ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка відеогри в жанрі 2D side-scrolling»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав ПОНОМАРЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Владислав ПОНОМАРЕНКО

Керівник: Олесь ДІБРІВНИЙ
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пономаренко Владиславу Олеговичу

1. Тема кваліфікаційної роботи: «Розробка відеогри в жанрі 2D side-scrolling»
2. Строк подання кваліфікаційної роботи «28» травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: результатом роботи є повноцінний ігровий прототип, який демонструє основи побудови ігрової логіки, роботи з фізикою, анімацією та управлінням сценами.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Огляд відеогр в жанрі 2D side-scrolling та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
    2. Огляд та аналіз засобів реалізації відеогри в жанрі 2D side-scrolling.
    3. Проєктування архітектури відеогри в жанрі 2D side-scrolling.
    4. Програмна реалізація та тестування відеогри в жанрі 2D side-scrolling.
5. Перелік графічного матеріалу: презентація

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Опис програми.
8. Екранні форми.
9. Демонстрація роботи застосунку
10. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Визначення об’єкта, предмета, мети кваліфікаційної роботи бакалавра та постановка завдань.	25.02-01.03.2025	
2.	Підбір та аналіз науково-технічної літератури	02.03-09.03.2025	
3.	Аналіз відеоігор-аналогів у жанрі 2D side-scrolling платформерів.	10.03-16.03.2025	
4.	Визначення функціональних та нефункціональних вимог до гри.	17.03-21.03.2025	
5.	Вибір ігрового рушія та обґрунтування використаних технологій.	21.03-25.03.2025	
6.	Проєктування архітектури гри та основних класів.	25.03-29.03.2025	
7.	Розробка прототипу рівня та механіки переміщення персонажа.	01.04-10.04.2025	
8.	Імплементація візуального оформлення (піксель-арт, tileset, анімації).	11.04-17.04.2025	
9.	Розробка UI (користувацького інтерфейсу)	18.04-26.04.2025	
10.	Реалізація системи звукового супроводу та музики.	27.04-04.05.2025	
11.	Розробка демонстраційних матеріалів	05.05-12.05.2025	
12.	Попередній захист роботи	12.05-16.05.2025	

Здобувач вищої освіти _____ Владислав ПОНОМАРЕНКО
(підпис)

Керівник кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 48 стор., 6 табл., 22 рис., 20 джерел.

Об’єкт дослідження: процес дослідження, аналізу та розробки гри у жанрі platformer.

Мета роботи: спроєктувати архітектуру гри, реалізувати функціональні рішення, відсутні в проаналізованих аналогах та підвищити зручність використання та доступності гри за рахунок реалізації інтуїтивного керування та налаштувань візуальної доступності.

Предмет дослідження — програмні засоби, методи та технології реалізації ігрових механік, анімації та взаємодії користувача з ігровим середовищем у 2D side-scrolling грі.

Короткий зміст роботи — робота присвячена розробці 2D-платформерної гри у середовищі Unity із використанням мови програмування C#. Реалізовано базову ігрову механіку, включно з керуванням персонажем, анімаціями, стрибками, атаками та взаємодією з об’єктами оточення, зокрема різними типами платформ. Особливу увагу приділено адаптивному інтерфейсу користувача та зручності навігації в грі. Результатом роботи є повноцінний ігровий прототип, який демонструє основи побудови ігрової логіки, роботи з фізикою, анімацією та управлінням сценами.

КЛЮЧОВІ СЛОВА: ІГРОВИЙ РУШІЙ, UNITY, ПЛАТФОРМЕР, АНІМАЦІЯ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, ПРОГРАМУВАННЯ, МЕХАНІКА ГРИ, РОЗРОБКА ІГОР.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Unity — кросплатформний ігровий рушій для створення тривимірних (3D) та двовимірних (2D) ігор

2D — Двовимірний (двомірний) простір

UI — User Interface — користувацький інтерфейс

UX — User Experience — користувацький досвід

FPS — Frames Per Second — кількість кадрів за секунду

GPU — Graphics Processing Unit — графічний процесор

CPU — Central Processing Unit — центральний процесор

API — Application Programming Interface — інтерфейс прикладного програмування

URP — Universal Render Pipeline — універсальний рендеринговий конвеєр Unity

IDE — Integrated Development Environment — інтегроване середовище розробки

C# — Об’єктно-орієнтована мова програмування, що використовується в Unity

HUD — Heads-Up Display — графічні елементи інтерфейсу поверх зображення гри

FOV — Field of View — кут огляду

OS — Operating System — операційна система

Sprite — Спрайт — окрема графічна одиниця, що використовується для відображення об'єктів у 2D-грі

Tileset — Набір тайлів (плиток), які складаються у ігрові рівні або фони

Pixel Art — Піксель-арт — стиль графіки, у якому зображення створені з окремих пікселів, зазвичай з обмеженою палітрою

Collider — Компонент у фізичному рушії, який визначає область зіткнення об’єкта

Animation Frame — Кадр анімації — окрема частина послідовності руху спрайту

Level Design — Проєктування рівнів — процес створення структури, логіки та розміщення об'єктів у грі

ВСТУП

Сфера розробки відеоігор на сьогоднішній день є однією з найдинамічніших галузей програмної інженерії, яка об'єднує в собі досягнення комп'ютерних наук, графічного дизайну, музичного мистецтва та сценарної майстерності. Зростання попиту на ігрові продукти, розвиток платформ для їх поширення, а також відкритість інструментів для створення ігор створили умови, у яких індивідуальні розробники та малі студії здатні створювати повноцінні ігрові проєкти, що успішно конкурують на ринку. У цьому контексті важливо не лише створити технічно стабільну гру, але й забезпечити інноваційні та креативні рішення, що зробить гру більш конкурентоспроможною та привабливою для широкої аудиторії.

Одним із популярних жанрів, що зберігає актуальність протягом десятиліть, є 2D side-scrolling platformer. Популярність цього жанру зумовлена його доступністю для гравців, простотою освоєння та великою різноманітністю можливих сюжетів і механік. Класичні ігри на кшталт *Super Mario Bros.* чи *Castlevania* заклали основу для розвитку жанру, а сучасні проєкти, такі як *Hollow Knight* чи *Blasphemous*, демонструють нові художні та технічні підходи в його реалізації.

У рамках цієї дипломної роботи здійснюється розробка 2D side-scrolling гри з використанням рушія Unity. Особлива увага приділяється не лише ігровій механіці та візуальному оформленню, а й важливим аспектам зручності користувача, зокрема — доступності гри для осіб з порушенням зору шляхом реалізації режимів високої контрастності та кольорової сліпоти. Крім того, впроваджується система динамічного звукового супроводу, що адаптується до ситуацій у грі .

Метою роботи є встановлення принципів побудови повноцінного функціонального ігрового процесу, включаючи ігрову логіку, керування персонажем, художнє оформлення та систему доступності в межах жанру 2D side-scrolling. У межах дослідження проведено аналіз предметної області, розглянуто наявні аналоги, обґрунтовано вибір середовища розробки та прийнято ключові проєктні рішення щодо реалізації гри. Основною ціллю є створення інтерактивного розважального продукту, який забезпечує користувачам можливість цікавого дозвілля.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення поняття комп'ютерних ігор

Комп'ютерна гра — це інтерактивна програмна система, призначена для організації процесу дозвілля або навчання шляхом імітації певних сценаріїв, середовищ чи ситуацій, у яких користувач взаємодіє з віртуальними об'єктами за допомогою певних правил і механік.

Ключовими характеристиками комп'ютерних ігор є інтерактивність, цілеспрямованість, системність і створення віртуального середовища. Комп'ютерні ігри можуть виконувати як розважальну, так і освітню, тренувальну або дослідницьку функцію, а їхня складність варіюється від простих логічних задач до багаторівневих симуляцій реальних або вигаданих світів [1].

1.2 Історична довідка про розвиток комп'ютерних ігор

Історія комп'ютерних ігор бере свій початок у 1950-х роках. Однією з перших ігор стала "Tennis for Two" (1958), розроблена Вільямом Хігінботемом, яка демонструвала спрощену модель тенісу на осцилографі [2]. У 1972 році компанія Atari випустила "Pong", який став першою комерційно успішною аркадною грою.

Подальший розвиток ігрової індустрії включає такі знакові події:

1978 рік — вихід "Space Invaders", який започаткував популярність жанру "шутерів" [3];

1980 рік — випуск "Pac-Man" компанією Namco, що популяризував концепцію ігор із персонажами;

1985 рік — реліз "Super Mario Bros." від Nintendo, який сформував основи жанру платформерів.

У 1990-х роках з розвитком технологій 3D-графіки з'явилися нові жанри, такі як шутери від першої особи ("Doom", 1993) і рольові ігри ("Diablo", 1996). Паралельно почався розвиток багатокористувацьких мережевих ігор.

Сьогодні ігрова індустрія є однією з найбільш динамічних сфер цифрової економіки, інтегруючи досягнення штучного інтелекту, мобільних технологій, віртуальної та доповненої реальності [4].

1.3 Основні компоненти розробки комп'ютерних ігор

Розробка комп'ютерних ігор є багатопрофільним процесом, що охоплює кілька основних напрямів:

Програмна архітектура — проєктування системи обробки графіки, фізики, анімації, взаємодії користувача.

Геймдизайн — створення правил гри, механік, сценаріїв і взаємодії гравця із середовищем.

Графічний дизайн — розробка візуального оформлення гри (2D/3D моделі, текстури, анімації).

Аудіодизайн — створення музичного супроводу, звукових ефектів і озвучення персонажів.

Тестування — перевірка ігрового процесу на наявність помилок, оптимізація продуктивності та балансу гри.

1.4 Інструменти та технології розробки

Для створення сучасних відеоігор широко використовуються ігрові рушії, серед яких основними є:

Unity — універсальний рушій для розробки 2D та 3D ігор, що підтримує мультиплатформеність і має розвинену екосистему [5].

Unreal Engine — рушій, орієнтований на створення графічно насичених ігор з широкими можливостями для моделювання фізики і реалістичної візуалізації.

Godot Engine — відкритий і безкоштовний рушій, особливо популярний серед розробників 2D проєктів.

Використання готових інструментів дозволяє суттєво скоротити час розробки та зосередити увагу на унікальних аспектах проєкту.

1.5 Тенденції та актуальні проблеми галузі

Серед основних тенденцій розвитку ігрової індустрії можна виділити:

- Поширення інді-розробки завдяки доступності інструментів;
- Впровадження віртуальної та доповненої реальності;
- Використання хмарних технологій для стрімінгу ігор;
- Зростання уваги до інклюзивності, зокрема розробки ігор для людей із порушеннями зору, слуху або моторики.

Серед актуальних викликів галузі варто зазначити:

- Оптимізацію продуктивності на різних платформах;
- Забезпечення балансу між інноваційністю та комфортом користувача;
- Ускладнення процесу забезпечення кросплатформеності;
- Розробку процедурної генерації контенту та систем на основі штучного інтелекту [6].

2 ОГЛЯД ТА АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВІДЕОГРИ В ЖАНРІ 2D SIDE-SCROLLING

2.1 Поняття ігрового рушія

Ігровий рушій — це спеціалізоване програмне забезпечення, що забезпечує базову інфраструктуру для розробки комп'ютерних ігор. Рушії надають розробникам набір інструментів для роботи з графікою, фізикою, звуком, анімацією, штучним інтелектом, мережевими взаємодіями та іншими складовими, що необхідні для створення повноцінного ігрового продукту [1].

Використання готових рушіїв значно спрощує процес розробки, дозволяючи зосередитися на унікальних аспектах гри, а не на створенні базових механізмів із нуля.

2.2 Критерії вибору ігрового рушія

Основними критеріями вибору рушія для розробки є:

- Підтримка потрібного типу графіки (2D, 3D або комбінована);
- Гнучкість налаштувань і можливість розширення функціоналу;
- Зручність роботи з фізикою і анімацією;
- Можливість створення ігрового процесу з необхідною логікою;
- Підтримка цільових платформ для публікації гри;
- Якість документації та наявність спільноти користувачів;
- Підтримка систем доступності для користувачів із особливими потребами.

2.3 Аналіз популярних ігрових рушіїв

1. Unity

Unity є одним з найпопулярніших ігрових рушіїв у світі для розробки як 2D, так і 3D ігор. Особливості:

- Інструменти для створення 2D-графіки (Tilemap, Sprite Renderer, Animator);
- Широка підтримка платформ (Windows, Android, iOS, WebGL тощо);
- Інтеграція Universal Render Pipeline (URP) для вдосконаленої графіки та оптимізації;
- Великий обсяг документації та активна спільнота розробників;
- Можливість швидкої інтеграції аудіо, візуальних ефектів та мережевих функцій.

Unity відзначається також можливістю розробки інтуїтивного інтерфейсу користувача та широкими можливостями налаштування процесу розробки через C#.

2. Unreal Engine

Unreal Engine — високотехнологічний рушій, орієнтований переважно на розробку 3D-ігор, проте має також підтримку 2D через систему Paper2D. Особливості:

- Висока якість графіки завдяки рушію рендерингу Lumen;
- Візуальне програмування через Blueprint System;
- Підтримка великих проєктів з високими вимогами до продуктивності;
- Інтегровані засоби для створення кінематографічних сцен.

