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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка ігрового застосунку «Project Mebius» в жанрі Fantasy RPG з елементами стратегії та колекціонування»

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

_____ Ілля ГРЕБЕНЮК
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Ілля ГРЕБЕНЮК

Керівник: Оксана ЗОЛОТУХІНА
канд. техн. наук, доц.

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

Гребенюку Іллі В’ячеславовичу

1. Тема кваліфікаційної роботи: «Розробка ігрового застосунку «Project Mebius» в жанрі Fantasy RPG з елементами стратегії та колекціонування»
керівник кваліфікаційної роботи канд. техн. наук, доцент ЗОЛОТУХІНА Оксана,
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 ігрових застосунків жанру Fantasy RPG, опис методів процедурної генерації ігрового контенту, технічна документація Unity та C#, матеріали реалізованого Unity-проєкту «Project Mebius».

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

1. Аналіз предметної галузі та існуючих ігрових рішень жанру Fantasy RPG.
2. Обґрунтування засобів і методів реалізації ігрового застосунку.
3. Проєктування та програмна реалізація ігрового застосунку «Project Mebius».
4. Тестування ігрового застосунку.

5. Перелік графічного матеріалу: *презентація*

- Аналіз аналогів.
- Функціональні та нефункціональні вимоги.
- Програмні засоби реалізації.
- Інформаційна модель предметної галузі.
- Загальна архітектура застосунку.
- Ігровий цикл застосунку.
- Екранні форми.
- 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	23.02-28.02.2026	
2	Аналіз та дослідження існуючих ігрових аналогів	01.03-07.03.2026	
3	Аналіз вимог до ігрового застосунку	07.03-10.03.2026	
4	Проєктування ігрового застосунку	10.03-10.04.2026	
5	Програмна реалізація застосунку	10.04-06.05.2026	
6	Тестування застосунку	06.05-08.05.2026	
7	Оформлення роботи: вступ, висновки, реферат	08.05-10.05.2026	
8	Розробка демонстраційних матеріалів	10.05-11.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	
10	Подання кваліфікаційна робота	23.05-30.05.2026	

Здобувач вищої освіти _____ Ілля ГРЕБЕНЮК

(підпис)

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 95 стор., 24 табл., 12 рис., 20 джерел.

Мета роботи – розробка ігрового застосунку «Project Mebius» в жанрі Fantasy RPG з елементами стратегії та колекціонування.

Об'єкт дослідження – процес проєктування та розробки ігрових застосунків жанру Fantasy RPG з елементами стратегії та колекціонування.

Предмет дослідження – програмні засоби та методи реалізації ігрового застосунку в середовищі Unity.

Короткий зміст роботи: У роботі проаналізовано особливості жанру Fantasy RPG, існуючі ігрові аналоги та вимоги до застосунку. Обґрунтовано вибір Unity, C#, засобів створення інтерфейсу, процедурної генерації та локального JSON-збереження. Спроєктовано архітектуру, ігровий цикл і основні програмні модулі. Реалізовано генерацію героїв, ворогів і нагород, бібліотеку персонажів, формацію, фракційні синергії, кампанію, башту, автоматизований бій, завдання, AFK-нагороди та розвиток героїв. Проведено тестування основних сценаріїв роботи застосунку.

Сферою використання застосунку є демонстрація програмної реалізації Fantasy RPG з елементами стратегії та колекціонування.

КЛЮЧОВІ СЛОВА: PROJECT MEBIUS, FANTASY RPG, UNITY, C#, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, КОЛЕКЦІОНУВАННЯ ГЕРОЇВ, АВТОБІЙ, JSON

