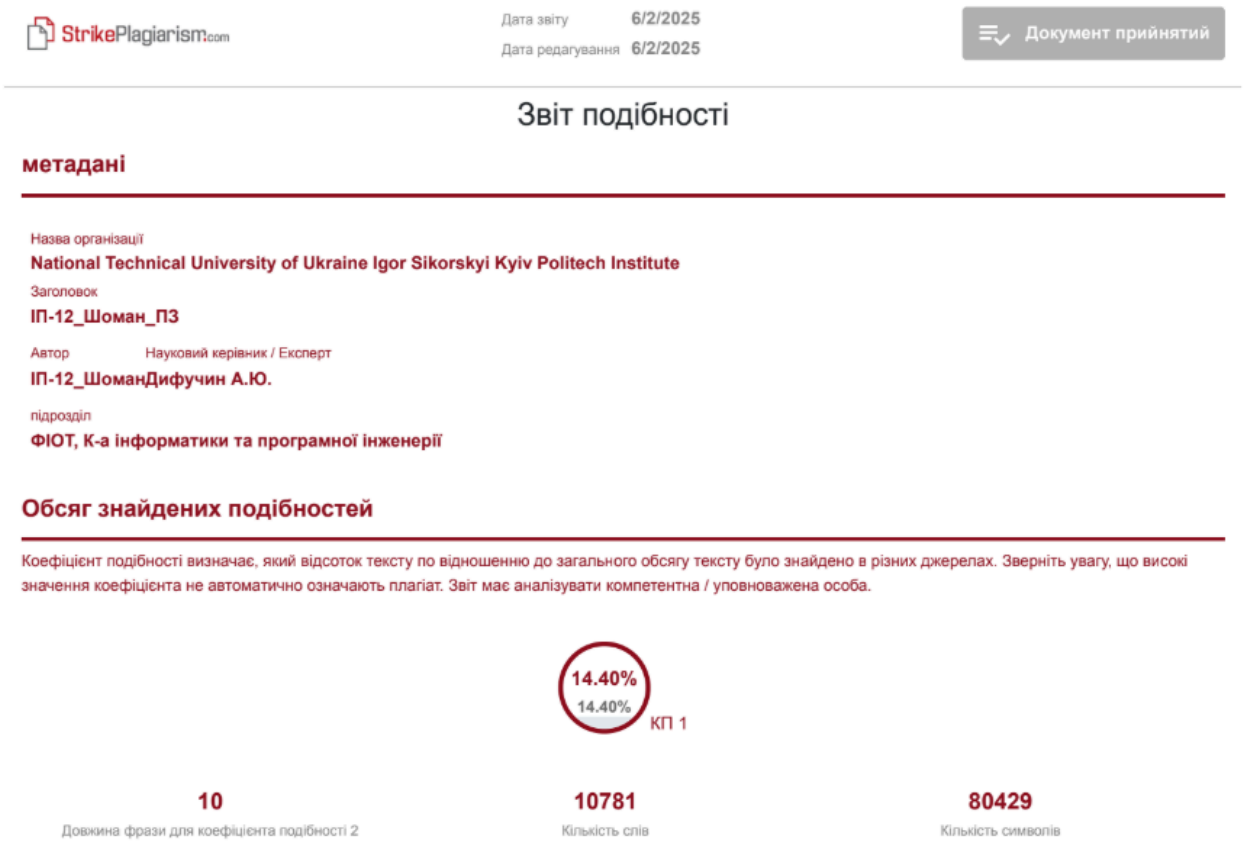
17. Tilesetter. *Steam*. URL: https://store.steampowered.com/app/1105890/Tilesetter/ (дата звернення 10.05.2025).

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_Шоман_ПЗ
Автор Науковий керівник / Експерт
ІП-12_ШоманДифучин 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.

14.40%
14.40%
КП 1

10	10781	80429
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-1230.045480.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Данило ШОМАН

Київ – 2025

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

Файл main.py
Реалізація функціональної задачі серверної взаємодії застосунку

```
    def log_user(self):
        self.own_public_levels = self.get_public_levels_by_author()

self.server_menu.public_level_list.update_items(self.own_public_levels)
        self.server_menu.local_level_list.update_items_local()
        self.public_levels = self.client.get_public_levels()
        self.public_levels_menu.level_list.update_items(self.public_levels)
        self.suggestions = self.client.view_change_requests()
        self.suggestions_menu.suggestions_list.update_items(self.suggestions)

    def get_public_levels_by_author(self):
        author_username = self.client.username
        return self.client.get_public_levels_by_author(author_username)

    def decide_chosen_level(self, id):
        self.chosen_level = self.client.get_level_by_id(id)
        self.public_level_menu.input_fields.update_info(self.chosen_level)

    def decide_chosen_level_local(self, name):
        file_path = path.join(script_directory, 'saved_levels',
f'{name}.json')
        with open(file_path, 'rb') as file:
            grid = pickle.load(file)
        self.level_builder.update_from_grid(grid)
        self.chosen_level_local = [self.serialize_grid(grid[0]), grid[1]]
        self.local_level_menu.input_fields.update_info(name)

    def publish_level(self):
        grid = self.level_builder.generate_grid()
        if validate_level_data(grid):
            self.client.upload_level(
                title=self.local_level_menu.input_fields.get_level(),

description=self.local_level_menu.input_fields.get_description(),
                data=[self.serialize_grid(grid[0]), grid[1]],
                is_public=True
            )
        else:
            print('Bad data')

    def delete_public_level(self):
        level_id_to_delete = self.chosen_level.get('id')
        self.client.delete_level(level_id_to_delete)

    def update_level_builder_grid(self):
        grid = self.chosen_level.get('data')
        self.level_builder.update_from_grid([self.deserialize_grid(grid[0]),
grid[1]])

    def save_own_level(self, tittle, description):
        id = self.chosen_level.get('id')
        data = self.level_builder.generate_grid()
        if validate_level_data(data):
            self.client.edit_level(
```

```
                level_id=id,
                title=tittle,
                description=description,
                data=[self.serialize_grid(data[0]), data[1]]
            )
        else:
            print('Bad data')

    def delete_own(self):
        if self.client.delete_account():
            print("Акаунт успішно видалено")
        else:
            print("Помилка при видаленні акаунта")

    def get_suggestions(self):
        return self.client.view_change_requests()

    def decide_chosen_suggestion(self, id):
        self.chosen_suggestion = self.client.get_change_request_by_id(id)
        self.suggestion_menu.labels.update_info(self.chosen_suggestion)

    def get_suggestion_grid(self):
        return
self.deserialize_grid(self.chosen_suggestion.get('proposed_data')[0])

    def apply_changes(self):
        level_id = self.chosen_suggestion.get('level_id')
        change_id = self.chosen_suggestion.get('id')
        if self.client.accept_change(level_id, change_id):
            print("Зміну успішно схвалено")
        else:
            print("Не вдалося схвалити зміну")

    def deny_changes(self):
        level_id = self.chosen_suggestion.get('level_id')
        change_id = self.chosen_suggestion.get('id')
        if self.client.reject_change(level_id, change_id):
            print("Зміну відхилено")
        else:
            print("Не вдалося відхилити зміну")

    def get_public_levels(self):
        return self.client.get_public_levels()

    def decide_chosen_strlevel(self, id):
        self.chosen_strlevel = self.client.get_level_by_id(id)
        self.stranger_level_menu.labels.update_info(self.chosen_strlevel)
        self.suggest_menu.labels.update_info(self.chosen_strlevel)

    def get_strlevel_grid(self):
        return self.deserialize_grid(self.chosen_strlevel.get('data')[0])

    def update_level_builder_grid_str(self):
        grid = self.chosen_strlevel.get('data')
        self.level_builder.update_from_grid([self.deserialize_grid(grid[0]),
grid[1]])

    def suggest(self):
        grid = self.level_builder.generate_grid()
        change_data = {
            "data": [self.serialize_grid(grid[0]), grid[1]],
            "comment": self.suggest_menu.input_fields.get_comment()
        }
        if validate_level_data(grid):
```

```
self.client.propose_change(level_id=self.chosen_strlevel.get('id'),
data=change_data.get('data'),
                                       comment=change_data.get('comment'))
        else:
            print('Bad data')

    def deserialize_grid(self, data_from_client: dict) -> dict:
        return {
            layer: {
                tuple(map(int, key.split(','))): value
                for key, value in layer_data.items()
            }
            for layer, layer_data in data_from_client.items()
        }

    def serialize_grid(self, grid_data: dict) -> dict:
        return {
            layer: {
                f'{key[0]},{key[1]}': value
                for key, value in layer_data.items()
            }
            for layer, layer_data in grid_data.items()
        }
```

