НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"___" _____ 20 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему
Програмне забезпечення для створення інтерактивної мультимедійної системи в двомірному середовищі
Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи

_____ Василюк-Зайцева С.В._
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _Савчук Андрій Андрійович_
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"___"_____ 20 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту
_____Савчуку Андрію Андрійовичу_____
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи Програмне забезпечення для створення інтерактивної мультимедійної системи в двомірному середовищі
затверджена наказом ректора НУБіП України від "16" грудня 2024р. №2249
Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи

Перелік питань, які потрібно розробити

- системний аналіз предметної області;
- проєктування інформаційного та програмного забезпечення;
- розробка інформаційного та програмного забезпечення;
- рекомендації щодо впровадження та експлуатації системи;

Дата видачі завдання "_____"_____20___р.
Керівник бакалаврської кваліфікаційної роботи
_____ _____ _Василюк-Зайцева С.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Савчук А.А.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

2D – двовимірний; використовується для позначення ігор, де об'єкти існують у двох вимірах: по горизонталі (X) та вертикалі (Y).

Unity – багатоплатформенне середовище розробки ігор, що використовується для створення гри.

C# – мова програмування, яка застосовується у середовищі Unity для написання сценаріїв (скриптів).

Tilemap – система у Unity, що дозволяє створювати мапи з тайлів (плиток), використовується для побудови рівнів гри.

Level Editor – інструмент у грі, що дозволяє користувачу самостійно створювати рівні.

Prefab – попередньо налаштований об'єкт у Unity, який можна багаторазово використовувати у грі.

Collider – компонент у Unity, який дозволяє визначати зіткнення об’єктів у грі.

Script – скрипт, тобто програмний код, який визначає поведінку об’єктів у грі.

UI (User Interface) – інтерфейс користувача, елементи якого дозволяють взаємодіяти з грою (меню, кнопки тощо).

JSON (JavaScript Object Notation) – формат зберігання даних, який використовується для серіалізації та збереження.

Save System – система збереження, що забезпечує запис створених рівнів на диск користувача.

Sprite (спрайт) – графічне зображення, що представляє об'єкт у 2D-грі; зазвичай є частиною анімації або візуального оформлення персонажів, предметів тощо.

Sprite Atlas (атлас спрайтів) - колекція окремих спрайтів, об'єднаних в одну текстуру для оптимізації продуктивності гри.

Rigidbody2D - компонент фізичного рушія Unity для 2D об'єктів, що додає реалістичну фізичну поведінку.

Animation Controller - система керування анімаціями в Unity, що дозволяє створювати переходи між різними анімаційними станами.

Particle System - система частинок у Unity для створення візуальних ефектів.

Canvas - основний компонент UI системи Unity для розміщення елементів інтерфейсу.

Cinemachine - інструмент Unity для керування камерою та створення кінематографічних ефектів.

Physics2D - фізичний рушій Unity для двовимірних ігор.

Scene - сцена в Unity, що представляє окремий рівень або частину гри.

PlayerPrefs - система Unity для збереження налаштувань гравця між сеансами гри.

Coroutine - механізм у Unity для виконання асинхронних операцій.

IDE – Integrated Development Environment – інтегроване середовище розробки (наприклад, Visual Studio)

API – Application Programming Interface – інтерфейс прикладного програмування

Trigger Collider – тип колайдера, який викликає події зіткнення, але не має фізичного впливу

MonoBehaviour – базовий клас Unity, від якого успадковуються всі компоненти на сцені

Animator – компонент Unity, що відтворює анімації на основі Animator Controller

Time.deltaTime – час, що минув між кадрами (використовується для незалежного від FPS руху)

Raycast – метод Unity, що дозволяє визначити, на що "дивиться" промінь з точки у просторі (зазвичай від миші)

JsonUtility – клас Unity для перетворення об’єктів у JSON і навпаки

System.IO – простір імен .NET для читання/запису файлів

SaveManager – реалізований у проєкті компонент для збереження рівнів (назва кастомна, але використовується)

Content Size Fitter – UI-компонент, який автоматично підлаштовує розмір елементів під вміст

Horizontal/Vertical Layout Group – UI-компоненти Unity для автоматичного розташування елементів по горизонталі/вертикалі

Canvas Scaler – компонент, який адаптує UI до розміру екрану

Layer-based Collision Matrix – система Unity для налаштування того, які об’єкти можуть зіштовхуватись

Interpolate (Rigidbody2D) – параметр, який згладжує рух об’єкта між кадрами

Trigger – подія в колайдері, що активується при вході іншого об’єкта (OnTriggerEnter2D тощо)

HUD – Heads-Up Display – накладений інтерфейс з інформацією (здоров’я, очки тощо)

Scroll View – компонент Unity для прокрутки вмісту

Canvas (режим Screen Space - Overlay) – режим рендерингу UI у Unity

OnClick – метод обробки події натискання кнопки у Unity

ВСТУП

У сучасному інформаційному просторі особливу увагу привертають мультимедійні програмні засоби, які забезпечують взаємодію користувача з цифровим середовищем через графічні, звукові та інтерактивні компоненти. Активний розвиток 2D-технологій та доступних інструментів для створення комп'ютерних ігор відкриває нові можливості для користувачів, які прагнуть самостійно наповнювати середовище комп'ютерної гри різноманітними об'єктами. Такий підхід дає змогу не лише реалізовувати творчі задуми, а й знайомитися з базовими принципами побудови логіки, структури рівнів та інтерактивної взаємодії. Серед найбільш популярних рішень — прості у використанні редактори, що дозволяють розміщувати на сцені ігрового рівня перешкоди, зони фінішу, корисні елементи для ігрового персонажу та плитки, які формують геометрію рівня і слугують основою для пересування ігрового персонажа.

Мета дипломної роботи «Програмне забезпечення для створення інтерактивної мультимедійної системи в двомірному середовищі» полягає у створенні програмного засобу, який дозволяє користувачам створювати власні 2D-рівні з набором об'єктів комп'ютерної гри, зберігати ці рівні у внутрішній галереї, а також запускати їх у режимі гри. Реалізований функціонал дає змогу візуально формувати ігровий простір, розміщуючи на ньому перешкоди для ігрового персонажу у вигляді шипів, корисні елементи такі як зілля здоров'я, плитки-платформи, що формують ігрове поле, а також елементи завершення рівня у вигляді фінішних зон. При цьому плитки є функціональними складовими рівня, на яких може пересуватись ігровий персонаж.

У процесі реалізації системи передбачається також створення інтерфейсу для збереження, редагування та видалення створених рівнів. Користувач має змогу запускати рівень у режимі проходження, де керує персонажем, долаючи розміщені перешкоди. Якщо ігровий персонаж гравця досягає фінішної зони, то рівень вважається пройденим. Якщо ж гравець виходить з рівня до завершення,

його можна пізніше повторно відкрити та пройти. Подібні розробки мають не лише розважальне, але й освітнє значення, адже вони можуть використовуватись у процесі навчання для ознайомлення з основами геймдизайну, логіки взаємодії об’єктів та структурного мислення. Такі системи сприяють розвитку уяви, просторового мислення та навичок планування.

Для реалізації проєкту було використано ігровий рушій Unity з акцентом на його 2D-функціональність, мову програмування C#, систему серіалізації даних у форматі JSON та середовище розробки Visual Studio. Unity 2D надає потужний набір інструментів для створення тайлових карт, спрайтових анімацій, обробки колізій та взаємодії об’єктів у двовимірному середовищі. Система збереження рівнів реалізована через JSON, що дозволяє зручно серіалізувати й десеріалізувати інформацію про об’єкти на рівні. У межах користувацького інтерфейсу було застосовано систему Unity UI, яка включає Canvas, Layout Group, Button, TextMeshPro та інші компоненти для створення адаптивного інтерфейсу. Поведінка ігрового персонажа реалізована через фізичний рушій Physics2D з використанням Rigidbody2D, BoxCollider2D, Trigger Collider, а також механізмів інтерполяції та перевірки зіткнень. Увесь ігровий процес програмувався на C# з використанням об’єктно-орієнтованого підходу, інкапсуляції логіки в окремі компоненти MonoBehaviour, асинхронних операцій через корутини та делегатів для подій взаємодії з об’єктами.

Результати розробки програмного забезпечення були представлені у вигляді тез, присвячені темі «Програмне забезпечення для створення інтерактивної мультимедійної системи в двомірному середовищі» були опубліковані на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем '2025» [20].

У межах кваліфікаційної роботи представлено чотири розділи, вступ, висновки, додатки та списки використаних джерел. У першому розділі висвітлено аналіз предметної області та постановку задачі, другий присвячено проєктуванню інформаційного забезпечення, третій описує процес розробки

програмного забезпечення, а четвертий містить рекомендації щодо тестування та впровадження системи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Основна мета програмного забезпечення для створення інтерактивної мультимедійної системи в двомірному середовищі, полягає в наданні користувачу можливості швидко формувати повноцінну ігрову сцену з платформами, ігровим персонажем, супротивниками, перешкодами, підсилювачами, а також зоною завершення рівня.

Серед ключових завдань, які має виконувати програмне забезпечення, можна виділити наступне:

Головне меню, стилізоване для зручної взаємодії з користувачем, що містить доступ до основних модулів: створення рівня, перегляду галереї рівнів, налаштувань, інструкції, продовження гри та виходу з програми.

Редактор рівнів, який надає користувачу прості інструменти для розміщення об’єктів на сцені. До основних компонентів відносяться:

Платформи — елементи, по яких пересувається персонаж та знаходяться інші об’єкти;

Ігровий персонаж — об’єкт, яким керує користувач;

Вороги — об’єкти, що взаємодіють з персонажем і становлять для нього загрозу;

Перешкоди — статичні об’єкти, зіткнення з якими завдає шкоди;

Підсилювачі — корисні об’єкти, що допомагають ігровому персонажу;

Фінішна зона — ціль рівня, досягнення якої завершує його проходження.

Система збереження, Кожен створений рівень може бути збережений у форматі **JSON**, що дозволяє фіксувати параметри об’єктів, їхнє розташування та логіку взаємодії, з подальшим редагуванням або проходженням.

Галерея рівнів — модуль, у якому користувач може:

запускати збережені рівні у режимі гри;

редагувати створені рівні;

видаляти створені рівні.

Режим проходження гри, де користувач керує персонажем і взаємодіє з об’єктами сцени. У разі програшу, є можливість завершити рівень і згодом повернутися до нього або почати заново рівень.

Модуль налаштувань, що дає змогу змінювати параметри:

гучність музики;

керування персонажем;

зберігати обрані параметри.

Інструкція користувача, яка містить базову інформацію щодо використання редактора рівнів та основи взаємодії з програмою.

Функція виходу з програми, дозволяє завершити поточну головне меню.

1.2 Опис предметної області

Предметна область дипломної роботи стосується розробки та дослідження інтерактивних мультимедійних систем у двовимірному середовищі з фокусом на редактори рівнів для комп'ютерних ігор. Ця галузь динамічно розвивається в останні роки, оскільки користувачі прагнуть не лише споживати готовий контент, але й створювати власний.

Двовимірні комп'ютерні ігри з можливістю редагування рівнів являють собою важливий сегмент індустрії розваг та освіти [4]. Вони поєднують елементи візуального дизайну, інтерактивної взаємодії та базової логіки програмування в доступному для широкої аудиторії форматі. Редактори рівнів надають користувачам інструменти для розміщення ігрових об'єктів, налаштування їхніх параметрів та створення унікальних сценаріїв проходження.

Типовий редактор рівнів для 2D-гри включає функціонал для управління геометрією ігрового простору (платформи, перешкоди), розміщення інтерактивних елементів (вороги, підсилювачі, ігрові механіки) та визначення умов завершення рівня. Особливу роль відіграють системи збереження та

завантаження створеного контенту, що дозволяють зберігати результати творчої роботи та ділитися ними.

З технічної точки зору, такі системи реалізуються за допомогою ігрових рушіїв, які надають базову інфраструктуру для візуалізації, обробки колізій, анімації та інших аспектів інтерактивного середовища[1, с. 78]. Окрім розважального аспекту, ці системи мають значний освітній потенціал, сприяючи розвитку алгоритмічного мислення, просторової уяви та базових навичок дизайну.

Розробка програмного забезпечення в даній області потребує врахування принципів інтуїтивного інтерфейсу користувача, оптимізації продуктивності та забезпечення консистентної взаємодії між різними компонентами системи. Важливим аспектом є також баланс між простотою використання для непрофесійних користувачів та функціональною гнучкістю, необхідною для реалізації різноманітних творчих задумів.

Серед найбільш відомих прикладів подібних систем можна виділити редактор рівнів гри Super Mario Maker від Nintendo, який надає користувачам інтуїтивно зрозумілий інтерфейс для побудови власних рівнів у класичному платформері. Інша популярна система — Tiled Map Editor, що використовується для створення тайлових карт у 2D-іграх з підтримкою різних рушіїв, зокрема Unity. Також варто згадати Celeste Map Editor — внутрішній редактор гри Celeste, що дозволяє модифікувати ігрові локації. Порівняно з наведеними прикладами, розроблена у межах дипломної роботи система вирізняється поєднанням простого інтерфейсу з підтримкою розширеної функціональності, зокрема бойових елементів, індикаторів здоров’я, підтримки серіалізації у форматі JSON та можливості інтеграції користувацьких рівнів без зовнішніх інструментів. Основний акцент зроблено на інтеграції створення та проходження рівнів у межах єдиного середовища, що робить систему зручною як для гравців, так і для початківців-розробників.