Однак для невеликих 2D-проєктів Unreal може бути надлишковим через складність налаштування і вищі системні вимоги.

3. Godot Engine

Godot — рушій із відкритим кодом, який активно використовується для 2D-розробки.

Особливості:

- Вбудований потужний редактор сцен і анімацій;
- Простота роботи з 2D-графікою;
- Підтримка мови GDScript, а також C#, C++ та VisualScript;
- Відсутність ліцензійних платежів незалежно від доходу проєкту.

4. Godot є чудовим вибором для невеликих інді-проєктів, однак потребує додаткового налаштування для розширеної оптимізації або публікації на великих платформах.

Cocos2d-x

Cocos2d-x — легкий фреймворк для створення 2D-ігор, переважно для мобільних платформ.

Особливості:

- Висока продуктивність завдяки нативному коду (C++);
- Підтримка мультиплатформності;
- Широка гнучкість в управлінні ресурсами.

Однак Cocos2d-x вимагає від розробника більш глибоких знань програмування на низькому рівні, що ускладнює розробку для початківців.

3 ВИБІР СЕРЕДОВИЩА РОЗРОБКИ І ОБҐРУНТУВАННЯ ТЕХНОЛОГІЙ

3.1 Вибір середовища розробки

Для реалізації 2D side-scrolling платформера обране середовище розробки Unity.

Вибір обумовлений наступними факторами:

Підтримка 2D-проєктів. Unity має спеціалізовані інструменти для створення 2D-ігор, включно з редактором спрайтів, системою анімацій, фізичним рушієм для двовимірних об'єктів та готовими шаблонами для швидкого старту розробки.

Гнучкість і масштабованість. Unity дозволяє легко змінювати розширення екрана, масштабувати рівні та налаштовувати керування, що особливо важливо для side-scrolling механіки.

Підтримка мультиплатформеності. Ігри, створені в Unity, можуть бути експортовані на різні платформи, зокрема Windows, Android, iOS, WebGL, що забезпечує розширення потенційної аудиторії.

Наявність великої кількості готових рішень. Unity Asset Store пропонує велику кількість безкоштовних та комерційних пакетів для прискорення розробки — від анімаційних спрайтів до готових систем фізики чи інтерфейсів користувача.

Підтримка розширень візуальної обробки. Зокрема, інтеграція Universal Render Pipeline (URP) дозволяє застосовувати налаштування графіки для адаптації гри під потреби людей із порушеннями зору.

- Широкі можливості інтеграції звукових систем;
- Наявність документації та великої кількості навчальних матеріалів;

- Оптимальна продуктивність при розробці середнього масштабу проєктів.

У якості середовища розробки коду (IDE) використовується Visual Studio 2022 із встановленим модулем для розробки під Unity. Це забезпечує комфортну роботу із C#, інтеграцію з системою контролю версій Git, автодоповнення та діагностику помилок.

3.2 Обґрунтування вибору технологій

Графічний рушій: Universal Render Pipeline (URP)

Для досягнення високої оптимізації та гнучкого контролю над візуальними ефектами обрано використання URP (Universal Render Pipeline)**.** URP дозволяє:

- Оптимізувати обробку графіки для різних пристроїв;
- Легко налаштовувати постпроцесінгові ефекти, такі як кольорокорекція, розмиття руху, фільтри для режимів високої контрастності та кольорової сліпоти;
- Забезпечити плавний рендеринг 2D-графіки.
- Широкі можливості об'єктно-орієнтованого програмування;
- Інтеграцію з усіма компонентами Unity API;
- Простоту підтримки й розширення проєкту у майбутньому.

4 ОГЛЯД ВІДЕОІГОР У ЖАНРІ 2D SIDE-SCROLLING ПЛАТФОРМЕР

4.1 Поняття жанру 2D side-scrolling платформера

Жанр 2D side-scrolling платформер передбачає ігровий процес, у якому гравець керує персонажем, що пересувається у двовимірному просторі переважно зліва направо (або навпаки), взаємодіючи з елементами рівня через стрибки, ухилення, атаки тощо. Основними характеристиками цього жанру є наявність платформ, перешкод, ворогів, а також акцент на точність керування та рефлекси гравця [1].

Цей жанр є одним із найстаріших і найпопулярніших у відеоігровій індустрії, завдяки простоті сприйняття, універсальності і великому потенціалу для творчої реалізації.

Основні характеристики жанру:

- Двовимірна проекція з побічним (side-view) відображенням ігрового світу.
- Використання тайлсетів та анімованих спрайтів
- Геймплей, орієнтований на платформерні механіки — стрибки, лазання, ухилення, битви тощо.
- Рівневий або відкритий дизайн
- Інтуїтивне управління, що передбачає точну взаємодію користувача з персонажем.
- Рівні з чітко визначеною метою
- Зростаюча складність

4.2 Історія розвитку жанру

Жанр 2D платформера бере свій початок у ранні роки розвитку відеоігор, ще у 1980-х роках, коли обмеження апаратного забезпечення комп'ютерів і ігрових приставок визначали вигляд ігрового процесу. Однією з перших ігор, що заклала основи жанру, була Donkey Kong (1981) від Nintendo, де гравець управляв персонажем, який підіймався платформами, уникаючи перешкод. Вона ввела концепцію багаторівневого екрану з вертикальним і горизонтальним рухом — те, що стало основою для платформерів [1]. Справжній прорив стався з виходом Super Mario Bros. (1985), яка задала нові стандарти для жанру: плавне управління, чітко структуровані рівні, бонуси, вороги та секрети. Саме ця гра зробила платформери надзвичайно популярними на десятиліття вперед [2]. У наступні роки жанр розвивався завдяки таким серіям, як Mega Man, Castlevania, Metroid, які додавали елементи стрілянини, дослідження світу та нелінійного проходження [3]. У 1990-х розвиток платформерів досягнув нового рівня з появою 16-бітних приставок, таких як SNES і Sega Genesis. Цей період подарував світу такі хіти, як Sonic the Hedgehog і Donkey Kong Country, які вражали гравців яскравою графікою, швидким темпом і новими механіками [4]. З переходом індустрії до 3D-графіки наприкінці 1990-х жанр 2D платформерів тимчасово втратив популярність, поступившись місцем 3D-проєктам, але залишився в пам'яті гравців як класика [5]. Починаючи з 2000-х років, разом зі зростанням інді-розробки, відбулося відродження жанру. Такі ігри, як Braid (2008)(див. рисунок 4.1), Super Meat Boy (2010) (див. рисунок 4.2), Celeste (2018) і Hollow Knight (2017), поєднали класичну платформену механіку з новими ідеями, глибоким сюжетом і художнім стилем, що надало жанру нового дихання [6]. Сучасні 2D платформери демонструють величезне розмаїття — від атмосферних пригод до хардкорних випробувань — і продовжують еволюціонувати завдяки творчості розробників і любові гравців до класики.

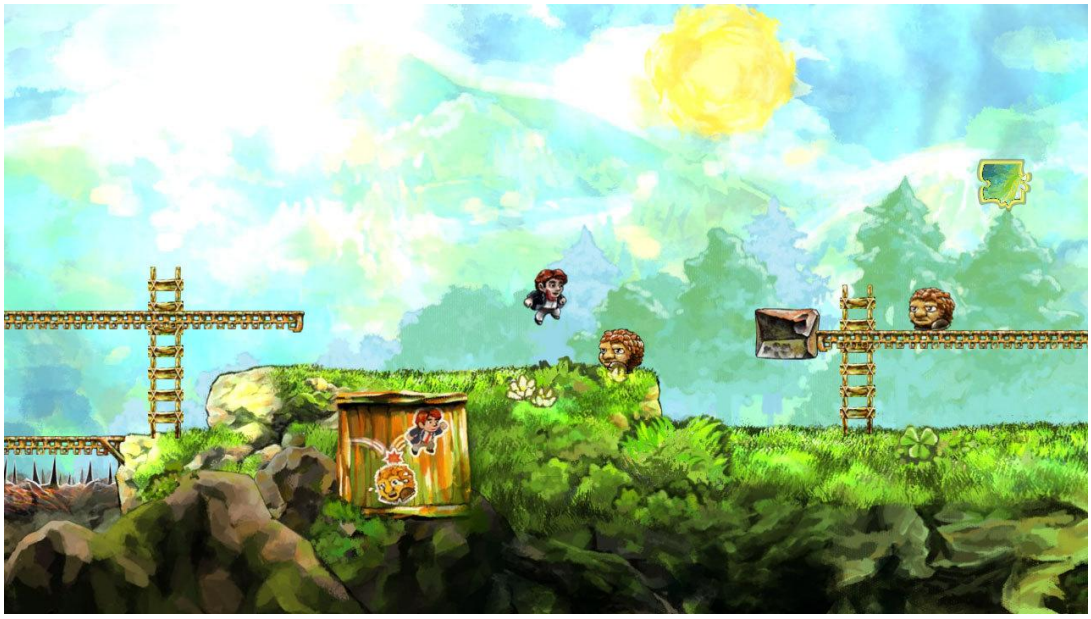


Рис. 4.1 Кадр з гри Braid (2008)

Рис. 4.2 Кадр з гри Super Meat Boy (2010)

Назва жанру «платформер» походить від англійського слова platform — «платформа». У контексті відеоігор це слово стало використовуватись для позначення ігор, у яких основна механіка полягає у пересуванні персонажа між

платформами, що розташовані на різних висотах і вимагають точного стрибка чи переміщення.

4.3 Аналіз популярних представників жанру

Таблиця 4.1

Огляд аналогічних ігор в жанрі платформер

Назва гри	**Рік виходу**	**Основні особливості**	**Вплив на жанр**
Super Mario Bros. (Nintendo)	1985	Просте управління, чітка фізика стрибків, багатошаровий дизайн рівнів	Створення класичної структури платформерів
Castlevania (Konami)	1985	Атмосферний готичний стиль, бойові механіки з використанням зброї	Введення бойових елементів і дослідження локацій
Hollow Knight (Team Cherry)	1986	Розвинена система бою, дослідження великого відкритого світу	Розвиток концепції «метроїдванії»
Blasphemous (The Game Kitchen)	2017	Поєднання платформінгу з важким бойовим процесом	Підвищення ролі сюжету і візуального стилю в 2D платформерах
Ori and the Blind Forest (Moon Studios)	2015	Високий рівень анімації,динамічний платформінг	Інтеграція наративної складової

4.4 Огляд відеоігор в жанрі 2D side-scrolling платформер

1. Super Mario Bros.: Історія, геймплей та значення гри

1.1 Історія розробки

Гра Super Mario Bros. була розроблена компанією Nintendo і випущена у 1985 році для консолі Nintendo Entertainment System (NES). Її створили Сіґеру Міямото (Shigeru Miyamoto) як головний дизайнер і Такаші Тезука як співавтор. Гра була продовженням аркадної гри Mario Bros. (1983), але з новим підходом до створення відкритого світу в межах 2D-платформера.

Ідея гри полягала в поєднанні простоти управління з цікавим рівневим дизайном, який дозволяє гравцям поступово опановувати ігрові механіки. Завдяки технічним обмеженням NES, розробникам довелося креативно підходити до дизайну ворогів, локацій та анімацій.

Super Mario Bros. став культовим проєктом, який визначив базові принципи жанру 2D side-scrolling platformer та задав стандарти дизайну для подальших десятиліть.

1.2 Аналіз геймплею

Основна механіка гри полягає в переміщенні персонажа Маріо зліва направо по рівнях, долаючи перешкоди, уникаючи ворогів та збираючи бонуси (монети, гриби, зірки). Гравець має обмежену кількість життів і повинен дістатися до прапора наприкінці рівня, щоб перейти до наступного етапу.

Ключові особливості геймплею:

- Поступове зростання складності: перші рівні слугують навчанням, далі вводяться нові типи ворогів, перешкод і платформ.

- Рівні-пастки та секрети: гра містить приховані блоки, труби, бонусні кімнати, що стимулює дослідження.

- Інтуїтивне управління: два основні режими дій — біг та стрибок — реалізовані через прості натискання кнопок.

- Фізика стрибка: інерція, змінна висота стрибка залежно від тривалості натискання — інновація на той час.

- Унікальні вороги: кожен тип ворога має передбачувану поведінку, яку можна вивчити й використати.

Цей підхід забезпечує баланс між викликом і досяжністю, що стало моделлю для наступних платформерів.

1.3 Сенс гри та вплив

Попри простий сюжет, гра не лише розважає, але й формує культуру ігрового дизайну. Сенс гри можна трактувати як подорож героя з подоланням труднощів, що символізує класичну архетипову «мандрівку».

Super Mario Bros. (див. рисунок 4.3):

- започаткувала масову популярність домашніх консолей;

- створила еталон механік платформера;

- продемонструвала, як ігровий дизайн може навчати гравця без інструкцій;

- зробила Маріо глобально впізнаваним персонажем.

