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

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка 2D-платформера «Blade of Destiny» засобами Unreal Engine»

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*

_____ Михайло КУШНІР
*(підпис)*

Виконав: здобувач вищої освіти групи ПД-42
Михайло КУШНІР

Керівник: Максим КУКЛІНСЬКИЙ
*канд. техн. наук*

Рецензент: _____

**Київ 2026**

**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
« ____ » _____ 2026 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кушніру Михайлу Володимировичу

1. Тема кваліфікаційної роботи: «Розробка 2D-платформера «Blade of Destiny» засобами Unreal Engine 4»
керівник кваліфікаційної роботи канд. техн. наук Максим КУКЛІНСЬКИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку 2D-платформерів, принципи проєктування ігрових механік, методи організації ігрового процесу та ігрового циклу, технічна документація Unreal Engine 4, матеріали аналізу існуючих програмних продуктів та сучасних ігрових рушіїв.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз предметної галузі 2D-платформерів, дослідження сучасних підходів до розробки ігор, ігрових механік та особливостей організації ігрового процесу.
2. Проєктування 2D-платформера «Blade of Destiny», розробка структури гри, рівнів, ігрового циклу (core loop), та основних механік взаємодії.

3. Програмна реалізація та опис функціонування 2D-платформера «Blade of Destiny», використання Unreal Engine 4, мови програмування C++ та системи Blueprint для реалізації геймплейних механік і інтерфейсу.
4. Тестування програмного продукту, перевірка працездатності, аналіз результатів та оцінка якості реалізованих функцій.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до програмного забезпечення.
   3. Засоби реалізації.
   4. Діаграма ігрового процесу (core loop).
   5. Архітектура програмного продукту.
   6. Інтерфейс користувача (UI).
   7. Екранні форми гри.
   8. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Підбір та аналіз науково-технічної літератури з розробки 2D-платформерів | 23.02–28.02.2026 | |
| 2 | Аналіз та дослідження існуючих ігор-аналогів | 01.03–07.03.2026 | |
| 3 | Огляд ігрових механік, алгоритмів та технологій розробки 2D-ігор | 08.03–15.03.2026 | |
| 4 | Проєктування 2D-платформера «Blade of Destiny» (структура, рівні,core loop) | 16.03–27.03.2026 | |
| 5 | Програмна реалізація 2D-платформера засобами Unreal Engine 4 | 28.03–17.04.2026 | |
| 6 | Тестування програмного продукту | 18.04–27.04.2026 | |
| 7 | Оформлення роботи: вступ, висновки, реферат | 28.04–04.05.2026 | |
| 8 | Розробка демонстраційних матеріалів | 05.05–10.05.2026 | |
| 9 | Попередній захист роботи | 11.05–22.05.2026 | |

Здобувач вищої освіти _____ Михайло КУШНІР
*(підпис)*

Керівник
кваліфікаційної роботи _____ Максим КУКЛІНСЬКИЙ
*(підпис)*

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 66 стор., 5 табл., 23 рис., 23 джерел.

*Мета роботи* - підвищення якості ігрового процесу та покращення користувацького досвіду в 2D-платформері шляхом реалізації сучасних ігрових механік, системи взаємодії об'єктів та елементів розвитку персонажа.

*Об'єкт дослідження* – процес проєктування та розробки 2D-платформерів із використанням сучасних ігрових рушіїв.

*Предмет дослідження* - методи, технології та програмні засоби реалізації ігрових механік, системи рівнів, взаємодії об'єктів, ігрового циклу та інтерфейсу користувача в 2D-платформері засобами Unreal Engine 4, C++ і Blueprint.

*Короткий зміст роботи:* У роботі проаналізовано сучасні підходи до розробки 2D-платформерів, досліджено ігрові механіки та особливості побудови рівнів. Проведено аналіз існуючих ігор-аналогів та визначено їх переваги і недоліки. Розроблено концепцію гри «Blade of Destiny», що включає систему рівнів, ігровий цикл (core loop), систему прокачки персонажа та внутрішньоігровий магазин. Програмно реалізовано основні геймплейні механіки, інтерфейс користувача та взаємодію між компонентами гри.

Проведено тестування програмного продукту та оцінено його працездатність і якість реалізації функціональних можливостей.

*Сфера використання* - розважальні програмні продукти, навчальні проєкти з розробки ігор, а також подальше розширення функціоналу гри.

КЛЮЧОВІ СЛОВА: 2D-ПЛАТФОРМЕР, ІГРОВИЙ ПРОЦЕС, ГЕЙМПЛЕЙ, РІВНІ, Unreal Engine 4, BLUEPRINT

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

UI (User Interface) – інтерфейс користувача

HUD (Heads-Up Display) – елементи інтерфейсу, що відображаються під час гри (здоров’я, ресурси, показники)

FPS (Frames Per Second) – кількість кадрів за секунду

API (Application Programming Interface) – інтерфейс взаємодії програмних компонентів

IDE (Integrated Development Environment) – інтегроване середовище розробки

ООП – об’єктно-орієнтоване програмування

CPU (Central Processing Unit) – центральний процесор

GPU (Graphics Processing Unit) – графічний процесор

VFX (Visual Effects) – візуальні ефекти

SFX (Sound Effects) – звукові ефекти

# ВСТУП

*Актуальність:* Зараз 2D-платформери все ще залишаються популярними, бо вони прості, зрозумілі і швидко затягують у процес [1, 2]. Але коли дивишся на більшість таких ігор, стає помітно, що багато з них дуже схожі між собою. Проходження часто лінійне, механіки повторюються, і через це інтерес до гри поступово зникає. Проблема не в тому, що жанр поганий, а в тому, що його часто роблять занадто однаково. У результаті гру можна пройти один раз і більше до неї не повертатися. Саме тому виникає ідея зробити платформер трохи глибшим. Додати розвиток персонажа, можливість проходити рівні порізному, елементи персоналізації. Такі речі не ускладнюють гру просто заради складності, а роблять її цікавішою і дають більше причин повернутися. Тому тема є актуальною, оскільки вимоги до ігор ростуть, і гравцю вже недостатньо просто бігати і стрибати, хочеться більшого різноманіття і відчуття прогресу [1, 3].

*Об'єктом* дослідження є процес проєктування та розробки 2D-платформерів із використанням сучасних ігрових рушіїв.

*Предметом дослідження* є методи, технології та програмні засоби реалізації ігрових механік, системи рівнів, взаємодії об'єктів, ігрового циклу та інтерфейсу користувача в 2D-платформері засобами Unreal Engine 4, C++ і Blueprint.

*Метою роботи* є підвищення якості ігрового процесу та покращення користувацького досвіду в 2D-платформері шляхом реалізації сучасних ігрових механік, системи взаємодії об'єктів та елементів розвитку персонажа.

*Методи дослідження:* включали аналіз сучасних 2D-платформерів з урахуванням особливостей їхнього ігрового процесу, структури рівнів та механік взаємодії. Основна увага приділялася визначенню елементів, які позитивно впливають на сприйняття гри користувачем, а також недоліків, що можуть знижувати зацікавленість під час проходження. Проведений аналіз дозволив визначити сильні та слабкі сторони жанру і врахувати їх під час створення власного проєкту. Під час розробки було використано Unreal Engine 4, мову програмування C++ та систему Blueprint [11-16]. Поєднання цих технологій

дозволило забезпечити достатній рівень контролю над логікою гри та спростити процес тестування й внесення змін до окремих механік. Розробка здійснювалася безпосередньо в середовищі рушія та включала реалізацію ігрових механік, інтерфейсу користувача та систем взаємодії.

Окрему увагу було приділено тестуванню програмного продукту [5]. Перевірка роботи механік здійснювалася після внесення змін до окремих елементів гри, що дозволило поступово покращувати стабільність роботи та загальну якість реалізації проєкту.

*Наукова новизна роботи* полягає у поєднанні класичних механік 2D-платформера із сучасними підходами до організації ігрового процесу. Зокрема, у проєкті реалізовано систему розвитку персонажа, елементи варіативного проходження та більш зрозумілий інтерфейс користувача, що дозволяє покращити загальне сприйняття гри та підвищити рівень зацікавленості користувача.

*Практична значущість результатів* полягає у створенні працездатного програмного продукту, який може бути основою для подальшого розвитку та вдосконалення. Розроблена гра дозволяє додавати нові рівні, механіки та елементи взаємодії без необхідності повної зміни структури проєкту. Отримані результати можуть бути використані як основа для подальшої розробки повноцінного ігрового продукту.

# 1. ОГЛЯД ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 2D-ПЛАТФОРМЕРІВ

Комп’ютерні ігри є однією з найбільш динамічних сфер сучасної індустрії інформаційних технологій. Розвиток ігрових рушіїв, інструментів програмування та графічних технологій значно спростив процес створення ігор, що сприяло активному розвитку незалежних розробників та невеликих команд. Серед великої кількості ігрових жанрів 2D-платформери займають особливе місце, оскільки поєднують відносну простоту реалізації з широкими можливостями для створення різноманітного ігрового процесу [1, 2].

2D-платформери є жанром ігор, у якому користувач керує персонажем, що переміщується між платформами, долає перешкоди, взаємодіє з ворогами та виконує поставлені ігрові завдання. Основою більшості ігрових механік у даному жанрі є рух персонажа зокрема: біг, стрибки та взаємодія з навколишніми об’єктами. Саме якість реалізації цих механік значною мірою впливає на загальне сприйняття гри користувачем. Сучасні 2D-платформери активно розвиваються та доповнюються новими елементами, серед яких система розвитку персонажа, внутрішньоігрова економіка, альтернативні маршрути проходження та різні варіанти завершення гри [3, 5].

Аналіз існуючих рішень показує, що значна частина ігор даного жанру має певні недоліки, серед яких лінійність проходження, обмежена варіативність геймплею, складність балансування та недостатній рівень персоналізації. Такі особливості негативно впливають на зацікавленість користувачів і знижують бажання повторно проходити гру.

Проєктування сучасних ігрових систем передбачає використання модульної архітектури, продуманого дизайну рівнів, системи прогресії персонажа та інтерактивного інтерфейсу користувача. Поєднання цих складових дає змогу створити гнучкий, збалансований і цікавий ігровий процес [1, 6, 8]. Предметна

галузь 2D-платформерів підтверджує доцільність створення програмного продукту, який поєднуватиме класичні механіки жанру із сучасними підходами до організації геймплею. Саме такі висновки стали основою для подальшої розробки гри «Blade of Destiny».

### 1.1 Розвиток жанру 2D-платформерів у сучасній ігровій індустрії

2D-платформери є одним із найстаріших жанрів комп'ютерних ігор, що сформувався ще на ранніх етапах розвитку ігрової індустрії [1, 2]. У 1980-х роках представники даного жанру вперше набули широкої популярності завдяки поєднанню простих і зрозумілих ігрових механік. Основою таких ігор було переміщення персонажа між платформами, подолання перешкод та уникнення небезпек. З розвитком тривимірної графіки інтерес користувачів до 2D-платформерів почав поступово знижуватись, оскільки більша увага приділялася 3D-іграм, проте розвиток незалежної інді-індустрії дозволив жанру знову привернути увагу гравців. Значну роль у цьому відіграло вдосконалення графічної складової, ускладнення ігрових механік та покращення взаємодії між елементами ігрового процесу.

Важливий вплив на розвиток жанру мали також сучасні ігрові рушії, які постійно вдосконалювалися та надавали розробникам більше можливостей для створення якісних проєктів. У 2D-платформерах почали активно поєднуватися елементи різних жанрів, зокрема рольових ігор, екшенів та пригодницьких ігор. Це дозволило зробити ігровий процес більш різноманітним, ускладнити структуру рівнів та підвищити загальну якість реалізації проєктів. Невеликі команди та окремі розробники отримали можливість реалізовувати власні проєкти без необхідності використання значних ресурсів. У результаті такі зміни забезпечили більш тривале залучення користувачів до проходження гри.

Впровадження систем прогресії персонажа, нелінійності проходження та різних варіантів завершення гри стало важливим етапом розвитку жанру [1, 3, 5].

Такі підходи дозволили підвищити рівень реіграбельності та зробити проходження більш варіативним.

Таким чином, жанр двовимірних платформерів пройшов складний шлях розвитку, від простих аркадних ігор до сучасних програмних продуктів із розвиненою структурою та широкими функціональними можливостями. Розвиток жанру характеризується поєднанням класичних механік із сучасними підходами до організації ігрового процесу. Саме такі підходи були враховані під час розробки гри «Blade of Destiny».

### 1.2 Основні підходи до реалізації ігрових механік

Ігрові механіки є однією з основних складових будь-якої гри, оскільки саме вони формують загальне сприйняття ігрового процесу користувачем. Під час практичної реалізації навіть незначні зміни у русі персонажа або механіці стрибка можуть суттєво впливати на відчуття від гри. Особливо це помітно у 2D-платформерах, де основою геймплею є рух персонажа зокрема: біг, стрибки та падіння. Саме від якості реалізації цих механік залежить комфортність проходження та загальне сприйняття ігрового процесу [1, 2].

Під час розробки було встановлено, що зміна окремих параметрів, таких як швидкість руху або сила гравітації, значно впливає на проходження рівнів та поведінку персонажа. Навіть незначні зміни можуть створювати ефект надто різкого або, навпаки, занадто плавного руху, що одразу помітно під час гри. Саме тому процес налаштування базових механік потребує постійного тестування та коригування.

Практика показала, що одним із найважливіших елементів у 2D-платформерах є система керування персонажем [9]. Навіть незначна затримка реакції або некоректна поведінка персонажа може негативно впливати на комфорт гри та викликати незадоволення користувача. У процесі розробки окремі механіки, зокрема стрибки та взаємодія з платформами, неодноразово перероблялися для досягнення більш природного керування.

Також було встановлено, що надмірне об’єднання різних механік у єдину систему ускладнює процес розробки та тестування [8]. Розділення систем руху, атаки та взаємодії з об’єктами дозволило спростити подальше внесення змін та покращило стабільність реалізації окремих компонентів гри.

Окрему увагу під час розробки було приділено додатковим ігровим механікам. З одного боку, використання бойової системи, бонусів та інтерактивних елементів дозволяє зробити ігровий процес більш різноманітним. З іншого боку, надмірна кількість механік може перевантажити гру та ускладнити її сприйняття користувачем. Саме тому частина додаткових ідей була відкладена з метою збереження цілісності та зрозумілості геймплею.

Важливим елементом 2D-платформерів є противники, оскільки саме вони значною мірою впливають на динаміку проходження рівнів [1, 3]. Навіть прості вороги здатні змінювати поведінку гравця, змушуючи його адаптуватися до ситуації та змінювати стиль проходження. Разом із цим надмірна складність або велика кількість противників може негативно впливати на баланс гри, тому їх реалізація потребує додаткового тестування та налаштування.

Для кращого розуміння структури взаємодії основних компонентів ігрового процесу було сформовано модель реалізації ігрових механік 2D-платформера, яка наведена на рисунку 1.1.

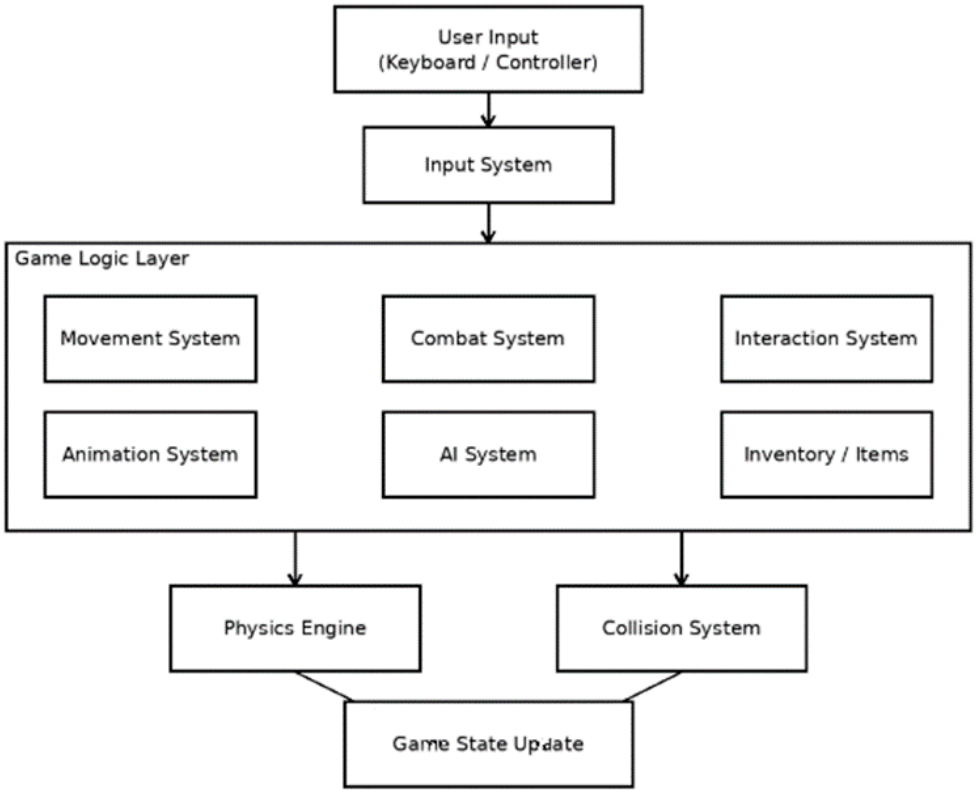



Рис. 1.1 Модель реалізації ігрових механік 2D-платформера

У підсумку складається відчуття, що хороший 2D-платформер це не про складні механіки, а про те, як вони працюють разом. Коли керування відчувається нормально, фізика не заважає, а рівень не викликає питань тоді гра відчуваєтсья коректно, і саме цього, як показує практика, досягти найскладніше.

### 1.3 Характеристика базових геймплейних елементів

Базові геймплейні елементи також є основою будь-якого 2D-платформера, оскільки саме навколо них формується загальний ігровий процес та взаємодія користувача з ігровим середовищем. До основних елементів належать персонаж гравця, взаємодія з оточенням, противники, інтерактивні об’єкти та інтерфейс користувача [1, 3]. Поєднання цих складових забезпечує цілісність і логічність геймплею.

Одним із ключових елементів є персонаж гравця, через якого здійснюється взаємодія з ігровим світом. Основними характеристиками персонажа є швидкість пересування, механіка стрибків, можливість атаки та взаємодія з навколишніми об’єктами. Важливу роль також відіграє передбачуваність керування, оскільки навіть незначна затримка реакції персонажа або некоректна поведінка під час руху одразу помітні під час гри та негативно впливають на загальне враження користувача [2].

Ігрове середовище, зокрема платформи та перешкоди, визначає структуру рівнів і формує логіку проходження. Платформи можуть бути статичними або рухомими, безпечними або небезпечними, що безпосередньо впливає на складність гри та темп проходження. Невдале проєктування рівнів може призвести до того, що гра стане надто простою або, навпаки, занадто складною, що негативно впливатиме на баланс ігрового процесу та зацікавленість користувача [2]. Структуру взаємодії базових геймплейних елементів 2D-платформера представлено на рисунку 1.2.