1.3 Аналіз вимог до програмної системи

Розроблювана система представляє собою двовимірну платформер-гру з елементами бойової механіки та редактором рівнів. Гра реалізована на ігровому рушії Unity з використанням мови програмування C# [2].

Основною особливістю системи є поєднання класичного платформера з можливістю створення користувацького контенту через вбудований редактор рівнів. При створенні бойової системи враховувались принципи рефакторингу, описані у [3, с. 81]. Гравці можуть не лише проходити заздалегідь створені рівні, але й розробляти власні, використовуючи інтуїтивно зрозумілий інтерфейс редактора.

Ігровий процес побудований на класичних механіках платформера: переміщення, стрибки та бойова система. Гравець керує персонажем, який може рухатися вліво та вправо, виконувати стрибки та атакувати ворогів. Система здоров'я відстежує стан персонажа та реагує на отримані пошкодження.

Вороги в грі представлені статичними об'єктами, які завдають шкоди при контакті з гравцем. Це створює додатковий рівень складності для проходження рівнів, вимагаючи від гравця уважного планування маршруту та точності рухів для уникнення зіткнень.

Редактор рівнів надає користувачам наступні можливості:

Розміщення платформ та інших ігрових об'єктів

Встановлення точки появи гравця

Розміщення ворогів

Збереження створених рівнів

Система збереження використовує формат JSON для зберігання даних про рівні та прогрес гравця. Це забезпечує легкість модифікації та переносимість даних.

Користувацький інтерфейс розроблений з урахуванням принципів інтуїтивності та зручності використання.

1.4 Моделювання предметної області

Діаграма прецедентів є одним із ключових інструментів моделювання предметної області, що дозволяє візуалізувати зовнішню поведінку програмного забезпечення через дії користувача. У контексті даної дипломної роботи діаграма зосереджена на основному учаснику — гравцеві, який взаємодіє із системою через інтерфейс гри та редактор рівнів. Вона демонструє п’ять головних сценаріїв, які реалізовані у вигляді окремих прецедентів.

Першим і основним прецедентом є «Проходження створених рівнів». Це функція, що передбачає керування ігровим персонажем, взаємодію з об’єктами сцени, уникнення перешкод, досягнення цілі рівня та загалом реалізує ігровий процес. Вона охоплює більшу частину логіки гри, включаючи обробку зіткнень, реакцію на ворогів та підсилювачі.

Другим є прецедент «Створення / Редагування рівнів», який надає гравцеві можливість виступати в ролі розробника рівнів. У цьому режимі користувач за допомогою візуального редактора може створювати нові ігрові сцени або редагувати вже створені їм рівні. Редактор дозволяє розміщувати ворогів з якими ігровий персонаж буде взаємодіяти, об’єкти які дозволять ігровому персонажу поповнювати його здоров’я, сам ігровий персонаж який буде виступати першою точкою для початку проходження рівня, кінцева точка для завершення рівня та платформи на яких буде построєний рівень.

Третім важливим елементом є «Збереження створених рівнів» — функція, що дозволяє зафіксувати поточний стан рівня у форматі JSON. Це забезпечує можливість зберігати структуру сцени, розташування об’єктів та їхні характеристики, що необхідно для подальшого завантаження або редагування.

Прецедент «Завантаження створених рівнів» реалізує функціональність імпорту раніше збережених даних. Завдяки цьому користувач може відновити доступ до своїх проєктів, переглянути, видалити або запустити у грі.

П’ятий прецедент — «Зміна налаштувань звуку та управління» — надає змогу персоналізувати досвід взаємодії з програмою, зокрема через регулювання гучності, зміні управління.

Діаграма наведена у рисунку 1.1

Рис. 1.1 Діаграма прецедентів

Діаграма послідовності ілюструє етапи взаємодії ключових учасників процесу під час створення та використання програмного продукту, розробленого для побудови та проходження користувацьких 2D-рівнів. У діаграмі відображено п’ять основних об’єктів: розробник, інтерфейс розробника, ігровий рушій, гра та гравець. Вони взаємодіють між собою у визначеній послідовності відповідно до свого функціонального призначення.

На першому етапі розробник працює з інтерфейсом розробника для створення програмного середовища. У цей момент він не формує рівні напряму, а створює необхідну інфраструктуру — графічні елементи, об’єкти, платформи, ворогів, підсилювачі, ігрового персонажа, ворогів ігрового персонажа, а також програмну логіку у вигляді скриптів. Усі ці компоненти згодом

використовуватимуться гравцем у редакторі рівнів. Після внесення змін до проєкту відбувається збереження — інтерфейс надсилає відповідний запит до рушія, а у відповідь отримує підтвердження про успішне збереження.

На наступному етапі розробник проводить тестування функціональності створеної програми. У разі виявлення помилок він виконує повторне редагування коду або об’єктів та ініціює повторне збереження. Після усунення недоліків запускається компіляція проєкту, результатом якої є створення повноцінного ігрового застосунку.

Після чого гравець, який взаємодіє з грою через доступний інтерфейс надсилається запит на завантаження гри та її компонентів. Після запуску користувач може проходити рівні, створені ним раніше, або редагувати існуючі. Ігровий процес включає активну взаємодію з елементами сцени, збереження прогресу та завершення сесії. Усі ці дії опрацьовуються рушієм у режимі реального часу.

Діаграму наведено на рисунку 1.2

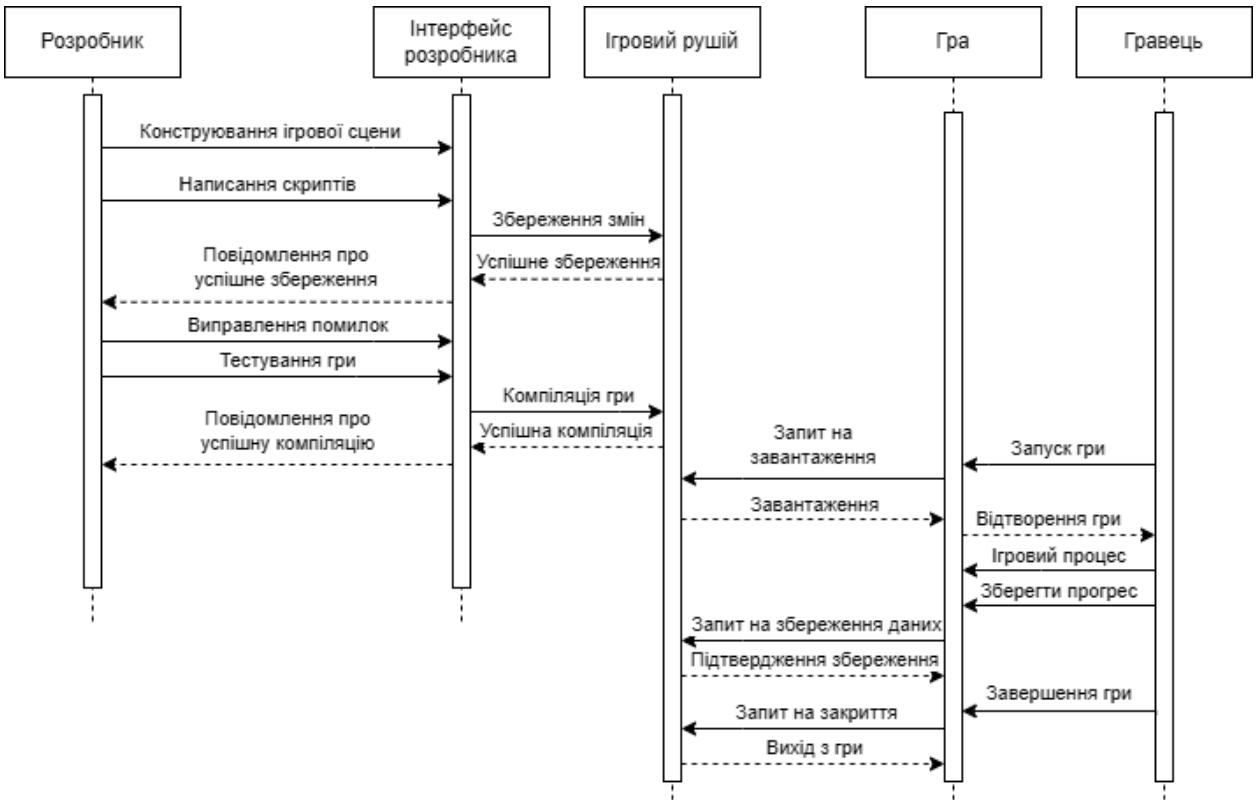

Рис. 1.2 Діаграма послідовності

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Логічна модель даних, представлена на діаграмі, формалізує структуру інформаційного забезпечення ігрової системи та демонструє взаємозв'язки між основними сутностями, необхідними для зберігання та обробки даних у грі. Кожна сутність виконує свою роль у моделюванні ігрового середовища, дозволяючи централізовано керувати налаштуваннями рушія, організацією рівнів, персонажами, ворогами та об'єктами.

Центральною частиною є сутність GAME_ENGINE, яка відповідає за загальну координацію роботи всієї системи. Вона зберігає базову інформацію про версію рушія, активний рівень та набір основних налаштувань, на які спирається виконання гри. Встановлення активного рівня через поле active_level дозволяє динамічно переключати контекст гри, а зв'язок з параметрами через core_settings забезпечує гнучкість у конфігурації.

Налаштування самого рушія зберігаються в сутності ENGINE_SETTINGS, яка пов'язана з GAME_ENGINE відношенням один до одного. Тут містяться критичні параметри: тип *API* візуалізації, фізичний рушій, максимальна кількість кадрів на секунду та логічний параметр debug_mode. Це дозволяє адаптувати роботу рушія до різних технічних умов і режимів розробки.

Кожен LEVEL представляє собою окрему ігрову сцену. Він зберігає унікальний ідентифікатор, назву, а також пов'язаний з рушієм через зовнішній ключ. Крім цього, рівень включає посилання на ігрового персонажа, набір ворогів і об'єктів, що дозволяє зручно згрупувати всі сутності, що належать до конкретного середовища. Таким чином, кожен рівень є самостійним контейнером для всієї внутрішньої активності гри.

Персонажі моделюються через сутність GAME_CHARACTER, яка належить до певного рівня. Кожен рівень передбачає наявність одного активного персонажа. Цей об'єкт містить інформацію про рівень здоров'я та початкову

позицію, що дозволяє відображати та керувати героєм у межах рівня. Такий підхід підтримує концепцію одиночної гри з фокусом на індивідуальну взаємодію гравця з середовищем.

Вороги представлені сутністю ENEMY, яка також належить до конкретного рівня, але у відношенні "один до багатьох". Це означає, що один рівень може містити декілька ворогів. У структурі ворога зберігається його тип та сила атаки. При цьому реалізація ворога спрощена — він не має складної поведінки чи анімацій, але завдає шкоди персонажу при зіткненні, що повністю відповідає потребам цієї гри.

Для реалізації взаємодії гравця з навколишнім середовищем передбачена сутність GAME_OBJECT, яка також має відношення "один до багатьох" з LEVEL. Об’єкти в грі, зокрема пастки чи предмети, описуються через тип і ефект. У поточному проєкті реалізовано лише предмети типу "зілля здоров’я", які при взаємодії з персонажем відновлюють його здоров’я. Ефекти описуються у вигляді JSON-структур, що дозволяє зберігати параметри у гнучкому форматі для розширення в майбутньому [8]. Діаграму наведено на рисунку 2.1