ЗМІСТ

ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ ІГРОВИХ РІШЕНЬ 14
1.1 Особливості жанру Fantasy RPG з елементами стратегії та колекціонування ..14
1.2 Аналіз існуючих ігрових застосунків жанру Fantasy RPG 18
1.3 Визначення функціональних і нефункціональних вимог до ігрового застосунку24
1.4 Інформаційна модель предметної галузі 30
2 ЗАСОБИ ТА МЕТОДИ РЕАЛІЗАЦІЇ ІГРОВОГО ЗАСТОСУНКУ 35
2.1 Середовище розробки Unity 35
2.2 Мова програмування C# 37
2.3 Засоби створення інтерфейсу та візуального оформлення 39
2.4 Методи процедурної генерації в ігрових застосунках 42
2.5 Засоби збереження даних та організації ресурсів проєкту 47
2.6 Допоміжні засоби підготовки, перевірки та збирання проєкту 50
3 ПРОЄКТУВАННЯ, РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ 54
3.1 Загальна архітектура ігрового застосунку 54
3.2 Проєктування ігрового циклу та основних сценаріїв користувача 59
3.3 Проєктування системи процедурної генерації героїв, ворогів і нагород 64
3.4 Реалізація системи колекціонування, формування команди та розвитку героїв69
3.5 Реалізація бойової системи та режимів проходження 75
3.6 Реалізація інтерфейсу користувача та системи збереження даних 81
3.7 Тестування ігрового застосунку 89
ВИСНОВКИ 95
ПЕРЕЛІК ПОСИЛАНЬ 99
ДОДАТОК А ..... **Error! Bookmark not defined.**
ДОДАТОК Б 101

ВСТУП

Ігрова індустрія залишається однією з найбільш динамічних сфер цифрових технологій. Ігри сьогодні розглядаються не лише як розважальні продукти, а й як складні програмні системи, у яких поєднуються графіка, алгоритми генерації, моделювання поведінки, інтерфейс користувача, збереження даних і механіки довготривалої взаємодії з гравцем. За даними Newzoo, у 2025 році світова аудиторія ігор становить близько 3,6 млрд гравців, а обсяг світового ринку оцінюється у 188,8 млрд доларів США [1]. Це підтверджує актуальність розробки нових ігрових застосунків, здатних поєднувати цікаву ігрову механіку, зручний інтерфейс і технічну стабільність.

Окреме місце в сучасній ігровій індустрії займають рольові ігри, зокрема Fantasy RPG. Для цього жанру характерні розвиток персонажів, поступове відкриття нового контенту, формування команди, проходження етапів, отримання нагород і взаємодія з ігровим світом. Такі ігри часто поєднують елементи стратегії та колекціонування, оскільки користувач не лише проходить окремі рівні, а й приймає рішення щодо складу команди, покращення персонажів, використання ресурсів і вибору подальшої тактики. Саме ця комбінація робить жанр зручним для створення довготривалого ігрового процесу, у якому важливе значення мають як випадковість, так і стратегічне планування [2; 3].

Важливою тенденцією у розробці ігор є використання процедурної генерації. Вона дає змогу автоматично створювати або комбінувати ігровий контент: персонажів, ворогів, предмети, нагороди, рівні, події або інші елементи гри. У наукових працях із процедурної генерації зазначається, що такий підхід може застосовуватися для підвищення варіативності гри, зменшення ручної роботи під час створення контенту та забезпечення унікального досвіду для користувача [4; 5; 6]. Для RPG-застосунків процедурна генерація є особливо доцільною, оскільки дозволяє створювати велику кількість героїв, ворогів і бойових ситуацій без необхідності вручну описувати кожен окремий випадок.

Актуальність теми кваліфікаційної роботи зумовлена потребою у створенні ігрового застосунку, який поєднує механіки Fantasy RPG, стратегічного формування команди, колекціонування персонажів і процедурної генерації ігрового контенту. Більшість популярних рольових ігор використовують значну кількість готового контенту, складні системи прогресії та різноманітні бойові механіки. Проте для дипломного проєкту важливо не просто відтворити зовнішню форму жанру, а реалізувати зрозумілу програмну структуру, у якій основні системи гри працюють як єдиний застосунок: генерація героїв, створення ворогів, формування команди, проходження етапів, автоматизований бій, отримання нагород і збереження прогресу.

У межах кваліфікаційної роботи розробляється ігровий застосунок «Project Mebius» в жанрі Fantasy RPG з елементами стратегії та колекціонування. Застосунок реалізовано в середовищі Unity з використанням мови програмування C#. Unity є одним із поширених ігрових рушіїв, що надає засоби для створення 2D- та 3D-застосунків, роботи з графічними ресурсами, інтерфейсом, сценами, анімаціями, звуком і програмною логікою [12]. Мова C# підтримує об’єктно-орієнтований підхід, що є зручним для опису ігрових сутностей, сервісів, моделей даних і взаємодії між модулями [15].