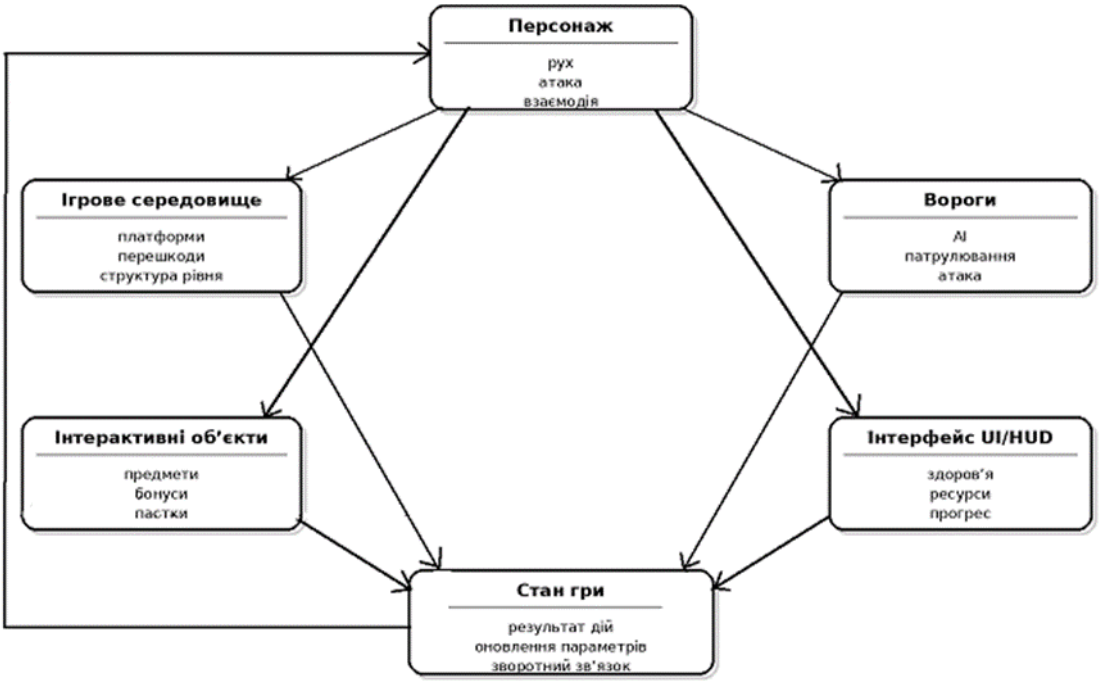



Рис. 1.2 Розширена модель взаємодії базових геймплейних елементів

Важливим елементом геймплею є противники, оскільки саме вони створюють динаміку та напруженість проходження. Вороги змушують гравця не лише рухатися вперед, а й приймати рішення щодо атаки, уникнення небезпеки або зміни маршруту [1]. Поведінка противників може реалізовуватися через прості сценарії патрулювання або переслідування, а також через складніші механіки взаємодії. Чим різноманітнішою є поведінка ворогів, тим більш цікавим і непередбачуваним стає ігровий процес.

На загальну логіку гри також впливають інтерактивні елементи. До них належать бонуси, ресурси, пастки, двері та механізми активації подій. Такі об’єкти доповнюють ігровий процес та забезпечують додаткові можливості взаємодії з оточенням. Їх використання дозволяє зробити рівні більш різноманітними та підтримувати зацікавленість гравця під час проходження.

Інтерфейс користувача виконує допоміжну, але важливу функцію в організації ігрового процесу. Саме через інтерфейс відображається інформація про стан персонажа, ресурси, показники та інші елементи, необхідні для прийняття рішень під час гри [9]. Важливим аспектом під час створення інтерфейсу є його зрозумілість та відсутність надмірного перевантаження інформацією, оскільки це може відволікати користувача від основного ігрового процесу.

Таким чином, базові геймплейні елементи утворюють взаємопов’язану систему, у якій кожен компонент виконує власну функцію. Правильне балансування взаємодії між цими елементами дозволяє створити цілісний, динамічний та цікавий ігровий процес.

## 1.4 Підходи до проєктування рівнів (Level Design)

Проєктування рівнів є одним із ключових етапів розробки 2D-платформера, оскільки саме структура рівня формує логіку проходження, темп гри та загальне сприйняття ігрового процесу. На початковому етапі може здаватися, що для створення рівня достатньо розмістити платформи та противників, проте вже після

перших тестувань зазвичай виявляються проблеми з балансом складності та логікою проходження [2]. Одні ділянки можуть бути занадто простими, тоді як інші, надмірно складними навіть у тих місцях, де це не планувалося.

Під час проєктування рівнів важливу роль відіграє поступове ускладнення ігрового процесу. На практиці більш ефективним є підхід, за допомогою якого спочатку гравцю надається можливість звикнути до керування та базових механік, а вже після цього вводяться складніші елементи проходження [1, 2]. Навіть за такого підходу окремі елементи рівня часто потребують повторного коригування та тестування для досягнення комфортного темпу гри.

У процесі розробки було встановлено, що навіть один невдало реалізований стрибок або неправильно розташований противник можуть негативно впливати на загальну структуру проходження та порушувати темп рівня. Відображення логіки проходження рівня було сформовано модель, наведену на рисунку 1.3.

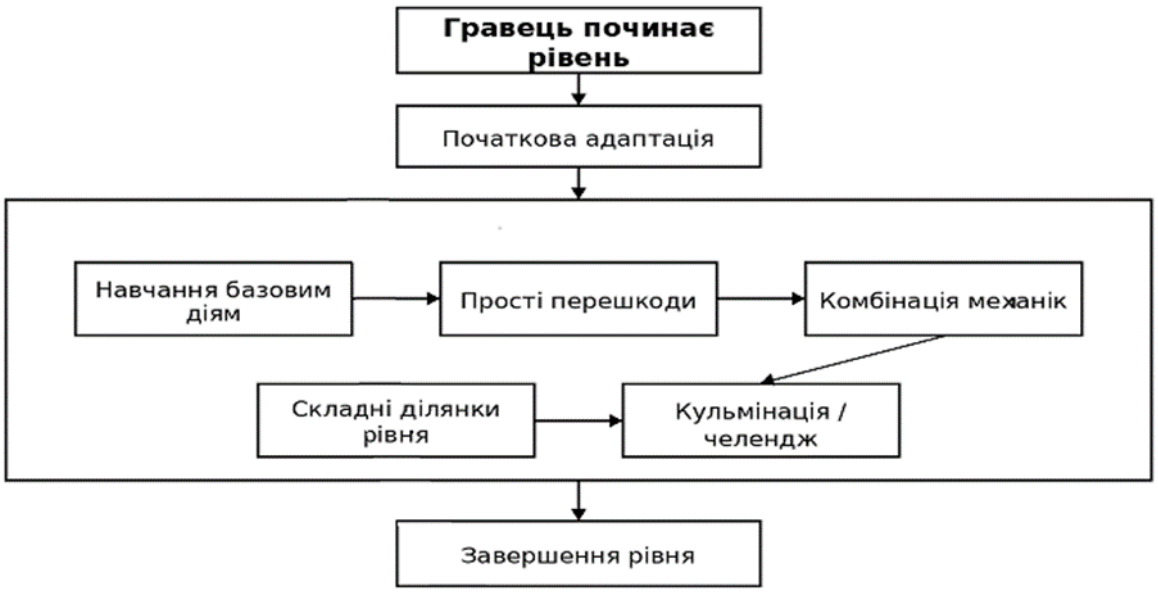



Рис. 1.3 Модель проходження рівня

Темп проходження є одним із найважливіших аспектів Level Design. Надмірна кількість подій або перешкод на короткій ділянці рівня може

перевантажувати користувача, тоді як великі порожні простори роблять проходження менш цікавим. Саме тому оцінка темпу здійснюється переважно під час практичного тестування гри.

Для спрощення контролю складності рівні доцільно розділяти на окремі функціональні частини. Зазвичай структура рівня складається з початкової зони, основної частини та фінального етапу проходження. Такий підхід дозволяє поступово вводити нові механіки та уникати хаотичної побудови рівня. На рисунку 1.4 представлено приклад поділу рівня на основні функціональні зони.

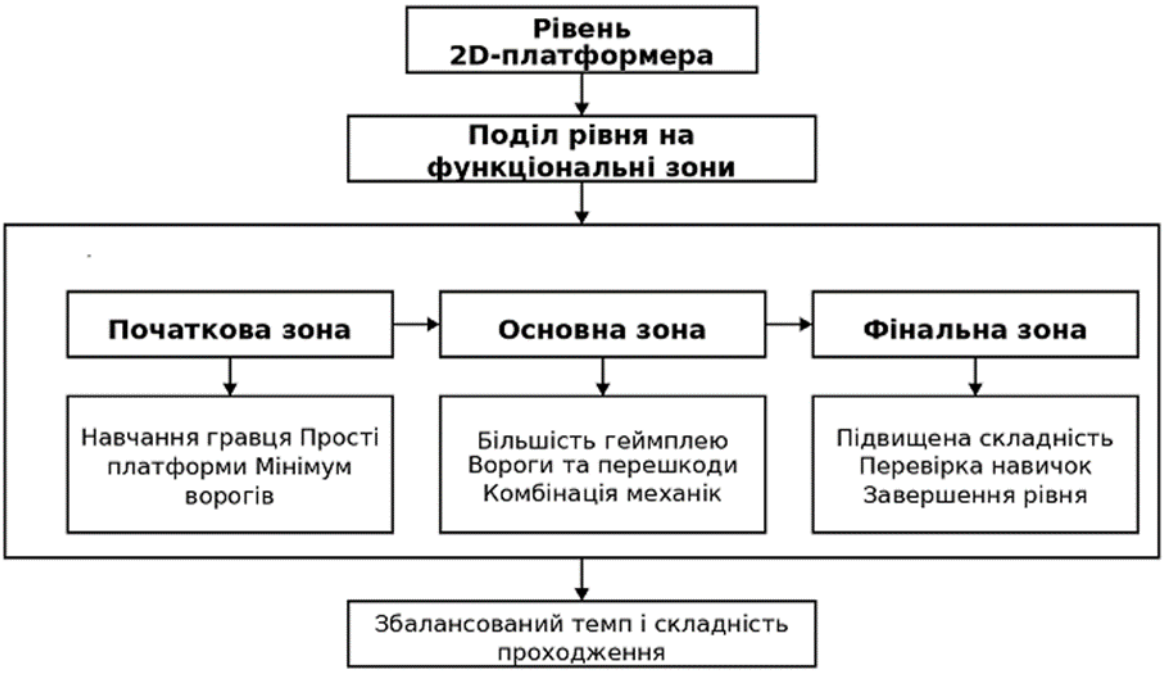



Рис. 1.4 Зональна структура рівня

Практика показала, що надто раннє введення складних елементів може дезорієнтувати гравця та ускладнити адаптацію до ігрового процесу. Саме тому початкові ділянки рівня зазвичай мають більш просту структуру, а складність підвищується поступово.

Важливим елементом сучасного проєктування рівнів є варіативність проходження. Спочатку один маршрут може здаватися достатнім, проте після тестування часто стає помітно, що надмірна лінійність знижує зацікавленість користувача. Навіть невеликі альтернативні маршрути або секретні зони

дозволяють зробити проходження більш різноманітним і стимулюють дослідження рівня [21].

Окрему увагу необхідно приділяти розташуванню платформ, противників та інтерактивних об’єктів. Якщо елементи рівня розміщені без логічного зв’язку, користувач починає помилятися не через складність проходження, а через недостатню зрозумілість ігрової ситуації. У деяких випадках навіть незначне зміщення об’єкта може суттєво покращити сприйняття рівня без необхідності додавання нових механік. Роль у структурі рівня також відіграють винагороди -бонуси, ресурси або чітко виражене завершення етапу формують у гравця відчуття прогресу та мотивацію до подальшого проходження [3]. Разом із цим надмірна кількість винагород може негативно впливати на баланс гри, тому їх використання потребує додаткового налаштування та тестування.

В таблиці 1.1 зібрані основні елементи, які використовуються при побудові рівня. Це скоріше орієнтир, ніж жорстке правило, але під час роботи допомагає не втрачати загальну логіку.

Таблиця 1.1

Основні елементи

| Елемент рівня | Опис | Призначення |
|---|---|---|
| Початкова зона | Частина рівня для ознайомлення з механіками | Навчання гравця та адаптація до керування |
| Основна зона | Основна частина з перешкодами та ворогами | Формування основного геймплею |
| Фінальна зона | Завершальна частина з підвищеною складністю | Перевірка навичок гравця |
| Платформи | Об’єкти для пересування персонажа | Реалізація руху та стрибків |
| Перешкоди | Пастки або складні елементи | Ускладнення проходження |

Продовження таблиці 1.1

Основні елементи

| | | |
|---|---|---|
| Вороги | Противники з різною поведінкою | Створення виклику для гравця |
| Бонуси / нагороди | Монети, предмети, покращення | Мотивація та винагорода |
| Альтернативні шляхи | Додаткові маршрути або секрети | Підвищення варіативності |
| Контрольні точки | Місця збереження прогресу | Полегшення проходження |
| Завершення рівня | Кінцева точка (фініш) | Логічне завершення гри |

Якщо говорити про «Blade of Destiny», то більшість цих речей довелося перевіряти саме на практиці. Деякі ділянки рівнів перероблювались по кілька разів через те, що вони не відчувались правильно. І це якраз той момент, який складно передати через теорію. У підсумку складається відчуття, що побудова рівня це більше про баланс і відчуття, ніж про якісь чіткі правила. Те, що виглядає добре на схемі, не завжди працює в грі і навпаки. Тому основна частина роботи  це постійна перевірка, правки і ще раз перевірка.

### 1.5 Проблеми та тенденції розвитку жанру

Аналіз сучасних 2D-платформерів показує, що універсальних рішень у межах даного жанру практично не існує. Більшість проєктів намагаються вирішити певні проблеми організації ігрового процесу, проте разом із цим нерідко створюють нові труднощі, пов’язані з балансом, навігацією або складністю проходження. Саме тому аналіз існуючих ігор-аналогів є важливою частиною процесу розробки. Одним із прикладів сучасного підходу до реалізації 2D-платформера є гра Hollow Knight. Її основною особливістю є високий рівень свободи дослідження ігрового світу. Гравець має можливість самостійно обирати

напрямок руху, знаходити приховані локації та досліджувати навколишнє середовище [21]. Разом із цим надмірна свобода може ускладнювати навігацію, особливо на початкових етапах гри, коли користувач ще не орієнтується в структурі світу.

Інший підхід демонструє гра Celeste, де основна увага приділяється точності керування та логічній побудові проходження [22]. У грі чітко визначені цілі та структура рівнів, що забезпечує зрозумілість ігрового процесу. Проте подібний підхід обмежує варіативність проходження, оскільки користувач переважно рухається за одним визначеним сценарієм.

Гра Magic Rampage поєднує елементи кількох жанрових підходів, додаючи до класичного платформера бойову систему, екіпіровку та елементи розвитку персонажа [23]. Це дозволяє зробити ігровий процес більш різноманітним, однак у процесі проходження може виникати відчуття повторюваності рівнів, оскільки механіки ускладнюються швидше, ніж змінюється саме ігрове середовище.

Під час розробки гри «Blade of Destiny» подібні проблеми також були враховані. Зокрема, спроби створити більш відкриту структуру рівнів призвели до ускладнення навігації, через що окремі елементи довелося спростити та зробити більш зрозумілими для користувача. Аналогічна ситуація виникла і з додатковими механіками, частину яких було вирішено не реалізовувати для збереження цілісності та зрозумілості ігрового процесу.

Таким чином, сучасний розвиток жанру 2D-платформерів характеризується прагненням до більшої гнучкості, варіативності та свободи дій гравця. Разом із цим зростає і складність процесу розробки, оскільки виникає необхідність балансування великої кількості взаємопов’язаних елементів [1-3]. Практика показує, що найбільш ефективним підходом є не максимальна кількість механік, а їх продумане та збалансоване поєднання. Саме такий підхід був використаний під час розробки гри «Blade of Destiny».

# 2. АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

На початковому етапі розробки основна увага була зосереджена безпосередньо на створенні гри «Blade of Destiny», без детального аналізу існуючих проєктів. Проте вже після перших етапів реалізації стало помітно, що більшість успішних 2D-платформерів мають продуману структуру ігрового процесу та механік [1, 3].

У процесі аналізу існуючих ігор увага почала приділятися не лише загальному враженню від проходження, а й окремим елементам реалізації. Зокрема, було помітно, що на сприйняття гри значно впливають структура рівнів, плавність керування, темп проходження та спосіб взаємодії користувача з ігровим середовищем. Одні рівні проходяться комфортно та логічно, тоді як інші можуть викликати складнощі навіть за схожої структури. Особливо це стає помітно під час практичної роботи над власним 2D-платформером. На перший погляд жанр виглядає відносно простим, оскільки його основу складають платформи, противники та механіки руху. Проте в процесі розробки стало зрозуміло, що навіть незначні зміни, такі як розташування платформ або налаштування швидкості руху персонажа, можуть суттєво впливати на сприйняття рівня та загальний комфорт проходження [2].

Саме тому перед подальшою розробкою було проведено детальніший аналіз існуючих ігор даного жанру. Це дозволило краще зрозуміти принципи побудови геймплею, організації рівнів та реалізації окремих механік. Після проведеного аналізу окремі рішення, використані під час створення «Blade of Destiny», були переглянуті та скориговані відповідно до отриманих висновків.

## 2.1 Аналіз ігор-аналогів

Після початку практичної розробки значення аналізу ігор-аналогів починає сприйматися інакше, ніж на початковому етапі. Спочатку аналіз інших проєктів може виглядати як формальна частина дослідження, проте під час створення власного прототипу стає помітно, наскільки велике значення мають окремі елементи реалізації геймплею та структури рівнів.

У процесі аналізу увага починає приділятися не лише загальному ігровому процесу, а й деталям організації гри. Зокрема, важливими стають способи побудови рівнів, введення нових механік, організація навігації та складності проходження. Багато рішень, які під час звичайного проходження здаються простими, на практиці виявляються значно складнішими з точки зору реалізації та балансування.

Під час розробки «Blade of Destiny» стало помітно, що окремі механіки та підходи, які в інших іграх виглядали зрозумілим, потребують значної кількості тестування та доопрацювання. Саме це підтвердило необхідність більш детального аналізу існуючих представників жанру. Для дослідження було обрано кілька різних 2D-платформерів, а саме Hollow Knight, Celeste та Magic Rampage, оскільки вони демонструють різні підходи до реалізації геймплею, структури рівнів та організації ігрового процесу [21-23]. Використання кількох прикладів дозволяє ширше оцінити особливості жанру та уникнути формування висновків лише на основі одного типу реалізації.

Для проведення більш структурованого аналізу були визначені основні критерії оцінювання:

- організація рівнів;
- механіки взаємодії;
- складність проходження;
- варіативність геймплею;
- спосіб подачі інформації користувачу.

Практичний аналіз показав, що саме ці складові найбільше впливають на загальне сприйняття гри та комфорт проходження користувачем.

### 2.1.1 Аналіз гри Hollow Knight

Якщо розглядати Hollow Knight як приклад сучасного 2D-платформера, то однією з перших особливостей, які помітні під час проходження, є високий рівень свободи дослідження. Гра не обмежує користувача одним маршрутом проходження, що суттєво впливає на загальне сприйняття ігрового процесу. Гравець має можливість самостійно обирати напрямок руху, досліджувати нові локації та відкривати приховані елементи світу. На початковому етапі це створює позитивне враження та підвищує зацікавленість до дослідження ігрового середовища. Разом із цим стає зрозуміло, що реалізація подібної свободи створює і певні труднощі. У деяких ситуаціях користувач може втрачати орієнтацію або не розуміти подальший напрямок проходження. На практиці підтримувати баланс між свободою дій та зрозумілістю навігації виявляється значно складніше, ніж це може здаватися на перший погляд [21].

Важливим аспектом гри є структура рівнів. На відміну від класичних платформерів, у Hollow Knight ігровий світ побудований як єдина взаємопов’язана система, у межах якої всі локації поєднані між собою [2]. Такий підхід створює відчуття цілісності світу та додає глибини дослідженню. Проте велика кількість переходів між зонами може ускладнювати навігацію, особливо під час повторного повернення до вже пройдених ділянок. Наявність прихованих елементів та секретних зон, стимулює дослідження рівнів та підтримує зацікавленість користувача під час проходження. Разом із цим частина контенту може залишатися непоміченою без додаткового дослідження, що також впливає на сприйняття гри.