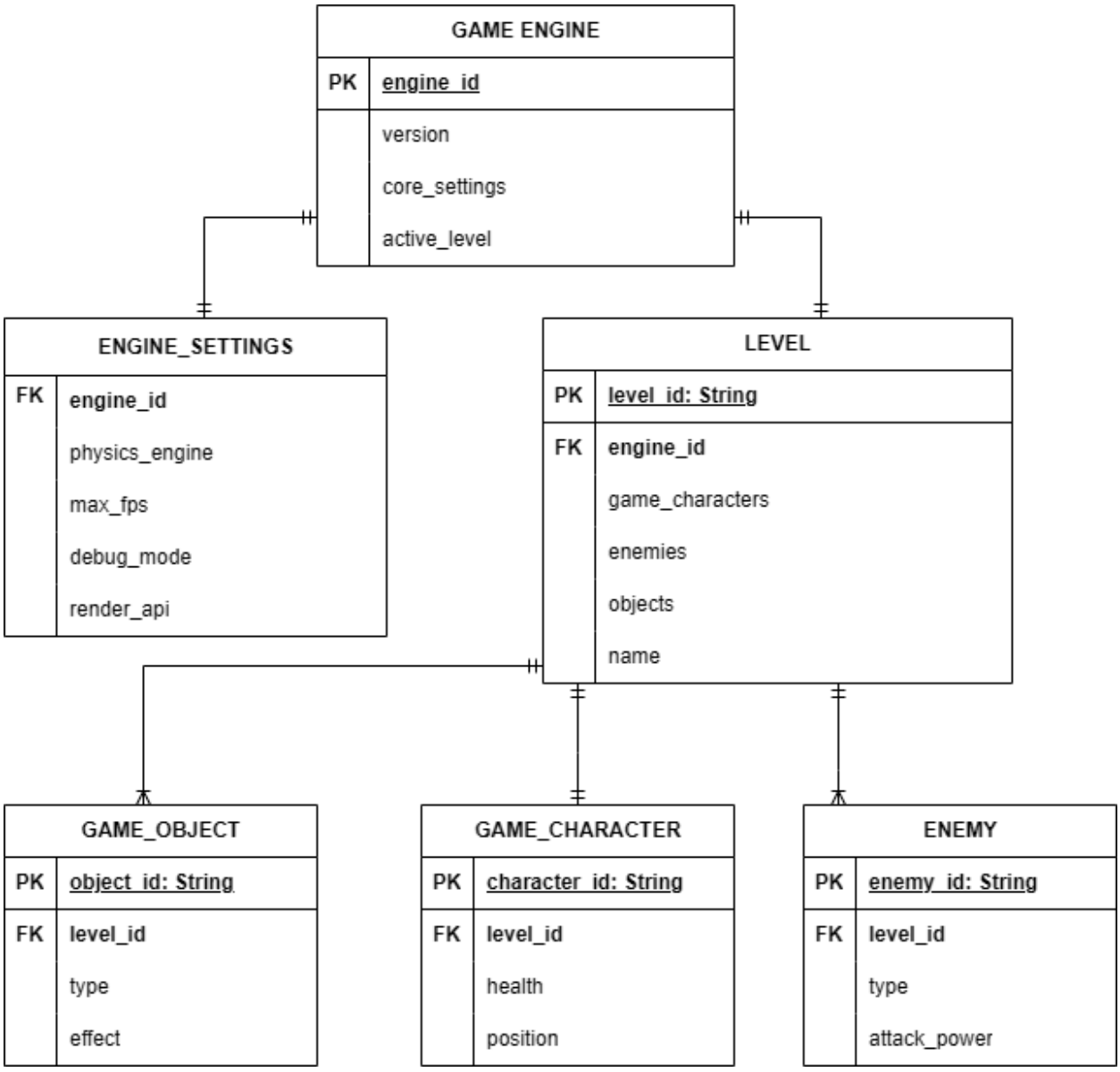

Рис. 2.1 Логічна модель даних

2.2 Діаграма абстракцій

У наведеній схемі представлені основні абстракції, що використовуються для опису об’єктів у межах ігрового середовища. Кожна з них має набір властивостей, які визначають її стан, та обов’язки, що відображають її призначення у загальній логіці ігрового процесу. Дану діаграму наведено на рисунку 2.2

Загалом, представлена система абстракцій охоплює основні типи об’єктів, що зустрічаються у грі, і визначає їхню взаємодію у межах рівня.

Ігровий персонаж є центральним елементом взаємодії користувача з грою [10, с. 95]. До основних властивостей належать рівень здоров'я, швидкість руху, початкова позиція та сила атаки. Основні обов'язки включають переміщення у межах рівня, взаємодію з іншими об'єктами та можливість атакувати. Принципи побудови шаблонів поведінки для об'єктів частково базуються на підходах із [7, с. 112]. Ігровий персонаж може змінювати своє положення в просторі та реагувати на різні ситуації, що виникають під час проходження рівня.

Ворог представлений як об'єкт, що має рівень здоров'я, початкову позицію, силу атаки та унікальну назву. Його обов'язком є завдання шкоди при взаємодії з ігровим персонажем. Ворог не має активної поведінки чи пересування, а лише виконує функцію небезпеки, з якою гравець стикається під час проходження рівня. Таким чином, ворог виступає елементом, що створює загрозу, з якою необхідно впоратись.

Пастка є об'єктом оточення, що автоматично завдає шкоди при активації. До її властивостей належать початкова позиція, тип, кількість шкоди та назва. Вона виконує роль перешкоди, що ускладнює проходження рівня, не потребуючи додаткової взаємодії з боку гравця. Пастка спрацьовує за умов потрапляння у її зону дії, завдаючи миттєвий ефект.

Предмет реалізований як допоміжний об'єкт, який можна використати під час проходження рівня. Він має просту структуру, що включає тип та назву, а його основною функцією є накладання ефекту на гравця. Предмет виконує роль зілля здоров'я, що дозволяє частково відновити життєві показники персонажа при підборі.

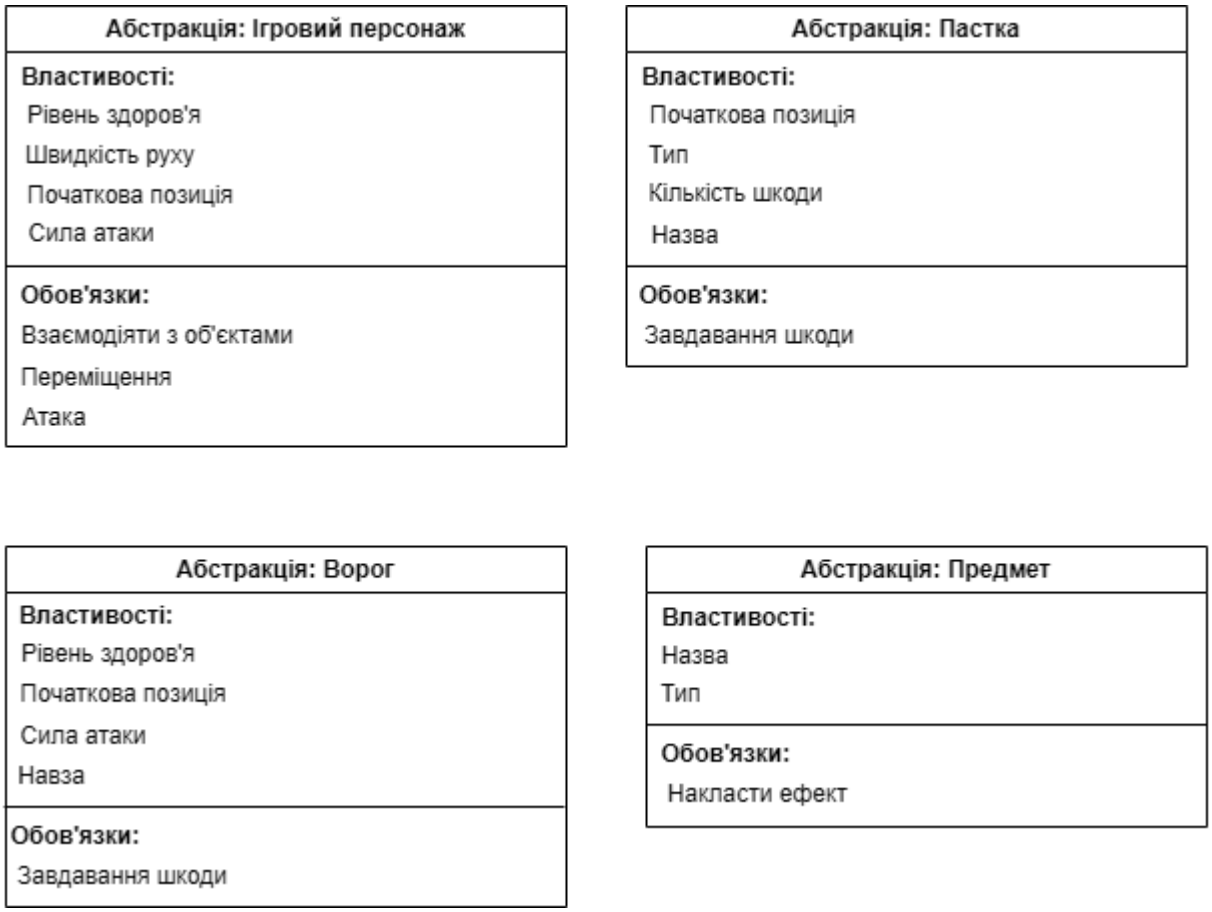

Рис. 2.2 Діаграма абстракцій

2.3 Діаграма компонентів

Діаграма компонентів, демонструє архітектурну структуру програмного забезпечення, орієнтованого на створення та використання рівнів у рамках ігрового середовища. Система поділена на три основні підсистеми: головне меню, управління файлами та сцена. Опис архітектури системи враховує основні поняття проектування програмного забезпечення, представлені у [15, с. 322].

Підсистема головного меню містить два ключові компоненти: «Редактор рівнів» і «Галерея рівнів». Компонент «Редактор рівнів» надає користувачу можливість створювати нові рівні та редагувати існуючі. У процесі редагування можна додавати різні об'єкти, розміщувати платформи. Після завершення редагування рівень передається до підсистеми управління файлами, де за допомогою компонента «Запис даних» зберігається у форматі JSON.

Компонент «Галерея рівнів» призначений для перегляду збережених рівнів і надає користувачу три основні функції: запуск рівня для проходження, відкриття рівня у редакторі для його подальшої модифікації, а також видалення рівня. У разі запуску або редагування ініціюється завантаження відповідної сцени.

Підсистема сцени містить компонент «Стан рівня», який відповідає за відтворення рівня в ігровому середовищі. У режимі проходження гравець взаємодіє з елементами рівня через ігрового персонажа. У режимі редагування або створеня— сцена працює у спеціальному режимі, що дозволяє змінювати складові рівня безпосередньо у візуальному просторі.

Підсистема управління файлами реалізує механізм збереження та завантаження даних. Компонент «Запис даних» відповідає за серіалізацію рівня у форматі JSON та його збереження у файловій системі. Компонент «Завантаження» зчитує збережені рівні з пам’яті, після чого вони стають доступними для перегляду в галереї або для використання у сцені. Метрики програмного забезпечення дозволяють об'єктивно оцінити якість гри [9, с. 528].

Діаграму наведено на рисунку 2.3

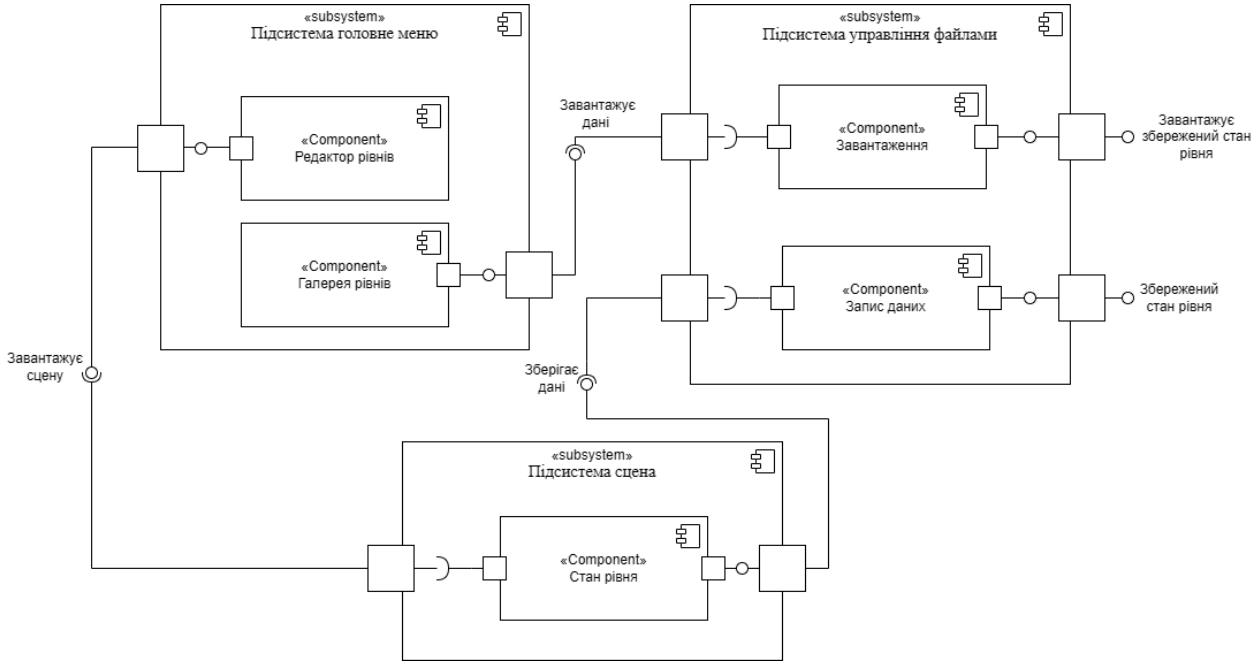

Рис. 2.3 Діаграма компонентів

3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

В рамках розробки двовимірної платформер-гри особлива увага була приділена створенню надійної та ефективної системи управління інформаційною базою. Дана система є ключовим компонентом, що забезпечує взаємодію між різними частинами гри та відповідає за збереження всіх необхідних даних.

Використання рушія Unity

В рамках розробки проєкту ігровий рушій Unity виступає фундаментальним інструментом для реалізації системи управління даними. Для забезпечення інтуїтивної взаємодії користувача з грою реалізовано комплексну систему інтерфейсу, що базується на вбудованих компонентах для відображення тексту та інтерактивних кнопок. Ці елементи формують зручний інтерфейс для взаємодії з рівнями гри.