- Її успіх заклав основу для розвитку серії, яка включає десятки продовжень, спін-офів, фільмів і коміксів.

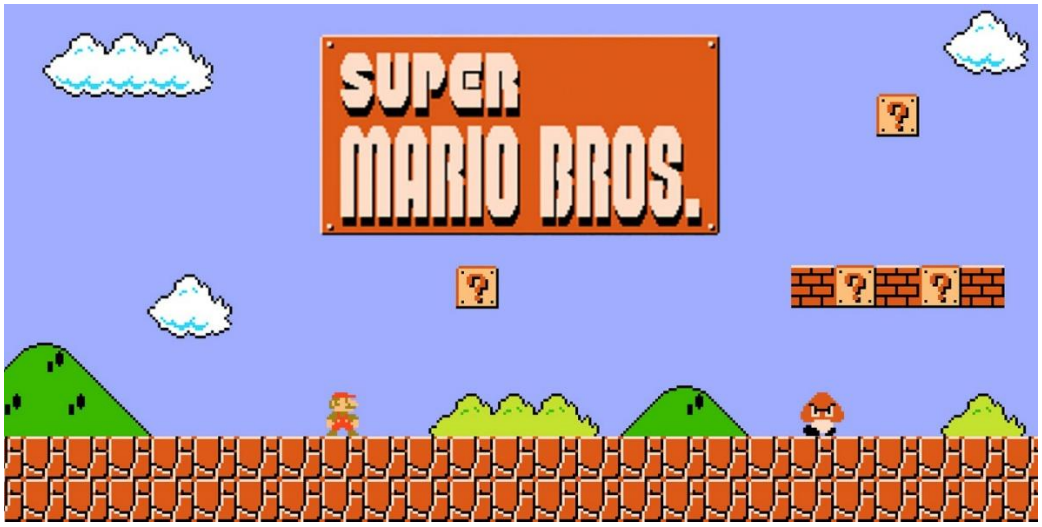


Рис. 4.3 Кадр з гри Super Mario Bros.

2. Castlevania: Історія створення, геймплейні особливості та концептуальне наповнення

2.1 Історія розробки

Castlevania — це серія відеоігор, перша частина якої була випущена компанією Konami у 1986 році для японської платформи Family Computer Disk System (Famicom Disk System), а згодом — адаптована для Nintendo Entertainment System (NES). Розробкою займалася команда Konami Kūhaku Dōmei, відома пізніше як Konami Computer Entertainment Tokyo. Основна мета першої гри — надати гравцю кінематографічний досвід у межах 2D-платформера із хорор-естетикою.

Світ гри базується на класичних готичних і жахливих образах — замок Дракули, скелети, вампіри, перевертні. Така естетика виділяла Castlevania серед інших проєктів свого часу, закладаючи фундамент для численних продовжень і трансмедійних адаптацій (комікси, аніме, серіали).

2.2 Аналіз геймплею

Castlevania поєднує традиційну платформену механіку з елементами бойової системи та дослідження. Гравець керує персонажем Саймоном Бельмонтом (Simon Belmont), який повинен пройти через низку рівнів, щоб перемогти Дракулу.

Основні геймплейні характеристики:

- Зброя: основна — батіг (Vampire Killer), вторинна — хрест, сокира, свята вода тощо. Вторинні види зброї споживають «серця» як ресурс.
- Рівнева структура: кожен рівень має чітко визначений маршрут з ворогами, перешкодами та босом у кінці.
- Висока складність: обмежений контроль під час стрибків, сильні вороги, відсутність можливості зберігатися (у перших частинах).

- Готичне оформлення: візуальний стиль і музика формують атмосферу напруги й тривожності.

- Лінійний прогрес: перші частини не мали вільного дослідження, але серія згодом перейшла до структури Metroidvania (починаючи з *Castlevania: Symphony of the Night*).

- Геймплей вирізняється системністю: кожен елемент (тип ворога, розміщення платформ, таймінг атаки) має критичне значення для проходження.

2.3 Сенс гри та її внесок у жанр

Хоча сюжет Castlevania досить простий — перемогти Дракулу й очистити замок від зла — гра втілює боротьбу світла з темрявою, а персонаж гравця виступає носієм порядку та надії у світі хаосу. Сенс гри можна трактувати як персональну мандрівку героя, з подоланням страху та фізичних випробувань, що мають символічний вимір.

Значення Castlevania (див. рисунок 4.4-4.6) в історії ігрової індустрії:

- Визначила жанр готичного екшен-платформера.

- Внесла внесок у виникнення піджанру "Metroidvania", що поєднує нелінійне дослідження з RPG-елементами.

- Створила приклад геймдизайну з послідовною атмосферністю та високою складністю, який досі наслідується.

- Започаткувала одну з найвідоміших ігрових франшиз, яка активно розвивається в нових форматах.

Рис. 4.4 Кадр з гри Castlevania

Рис. 4.5 Кадр з гри Castlevania 2

Рис. 4.6 Кадр з гри Castlevania

3. Hollow Knight: Історія створення, геймплейний аналіз і концептуальне наповнення

3.1 Історія розробки

Hollow Knight — інді-гра в жанрі 2D-платформера з елементами дослідження та боїв, розроблена австралійською студією Team Cherry. Розробка почалася у 2013 році як проєкт невеликої групи ентузіастів, а фінансування здійснювалося через краудфандингову платформу Kickstarter, де гра зібрала понад $50,000 (перевищивши початкову ціль у $35,000) [1].

Гра вийшла у 2017 році для ПК, а згодом була портована на Nintendo Switch, PlayStation 4 та Xbox One. Hollow Knight одразу здобула визнання критиків і гравців завдяки глибокій атмосфері, складному, але чесному геймплею, та інноваційному підходу до жанру "Metroidvania".

3.2 Аналіз геймплею

Гравець керує безіменним лицарем, який досліджує підземний світ Галоунесту (Hallownest) — зруйноване королівство комах. Геймплей є симбіозом платформінгу, бойової системи та нелінійного дослідження.

Ключові геймплейні особливості:

- Нелінійна структура світу: гравець вільно досліджує розгалужену карту з численними секретами.
- Розвиток персонажа: застосування чармів (талантів), нових навичок (подвійний стрибок, телепорт, удари в повітрі).
- Система бою: точні та тактичні бої з ворогами й босами; управління ресурсом "Душа", яка накопичується при ударах і використовується для лікування або заклять.
- Атмосферний саундтрек і стиль: музика Крістофера Ларкіна та візуальне оформлення створюють меланхолічну, іноді гнітючу атмосферу.
- Система прогресу: гравець не отримує чітких інструкцій; розвиток залежить від самостійного дослідження та прийняття рішень.
- Особливою рисою Hollow Knight є баланс між свободою гравця та складністю, що заохочує ретельне дослідження, а не прямолінійне проходження.

3.3 Сенс гри та її наративна структура

На наративному рівні Hollow Knight розповідає історію занепалого королівства, у якому сили зараження охопили істот, позбавляючи їх волі. Головний герой — один із "порожніх" (Hollow), створених для стримування цієї чуми.

Гра досліджує теми:

- самопожертви;
- влади та її наслідків;
- екзистенційної порожнечі;
- розпаду цивілізації;

- боротьби із власною сутністю.

На відміну від традиційних платформерів, Hollow Knight не дає чітких відповідей, а вимагає інтерпретації. Відкритий світ, фрагментарне подання історії через діалоги, символіку та описи предметів створює глибокий метафоричний пласт, що резонує з гравцем на психологічному рівні.

3.4 Значення гри в розвитку жанру

Hollow Knight значно вплинув на сучасний стан жанру Metroidvania, поєднуючи класичні елементи з новими інноваціями:

- Високий рівень полірування й художньої реалізації в інді-розробці.
- Вдало інтегрований саунд-дизайн і атмосферне оформлення.
- Гнучка бойова система та великий обсяг контенту, який перевершує багато ААА-проєктів.
- Підхід до наративу через середовище (environmental storytelling).

Таким чином, гра не лише демонструє технічну досконаліть, а й слугує прикладом арт-наративної гри, у якій ігровий процес, візуальний стиль і сенс становлять єдине ціле. На рисунку 4.7 показано кадр з гри Hollow Knight.

Рис. 4.7 Кадр з гри Hollow Knight

4. Blasphemous: Історія створення, геймплейний аналіз і концептуальне наповнення

4.1 Історія розробки

Blasphemous — це 2D-платформер у стилі Metroidvania з акцентом на бойову складову, розроблений іспанською студією The Game Kitchen. Ідея гри виникла після завершення їхнього попереднього проєкту The Last Door, який теж мав похмуру атмосферу, однак був реалізований у жанрі point-and-click.

Фінансування Blasphemous здійснювалося через платформу Kickstarter, де в 2017 році гра зібрала понад $330,000, що дозволило команді значно розширити масштаби розробки. Повноцінний реліз відбувся у вересні 2019 року на платформах PC, PlayStation 4, Xbox One і Nintendo Switch [1].

Розробники намагалися створити унікальну гру, натхненну культурою та релігійною символікою Іспанії, зокрема з андалузького регіону. Цей культурний контекст став основою для візуального стилю, архітектури світу, а також сюжету гри.

4.2 Аналіз геймплею

Blasphemous поєднує класичний платформінг із хардкорною бойовою системою, схожою на Souls-like ігри. Гравець керує персонажем на ім’я The Penitent One — мовчазним воїном, який пережив "Миродиво" (The Miracle) — надприродну катастрофу, що спотворила світ.

Основні геймплейні механіки:

- Бойова система: ближній бій з використанням меча Mea Culpa, блокування, ухилення та активація особливих атак.
- Система кар на смерть: після загибелі герой втрачає частину шкали "Fervour" (аналог мани), яку потрібно повертати, повертаючись до місця смерті.
- Дослідження світу: складна розгалужена карта, наповнена таємницями, побічними квестами, платформінг-завданнями та босами.

- Покращення персонажа: використання реліквій, молитв, сердечок меча та перлів для модифікації бойових характеристик.

- Гра має високий рівень складності, вимагає точної координації дій та глибокого вивчення поведінки супротивників. При цьому до уваги береться і складна вертикальна побудова рівнів, яка нагадує класичні платформери серії Castlevania.

4.3 Сенс гри та наратив

Наратив у Blasphemous значною мірою подається через символіку, візуальні образи та метафори.

Сюжет розвивається у світі Cvstodia, де люди спотворені через чудесне, але жорстоке явище — "Миродиво". Герой уособлює вічне каяття та пошук прощення, а його шлях — це паломництво, наповнене жертвами та випробуваннями.

Особливістю Blasphemous є глибоке культурне коріння, натхнене католицькими ритуалами, середньовічною іконографією, та святами Іспанії. Візуальний стиль, що черпає з традицій релігійного живопису та бароко, підсилює враження духовного трилера у вигляді гри.

4.4 Значення гри в контексті жанру

Blasphemous (див. рисунок 4.8) — це не лише черговий представник жанру Metroidvania, а гра, що розширює його межі завдяки:

- глибокому наративу з релігійним підтекстом;

- стилізованому візуальному ряду, що вирізняє її серед інших платформерів;

- соулзлайк-бойовій системі, інтегрованій у 2D-простір;

- високій увазі до деталей (анімації страти, дизайн ворогів, історії кожного предмета).

Ці особливості забезпечили грі культовий статус у спільноті гравців та позитивну оцінку критиків, а також відкрили шлях для сиквелу (Blasphemous II, 2023).