Файл level_builder.py

Реалізація функціональної задачі створення редактору рівнів

```
class LevelBuilder:
    def __init__(self, soil_tiles, switch):
        script_directory = path.dirname(path.realpath(__file__))
        self.display_surface = pygame.display.get_surface()
        self.layout_elements = {}
        self.switch = switch
        self.from_where = ''
        self.switch_locker = True
        self.window_size = self.display_surface.get_size()

        self.import_graphics()
        self.soil_tiles = soil_tiles

        self.starting_point = vector()
        self.layout_active = False
        self.layout_offset = vector()
        self.selected_item_index = 2
        self.previously_selected_cell = None
        self.items_selector = Items_selector()

        self.layout_objects = pygame.sprite.Group()
        self.foreground_object = pygame.sprite.Group()
        self.background_object = pygame.sprite.Group()
        self.grid_lines_surface = pygame.Surface(self.window_size)
        self.grid_lines_surface.set_colorkey('green')
        self.grid_lines_surface.set_alpha(25)
        self.object_drag_active = False
        self.object_spawn_timer = Timer(300)

        self.create_player((200, self.window_size[1] / 2))
        self.create_sky((self.window_size[0] / 2, self.window_size[1] / 2))
```

```
        self.layout_clouds = []
        self.cloud_surface = read_folder(path.join(script_directory,
'graphics', 'clouds'))
        self.cloud_spawn_time = pygame.USEREVENT + 1
        pygame.time.set_timer(self.cloud_spawn_time, 2000)
        self.startup_clouds()


    def create_player(self, pos):
        self.player = LayoutMovableObject(
            pos=pos,
            frames=self.animation_assets[0]['frames'],
            tile_id=0,
            starting_point=self.starting_point,
            group=[self.layout_objects, self.foreground_object])

    def create_sky(self, pos):
        self.sky_handle = LayoutMovableObject(
            pos=pos,
            frames=[self.sky_handle_surf],
            tile_id=1,
            starting_point=self.starting_point,
            group=[self.layout_objects, self.background_object])

    def import_graphics(self):
        def gpath(*args):
            return path.join(script_directory, 'graphics', *args)

        def load_img(*args):
            return load(gpath(*args)).convert_alpha()

        def load_folder(*args):
            return read_folder(gpath(*args))

        def load_multiple(*args):
            return read_multiple_folders(gpath(*args))

        script_directory = path.dirname(path.realpath(__file__))
        graphics_dir = path.join(script_directory, 'graphics')

        self.water_bottom = load_img('terrain', 'water', 'water_bottom.png')
        self.platform = load_img('terrain', 'platform', 'platform.png')
        self.sky_handle_surf = load_img('cursors', 'handle.png')
        self.bricks_bg = load_img('terrain', 'bricks', 'bricks_bg',
'bricks_bg.png')
        self.bricks_fg = load_img('terrain', 'bricks', 'bricks_fg',
'bricks_fg.png')

        info_dir = lambda name: load_img('info', name, f'{name}.png')
        for info in ['interract_info', 'finish_info', 'sign_bottom',
'sign_top', 'sign_left', 'sign_right']:
            setattr(self, info, info_dir(info))

        self.level_builder_map = {}
        definitions = [
            (0, 'player', 'object', None, ['player', 'idle_right']),
            (1, 'sky', 'object', None, None),
            (2, 'terrain', 'tile', ['preview', 'soil.png'], None),
            (3, 'water', 'tile', ['preview', 'water.png'], ['terrain',
'water', 'animation']),
            (4, 'coin', 'tile', ['preview', 'golden_coin.png'], ['items',
'golden_coin']),
            (5, 'coin', 'tile', ['preview', 'silver_coin.png'], ['items',
'silver_coin']),
```

```
            (6, 'coin', 'tile', ['preview', 'diamond.png'], ['items',
'diamond']),
            (7, 'enemy', 'tile', ['preview', 'spikes.png'], ['enemies',
'spikes']),
            (8, 'enemy', 'tile', ['preview', 'pig.png'], ['enemies', 'pig',
'idle']),
            (9, 'enemy', 'tile', ['preview', 'cannon_left.png'], ['enemies',
'cannon_left', 'idle']),
            (10, 'enemy', 'tile', ['preview', 'cannon_right.png'],
['enemies', 'cannon_right', 'idle']),
            (11, 'palm_fg', 'object', ['preview', 'small_fg.png'],
['terrain', 'palm', 'small_fg']),
            (12, 'palm_fg', 'object', ['preview', 'large_fg.png'],
['terrain', 'palm', 'large_fg']),
            (13, 'palm_fg', 'object', ['preview', 'left_fg.png'], ['terrain',
'palm', 'left_fg']),
            (14, 'palm_fg', 'object', ['preview', 'right_fg.png'],
['terrain', 'palm', 'right_fg']),
            (15, 'palm_bg', 'object', ['preview', 'small_bg.png'],
['terrain', 'palm', 'small_bg']),
            (16, 'palm_bg', 'object', ['preview', 'large_bg.png'],
['terrain', 'palm', 'large_bg']),
            (17, 'palm_bg', 'object', ['preview', 'left_bg.png'], ['terrain',
'palm', 'left_bg']),
            (18, 'palm_bg', 'object', ['preview', 'right_bg.png'],
['terrain', 'palm', 'right_bg']),
            (19, 'bricks_fg', 'tile', ['preview', 'bricks_fg.png'],
['terrain', 'bricks', 'bricks_fg']),
            (20, 'bricks_bg', 'tile', ['preview', 'bricks_bg.png'],
['terrain', 'bricks', 'bricks_bg']),
            (21, 'enemy', 'tile', ['preview', 'saw_bottom.png'], ['enemies',
'saw_bottom']),
            (22, 'enemy', 'tile', ['preview', 'saw_top.png'], ['enemies',
'saw_top']),
            (23, 'enemy', 'tile', ['preview', 'saw_left.png'], ['enemies',
'saw_left']),
            (24, 'enemy', 'tile', ['preview', 'saw_right.png'], ['enemies',
'saw_right']),
            (25, 'platform', 'tile', ['preview', 'platform.png'], ['terrain',
'platform']),
            (26, 'interract_info', 'tile', ['preview', 'info.png'], ['info',
'interract_info', 'interract_info']),
            (27, 'finish_info', 'tile', ['preview', 'finish_info.png'],
['info', 'finish_info', 'finish_info']),
            (28, 'sign_bottom', 'tile', ['preview', 'sign_bottom.png'],
['info', 'sign_bottom', 'sign_bottom']),
            (29, 'sign_top', 'tile', ['preview', 'sign_top.png'], ['info',
'sign_top', 'sign_top']),
            (30, 'sign_left', 'tile', ['preview', 'sign_left.png'], ['info',
'sign_left', 'sign_left']),
            (31, 'sign_right', 'tile', ['preview', 'sign_right.png'],
['info', 'sign_right', 'sign_right']),
            (32, 'enemy', 'tile', ['preview', 'mace.png'], ['enemies',
'chained_mace']),
            (33, 'coin', 'tile', ['preview', 'heal_potion.png'], ['items',
'heal_potion']),
        ]

        for idx, style, type_, preview, graphics in definitions:
            self.level_builder_map[idx] = {
                'style': style,
                'type': type_,
                'preview': gpath(*preview) if preview else None,
                'graphics': gpath(*graphics) if graphics else None,
```

```
            }

        self.animation_assets = {}
        for key, value in self.level_builder_map.items():
            path_to_graphics = value.get('graphics')
            if path_to_graphics:
                frames = read_folder(path_to_graphics)
                self.animation_assets[key] = {
                    'frame_index': 0,
                    'frames': frames,
                    'length': len(frames)
                }

        self.preview_surfs = {key: load(value['preview']) for key, value in
self.level_builder_map.items() if value['preview']}

    def update_frames(self, dt):
        speed = 8 * dt
        for anim in self.animation_assets.values():
            index = anim.get('frame index', 0) + speed
            anim['frame index'] = index if index < anim['length'] else 0

    def mouse_which_object(self):
        for entity in self.layout_objects:
            if entity.rect.collidepoint(mouse_pos()):
                return entity

    def update_from_grid(self, coords):
        self.layout_elements = {}
        self.layout_objects.empty()
        self.foreground_object.empty()
        self.background_object.empty()
        grid = coords[0]
        topleft = coords[1]
        left = topleft[0]
        top = topleft[1]

        for layer_name, layer in grid.items():
            for pos_str, value in layer.items():
                if isinstance(pos_str, str):
                    x, y = map(int, pos_str.split(','))
                else:
                    x, y = pos_str

                abs_x = x + left * tile_size
                abs_y = y + top * tile_size
                tile_x = abs_x // tile_size
                tile_y = abs_y // tile_size
                tile_pos = (tile_x, tile_y)

                if tile_pos not in self.layout_elements:
                    self.layout_elements[tile_pos] =
LayoutObject(self.level_builder_map)

                entity = self.layout_elements[tile_pos]

                if layer_name == 'terrain':
                    if value == '0':
                        entity.terrain_neighbors = []
                        entity.has_terrain = True
                    else:
                        entity.terrain_neighbors = list(value)
                        entity.has_terrain = True
```

```
                elif layer_name == 'water':
                    entity.water_on_top = (value == 'bottom')

                elif layer_name == 'platforms':
                    entity.has_platform = True

                elif layer_name == 'bricks bg':
                    entity.has_bricks_bg = True

                elif layer_name == 'bricks fg':
                    entity.has_bricks_fg = True

                elif layer_name == 'coins':
                    entity.coin = value

                elif layer_name == 'enemies':
                    entity.enemy = value

                elif layer_name == 'interract_info':
                    entity.has_interract_info = True

                elif layer_name == 'finish_info':
                    entity.has_finish_info = True

                elif layer_name == 'sign bottom':
                    entity.has_sign_bottom = True

                elif layer_name == 'sign top':
                    entity.has_sign_top = True

                elif layer_name == 'sign left':
                    entity.has_sign_left = True

                elif layer_name == 'sign right':
                    entity.has_sign_right = True

                elif layer_name in ['bg palms', 'fg objects']:
                    offset = vector(abs_x - tile_x * tile_size, abs_y -
tile_y * tile_size)
                    tile_id = value
                    entity.objects.append((tile_id, offset))

        for tile_pos, entity in self.layout_elements.items():
            for tile_id, offset in entity.objects:
                screen_pos = (vector(tile_pos) * tile_size) + offset +
self.layout_offset
                if tile_id != 0 and tile_id != 1:
                    groups = [self.layout_objects, self.background_object] if
self.level_builder_map[tile_id]['style'] == 'palm_bg' else [
                        self.layout_objects, self.foreground_object]
                    LayoutMovableObject(
                        pos=screen_pos,
                        frames=self.animation_assets[tile_id]['frames'],
                        tile_id=tile_id,
                        starting_point=self.starting_point,
                        group=groups,
                        is_input=True
                    )
                elif tile_id == 0:
                    self.create_player(screen_pos)