Архітектура системи побудована на основі компонентного підходу, використовуючи базові класи Unity для створення ігрових компонентів. Такий підхід забезпечує ефективну організацію взаємодії між різними частинами системи та підтримує їх модульність.

Для координації роботи різних підсистем реалізовано систему менеджерів, що відповідають за управління галереєю рівнів та системою інструкцій. Ці компоненти забезпечують централізоване управління відповідними функціями гри та їх взаємодію.

Управління ресурсами гри здійснюється через вбудовану систему Unity [16], що включає роботу з графічними елементами, шаблонами об'єктів та іншими ігровими ресурсами. Це забезпечує оптимальне використання системних ресурсів та швидкий доступ до необхідних елементів. Менеджер рівнів бере приклад, рекомендований у навчальних матеріалах Unity [18].

Керування файловою системою

Система роботи з файлами використовує стандартні механізми Unity та .NET Framework, що забезпечує надійність та кросплатформеність рішення. Для зберігання користувацьких рівнів використовується спеціальна захищена директорія, що гарантує збереження даних між сеансами гри.

Кожен рівень проходить процес валідації перед завантаженням, включаючи перевірку наявності всіх необхідних ігрових елементів. Це забезпечує стабільність роботи гри та запобігає виникненню помилок під час геймплею.

Реалізовано повний набір операцій для роботи з файлами рівнів, включаючи створення нових, завантаження існуючих та їх видалення. При виявленні некоректних даних система відображає відповідні попередження, забезпечуючи користувача необхідною інформацією про проблеми.

Підхід до серіалізації даних

Для ефективного зберігання даних використовується формат JSON, що забезпечує структуроване зберігання інформації про рівні. Кожен файл рівня містить детальну інформацію про його структуру, включаючи назву, розташування ігрових елементів, позиції персонажів та інші важливі параметри.

Серіалізація даних здійснюється за допомогою вбудованих інструментів Unity, що забезпечує повну сумісність з екосистемою рушія та ефективне перетворення даних. Структурована організація даних реалізується через спеціалізовані класи, що відповідають за зберігання інформації про окремі елементи рівня.

Керування налаштуваннями гравця

Система налаштувань використовує вбудовані механізми Unity для збереження користувацьких параметрів. Це дозволяє зберігати різноманітні налаштування гри, включаючи стан редактора та тип активного рівня.

Особлива увага приділена системі збереження прогресу проходження гри та користувацьких налаштувань інтерфейсу. Реалізовано розділ інструкцій з інтуїтивною навігацією між сторінками, що забезпечує користувачам швидкий доступ до необхідної інформації про функціонал гри.

Така організація системи управління інформаційною базою забезпечує надійне функціонування всіх компонентів гри та створює міцний фундамент для подальшого розширення функціональності.

3.2 Розробка інформаційної бази

Структура JSON файлу збереження

В основі системи збереження даних лежить формат JSON, який забезпечує ефективну серіалізацію та десеріалізацію ігрових даних. Цей формат був обраний завдяки його універсальності та широкій підтримці в Unity. Структура файлу організована ієрархічно, що дозволяє логічно групувати пов'язані дані та забезпечує простий доступ до необхідної інформації. Для зберігання користувацьких рівнів використовується формат JSON, що забезпечує структуроване збереження інформації про рівні [8]. Також було використано рекомендації з роботи з файлами із System.IO [5].

Кожен файл збереження містить метадані, що включають версію формату, часові мітки створення та модифікації, а також службову інформацію для валідації даних. Така організація забезпечує надійне збереження та відновлення стану гри.

Формат даних рівня

Дані рівня зберігаються у вигляді структурованого JSON документа, що містить всю необхідну інформацію для відтворення ігрового простору. Кожен рівень включає унікальний ідентифікатор, назву, розміри ігрового поля та колекцію ігрових об'єктів.

Особлива увага приділяється збереженню позицій та властивостей кожного елемента рівня. Система зберігає інформацію про розташування тайлів, їх типи та властивості, що забезпечує точне відтворення дизайну рівня при завантаженні.

Зберігання даних гравця

Для зберігання даних гравця використовується комбінація системи PlayerPrefs та JSON файлів. Це дозволяє ефективно зберігати як прості параметри, так і складні структури даних.

Система відстежує поточний стан гравця, включаючи його позицію на рівні, здоров'я та інші ігрові параметри. Також зберігається інформація про режим гри та статус редактора рівнів.

Зберігання даних ворогів

Інформація про ворогів зберігається як частина даних рівня та включає детальні параметри кожного противника. Система зберігає тип ворога, його початкову позицію, параметри поведінки та інші специфічні характеристики.

Для кожного типу ворогів визначено унікальний набір параметрів, що впливають на їх поведінку та взаємодію з гравцем. Це забезпечує різноманітність ігрового процесу та можливість створення складних ігрових ситуацій.

Зберігання налаштувань

Система налаштувань використовує механізм PlayerPrefs для зберігання користувацьких параметрів гри. Зберігаються різноманітні налаштування, включаючи параметри інтерфейсу, керування та звуку.

3.3 Вибір інструментарію для створення програмного забезпечення

Можливості Unity 2D

В рамках розробки даного проєкту було обрано ігровий рушій Unity з акцентом на його 2D функціональність. Unity 2D надає розробникам широкий спектр спеціалізованих інструментів та оптимізованих компонентів [6, с. 154]. Тайлові карти стали основою для створення ігрових рівнів.

Система спрайтів Unity 2D використовується в проєкті для ефективного управління графічними ресурсами. Вона забезпечує автоматичну оптимізацію текстур через створення атласів спрайтів, що значно зменшує кількість звернень до графічного процесора та підвищує продуктивність гри. Вбудований редактор

спрайтів дозволив точно налаштувати колізії для персонажів та об'єктів, забезпечуючи природну взаємодію між елементами гри.

Тайлові карти стали основою для створення ігрових рівнів. Система Tilemap у Unity надала можливість швидко проєктувати та модифікувати ігрові локації, використовуючи набори тайлів з різними властивостями. Особливо корисною виявилася функціональність автоматичного підбору сусідніх тайлів (Auto-tiling), що дозволила створювати візуально привабливі та різноманітні ландшафти з мінімальними зусиллями.

Анімаційна система Unity забезпечила плавні рухи персонажів. Використання механізму Animator Controller дозволило створити складну систему переходів між різними станами персонажа (біг, стрибок, атака) з можливістю тонкого налаштування параметрів переходів. Інтеграція анімацій з фізичною системою забезпечила природність рухів та реакцій персонажів на взаємодію з оточенням.

Інструменти оптимізації Unity 2D, такі як динамічне групування спрайтів (Dynamic Batching) та кулінг невидимих об'єктів, дозволили досягти стабільної частоти кадрів [19] навіть при великій кількості об'єктів на екрані.

Таким чином, використання Unity 2D як основної технологічної платформи забезпечило оптимальний баланс між продуктивністю розробки, якістю візуального представлення та ефективністю виконання, що дозволило успішно реалізувати всі заплановані функціональні та естетичні аспекти проєкту.

Мова програмування C#

Вибір C# як основної мови програмування для розробки проєкту обумовлений низкою суттєвих переваг, які ця мова надає в контексті розробки ігор на платформі Unity.

C# поєднує в собі високу продуктивність розробки з ефективністю виконання коду [2]. Статична типізація мови забезпечує раннє виявлення помилок на етапі компіляції, що значно зменшує кількість потенційних помилок під час виконання. Це особливо важливо для ігрових проєктів [11, с. 205], де

стабільність роботи є критичним фактором. Деякі функції були реалізовані за допомогою прикладів зі Stack Overflow [13].

Мова має потужну підтримку об'єктно-орієнтованого програмування, що дозволяє створювати чітку та масштабовану архітектуру проєкту. Механізми інкапсуляції, наслідування та поліморфізму були активно використані для створення гнучкої системи компонентів гри.

C# надає доступ до сучасних мовних конструкцій, таких як лямбда-вирази, LINQ, асинхронне програмування з async/await, що значно підвищує читабельність та компактність коду. Ці можливості дозволяють писати елегантні рішення для складних ігрових механік.

Важливою перевагою є також потужна екосистема .NET, що надає доступ до великої кількості бібліотек та інструментів. Це дозволяє ефективно вирішувати різноманітні завдання, від роботи з колекціями даних до серіалізації об'єктів.

C# є офіційно рекомендованою мовою для розробки на Unity, що забезпечує найкращу інтеграцію з рушієм та доступ до всіх його можливостей. Документація Unity та більшість навчальних матеріалів орієнтовані саме на C#, що спрощує процес розробки та пошук рішень для типових задач.

Порівняно з альтернативами, такими як UnityScript (застаріла версія JavaScript для Unity) або Boo, C# забезпечує кращу продуктивність, більш потужні мовні можливості та кращу підтримку з боку спільноти розробників.

Важливим фактором вибору стала також можливість використання сучасних IDE, зокрема Visual Studio, що надають потужні інструменти для написання, рефакторингу та відлагодження коду на C#.

У розробленому проєкті C# використовувався для реалізації всіх аспектів ігрової логіки. Система компонентів Unity, що базується на класах MonoBehaviour, була розширена власними компонентами для реалізації специфічної поведінки ігрових об'єктів.

Для керування персонажем було створено систему класів, що відповідають за рух, стрибки, атаки та взаємодію з оточенням. Використання подій та делегатів

C# дозволило реалізувати гнучку систему комунікації між компонентами без створення жорстких залежностей.

Система збереження та завантаження рівнів використовує серіалізацію об'єктів у формат JSON з використанням атрибутів C# для контролю процесу серіалізації. Це дозволило створити гнучку та розширювану систему збереження даних. Для збереження та серіалізації даних у форматі JSON використовується JsonUtility з Unity API [17].

Користувацький інтерфейс реалізовано з використанням подійної моделі C#, що забезпечує чітке розділення між логікою гри та відображенням.

Асинхронне програмування з використанням async/await та корутин Unity дозволило ефективно реалізувати операції завантаження ресурсів, анімації інтерфейсу та інші процеси, що вимагають виконання в часі без блокування основного потоку гри.

При розробці користувацького інтерфейсу враховувались рекомендації з юзабіліті [14, с. 45].

Середовище розробки Visual Studio

Visual Studio представляє собою потужне інтегроване середовище розробки (IDE), що надає комплексний набір інструментів для створення програмного забезпечення. Для розробки ігрових проєктів на Unity це середовище має низку суттєвих переваг.

Перш за все, Visual Studio забезпечує глибоку інтеграцію з Unity через спеціальні розширення, що дозволяють безпосередньо керувати проєктом Unity з середовища IDE. Це значно прискорює робочий процес, оскільки розробник може редагувати код та тестувати зміни без постійного перемикання між різними програмами.

Середовище надає потужні інструменти для написання коду, включаючи IntelliSense – систему автодоповнення, що розуміє контекст коду та пропонує релевантні варіанти завершення. Це суттєво підвищує швидкість написання коду та зменшує кількість синтаксичних помилок.

Інструменти рефакторингу Visual Studio дозволяють легко змінювати структуру коду без порушення його функціональності. Це особливо важливо при розробці ігор, де архітектура проєкту часто еволюціонує в процесі розробки.

Вибір Visual Studio як основного середовища розробки обумовлений його статусом офіційно рекомендованого IDE для Unity. Microsoft та Unity Technologies підтримують тісну співпрацю, що забезпечує оптимальну інтеграцію між цими інструментами.

Порівняно з альтернативними середовищами, такими як Visual Studio Code або JetBrains Rider, повна версія Visual Studio надає більш комплексний набір інструментів для розробки на C#, включаючи вбудовані засоби профілювання та аналізу продуктивності.

Важливим фактором вибору стала також наявність безкоштовної версії Visual Studio Community, що надає доступ до більшості професійних функцій без додаткових витрат, що особливо важливо для невеликих проєктів.

У розробленому проєкті Visual Studio використовувалось як основне середовище для написання та відлагодження програмного коду. Інтеграція з Unity через розширення Visual Studio Tools for Unity забезпечила безперебійний робочий процес з можливістю швидкого переходу між редагуванням коду та тестуванням змін у грі.

Функція відлагодження з точками зупинки активно використовувалась для аналізу поведінки гри в критичних моментах, таких як колізії персонажа з об'єктами оточення або обробка користувацького вводу. Це дозволило швидко виявляти та виправляти помилки в логіці гри.

Інструменти рефакторингу були застосовані для оптимізації структури проєкту [3, с. 448] в процесі його розвитку. Зокрема, було проведено виділення спільної функціональності в базові класи та інтерфейси, що підвищило модульність та повторне використання коду.

Для реалізації системи керування персонажем використовувались можливості Visual Studio з навігації по коду та пошуку залежностей, що

дозволило ефективно працювати зі складною системою взаємодіючих компонентів.

При розробці редактора рівнів інструменти аналізу коду Visual Studio допомогли виявити потенційні проблеми з продуктивністю при роботі з великими рівнями. Це дозволило оптимізувати критичні ділянки коду ще на етапі розробки.

Система контролю версій, інтегрована у Visual Studio, забезпечила надійне збереження історії змін проєкту та можливість безпечного експериментування з новими функціями без ризику втрати стабільної версії.