Розроблений застосунок орієнтований на 2D pixel fantasy стиль і містить набір взаємопов’язаних ігрових систем. У ньому реалізовано процедурну генерацію героїв, ворогів і нагород, систему колекціонування персонажів, гача-механіку, формування бойової команди, фракційні синергії, кампанію, режим башти, автоматизовану бойову систему, завдання, тренування героїв, підвищення зірковості та збереження прогресу користувача. Також у застосунку передбачено україномовний інтерфейс, що підвищує доступність програмного продукту для українських користувачів.

Аналіз останніх досліджень і публікацій показує, що питання створення ігрових систем розглядаються з різних позицій: як частина програмної інженерії, як напрям інтерактивного дизайну, як задача процедурної генерації контенту та як процес побудови користувацького досвіду. У роботах, присвячених процедурній

генерації, розглядаються способи автоматичного створення ігрового контенту та оцінювання якості результатів генерації [4; 5; 6]. У матеріалах і документації Unity описано практичні підходи до створення інтерактивних застосунків, організації ресурсів, побудови інтерфейсу та збереження даних [12; 13; 14; 16; 17]. Водночас розробка конкретного ігрового застосунку потребує поєднання цих підходів у межах єдиної архітектури, адаптованої до обраного жанру, технічних можливостей і поставлених вимог.

Об’єктом дослідження є процес проєктування та розробки ігрових застосунків жанру Fantasy RPG з елементами стратегії та колекціонування.

Предметом дослідження є програмні засоби та методи реалізації ігрового застосунку в середовищі Unity, що охоплюють процедурну генерацію персонажів і ворогів, формування команди, автоматизовану бойову систему, систему прогресії, інтерфейс користувача та збереження ігрових даних.

Метою роботи є розробка ігрового застосунку «Project Mebius» в жанрі Fantasy RPG з елементами стратегії та колекціонування, який поєднує процедурну генерацію ігрового контенту, систему колекціонування героїв, стратегічне формування команди, автоматизовані бої, прогресію персонажів і збереження результатів гри.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. проаналізувати особливості жанру Fantasy RPG з елементами стратегії та колекціонування;
2. розглянути існуючі ігрові рішення, близькі за жанром, механіками та принципами розвитку персонажів;
3. визначити функціональні та нефункціональні вимоги до ігрового застосунку;
4. обґрунтувати вибір Unity, C# та інших засобів реалізації;
5. спроєктувати архітектуру застосунку, ігровий цикл і основні програмні модулі;
6. реалізувати системи процедурної генерації героїв, ворогів, етапів і нагород;

7. реалізувати механіки колекціонування, формування команди, розвитку героїв, кампанії, башти та автоматизованого бою;

8. створити інтерфейс користувача, систему збереження даних і базові налаштування застосунку;

9. провести тестування основних функціональних можливостей гри.

У роботі використано методи аналізу предметної галузі, порівняльного аналізу існуючих ігрових рішень, об’єктно-орієнтованого проєктування, процедурної генерації, UML-моделювання, модульної організації програмної логіки та функціонального тестування. Застосування цих методів дозволило послідовно перейти від аналізу жанру та вимог до створення програмної реалізації і перевірки працездатності основних механік.

Практична новизна роботи полягає у розробці ігрового застосунку, в якому в межах одного Unity-проєкту поєднано процедурну генерацію героїв, ворогів і нагород, гача-механіку, стратегічне формування команди, фракційні синергії, автоматизовану бойову систему, AFK-нагороди, завдання, розвиток персонажів і JSON-збереження прогресу. Особливістю реалізації є те, що більшість основних систем працюють у межах єдиного runtime-модуля, який створює інтерфейс, керує ігровими екранами та забезпечує взаємодію між підсистемами.

Практичне значення роботи полягає в тому, що розроблений застосунок може використовуватися як робоча версія ігрового продукту, який демонструє основні принципи побудови Fantasy RPG з елементами стратегії та колекціонування. Отримані результати можуть бути використані для подальшого розширення гри: додавання нових героїв, режимів проходження, бойових ефектів, сюжетних елементів, системи досягнень або покращеної візуальної частини. Крім того, структура проєкту може бути використана як приклад реалізації процедурної генерації та ігрової прогресії в Unity.