З точки зору реалізації геймплею, важливу роль відіграє поєднання механік руху та бойової системи [1]. Гравець постійно змушений реагувати на зміну ситуації, комбінувати дії та адаптуватися до поведінки противників. Це робить ігровий процес більш динамічним і підтримує напруження навіть під час дослідження рівнів.

На рисунку 2.1 представлено приклад ігрового процесу гри Hollow Knight, що демонструє особливості побудови рівнів та взаємодії гравця з ігровим середовищем.


Рис. 2.1 Приклад ігрового процесу гри Hollow Knight

Під час аналізу також було помітно, що гра ефективно спрямовує користувача без використання прямих підказок або детальних інструкцій. Напрямок руху часто визначається через структуру рівнів, розташування об’єктів та особливості дизайну локацій. Подібний підхід дозволяє зберігати відчуття свободи, одночасно підтримуючи логіку проходження.

Таким чином, Hollow Knight демонструє ефективну реалізацію дослідження та нелінійності у 2D-платформерах. Разом із цим аналіз гри показує, що надмірна свобода без достатньо зрозумілої системи навігації може ускладнювати проходження та негативно впливати на загальне сприйняття гри користувачем.

### 2.1.2 Аналіз гри Celeste

Порівняно з Hollow Knight, гра Celeste використовує зовсім інший підхід до організації ігрового процесу. Уже з перших хвилин проходження помітно, що гра має чітко визначену структуру рівнів і практично не надає користувачу свободи вибору маршруту. Подібний підхід спрощує сприйняття гри, оскільки основна увага гравця зосереджується безпосередньо на проходженні рівнів та виконанні поставлених завдань.

Однією з головних особливостей Celeste є точність керування персонажем [9, 22]. Під час проходження добре відчувається контроль над рухом, а помилки користувача здебільшого залежать саме від правильності виконання дій, а не від некоректної роботи механік. Практика показує, що подібний рівень контролю є важливим фактором для комфортного проходження складних ділянок гри [22]. У Celeste механіки руху побудовані таким чином, що користувач швидко звикає до поведінки персонажа та починає орієнтуватися переважно на власну реакцію і навички.

Окрему увагу варто приділити системі поступового ускладнення механік [1]. Нові елементи вводяться послідовно: спочатку користувачу демонструється принцип роботи механіки, після чого складність поступово підвищується, а окремі елементи починають комбінуватися між собою. Завдяки цьому процес навчання відбувається природно та не перевантажує користувача надмірною кількістю нових елементів одночасно. Разом із цим гра має і певні обмеження. Основним із них є низький рівень варіативності проходження. Користувач рухається переважно за одним визначеним маршрутом, що робить повторне проходження менш різноманітним. Структура рівнів побудована у вигляді окремих випробувань, кожне з яких перевіряє конкретну навичку гравця. Подібний підхід дозволяє ефективно контролювати складність, проте частково обмежує свободу дослідження.

Інтерфейс користувача у грі реалізований максимально просто та не перевантажений додатковими елементами. Це дозволяє зосередити увагу безпосередньо на геймплеї. Разом із цим у грі практично відсутні додаткові

системи розвитку персонажа, екіпірування або внутрішньоігрової прогресії, оскільки основний акцент зроблено саме на проходженні рівнів.

Під час аналізу також було помічено, що Celeste ефективно працює із системою помилок користувача. Гравець часто припускається помилок під час проходження, проте швидке повернення до гри дозволяє підтримувати темп і не створює надмірного відчуття покарання за невдалі дії. Приклад реалізації геймплею та структури проходження гри Celeste наведено на рисунку 2.2.

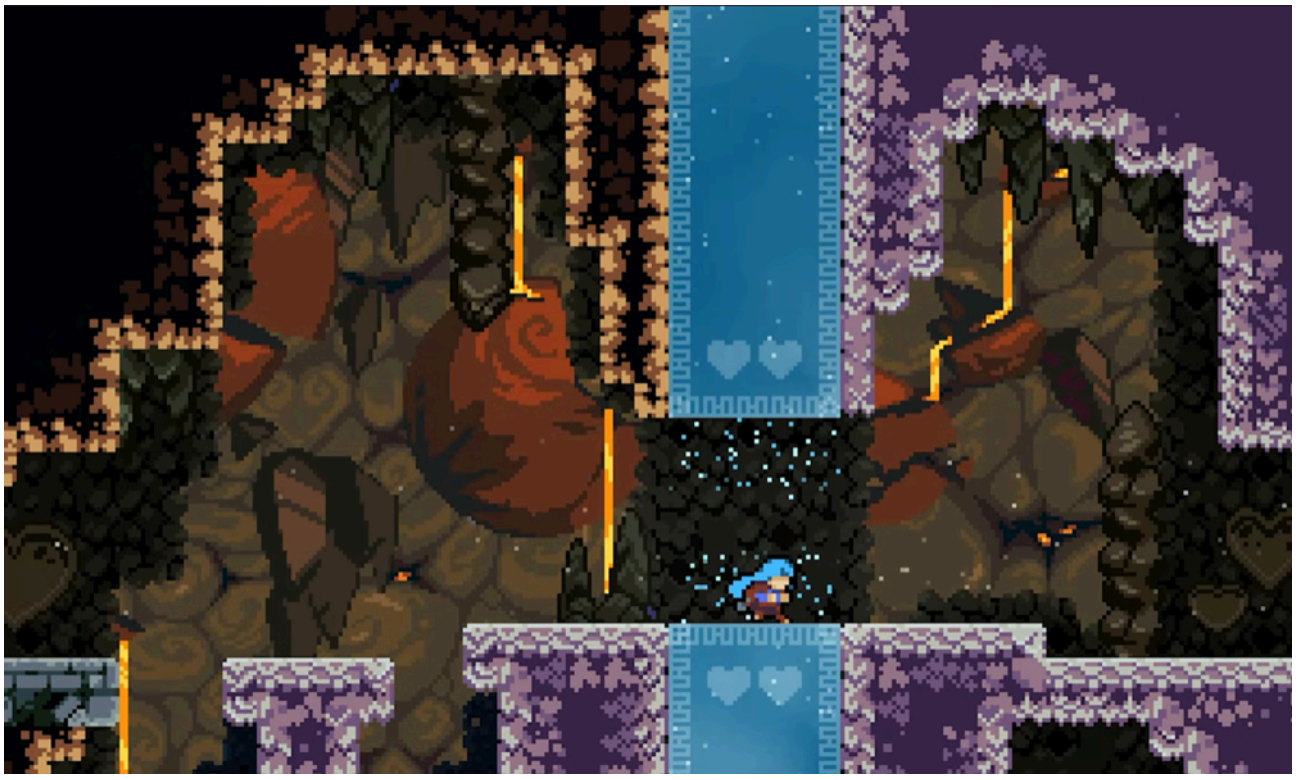


Рис. 2.2 Приклад ігрового процесу гри Celeste

Celeste демонструє ефективний підхід до реалізації точного та контрольованого геймплею. Незважаючи на обмежену свободу проходження, гра компенсує це якістю керування, продуманою структурою рівнів та поступовим ускладненням механік. Проведений аналіз показав, що навіть відносно прості механіки можуть забезпечувати комфортний та цікавий ігровий процес за умови правильної реалізації керування та балансу складності.

### 2.1.3 Аналіз гри Magic Rampage

Гра Magic Rampage демонструє підхід до розширення класичного 2D-платформера за рахунок використання додаткових ігрових механік. Основна увага приділяється не лише проходженню рівнів, а й системі розвитку персонажа, екіпіруванню та використанню ресурсів [3]. Подібний підхід помітно впливає на загальне сприйняття гри та створює додаткову мотивацію для проходження. Під час гри користувач поступово отримує нові предмети, покращує характеристики персонажа та відкриває додаткові можливості [23]. Це формує відчуття прогресу та підтримує зацікавленість навіть після проходження кількох рівнів. У результаті увага гравця концентрується не лише на самому проходженні, а й на розвитку персонажа та отриманні нових покращень.

Разом із цим під час аналізу рівнів стає помітно, що їх структура є відносно простою. У грі практично відсутні складні маршрути або велика кількість альтернативних шляхів проходження. Через це система розвитку персонажа іноді виглядає більш глибокою, ніж саме ігрове середовище. З одного боку, це робить гру більш доступною та зрозумілою для користувача, оскільки гравець рідше втрачає орієнтацію під час проходження. З іншого боку, з часом може виникати відчуття повторюваності рівнів, оскільки нові механіки не завжди компенсують схожість ігрових ситуацій.

Інтерфейс користувача, порівняно з класичними платформерами, він містить значно більше інформації, що пов’язано з наявністю систем екіпірування, ресурсів та характеристик персонажа. Проте під час активного ігрового процесу велика кількість елементів інтерфейсу може частково перевантажувати користувача, особливо в ситуаціях, коли необхідно одночасно контролювати дії персонажа та додаткові показники.

З точки зору розробки, реалізація подібних механік значно ускладнює процес балансування гри [5]. Необхідно враховувати не лише складність рівнів, а й характеристики персонажа, параметри екіпірування та систему розвитку. Навіть незначні зміни в силі персонажа можуть суттєво впливати на складність проходження. У результаті балансування потребує одночасного налаштування як

самих рівнів, так і всіх допоміжних систем гри. Також було помітно, що відчуття прогресу у Magic Rampage формується переважно через розвиток персонажа, а не лише через проходження рівнів. Це змінює загальну логіку геймплею та робить систему покращень одним із ключових елементів мотивації користувача.

Рисунок 2.3 демонструє приклад ігрового процесу гри Magic Rampage, що демонструє поєднання платформингових механік із системою розвитку персонажа та екіпірування.


Рис. 2.3 Приклад ігрового процесу гри Magic Rampage

Magic Rampage демонструє ефективний приклад розширення класичного платформера за допомогою додаткових ігрових систем. Разом із цим аналіз гри показує, що збільшення кількості механік значно ускладнює процес балансування та підтримання різноманітності ігрового процесу.

## 2.2 Порівняння функціональних можливостей

Після того як було розглянуто окремі ігри, перейдемо до їхнього порівняння. Саме під час такого аналізу формується більш цілісне розуміння особливостей жанру. Якщо розглядати кожен проєкт окремо, більшість рішень виглядають цілком виправданими. Проте при прямому порівнянні відмінності між іграми стають значно помітнішими. У деяких випадках механіки, які добре працюють в одній грі, в іншому контексті вже не сприймаються настільки ефективно. Саме тому виникла необхідність не лише описати окремі ігри, а й систематизувати їх за конкретними критеріями.

Для порівняння були вибрані саме ті характеристики, які найбільше впливають на те, як гра відчувається під час проходження. Серед них: архітектура геймплею, структура рівнів, складність, система розвитку персонажа, варіативність проходження, внутрішньоігрова економіка, персоналізація та інтерфейс. Уже під час аналізу стало помітно, що навіть схожі на перший погляд ігри можуть сприйматись зовсім порізному саме через ці речі.

Для більш наочного аналізу особливостей розглянутих ігор було сформовано порівняльну таблицю функціональних можливостей 2D-платформерів табл. 2.1.

Таблиця 2.1

Порівняння функціональних можливостей 2D-платформерів

| Критерій | Hollow Knight | Celeste | Magic Rampage | Blade of Destiny |
|---|---|---|---|---|
| Архітектура геймплею | Нелінійна | Лінійна | Комбінована | Комбінована |
| Структура рівнів | Складна, багаторівнева | Лінійна | Спрощена | Різнорівнева |
| Складність | Висока | Висока | Середня | Адаптивна |

Продовження таблиці 2.1

Порівняння функціональних можливостей 2D-платформерів

| Бойова система | Розвинена | Обмежена | Розширена | Збалансована |
|---|---|---|---|---|
| Система розвитку персонажа | + | − | + | + |
| Варіативність проходження | Висока | Низька | Середня | Висока |
| Наявність секретів | + | − | − | + |
| Внутрішньоігрова економіка | − | − | + | + |
| Персоналізація | − | − | + | + |
| Інтерфейс (HUD) | Мінімалістичний | Мінімалістичний | Насичений | Оптимізований |

Аналіз даних, наведених у таблиці 2.1, дозволяє зробити декілька важливих спостережень, які не завжди очевидні при окремому розгляді кожної гри [21-23].

У процесі порівняння різних ігор досить швидко починаєш помічати, наскільки порізному може будуватись один і той самий жанр. На перший погляд усі ці ігри звичайні 2D-платформери, але вже після кількох годин аналізу стає видно, що відчуваються вони зовсім не однаково.

Наприклад, у Hollow Knight найбільше запам’ятовується саме свобода. Гра майже не обмежує гравця і дозволяє самому вирішувати, куди рухатись далі. Спочатку це справді виглядає дуже цікаво, тому що з’являється відчуття дослідження світу. Але з часом починаєш помічати й іншу сторону такого підходу у деяких моментах легко втратити орієнтацію або просто не розуміти, куди потрібно йти далі. У Celeste ситуація майже протилежна. Там гра постійно тримає гравця в чітких рамках і фактично веде вперед, і у процесі проходження це навіть

відчувається комфортніше, тому що увага концентрується не на пошуку маршруту, а саме на геймплеї та точності керування. І саме в цей момент стає зрозуміло, що питання тут уже не в тому, який підхід правильний, а в тому, яке відчуття сама гра хоче дати гравцеві. Magic Rampage на фоні цих двох ігор виглядає як спроба знайти щось середнє. Тут уже додаються елементи розвитку персонажа, економіка, екіпіровка. Через це гра сприймається трохи інакше, тому що з’являється відчуття прогресу. Але під час аналізу стало помітно, що поєднати такі системи з платформінгом не так просто, як здається на початку.

Для «Blade of Destiny» цей етап порівняння виявився корисним. Після аналізу деякі рішення довелося переглядати повністю. Наприклад, стало зрозуміло, що повністю нелінійна структура може створити проблеми з навігацією, але й занадто жорстка лінійність теж робить гру менш цікавою. Саме тому в результаті був обраний більш комбінований підхід: залишити контроль над проходженням, але при цьому дати гравцю певну свободу.

Також було вирішено додати систему розвитку персонажа та елементи економіки, але не робити їх надто складними. Головна проблема це не придумати багато механік, а зробити так, щоб вони нормально працювали разом і не заважали одна одній [5, 8].

У підсумку це порівняння допомогло не просто побачити відмінності між іграми, а значно краще зрозуміти, як саме формується ігровий процес і чому навіть дрібні рішення можуть сильно впливати на загальне відчуття від гри.

### 2.3 Аналіз переваг та недоліків існуючих рішень

Цей підпункт виявився одним із найцікавіших, оскільки тут вже не просто аналізуються ігри та окремі механіки, а починаєш оцінювати, що працює добре, а що створює проблеми під час проходження. Наприклад, не одразу було зрозуміло, наскільки важливим є саме відчуття керування. Поки просто проходиш гру, це сприймається як щось природне. Але під час створення власного проєкту швидко стає помітно, що це одна з найскладніших частин у реалізації.

У таких іграх як Hollow Knight чи Celeste керування реалізоване дуже точно. Саме тому під час проходження користувач майже не задумується про сам рух персонажа, а концентрується безпосередньо на геймплеї. Навіть незначна неточність одразу починає відчуватись, і гра ніби втрачає цілісність, навіть якщо всі інші елементи працюють нормально. На початкових етапах цей момент значно недооцінювався.

Більш важливою під час розробки виявилась і структура рівнів. Раніше рівні сприймались як щось другорядне: достатньо розставити платформи, ворогів і перешкоди. Проте на практиці саме рівень визначає те, як буде відчуватись гра загалом. Добре побудований рівень здатний зробити цікавою навіть просту механіку. І навпаки - складні системи не рятують ситуацію, якщо сам простір організований хаотично або не підтримує темп проходження. У цьому аспекті Hollow Knight та Celeste є дуже показовими прикладами, хоча підходи до побудови рівнів у них суттєво відрізняються. Неоднозначною виявилась і система розвитку персонажа. Спочатку може здаватись, що сучасний платформер обов’язково повинен містити прокачку, економіку та додаткові механіки прогресії. Але після аналізу Celeste стає зрозуміло, що за умови правильно організованого геймплею гра може утримувати інтерес навіть без складної системи розвитку. Водночас у довгостроковій перспективі елементи прогресії все ж позитивно впливають на мотивацію користувача.

Окрему увагу привернули внутрішньоігрова економіка та персоналізація. У Magic Rampage ці системи дійсно додають варіативності. Проте тут виникає інша проблема: якщо економіка, екіпіровка або персоналізація не пов’язані безпосередньо з рівнями та основним геймплеєм, вони починають виглядати як окремі елементи, які існують самі по собі. Саме цей момент виявився одним із найскладніших у балансуванні.

Окремо варто згадати лінійність. У Celeste вона працює ефективно, оскільки вся структура гри побудована навколо цього підходу. Але при використанні подібної моделі в інших проєктах надмірна лінійність може почати обмежувати користувача та знижувати варіативність проходження. У цьому випадку

універсального рішення фактично не існує, оскільки все залежить від загальної концепції гри. Однією з найскладніших частин у розробці виявився баланс складності. Створити складну гру відносно просто, проте значно важче зробити так, щоб складність сприймалась як справедлива. Практика показує, що користувач готовий програвати та повторювати складні ділянки, якщо розуміє причину власної помилки. Якщо ж смерть або поразка виглядає випадковою чи нелогічною, це швидко викликає роздратування.

Також досить часто виникає проблема інтеграції різних систем між собою. У багатьох іграх механіки розвитку, економіка, інтерфейс або додаткові елементи існують окремо та не формують цілісну структуру. Через це навіть велика кількість можливостей не завжди робить гру більш цікавою.

Інтерфейс користувача, мінімалістичний HUD виглядає акуратно та не перевантажує екран, проте іноді користувачу може бракувати необхідної інформації. Надмірно насичений інтерфейс, навпаки, починає заважати сприйняттю геймплею. У результаті найбільш складним виявляється саме пошук балансу між простотою та інформативністю.

Сучасні 2D-платформери мають багато сильних сторін, але рідко поєднують їх одночасно в межах однієї гри. Саме це стало одним із головних висновків під час аналізу. У процесі розробки «Blade of Destiny» основний акцент поступово змістився не на кількість механік, а на їх збалансовану взаємодію, а саме:

- забезпечення стабільного керування;
- збереження варіативності без надмірного ускладнення;
- впровадження системи розвитку без перевантаження геймплею;
- інтеграція всіх основних механік у єдину систему.

У підсумку проведений аналіз дозволив не лише визначити переваги та недоліки існуючих рішень, а й сформувати більш цілісне розуміння того, як саме будується якісний ігровий процес.

## 2.4 Визначення підходів до реалізації геймплею

Під час аналізу ігор-аналогів поступово стало зрозуміло, що різні 2D-платформери вирішують однакові задачі. Одні більше зосереджуються на дослідженні світу, інші - на складності проходження або розвитку персонажа. Через це вже на етапі проєктування виникла потреба визначити, які саме елементи будуть використані у «Blade of Destiny» та як вони мають працювати разом. У процесі формування власної концепції основна увага приділялась не окремим механікам, а тому, як вони впливають на загальне сприйняття гри. Важливо було побудувати систему таким чином, щоб рівні, бойова система, дослідження та розвиток персонажа не виглядали відокремленими частинами, а формували єдиний ігровий процес. Саме тому подальший етап роботи був пов’язаний із визначенням основних вимог і підходів до реалізації програмного продукту.

*Комбінована архітектура геймплею*

На певному етапі розробки виникла ідея зробити структуру гри більш відкритою, приблизно як у Hollow Knight. Спочатку такий підхід виглядав досить цікавим, оскільки дозволяв надати користувачу більше свободи та зробити проходження менш передбачуваним. Проте в процесі роботи стало помітно, що разом із цією свободою з’являються й додаткові труднощі. Значно складніше стає контролювати темп гри, навігацію та загальну складність рівнів.