Система UI Unity

Архітектурно система Unity UI побудована навколо концепції Canvas (полотна), що є контейнером для всіх елементів інтерфейсу. Canvas використовує спеціалізований рендерер, оптимізований для 2D-графіки, що забезпечує ефективне відображення елементів інтерфейсу.

Важливою технічною особливістю є використання Mesh-based рендерингу, де всі елементи інтерфейсу перетворюються на полігональні сітки, що дозволяє застосовувати до них стандартні графічні ефекти та трансформації. Це також забезпечує можливість батчингу — об'єднання кількох елементів інтерфейсу в один рендер-виклик, що значно підвищує продуктивність.

Система подій у Unity UI реалізована через патерн "Спостерігач" (Observer), де кожен елемент інтерфейсу може генерувати події, на які можуть підписуватися інші компоненти. Це забезпечує гнучкість та розділення відповідальності між різними частинами системи.

Для позиціонування елементів використовується система Rect Transform, що розширює стандартний Transform компонент додатковими параметрами для 2D-позиціонування. Ключовою особливістю Rect Transform є концепція якорів (anchors), що дозволяють елементам адаптуватися до зміни розміру батьківського контейнера.

Система макетів (Layout System) забезпечує автоматичне розташування елементів відповідно до заданих правил. Вона включає компоненти, такі як

Horizontal Layout Group, Vertical Layout Group, Grid Layout Group та Content Size Fitter, що дозволяють створювати адаптивні інтерфейси без необхідності ручного позиціонування кожного елемента.

Важливою частиною системи є Canvas Scaler, що забезпечує адаптацію інтерфейсу до різних розмірів екрану та роздільних здатностей. Canvas Scaler підтримує кілька режимів масштабування: Constant Pixel Size, Scale With Screen Size та Constant Physical Size, що дозволяє обрати оптимальний підхід для конкретного проєкту.

Система Physics2D

Система Physics2D у Unity представляє собою спеціалізований фізичний рушій, оптимізований для двовимірних ігор. Ця система має низку суттєвих переваг, що роблять її оптимальним вибором для реалізації фізичної взаємодії в 2D проєктах.

Перш за все, Physics2D забезпечує високоточну симуляцію фізичних процесів у двовимірному просторі з оптимальним використанням обчислювальних ресурсів. Система використовує ефективні алгоритми для розрахунку колізій та фізичних взаємодій, що дозволяє підтримувати стабільну частоту кадрів навіть при великій кількості об'єктів.

Важливою перевагою є різноманітність доступних типів колайдерів (Box, Circle, Polygon, Edge, Composite), що дозволяє точно визначати форму об'єктів для фізичних взаємодій. Це забезпечує реалістичні колізії та природну поведінку об'єктів при зіткненнях.

Система Rigidbody2D надає гнучкі можливості для налаштування фізичних властивостей об'єктів, таких як маса, гравітація, тертя та пружність. Це дозволяє створювати об'єкти з різною фізичною поведінкою, від легких повітряних платформ до важких перешкод.

Вибір Physics2D як основної системи для реалізації фізичної взаємодії обумовлений її нативною інтеграцією з рушієм Unity та оптимізацією спеціально для 2D проєктів. Це забезпечує оптимальну продуктивність без необхідності адаптації тривимірної фізики для двовимірного простору.

Порівняно з альтернативними підходами, такими як створення власної системи колізій або використання сторонніх фізичних рушіїв, Physics2D надає готове, добре протестоване рішення, що значно скорочує час розробки та забезпечує стабільну роботу на різних платформах.

Система також має потужні інструменти для налагодження та візуалізації фізичних взаємодій, що спрощує процес розробки та тестування ігрової механіки.

У розробленому проєкті система Physics2D використовувалась для реалізації всіх аспектів фізичної взаємодії, включаючи рух персонажа, стрибків, колізії з оточенням та об'єктами.

Для персонажа гравця застосовано компонент Rigidbody2D з налаштуваннями, що забезпечують плавний рух та реалістичні стрибки. Використання режиму Interpolate для Rigidbody2D дозволило уникнути візуальних артефактів при швидкому русі персонажа, забезпечуючи плавну анімацію навіть при змінній частоті кадрів.

Колізії персонажа з оточенням реалізовані через Circle Collider 2D для всього персонажу. Це дозволило точно визначати зіткнення з платформами та забезпечило природну поведінку при ходьбі по нерівних поверхнях.

Для платформ та інших елементів рівня використано різні типи колайдерів відповідно до їх форми. Статичні елементи оточення налаштовані як Static Collider для оптимізації продуктивності, оскільки вони не потребують повної фізичної симуляції.

Система шарів колізій (Layer-based Collision Matrix) застосована для контролю взаємодій між різними типами об'єктів. Це дозволило, наприклад, налаштувати проходження персонажа крізь певні типи платформ (односторонні платформи) при русі знизу вгору.

Система Physics2D також використана для реалізації взаємодії з інтерактивними об'єктами, такими як зілля здоров'я та шипи. Для цих об'єктів застосовано Trigger Collider, що дозволяє виявляти перетин з персонажем без фізичного блокування руху.

Для точки фінішу рівня також використано Trigger Collider, що активує завершення рівня при контакті з персонажем. Це забезпечує інтуїтивно зрозумілу механіку проходження рівнів.

У редакторі рівнів система Physics2D використовується для визначення позиції розміщення об'єктів через Raycast від позиції курсора. Це дозволяє точно позиціонувати елементи рівня відносно існуючих об'єктів та забезпечує інтуїтивний процес редагування.

3.4 Алгоритмізація та програмування програмних модулів

Рух гравця та бойова система

Реалізація руху персонажа через фізичний рушій Unity (Rigidbody2D) надає низку суттєвих переваг порівняно з альтернативними підходами. Перш за все, це забезпечує реалістичну фізичну поведінку персонажа, включаючи інерцію, гравітацію та взаємодію з об'єктами оточення. Такий підхід дозволяє створити природні рухи, що інтуїтивно зрозумілі для гравця.

Використання компонентного підходу, де логіка руху інкапсульована в окремому класі, забезпечує модульність та можливість легкого розширення функціональності. Це дозволяє незалежно модифікувати різні аспекти поведінки персонажа без впливу на інші системи.

Реалізація бойової системи через механізм колізій та тригерів Unity забезпечує точне визначення зіткнень та спрощує логіку обробки атак. Це дозволяє ефективно реалізувати механіку завдання шкоди та взаємодії з ворогами.

У розробленому проєкті система руху персонажа реалізована через клас, що успадковується від MonoBehaviour та керує компонентом Rigidbody2D [16]. Горизонтальний рух здійснюється шляхом зміни швидкості Rigidbody2D відповідно до користувацького вводу з клавіатури.

Для забезпечення плавності руху застосовано поступову зміну швидкості замість миттєвого встановлення, що створює відчуття інерції та ваги персонажа.

Це реалізовано через лінійну інтерполяцію між поточною та цільовою швидкістю з використанням параметра Time.deltaTime для незалежності від частоти кадрів.

Механіка стрибка реалізована через додавання вертикальної сили до Rigidbody2D при натисканні відповідної клавіші (Space). Для запобігання можливості стрибка в повітрі впроваджено систему перевірки контакту з поверхнею. Ця перевірка здійснюється через фізичний колайдер у нижній частині персонажа, що визначає наявність контакту з платформою.

Система анімації персонажа інтегрована з системою руху через компонент Animator. Зміна анімаційних станів (idle, run, jump) відбувається на основі поточної швидкості та стану персонажа. Для забезпечення коректного напрямку погляду персонажа реалізовано механізм зміни масштабу по осі X залежно від напрямку руху.

Бойова система включає механізм атаки, що активується при натисканні відповідної клавіші. При активації атаки відтворюється анімація атаки та створюється тимчасова зона ураження перед персонажем, реалізована через Trigger Collider. При перетині цього колайдера з колайдером ворога викликається метод, що зменшує здоров'я ворога.

Система здоров'я персонажа реалізована через числову змінну, що зменшується при отриманні шкоди від ворогів або небезпечних об'єктів. При зменшенні здоров'я до нуля викликається метод, що обробляє смерть персонажа та показує екран програшу.

Для забезпечення відзивчивості керування реалізовано буферизацію вводу, де команди гравця зберігаються протягом короткого проміжку часу та виконуються, як тільки персонаж опиняється у відповідному стані. Це особливо важливо для команди стрибка, що може бути активована трохи раніше приземлення.

Система також включає механізм взаємодії з об'єктами оточення, такими як зілля здоров'я та шипи. Ця взаємодія реалізована через систему тригерів Unity, де при перетині колайдера персонажа з колайдером об'єкта викликається відповідний метод обробки взаємодії.

Поведінка ворогів

При розробці проєкту було прийнято рішення реалізувати просту, але ефективну систему ворогів, що відповідає загальній концепції платформера. Статичний ворог, що завдає шкоди при контакті, є класичним елементом жанру платформерів і забезпечує базовий рівень складності для гравця без надмірного ускладнення ігрової механіки.

Такий підхід має низку переваг для навчального проєкту. По-перше, він дозволяє зосередитись на основних механіках платформера – русі персонажа, стрибках та взаємодії з оточенням. По-друге, статичні вороги легко інтегруються в систему редактора рівнів, дозволяючи користувачам створювати різноманітні випробування. По-третє, проста реалізація забезпечує стабільну роботу гри без ризику виникнення складних помилок у поведінці ворогів.

Такий тип ворога є ідеальним навчальним прикладом для студентів, які вивчають розробку ігор у навчальних закладах. Він демонструє кілька фундаментальних концепцій геймдеву:

Базова взаємодія об'єктів - студенти вивчають, як реалізувати виявлення зіткнень між ігровими об'єктами та реагувати на ці зіткнення.

Система здоров'я та шкоди - демонструє реалізацію механізму зменшення здоров'я та обробки стану невразливості.

Анімаційні стани - ворог має анімацію бездіяльності (стан "очікування" або "спокою"), що показує, як працювати з системою анімації Unity.

Баланс складності - студенти вчаться розміщувати ворогів на рівні таким чином, щоб створити збалансований ігровий досвід.

У розробленому проєкті ворог представлений як ігровий об'єкт з наступними компонентами:

Sprite Renderer для візуального відображення.

Collider2D (налаштований як тригер) для визначення зіткнень з персонажем.

Спеціалізований скрипт, що обробляє логіку взаємодії.

Основна функціональність ворога реалізована через механізм тригерів Unity. Коли персонаж гравця входить у зону тригера ворога, викликається метод OnTriggerEnter2D, що перевіряє тег об'єкта (чи це гравець) і, якщо умова виконується, викликає відповідний метод на об'єкті гравця для зменшення здоров'я.

Система реалізована таким чином, що ворог завдає фіксовану кількість шкоди при кожному контакті. Для запобігання багаторазовому завданню шкоди при тривалому контакті впроваджено механізм невразливості персонажа після отримання пошкодження.

У редакторі рівнів ворог представлений як один із доступних об'єктів, що можуть бути розміщені на рівні. При збереженні рівня позиція ворога зберігається в структурі даних рівня, а при завантаженні – відтворюється у відповідній позиції.

Візуальне представлення ворога реалізовано через анімований спрайт, що чітко ідентифікує його як небезпечний об'єкт для гравця. Важливо зазначити, що хоча ворог не рухається, він не є просто статичним спрайтом. Він має анімацію бездіяльності, що надає йому візуальної динаміки та життєвості. Це забезпечує інтуїтивне розуміння ігрової механіки без необхідності додаткових пояснень.

Функціональність редактора рівнів

Редактор рівнів у проєкті реалізовано з використанням низки ключових технологій Unity, що забезпечують ефективну роботу з ігровими об'єктами та інтерфейсом користувача:

Система Tilemap - основний інструмент для роботи з тайловими картами в Unity. Ця система дозволяє ефективно керувати великою кількістю тайлів, що формують ігровий рівень, з оптимальним використанням пам'яті та продуктивності.

Unity UI - для створення інтерфейсу редактора використано нативну систему користувацького інтерфейсу Unity. Вона забезпечує створення інтерактивних елементів керування, таких як кнопки, панелі інструментів та палітри тайлів.

Система Prefab - для роботи з ігровими об'єктами, такими як вороги, предмети для збирання та спеціальні точки (старт, фініш), використовується система префабів Unity, що дозволяє створювати шаблони об'єктів для багаторазового використання.

Система Raycasting - для визначення позиції розміщення об'єктів у світі на основі позиції курсора миші.

Система серіалізації - для збереження та завантаження створених рівнів використовується серіалізація даних у формат JSON з використанням класу JsonUtility.

Редактор рівнів надає користувачу наступні можливості:

Розміщення тайлів - користувач може вибирати тайли з палітри та розміщувати їх на карті, формуючи ландшафт рівня. Підтримується розміщення різних типів тайлів, таких як земля, платформи.

Розміщення об'єктів - користувач може розміщувати на рівні різні ігрові об'єкти, такі як статичний ворог, що завдає шкоди при контакті, предмети для збирання, що відновлюють здоров'я ігрового персонажа, і шипи які завдають шкоду ігровому персонажу.

Визначення точок старту та фінішу - користувач може визначити точку появи гравця на початку рівня та точку фінішу, досягнення якої означає успішне проходження рівня.