Апробація результатів роботи здійснюється шляхом демонстрації працездатності розробленого застосунку, перевірки основних ігрових сценаріїв, тестування генерації персонажів і ворогів, формування команди, проходження бою, отримання нагород і збереження прогресу. Результати реалізації

представлено у вигляді програмного продукту, пояснювальної записки, демонстраційних матеріалів і лістингів основних програмних модулів.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань і додатків. У першому розділі розглянуто предметну галузь, особливості жанру Fantasy RPG, існуючі ігрові рішення та вимоги до застосунку. У другому розділі обґрунтовано вибір засобів і методів реалізації, зокрема Unity, C#, інструментів створення інтерфейсу, процедурної генерації та збереження даних. У третьому розділі описано проєктування, реалізацію та тестування ігрового застосунку, включаючи архітектуру, ігровий цикл, генерацію, систему героїв, бойову систему, інтерфейс і перевірку функціональності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ ІГРОВИХ РІШЕНЬ

1.1 Особливості жанру Fantasy RPG з елементами стратегії та колекціонування

Ігрові застосунки жанру Fantasy RPG належать до рольових ігор, у яких користувач поступово розвиває персонажів, проходить різні етапи, отримує нагороди та відкриває нові можливості. На відміну від ігор, побудованих лише на швидкій реакції або коротких рівнях, рольові ігри мають довший цикл проходження. Гравець повертається до гри не тільки для проходження чергового бою, а й для покращення команди, накопичення ресурсів, отримання нових героїв та підготовки до складніших випробувань [2; 3].

Fantasy-напрям у рольових іграх передбачає використання вигаданого світу, у якому можуть існувати магія, різні фракції, фантастичні істоти, герої з особливими здібностями, підземелля, башти або інші умовні локації. Така тематика дає розробнику більше свободи у створенні персонажів і механік. У ній не обов’язково повністю дотримуватися реалістичних правил, однак ігровий світ усе одно повинен мати внутрішню логіку. Користувач має розуміти, чому певний герой виконує роль захисника, інший є атакувальним персонажем, а ще один призначений для підтримки команди.

Однією з головних ознак RPG є система характеристик персонажів. Зазвичай кожен герой має показники здоров’я, атаки, захисту, швидкості, магічної сили або інших параметрів, які впливають на його поведінку в бою. Через ці характеристики формується відмінність між ролями персонажів. Наприклад, герой із високим запасом здоров’я краще підходить для передньої лінії, а персонаж із високою атакою або магічною силою може бути корисним для завдання основної шкоди ворогам. Якщо в грі є цілителі або персонажі підтримки, вони не завжди мають найбільшу силу атаки, але можуть значно підвищувати виживання всієї команди.

Стратегічна складова в таких іграх проявляється не обов’язково у прямому керуванні кожною дією під час бою. Часто основне рішення користувач приймає ще до початку битви: обирає героїв, розставляє їх по позиціях, визначає, кого потрібно розвивати першим, і вирішує, на що витратити ресурси. Тому навіть автоматизований бій може мати стратегічний зміст, якщо його результат залежить від підготовки команди, ролей персонажів, їхнього рівня та взаємодії між ними.

Для командних RPG важливим є розміщення героїв. Передня лінія зазвичай призначена для персонажів, які мають приймати на себе основну частину пошкоджень. Задня лінія більше підходить для магів, лучників, цілителів або інших героїв, які повинні діяти з безпечнішої позиції. Такий принцип робить формування команди не випадковим, а залежним від логіки бою. Якщо користувач поставить слабкого героя на передню позицію, команда може швидко втратити важливого персонажа. Якщо ж у складі немає атакувальних героїв, бій може затягнутися або завершитися поразкою.

У багатьох Fantasy RPG використовується система фракцій. Вона виконує не лише декоративну роль, а й допомагає урізноманітнити ігрову стратегію. Герої можуть належати до різних груп, орденів, кланів або стихій, а поєднання кількох персонажів однієї фракції може давати додаткові переваги. Завдяки цьому гравець оцінює команду не тільки за силою окремих героїв, а й за тим, як вони працюють разом. Це робить систему колекціонування більш змістовною, оскільки навіть герой середнього рівня може бути корисним через свою роль або фракційну належність.