У результаті було обрано комбінований варіант. Основні рівні проходяться послідовно, але всередині них залишаються альтернативні маршрути, секретні ділянки та додаткові елементи дослідження. Такий підхід дозволяє зберігати контроль над структурою проходження, але водночас не робить гру надто прямолінійною.

*Проєктування ігрового циклу (Core Loop)*

На початку може здаватись, що гра будується навколо окремих механік. Але під час розробки поступово стає зрозуміло, що значно важливішим є загальний цикл дій, який постійно повторюється протягом проходження.

У межах проєкту цей цикл побудований таким чином:

- дослідження;
- взаємодія або бій;
- винагорода;
- розвиток;
- повернення до дослідження.

Саме така структура забезпечує постійний рух ігрового процесу. Якщо один із етапів працює слабше або випадає повністю, це одразу починає впливати на загальне сприйняття гри.

*Система руху та взаємодії*

Найбільша кількість експериментів під час розробки стосувалася саме системи руху. Спочатку рух персонажа може здаватись базовою механікою, але на практиці він визначає майже все, що пов’язано з відчуттям гри. Навіть незначні зміни швидкості пересування або сили гравітації суттєво впливають на сприйняття геймплею.

Окремі параметри доводилося неодноразово змінювати лише для того, щоб стрибки або переміщення відчувалися природно. Основним завданням було створення передбачуваного керування, за якого користувач чітко розуміє результат кожної дії. Взаємодія з об’єктами реалізована через систему подій.

*Бойова система як розширення платформінгу*

Бойова система не розглядалася як окремий елемент, який існує незалежно від основного геймплею. Саме тому бій інтегрований безпосередньо в систему руху. Під час взаємодії з противниками персонаж постійно переміщується, ухиляється від атак та використовує особливості рівня. Завдяки цьому бойова система сприймається як продовження платформінгу, а не окрема механіка.

*Внутрішньоігрова економіка та персоналізація*

На етапі проєктування стало зрозуміло, що внутрішньоігрова економіка може позитивно впливати на мотивацію користувача. Проте надмірне ускладнення подібних систем починає відволікати від основного геймплею. Саме тому

економіка реалізована у спрощеному вигляді: користувач збирає ресурси та використовує їх у магазині.

Персоналізація додана переважно як візуальний елемент. Вона не впливає критично на баланс гри, проте дозволяє зробити персонажа більш індивідуальним.

*Дизайн рівнів і контроль складності*

Побудова рівнів реалізована за принципом поступового ускладнення. Основна складність полягала не в самому підвищенні рівня складності, а в тому, щоб зробити його логічним та зрозумілим для користувача.

Система контрольних точок (checkpoints) була додана не відразу. Проте після тестування стало помітно, що без неї окремі ділянки викликають надмірну фрустрацію. Також були реалізовані додаткові опціональні секції для користувачів, які хочуть отримати більш складний ігровий досвід.

*Інтерфейс користувача (HUD)*

Під час створення інтерфейсу основна увага приділялася мінімізації зайвих елементів. На екрані залишено лише ту інформацію, яка дійсно необхідна під час проходження. Практика показала, що надлишок інформації швидше відволікає користувача, ніж допомагає йому. Саме тому другорядні елементи були перенесені в меню або паузу.

*Модульність і масштабованість*

У процесі розробки стало очевидно, наскільки важливою є структура коду та логіки гри. Якщо всі механіки реалізовані в межах однієї системи, подальше внесення змін швидко стає складним. Саме тому було використано модульний підхід: окремо реалізовано рух, бойову систему, інтерфейс та інші компоненти.
Такий підхід значно спрощує подальше тестування, редагування та масштабування проєкту.

Загалом більшість підходів сформувалися поступово вже в процесі розробки, після тестування окремих механік та виправлення помилок. Практичний досвід показав, що основне значення має не кількість функціональних можливостей, а те, наскільки добре всі системи працюють разом.

## 2.5 Формування вимог до програмного продукту

Після аналізу існуючих ігор-аналогів і перших етапів розробки стало зрозуміло, що подальше створення гри потребує більш чіткої структури. На початкових етапах багато рішень перевірялися практично: окремі механіки тестувалися, змінювалися або повністю перероблялися залежно від того, як вони сприймалися безпосередньо під час гри. Проте з часом стало помітно, що без визначених вимог проєкт починає поступово втрачати цілісність, а окремі системи ускладнюються більше, ніж це необхідно. Саме тому виникла потреба сформувати набір вимог, які визначають не лише основні можливості гри, а й загальні межі подальшої розробки. Такий підхід дозволяє краще контролювати структуру проєкту, підтримувати узгодженість механік і спрощує процес реалізації нових елементів.

Традиційно вимоги до програмного продукту поділяються на функціональні та нефункціональні. Функціональні вимоги описують можливості гри та дії, які повинні бути реалізовані. Нефункціональні вимоги визначають якість роботи системи, стабільність, продуктивність та зручність використання.

*Функціональні вимоги*

Функціональні вимоги визначають основні можливості гри та описують, які саме дії повинні бути реалізовані для забезпечення повноцінного ігрового процесу. Під час формування вимог основна увага приділялась не кількості механік, а тому, наскільки вони поєднуються між собою та підтримують загальну структуру гри.

До основних функціональних вимог належать:

- реалізація системи керування персонажем, що включає рух, стрибки, зміну напрямку та взаємодію з платформами;
- реалізація бойової системи з можливістю атаки ворогів, обробкою зіткнень та реакцією на отримання шкоди;
- створення системи рівнів із поступовим ускладненням і переходом між окремими етапами проходження;

- реалізація інтерактивних елементів середовища, зокрема платформ, пасток, рухомих об’єктів та тригерів;
- впровадження системи ворогів із базовою поведінкою та взаємодією з гравцем;
- реалізація системи винагороди за проходження рівнів і виконання окремих дій;
- створення системи розвитку персонажа, що дозволяє покращувати окремі характеристики або здібності;
- впровадження внутрішньоігрової економіки та магазину для використання отриманих ресурсів;
- реалізація елементів персоналізації персонажа;
- забезпечення роботи інтерфейсу користувача (HUD), який відображає основну інформацію про стан гри;
- реалізація системи збереження та завантаження прогресу;
- забезпечення роботи головного меню, переходів між рівнями та базових ігрових станів.

*Нефункціональні вимоги*

Нефункціональні вимоги визначають якість роботи програмного продукту та описують умови, за яких гра повинна функціонувати стабільно і комфортно для користувача. На практиці саме ці вимоги найбільше впливають на загальне враження від гри, оскільки навіть при наявності цікавих механік нестабільна робота або незручний інтерфейс можуть значно погіршити ігровий досвід.

До основних нефункціональних вимог належать:

- стабільна робота гри без критичних помилок та аварійного завершення під час безперервного ігрового сеансу тривалістю не менше 1 години;
- забезпечення плавного ігрового процесу із середньою частотою не менше 60 FPS на рекомендованій конфігурації ПК;
- підтримка стабільної частоти кадрів не нижче 30 FPS на мінімальній конфігурації системи;

- коректна робота фізики, колізій та систем взаємодії без помилкових зіткнень або проходження об’єктів один крізь одного;
- зрозумілий та зручний інтерфейс користувача з відображенням основної інформації без перекриття важливих елементів ігрової сцени;
- час завантаження рівнів не повинен перевищувати 10 секунд на рекомендованій конфігурації ПК;
- можливість подальшого розширення гри без необхідності повної перебудови основних систем проєкту;
- оптимізоване використання ресурсів процесора та оперативної пам’яті для забезпечення стабільної роботи гри;
- підтримка стабільної роботи гри на персональних комп’ютерах із різними конфігураціями апаратного забезпечення;
- надійне збереження ігрового прогресу без втрати даних після завершення роботи програми;
- забезпечення сумісності з операційними системами Windows 10 та Windows 11;
- підтримка коректної роботи гри при роздільній здатності екрана не менше 1920×1080.

Під час тестування стало помітно, що навіть незначні проблеми з продуктивністю або затримками керування швидко впливають на відчуття гри. Саме тому значна увага приділялась оптимізації рівнів, кількості об’єктів у сцені та обробці логіки в Blueprint.

*Мінімальні та рекомендовані системні вимоги*

Для забезпечення стабільної роботи гри було сформовано мінімальні та рекомендовані вимоги до апаратного забезпечення персонального комп’ютера.

*Мінімальні системні вимоги:*

- операційна система: Windows 10;
- процесор: Intel Core i3 або аналогічний;
- оперативна пам’ять: 4 ГБ RAM;
- відеокарта: NVIDIA GeForce GTX 750 Ti або аналогічна;

- вільне місце на диску: 2 ГБ;
- DirectX: версія 11.

*Рекомендовані системні вимоги:*

- операційна система: Windows 10 / 11;
- процесор: Intel Core i5 або аналогічний;
- оперативна пам’ять: 8 ГБ RAM;
- відеокарта: NVIDIA GeForce GTX 1050 Ti або аналогічна;
- вільне місце на диску: 4 ГБ;
- DirectX: версія 12.

Сформовані вимоги дозволяють підтримувати стабільну роботу гри та забезпечують комфортний ігровий процес без суттєвих втрат продуктивності. Крім цього, визначення функціональних і нефункціональних вимог стало основою для подальшого проєктування архітектури гри, реалізації механік та організації процесу тестування програмного продукту.

# 3. ОГЛЯД ЗАСОБІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ

Після визначення основних вимог стало зрозуміло, що подальша розробка потребує вибору відповідних інструментів та технологій, оскільки саме від них залежить зручність роботи, стабільність системи та загальна якість реалізації гри. На початковому етапі основна увага приділялась Unreal Engine 4, проте в процесі роботи стало помітно, що сам рушій є лише основою, а ефективність розробки значною мірою залежить від способу використання його можливостей.

Під час створення «Blade of Destiny» використовувалось поєднання кількох підходів. Частина логіки реалізовувалась засобами C++, що забезпечувало більший контроль над системою, тоді як Blueprint застосовувався для швидкого тестування механік та перевірки нових ідей. Використання Blueprint значно прискорило процес розробки та спростило внесення змін до окремих елементів гри. Для реалізації 2D-складової проєкту використовувалась технологія Paper2D. Незважаючи на те, що вона не є основним інструментом Unreal Engine 4, її можливостей виявилось достатньо для створення рівнів, роботи зі спрайтами та організації базового ігрового середовища.

## 3.1 Огляд Unreal Engine 4 як середовище розробки

До вибору рушія довелося приділити додаткову увагу, тому що для 2D-проєкту найбільш очевидними варіантами виглядали Unity або Godot. Але вже під час реальної роботи поступово з’явилась потреба у більшому контролі над системою та меншій кількості обмежень під час побудови механік і структури гри. У процесі роботи найбільш ефективним виявилось поєднання Blueprint і внутрішніх систем Unreal Engine. Досить швидко стає помітно, що під час постійного тестування механік або перевірки нових ідей можливість одразу бачити результат безпосередньо в редакторі значно спрощує розробку, також ставало помітно, що без цього багато простих задач займали б значно більше часу.

Ще один момент, який добре помітний, у рушії вже є велика кількість готових систем. Фізика, колізії та базова взаємодія між об’єктами не потребують створення з нуля. Саме через це процес розробки стає швидшим, особливо на етапі, коли проєкт тільки починає формуватись як цілісна система.

Для обґрунтування вибору середовища розробки було проведено порівняння основних характеристик популярних ігрових рушіїв, результати якого представлено в таблиці 3.1.

Таблиця 3.1

Порівняльна характеристика ігрових рушіїв

| Критерій | Unreal Engine 4 | Unity | Godot | Критерій |
|---|---|---|---|---|
| Основна мова програмування | C++, Blueprint | C# | GDScript, C# | Основна мова програмування |
| Підтримка 2D | Paper2D (обмежена) | SpriteRenderer (розвинена) | Вбудований 2D рушій | Підтримка 2D |
| Підтримка 3D | Високий рівень | Високий рівень | Середній рівень | Підтримка 3D |
| Поріг входження | Високий | Середній | Низький | Поріг входження |
| Продуктивність | Висока | Середня | Середня | Продуктивність |
| Гнучкість | Дуже висока | Висока | Середня | Гнучкість |
| Вартість | Безкоштовно + роялті | Підписка | Безкоштовно | Вартість |
| Візуальне програмування | Blueprint | Частково | Обмежене | Візуальне програмування |
| Підтримка спільноти | Велика | Дуже велика | Зростаюча | Підтримка спільноти |

Якщо дивитися на інші рушії, то вони теж мають свої переваги. Unity простіший у вході для 2D-ігор. Godot взагалі виглядає легким і зрозумілим. Але не вистачало гнучкості, хотілося не просто зробити як треба, а мати можливість гнучко адаптувати систему.

Unreal Engine 4 складніший, але дає більше контролю, і це починає відчуватись, коли додаєш щось нове або переробляєш старе. Навіть коли логіка стає складнішою, все працює досить передбачувано.

У підсумку був вибраний рушій Unreal Engine 4 не тому, що він найпростіший, а тому що він дозволяє більше, та після роботи з ним стало зрозуміло, що цей вибір виявився доцільним.

### 3.2 Мова програмування C++

У процесі роботи досить швидко стає зрозуміло, наскільки сильно Unreal Engine 4 зав’язано саме на C++. Через нього працює логіка, фізика, взаємодія між об’єктами та обробка подій. Найбільше це відчувається тоді, коли механік у грі стає більше, бо увага вже йде не на окремі речі, а на те, чи нормально вся система працює разом і наскільки стабільно поводиться гра [16,18].

На рисунку 3.1 зображено архітектура взаємодії основних класів і систем Unreal Engine 4, реалізованих засобами C++.

Якщо говорити простіше, уся логіка гри будується навколо базових класів Unreal Engine, які працюють на основі C++. Наприклад, GameMode відповідає за загальні правила гри, а GameInstance  за збереження стану між рівнями. У процесі роботи поступово починаєш краще розуміти, як ці компоненти пов’язані між собою, і через це вже значно легше орієнтуватися в самому проєкті.

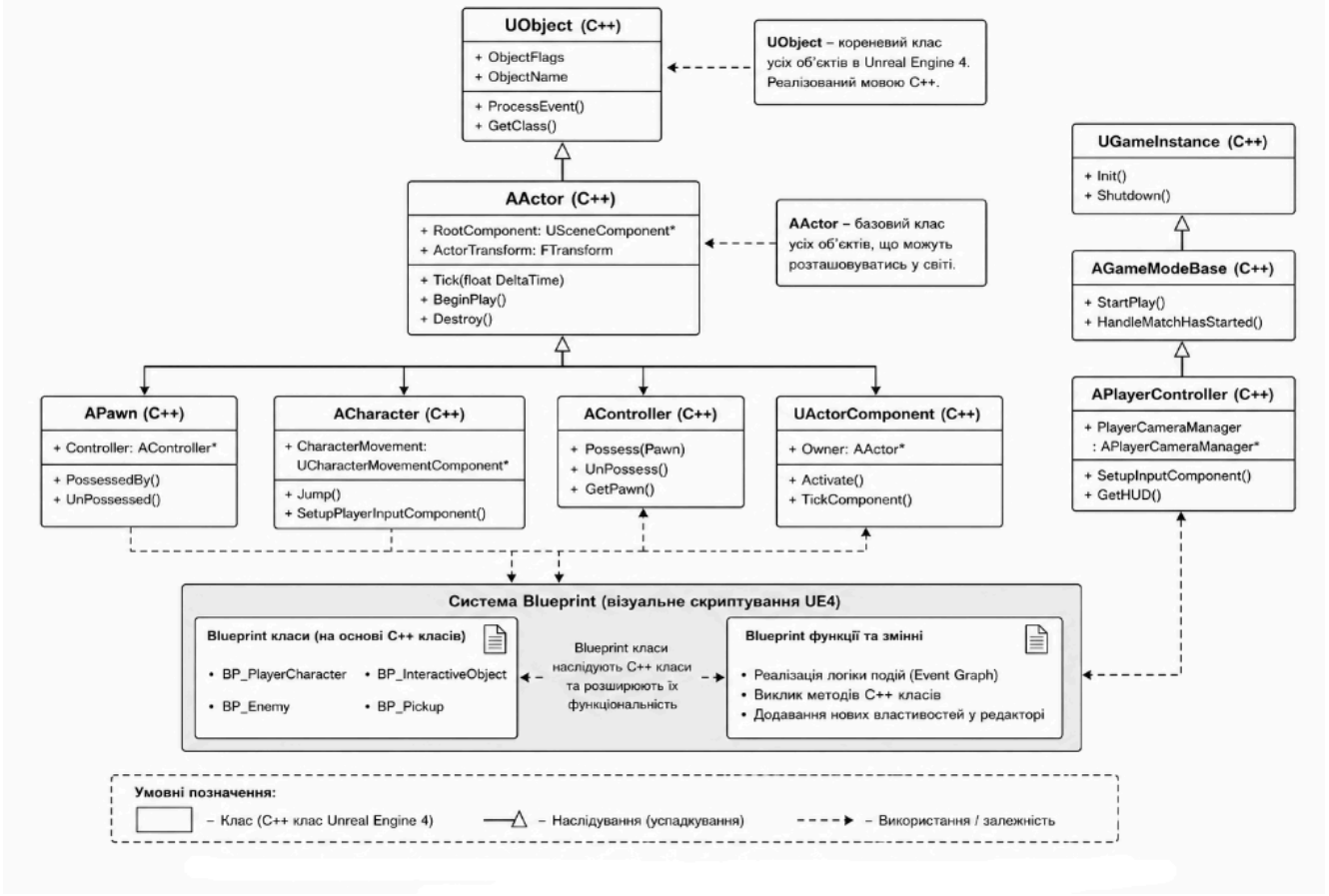



Рис. 3.1 Архітектура взаємодії основних класів і систем Unreal Engine 4, реалізованих засобами C++

Окремо стає помітною роль Actor, тому що саме на ньому будується більшість об'єктів у грі: персонажі, вороги, елементи рівня та інтерактивні об'єкти. Через це вся структура починає виглядати більш цілісною і зрозумілою.

У підсумку C++ у цьому проєкті важливий не як окрема мова для написання великої кількості коду вручну, а як основа, на якій побудований Unreal Engine 4 і всі системи, з якими працює Blueprint.

### 3.3 Система візуального програмування Blueprint

Blueprint став одним із тих інструментів, без яких розробка пішла б значно повільніше. На початку було відчуття, що це щось додаткове, але в процесі стало зрозуміло, що він дозволяє вирішувати значну кількість задач [12, 13]. Замість

коду є вузли, які з’єднуються між собою. Це виглядає трохи дивно, але сприймається досить швидко, в якийсь момент навіть логіка сприйматися у вигляді послідовності дій, а саме такими ланцюжками дій.