Збереження та завантаження рівнів - користувач може зберегти створений рівень для подальшого використання та завантажити раніше створені рівні для редагування.

Процес створення рівня в редакторі включає наступні етапи:

Користувач створює новий рівень або завантажує існуючий для редагування.

За допомогою інструменту розміщення тайлів користувач формує ландшафт рівня, розміщуючи тайли землі та платформ.

Користувач розміщує на рівні ігрові об'єкти, такі як ворог, що завдає шкоди при контакті, предмети для збирання, шипи які наносять шкоду.

Користувач визначає точку появи гравця та точку фінішу рівня. Лістинг методу PlaceObject, що реалізує механізм розміщення об'єктів, наведено в додатку А.

Після завершення редагування або створення користувач зберігає рівень, присвоюючи йому унікальну назву.

Таким чином, редактор рівнів надає користувачам інтуїтивно зрозумілий інструмент для створення власних ігрових рівнів.

Реалізація системи збереження/завантаження

Система збереження та завантаження даних у розробленому проєкті базується на технології серіалізації JSON (JavaScript Object Notation). JSON як формат даних був створений Дугласом Крокфордом на початку 2000-х років і швидко став одним із найпопулярніших форматів для обміну даними завдяки своїй простоті, читабельності та компактності.

У контексті розробки ігор JSON має низку переваг порівняно з бінарними форматами: він людиночитабельний, що спрощує відлагодження; не залежить від платформи, що забезпечує сумісність між різними операційними системами; та має вбудовану підтримку в більшості мов програмування.

Unity надає нативну підтримку серіалізації JSON через клас JsonUtility, що дозволяє легко перетворювати об'єкти C# у JSON-рядки і навпаки. Ця технологія була обрана для проєкту через її простоту використання та ефективність для невеликих наборів даних, характерних для 2D-платформера. Система збереження/завантаження в проєкті реалізована за допомогою класу JsonUtility та System.IO.File [5, 17]. Для оптимізації читання/запису JSON-конструкцій використовувались матеріали [12].

Для роботи з файловою системою використовується клас System.IO.File, що надає методи для створення, читання та запису файлів. Це стандартний підхід у .NET, який забезпечує надійний доступ до файлової системи на всіх платформах, підтримуваних Unity.

Система збереження/завантаження в проєкті реалізована за допомогою кількох взаємопов'язаних компонентів:

Менеджер збереження (SaveManager) - центральний компонент, що координує процеси збереження та завантаження даних. Він надає публічні методи для збереження та завантаження рівнів, які можуть викликатися з інших частин програми.

Класи даних - спеціалізовані класи, що представляють структуру даних для збереження. Основним є клас LevelData, що містить всю необхідну інформацію про рівень.

Система валідації - компонент, що перевіряє коректність даних перед збереженням та після завантаження, забезпечуючи цілісність ігрового процесу.

Інтерфейс користувача - елементи UI, що дозволяють користувачу ініціювати збереження та завантаження рівнів, а також переглядати доступні збережені рівні.

Процес збереження рівня включає наступні етапи:

Збір даних - система збирає інформацію про всі елементи поточного рівня, включаючи розташування тайлів, позиції об'єктів, точки старту та фінішу.

Формування об'єкта даних - зібрана інформація структурується в об'єкт класу LevelData, що має відповідні поля для всіх типів даних.

Серіалізація - об'єкт LevelData перетворюється в JSON-рядок за допомогою JsonUtility.

Запис у файл - серіалізований JSON-рядок записується у файл з унікальним ім'ям, що базується на назві рівня.

Оновлення метаданих - система оновлює список доступних рівнів, щоб новий рівень був доступний для вибору в галереї рівнів.

Процес завантаження рівня виконує зворотні операції:

Читання файлу - система зчитує JSON-дані з вибраного файлу.

Десеріалізація - JSON-рядок перетворюється в об'єкт LevelData за допомогою JsonUtility.

Валідація даних - система перевіряє коректність завантажених даних, зокрема наявність обов'язкових елементів.

Очищення поточного рівня - всі існуючі елементи рівня видаляються для підготовки до завантаження нових.

Відтворення рівня - на основі даних з об'єкта LevelData система створює всі елементи рівня: розміщує тайли на Tilemap, інстанціює об'єкти та встановлює точки старту і фінішу.

Система зберігає різноманітні типи даних, необхідні для повного відтворення рівня:

Метадані рівня - назва рівня.

Дані тайлової карти - для кожного тайла зберігається його тип, позиція на карті.

Дані об'єктів - для кожного об'єкта зберігається його тип, позиція. Для ворога зберігається його позиція на рівні, що дозволяє точно відтворити розташування перешкод.

Спеціальні точки - позиції точок старту та фінішу, необхідні для коректного початку та завершення рівня.

Лістинг методу SaveLevelConfirmed, що реалізує весь процес серіалізації рівня, подано в додатку Б.

Реалізація системи інтерфейсу

Система інтерфейсу в проєкті побудована за принципом розділення відповідальності, де кожен екран або функціональний блок інтерфейсу керується окремим компонентом. Це забезпечує модульність та можливість незалежного розвитку різних частин інтерфейсу.

Основні компоненти системи інтерфейсу включають:

Головне меню - відповідає за навігацію між різними екранами та основні налаштування гри.

Галерея рівнів - відображає список доступних створених користувачем рівнів та забезпечує взаємодію з ними.

Інтерфейс редактора рівнів - забезпечує доступ до інструментів редагування та відображає поточний стан редактора.

Ігровий HUD - відображає критичну інформацію під час гри, таку як здоров'я персонажа.

Система інструкцій - відповідає за відображення підказок та інструкцій для користувача.

Система налаштування керування - дозволяє користувачу налаштовувати клавіші керування та регулювання гучності звуку.

Метод LoadLevel, що реалізує завантаження рівня, наведено в додатку В.

Реалізація головного меню

Головне меню реалізовано як окрема сцена з використанням системи UI Unity, Canvas у режимі Screen Space - Overlay. Меню включає кілька панелей, що відображаються залежно від вибору користувача: головна панель, панель налаштувань, панель вибору рівня та панель інструкцій.

Основна панель головного меню містить наступні кнопки:

Продовжити гру - дозволяє користувачу продовжити проходження рівня, який не був завершений. При натисканні система перевіряє наявність збереженого стану незавершеного рівня та завантажує його, відновлюючи позицію гравця, стан об'єктів та прогрес. Якщо збереженого стану немає, кнопка неактивна.

Редактор рівнів - відкриває інтерфейс редактора рівнів, де користувач може створювати нові або редагувати існуючі рівні. Редактор надає повний набір інструментів для розміщення тайлів та об'єктів.

Галерея рівнів – відкриває панель яка дозволяє користувачу керувати створеними рівнями.

Налаштування - відкриває панель з можливістю регулювання гучності звуку, налаштування керування.

Інструкція - відображає панель з поясненнями ігрових механік та інтерфейсу.

Вийти з гри - закриває додаток.

Навігація між панелями реалізована через систему кнопок, де кожна кнопка має прикріплений обробник події OnClick, що викликає відповідний метод. Це забезпечує інтуїтивно зрозумілу навігацію та чітку структуру меню.

Реалізація галереї рівнів

Галерея рівнів реалізована як динамічно генерований список доступних рівнів. Реалізацію методу LoadLevelsList, що формує список рівнів, подано в додатку Д. Система сканує директорію з файлами рівнів та створює для кожного рівня окремий елемент інтерфейсу, що включає назву рівня та кнопки для гри, редагування та видалення.

Кожен елемент рівня в галереї обробляє взаємодію користувача з конкретним рівнем. При натисканні на кнопку "Грати" викликається метод, що завантажує відповідний рівень для гри. При натисканні на кнопку "Редагувати" рівень завантажується в редактор рівнів для модифікації.

Для ефективного відображення великої кількості рівнів використано компонент Scroll View, що дозволяє прокручувати список, коли кількість рівнів перевищує розмір екрану.

Реалізація інтерфейсу редактора рівнів

Інтерфейс редактора рівнів включає кілька функціональних блоків:

Палітра тайлів - відображає доступні типи тайлів, які можна розмістити або видалити на рівні. Реалізована як сітка кнопок, де кожна кнопка представляє окремий тип тайлу.

Панель об'єктів - містить кнопки для вибору різних типів об'єктів, таких як ворог, шипи, зілля здоров'я, ігровий персонаж та фінішна позиція.

Панель керування рівнем - містить кнопки для збереження, а також поле для введення назви збереженого рівня.

Реалізація ігрового HUD

Ігровий HUD відображає критичну інформацію під час гри, таку як поточний рівень здоров'я персонажа. Реалізований як Canvas у режимі Screen Space - Overlay, що забезпечує стабільне відображення незалежно від положення камери.

Для відображення здоров'я використано компонент Slider, що візуально представляє поточний рівень здоров'я як заповнену частину шкали. Оновлення значення здоров'я здійснюється через виклик методу, що змінює значення Slider відповідно до поточного стану персонажа.

Реалізація системи інструкцій

Система інструкцій відповідає за відображення підказок та інструкцій для користувача. Інструкції відображаються як текстові панелі з поясненнями так і ілюстрації, що допомагають користувачу зрозуміти ігрові механіки та інтерфейс.

Навігація між різними сторінками інструкцій реалізована через кнопки "Далі" та "Назад", що дозволяють користувачу переглядати інструкції у зручному темпі. Також реалізована кнопка "Вийти у меню"

Налаштування керування

Система налаштування керування дозволяє користувачу змінювати клавіші, що використовуються для різних дій у грі, таких як рух, стрибок та атака.

Інтерфейс налаштування керування включає список доступних дій та відповідних клавіш. При натисканні на кнопку зміни клавіші система переходить у режим очікування натискання нової клавіші, яка потім зберігається як нове значення для відповідної дії. Для кожної дії відображається поточна призначена клавіша.

Регулювання гучності музики та звуків

Система налаштування аудіо дозволяє користувачам окремо регулювати гучність музики, забезпечуючи комфортний аудіо досвід.

Інтерфейс налаштування аудіо включає слайдер:

Слайдер гучності музики - дозволяє регулювати гучність фонової музики від 0% (повністю вимкнено) до 100% (максимальна гучність). При зміні положення слайдера система миттєво застосовує нове значення гучності до всіх музичних треків.

4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

4.1 Тестування системи

Функціональне тестування головного меню

Функціональне тестування головного меню включало комплексну перевірку всіх елементів інтерфейсу та їх взаємодії з іншими компонентами системи.

Перевірка відображення елементів інтерфейсу

Було підтверджено, що всі елементи головного меню коректно відображаються на різних розмірах екрану. Текстові елементи чіткі та розбірливі, кнопки мають достатній розмір. Фоновий малюнок гармонійно поєднується з іншими елементами інтерфейсу.

Тестування функціональності кнопок

Кожна кнопка головного меню була протестована на коректність роботи:

Кнопка "Продовжити гру" - перевірено, що кнопка активна лише за наявності збереженого прогресу. При натисканні система коректно завантажує останній збережений стан гри, відновлюючи позицію персонажа, стан об'єктів та прогрес проходження. Якщо збережень немає, кнопка неактивна.

Кнопка "Редактор рівнів" - підтверджено, що при натисканні відбувається коректний перехід до інтерфейсу редактора рівнів. Перевірено збереження налаштувань користувача при переході між меню та редактором.

Кнопка "Галерея" - перевірено, що при натисканні система коректно завантажує та відображає список доступних рівнів. Протестовано сценарії з різною кількістю рівнів, включаючи випадок відсутності збережених рівнів.

Кнопка "Налаштування" - підтверджено, що при натисканні відкривається панель налаштувань з усіма доступними опціями. Перевірено, що зміни в налаштуваннях зберігаються після закриття панелі та перезапуску гри.

Кнопка "Інструкція" - перевірено, що при натисканні відображається панель з інструкціями щодо ігрового процесу. Протестовано навігацію між різними сторінками інструкції.

Кнопка "Вихід" - підтверджено, що при натисканні гра коректно завершує роботу.

Рисунок 4.1 демонструє результат успішного тестування головного меню, де всі елементи інтерфейсу відображаються коректно та функціонують відповідно до вимог. Головне меню працює плавно, без затримок при відображенні елементів або обробці дій користувача. Підтверджено, що система ефективно використовує ресурси пристрою та не викликає надмірного навантаження.

Рис. 4.1 Головне меню

Функціональне тестування галереї рівнів

Функціональне тестування галереї рівнів включало комплексну перевірку всіх елементів інтерфейсу та їх взаємодії з системою управління рівнями.

Перевірка завантаження та відображення списку рівнів

Було підтверджено, що система коректно сканує директорію з файлами рівнів та відображає повний список доступних рівнів. Тестування включало різні сценарії:

Відображення великої кількості рівнів

Коректна робота при відсутності збережених рівнів

Правильне відображення рівнів з довгими назвами

Перевірено, що для кожного рівня коректно відображається назва.

Тестування функціональності кнопки "Грати"

Перевірка коректного завантаження обраного рівня

Тестування переходу до ігрової сцени з правильними параметрами

Підтверджено, що при натисканні на кнопку "Грати" система коректно ініціює процес завантаження рівня, безпомилково переходить до ігрового процесу.