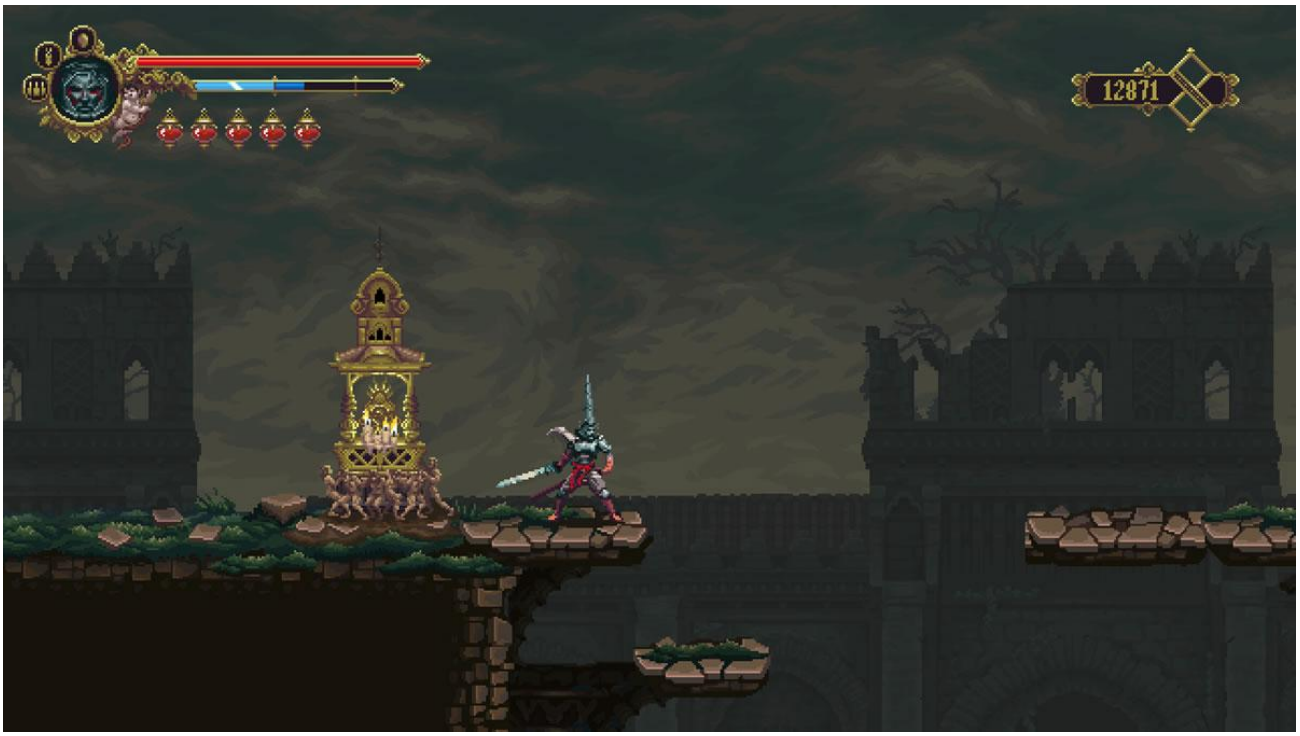


Рис. 4.8 Кадр з гри Blasphemous

5. Celeste: Історія створення, геймплейний аналіз та концептуальне наповнення

5.1 Історія розробки

Celeste — це інді-гра в жанрі платформера, розроблена студією Matt Makes Games, очолюваною геймдизайнером Меттом Торсоном. Початково гра була створена як невеликий прототип на платформі PICO-8 усього за чотири дні, проте згодом була значно розширена та перероблена у повноцінну гру.

Реліз Celeste відбувся 25 січня 2018 року для платформ PC, Nintendo Switch, PlayStation 4, Xbox One та пізніше для Google Stadia. Гру було створено на рушії

XNA, що дозволило зосередитися на точній фізиці руху та чіткому контролі персонажа [1].

Проєкт здобув міжнародне визнання та численні нагороди, зокрема премії The Game Awards 2018 за «Найкращу інді-гру» та «Гру з найкращим впливом».

5.2 Аналіз геймплею

Гравець керує молодою дівчиною на ім’я Меделін, яка намагається піднятися на вершину гори Селеста. Ігровий процес зосереджений на точному платформінгу, з акцентом на подолання складних перешкод та механік, що змінюються залежно від рівня.

Основні геймплейні особливості:

- Контрольований стрибок, ковзання по стінах та ривок у повітрі (dash);
- Рівні-пазли, що вимагають точності, реакції та пам’яті;
- Система збереження прогресу через "екрани" — кожен відрізок рівня є самостійним викликом;
- Можливість проходження додаткових рівнів зі збільшеною складністю (B-Sides і C-Sides);
- Покрокове навчання механікам без нав’язливих інструкцій.

Незважаючи на високу складність, гра має гуманну філософію: кількість смертей не карається, а є частиною процесу навчання. Це дозволяє гравцеві вчитись на помилках і відчути поступовий розвиток власних навичок.

5.3 Сенс гри та наратив

Головною темою Celeste є особистісна боротьба, подолання тривоги, депресії та страхів. Гора, на яку піднімається Меделін, є метафорою внутрішнього шляху людини, що прагне подолати себе.

Кожна частина шляху відображає емоційні стани героїні, а з’явлення її «темної сторони» (частини, яка сумнівається та боїться) стає центральним елементом наративу. У діалогах піднімаються такі теми, як:

самоприйняття;

важливість співпереживання;

підтримка з боку оточення;

роль невдач як частини шляху.

Таким чином, гра перетворює платформер у глибоко особистий досвід, поєднуючи складні механіки з душевним наративом.

5.4 Значення гри в жанрі 2D-платформерів

Celeste стала важливою віхою в еволюції жанру завдяки:

- ідеальній точності керування персонажем;

- емоційно насиченому сюжету, який інтегрується в геймплей;

- стилізованій піксельній графіці з музичним супроводом від композиторки Lena Raine;

- відкритості для гравців з різним рівнем підготовки (гра має вбудований режим допомоги — Assist Mode);

- етичному меседжу про прийняття себе.

Завдяки цим якостям Celeste стала прикладом того, як інді-ігри можуть не тільки розважати, а й передавати глибокі людські переживання через інтерактивні засоби.

4.4 Сучасні тенденції у 2D side-scrolling платформерах

Як видно з аналізу, розвиток жанру пов'язаний із постійним розширенням функціональних можливостей і глибини оповіді при збереженні основної механіки переміщення та взаємодії в обмеженому просторі.

Сучасні платформери часто включають:

- Наявність нелінійних карт (елементи метроїдванії);

- Високий рівень візуальної деталізації і анімації;

- Використання процедурної генерації рівнів;

- Додаткові системи прогресії персонажа (наприклад, розвиток навичок, екіпірування);

- Врахування принципів інклюзивного дизайну (наприклад, режими високої контрастності, підтримка людей із дальтонізмом).

Таким чином, сучасні 2D side-scrolling платформери прагнуть забезпечити як високий рівень виклику для гравця, так і комфорт для широкого кола користувачів.

Таблиця 4.2

Порівняння функціоналу аналогів та розробленого проекту

Назва гри	Parallax Shader	Dynamic Audio	High contrast mode	Save system	Візуальний стиль	Двигун (Engine)	Особливості управління
Blasphemous	+	+	-	+	Піксель-арт	Unity	Висока чутливість управління
Hollow Knight	+	+	-	+	Стилізована 2D-анімація	Unity	Добре оптимізоване
Castlevania	-	-	-	-	Піксель-арт, ретро	Custom / Internal	Жорстке керування
Super Mario	-	-	-	-	Піксель-арт, ретро	Nintendo Proprietary	Просте, точне управління
Metroid	-	-	-	-	Піксель-арт, ретро	Nintendo Proprietary	Просте, точне управління
Dead Cells	+	+	-	+	Піксель-арт з ефектами	Custom (Heaps)	Дуже чутливе і швидке
Parkour Kingdom	-	+	+	+	Піксель-арт	Unity	Просте, точне управління

5 ФОРМУЛЮВАННЯ ВИМОГ ДО 2D SIDE-SCROLLING ГРИ

Розробка 2D side-scrolling платформи вимагає чіткого визначення як функціональних, так і нефункціональних вимог до майбутнього програмного продукту. Коректне формулювання вимог дозволяє уникнути неоднозначностей на етапі розробки, забезпечити відповідність гри очікуванням користувачів та полегшити подальшу підтримку і розвиток проєкту.

5.1 Функціональні вимоги

Функціональні вимоги описують основні можливості гри, які мають бути реалізовані:

Реалізація управління персонажем: рух ліворуч/праворуч, стрибок, атака.

Рівні та локації**:**

Створення кількох ігрових рівнів із поступовим ускладненням.

Наявність чекпоінтів для збереження прогресу.

Звуковий супровід**:**

динамічний звуковий супровід.

Параметри доступності**:**

Наявність режиму високої контрастності.

Наявність режиму для людей із кольоровою сліпотою.

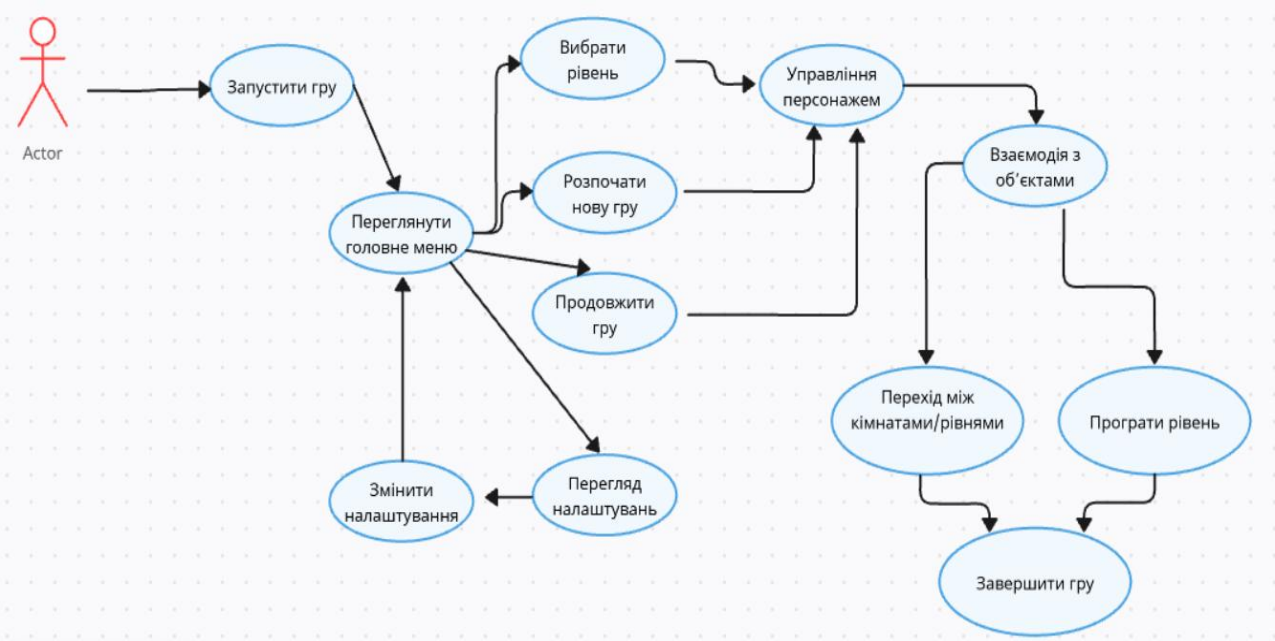


Рис. 5.1 Діаграма дій користувача

5.2 Нефункціональні вимоги

Нефункціональні вимоги визначають загальні характеристики системи:

Продуктивність**:**

Стабільна частота кадрів (FPS) не нижче 60 кадрів на секунду на базовому обладнанні (CPU 1 GHz, 2GB RAM, integrated GPU) для плавності геймплею.

Кросплатформеність**:**

Можливість запуску гри на Windows.

Якість графіки**:**

Використання 2D графіки з підтримкою адаптивної роздільної здатності.

Надійність**:**

Відсутність критичних помилок, що призводять до зависання або аварійного завершення гри.

Юзабіліті**:**

Інтуїтивно зрозумілий інтерфейс користувача.

Відповідність чіткості інтерфейсу основним стандартам.

Візуальна цілісність: єдиний графічний стиль для всіх елементів гри.

Швидкий час завантаження рівнів: перехід між сценами не перевищує 3-5 секунд.

Розширюваність**:**

Можливість легкого додавання нових рівнів та механік без суттєвої переробки базової архітектури.

Інтеграція системи доступності**:**

Можливість вмикання і вимикання режимів високої контрастності та режимів для кольорової сліпоти через меню налаштувань.

6 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ГРИ

Проєктування архітектури гри є критично важливим етапом розробки, що забезпечує структуровану організацію коду, полегшує масштабування проєкту, впровадження нових функціональних можливостей та технічне обслуговування. У межах розробки 2D side-scrolling гри особлива увага приділяється розділенню ігрової логіки, управління ресурсами, відображення інтерфейсу користувача та обробки фізики об'єктів.

6.1 Основні компоненти архітектури

Архітектура гри будується за модульним принципом і передбачає наявність наступних ключових підсистем:

Game Manager — головний контролер гри, який відповідає за зміну станів гри (початок рівня, пауза, завершення гри).

Player Controller — модуль для управління рухами та діями персонажа.

Level Manager — завантаження, ініціалізація та управління ігровими рівнями.

UI Manager — управління елементами інтерфейсу користувача, такими як меню, індикатори здоров’я, очок тощо.

Audio Manager — система керування музикою та звуковими ефектами, з підтримкою динамічного звукового супроводу.

Accessibility Manager — модуль налаштування параметрів доступності (режими високої контрастності, кольорова сліпота).

Save System — збереження і завантаження даних про прогрес гравця.

6.2 Взаємодія між компонентами

Всі основні компоненти взаємодіють через централізовану шину повідомлень (Event Bus), що забезпечує мінімальну залежність між модулями. Наприклад:

- Player Controller надсилає подію PlayerDamaged, на яку реагує UI Manager (оновлює індикатор здоров'я) та Audio Manager (відтворює звук ушкодження).
- Level Manager повідомляє Game Manager про завершення рівня, що тригерить перехід до наступного рівня або виведення екрану результатів.
- Accessibility Manager оновлює візуальні налаштування при зміні відповідних параметрів у меню.
- Такий підхід забезпечує масштабованість та гнучкість системи.

6.3 Структура класів

Основні класи та їх короткий опис:

Таблиця 6.1

Основні класи

Клас	Опис
GameManager	Керує глобальними станами гри.
PlayerController	Реалізує логіку руху та взаємодії персонажа.
LevelManager	Завантажує сцени та ініціалізує об'єкти рівня.