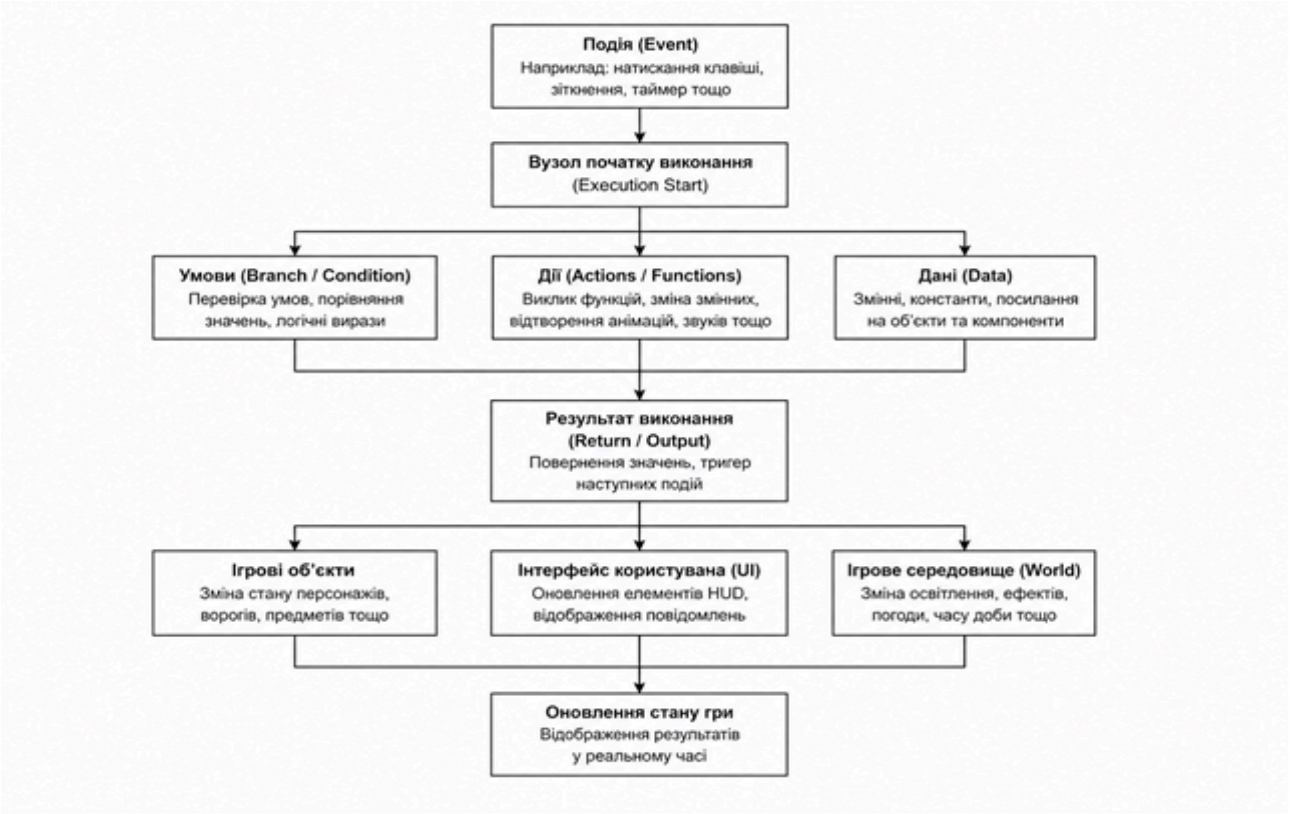



Рис. 3.2 Структурна схема обробки логіки у Blueprint

На рисунку 3.2 представлено, як це працює на практиці: є подія наприклад, натискання кнопки або зіткнення. Далі від неї будується логіка перевірки, виклики функцій, зміни змінних. Усе це виглядає як послідовність вузлів, і ти буквально бачиш, як рухається логіка, це значно спрощує процес налагодження. Бо коли щось не працює, простіше знайти помилку, ніж у коді, тому що видно де саме обривається логіка. У межах одного Blueprint можна зробити досить складні речі: умови, цикли, функції. Але при цьому все залишається наочним. Це особливо допомагає, коли потрібно швидко щось змінити або перевірити нову ідею. У Blueprint можна за кілька хвилин, без перекомпіляції.  Але повністю покладатися на нього теж не вийде. Для складних речей він починає перевантажуватись.

У підсумку Blueprint перестав сприйматись просто як допоміжний інструмент і став повноцінною частиною всього процесу розробки, тому що без нього навіть на базові механіки та перевірку простих речей доводилося б витрачати значно більше часу.

### 3.4 Технологія Paper2D

Робота з Paper2D виявилась простішою, ніж з іншими інструментами. Тут не потрібно було тривале освоєння, інструмент має зрозумілу структуру роботи, бо вся робота будується навколо спрайтів [14, 15].

У «Blade of Destiny» Paper2D став основою всієї частини гри. Саме через нього будуються персонажі, вороги, платформи та самі рівні. У процесі роботи у процесі роботи стало помітно, що саме Paper2D формує загальний вигляд гри і те, як вона сприймається візуально.

Основні можливості досить стандартні:

- спрайти (окремі зображення)
- анімації через Flipbook
- побудова рівнів через TileMap

Але на практиці цього достатньо для реалізації 2D-платформера, особливо якщо правильно організувати роботу з ресурсами. На рисунку 3.3 наведено схему використання основних можливостей Paper2D у межах проєкту «Blade of Destiny».

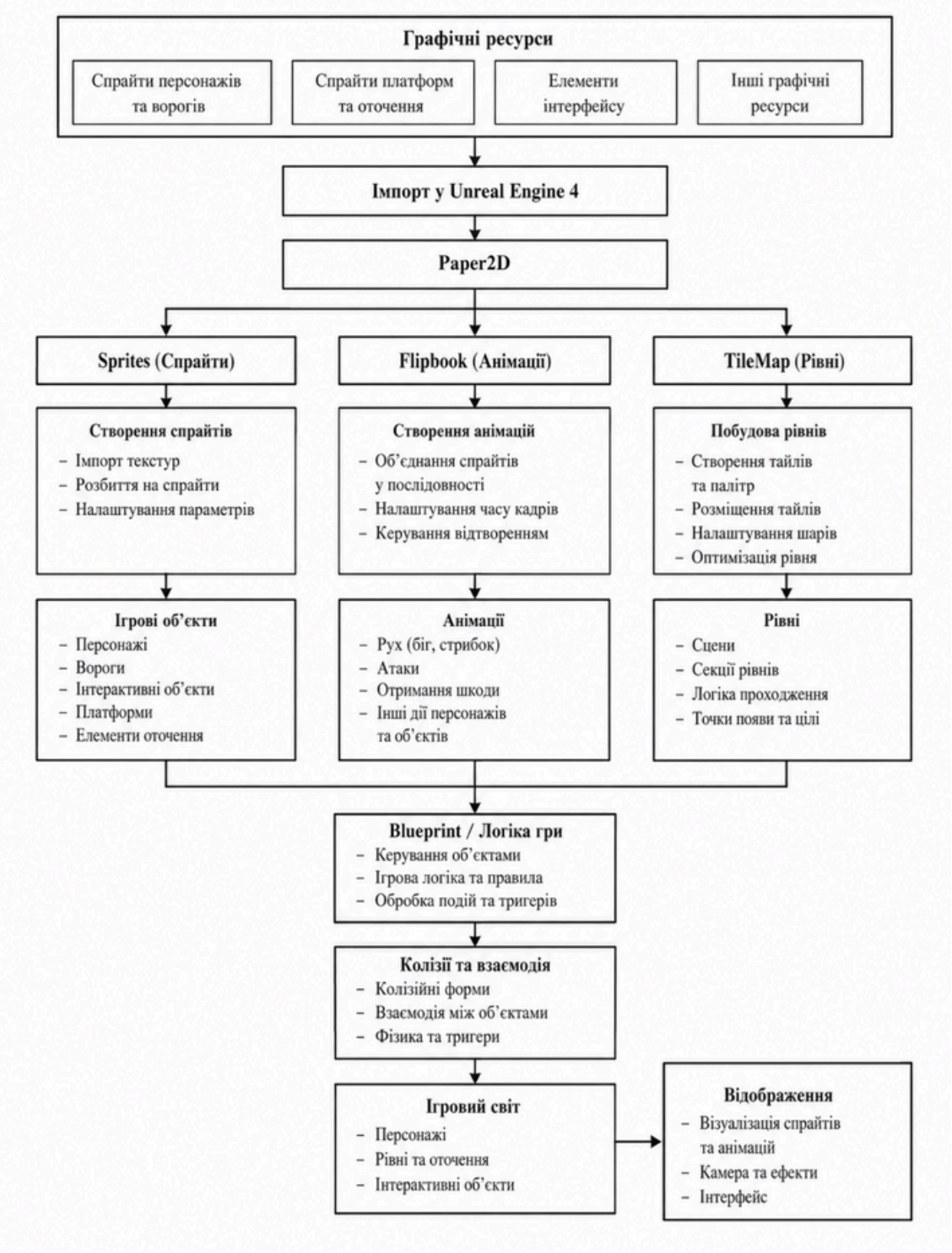



Рис. 3.3 Структура використання Paper2D у проєкті «Blade of Destiny»

Сам процес роботи виглядає досить просто: спочатку готується графіка, після чого вона імпортується в Unreal Engine. Далі зображення перетворюються у спрайти, з яких уже формуються анімації та ігрові об'єкти. У процесі роботи

добре стає помітно, як із окремих зображень поступово починає складатись уже повноцінний рівень або персонаж у грі.

Спрайти це лише графічне представлення об’єкта, яка показує об’єкт гравцю. Поки до них не підключена логіка, вони не мають функціональної поведінки, тому що вся поведінка вже задається через Blueprint та системи Unreal Engine: рух, взаємодія, фізика, події. І тільки після того, як об’єкти додаються в рівень і починають працювати разом із колізіями та іншими механіками, з’являється відчуття, що це вже не просто набір зображень, а повноцінна частина гри. У проєкті Paper2D використовується для:

- персонажів
- ворогів
- платформ
- елементів оточення

Момент в який стає зрозуміло, що цього вже достатньо, щоб зібрати повноцінний ігровий рівень без використання якихось складних додаткових інструментів. Але під час роботи добре починає відчуватись ще одна річ узгодженість стилю, навіть якщо технічно все працює нормально, різні спрайти можуть виглядати стилістично неузгодженими. Через це частина часу йшла вже не на саму реалізацію, а на підбір і адаптацію графіки, щоб усе сприймалось як одна гра, а не набір окремих елементів.

У підсумку Paper2D показав себе як досить практичний інструмент для розробки 2D-платформера. Він не виглядає найскладнішою частиною Unreal Engine, але при цьому ефективно забезпечує реалізацію базових задач, пов’язані з візуальною частиною гри, і без проблем працює разом з іншими системами рушія

### 3.5 Додаткові інструменти розробки

Розробка гри це не тільки Unreal Engine 4. Коли проєкт починає рости, з’являється багато речей, без яких нормально підтримувати роботу вже складно. Більшість часу робота проходила безпосередньо в Unreal Engine 4. Саме там створювались рівні, перевірялись механіки, налаштовувалась взаємодія між об’єктами і тестувалась поведінка гри після змін. З часом редактор уже сприймається не просто як програма, а як місце, де фактично існує весь проєкт.

Окремо використовувався Git для контролю версій [19, 20]. Спочатку може здаватися не надто важливим, але коли змін у проєкті стає багато, без нього дуже легко щось втратити або випадково зламати. Особливо це помітно в моменти, коли одна невдала зміна тягне за собою проблеми в інших частинах гри.

Тестування також постійно виконувалось прямо в Unreal Engine 4. Після будьякої зміни можна було одразу запустити рівень і перевірити, як усе працює вже в самій грі. Саме це найбільше допомагало під час роботи з механіками, які ще багато разів змінювались і перероблялись.

У підсумку додаткові інструменти не виглядають головною частиною розробки, але без них увесь процес ускладнює контроль над проєктом. І чим більшим стає проєкт, тим сильніше це відчувається.

# 4. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ 2D-ПЛАТФОРМЕРА «BLADE OF DESTINY»

У цьому розділі розглядається процес створення гри «Blade of Destiny»: від побудови загальної структури до реалізації ігрових механік та перевірки їхньої спільної роботи.

Саме на етапі практичної реалізації стає помітно, наскільки ефективно обрані підходи працюють у межах проєкту. Частина рішень підтверджує свою ефективність під час тестування, тоді як окремі елементи потребують доопрацювання або зміни вже в процесі розробки. Через це створення гри має ітеративний характер, коли механіки та окремі системи неодноразово перевіряються та коригуються.


Рис. 4.1 Загальний вигляд ігрового процесу «Blade of Destiny»

На рисунку 4.1, представлено загальний вигляд ігрового процесу, що дозволяє оцінити організацію рівня, взаємодію персонажа з ігровим середовищем та загальний візуальний стиль гри.

Розробка «Blade of Destiny» реалізована із використанням C++ та Blueprint. Мова програмування C++ застосовувалась для реалізації систем, які потребують більшого контролю, тоді як Blueprint використовувався для швидкого створення та тестування механік. Подібне поєднання дозволило спростити процес розробки без надмірного ускладнення структури проєкту.

У процесі роботи стало помітно, що основна складність полягає не лише у створенні окремих механік, а й у забезпеченні їхньої коректної взаємодії між собою. Рівні, система керування, противники та інші елементи гри повинні працювати як цілісна система, оскільки будь-які невідповідності швидко стають помітними під час гри.

Процес розробки не є повністю лінійним. Більшість механік та окремих елементів неодноразово тестувалися та доопрацьовувалися після перевірки в ігровому середовищі.

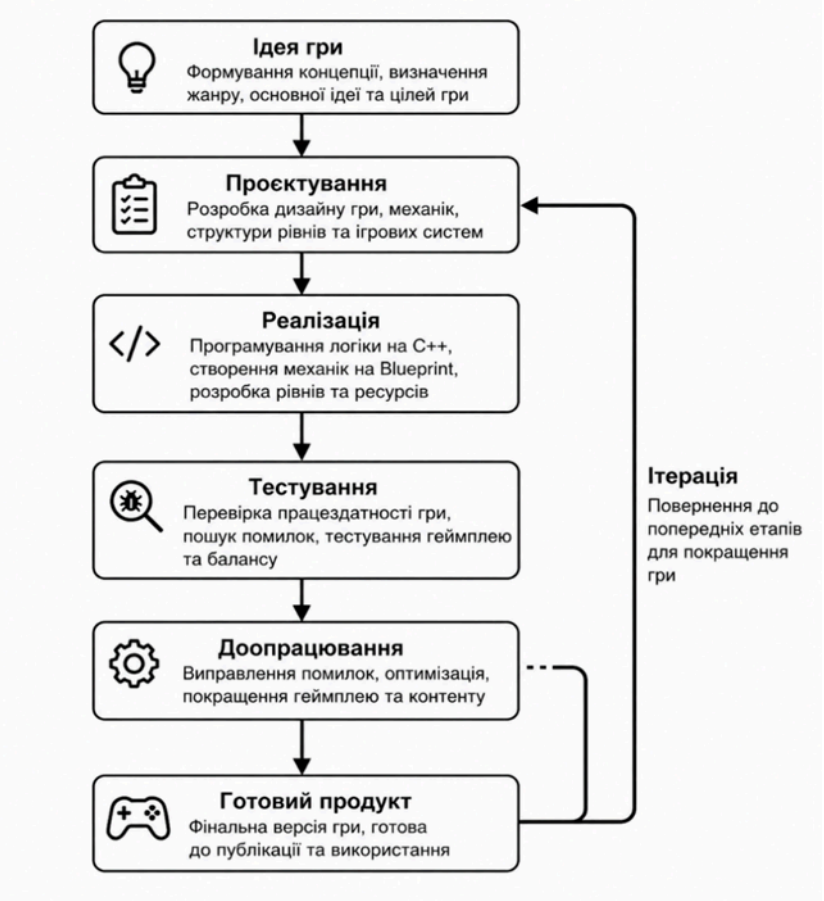



Рис. 4.2 Загальний процес розробки гри

Схема на рисунку 4.2, ілюструє загальну логіку створення програмного продукту, де окремі етапи можуть повторюватися до досягнення необхідного результату.

Далі будуть розглянуті основні складові гри: загальна структура проєкту, процес створення рівнів, реалізація механік, організація ігрового циклу та інтерфейсу користувача. Також буде описано процес тестування окремих елементів під час розробки.

## 4.1 Архітектура гри

На початку розробки не було чіткого уявлення, якою саме має бути архітектура гри, все виглядало досить просто: є персонаж, є рівень, є взаємодія. Але вже після додавання перших механік стало зрозуміло, що якщо не розділити систему на частини, дуже швидко з’являється плутанина.

У процесі роботи поступово сформувався модульний підхід. Подібний підхід сформувався насамперед через практичні потреби розробки, тому що інакше стає складно щось змінювати. Коли логіка починає змішуватися в одному місці, будь-яка правка тягне за собою нові проблеми. Саме тому було прийнято рішення розділити гру на окремі компоненти, які взаємодіють між собою, але не залежать один від одного надмірно.

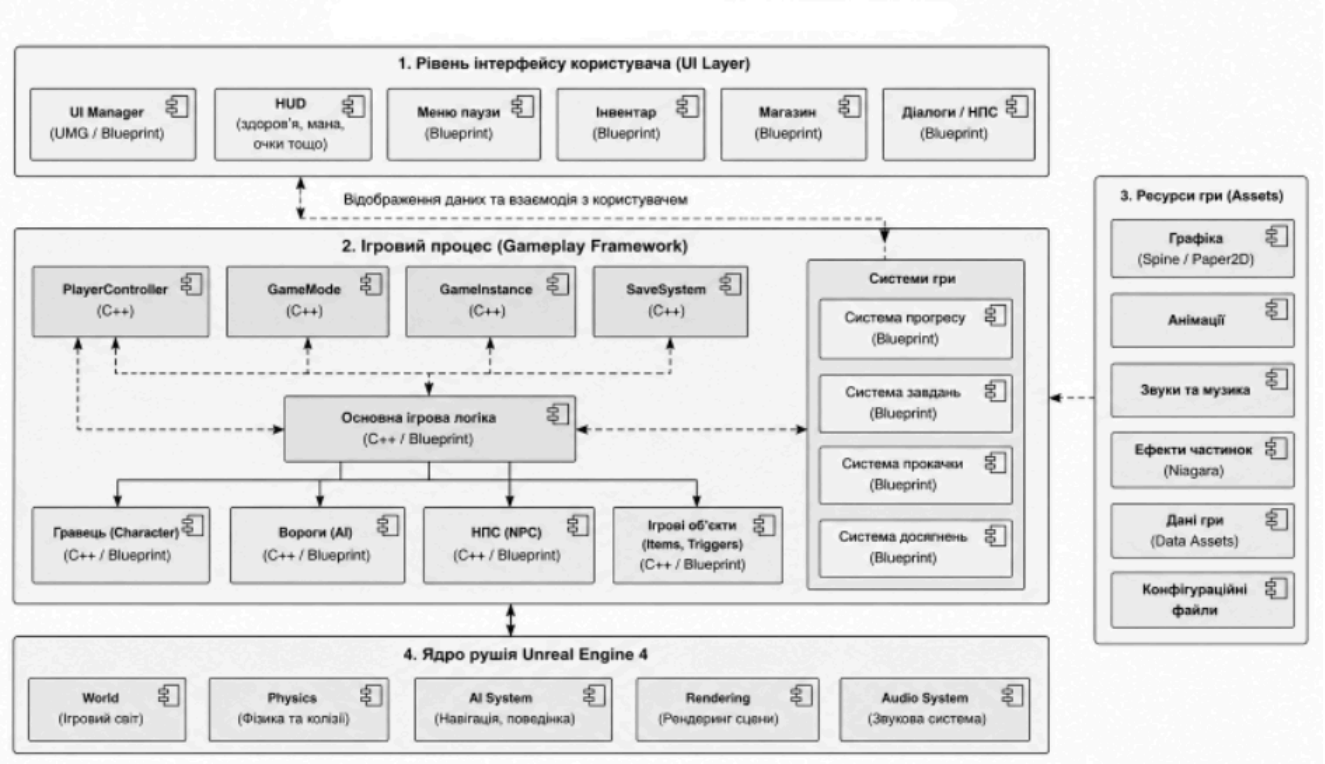



Рис. 4.3 Архітектурна структура гри «Blade of Destiny»

Рисунок 4.3 демонструє архітектурну структуру гри «Blade of Destiny», побудовану на основі Unreal Engine 4. Архітектура поділена на декілька основних рівнів: інтерфейс користувача, ігровий процес, ресурси гри та ядро рушія.

На рівні інтерфейсу користувача реалізовано елементи взаємодії з гравцем, зокрема HUD, меню паузи, інвентар, магазин та діалогову систему. Дані компоненти створені переважно за допомогою UMG та Blueprint і забезпечують відображення інформації та обробку дій користувача. Основна логіка гри знаходиться у блоці Gameplay Framework. Обробка вводу користувача здійснюється через компонент PlayerController, після чого керування передається до GameMode та основної ігрової логіки. Для збереження глобального стану гри використовується GameInstance, а система SaveSystem відповідає за збереження прогресу гравця.