Тестування функціональності кнопки "Редагувати"

Перевірка коректного завантаження рівня в редактор

Тестування збереження всіх елементів рівня

Перевірка можливості модифікації та повторного збереження рівня

Підтверджено, що редактор коректно відкривається з усіма елементами обраного рівня, дозволяючи вносити зміни та зберігати оновлену версію.

Тестування функціональності кнопки "Видалити"

Тестування процесу видалення файлу рівня

Перевірка оновлення списку рівнів після видалення

Підтверджено, що система, коректно видаляє файл рівня та оновлює інтерфейс без необхідності перезавантаження галереї.

Тестування системи прокрутки та навігації

Перевірка плавності прокрутки списку

Тестування коректної роботи скроллбару

Перевірка відображення всіх елементів при різних розмірах екрану

Підтверджено, що система прокрутки працює плавно та дозволяє ефективно переглядати список навіть при наявності великої кількості рівнів.

Тестування продуктивності та стабільності

Перевірка швидкості завантаження списку рівнів

Тестування стабільності при тривалому використанні

Рисунок 4.2 демонструє результат успішного тестування галереї рівнів, де всі елементи інтерфейсу відображаються коректно та функціонують відповідно до вимог.

Рис. 4.2 Галерея рівнів

Функціональне тестування редактора рівнів

Функціональне тестування редактора рівнів включало комплексну перевірку всіх інструментів та функцій, необхідних для створення та редагування ігрових рівнів.

Перевірка інтерфейсу редактора

Було підтверджено, що всі елементи інтерфейсу редактора коректно відображаються. Перевірено, що робоча область має достатній розмір для зручного редагування, а панелі інструментів не перекривають важливі елементи інтерфейсу.

Тестування включало:

Тестування масштабування робочої області

Перевірку навігації по великих рівнях

Перевірку розташування інтерфейсних елементів для зручної взаємодії з ними

Тестування інструментів розміщення тайлів

Перевірка вибору тайлів з палітри та їх розміщення на рівні

Перевірка видалення та заміни існуючих тайлів

Підтверджено, що всі інструменти працюють коректно та дозволяють ефективно створювати різноманітні структури.

Тестування розміщення об'єктів

Тестування вибору та розміщення різних типів об'єктів

Тестування переміщення та видалення існуючих об'єктів

Перевірка розміщення точки появи гравця

Тестування розміщення точки фінішу

Підтверджено, що система коректно обробляє спеціальні точки та створення різних об’єктів в сцені редактора рівнів.

Тестування панелі керування рівнем

Тестування збереження рівня з введенням назви в текстове поле та натисканням кнопки "ОК"

Перевірка блокування збереження при відсутності введеної назви рівня

Тестування кнопки "Назад" для повернення до головного меню

Тестування поведінки системи при спробі зберегти рівень з уже існуючою назвою

Підтверджено, що всі функції керування рівнем працюють коректно та забезпечують зручний робочий процес.

Рисунки 4.3, 4.4, 4.5 демонструють результат успішного тестування редактора рівнів, де всі елементи інтерфейсу та інструменти функціонують відповідно до вимог.

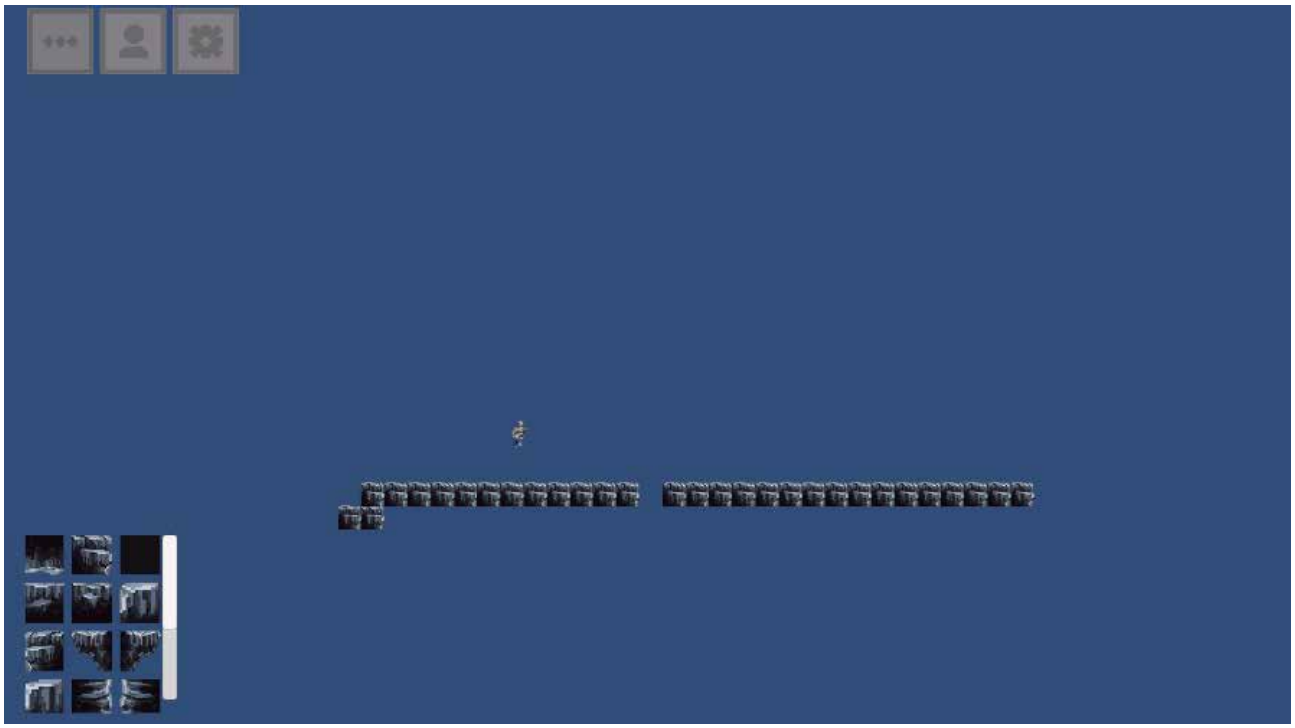

Рис. 4.3 Редактор рівнів "Палітра тайлів"

Рис. 4.4 Редактор рівнів "Розміщення об'єктів"

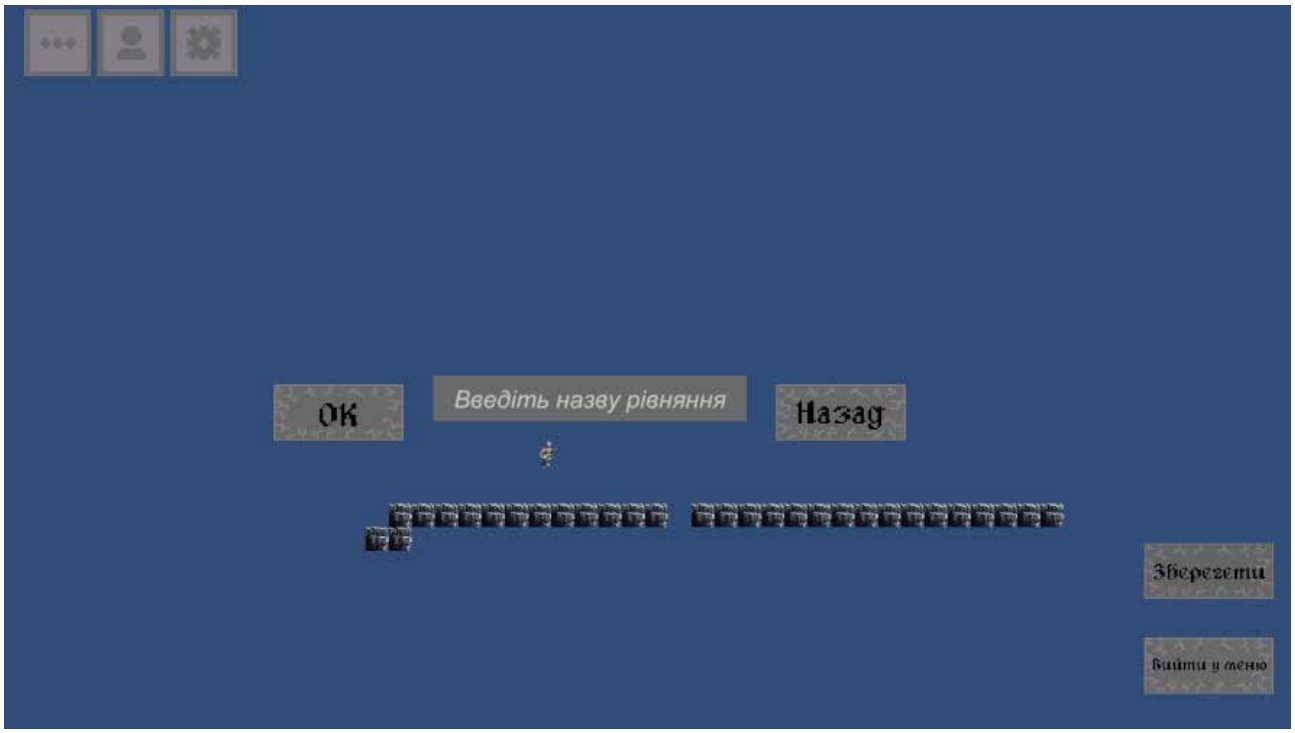

Рис. 4.5 Редактор рівнів "Керування рівнем"

Функціональне тестування меню налаштувань

Функціональне тестування меню налаштувань включало комплексну перевірку всіх елементів інтерфейсу та їх взаємодії з системою налаштувань.

Перевірка регулювання гучності музики

Було проведено детальне тестування слайдера регулювання гучності музики:

Перевірка коректного відображення поточного рівня гучності при відкритті меню налаштувань

Тестування плавної зміни гучності при переміщенні слайдера

Перевірка збереження встановленого рівня гучності після закриття меню

Тестування граничних значень

Підтверджено, що система коректно застосовує зміни гучності до всіх музичних треків у грі.

Тестування налаштувань керування

Проведено комплексне тестування системи налаштування керування ігровим персонажем:

Перевірка відображення поточних призначених клавіш для всіх дій

Тестування процесу зміни клавіш

Тестування збереження змінених налаштувань керування після закриття меню

Підтверджено, що система коректно відображає, змінює та зберігає налаштування керування, забезпечуючи персоналізацію ігрового досвіду.

Тестування інтерфейсу меню налаштувань

Перевірено загальну функціональність інтерфейсу меню налаштувань:

Перевірка коректного відображення всіх елементів інтерфейсу на різних розмірах екрану

Тестування кнопки повернення до головного меню

Перевірка збереження всіх змінених налаштувань при виході з меню

Підтверджено, що інтерфейс меню налаштувань зручний у використанні та забезпечує інтуїтивно зрозумілий доступ до всіх функцій налаштування гри.

Рисунок 4.6 демонструє інтерфейс меню налаштувань з двома основними секціями: регулювання гучності музики та налаштування керування.

Рис. 4.6 Налаштування

Тестування ігрового процесу

Тестування HUD

Проведено комплексне тестування відображення та функціональності ігрового інтерфейсу:

Перевірено коректне відображення індикатора здоров'я у вигляді перевернутого щита в куті екрану

Протестовано зміну стану індикатора при отриманні пошкоджень - підтверджено, що щит змінює свій вигляд, з'являються тріщини, які візуально відображають поточний стан здоров'я персонажа

Перевірено чіткість відображення індикатора на різних фонах рівнів

Протестовано коректне оновлення індикатора при відновленні здоров'я

Підтверджено, що ігровий HUD чітко та інформативно відображає стан персонажа, дозволяючи гравцю легко відстежувати рівень здоров'я під час проходження рівня.

Взаємодія з ворогами

Перевірено, що при зіткненні з ворогом персонаж на мить загоряється червоним кольором

Підтверджено, що при кожному зіткненні персонаж втрачає 1 одиницю здоров'я

Протестовано коректне оновлення індикатора здоров'я після отримання пошкодження

Перевірено наявність періоду невразливості після отримання пошкодження

Взаємодія з пастками

Підтверджено, що при наступанні на шипи персонаж на мить загоряється червоним кольором

Перевірено, що шипи наносять 1 одиницю пошкодження за кожне зіткнення

Протестовано коректне оновлення індикатора здоров'я після отримання пошкодження

Взаємодія з зіллями здоров'я

Перевірено, що при зіткненні з зіллям здоров'я воно зникає з рівня

Підтверджено, що здоров'я персонажа збільшується на 1 одиницю

Протестовано коректне оновлення індикатора здоров'я після підбирання зілля

Перевірено, що зілля не відновлює здоров'я понад максимальне значення індикатору здоров’я ігрового персонажу

Взаємодія з точкою фінішу

Підтверджено, що при досягненні точки фінішу рівень успішно завершується

Перевірено коректний перехід повернення до меню

Тестування показало, що всі механіки взаємодії з об'єктами працюють коректно та відповідають очікуваній поведінці. Візуальні індикатори чітко сигналізують гравцю про небезпечні ситуації, а механіка відновлення здоров'я через зілля забезпечує необхідний баланс складності.

Рисунок 4.7 демонструє ігровий процес з усіма протестованими елементами: персонаж, індикатор здоров'я у вигляді щита, вороги, перешкоди на рівні, зілля здоров’я та фініш.