                elif tile_id == 1:
                    self.create_sky(screen_pos)

    def get_cell_coordinates(self, obj=None):
```

```
        pos = vector(mouse_pos()) if obj is None else
vector(obj.distance_to_starting_point)
        local_pos = pos - self.starting_point
        col = floor(local_pos.x / tile_size)
        row = floor(local_pos.y / tile_size)
        return col, row

    def generate_grid(self):
        for entity in self.layout_elements.values():
            entity.objects.clear()

        for object in self.layout_objects:
            entity = self.get_cell_coordinates(object)
            offset = vector(object.distance_to_starting_point) -
vector(entity) * tile_size

            if entity in self.layout_elements:
                self.layout_elements[entity].add_object(object.tile_id,
offset)
            else:
                self.layout_elements[entity] =
LayoutObject(self.level_builder_map, object.tile_id, offset)

        layer_names = [
            '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'
        ]
        layers = {name: {} for name in layer_names}

        min_x = min(tile[0] for tile in self.layout_elements)
        min_y = min(tile[1] for tile in self.layout_elements)
        offset = [min_x, min_y]

        for (entity_x, entity_y), entity in self.layout_elements.items():
            rel_x = (entity_x - min_x) * tile_size
            rel_y = (entity_y - min_y) * tile_size
            base_pos = (rel_x, rel_y)

            if entity.has_water:
                layers['water'][base_pos] = entity.get_water()

            if entity.has_platform:
                layers['platforms'][base_pos] = 'platform'

            if entity.has_terrain:
                terrain = entity.get_terrain()
                layers['terrain'][base_pos] = terrain if terrain in
self.soil_tiles else '0'

            if entity.coin:
                coin_pos = (rel_x + tile_size // 2, rel_y + tile_size // 2)
                layers['coins'][coin_pos] = entity.coin

            if entity.enemy:
                layers['enemies'][base_pos] = entity.enemy

            if entity.has_bricks_bg:
                layers['bricks bg'][base_pos] = 'bricks_bg'

            if entity.has_bricks_fg:
                layers['bricks fg'][base_pos] = 'bricks_fg'
```

```
            for obj_id, offset_vec in entity.objects:
                target = 'bg palms' if self.level_builder_map.get(obj_id,
{}).get('style') == 'palm_bg' else 'fg objects'
                obj_pos = (int(rel_x + offset_vec.x), int(rel_y +
offset_vec.y))
                layers[target][obj_pos] = obj_id

            info_flags = {
                'interract_info': entity.has_interract_info,
                'finish_info': entity.has_finish_info,
                'sign bottom': entity.has_sign_bottom,
                'sign top': entity.has_sign_top,
                'sign left': entity.has_sign_left,
                'sign right': entity.has_sign_right,
            }

            for name, flag in info_flags.items():
                if flag:
                    layers[name][base_pos] = name.replace(' ', '_')

        return [layers, offset]

    def event_loop(self):
        for event in pygame.event.get():
            if self._handle_quit(event):
                return

            if self._handle_return(event):
                return

            self.actions_input(event)
            self.buttons_click(event)
            self.update_object_drag(event)
            self.layout_add()
            self.layout_remove()
            self.selection_item_hotkeys(event)
            self.add_clouds(event)

    def _handle_quit(self, event):
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        return False

    def _handle_return(self, event):
        if (
                event.type == pygame.KEYDOWN and
                event.key == pygame.K_RETURN and
                self.switch_locker
        ):
            grid_data = self.generate_grid()
            if len(grid_data[0]['terrain']) != 0:
                self.switch_locker = False
                self.switch({'from': 'level_builder', 'to': 'level'},
grid_data[0])
                return True
        return False

    def actions_input(self, event):
        self._handle_middle_mouse(event)
        self._handle_scroll(event)
        self._update_layout()
```

```
    def _handle_middle_mouse(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN and mouse_buttons()[1]:
            self.layout_active = True
            self.layout_offset = vector(mouse_pos()) - self.starting_point
        elif not mouse_buttons()[1]:
            self.layout_active = False

    def _handle_scroll(self, event):
        if event.type == pygame.MOUSEWHEEL:
            delta = event.y * 50
            if pygame.key.get_pressed()[pygame.K_LCTRL]:
                self.starting_point.y -= delta
            else:
                self.starting_point.x -= delta
            for entity in self.layout_objects:
                entity.layout_pos(self.starting_point)

    def _update_layout(self):
        if self.layout_active:
            self.starting_point = vector(mouse_pos()) - self.layout_offset
            for entity in self.layout_objects:
                entity.layout_pos(self.starting_point)

    def selection_item_hotkeys(self, event):
        if event.type != pygame.KEYDOWN:
            return

        key_delta = {
            pygame.K_RIGHT: 1,
            pygame.K_LEFT: -1
        }

        delta = key_delta.get(event.key)
        if delta:
            self.selected_item_index += delta
            self.selected_item_index = max(2, min(self.selected_item_index,
33))

    def buttons_click(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN and
self.items_selector.mm_rect.collidepoint(
                mouse_pos()) and self.switch_locker == True:
            self.switch_locker = False
            self.switch({'from': 'level_builder', 'to':
f'{self.from_where}'})
        if event.type == pygame.MOUSEBUTTONDOWN and
self.items_selector.rect.collidepoint(mouse_pos()):
            new_index = self.items_selector.on_click(mouse_pos(),
mouse_buttons())
            self.selected_item_index = new_index if new_index else
self.selected_item_index
        if event.type == pygame.MOUSEBUTTONDOWN and
self.items_selector.sv_rect.collidepoint(
                mouse_pos()) and self.switch_locker and
len(self.generate_grid()[0]['terrain']) != 0:
            self.switch_locker = False
            self.switch({'from': 'level_builder', 'to': 'save_menu'})

    def layout_add(self):
        if not self._is_layout_clickable():
            return

        chosen_cell = self.get_cell_coordinates()
        selected = self.level_builder_map[self.selected_item_index]
```

```
        if selected['type'] == 'tile':
            if chosen_cell == self.previously_selected_cell:
                return

            tile = self.layout_elements.get(chosen_cell)
            if tile:
                tile.add_object(self.selected_item_index)
            else:
                self.layout_elements[chosen_cell] =
LayoutObject(self.level_builder_map, self.selected_item_index)

            self.adjust_tile_correlation(chosen_cell)
            self.previously_selected_cell = chosen_cell

        else:
            if not self.object_spawn_timer.is_active():
                groups = [self.layout_objects, self.background_object] \
                    if selected.get('style') == 'palm_bg' \
                    else [self.layout_objects, self.foreground_object]

                LayoutMovableObject(
                    pos=mouse_pos(),

frames=self.animation_assets[self.selected_item_index]['frames'],
                    tile_id=self.selected_item_index,
                    starting_point=self.starting_point,
                    group=groups
                )
                self.object_spawn_timer.activate()

    def adjust_tile_correlation(self, cell_pos):
        offsets_map = {
            '1': (0, -1), '2': (1, -1), '3': (1, 0), '4': (1, 1),
            '5': (0, 1), '6': (-1, 1), '7': (-1, 0), '8': (-1, -1)
        }

        offset_x, offset_y = cell_pos
        half = 1

        for dx in range(-half, half + 1):
            for dy in range(-half, half + 1):
                cell = (offset_x + dx, offset_y + dy)
                if cell not in self.layout_elements:
                    continue

                cell_data = self.layout_elements[cell]
                cell_data.terrain_neighbors = []
                cell_data.water_on_top = False

                for name, (nx, ny) in offsets_map.items():
                    neighbor = (cell[0] + nx, cell[1] + ny)
                    if neighbor not in self.layout_elements:
                        continue

                    neighbor_data = self.layout_elements[neighbor]

                    if name == '1' and cell_data.has_water and
neighbor_data.has_water:
                        cell_data.water_on_top = True

                    if neighbor_data.has_terrain:
                        cell_data.terrain_neighbors.append(name)
```

```
    def _is_layout_clickable(self):
        if not mouse_buttons()[0] or self.object_drag_active:
            return False
        pos = mouse_pos()
        return not (
                self.items_selector.rect.collidepoint(pos) or
                self.items_selector.mm_rect.collidepoint(pos) or
                self.items_selector.sv_rect.collidepoint(pos)
        )

    def layout_remove(self):
        if not mouse_buttons()[2] or
self.items_selector.rect.collidepoint(mouse_pos()):
            return

        obj = self.mouse_which_object()
        if obj and self.level_builder_map.get(obj.tile_id, {}).get('style')
not in ('player', 'sky'):
            obj.kill()
            return

        current_cell = self.get_cell_coordinates()
        tile = self.layout_elements.get(current_cell)

        if tile:
            tile.remove_object(self.selected_item_index)
            if tile.is_empty:
                del self.layout_elements[current_cell]
            self.adjust_tile_correlation(current_cell)

    def update_object_drag(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN and mouse_buttons()[0]:
            self._start_object_drag(event.pos)
        elif event.type == pygame.MOUSEBUTTONUP and self.object_drag_active:
            self._end_object_drag()

    def _start_object_drag(self, pos):
        for entity in self.layout_objects:
            if entity.rect.collidepoint(pos):
                entity.start_drag()
                self.object_drag_active = True
                break

    def _end_object_drag(self):
        for entity in self.layout_objects:
            if entity.selected:
                entity.end_drag(self.starting_point)
                self.object_drag_active = False
                break

    def draw_grid_lines(self):
        columns = self.window_size[0] // tile_size
        rows = self.window_size[1] // tile_size

        offset_x = self.starting_point.x - floor(self.starting_point.x /
tile_size) * tile_size
        offset_y = self.starting_point.y - floor(self.starting_point.y /
tile_size) * tile_size

        self.grid_lines_surface.fill('green')

        for col in range(columns + 1):
            x = offset_x + col * tile_size
```

```
            pygame.draw.line(self.grid_lines_surface, 'black', (x, 0), (x,
self.window_size[1]))

        for row in range(rows + 1):
            y = offset_y + row * tile_size