Ігрові об’єкти, зокрема персонаж гравця, вороги, NPC та інтерактивні елементи рівня, реалізовані на базі класів Actor та Character із використанням C++

і Blueprint. Взаємодія між об’єктами виконується через події, тригери, фізику та систему колізій Unreal Engine.

Окремо в архітектурі виділено підсистеми гри: систему прогресу, завдань, прокачування та досягнень. Вони взаємодіють з основною логікою гри та забезпечують реалізацію ігрових механік.

Усі програмні компоненти використовують ресурси гри, до яких належать графіка, анімації, звукові ефекти, музика, дані гри та конфігураційні файли. Робота всіх елементів підтримується ядром Unreal Engine 4, яке забезпечує функціонування фізики, рендерингу, аудіосистеми та ігрового світу.

Результат цієї взаємодії передається до системи відображення тобто гравець бачить те, що відбулося. І саме тут найкраще розглянути, чи працює вся архітектура правильно. Бо якщо на екрані все виглядає логічно і передбачувано, значить система побудована вдало. Такий поділ на компоненти значно спрощує розробку. Легше тестувати окремі частини, легше знаходити помилки і, що важливо, легше додавати нові механіки. У випадку «Blade of Destiny» це стало особливо помітно, коли почали з’являтися нові елементи геймплею.

У результаті архітектура гри не з’явилась як готове рішення. А збиралась через постійні правки і відсікання того, що не працює. Залишилось тільки елементи, які забезпечують стабільність системи. У підсумку вийшла структура, з якою зручно працювати і яку можна без проблем розширювати.

### 4.2 Проєктування рівнів та ігрового середовища

Проєктування рівнів швидко показує складність процесу. Спочатку здається розставив платформи, додав ворогів і має працювати. Але вже на перших запусках видно, що рівень не читається: темп проходження порушується, складність стрибає, місцями незрозуміло, куди рухатись, з часом фокус зміщується.

Рівень це  не декорації, а шлях гравця, де він розганяється, де робить помилки, де встигає зрозуміти, що від нього хочуть. Саме ця послідовність дій вирішує, чи гра відчувається плавною, чи втрачає цілісність. Тому більшість

рішень тут не з першого разу. Рівні доводилося спрощувати, переставляти, інколи збирати заново, поки рух і складність не починали працювати узгоджено. Це не про ідею на папері, а про те, як воно відчувається в грі. Для кращого розуміння логіки побудови рівнів та організації ігрового процесу було сформовано відповідну модель проходження, наведену на рисунку 4.4.

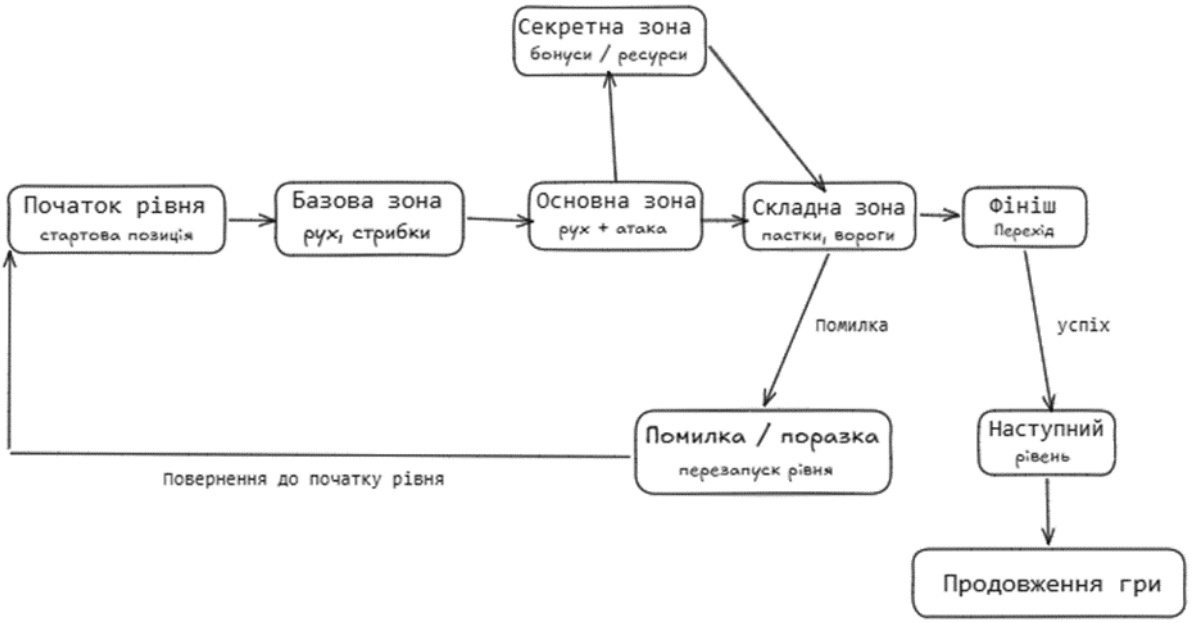



Рис. 4.4 Узагальнена модель проходження рівня

Найкраще працює підхід поступового ускладнення, тобто спочатку гравець знайомиться з базовими елементами, а вже потім починає використовувати їх у складніших комбінаціях. Незважаючи на очевидність такого підходу, реалізувати це правильно з першого разу майже не виходить.

Було помітно, що без чіткого поділу рівня на частини складно контролювати складність. Саме тому з’явилось рішення умовно розділити рівень на кілька зон:

- початкова для адаптації
- базова для освоєння механік
- основна для комбінування дій
- складна як фінальний виклик

Спостерігаючи на проходження рівня вже після реалізації, воно виглядає досить логічно: гравець рухається від простого до складного. Але під час

розробки це не виглядає як пряма лінія. Часто доводиться повертатися назад і змінювати навіть базові речі. Наприклад, були ситуації, коли один невдало поставлений ворог повністю порушував темп проходження. Гравець або зупинявся надто довго, або навпаки втрачав контроль. І такі речі не видно на схемі вони стають очевидними тільки під час гри.

Окрему увагу було приділено варіативності, спочатку рівні робились повністю лінійними, і це виглядало нормально. Але після кількох проходжень стало відчутно, що грі не вистачає глибини. Саме тому були додані альтернативні ділянки та невеликі секретні зони. І цікаво те, що навіть мінімальні відхилення від основного маршруту одразу роблять рівень більш живим. Один із підходів до побудови нелінійного проходження із використанням альтернативних маршрутів представлено на рисунку 4.5.

Рис. 4.5 Приклад структури рівня з альтернативними шляхами

Обробка помилок гравця, спочатку розглядалися різні варіанти, але на практиці найкраще спрацювало просте рішення: повернення на початок рівня, з одного боку це виглядає більш суворим, але з іншого дає чітке розуміння правил гри. Гравець не плутається і швидко адаптується до системи. Підсумовуючи результати розробки, то найважливішим у проєктуванні рівнів виявилась не кількість елементів, а їх розташування. Інколи достатньо змістити платформу або ворога на кілька одиниць і рівень починає відчуватись зовсім інакше.

Рівні у «Blade of Destiny» не з’являються одразу в готовому вигляді. Більшість із них кілька разів перероблялися, бо те, що виглядає нормально в редакторі, у грі часто працює зовсім інакше. У підсумку саме постійні перевірки і виправлення формують фінальний результат, а не початкова ідея.

### 4.3 Реалізація геймплейних механік

Якщо говорити про практичну частину гри, то саме геймплейні механіки найбільше впливають на те, як гравець відчуває гру. Можна зробити гарний рівень або зручний інтерфейс, але якщо рух, стрибок або атака працюють незручно, це одразу псує враження.

У «Blade of Destiny» основні механіки реалізовані через поєднання C++ та Blueprint. Частину базової логіки винесено в код, а події та взаємодії налаштовуються через Blueprint. У процесі стало зрозуміло, що тягнути все через один підхід, виявилось неефективним: десь потрібен контроль, десь швидкість. Код закриває складні речі, Blueprint дає можливість швидко вносити зміни і одразу перевірити. Такий поділ з’явився не відразу до нього довелося прийти через роботу. Коли все розділено таким чином, система стає простішою і зрозумілішою.

У підсумку залишився базовий набір механік: рух, стрибок, взаємодія з платформами, атака, зіткнення з ворогами, збір предметів і активація подій. Саме на цьому тримається вся гра.

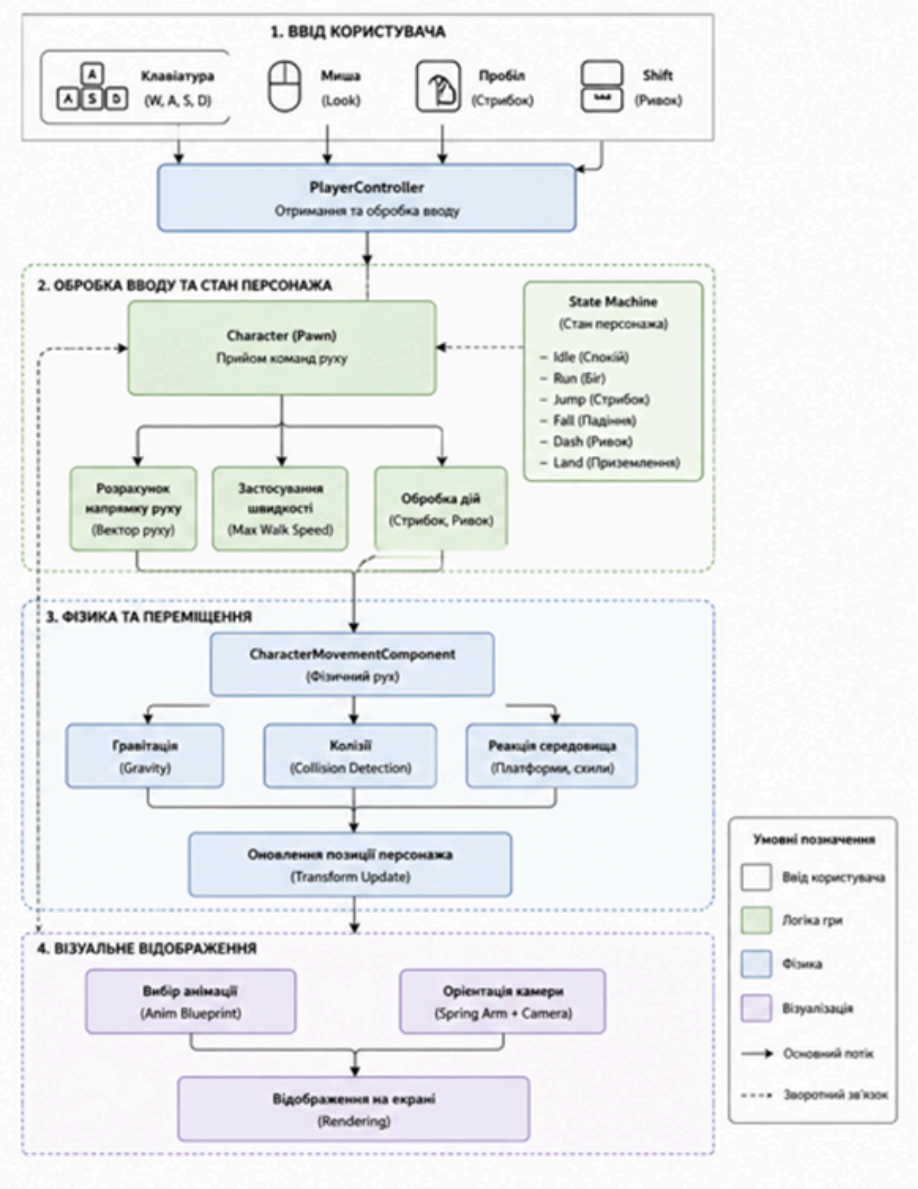



Рис. 4.6 Схема реалізації механіки руху персонажа

На рисунку 4.6 показано загальну логіку руху персонажа. Спочатку система отримує ввід від користувача, після чого визначається напрям руху. Далі ці дані передаються до логіки персонажа, де формується швидкість, стан персонажа та відповідна анімація.

Під час розробки довелося приділити увагу руху. На перший погляд це проста механіка, але на практиці навіть невеликі зміни швидкості, гравітації або висоти стрибка сильно змінюють відчуття гри. У грі відчуття керування вирішує дуже багато. Як тільки рух налаштований невдало це відчувається одразу. Занадто різко  складно контролювати. Занадто повільно  зникає динаміка і гра починає

тягнутися. Тому тут усе тримається на підборі. Міняються значення, перевіряється поведінка, і так кілька разів, поки рух не починає відчуватись нормально.

Це потрібно не тільки для логіки, а й для правильного відображення анімацій. Тобто гра повинна не просто обробити дію, а ще й показати її гравцю зрозуміло.

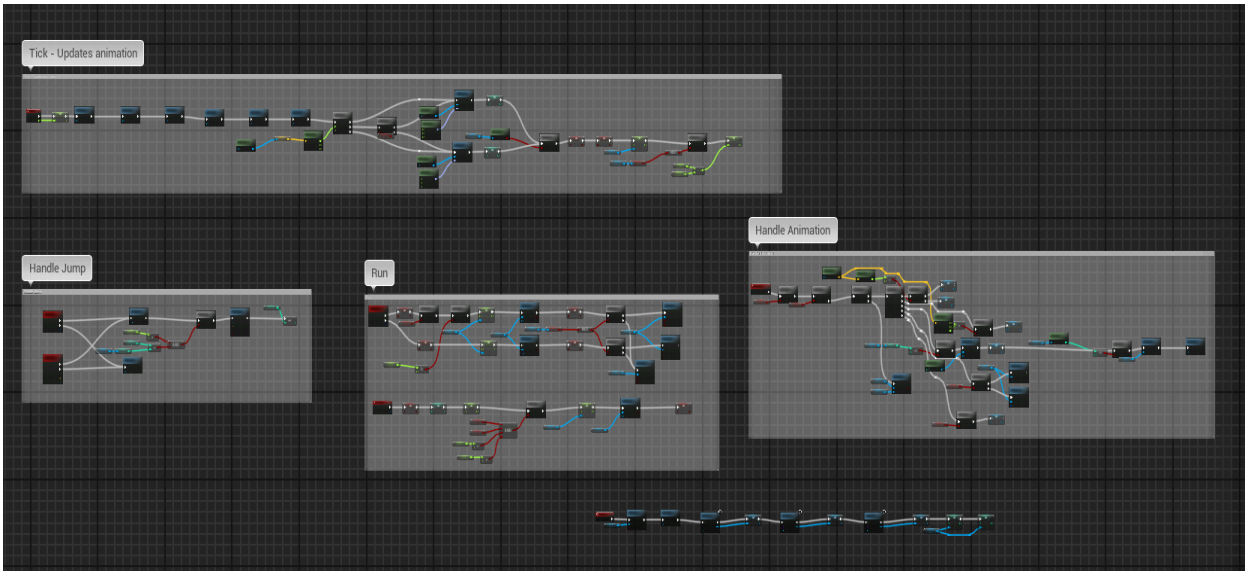



Рис. 4.7 Реалізація механіки руху персонажа у Blueprint

На рисунку 4.7 показано шматок Blueprint, де обробляється рух або стрибок персонажа. Це не про схему на папері, а те, як воно реально зібрано в проєкті і як працює в грі. У роботі Blueprint відчувається як інструмент для швидких змін. Потрібно підкрутити реакцію на кнопку або перевірити, як поводиться персонаж це можна швидко перевірити, без зайвого переписування коду.

Бойова система тут не винесена окремо. Вона працює разом із рухом. Просто натиснути атаку недостатньо, потрібно підійти, зайняти позицію і вибрати момент. Саме в цьому з’являється комфортне сприйняття геймплею.

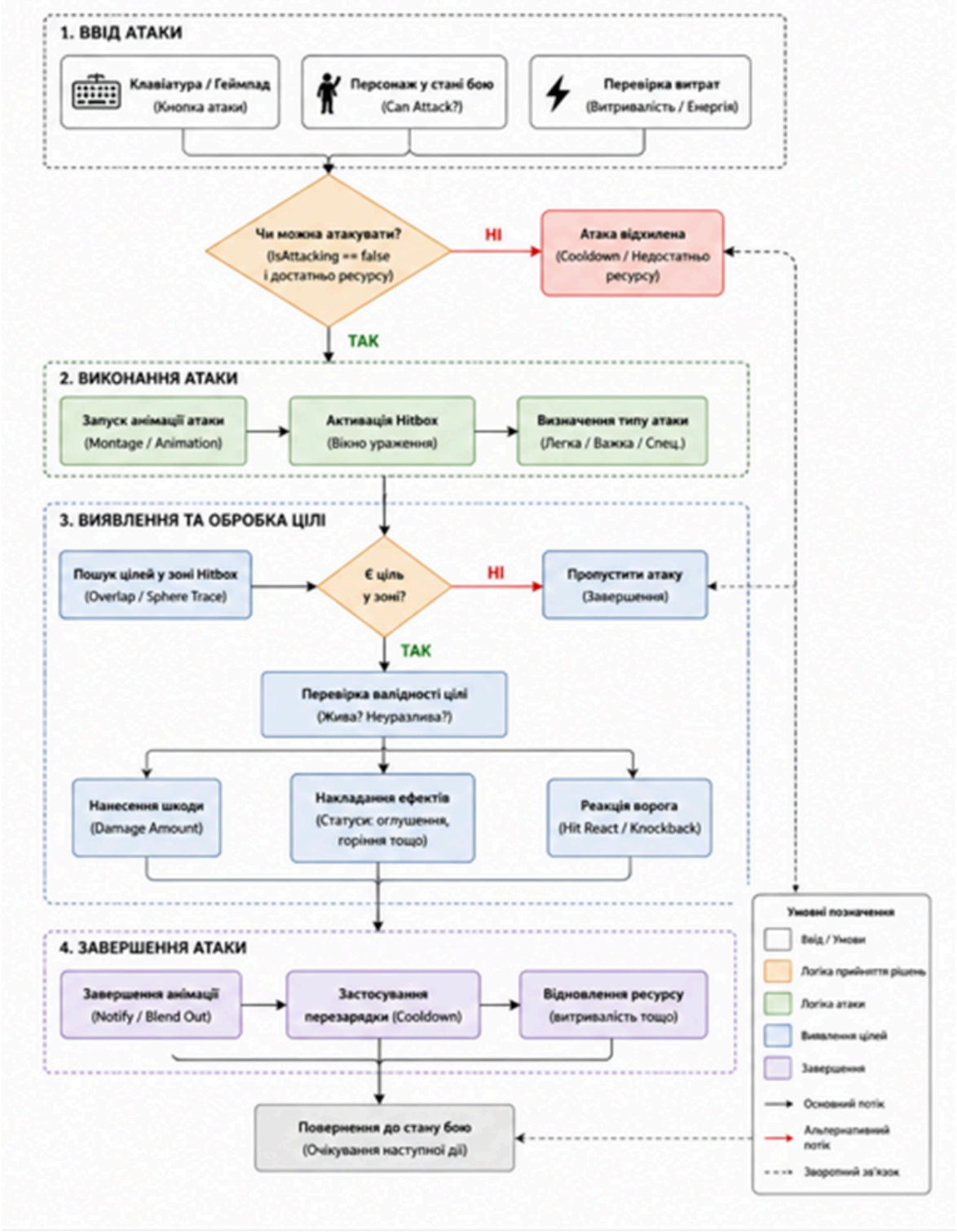



Рис. 4.8 Схема реалізації бойової взаємодії

Рисунок 4.8 показує загальну логіку атаки. Після натискання відповідної клавіші система перевіряє, чи може персонаж виконати атаку. Далі визначається зона взаємодії та перевіряється наявність ворога або іншого об’єкта в цій зоні. Якщо ціль знайдено, до неї застосовується відповідний ефект.