Продовження таблиці 6.1

Основні класи

Клас	Опис
UIManager	Керує відображенням інтерфейсу.
AudioManager	Забезпечує програвання звуків та зміну музичних тем.
AccessibilityManager	Реалізує візуальні налаштування для людей з порушенням зору.

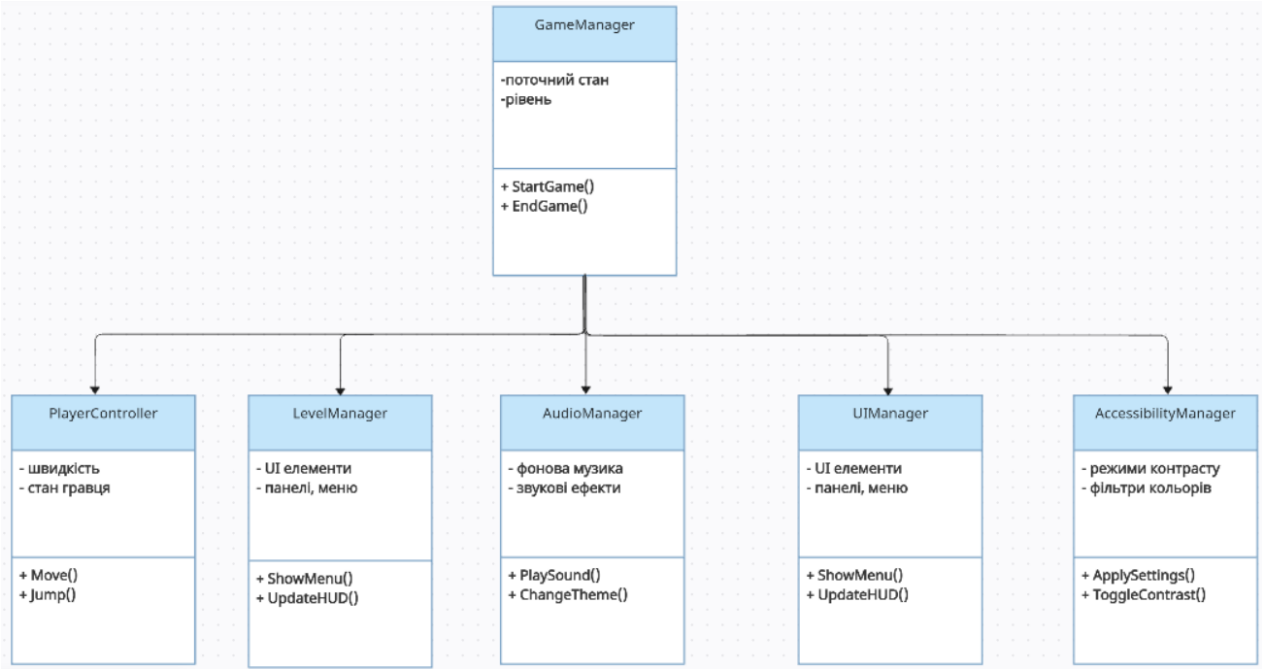


Рис. 6.1 Діаграма основних класів

6.4 Технічні рішення

Unity Engine (версія 2025 LTS) як основна платформа розробки.

Universal Render Pipeline (URP) для побудови графіки та налаштування режимів високої контрастності.

C# для реалізації скриптів і логіки гри.

Post Processing Stack для гнучкої обробки візуальних ефектів.

Імплементовані рішення, відсутні в аналогах

У процесі розробки гри було реалізовано низку функціональних рішень, які вирізняють розроблений продукт серед існуючих аналогів у жанрі 2D side-scrolling платформерів:

1. Система динамічного звукового супроводу

Розроблено систему яка дозволяє уникати повторення звукових ефектів, забезпечуючи збільшене різноманіття аудіо-вражень.

2. Налаштування режимів високої контрастності та кольорової сліпоти

З метою підвищення доступності гри для користувачів з порушеннями зору, реалізовано:

режим високої контрастності, який змінює палітру гри на поєднання кольорів із максимальним контрастом між фоном і активними об'єктами;

режими для різних типів кольорової сліпоти (дейтеранопія, протанопія, тританопія) із відповідною адаптацією кольорів інтерфейсу, платформ, ворогів та індикаторів.

Ці налаштування доступні в меню гри та можуть змінюватися в реальному часі без перезапуску. Такий рівень адаптивності є рідкістю серед 2D платформерів і не реалізований у більшості відомих представників жанру.

7 ПРОЄКТНІ РІШЕННЯ

На етапі початкового проєктування були прийняті ключові рішення щодо технічної та художньої реалізації гри. Вибір кожного елемента ґрунтується на аналізі вимог до гри, досвіді створення подібних продуктів та технічних можливостях доступних інструментів.

7.1 Вибір ігрового рушія

Для реалізації гри обрано ігровий рушій Unity, що забезпечує широкі можливості для створення 2D-проєктів. Основними аргументами на користь вибору стали:

- підтримка 2D-функціоналу на рівні рушія (включаючи фізику, анімацію, спрайти);
- розвинуте середовище для швидкої розробки прототипів;
- велика кількість плагінів, документації та активна спільнота;
- можливість кросплатформенного експорту;
- підтримка Universal Render Pipeline (URP), що дозволяє реалізовувати візуальні ефекти, у тому числі для потреб доступності.

7.2 Вигляд камери

Оскільки гра належить до жанру 2D side-scrolling platformer, обрано класичну перспективу бічного огляду (side-view). Такий вигляд:

- забезпечує максимальну наочність ігрового простору;
- дозволяє гравцю легко орієнтуватися у просторі;
- відповідає жанровим очікуванням цільової аудиторії;

- спрощує реалізацію взаємодії між об’єктами у двовимірному просторі.

7.3 Сетинг

Сюжетна та візуальна складова гри оформлена у фентезійному сетингу. Такий вибір дозволяє реалізувати:

- багатий дизайн рівнів з використанням магічних і казкових елементів;
- систему класів і супротивників з особливими здібностями;
- атмосферу пригод, що приваблює широку аудиторію гравців;
- розширені можливості для створення оригінальних механік і сюжетних гачків.

7.4 Проєктування локацій гри

Локації (кімнати, рівні) у 2D side-scrolling грі є основними структурними одиницями ігрового світу, які забезпечують поступове проходження, зростання складності, вивчення механік та сюжетний розвиток. Проєктування кімнат є важливим етапом, оскільки впливає на ігровий досвід, баланс та реіграбельність.

Геометрія та побудова кімнат (показано на рисунках 7.1, 7.2)

Кожна кімната проєктується з урахуванням:

- балансу порожнього та зайнятого простору;
- наявності декількох рівнів висоти (платформи, уступи);
- відсутності надмірної складності на ранніх етапах;
- чіткої візуальної ідентифікації меж кімнати та об’єктів.

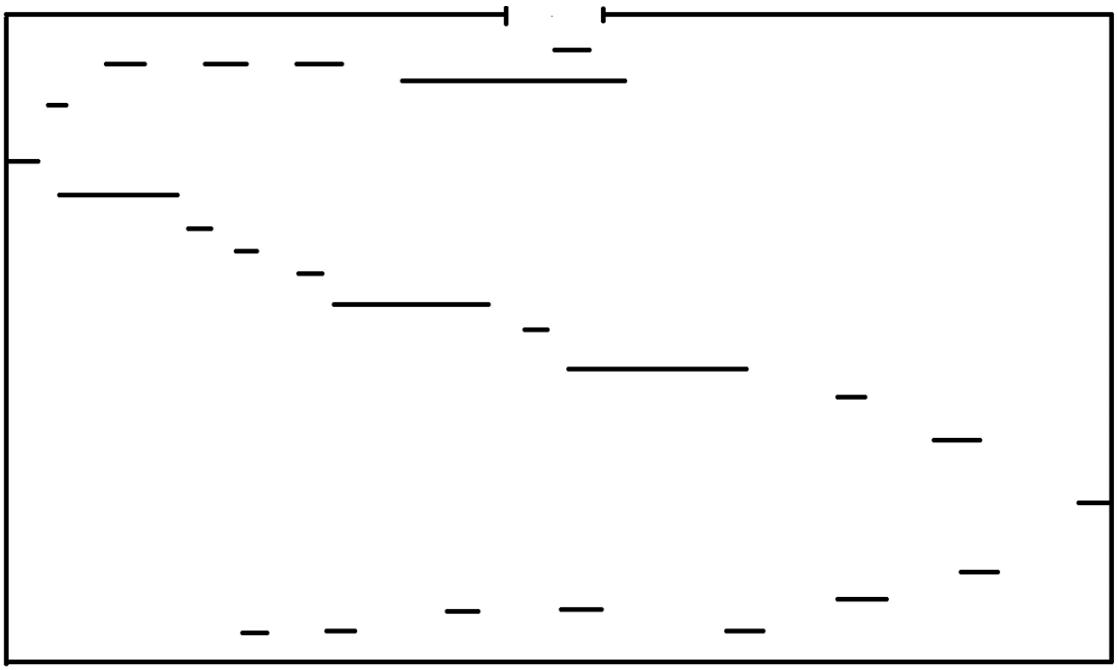


Рис. 7.1 Схема початкової кімнати

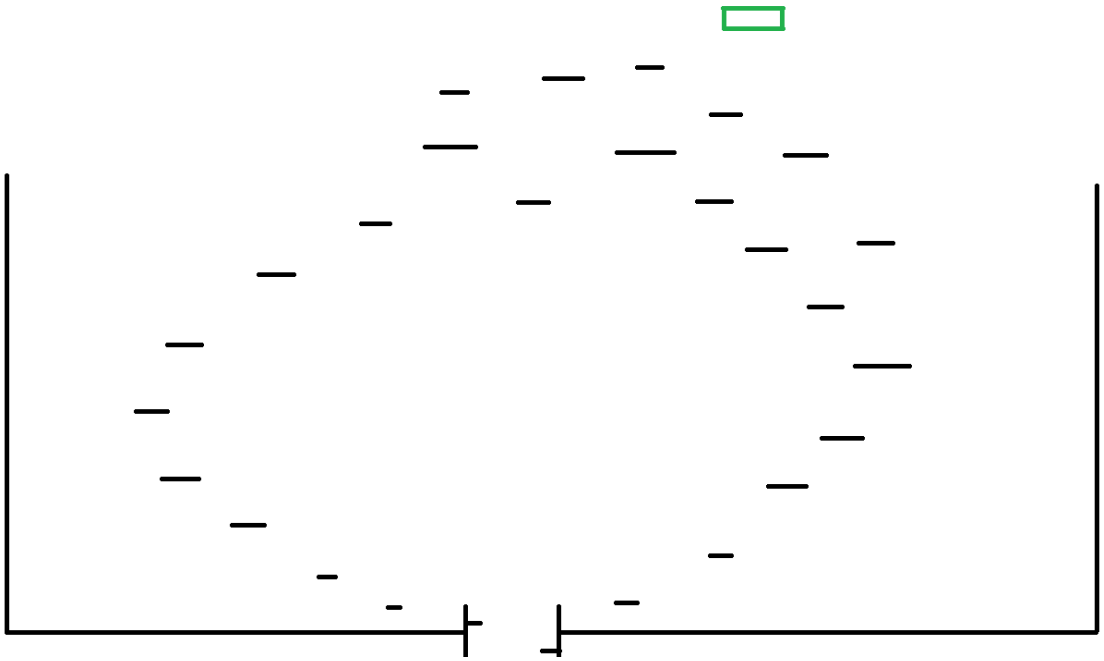


Рис. 7.1 Схема кімнати №2

Використано тайл-мапи (tilemaps) в Unity для створення модульних кімнат, що спрощує процес редагування, а також дозволяє динамічно генерувати варіації.

7.5 Переміщення між кімнатами

Забезпечено логіку переходу між кімнатами через тригер-зони. Завантаження здійснюється асинхронно, що дозволяє уникнути помітних затримок. Камера слідує за гравцем у межах поточної кімнати, з фіксацією положення при переходах.

7.6 Візуальна та геймплейна різноманітність

Для кожної кімнати передбачено унікальне оформлення, яке відображає тематичну зону (наприклад, ліс, підземелля, храм). Це досягається шляхом зміни фонів, кольорової палітри, освітлення та візуальних ефектів. Рівень складності підвищується поступово, за рахунок ускладнення геометрії, збільшення кількості ворогів та швидкості подій.

8 ВІЗУАЛЬНА СКЛАДОВА

Візуальне оформлення є ключовим елементом, що формує загальне враження від гри, створює атмосферу і сприяє зануренню гравця у віртуальний світ. У межах розробки 2D side-scrolling гри було обрано художній стиль, технічні засоби реалізації графіки та підхід до забезпечення візуальної доступності.

8.1 Стиль графіки

Для художнього оформлення проєкту обрано піксель-арт стиль, який поєднує ностальгічну естетику класичних ретро-платформерів із сучасними можливостями художнього дизайну. Піксельна графіка є ресурсоефективною, зручною для масштабування та легкою для створення авторських спрайтів з високим рівнем деталізації.