            pygame.draw.line(self.grid_lines_surface, 'black', (0, y),
(self.window_size[0], y))

        self.display_surface.blit(self.grid_lines_surface, (0, 0))

    def draw_level_layout(self):
        self.background_object.draw(self.display_surface)

        animated_types = {
            'water': (3, self.water_bottom),
            'coin': lambda tile: tile.coin,
            'enemy': lambda tile: tile.enemy
        }

        static_tiles = {
            'terrain': lambda tile: self._get_terrain_surface(tile),
            'bricks_fg': self.bricks_fg,
            'bricks_bg': self.bricks_bg,
            'platform': self.platform,
            'interract_info': self.interract_info,
            'finish_info': self.finish_info,
            'sign_bottom': self.sign_bottom,
            'sign_top': self.sign_top,
            'sign_left': self.sign_left,
            'sign_right': self.sign_right
        }

        for cell_pos, tile in self.layout_elements.items():
            pos = self.starting_point + vector(cell_pos) * tile_size

            if tile.has_water:
                if tile.water_on_top:
                    self.display_surface.blit(self.water_bottom, pos)
                else:
                    self._draw_animated(3, pos)

            for attr, surf in static_tiles.items():
                if getattr(tile, f'has_{attr}', False):
                    surface = surf(tile) if callable(surf) else surf
                    self.display_surface.blit(surface, pos)

            if tile.coin:
                self._draw_animated(tile.coin, pos, center=True)

            if tile.enemy:
                self._draw_animated(tile.enemy, pos, midbottom=True)

        self.foreground_object.draw(self.display_surface)

    def display_scenery(self, dt):
        self.display_surface.fill('#cce7ff')
        y = self.sky_handle.rect.centery
        height = self.window_size[1]

        if y > 0:
            self._draw_horizon_lines(y)
            self.render_clouds(dt, y)

        if 0 < y < height:
```

```
            self._draw_sea(y, height)
        elif y < 0:
            self.display_surface.fill('#a8c9a1')

    def _draw_horizon_lines(self, y):
        w = self.window_size[0]
        pygame.draw.rect(self.display_surface, '#f6d6bd', pygame.Rect(0, y -
10, w, 10))
        pygame.draw.rect(self.display_surface, '#f6d6bd', pygame.Rect(0, y -
16, w, 4))
        pygame.draw.rect(self.display_surface, '#f6d6bd', pygame.Rect(0, y -
20, w, 2))

    def _draw_sea(self, y, height):
        w = self.window_size[0]
        pygame.draw.rect(self.display_surface, '#a8c9a1', pygame.Rect(0, y,
w, height))
        pygame.draw.line(self.display_surface, '#f5f1de', (0, y), (w, y), 3)

    def render_clouds(self, dt, horizon_y):
        for cloud in self.layout_clouds:
            cloud_surf = cloud.get('surf')
            cloud_pos = cloud.get('pos')
            cloud_speed = cloud.get('speed', 0)

            if not cloud_surf or not cloud_pos:
                continue

            cloud_pos[0] -= cloud_speed * dt
            draw_x = cloud_pos[0]
            draw_y = horizon_y - cloud_pos[1]

            self.display_surface.blit(cloud_surf, (draw_x, draw_y))

    def add_clouds(self, event):
        if event.type != self.cloud_spawn_time:
            return

        self._spawn_cloud()
        self._cleanup_clouds()

    def _spawn_cloud(self):
        surf = choice(self.cloud_surface)
        if randint(0, 4) < 2:
            surf = pygame.transform.scale2x(surf)

        x = self.window_size[0] + randint(50, 100)
        y = randint(0, self.window_size[1])
        speed = randint(20, 50)

        self.layout_clouds.append({'surf': surf, 'pos': [x, y], 'speed':
speed})

    def _cleanup_clouds(self):
        self.layout_clouds = [
            cloud for cloud in self.layout_clouds if cloud['pos'][0] > -400
        ]

    def startup_clouds(self):
        for _ in range(20):
            cloud = self._generate_cloud()
            self.layout_clouds.append(cloud)

    def _generate_cloud(self):
```

```
        surf = choice(self.cloud_surface)
        if randint(0, 4) < 2:
            surf = pygame.transform.scale2x(surf)

        x = randint(0, self.window_size[0])
        y = randint(0, self.window_size[1])
        speed = randint(20, 50)

        return {'surf': surf, 'pos': [x, y], 'speed': speed}

    def _draw_animated(self, anim_id, pos, center=False, midbottom=False):
        frames = self.animation_assets[anim_id]['frames']
        index = int(self.animation_assets[anim_id]['frame index'])
        surf = frames[index]

        if center:
            rect = surf.get_rect(center=(pos[0] + tile_size // 2, pos[1] +
tile_size // 2))
        elif midbottom:
            rect = surf.get_rect(midbottom=(pos[0] + tile_size // 2, pos[1] +
tile_size))
        else:
            rect = surf.get_rect(topleft=pos)

        self.display_surface.blit(surf, rect)

    def _get_terrain_surface(self, tile):
        terrain_key = ''.join(tile.terrain_neighbors)
        key = terrain_key if terrain_key in self.soil_tiles else '0'
        return self.soil_tiles[key]

    def selected_object_preview(self):
        if self._is_mouse_over_ui():
            return

        selected_object = self.mouse_which_object()

        if selected_object:
            rect = selected_object.rect.inflate(10, 10)
            self._draw_selection_outline(rect)
        else:
            tile_type =
self.level_builder_map[self.selected_item_index]['type']
            surf = self.preview_surfs[self.selected_item_index].copy()
            surf.set_alpha(200)

            if tile_type == 'tile':
                cell = self.get_cell_coordinates()
                rect = surf.get_rect(topleft=self.starting_point +
vector(cell) * tile_size)
            else:
                rect = surf.get_rect(center=mouse_pos())

            self.display_surface.blit(surf, rect)

    def _is_mouse_over_ui(self):
        pos = mouse_pos()
        return (
                self.items_selector.rect.collidepoint(pos) or
                self.items_selector.mm_rect.collidepoint(pos) or
                self.items_selector.sv_rect.collidepoint(pos)
        )

    def _draw_selection_outline(self, rect):
```

```
        color = 'black'
        width = 3
        size = 15

        pygame.draw.lines(self.display_surface, color, False,
                          [(rect.left, rect.top + size), rect.topleft,
(rect.left + size, rect.top)], width)
        pygame.draw.lines(self.display_surface, color, False,
                          [(rect.right - size, rect.top), rect.topright,
(rect.right, rect.top + size)], width)
        pygame.draw.lines(self.display_surface, color, False,
                          [(rect.right - size, rect.bottom),
rect.bottomright, (rect.right, rect.bottom - size)], width)
        pygame.draw.lines(self.display_surface, color, False,
                          [(rect.left, rect.bottom - size), rect.bottomleft,
(rect.left + size, rect.bottom)], width)

    def run(self, dt):
        self.event_loop()
        self.update_frames(dt)
        self.layout_objects.update(dt)
        self.object_spawn_timer.update()
        self.display_surface.fill('gray')
        self.display_scenery(dt)
        self.draw_level_layout()
        self.draw_grid_lines()
        self.selected_object_preview()
        self.items_selector.render(self.selected_item_index)


class LayoutObject:
    def __init__(self, layout_map, tile_id=None, offset=vector()):
        self.layout_map = layout_map

        self.has_terrain = False
        self.terrain_neighbors = []

        self.has_water = False
        self.water_on_top = False

        self.has_platform = False

        self.has_interract_info = False
        self.has_finish_info = False
        self.has_sign_bottom = False
        self.has_sign_top = False
        self.has_sign_left = False
        self.has_sign_right = False

        self.has_bricks_fg = False
        self.has_bricks_bg = False

        self.coin = None
        self.enemy = None
        self.objects = []
        self.is_empty = True

        if tile_id is not None:
            self.add_object(tile_id, offset)

    def add_object(self, tile_id, offset=vector()):
        style = self._get_style(tile_id)

        if style in self._style_flags():
```

```
            setattr(self, self._style_flags()[style], True)
        elif style in self._style_fields():
            setattr(self, self._style_fields()[style], tile_id)
        else:
            if (tile_id, offset) not in self.objects:
                self.objects.append((tile_id, offset))

        self.is_empty = False

    def remove_object(self, tile_id):
        style = self._get_style(tile_id)

        if style in self._style_flags():
            setattr(self, self._style_flags()[style], False)
        elif style in self._style_fields():
            setattr(self, self._style_fields()[style], None)

        self._check_content()

    def _check_content(self):
        self.is_empty = not (self.has_terrain or self.has_water or self.coin
or self.enemy)

    def _get_style(self, tile_id):
        return self.layout_map[tile_id]['style']

    def _style_flags(self):
        return {
            'terrain': 'has_terrain',
            'water': 'has_water',
            'platform': 'has_platform',
            'bricks_fg': 'has_bricks_fg',
            'bricks_bg': 'has_bricks_bg',
            'interract_info': 'has_interract_info',
            'finish_info': 'has_finish_info',
            'sign_bottom': 'has_sign_bottom',
            'sign_top': 'has_sign_top',
            'sign_left': 'has_sign_left',
            'sign_right': 'has_sign_right',
        }

    def _style_fields(self):
        return {
            'coin': 'coin',
            'enemy': 'enemy'
        }

    def get_water(self):
        return 'bottom' if self.water_on_top else 'top'

    def get_terrain(self):
        return ''.join(self.terrain_neighbors)


class LayoutMovableObject(pygame.sprite.Sprite):
    def __init__(self, pos, frames, tile_id, starting_point, group,
is_input=False):
        super().__init__(group)
        self.tile_id = tile_id
        self.frames = frames
        self.frame_index = 0
        self.image = self.frames[0]

        self.rect = self._init_rect(pos, is_input)
```

```
        self.distance_to_starting_point = vector(self.rect.topleft) -
starting_point
        self.selected = False
        self.mouse_offset = vector()

    def _init_rect(self, pos, is_input):
        if is_input:
            return self.image.get_rect(topleft=pos)
        return self.image.get_rect(center=pos)

    def start_drag(self):
        self.selected = True
        self.mouse_offset = vector(mouse_pos()) - vector(self.rect.topleft)

    def drag(self):
        if self.selected:
            self.rect.topleft = mouse_pos() - self.mouse_offset

    def end_drag(self, starting_point):
        self.selected = False
        self.distance_to_starting_point = vector(self.rect.topleft) -
starting_point

    def animate(self, dt):
        self.frame_index = (self.frame_index + 8 * dt) % len(self.frames)
        self.image = self.frames[int(self.frame_index)]
        self.rect = self.image.get_rect(midbottom=self.rect.midbottom)

    def layout_pos(self, starting_point):
        self.rect.topleft = starting_point + self.distance_to_starting_point

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

Файл network.py

Реалізація функціональної задачі створення запитів до серверу