На практиці бойова механіка потребувала не тільки технічної реалізації, а й балансування. Якщо атака спрацьовує занадто повільно, гравцю незручно. Якщо занадто швидко або сильно  гра стає надто простою. Тому бойову систему доводиться перевіряти безпосередньо в рівні, а не тільки на окремій тестовій сцені.

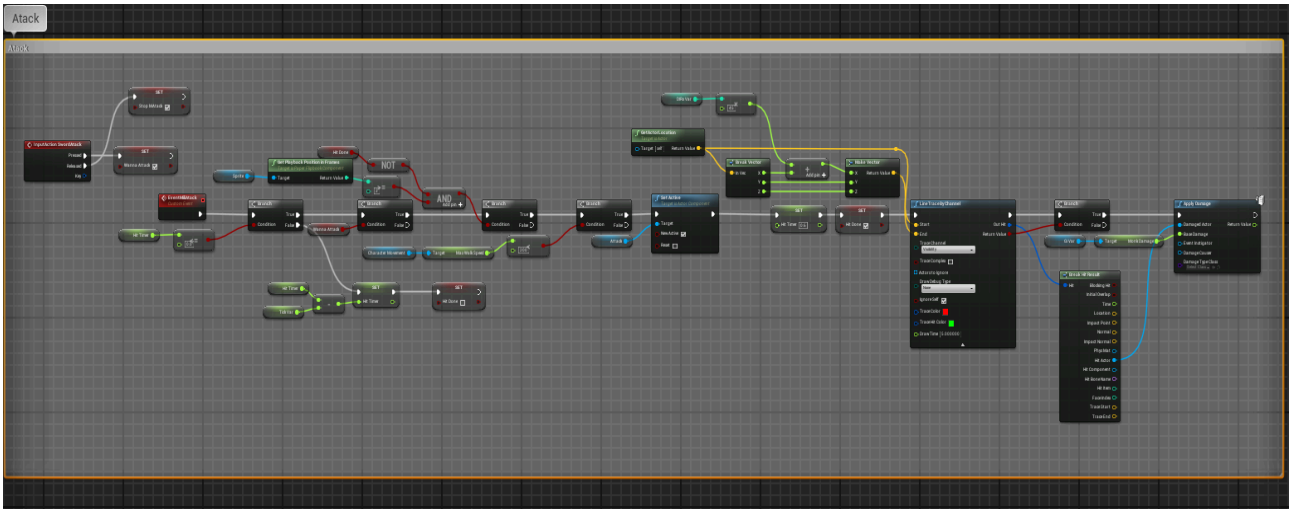



Рис. 4.9 Реалізація бойової системи у Blueprint

Фрагмент Blueprint на рисунку 4.9 добре показує, як у грі побудована бойова взаємодія. У процесі роботи саме через такі ноди починаєш краще бачити, як між собою пов’язані атака, перевірка зіткнення і реакція ворога на дії гравця. Через Blueprint це відчувається значно простіше, тому що логіку можна безпосередньо простежити прямо в редакторі. Видно, у який момент спрацьовує атака, як перевіряється зіткнення і що відбувається далі. Саме в такі моменти механіка починає сприйматись не як окрема дія, а як частина всієї системи гри.

Досить швидко стає помітно, що без взаємодії з оточенням рівень починає виглядати порожнім. Одного руху і бойової системи вже недостатньо, щоб гра нормально відчувалась. Саме тому поступово додаються платформи, пастки, предмети, тригери та інші інтерактивні елементи. На перший погляд це дрібниці, але саме через них рівень починає оживати. У грі це сприймається дуже прямо. Підібрав предмет  одразу є результат. Активував тригер  змінилось середовище:

відкрився прохід, з’явився ресурс або змінився стан об’єкта. І саме в таких моментах починає з’являтись відчуття, що дії гравця реально впливають на те, що відбувається навколо. Через це рівень перестає бути просто дорогою вперед і починає сприйматись як середовище, з яким взаємодієш.

Найскладнішим виявилось не зробити окрему механіку, а зібрати все в одну робочу систему. Рух, стрибки, атака, вороги, об’єкти рівня кожен елемент сам по собі не виглядає складним, але разом починають конфліктувати. Стає зрозуміло, що достатньо однієї невдалої взаємодії  і гра починає сприйматись менш комфортно. Десь встрачається темп, десь з’являється відчуття неконтрольованості, і навпаки, коли ці елементи починають працювати узгоджено, це відчувається одразу. Гра стає плавнішою, зрозумілішою і просто цікавішою в проходженні.

### 4.4 Проєктування основного ігрового циклу (Core Loop)

Якщо дивитися на гру вже під час самої розробки, у якийсь момент стає помітно, що все на екрані працює не окремо, а як один постійний процес. Рух, атака, фізика, взаємодія з об’єктами, події усе це безперервно оновлюється і працює одночасно.

Спочатку це не помітно, тому що увага більше йде на окремі механіки. Хочеться зробити нормальний рух, атаку або взаємодію з об’єктами. Але з часом стає видно, що проблема часто виникає не в самих механіках, а в тому, як вони поводяться разом. У якийсь момент гра може почати працювати нестабільно, хоча окремо все працює правильно. Десь зникає плавність, десь дії спрацьовують нестабільно, а десь система починає конфліктувати. Саме тоді починаєш поіншому дивитися на гру. Уже не як на набір окремих дій, а як на одну систему, де все постійно взаємодіє між собою.

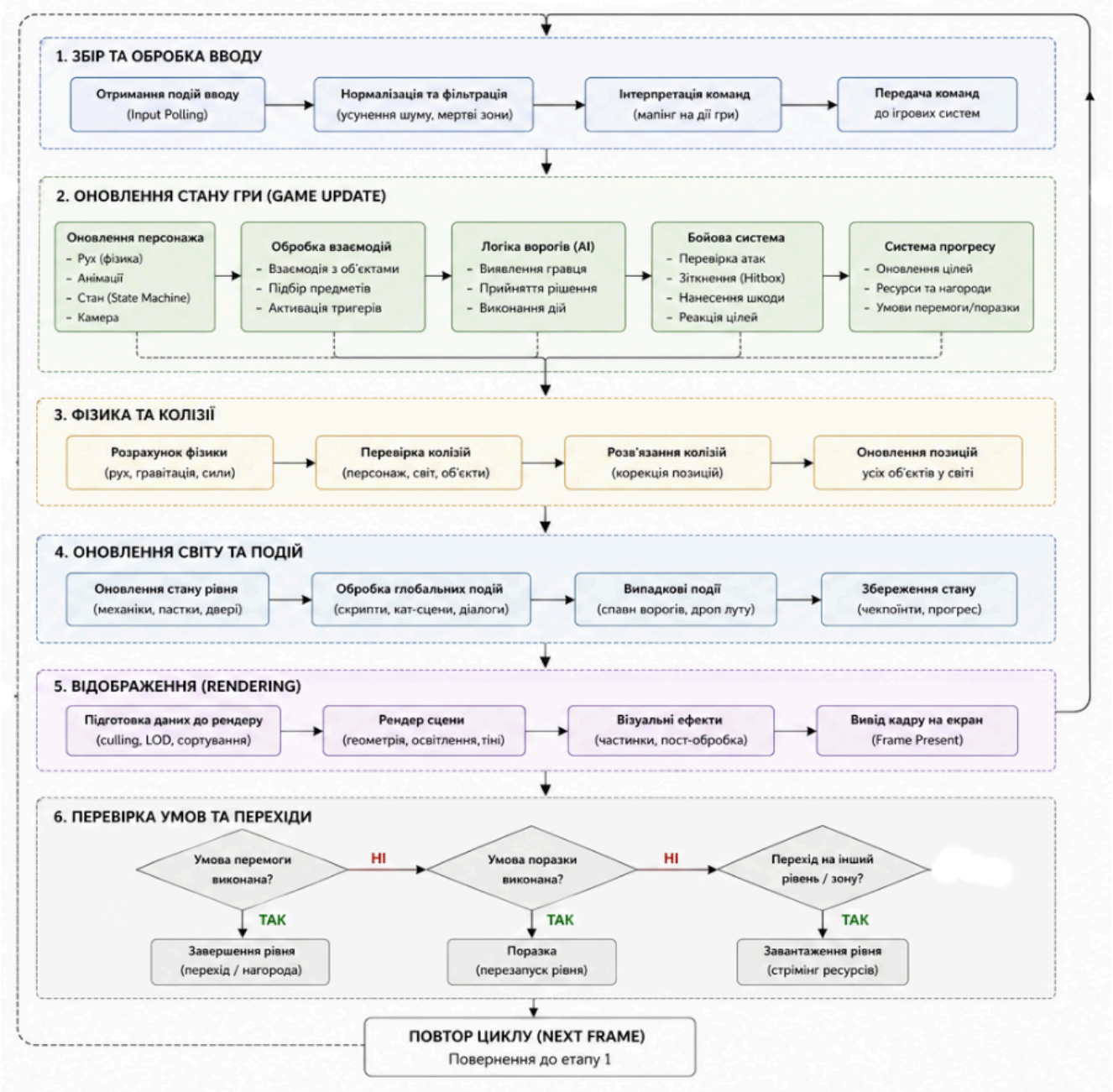



Рис. 4.10 Архітектура основного ігрового циклу (Core Loop)

Переглянувши схему на рисунку 4.10, все виглядає досить логічно, але на практиці кожен цей етап має свої нюанси. Цикл починається з вводу, гравець натискає кнопку, і здається, що персонаж просто рухається. Але насправді цей сигнал проходить кілька етапів обробки, перш ніж щось зміниться в грі. Далі йде оновлення стану гри. І от тут уже відбувається все основне рух персонажа, поведінка ворогів, атаки, взаємодія з об’єктами. Якщо говорити простіше, саме

тут реалізується основна логіка геймплею. І саме цей етап найбільше залежить від того, наскільки добре продумані механіки.

Слід згадати про фізику і колізії. Теоретично це виглядає як проста перевірка зіткнень. Але під час розробки саме тут найчастіше з'являються проблеми: персонаж може залипати, проходити крізь об'єкти або поводитися не так, як очікується. Тому цей етап довелося не раз підлаштовувати.

У процесі роботи, гра повинна постійно реагувати на дії гравця. Саме для цього і працює система обробки подій. Гравець може навіть не звертати на це увагу, але саме в цей момент спрацьовують тригери, змінюються стани об'єктів, відкриваються проходи або нараховуються ресурси. Через такі моменти рівень починає відчуватись живим. Без цього все навколо дуже швидко перетворюється просто на набір платформ і декорацій, які ніяк не реагують на гравця.

Далі все це переходить у відображення. Гравець бачить вже готовий результат рух, анімації, ефекти, інтерфейс. І якщо хоча б один із попередніх етапів працює неправильно, це одразу видно саме тут.

Останнім етапом стає перевірка умов гри. Саме в цей момент система визначає, що відбулося далі: гравець переміг, програв або гра просто продовжується без змін. Якщо нічого критичного не сталося, цикл запускається знову. І саме так гра працює постійно кадр за кадром. Гравець зазвичай цього не помічає, але саме через таке безперервне оновлення всі механіки, події та взаємодії відчуваються цілісно і плавно.

На рисунку 4.11 наведено схему реалізації ігрового циклу у середовищі Blueprint, яка використовується для організації взаємодії між основними процесами гри.

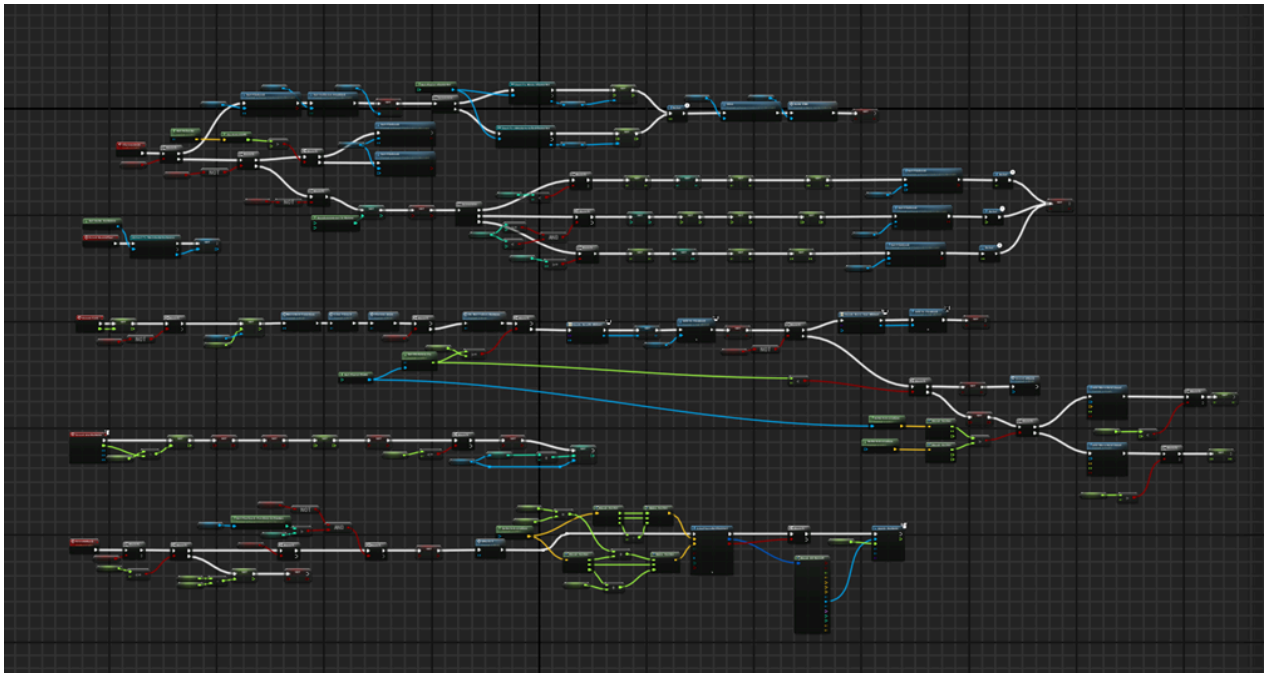


Рис. 4.11 Реалізація ігрового циклу у середовищі Blueprint

Коли працюєш в Unreal Engine, стає зрозуміло, що частина цього циклу вже реалізована самим рушієм. Але доступ до нього відбувається через події, зокрема Event Tick. Саме через нього відбувається постійне оновлення стану гри. Спочатку здається, що можна багато логіки реалізувати через Tick, але з часом стає зрозуміло, що це може створити проблеми з продуктивністю. Тому доводиться думати, що саме має оновлюватися кожен кадр, а що можна винести в події. Щоб краще узагальнити структуру циклу, основні його етапи зведено в таблицю.

У таблиці 4.1 представлено характеристику основних процесів ігрового циклу та їх функціональне призначення у структурі гри.

Таблиця 4.1

Основні етапи ігрового циклу

| Етап | Опис |
|---|---|
| Обробка вводу | Отримання дій користувача |
| Оновлення стану | Виконання логіки гри |
| Фізика | Рух і зіткнення |

Продовження таблиці 4.1

Основні етапи ігрового циклу

| | |
|---|---|
| Події | Активація тригерів та змін |
| Відображення | Формування кадру |
| Перевірка умов | Перехід між станами |

У теорії ігровий цикл виглядає досить простим, але справжня складність починається тоді, коли рух, бойова система, фізика і події повинні постійно працювати разом без конфліктів, тому під час розробки «Blade of Destiny» його доводилося багато разів змінювати, підлаштовувати і спрощувати, поки гра не почала поводитись стабільно та передбачувано. Ігровий цикл це не просто технічна схема, а основа всієї гри. Саме він об’єднує всі механіки в єдину систему і робить гру такою, якою її бачить гравець.

### 4.5 Інтерфейс користувача та тестування програмного продукту

Якщо дивитися на гру з точки зору гравця, то інтерфейс це перше, з чим він постійно взаємодіє. І навіть при якісному геймплею інтерфейс може зіпсувати загальне враження. Тому під час розробки «Blade of Destiny» довелося приділити цьому більше уваги, ніж здавалось на початку. Стає помітно, що зайві елементи інтерфейсу тільки перевантажують екран і відволікають від самої гри, тому в підсумку було залишено лише те, що дійсно потрібне під час проходження: здоров’я персонажа, ресурси та елементи, пов’язані з прогресом.


Рис. 4.12 Інтерфейс користувача у грі «Blade of Destiny»

На рисунку 4.12 показано загальний вигляд інтерфейсу під час гри. Всі елементи розміщені так, щоб не перекривати основну сцену та не відволікати від проходження рівня. Як показала практика, навіть невеликі зміни розташування можуть впливати на сприйняття, тому цей момент доводилось кілька разів переробляти [9].

Інтерфейс реалізовано за допомогою системи UMG, яка дозволяє досить швидко створювати і змінювати елементи без необхідності постійного втручання в код. Це було зручно під час тестування, коли потрібно було оперативно змінити відображення або додати новий елемент.

Окремо варто сказати про зворотний зв’язок. Гравець повинен розуміти, що відбувається в грі без додаткових пояснень. Тому були додані візуальні реакції на дії: отримання шкоди, виконання атаки, зміна стану персонажа. Це дрібні речі, але саме вони роблять гру більш живою. Після реалізації основних механік і інтерфейсу наступним важливим етапом стало тестування. І тут вже стало зрозуміло, що без цього стабільної роботи гри досягти неможливо.

Спочатку тестування виглядало досить просто запуск гри і перевірка, чи працюють основні механіки. Але з часом стало ясно, що цього недостатньо. Навіть якщо система може виглядати працездатною, під час проходження рівня

можуть з’являтись проблеми, які не видно одразу. Для перевірки коректності роботи ігрових механік, взаємодії об’єктів та стабільності гри було реалізовано процес тестування, представлений на рисунку 4.13.


Рис. 4.13 Процес тестування гри у середовищі Unreal Engine 4

У процесі тестування перевірялися основні сценарії:

- рух і стрибки персонажа
- взаємодія з платформами та об’єктами
- бойова система
- проходження рівнів

У процесі тестування особливо часто доводилось повертатися до перевірки колізій та взаємодії між об’єктами, оскільки саме ці елементи найчастіше працювали нестабільно під час реальної гри. Також поступово стало помітно, наскільки сильно на стабільність впливає кількість об’єктів і логіки в сцені. Через це окремі системи доводилося оптимізувати, прибирати зайві обчислення або спрощувати окремі механіки для забезпечення стабільнішої роботи гри.

Для систематизації перевірок, які виконувались під час тестування гри, основні напрями тестування та їх призначення наведено в таблиці 4.2.

Таблиця 4.2

Основні аспекти тестування гри

| **Об’єкт тестування** | **Що перевіряється** |
|---|---|
| Керування | Реакція на ввід, плавність руху |
| Механіки | Коректність роботи стрибків і атак |
| Взаємодія | Робота колізій і тригерів |
| Рівні | Прохідність і баланс |
| Продуктивність | Стабільність роботи гри |

Наведена таблиця дозволяє узагальнити основні напрями перевірки та структурувати процес тестування. Як показав процес розробки, тестування це не окремий етап, який робиться в кінці, а постійний процес. Практично після кожної зміни доводилося перевіряти, чи не чи не виникли нові помилки.

Інтерфейс і тестування напряму впливають на те, як сприймається гра. Навіть при хорошій ідеї і механіках без цих двох елементів гра не буде виглядати завершеною. Саме тому їм було приділено окрему увагу під час розробки «Blade of Destiny».

# ВИСНОВКИ

Результатом виконаної кваліфікаційної роботи стало створення 2D-платформера «Blade of Destiny», розробленого з використанням Unreal Engine 4. У грі реалізовано основні механіки жанру платформер, зокрема рух персонажа, стрибки, взаємодію з ігровими об’єктами та базову бойову систему. Також створено ігрові рівні з поступовим підвищенням складності проходження. Розроблений програмний продукт є працездатним, стабільно функціонує та забезпечує зрозумілий ігровий процес для користувача.