Рис. 4.7 Ігровий процес

4.2 Вимоги до апаратного та програмного забезпечення

У цьому розділі розглядаються технічні аспекти функціонування програмного забезпечення для створення інтерактивної мультимедійної системи в двомірному середовищі. Метою розділу є визначення та обґрунтування вимог до апаратного та програмного забезпечення, необхідних для ефективної роботи системи. Розділ охоплює аналіз архітектури програмного забезпечення, специфікацію компонентів системи та вимоги до пристроїв користувачів.

Вимоги до апаратного забезпечення пристроїв

Таблиця 4.1

Вимоги до апаратного забезпечення пристроїв

Компонент	**Мінімальні вимоги**	**Рекомендовані вимоги**
Операційна система	Windows 7/8/10 (64-біт)	Windows 10 (64-біт)
Процесор	Intel Core i3-3220 або AMD FX-4350	Intel Core i5-6600 або AMD Ryzen 5 1600
Оперативна пам'ять	4 ГБ RAM	8 ГБ RAM
Відеокарта	NVIDIA GeForce GTX 460 або AMD Radeon HD 7750 (1 ГБ VRAM)	NVIDIA GeForce GTX 960 або AMD Radeon R9 380 (2 ГБ VRAM)
DirectX	Версія 10	Версія 11

Додаткові програмні компоненти

.NET Framework : 4.7.1 або новіше

Microsoft Visual C++ Redistributable : 2015, 2017 і 2019 (x64)

Вимоги до накопичувача для збереження рівнів

Система збереження рівнів використовує локальне сховище на жорсткому диску користувача. Для забезпечення надійного збереження та швидкого завантаження користувацьких рівнів рекомендується:

Вільний простір: Мінімум 100 МБ вільного місця для збереження користувацьких рівнів. Кожен рівень зберігається у форматі JSON і займає від 10 КБ до 100 КБ залежно від складності та розміру рівня.

Тип накопичувача: Рекомендується використання SSD-накопичувача для забезпечення швидкого доступу до файлів збережень, особливо при роботі з великою кількістю рівнів у галереї.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення складається з наступних компонентів, необхідних для повноцінного функціонування системи:

Виконувані файли

Game2D.exe (651 КБ) - основний виконуваний файл програми, що забезпечує запуск та функціонування інтерактивної мультимедійної системи

UnityCrashHandler64.exe (1 158 КБ) - службовий компонент для обробки та реєстрації критичних помилок під час виконання програми

Бібліотеки та залежності

UnityPlayer.dll (30 110 КБ) - основна бібліотека Unity Engine, що забезпечує функціонування графічного рушія, системи фізики, аудіо та інших компонентів ігрового середовища.

Директорії з ресурсами

Game2D_Data - директорія, що містить всі необхідні ресурси для функціонування програми:

Managed/ - скомпільовані збірки C# коду, включаючи всі скрипти гри

Resources/ - графічні ресурси

StreamingAssets/ - аудіо файли

il2cpp_data/ - дані для роботи IL2CPP компілятора

globalgamemanagers - файли для управління глобальними ресурсами гри

Службові директорії

MonoBleedingEdge - компоненти середовища виконання Mono для підтримки .NET функціональності

Конфігураційні файли

Конфігураційні файли зберігаються в директорії користувача після першого запуску програми:

Файли збережених рівнів у форматі JSON

Налаштування користувача (гучність музики, конфігурація керування) зберігаються через систему PlayerPrefs

ВИСНОВКИ

У рамках дипломної роботи було реалізовано повнофункціональну гру жанру 2D-платформер з вбудованим редактором рівнів, що дає змогу користувачеві створювати, редагувати та проходити власні ігрові локації. Головна ідея полягала у створенні інтуїтивного інтерфейсу та зручного середовища для взаємодії, що забезпечується засобами Unity та мовою програмування C#.

Протягом роботи над проєктом було проведено детальний аналіз предметної області, визначено ключові вимоги до ігрового рушія, сценарії взаємодії гравця з рівнем, а також реалізовано логіку збереження і відновлення прогресу за допомогою формату JSON. Всі компоненти гри розроблені у вигляді окремих модулів.

Особливу увагу було приділено зручності користувача — інтерфейс гри розроблено з урахуванням принципів UI/UX. Передбачено різні панелі: головне меню, налаштування, редактор рівнів, інструкції. Проведене тестування підтвердило стабільну роботу усіх елементів інтерфейсу та коректну реакцію на дії користувача. Протестовано сценарії збереження рівнів, редагування, перегляду та відновлення проходження гри з місця завершення.

Редактор рівнів надає користувачу інструментарій для побудови власних рівнів, з можливістю розміщення об’єктів, ворогів, пасток, точок старту і фінішу. Це дозволяє гравцю бути не лише споживачем контенту, а й його творцем.

Система управління файлами та налаштуваннями реалізована із застосуванням вбудованих механізмів Unity, що гарантує простоту зберігання даних без залучення сторонніх СУБД. Таким чином, реалізована система є кросплатформенною, не вимагає серверної частини та забезпечує швидкий запуск без додаткових налаштувань.

Отримані результати підтверджують, що обрана архітектура та інструменти є доцільними, ефективними та добре масштабованими.

Результати цієї дипломної роботи підтвердили доцільність розробки 2D-гри з інтерактивним редактором рівнів, що дає змогу користувачам не лише

проходити готові рівні, а й створювати власні. Застосування рушія Unity та мови C# дозволило реалізувати функціональний та зручний інтерфейс, гнучку систему збереження даних і стабільну ігрову логіку. Отриманий програмний продукт може бути використаний як основа для комерційного або навчального застосування, а також має перспективи подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Borromeo N. A. Hands-On Unity 2020 Game Development: Build, Customize, and Optimize Professional Games Using Unity 2020 and C#. Packt Publishing, Limited, 2020. 580 p.
2. C# Guide – .NET managed language. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 21.05.2025).
3. Fowler M. Refactoring: Improving the Design of Existing Code. Pearson Education, Limited, 2018.
4. Game Developer | Game Industry News, Deep Dives, and Developer Blogs. URL: https://www.gamedeveloper.com/ (дата звернення: 21.05.2025).
5. GameDev.net. URL: https://www.gamedev.net/ (дата звернення: 21.05.2025).
6. Hardman C. Game Programming with Unity and C#: A Complete Beginner’s Guide. Apress, 2020. 597 p.
7. Head First Design Patterns: A Brain-Friendly Guide / K. Sierra et al. O'Reilly Media, Incorporated, 2004. 694 p.
8. JSON. URL: https://www.json.org/json-en.html (дата звернення: 21.05.2025).
9. Kan S. H. Metrics and Models in Software Quality Engineering (paperback). Pearson Education, Limited, 2002. 560 p.
10. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition). 3rd ed. Prentice Hall PTR, 2004. 704 p.
11. Martin R. C. Agile Software Development, Principles, Patterns, and Practices. 2nd ed. Prentice Hall, 2002. 529 p.
12. MDN Web Docs. URL: https://developer.mozilla.org/ (дата звернення: 21.05.2025).
13. Nielsen J. Usability Engineering. Elsevier Science & Technology Books, 1994.
14. Newest Questions. Stack Overflow. URL: https://stackoverflow.com/ (дата звернення: 21.05.2025).

15. Pressman R. S. Software Engineering: A Practitioner's Approach. 6th ed. McGraw-Hill Science/Engineering/Math, 2004. 896 p.
16. Unity – Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/ (дата звернення: 21.05.2025).
17. Unity – Scripting API: JsonUtility. URL: https://docs.unity3d.com/ScriptReference/JsonUtility.html (дата звернення: 21.05.2025).
18. Unity Learn. URL: https://learn.unity.com/ (дата звернення: 21.05.2025).
19. UnityGems.com. URL: http://unitygems.com/ (дата звернення: 21.05.2025).
20. Савчук А. А. Програмне забезпечення для створення інтерактивної мультимедійної системи в двомірному середовищі. Збірник наукових праць за матеріалами VII Всеукраїнської науково-практичної конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем 2025», м. Київ, 24 квітня. 2025 р. НУБІП України. Київ, 2025.

ДОДАТКИ

ДОДАТОК А

Редактор рівнів

```
public void PlaceObject(Vector3 worldPos)
{
   worldPos.z = 0;

   switch (selectedObjectType)
   {
      case "PlayerSpawn":
         if (playerSpawnPoint == null)
            playerSpawnPoint = Instantiate(playerSpawnPointPrefab, worldPos, Quaternion.identity);
         else
            playerSpawnPoint.transform.position = worldPos;
         break;

      case "Enemy":
         Instantiate(enemyPrefab, worldPos, Quaternion.identity);
         break;

      case "Spike":
         Instantiate(spikePrefab, worldPos, Quaternion.identity);
         break;

      case "Finish":
         if (finishPoint == null)
            finishPoint = Instantiate(finishPointPrefab, worldPos, Quaternion.identity);
         else
            finishPoint.transform.position = worldPos;
         break;
   }
}
```

ДОДАТОК Б

Система збереження рівня

```
private void SaveLevelConfirmed()
{
   LevelData levelData = new LevelData();
   levelData.levelName = levelNameInput.text;
   BoundsInt bounds = groundTilemap.cellBounds;
   for (int x = bounds.min.x; x < bounds.max.x; x++)
   {
      for (int y = bounds.min.y; y < bounds.max.y; y++)
      {
         Vector3Int tilePosition = new Vector3Int(x, y, 0);
         TileBase tile = groundTilemap.GetTile(tilePosition);
         if (tile != null)
         {
            int tileIndex = System.Array.IndexOf(availableTiles, tile);
            if (tileIndex != -1)
            {
               levelData.tiles.Add(new TileData(x, y, tileIndex));
            }
         }
      }
   }
   GameObject[] allObjects = GameObject.FindObjectsOfType<GameObject>();
   foreach (GameObject obj in allObjects)
   {
      if (obj.GetComponent<Enemy>() != null)
         levelData.objects.Add(new ObjectData("Enemy", obj.transform.position.x, obj.transform.position.y));
      else if (obj.GetComponent<HealPotion>() != null)
         levelData.objects.Add(new ObjectData("HealPotion", obj.transform.position.x, obj.transform.position.y));
      else if (obj.GetComponent<Spikes>() != null)
         levelData.objects.Add(new ObjectData("Spike", obj.transform.position.x, obj.transform.position.y));
      else if (obj.CompareTag("Finish"))
      {
         levelData.hasFinish = true;
         levelData.finishPosition = obj.transform.position;
         levelData.objects.Add(new ObjectData("FinishPoint", obj.transform.position.x, obj.transform.position.y));
      }
      else if (obj.GetComponent<Hero>() != null)
      {
         levelData.hasPlayer = true;
         levelData.playerPosition = obj.transform.position;
      }
   }
   string json = JsonUtility.ToJson(levelData, true);
   string path = Application.persistentDataPath + "/Levels/";
   System.IO.Directory.CreateDirectory(path);
   System.IO.File.WriteAllText(path + levelNameInput.text + ".json", json);
}
```

ДОДАТОК В

Система завантаження рівня

```
public void LoadLevel(string levelName)
{
   string path = Application.persistentDataPath + "/Levels/" + levelName + ".json";
   if (!File.Exists(path)) return;

   string json = File.ReadAllText(path);
   LevelData levelData = JsonUtility.FromJson<LevelData>(json);

   ClearLevel();
   foreach (TileData tileData in levelData.tiles)
   {
      Vector3Int position = new Vector3Int(tileData.x, tileData.y, 0);
      groundTilemap.SetTile(position, availableTiles[tileData.tileIndex]);
   }
   foreach (ObjectData objData in levelData.objects)
   {
      GameObject prefab = GetPrefabByType(objData.objectType);
      if (prefab != null)
      {
         Vector3 position = new Vector3(objData.x, objData.y, 0);
         Instantiate(prefab, position, Quaternion.identity);
      }
   }

   if (levelData.hasPlayer)
   {
      GameObject player = Instantiate(playerPrefab, levelData.playerPosition, Quaternion.identity);
   }
}
```

ДОДАТОК Д

Система галереї рівнів

```
public void LoadLevelsList()
{
   foreach (Transform child in levelButtonsContainer)
   {
      Destroy(child.gameObject);
   }

   string levelsPath = Application.persistentDataPath + "/Levels/";
   if (!Directory.Exists(levelsPath))
   {
      Directory.CreateDirectory(levelsPath);
      return;
   }

   string[] levelFiles = Directory.GetFiles(levelsPath, "*.json");
   foreach (string levelFile in levelFiles)
   {
      string levelName = Path.GetFileNameWithoutExtension(levelFile);
      GameObject buttonObj = Instantiate(levelButtonPrefab, levelButtonsContainer);
      LevelButtonController controller = buttonObj.GetComponent<LevelButtonController>();
      if (controller != null)
      {
         controller.Setup(levelName, this);
      }
   }
}

public void LoadLevel(string levelName, bool isEditing)
{
   PlayerPrefs.SetString("LoadLevelName", levelName);
   PlayerPrefs.SetInt("IsEditing", isEditing ? 1 : 0);
   PlayerPrefs.SetInt("IsCustomLevel", 1);
   UnityEngine.SceneManagement.SceneManager.LoadScene(isEditing ? 1 : 2);
}
```