```
import requests

BASE_URL = 'http://127.0.0.1:8000/api/'


class APIClient:
    def __init__(self):
        self.access_token = None
        self.refresh_token = None
        self.username = ''

    def register_user(self, username: str, password: str) -> bool:
        url = BASE_URL + 'auth/register/'
        data = {
            'username': username,
            'password': password,
            'password2': password  # <- обов'язково!
        }
        response = requests.post(url, json=data)
        print('Register:', response.status_code, response.text)
        return response.status_code == 201

    def login(self, username: str, password: str) -> bool:
```

```
    """
    Авторизація користувача. Отримання токенів.
    """
    url = BASE_URL + 'auth/token/'
    data = {'username': username, 'password': password}
    response = requests.post(url, json=data)  # □ Тут теж краще json=
    if response.status_code == 200:
        tokens = response.json()
        self.access_token = tokens['access']
        self.refresh_token = tokens['refresh']
        self.username = username
        return True
    print('Login:', response.status_code, response.text)
    return False

def _headers(self):
    """
    Заголовки з JWT токеном.
    """
    return {'Authorization': f'Bearer {self.access_token}'}

def get_public_levels(self):
    """
    Отримати всі публічні рівні.
    """
    url = BASE_URL + 'levels/public/'
    response = requests.get(url, headers=self._headers())
    if response.status_code == 200:
        return response.json()
    print('Public levels error:', response.status_code, response.text)
    return []

def get_level_by_id(self, level_id: int):
    """
    Отримати рівень по ID.
    """
    url = BASE_URL + f'levels/{level_id}/'
    response = requests.get(url, headers=self._headers())
    if response.status_code == 200:
        return response.json()
    print('Level by ID error:', response.status_code, response.text)
    return None

def get_change_request_by_id(self, change_id):
    """
    Отримати конкретний запит на зміну за його ID.
    """
    url = BASE_URL + f'levels/changes/{change_id}/'
    response = requests.get(url, headers=self._headers())
    if response.status_code == 200:
        return response.json()
    print('Change request error:', response.status_code, response.text)
    return None

def upload_level(self, title, description, data, is_public=True):
    """
    Завантаження нового рівня на сервер.
    """
    url = BASE_URL + 'levels/upload/'
    payload = {
        'title': title,
        'description': description,
        'data': data,
        'is_public': is_public
```

```
        }
        response = requests.post(url, headers=self._headers(), json=payload)
        return response.status_code == 201

    def get_public_levels_by_author(self, username):
        """
        Отримати публічні рівні певного автора.
        """
        url = BASE_URL + f'levels/public/by-author/{username}/'
        response = requests.get(url, headers=self._headers())
        if response.status_code == 200:
            return response.json()
        return []

    def propose_change(self, level_id, data, comment=""):
        """
        Надіслати пропозицію змін до чужого рівня.
        """
        url = BASE_URL + f'levels/{level_id}/propose-change/'
        payload = {
            'data': data,
            'comment': comment
        }
        response = requests.post(url, headers=self._headers(), json=payload)
        return response.status_code == 201

    def view_change_requests(self):
        """
        Отримати список усіх запитів на зміну до власних рівнів.
        """
        url = BASE_URL + 'levels/changes/'
        response = requests.get(url, headers=self._headers())
        if response.status_code == 200:
            return response.json()
        print('Change requests error:', response.status_code, response.text)
        return []

    def accept_change(self, level_id, change_id):
        """
        Схвалити зміну до рівня.
        """
        url = BASE_URL + f'levels/{level_id}/changes/{change_id}/accept/'
        response = requests.post(url, headers=self._headers())
        return response.status_code == 200

    def reject_change(self, level_id, change_id):
        """
        Відхилити зміну до рівня.
        """
        url = BASE_URL + f'levels/{level_id}/changes/{change_id}/reject/'
        response = requests.post(url, headers=self._headers())
        return response.status_code == 200

    def delete_level(self, level_id):
        """
        Видалити власний рівень.
        """
        url = BASE_URL + f'levels/{level_id}/delete/'
        response = requests.delete(url, headers=self._headers())
        return response.status_code == 204

    def edit_level(self, level_id, **kwargs):
        """
        Редагувати власний рівень. Можна передати:
```

```
        - title
        - description
        - data
        - is_public
        """
        url = BASE_URL + f'levels/{level_id}/edit/'
        response = requests.patch(url, headers=self._headers(), json=kwargs)
        return response.status_code == 200

    def delete_account(self) -> bool:
        """
        Видалення власного облікового запису користувачем.
        """
        url = BASE_URL + 'auth/delete/'
        response = requests.delete(url, headers=self._headers())
        print("Delete account:", response.status_code, response.text)
        return response.status_code == 204
```

Файл views.py
Реалізація функціональної задачі роботи сервера та авторизації

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def protected_view(request):
    user = request.user
    return Response({
        'message': f'Привіт, {user.username}! Це захищена інформація.'
    })


@api_view(['POST'])
def register_user(request):
    serializer = RegisterSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response({"detail": "Користувача зареєстровано успішно."},
status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)


@api_view(['DELETE'])
@permission_classes([IsAuthenticated])
def delete_user(request):
    user = request.user

    Level.objects.filter(author=user).delete()

    LevelChangeRequest.objects.filter(proposer=user).delete()

    user.delete()

    return Response({"detail": "Користувача та всі пов’язані дані
видалено."}, status=status.HTTP_204_NO_CONTENT)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def upload_level(request):
    serializer = LevelSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save(author=request.user)
        return Response(serializer.data, status=201)
    return Response(serializer.errors, status=400)
```

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def public_levels(request):
    levels = Level.objects.filter(is_public=True)
    serializer = LevelSerializer(levels, many=True)
    return Response(serializer.data)


@api_view(['GET'])
@permission_classes([IsAuthenticated])
def public_levels_by_author(request, username):
    levels = Level.objects.filter(is_public=True, author__username=username)
    serializer = LevelSerializer(levels, many=True)
    return Response(serializer.data)


@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_level_by_id(request, level_id):
    try:
        level = Level.objects.get(id=level_id)
        return Response({
            'id': level.id,
            'title': level.title,
            'description': level.description,
            'data': level.data,
            'is_public': level.is_public,
            'author': level.author.username,
        })
    except Level.DoesNotExist:
        return Response({'detail': 'Рівень не знайдено.'},
status=status.HTTP_404_NOT_FOUND)


@api_view(['POST'])
@permission_classes([IsAuthenticated])
def propose_change(request, level_id):
    try:
        original = Level.objects.get(id=level_id)
    except Level.DoesNotExist:
        return Response({'detail': 'Рівень не знайдено.'},
status=status.HTTP_404_NOT_FOUND)

    if original.author == request.user:
        return Response({'detail': 'Ви не можете пропонувати зміни до
власного рівня.'}, status=status.HTTP_400_BAD_REQUEST)

    proposed_data = request.data.get('data')
    comment = request.data.get('comment', '')

    if not proposed_data:
        return Response({'detail': 'Поле "data" є обов’язковим.'},
status=status.HTTP_400_BAD_REQUEST)

    LevelChangeRequest.objects.create(
        original_level=original,
        proposer=request.user,
        proposed_data=proposed_data,
        comment=comment
    )

    return Response({'detail': 'Зміни успішно запропоновано.'},
status=status.HTTP_201_CREATED)
```

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def view_change_requests(request):
    user = request.user
    # Знаходимо всі рівні, автором яких є поточний користувач
    user_levels = Level.objects.filter(author=user)

    # Отримуємо всі запити на зміну до цих рівнів
    changes =
LevelChangeRequest.objects.filter(original_level__in=user_levels)

    result = [{
        'id': change.id,
        'level_id': change.original_level.id,
        'level_title': change.original_level.title,
        'proposed_data': change.proposed_data,
        'status': change.status,
        'comment': change.comment,
        'proposer': change.proposer.username,
        'created_at': change.created_at
    } for change in changes]

    return Response(result)


@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_change_request_by_id(request, change_id):
    try:
        change = LevelChangeRequest.objects.get(id=change_id)
    except LevelChangeRequest.DoesNotExist:
        return Response({'detail': 'Запит на зміну не знайдено.'},
status=404)

    # Автор рівня або той, хто створив пропозицію, має бачити
    if change.original_level.author != request.user and change.proposer !=
request.user:
        return Response({'detail': 'У вас немає прав на перегляд цього
запиту.'}, status=403)

    result = {
        'id': change.id,
        'level_id': change.original_level.id,
        'level_title': change.original_level.title,
        'proposed_data': change.proposed_data,
        'status': change.status,
        'comment': change.comment,
        'proposer': change.proposer.username,
        'created_at': change.created_at
    }

    return Response(result)


@api_view(['POST'])
@permission_classes([IsAuthenticated])
def accept_change(request, level_id, change_id):
    try:
        level = Level.objects.get(id=level_id)
        change = LevelChangeRequest.objects.get(id=change_id,
original_level=level)
    except (Level.DoesNotExist, LevelChangeRequest.DoesNotExist):
```

```
        return Response({'detail': 'Рівень або запит не знайдено.'},
status=404)

    if level.author != request.user:
        return Response({'detail': 'Ви не маєте прав на зміну цього рівня.'},
status=403)

    level.data = change.proposed_data
    level.save()

    change.status = 'accepted'
    change.save()

    return Response({'detail': 'Зміни прийнято та застосовано.'})


@api_view(['POST'])
@permission_classes([IsAuthenticated])
def reject_change(request, level_id, change_id):
    try:
        level = Level.objects.get(id=level_id)
        change = LevelChangeRequest.objects.get(id=change_id,
original_level=level)
    except (Level.DoesNotExist, LevelChangeRequest.DoesNotExist):
        return Response({'detail': 'Рівень або запит не знайдено.'},
status=404)

    if level.author != request.user:
        return Response({'detail': 'Ви не маєте прав на зміну цього рівня.'},
status=403)

    change.status = 'rejected'
    change.save()

    return Response({'detail': 'Зміни було відхилено.'})


@api_view(['DELETE'])
@permission_classes([IsAuthenticated])
def delete_level(request, level_id):
    try:
        level = Level.objects.get(id=level_id)
    except Level.DoesNotExist:
        return Response({'detail': 'Рівень не знайдено.'},
status=status.HTTP_404_NOT_FOUND)

    if level.author != request.user:
        return Response({'detail': 'Ви не можете видалити цей рівень.'},
status=status.HTTP_403_FORBIDDEN)

    level.delete()
    return Response({'detail': 'Рівень успішно видалено.'},
status=status.HTTP_204_NO_CONTENT)


@api_view(['PATCH'])
@permission_classes([IsAuthenticated])
def edit_level(request, level_id):
    try:
        level = Level.objects.get(id=level_id)
    except Level.DoesNotExist:
        return Response({'detail': 'Рівень не знайдено.'}, status=404)

    if level.author != request.user:
```

```
        return Response({'detail': 'Ви не маєте прав редагувати цей
рівень.'}, status=403)

    data = request.data

    if 'title' in data:
        level.title = data['title']
    if 'description' in data:
        level.description = data['description']
    if 'data' in data:
        level.data = data['data']
    if 'is_public' in data:
        level.is_public = data['is_public']

    level.save()
    return Response({'detail': 'Рівень успішно оновлено.'})
```