Кольорова палітра гри відповідає обраному фентезійному сетингу — переважають темні, глибокі тони з яскравими акцентами на об'єктах взаємодії, персонажах та ворожих елементах. На рисунках 8.1-8.7 показано вигляд готових рівнів у грі.

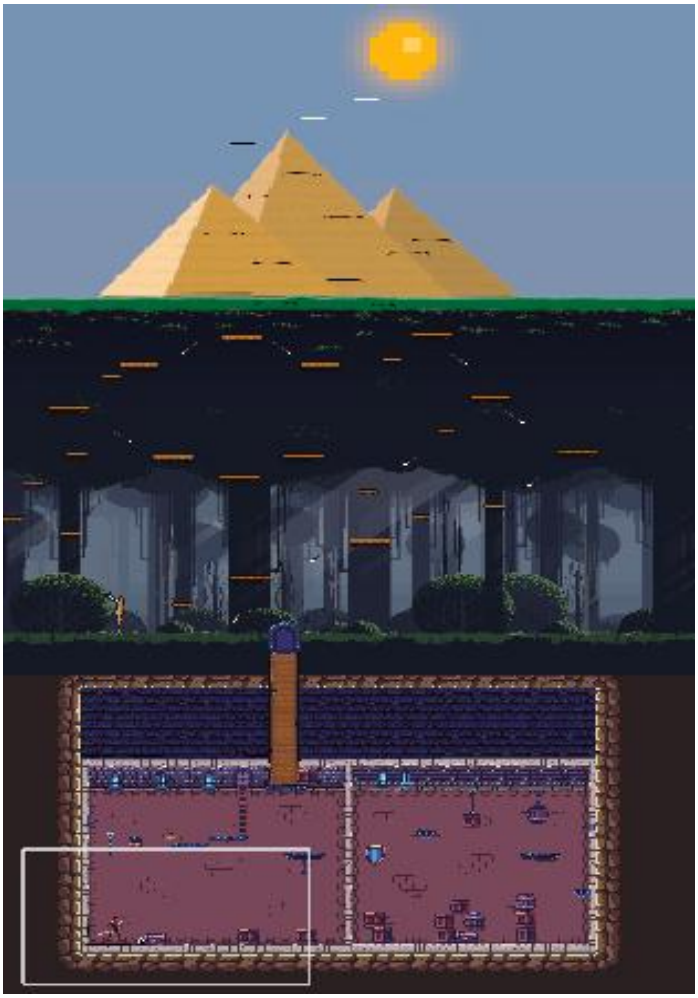


Рис. 8.1 Кімната №1,2

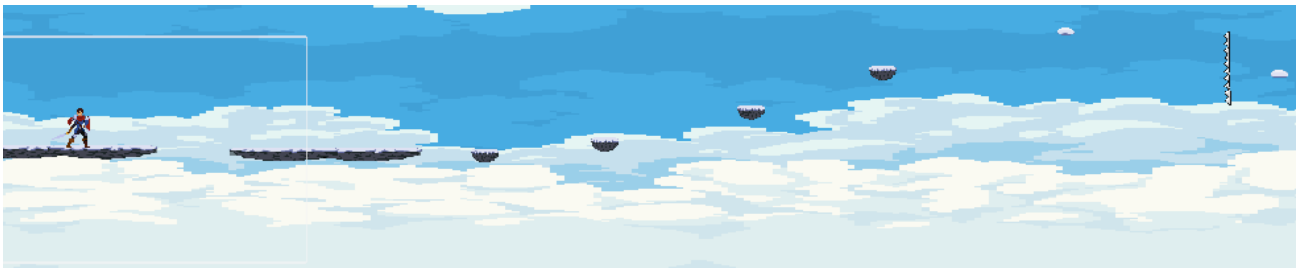


Рис. 8.2 Кімната №3

Рис. 8.3 Кімната №4

Рис. 8.4 Кімната №5

Рис. 8.5 Кімната №6

Рис. 8.6 Кімната №6

Рис. 8.7 Кімната №7

Для художнього оформлення проєкту обрано піксельну графіку, що поєднує естетику ігор з можливістю створення сучасних візуальних ефектів. Основні причини вибору:

- зменшення витрат на розробку та оптимізація ресурсоємності;
- висока впізнаваність стилю серед фанатів жанру;
- художня узгодженість із вибраним фентезійним сетингом;
- добрі можливості масштабування та деталізації на малих розширеннях.

8.2 Інтерфейс користувача

Інтерфейс реалізовано у вигляді мінімалістичного HUD (head-up display), який відображає лише необхідну інформацію: здоров’я персонажа, доступну кількість ресурсів (наприклад, життя, енергія тощо), та контекстні підказки.

Меню гри виконано в єдиному стилі з загальним оформленням і забезпечує логічну навігацію. На рисунках 8.1-8.3 показано меню гри.

Рис. 8.1 Головне меню

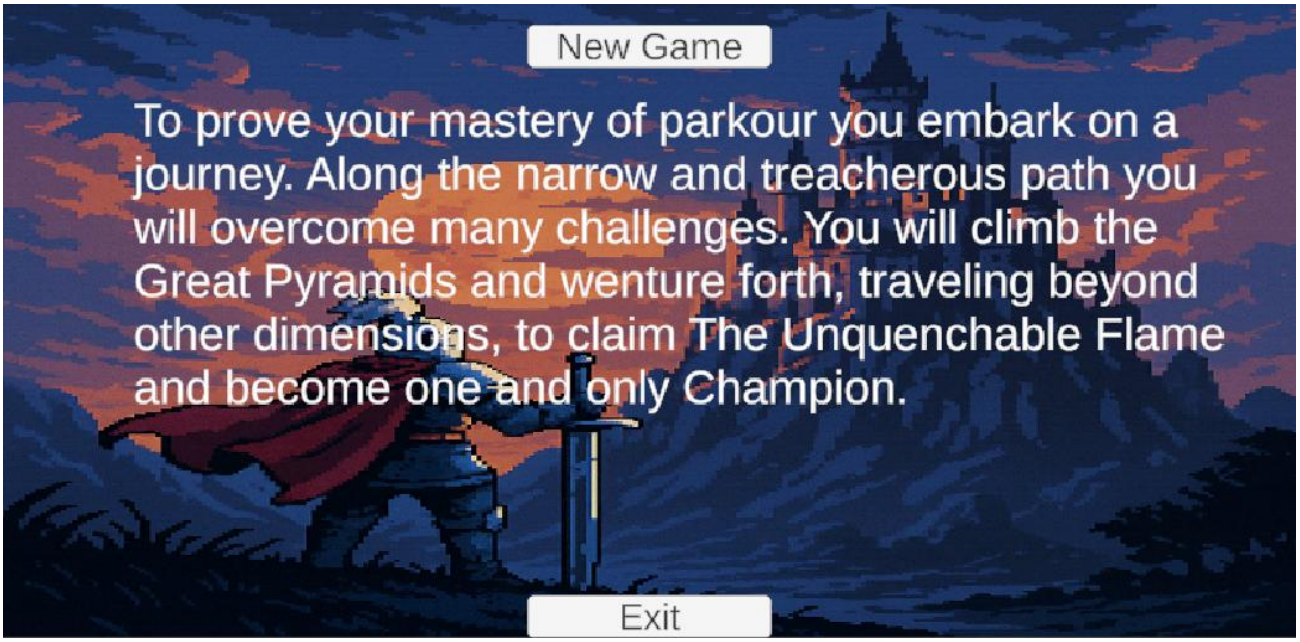


Рис. 8.2 Меню опису сюжету

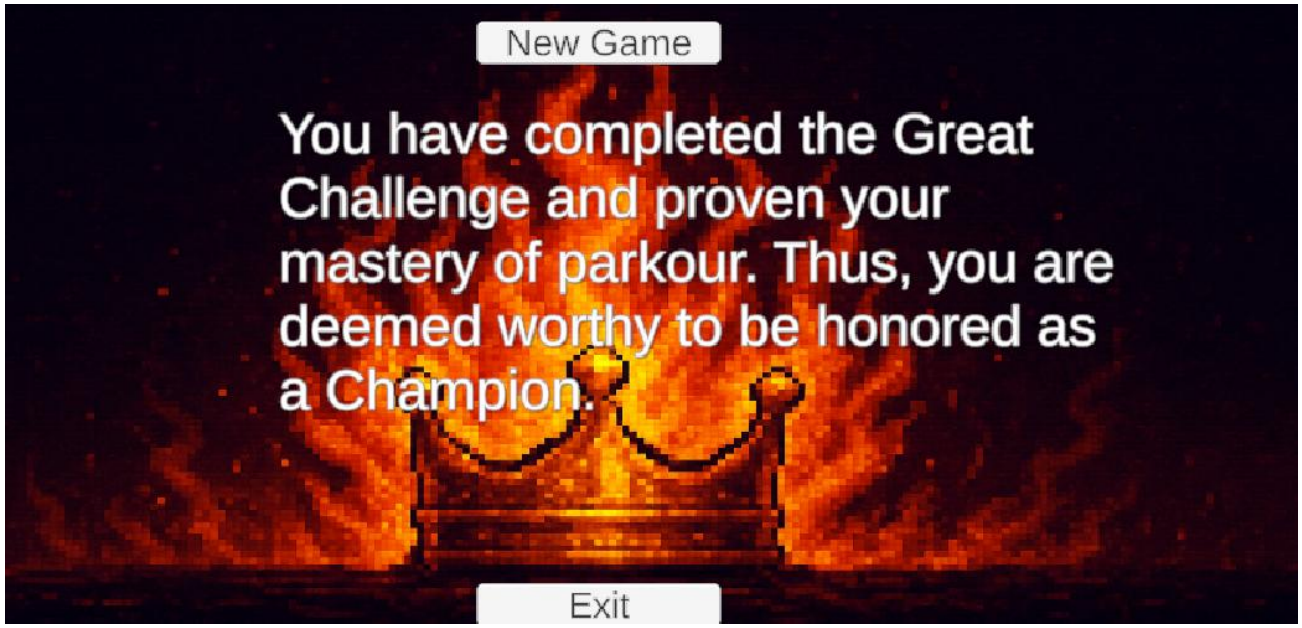


Рис. 8.3 Екран завершення гри

8.3 Анімація

Для головного героя, та інтерактивних об'єктів створено набір кадрових анімацій (sprite sheets), які включають стани ходьби, стрибка, атаки, смерті, взаємодії з оточенням тощо. Анімації синхронізовано зі звуковими ефектами та логікою керування.

8.4 Візуальні ефекти

Застосовано постобробку (Post Processing) та візуальні ефекти (VFX) для посилення ігрових подій: спалахи при атаках, частинки при взаємодії з середовищем. Ефекти не перевантажують екран і налаштовуються відповідно до вимог доступності.

8.5 Підтримка режимів доступності

У грі реалізовано режими високої контрастності та режими для користувачів із кольоровою сліпотою. Візуальні елементи автоматично адаптуються при зміні відповідного параметра в налаштуваннях:

- для високої контрастності збільшується різниця між фоном і активними елементами, підсилюються контури;
- для кольорової сліпоти (протанопія, дейтеранопія, тританопія) використовуються альтернативні палітри та іконографіка, не залежна від кольору.

9 ТЕСТУВАННЯ

Тестування є важливим етапом життєвого циклу програмного забезпечення, що дозволяє виявити та усунути помилки, а також перевірити відповідність реалізованої гри поставленим функціональним і нефункціональним вимогам. У межах даного проєкту застосовувалися методи ручного, модульного та інтеграційного тестування.

9.1 Цілі тестування

Основними цілями тестування розробленої гри є:

- перевірка коректності роботи основних ігрових механік;
- забезпечення стабільної роботи гри без збоїв;
- перевірка адаптації інтерфейсу під режим високої контрастності та режим для користувачів із кольоровою сліпотою;
- оцінка динамічного звукового супроводу відповідно до ігрових подій;
- перевірка зручності управління та логіки взаємодії з ігровим середовищем;
- оцінка продуктивності на різних пристроях (у межах доступних тестових платформ).

9.2 Методи тестування

У процесі тестування були використані такі підходи:

- Функціональне тестування — перевірка відповідності ключових функцій очікуваній поведінці: стрибки, атаки, взаємодія з об’єктами, навігація меню, зміна налаштувань.

- Модульне тестування — тестування окремих компонентів гри: контролера персонажа, системи колізій, логіки ворогів, скриптів інтерфейсу.

- Інтеграційне тестування — перевірка узгодженої роботи між окремими модулями, зокрема — між ігровою логікою, звуковою системою та візуальними ефектами.

- Тестування доступності — ручна перевірка коректності відображення кольорових режимів, читабельності тексту та контрасту елементів.

ВИСНОВКИ

1. Проаналізовано предметну область, сформульовано функціональні та нефункціональні вимоги, спроєктовано архітектуру гри, створено необхідну графіку та реалізовано програмну логіку.