У ході виконання кваліфікаційної роботи було досягнуто наступних результатів:

1. Проведено аналіз жанру 2D-платформерів, досліджено особливості сучасних ігор даного типу, розглянуто популярні механіки та підходи до побудови ігрового процесу. Виконано порівняння існуючих проєктів, визначено їх сильні та слабкі сторони, що дозволило сформувати загальну концепцію власної гри.
2. Визначено основні функціональні вимоги до гри, зокрема механіки переміщення персонажа, взаємодію з об’єктами, систему проходження рівнів та організацію взаємодії користувача з ігровим середовищем. Також спроєктовано загальну структуру гри та архітектуру програмного продукту.
3. Обрано інструменти розробки Unreal Engine 4, мову програмування C++ та систему Blueprint, які забезпечують ефективну реалізацію ігрових механік, швидке тестування функціоналу та зручну інтеграцію компонентів гри.
4. Реалізовано програмний продукт «Blade of Destiny», у якому створено систему керування персонажем, взаємодію з ігровими об’єктами, базову бойову систему, ворогів, елементи інтерфейсу користувача та ігрові рівні. Для окремих механік було проведено доопрацювання та оптимізацію з метою забезпечення коректної роботи гри.
5. Проведено тестування програмного продукту, під час якого перевірено стабільність роботи основних механік, взаємодію між компонентами

системи та коректність проходження рівнів. У процесі тестування було виявлено та усунено помилки, що дозволило покращити стабільність ігрового процесу та загальну якість роботи застосунку.

Таким чином, поставлені у кваліфікаційній роботі завдання виконано у повному обсязі, а розроблений програмний продукт може слугувати основою для подальшого розвитку. У майбутньому можливе розширення функціоналу гри шляхом додавання нових рівнів, механік, типів ворогів, системи збереження прогресу та покращення загального геймплею.

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

1. Кушнір М.В., Куклінський М.В. «Розробка 2D-платформера «Blade of Destiny» засобами Unreal Engine.» Матеріали II Всеукраїнської науковотехнічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційнокомунікаційних технологій. Збірник тез. - К.: ДУІКТ, 2025, С.399-401.

2. Кушнір М.В., Куклінський М.В. «Забезпечення безпеки та побудова архітектури 2D-платформера «Blade of Destiny» в Unreal Engine.» Матеріали Всеукраїнської науковотехнічної конференції «Застосування програмного забезпечення в інформаційнокомунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. ДУІКТ, 2026, С.138-139.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Fullerton T. Game Design Workshop: A Playcentric Approach to Creating Innovative Games. 5th ed. Boca Raton : CRC Press, 2024. 586 p.
2. Rogers S. Level Up! The Guide to Great Video Game Design. 3rd ed. Hoboken : Wiley, 2024. 600 p.
3. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. Boca Raton : CRC Press, 2019. 640 p.
4. Adams E. Fundamentals of Game Design. 3rd ed. Berkeley : New Riders, 2014. 576 p.
5. Keith C. Agile Game Development with Scrum. Upper Saddle River : Addison-Wesley, 2020. 312 p.
6. Gregory J. Game Engine Architecture. 3rd ed. Boca Raton : CRC Press, 2018. 1218 p.
7. Gregory J. Game Engine Architecture. 2nd ed. Boca Raton : CRC Press, 2014. 1052 p.
8. Nystrom R. Game Programming Patterns. Genever Benning, 2014. 354 p.
9. Hodent C. The Gamer’s Brain: How Neuroscience and UX Can Impact Video Game Design. Boca Raton : CRC Press, 2017. 390 p.
10. Brathwaite B., Schreiber I. Challenges for Game Designers. Boston : Course Technology, 2009. 317 p.
11. Epic Games. Unreal Engine 4.27 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-4-27-documentation?application_version=4.27 (дата звернення: 10.05.2026).
12. Epic Games. Blueprint Visual Scripting. URL: https://dev.epicgames.com/documentation/en-us/unrealengine/blueprint-visual-scripting?application_version=4.27 (дата звернення: 10.05.2026).
13. Epic Games. Blueprints in Unreal Engine. URL: https://dev.epicgames.com/documentation/unreal-engine/blueprints?application_version=4.27 (дата звернення: 10.05.2026).

14. Epic Games. Paper 2D in Unreal Engine. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/paper-2d?application_version=4.27 (дата звернення: 11.05.2026).
15. Epic Games. Paper 2D Content Examples. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/paper-2d-content-examples?application_version=4.27 (дата звернення: 11.05.2026).
16. Stroustrup B. The C++ Programming Language. 4th ed. Boston : Addison-Wesley, 2013. 1366 p.
17. Williams A. C++ Concurrency in Action. 2nd ed. Shelter Island : Manning Publications, 2019. 592 p.
18. ISO/IEC 14882:2020. Programming Languages — C++. Geneva : International Organization for Standardization, 2020. 1818 p.
19. Chacon S., Straub B. Pro Git. 2nd ed. New York : Apress, 2014. 456 p.
20. Microsoft. Git documentation. URL: https://learn.microsoft.com/en-us/devops/develop/git/what-is-git (дата звернення: 11.05.2026).
21. Team Cherry. Hollow Knight. URL: https://store.steampowered.com/app/367520/Hollow_Knight/ (дата звернення: 12.05.2026).
22. Maddy Makes Games Inc. Celeste. URL: https://store.steampowered.com/app/504230/Celeste/ (дата звернення: 12.05.2026).
23. Asantee Games. Magic Rampage. URL: https://store.steampowered.com/app/704920/Magic_Rampage/ (дата звернення: 12.05.2026).

# ДОДАТОК А. ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

## Розробка 2D-платформера «Blade of Destiny» засобами Unreal Engine

Виконав студент 4 курсу
групи ПД-42
Кушнір Михайло Володимирович
Керівник роботи
професор кафедри ІТ, канд.техн.наук Куклінський Максим Володимирович

Київ - 2026

# МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

**Мета роботи:** підвищити ефективність взаємодії користувача з 2D-платформером шляхом реалізації варіативного проходження, інтерактивних ігрових механік та динамічного ігрового процесу.

**Об’єкт дослідження:** процес створення 2D-платформерів із нелінійним проходженням для персональних комп’ютерів.

**Предмет дослідження:** засоби та технології реалізації ігрових механік, системи варіативного проходження, взаємодії гравця з ігровим середовищем і програмної логіки у 2D-платформері «Blade of Destiny».



# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ КЛАСИФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз сучасних 2D-платформерів та особливостей реалізації їх ігрових механік.
2. Дослідити можливості Unreal Engine 4 для створення 2D-ігор та обґрунтувати вибір засобів реалізації програмного продукту.
3. Спроєктувати архітектуру гри «Blade of Destiny» та структуру її основних програмних модулів.
4. Реалізувати систему керування персонажем і бойові механіки, що включають рух, стрибки, взаємодію з платформами, ближній та дальній бій.
5. Реалізувати систему проходження рівнів, секретних локацій та взаємодії з об’єктами ігрового середовища.
6. Реалізувати систему внутрішньоігрового магазину скінів та механізми розвитку персонажа після проходження рівнів.
7. Провести тестування програмного продукту та перевірити стабільність роботи основних ігрових механік.
8. Визначити можливості подальшого розширення функціональних можливостей гри «Blade of Destiny».

# АНАЛІЗ АНАЛОГІВ

| Критерій аналізу | Hollow Knight | Celeste | Magic Rampage | Blade of Destiny |
|---|---|---|---|---|
| Бойова система | Ближній бій | Відсутня (акцент на таймінгах та мікроконтролі) | Ближній та дальній бій з RPG-елементами | Гібридна: ближній та дальній бій залежно від обраного героя |
| Розвиток персонажа | часткове (з сюжетом) | – | з проходженням рівнів | з проходженням рівнів + сюжетом |
| Персоналізація | – | – | скіни | скіни |
| Оточення та рівні | Темні, атмосферні локації | Рівні з акцентом на паркур | Підземелля та бойові рівні | Ліси, підземелля, платформи, замок |
| Приховані локації | + | – | – | + |
| Внутрішньоігровий магазин | Покращення (через прихованого на локаціях торговця) | – | скіни | Скіни, покращення |
| Ігрові персонажі | Один фіксований | Один фіксований | Система класів (вибір архетипу на початку) | Два ігрових персонажі з унікальними мувсетами та механіками бою |



# ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Функціональні вимоги

1. Реалізувати систему керування персонажем (рух, стрибки, взаємодія з платформами).
2. Розробити гібридну бойову систему (ближній/дальній бій, атаки, отримання шкоди).
3. Створити систему рівнів із поступовим ускладненням та логікою переходів.
4. Інтегрувати інтерактивне середовище (рухомі платформи, пастки, перешкоди).
5. Коректна поведінка ворогів (патрулювання, визначення гравця, атака)
6. Налаштувати систему прогресії та покращення характеристик персонажа.
7. Реалізувати внутрішньоігровий магазин для кастомізації скінів.

## Нефункціональні вимоги

1. Забезпечити стабільну роботу гри без критичних збоїв (сесія від 1 години).
2. Оптимізувати продуктивність: ≥ 60 FPS на рекомендованих ПК, ≥ 30 FPS на мінімальних.
3. Налаштувати коректну роботу фізичного рушія та систем колізій об'єктів.
4. Обмежити час завантаження ігрових рівнів до 10 секунд.
5. Гарантувати сумісність гри з ОС Windows 10 та Windows 11.
6. Передбачити модульну архітектуру для подальшого розширення функціоналу.

**Мінімальні системні вимоги:** ОС: Windows 10 Процесор: Intel Core i3 / AMD Ryzen 3 Оперативна пам'ять: 4 GB RAM Відеокарта: NVIDIA GeForce GTX 750 Ti / AMD Radeon RX 560 Місце на диску: 2 GB
**Рекомендовані системні вимоги:** ОС: Windows 10 / 11 Процесор: Intel Core i5 / AMD Ryzen 5 Оперативна пам'ять: 8 GB RAM Відеокарта: NVIDIA GeForce GTX 1060 / AMD Radeon RX 570 Місце на диску: 4 GB

# ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

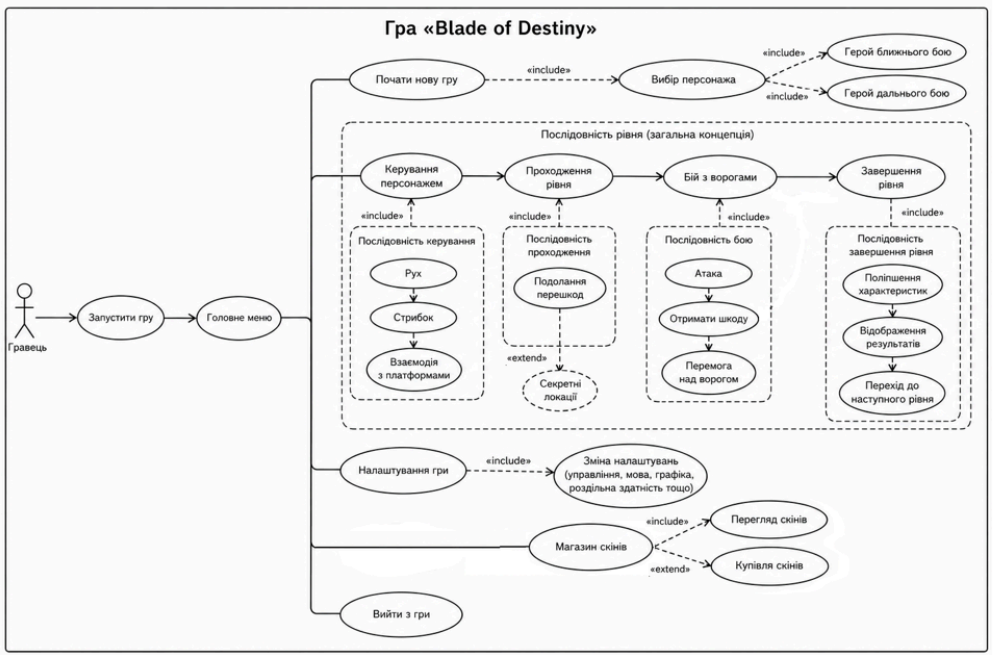





6

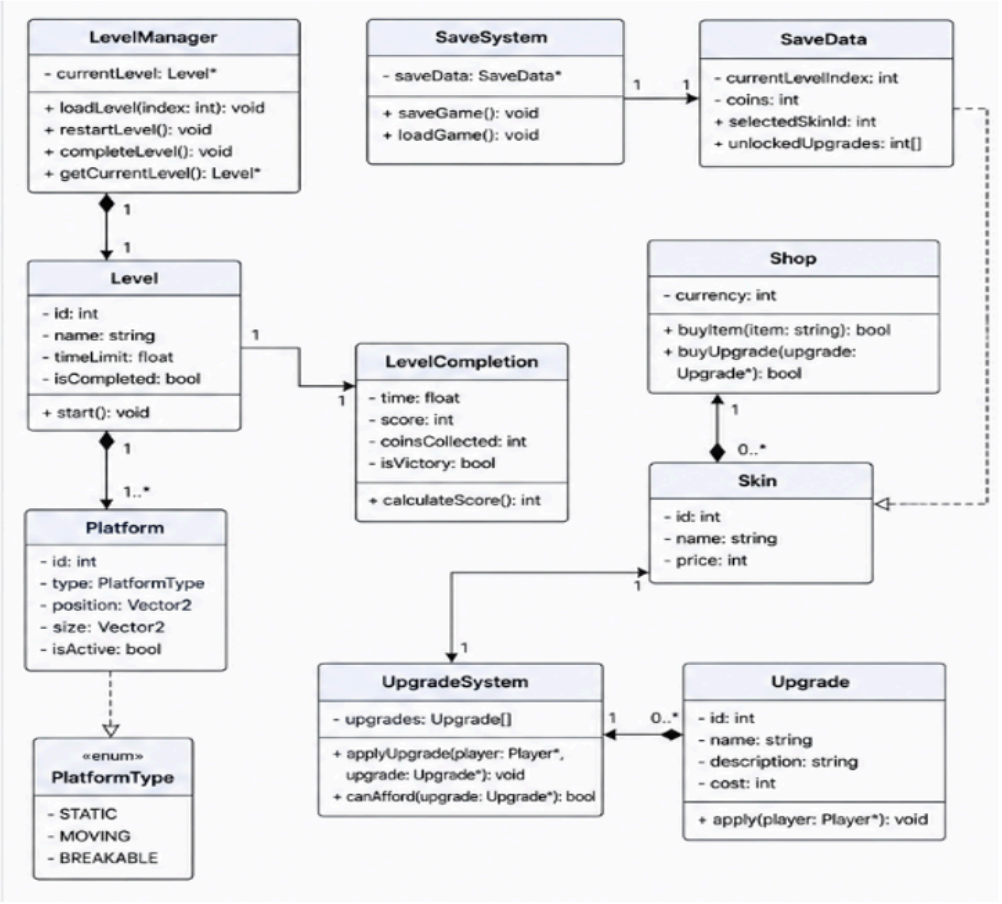
---

# ДІАГРАМА ПРЕЦЕДЕНТІВ

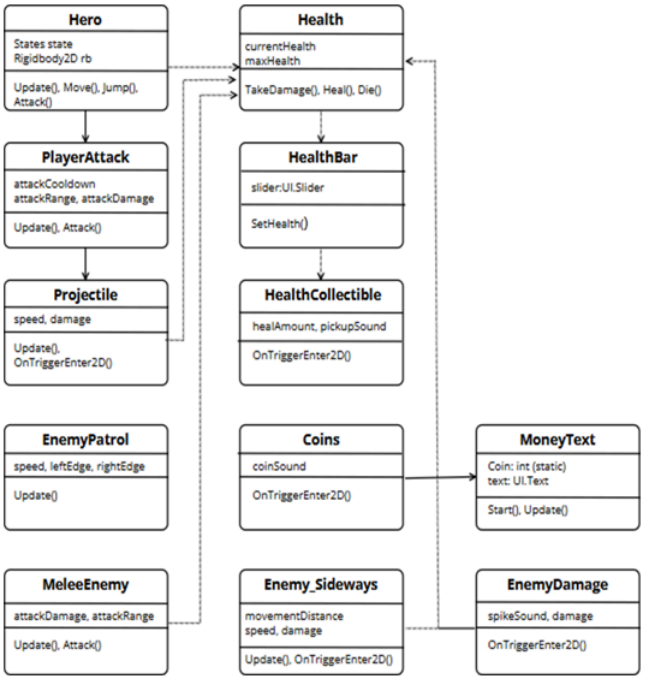



7

# ДІАГРАМА КЛАСІВ: МЕНЕДЖЕРИ ТА ОТОЧЕННЯ



# ДІАГРАМА КЛАСІВ: ГРАВЕЦЬ ТА ВОРОГИ






# Реалізація основних механік

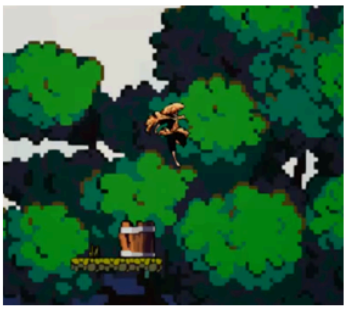


Рис.1.1 Система руху та платформінгу

Рис.1.2 Логіка поведінки противників

Рис.1.3 Реалізація бойової системи


Рис.1.4 Взаємодія з об’єктами рівня

# Інтерфейс користувача та внутрішньоігрові системи


Рис.1.1 Реалізовано внутрішньоігровий магазин (скіни)

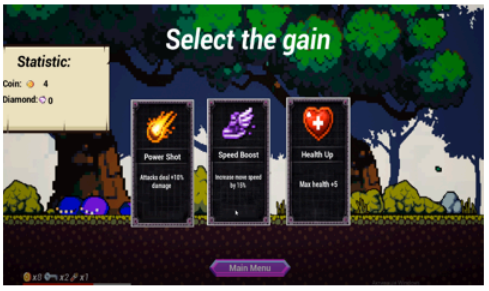



Рис.1.2 Впроваджено систему прокачки персонажа


1.2 Інтерфейс користувача (HUD)

# АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Кушнір М.В., Куклінський М.В. Розробка 2D-платформера «Blade of Destiny» засобами Unreal Engine. Матеріали II Всеукраїнської науковотехнічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційнокомунікаційних технологій. Збірник тез. - К.: ДУІКТ, 2025, С.399-401.
2. Кушнір М.В., Куклінський М.В. Забезпечення безпеки та побудова архітектури 2D-платформера «Blade of Destiny» в Unreal Engine. Матеріали VII Всеукраїнської науковотехнічної конференції «Застосування програмного забезпечення в інформаційнокомунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. ДУІКТ, 2026, С.138-139.



# ВИСНОВКИ

1. Проведено аналіз сучасних 2D-платформерів та визначено особливості реалізації їх основних ігрових механік, зокрема систем пересування персонажа, бойових механік, взаємодії з платформами та ігровим середовищем.
2. Досліджено можливості Unreal Engine 4 для створення 2D-ігор та визначено переваги використання даного рушія для реалізації програмного продукту.
3. Спроєктовано архітектуру гри «Blade of Destiny» та структуру основних програмних модулів із використанням UML-діаграм класів.
4. Реалізовано систему керування персонажем, що включає рух, стрибки, ближній та дальній бій, а також взаємодію з платформами та об’єктами середовища.
5. Розроблено систему проходження рівнів, секретних локацій та взаємодії з елементами ігрового світу.
6. Реалізовано систему внутрішньоігрового магазину скінів, накопичення ігрової валюти та механізми покращення характеристик персонажа після проходження рівнів.
7. Проведено тестування програмного продукту та перевірено стабільність роботи основних ігрових механік і систем гри.
8. Визначено можливості подальшого розвитку гри «Blade of Destiny», зокрема розширення бойової системи, додавання нових типів ворогів, рівнів і функціональних можливостей гри.