Файл items_selector.py
Реалізація функціональної задачі роботи редактора рівнів

```
class Items_selector:
    def __init__(self):
        self.display_surface = pygame.display.get_surface()
        self.fill_items_selector_data()
        self.form_data()
        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_selector_buttons()

    def fill_items_selector_data(self):
        def menu_path(filename):
            return path.join(graphics_directory, 'menu', filename)

        items = [
            (0, None, None),
            (1, None, None),

            (2, 'terrain', 'soil.png'),
            (25, 'terrain', 'platform.png'),
            (3, 'terrain', 'water.png'),

            (4, 'treasure', 'golden_coin.png'),
            (5, 'treasure', 'silver_coin.png'),
            (6, 'treasure', 'diamond.png'),
            (33, 'treasure', 'heal_potion.png'),

            (7, 'enemy', 'spikes.png'),
            (8, 'enemy', 'pig.png'),
            (9, 'enemy', 'cannon_left.png'),
            (10, 'enemy', 'cannon_right.png'),
            (21, 'enemy', 'saw_bottom.png'),
            (22, 'enemy', 'saw_top.png'),
            (23, 'enemy', 'saw_left.png'),
            (24, 'enemy', 'saw_right.png'),
            (32, 'enemy', 'mace.png'),

            (11, 'palms fg', 'palm_small_fg.png'),
            (12, 'palms fg', 'palm_large_fg.png'),
            (13, 'palms fg', 'palm_left_fg.png'),
            (14, 'palms fg', 'palm_right_fg.png'),
            (19, 'bricks fg', 'bricks_fg.png'),
```

```
            (15, 'palms bg', 'palm_small_bg.png'),
            (16, 'palms bg', 'palm_large_bg.png'),
            (17, 'palms bg', 'palm_left_bg.png'),
            (18, 'palms bg', 'palm_right_bg.png'),
            (20, 'bricks bg', 'bricks_bg.png'),

            (26, 'info', 'info.png'),
            (27, 'info', 'finish_info.png'),
            (28, 'info', 'sign_bottom.png'),
            (29, 'info', 'sign_top.png'),
            (30, 'info', 'sign_left.png'),
            (31, 'info', 'sign_right.png'),
        ]

        self.items_selector_data = {
            idx: {
                'type': item_type,
                'preview': menu_path(preview) if preview else None
            } for idx, item_type, preview in items
        }

    def form_data(self):
        grouped = defaultdict(list)
        for key, data in self.items_selector_data.items():
            type_ = data.get('type')
            if type_ is not None:
                grouped[type_].append((key, load(data['preview'])))
        self.menu_surfs = dict(grouped)

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

        mm_size, mm_margin = sx(45), 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))

        sv_topleft = vector(mm_topleft) - (mm_size + mm_margin, 0)
        self.sv_rect = pygame.Rect(sv_topleft, (mm_size, mm_size))
        self.sv_image = load(path.join(script_directory, 'graphics', 'menus',
'save_button.png')).convert_alpha()
        self.sv_image = pygame.transform.scale(self.sv_image, (mm_size,
mm_size))

        panel_size = (270, 180)
        margin = 6
        panel_topleft = (self.window_size[0] - panel_size[0] - margin,
self.window_size[1] - panel_size[1] - margin)
        self.rect = pygame.Rect(panel_topleft, panel_size)

        button_size = (self.rect.width / 3, self.rect.height / 2)
        button_margin = 5

        def button_rect(col, row):
            rect = pygame.Rect(self.rect.topleft, button_size)
            rect = rect.move(col * button_size[0], row * button_size[1])
            return rect.inflate(-button_margin, -button_margin)

        self.items_selector = pygame.sprite.Group()
        button_defs = [
```

```
            (button_rect(0, 0), 'terrain'),
            (button_rect(1, 0), 'treasure'),
            (button_rect(1, 1), 'enemy'),
            (button_rect(0, 1), 'palms fg', 'palms bg'),
            (button_rect(2, 0), 'bricks fg', 'bricks bg'),
            (button_rect(2, 1), 'info'),
        ]

        for rect, main_key, *alt_key in button_defs:
            main_items = self.menu_surfs[main_key]
            alt_items = self.menu_surfs[alt_key[0]] if alt_key else None
            Items_selector_button(rect, self.items_selector, main_items,
alt_items, button_type=main_key)

    def selected_illumination(self, index):
        tile_type = self.items_selector_data[index].get('type')
        for button in self.items_selector:
            if button.button_type == tile_type or button.button_type ==
'palms fg' and tile_type == 'palms bg' \
                    or button.button_type == 'bricks fg' and tile_type ==
'bricks bg':
                pygame.draw.rect(self.display_surface, '#f5f1de',
button.rect.inflate(4, 4), 5, 4)
                break

    def on_click(self, mouse_pos, mouse_button):
        clicked_sprite = next((sprite for sprite in self.items_selector if
sprite.rect.collidepoint(mouse_pos)), None)
        if clicked_sprite:
            if mouse_button[1]:
                if clicked_sprite.items_alt:
                    clicked_sprite.main_active = not
clicked_sprite.main_active
            if mouse_button[2]:
                clicked_sprite.toggle()
            return clicked_sprite.get_item_id()
        if self.mm_rect.collidepoint(mouse_pos) and mouse_button[0]:
            pass



    def render(self, index):
        self.items_selector.update()
        self.items_selector.draw(self.display_surface)
        self.display_surface.blit(self.image, self.mm_rect.topleft)
        self.display_surface.blit(self.sv_image, self.sv_rect.topleft)
        self.selected_illumination(index)


class Items_selector_button(pygame.sprite.Sprite):
    def __init__(self, rect, group, items, items_alt=None, button_type=None):
        super().__init__(group)
        self.image = pygame.Surface(rect.size)
        self.rect = rect

        self.items_main = items
        self.items_alt = items_alt or []
        self.index = 0
        self.main_active = True

        self.button_type = button_type

    def _current_items(self):
        return self.items_main if self.main_active else self.items_alt
```

```
def get_item_id(self):
    current = self._current_items()
    if current:
        return current[self.index][0]
    return None

def update(self):
    self.image.fill('#33323d')
    current = self._current_items()
    if current:
        surf = current[self.index][1]
        rect = surf.get_rect(center=self.image.get_rect().center)
        self.image.blit(surf, rect)

def toggle(self):
    current = self._current_items()
    if current:
        self.index = (self.index + 1) % len(current)
```

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-1230.045480.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Данило ШОМАН

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є комп’ютерна гра у жанрі платформер з редактором рівнів та її поведінка на платформі Windows.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження та оновлення даних на сервері та локально;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу;

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- 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-1230.045480.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Данило ШОМАН

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

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

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

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).

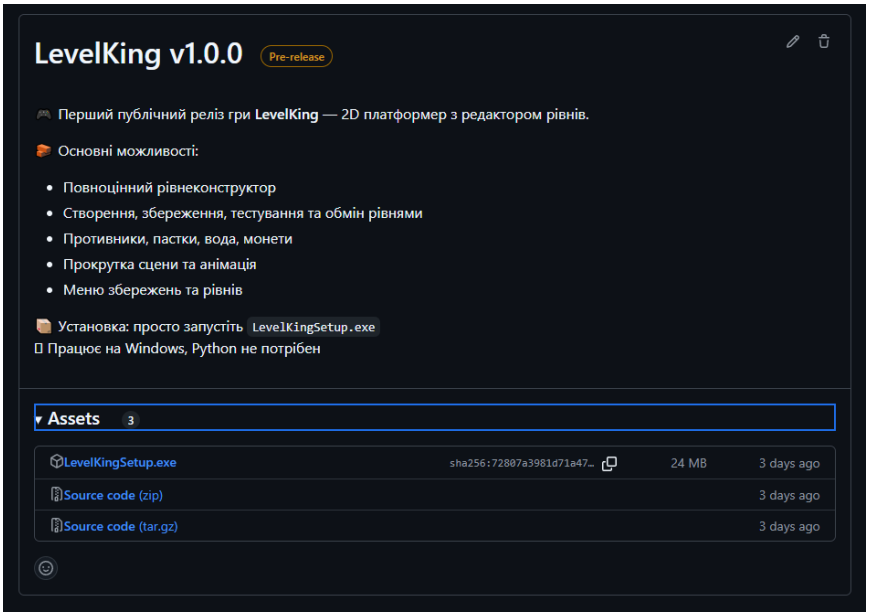


Рисунок 2.1 – Сторінка релізу

2.3 Перевірка коректної роботи

Після завершення встановлення гри, у вказаній директорії має з’явитися виконуваний файл `LevelKing.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.