2. Запропоновано та імплементовано унікальні функціональні рішення, відсутні в проаналізованих аналогах. Зокрема, реалізовано систему динамічного звукового супроводу, яка адаптується до ігрових подій, а також функції доступності — режими високої контрастності та кольорової сліпоти, що розширює потенційну аудиторію гри та покращує загальну зручність користування.

3. Реалізовано повноцінну 2D side-scrolling гру з елементами платформінгу, що зберігає класичні елементи жанру й водночас містить інноваційні компоненти. Використання рушія Unity забезпечило гнучкість, кросплатформеність та широкі можливості візуальної і звукової адаптації гри. Було досягнуто мети проєкту, що свідчить про ефективність обраної архітектури, інструментів і методів розробки.

4. Проведено тестування системи, підтверджено працездатність, стабільність її роботи та відповідність початковим вимогам.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Пономаренко В.О., Дібрівний О.А. Розробка та імплементація налаштувань для режиму високої контрастності та кольорової сліпоти в інтерактивних додатках. // Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». 24.04.2025, ДУІКТ, м. Київ, С. 269-273.

2. Пономаренко В.О., Дібрівний О.А. Розробка системи динамічного звукового супроводу для 2D платформера на Unity // Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». 24.04.2025, ДУІКТ, м. Київ, С. 273-276.

ПЕРЕЛІК ПОСИЛАНЬ:

1. Juul, J. *Half-Real: Video Games between Real Rules and Fictional Worlds.* Cambridge, MA: MIT Press, 2005. 248 с.
2. Nystrom, R. *Game Programming Patterns.* Genever Benning, 2014. 354 с.
3. Rollings, A., Morris, D. *Game Architecture and Design: A New Edition.* Indianapolis: New Riders Publishing, 2003. 652 с.
4. Adams, E., Dormans, J. *Game Mechanics: Advanced Game Design.* Berkeley, CA: New Riders, 2012. 352 с.
5. McShaffry, M., Graham, D. *Game Coding Complete* (4th ed.). Boston, MA: Course Technology PTR, 2013. 960 с.
6. Rabin, S. (Ed.). *Introduction to Game Development* (2nd ed.). Boston, MA: Charles River Media, 2010. 1000 с.
7. Fullerton, T. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games* (4th ed.). Boca Raton, FL: CRC Press, 2018. 600 с.
8. Schell, J. *The Art of Game Design: A Book of Lenses* (3rd ed.). Boca Raton, FL: CRC Press, 2019. 600 с.
9. Aarseth, E. Computer Game Studies, Year One. *Game Studies: The International Journal of Computer Game Research* (2001). URL: https://gamestudies.org/0101/editorial.html
10. Unity Technologies. *Unity Manual*. URL: https://docs.unity3d.com/Manual/index.html
11. Godot Engine. *Godot Documentation*. URL: https://docs.godotengine.org/en/stable/
12. Rollings, A., & Adams, E. *Andrew Rollings and Ernest Adams on Game Design*. Indianapolis: New Riders, 2003. 648 с.

13. Gregory, J. Game Engine Architecture. Boca Raton, FL: CRC Press, 2018. 1240 с.

14. Salen, K., Zimmerman, E. *Rules of Play: Game Design Fundamentals*. Cambridge, MA: MIT Press, 2004. 688 с.

15. Bierman, M. *Developing 2D Games with Unity: Independent Game Programming with C#*. Berkeley, CA: Apress, 2015. 233 с.

16. Goldstone, W. *Unity 2017 Game Development Essentials*. Birmingham: Packt Publishing, 2017. 530 с.

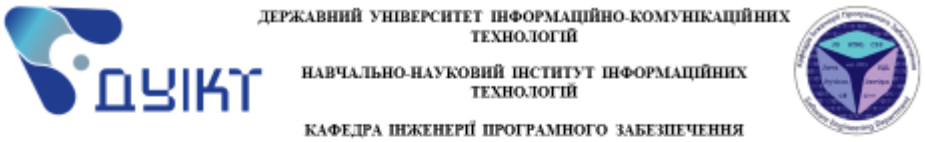
17. Feil, J., Scattergood, L. *Beginning Game Level Design*. Boston, MA: Cengage Learning PTR, 2005. 288 с.

18. Gibson, D. *Introduction to Game Design, Prototyping, and Development*. Boston, MA: Pearson Education, 2020. 880 с.

19. Totten, C. *Game Character Creation with Blender and Unity*. Hoboken, NJ: John Wiley & Sons, 2012. 352 с.

20. Novak, J. *Game Development Essentials: An Introduction*. Boston, MA: Delmar Cengage Learning, 2011. 544 с.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



Розробка відеогри в жанрі 2D side-scrolling

Виконав студент 4 курсу
Групи ПД-44
Пономаренко Владислав Олегович

Керівник роботи
Доктор філософії, доцент кафедри ІПЗ Дібрівний Олесь Андрійович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи — спроєктувати архітектуру гри, реалізувати функціональні рішення, відсутні в проаналізованих аналогах та підвищити зручність використання та доступності гри за рахунок реалізації інтуїтивного керування та налаштувань візуальної доступності.

Об'єкт дослідження — процес розробки інтерактивних двовимірних ігор у жанрі side-scrolling platformer.

Предмет дослідження — програмні засоби, методи та технології реалізації ігрових механік, анімації та взаємодії користувача з ігровим середовищем у 2D side-scrolling грі.

3

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Ознайомитися з технічною документацією та інструментами, що використовуються для створення 2D-ігор, зокрема середовищем Unity та мовою програмування C#.
2. Провести аналіз аналогічних ігор жанру 2D side-scrolling з метою виявлення сильних і слабких сторін існуючих рішень.
3. Сформулювати функціональні та нефункціональні вимоги до гри.
4. Розробити концепцію гри: визначити механіки, структуру рівнів та візуальний стиль.
5. Розробити базову ігрову логіку: рух персонажа, взаємодія з об'єктами середовища.
6. Реалізувати інтерфейс користувача (меню, HUD, інформаційні елементи).
7. Реалізувати вісім ігрових рівнів з можливістю переходу між ними.
8. Забезпечити збереження прогресу гри (запис/читання даних).
9. Провести тестування гри з метою виявлення та усунення помилок.

2

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

Реалізація головного меню та налаштувань.
Реалізація управління персонажем: рух ліворуч/праворуч, стрибок, атака.
Наявність восьми рівнів/локацій із переходами між ними.
Інтерфейс користувача (UI): відображення кількості життів, статусу гравця.
Анімації персонажів і середовища.
Фонове озвучення та звукові ефекти дій (стрибок, удар, пошкодження тощо).

Нефункціональні вимоги:

Підтримка запуску гри на Windows.
Чіткий інтерфейс користувача — інтуїтивно зрозуміле меню, інструкції та індикатори, відповідність принципам доступності.
Оптимізованість продуктивності та безперебійна робота — стабільна робота гри без зависань на базовому обладнанні (CPU 1 GHz, 2GB RAM, integrated GPU),захист від критичних помилок або некоректного завершення гри.
Масштабованість — можливість додавання нових рівнів, або механік у майбутньому.
Візуальна цілісність — єдиний графічний стиль для всіх елементів гри.
Швидкий час завантаження рівнів — перехід між сценами не перевищує 3–5 секунд.
Можливість адаптації для осіб з порушеннями зору (висококонтрастний режим та режим кольорової сліпоти).

4

АНАЛІЗ АНАЛОГІВ

Назва гри	Parallax Shader	Dynamic Audio	High contrast mode	Save system	Візуальний стиль	Двигун (Engine)	Особливості управління
Blasphemous	+	+	-	+	Піксель-арт, готичний	Unity	Висока чутливість управління
Hollow Knight	+	+	-	+	Стилізована 2D-анімація	Unity	Добре оптимізоване
Castlevania	-	+	-	-	Піксель-арт, ретро	Custom / Internal	Жорстке керування
Super Mario	-	-	-	-	Піксель-арт, ретро стиль	Nintendo Proprietary	Просте, точне управління
Metroid	-	-	-	-	Піксель-арт, ретро	Nintendo Proprietary	Просте, точне управління
Dead Cells	+	+	-	+	Піксель-арт з ефектами	Custom (Heaps)	Дуже чутливе і швидке
Parkour Kingdom	-	+	+	+	Піксель-арт	Unity	Просте, точне управління

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ОПИС ПРОГРАМИ

Розроблена програма є 2D side-scrolling платформером, що поєднує елементи пригодницької гри та паркуру. Гравець керує персонажем, який долає перешкоди, взаємодіє з навколишнім середовищем, та просувається крізь рівні з різною складністю. Ігровий процес побудований на класичних механіках платформерів, з акцентом на плавне управління, атмосферний звуковий супровід і доступність для користувачів із порушеннями зору.

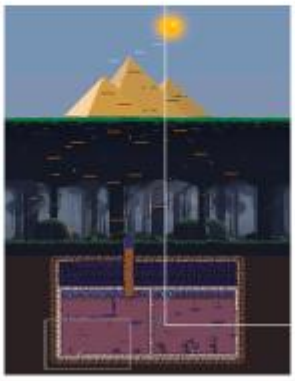
Технічні особливості

- Ігровий рушій: Unity з використанням Universal Render Pipeline.
- Мова програмування: C#.
- Графіка: піксель-арт стилістика
- Платформа: ПК (Windows)

7

ЕКРАННІ ФОРМИ

Перший ігровий рівень



Меню налаштуваннь додатку

8

ЕКРАННІ ФОРМИ

Зображення одного з рівнів

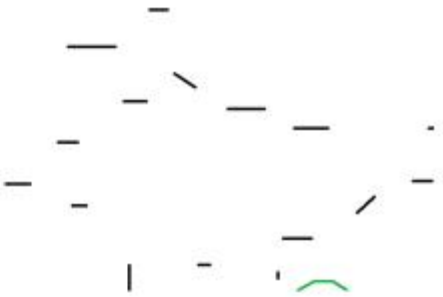
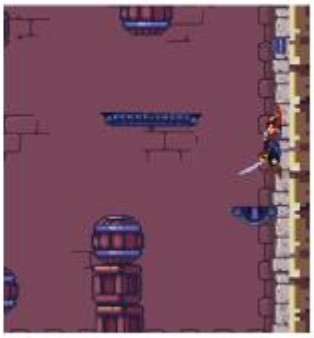


Схема платформ на ігровому рівні

9

ЕКРАННІ ФОРМИ

Головне меню



Анімація відштовхування від стіни

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Пономаренко В.О., Дібрівний О.А. Розробка та імплементація налаштувань для режиму високої контрастності та кольорової сліпоти в інтерактивних додатках: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ».С. 269-273. 24.04.2025, ДУІКТ, м. Київ

2. Пономаренко В.О., Дібрівний О.А. Розробка системи динамічного звукового супроводу для 2D платформера на Unity: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». С. 273-276. 24.04.2025, ДУІКТ, м. Київ

12

ВИСНОВКИ

1. Проаналізовано предметну область, сформульовано функціональні та нефункціональні вимоги, спроєктовано архітектуру гри, створено необхідну графіку та реалізовано програмну логіку.
2. Запропоновано та імплементовано унікальні функціональні рішення, відсутні в проаналізованих аналогах. Зокрема, реалізовано систему динамічного звукового супроводу, яка адаптується до ігрових подій, а також функції доступності — режими високої контрастності та кольорової сліпоти, що розширює потенційну аудиторію гри.
3. Реалізовано повноцінну 2D side-scrolling гру з елементами платформінгу, що відповідає класичним канонам жанру й водночас містить інноваційні компоненти. Використання рушія Unity забезпечило гнучкість, кросплатформеність та широкі можливості візуальної і звукової адаптації гри.
4. Проведено тестування системи, підтверджено працездатність, стабільність її роботи та відповідність початковим вимогам.