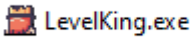


Рисунок 2.2 – Вміст встановленої директорії гри LevelKing у Windows

3 ВИКОНАННЯ ПРОГРАМИ

Після запуску гри користувач потрапляє у головне меню, з якого доступні наступні дії: перехід до редактора рівнів, вхід у серверну частину гри або вихід із гри (рис. 3.1).

Рисунок 3.1 – Головне меню гри

При виборі пункту «Editor», користувач переходить у редактор рівнів — інтуїтивно зрозуміле середовище для створення ігрових рівнів. У редакторі можна розміщувати елементи таких категорій, як: ландшафт, вороги, скарби, пастки, а також змінювати положення гравця чи горизонту (рис. 3.2).

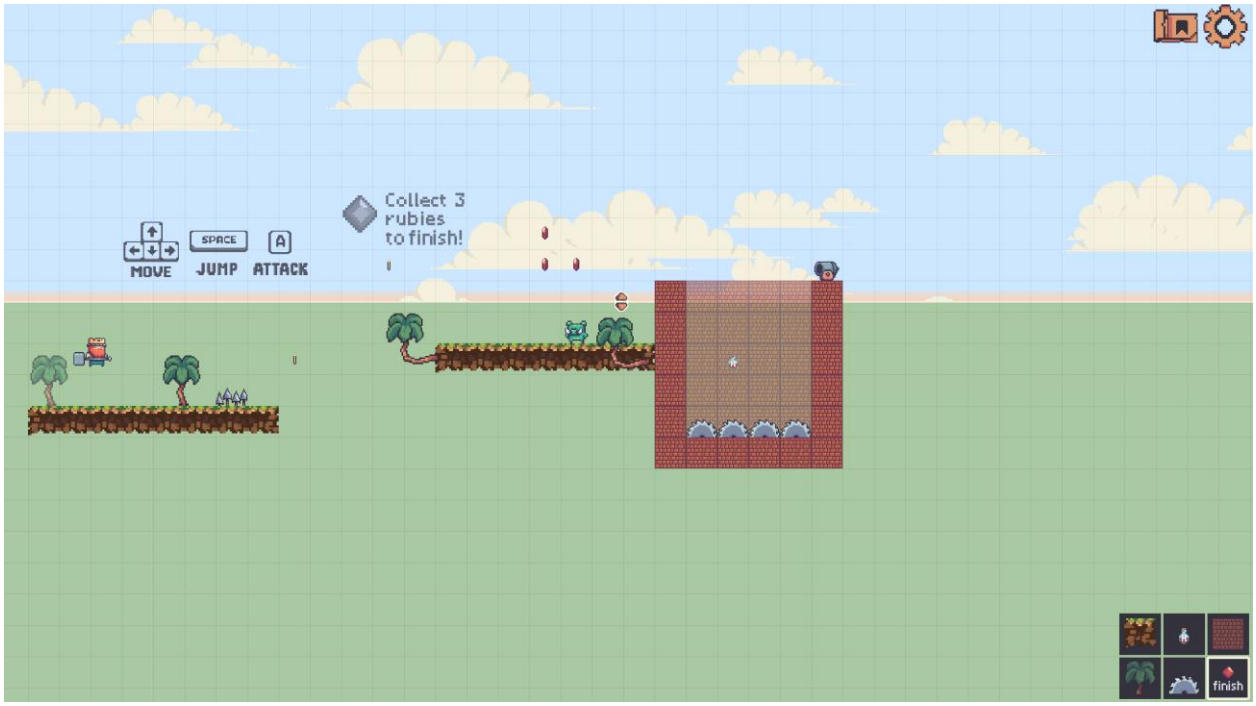


Рисунок 3.2 – Редактор рівнів

При виборі елементу головного меню з глобусом, користувач переходить у серверну частину гри, яка дозволяє увійти до облікового запису або перейти до реєстрації нового. Меню входу містить поля для логіну та паролю, а також кнопки «Увійти» та «Зареєструватися» (рис. 3.3).

Рисунок 3.3 – Меню входу

У разі реєстрації користувач вводить нові облікові дані. Поля ідентичні до меню входу (рис. 3.4).

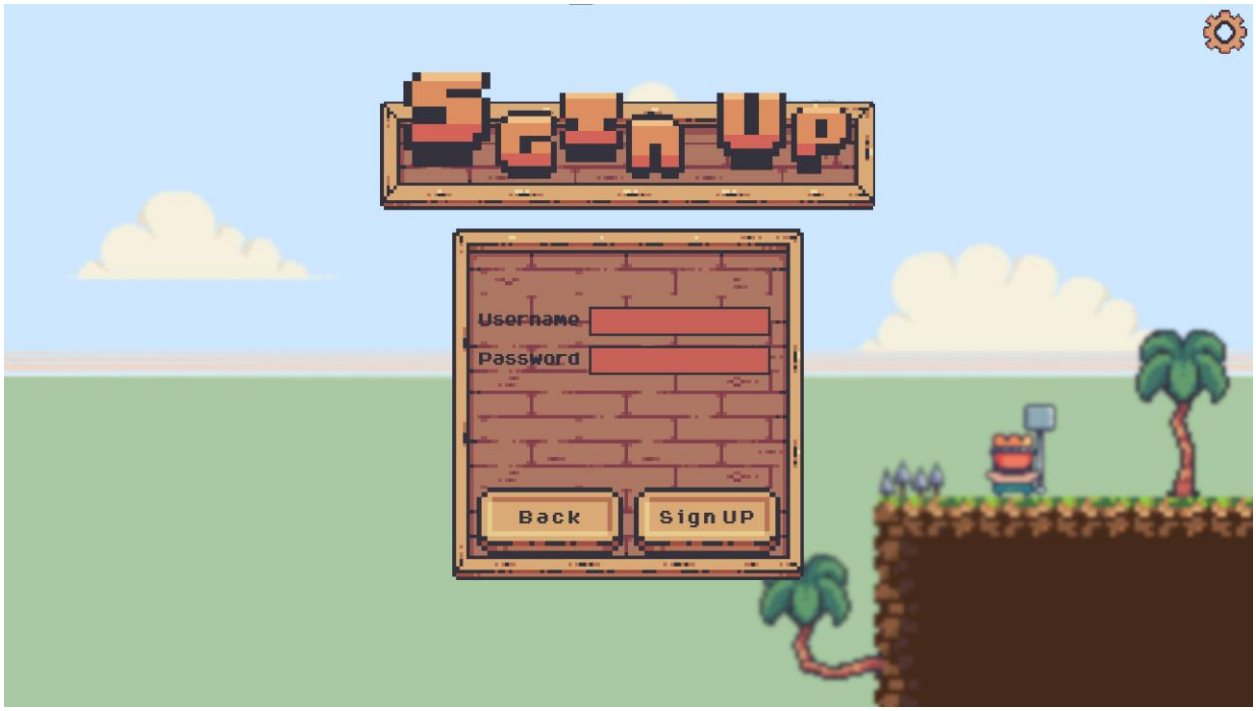


Рисунок 3.3 – Меню реєстрації

Після входу або реєстрації, користувач переходить до меню сервера. Звідси можна переглянути власні локальні або публічні рівні, перейти до їх меню, перейти до списку рівнів інших гравців, меню видалення акаунту або ж до меню пропозицій змін (рис. 3.4).

Рисунок 3.4 – Меню сервера

У меню локального рівня користувач може редагувати рівень або опублікувати його, вказавши назву та короткий опис (рис. 3.5).

Рисунок 3.5 – Меню локального рівня

У меню публічного рівня користувач може редагувати рівень, видалити його, змінювати назву та опис (рис. 3.6).

Рисунок 3.6 – Меню публічного рівня

Рівень після публікації стає доступним іншим користувачам у меню публічних рівнів (рис. 3.7).

Рисунок 3.7 – Меню публічних рівнів користувачів

Натиснувши на обраний рівень, користувачі переходять в меню цього рівня, звідки вони можуть спробувати рівень або перейти до меню пропозиції змін (рис. 3.8).

Рисунок 3.8 – Меню публічного рівня іншого користувача

В меню пропозиції змін, натиснувши кнопку «Редагувати», гравець переходить до редактора, вносить зміни до рівня, а потім може супроводити їх коментарем (рис. 3.8).

Рисунок 3.8 – Створення пропозиції змін

Автор рівня може бачити пропозиції в списку меню пропозицій (перехід з меню сервера) (рис. 3.9).

Рисунок 3.9 – Список пропозицій

Натиснувши на обрану пропозицію, користувач може тестувати змінену версію, після чого може прийняти або відхилити зміни (рис. 3.10).

Рисунок 3.10 – Перегляд та схвалення пропозиції

Також із меню сервера користувач має можливість повністю видалити свої дані. Перед цим відображається попередження із підтвердженням дії (рис. 3.11).

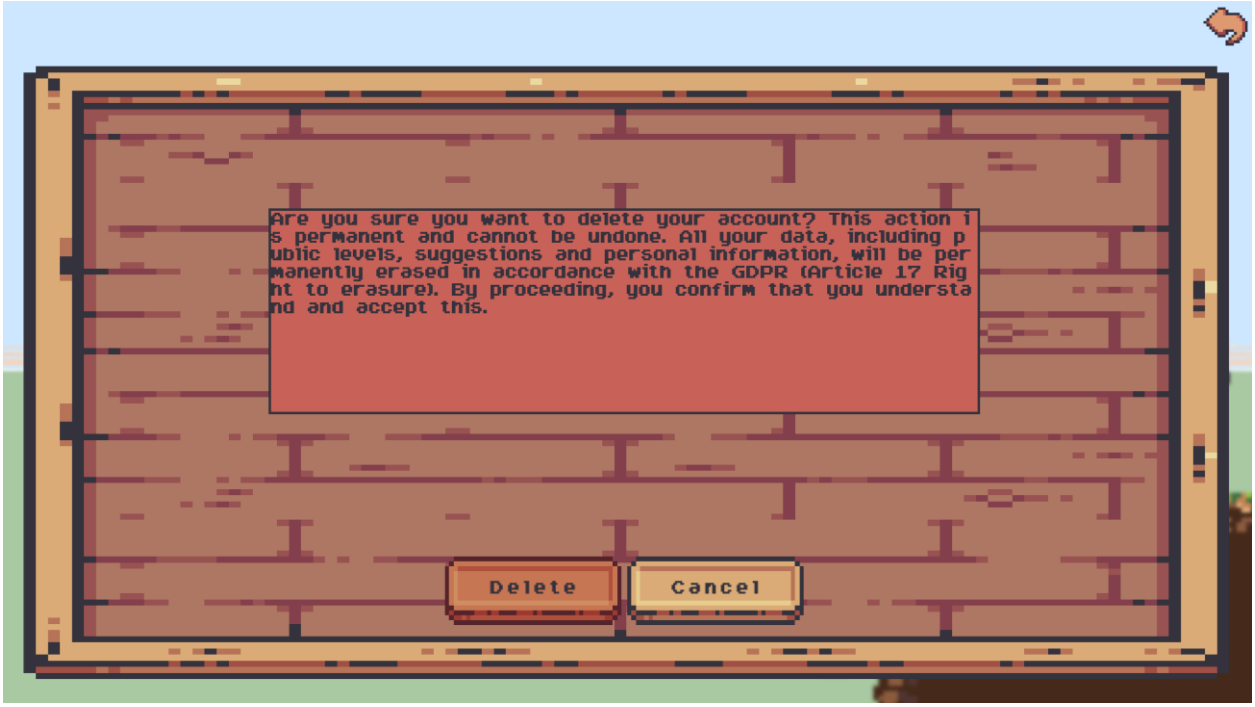


Рисунок 3.11 – Видалення даних користувача

Таким чином, «LevelKing» дозволяє не лише проходити рівні, а й активно брати участь у спільній творчості: створювати, обмінюватися, покращувати й обговорювати ігрові світи разом з іншими гравцями.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

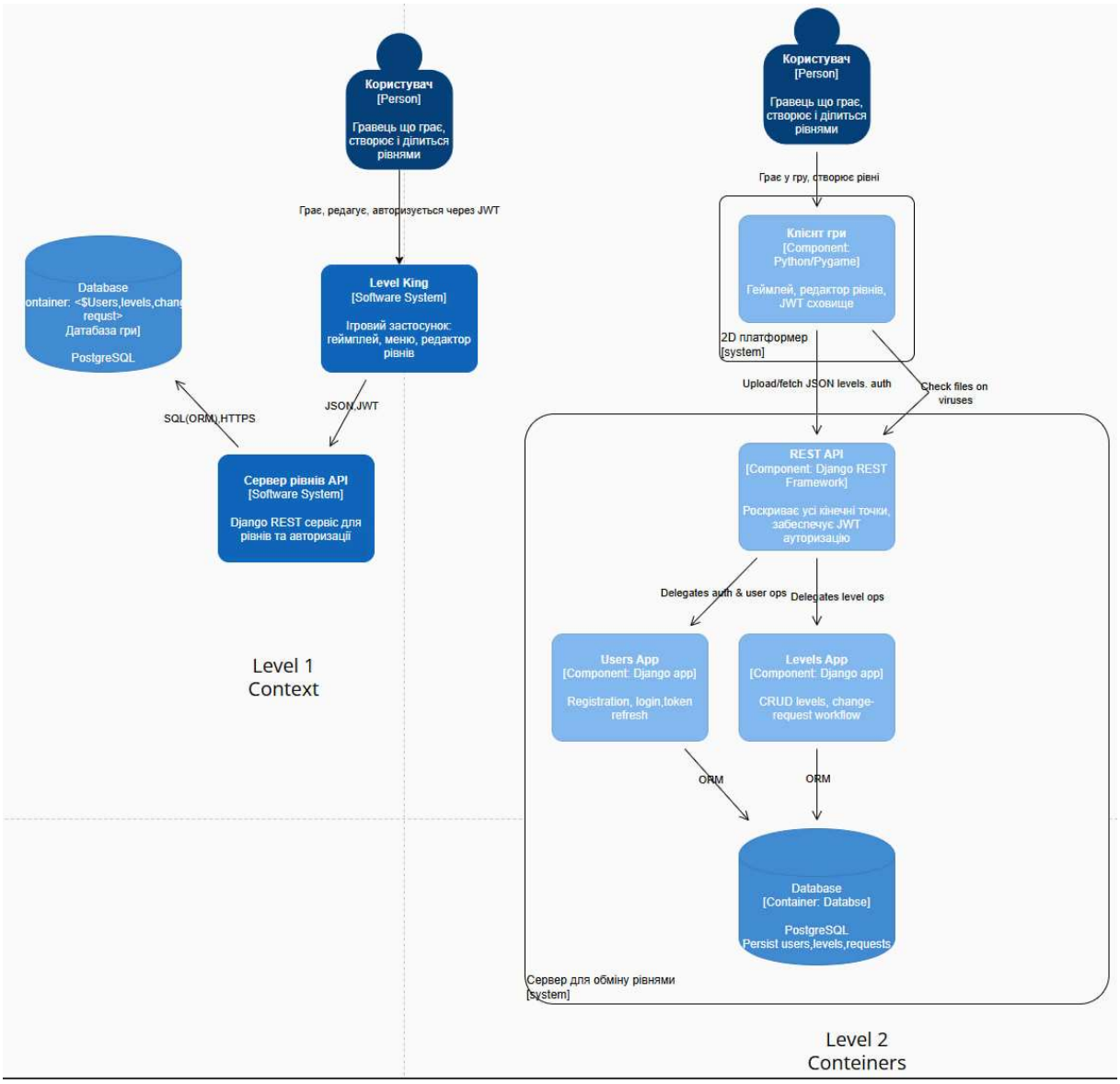
Класична гра у жанрі платформер, з розробкою редактора рівнів
(комплексна тема)
Графічний матеріал
КПІ.ІП-1230.045480.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Данило ШОМАН

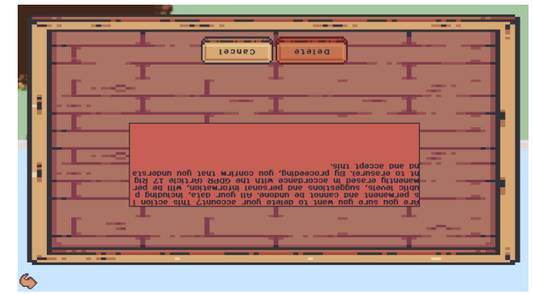
Київ – 2025

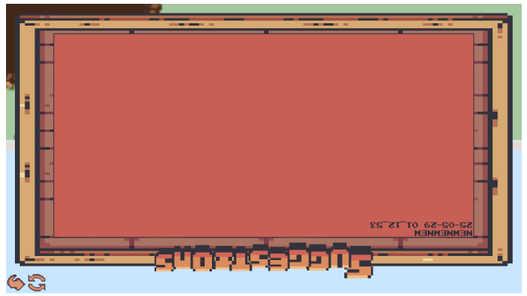


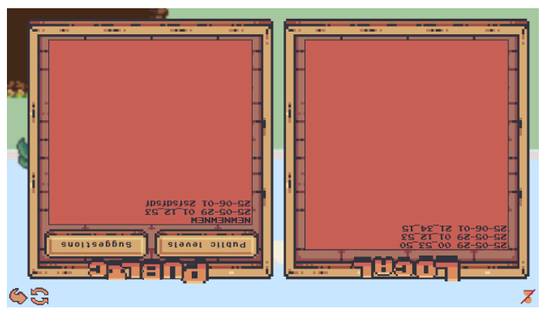
					КПІ.ІП-1230.045480.06.99.ССМ			
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна компонентів програмного забезпечення	Лит.	Маса	Масштаб
Розробив		Шоман Д.В.						
Перевірив		Дифучин А.Ю.				Аркуш 1	Аркушів 1	
Т. контр.								
					Класична гра у жанрі платформер, з розробкою редактора рівнів(комплексна тема)	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12		
Н. контр.		Головченко М.М.						
Затвердив		Жаріков Е.В.						

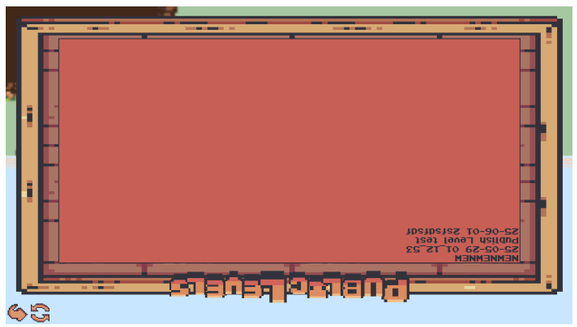
КПІ.ІП-1230.045480.06.99.СБД

					КПІ.ІП-1230.045480.06.99.СБД	
					Схема бази даних	Лит. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підп.	Дата		Аркуш 1 / Аркушів 1
Розробив		Шоман Д.В.				
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.				









Змн.	Арк.	№ докумекта	Підпис	Дата	Кресленна вигляду екранних форм	
Розробив		Шоман Д.В.				
Перевірив		Дифучин А.Ю.				
Т. контр.						Арк.ш 1 Аркушів 1
Н. контр.		Голованенко М.М.			Класична гра у жанрі платформемер, з розробкою редактора рівнів(комплексна тема)	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12
Затвердив		Жарікав Е.В.				

КПІ.ІП-1230.045480.06.99.КЕ