13

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
   public Canvas menuCanvas;



   public void Storyboard()
   {

GameManager.Instance.SaveProgress("Storyboard");
      SceneManager.LoadScene("Storyboard");
      Destroy(menuCanvas.gameObject);
   }

   public void ContinueGame()
   {
      GameManager.Instance.LoadSavedLevel();
   }

   public GameObject levelSelectPanel;

   public void OpenLevelSelect()
   {
      levelSelectPanel.SetActive(true);
   }

   public void CloseLevelSelect()
   {
      levelSelectPanel.SetActive(false);
   }

   public void LoadLevel(string levelName)
   {
      SceneManager.LoadScene(levelName);
   }

   public Canvas settingsCanvas;
   public void OpenSettings()
   {
      settingsCanvas.gameObject.SetActive(true);
      Debug.Log("Settings opened");
      // Відображення меню налаштувань
   }

   public void QuitGame()
   {
      Application.Quit();
      Debug.Log("Game Quit");
   }
}
```

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

public class FCanvasMenu : MonoBehaviour
{
   public Canvas fCanvasUI;

   public void NewGame()
   {

GameManager.Instance.SaveProgress("SampleScene");
      SceneManager.LoadScene("SampleScene");
      Destroy(fCanvasUI.gameObject);
   }

   public void QuitGame()
   {
      Application.Quit();
      Debug.Log("Game Quit");
   }
}
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
   public static GameManager Instance;

   private void Awake()
   {
      if (Instance == null)
      {
         Instance = this;
         DontDestroyOnLoad(gameObject);
      }
      else
      {
         Destroy(gameObject);
      }
   }

   public void SaveProgress(string sceneName)
   {
      PlayerPrefs.SetString("SavedLevel", sceneName);
      PlayerPrefs.Save();
      Debug.Log("Прогрес збережено: " +
sceneName);
   }

   public void LoadSavedLevel()
   {
      if (PlayerPrefs.HasKey("SavedLevel"))
```

```
        {
            string sceneToLoad =
PlayerPrefs.GetString("SavedLevel");
            SceneManager.LoadScene(sceneToLoad);
        }
        else
        {
            Debug.LogWarning("Немає збереженого
рівня");
        }
    }
}

using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.Universal;

public class AccessibilityManager : MonoBehaviour
{
    public Volume postProcessingVolume;

    private ColorAdjustments colorAdjustments;

    void Start()
    {
        if (postProcessingVolume.profile.TryGet(out
colorAdjustments))
        {
        }
    }

    public void SetHighContrast(bool enable)
    {
        if (colorAdjustments != null)
        {
            colorAdjustments.contrast.value = enable ? 50f :
0f;
            colorAdjustments.saturation.value = enable ? -
20f : 0f;
        }
    }

    public void SetColorBlindMode(string mode)
    {
        if (colorAdjustments == null) return;

        switch (mode)
        {
            case "deuteranopia":
                colorAdjustments.colorFilter.value = new
Color(0.8f, 1f, 0.8f);
                break;
            case "protanopia":
                colorAdjustments.colorFilter.value = new
Color(1f, 0.8f, 0.8f);
                break;
            case "tritanopia":
                colorAdjustments.colorFilter.value = new
Color(0.8f, 0.8f, 1f);
                break;
            default:
                colorAdjustments.colorFilter.value =
Color.white;
                break;
        }
    }
}
using UnityEngine;

public class PlayerMovementWithFootsteps :
MonoBehaviour
{
    public float moveSpeed = 5f;
    public Rigidbody2D rb;

    private Vector2 movement;
    private FootstepSoundPlayer footstepSound;

    private float stepTimer = 0f;
    public float stepRate = 0.4f;
    void Start()
    {
        footstepSound =
GameObject.Find("FootstepManager").GetComponent
<FootstepSoundPlayer>();
    }

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

        if (Mathf.Abs(movement.x) > 0.1f)
        {
            stepTimer += Time.deltaTime;
            if (stepTimer >= stepRate)
            {
                footstepSound.PlayFootstep();
                stepTimer = 0f;
            }
        }
        else
        {
            stepTimer = 0f;
        }
    }

    void FixedUpdate()
    {
        rb.velocity = new Vector2(movement.x *
moveSpeed, rb.velocity.y);
    }
}
using UnityEngine;

public class WallJump : MonoBehaviour
```

```
{
    public float moveSpeed = 5f;
    public float jumpForce = 12f;
    public float wallJumpForce = 10f;
    public float wallSlideSpeed = 1f;

    public Sensor_HeroKnight wallSensorLeft;
    public Sensor_HeroKnight wallSensorRight;
    public Sensor_HeroKnight groundSensor;

    private Rigidbody2D rb;
    private bool isGrounded;
    private bool isTouchingWall;
    private int wallDirection;

    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    private void Update()
    {
        float horizontal =
Input.GetAxisRaw("Horizontal");

        isGrounded = groundSensor.State();
        bool touchingLeftWall = wallSensorLeft.State();
        bool touchingRightWall =
wallSensorRight.State();
        isTouchingWall = (touchingLeftWall ||
touchingRightWall) && !isGrounded;
        wallDirection = touchingLeftWall ? 1 :
(touchingRightWall ? -1 : 0);

        rb.velocity = new Vector2(horizontal *
moveSpeed, rb.velocity.y);

        if (Input.GetButtonDown("Jump"))
        {
            if (isGrounded)
            {
                rb.velocity = new Vector2(rb.velocity.x,
jumpForce);
            }
            else if (isTouchingWall)
            {
                rb.velocity = new Vector2(wallDirection *
wallJumpForce, jumpForce);
            }
        }

        if (isTouchingWall && rb.velocity.y < 0)
        {
            rb.velocity = new Vector2(rb.velocity.x, -
wallSlideSpeed);
        }
    }
}
```

```
using UnityEngine;

public class CrushingTrap : MonoBehaviour
{
    public float topY;
    public float bottomY;
    public float speed = 2f;
    public float pauseTime = 1f;
    private bool movingDown = true;
    private float pauseTimer = 0f;

    void Update()
    {
        if (pauseTimer > 0f)
        {
            pauseTimer -= Time.deltaTime;
            return;
        }

        if (movingDown)
        {
            transform.position =
Vector3.MoveTowards(transform.position, new
Vector3(transform.position.x, bottomY,
transform.position.z), speed * Time.deltaTime);
            if (Mathf.Abs(transform.position.y - bottomY)
< 0.01f)
            {
                movingDown = false;
                pauseTimer = pauseTime;
            }
        }
        else
        {
            transform.position =
Vector3.MoveTowards(transform.position, new
Vector3(transform.position.x, topY,
transform.position.z), speed * Time.deltaTime);
            if (Mathf.Abs(transform.position.y - topY) <
0.01f)
            {
                movingDown = true;
                pauseTimer = pauseTime;
            }
        }
    }
}
using UnityEngine;

public class DisappearAndRespawnPlatform :
MonoBehaviour
{
    [SerializeField] private float disappearDelay = 1f;
    [SerializeField] private float respawnDelay = 3f;

    private Vector3 originalPosition;
    private Quaternion originalRotation;
    private bool hasTouched = false;
```

```
    private Collider2D col;
    private SpriteRenderer sr;

    void Start()
    {
        originalPosition = transform.position;
        originalRotation = transform.rotation;
        col = GetComponent<Collider2D>();
        sr = GetComponent<SpriteRenderer>();
    }

    private void OnCollisionEnter2D(Collision2D
collision)
    {
        if (!hasTouched &&
collision.gameObject.CompareTag("Player"))
        {
            hasTouched = true;
            Invoke(nameof(Disappear), disappearDelay);
        }
    }

    void Disappear()
    {
        sr.enabled = false;
        col.enabled = false;

        Invoke(nameof(Respawn), respawnDelay);
    }

    void Respawn()
    {
        transform.position = originalPosition;
        transform.rotation = originalRotation;

        sr.enabled = true;
        col.enabled = true;
        hasTouched = false;
    }
}
using UnityEngine;

public class FallingPlatform : MonoBehaviour
{
    public float fallDelay = 0.5f;
    public float returnDelay = 2f;
    public float fallDistance = 5f;
    public float fallSpeed = 2f;
    public float returnSpeed = 2f;

    private Vector3 startPos;
    private bool isFalling = false;
    private bool isReturning = false;
    private float fallTimer;
    private float returnTimer;

    void Start()
    {
        startPos = transform.position;
    }

    void Update()
    {
        if (isFalling)
        {
            transform.position =
Vector3.MoveTowards(transform.position, startPos -
Vector3.up * fallDistance, fallSpeed *
Time.deltaTime);

            if (Vector3.Distance(transform.position,
startPos - Vector3.up * fallDistance) < 0.01f)
            {
                isFalling = false;
                returnTimer = returnDelay;
            }
        }
        else if (isReturning)
        {
            transform.position =
Vector3.MoveTowards(transform.position, startPos,
returnSpeed * Time.deltaTime);

            if (Vector3.Distance(transform.position,
startPos) < 0.01f)
            {
                isReturning = false;
            }
        }
        else if (returnTimer > 0)
        {
            returnTimer -= Time.deltaTime;
            if (returnTimer <= 0)
            {
                isReturning = true;
            }
        }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.collider.CompareTag("Player"))
        {
            Invoke(nameof(StartFalling), fallDelay);
        }
    }

    void StartFalling()
    {
        isFalling = true;
        isReturning = false;
    }
}
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.Universal;
```

```
public class AccessibilityManager : MonoBehaviour
{
    public Volume postProcessingVolume;

    private ColorAdjustments colorAdjustments;

    void Start()
    {
        if (postProcessingVolume.profile.TryGet(out
colorAdjustments))
        {
        }
    }

    public void SetHighContrast(bool enable)
    {
        if (colorAdjustments != null)
        {
            colorAdjustments.contrast.value = enable ? 50f :
0f;
            colorAdjustments.saturation.value = enable ? -
20f : 0f;
        }
    }

    public void SetColorBlindMode(string mode)
    {
        if (colorAdjustments == null) return;

        switch (mode)
        {
            case "deuteranopia":
                colorAdjustments.colorFilter.value = new
Color(0.8f, 1f, 0.8f);
                break;
            case "protanopia":
                colorAdjustments.colorFilter.value = new
Color(1f, 0.8f, 0.8f);
                break;
            case "tritanopia":
                colorAdjustments.colorFilter.value = new
Color(0.8f, 0.8f, 1f);
                break;
            default:
                colorAdjustments.colorFilter.value =
Color.white;
                break;
        }
    }
}
using UnityEngine;
using UnityEngine.Audio;
using UnityEngine.UI;

public class VolumeSettings : MonoBehaviour
{
    public AudioMixer audioMixer;
    public Slider volumeSlider;

    void Start()
    {
        float savedVolume =
PlayerPrefs.GetFloat("MasterVolume", 0.75f);
        volumeSlider.value = savedVolume;
        SetVolume(savedVolume);
    }

    public void SetVolume(float volume)
    {
        audioMixer.SetFloat("MasterVolume",
Mathf.Log10(volume) * 20);
        PlayerPrefs.SetFloat("MasterVolume", volume);
    }
}
using UnityEngine;

public class DisappearAndRespawnPlatform :
MonoBehaviour
{
    [SerializeField] private float disappearDelay = 1f
[SerializeField] private float respawnDelay = 3f;
private Vector3 originalPosition;
    private Quaternion originalRotation;
    private bool hasTouched = false;
    private Collider2D col;
    private SpriteRenderer sr;

    void Start()
    {
        originalPosition = transform.position;
        originalRotation = transform.rotation;
        col = GetComponent<Collider2D>();
        sr = GetComponent<SpriteRenderer>();
    }

    private void OnCollisionEnter2D(Collision2D
collision)
    {
        if (!hasTouched &&
collision.gameObject.CompareTag("Player"))
        {
            hasTouched = true;
            Invoke(nameof(Disappear), disappearDelay);
        }
    }

    void Disappear()
    {
        sr.enabled = false;
        col.enabled = false;

        Invoke(nameof(Respawn), respawnDelay);
    }

    void Respawn()
```

```
    {
        transform.position = originalPosition;
        transform.rotation = originalRotation;

        sr.enabled = true;
        col.enabled = true;
        hasTouched = false;
    }
}
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenuManager : MonoBehaviour
{
    public GameObject pauseMenu;
    public GameObject settingsMenu;

    private bool isPaused = false;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (settingsMenu.activeSelf)
            {
                CloseSettings();
            }
            else
            {
                TogglePause();
            }
        }
    }

    public void TogglePause()
    {
        isPaused = !isPaused;
        pauseMenu.SetActive(isPaused);
        Time.timeScale = isPaused ? 0f : 1f;
    }

    public void OpenSettings()
    {
        settingsMenu.SetActive(true);
        pauseMenu.SetActive(false);
    }

    public void CloseSettings()
    {
        settingsMenu.SetActive(false);
        pauseMenu.SetActive(true);
    }

    public void QuitGame()
    {
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
#else
        Application.Quit();
#endif
    }
}
```

ДОДАТОК В. ТЕСТОВІ СЦЕНАРІЇ

Тестування головного меню гри

Тест №1 Функціональне тестування кнопок меню

Передумови: Гра запущена

Таблиця 1

Функціональне тестування кнопок меню

Дія	Очікуваний результат
Натиснути кнопку «Розпочати гру»	Відбувається перехід до першого рівня гри
Натиснути кнопку «Налаштування»	Відкривається меню налаштувань гри
Натиснути кнопку «Вихід»	Відбувається завершення роботи програми

Тестування ігрового процесу

Тест №2 Перевірка роботи кнопок меню

Передумови: Завантажений рівень гри

Таблиця 2

Перевірка роботи кнопок меню

Дія	Очікуваний результат
Натиснути клавішу «→» або	Персонаж рухається праворуч
Натиснути клавішу «←»	Персонаж рухається ліворуч
Натиснути клавішу «пробіл»	Персонаж виконує стрибок
Персонаж падає у прірву	Відбувається перезапуск рівня або втрата життя

Тестування системи налаштувань доступності

Тест №3 Перевірка зміни візуальних режимів (висока контрастність, дальтонізм)

Передумови: Відкрите меню налаштувань

Таблиця 3

Тестування системи налаштувань доступності

Дія	Очікуваний результат
Увімкнути режим високої контрастності	Інтерфейс гри переходить у контрастний режим
Увімкнути режим дальтонізму	Кольорова палітра змінюється відповідно до обраного режиму
Вимкнути обидва режими	Інтерфейс повертається до стандартного вигляду