Колекціонування персонажів є ще однією важливою рисою цього жанру. Користувач поступово отримує нових героїв, переглядає їх у бібліотеці, порівнює характеристики, додає окремих персонажів до команди або залишає їх для майбутнього розвитку. У таких іграх сама поява нового героя часто сприймається як нагорода. Особливо це стосується персонажів вищої рідкісності, які мають кращі характеристики, цікавішу роль або корисні здібності.

Для отримання нових персонажів часто застосовується система випадкового заклику. Вона дозволяє зробити процес поповнення колекції непередбачуваним.

Водночас така механіка потребує балансу. Якщо сильні герої випадають занадто часто, вони перестають сприйматися як цінні. Якщо ж шанс отримання рідкісного героя надто низький, користувач може втратити інтерес. Тому система випадкового отримання персонажів повинна мати зрозумілу логіку й бути пов’язаною з ресурсами, які гравець заробляє під час проходження.

Окрему роль у Fantasy RPG відіграє прогресія. Вона може включати підвищення рівня героя, покращення характеристик, відкриття нових здібностей, підвищення рангу або зірковості. Завдяки цьому навіть початковий персонаж може залишатися корисним, якщо користувач вкладає в нього ресурси. Прогресія також створює відчуття поступового руху вперед, оскільки після кожного етапу гравець отримує певний результат: ресурси, досвід, нового героя або можливість пройти складніший рівень.

Автоматизована бойова система часто використовується в RPG із великою кількістю повторюваних боїв. У такому випадку гравець не керує кожною атакою вручну, а спостерігає за результатом роботи підготовленої команди. Це зручно для ігор, де передбачено багато етапів кампанії, башти або інших режимів проходження. Проте автобій не повинен бути повністю випадковим. Його результат має залежати від характеристик героїв, їхніх ролей, швидкості, здібностей, фракційних бонусів та загального рівня підготовки команди.

Також для цього жанру характерна наявність кількох режимів проходження. Основна кампанія зазвичай використовується для поступового відкриття контенту та зростання складності. Додатковий режим, наприклад башта, може слугувати окремим випробуванням для вже сформованої команди. Такі режими дають користувачеві більше варіантів дій і дозволяють не обмежувати весь ігровий процес лише одним типом боїв.

Система ресурсів у Fantasy RPG пов’язує між собою більшість інших механік. Ресурси можуть витрачатися на заклик героїв, тренування, підвищення рівня, покращення зірковості або інші дії. Через це користувач постійно приймає невеликі стратегічні рішення: накопичувати ресурси, витратити їх на нового героя

або посилити вже наявного персонажа. Саме такі рішення підтримують інтерес до гри між боями.

У деяких рольових іграх використовуються idle- або AFK-механіки. Вони дозволяють отримувати частину нагород за час, коли користувач не бере активної участі у грі. Такий підхід добре поєднується з колекційними RPG, оскільки допомагає зменшити кількість одноманітного повторення одних і тих самих дій. Водночас AFK-нагороди повинні мати обмеження, щоб користувач не отримував надмірну перевагу без проходження основного ігрового контенту.

З технічної точки зору Fantasy RPG з елементами стратегії та колекціонування є досить складним типом застосунку. У ньому потрібно поєднати моделі героїв, ворогів, ресурсів, нагород, етапів, бойових одиниць, інтерфейсу та збереження прогресу. Особливо важливо, щоб ці частини не існували окремо. Наприклад, герой, отриманий через заклик, повинен з’явитися в колекції, мати власні характеристики, додаватися до команди, брати участь у бою та зберігатися після перезапуску гри.

У проєкті «Project Mebius» жанр Fantasy RPG обрано тому, що він дозволяє поєднати рольову прогресію, стратегічне формування команди та колекціонування героїв у межах одного ігрового застосунку. Для такого проєкту важливими є не лише окремі екрани або візуальні елементи, а й цілісний ігровий цикл. Користувач отримує героя, переглядає його характеристики, додає до формації, проходить бій, отримує нагороду, витрачає ресурси на розвиток і продовжує рух далі.

Отже, Fantasy RPG з елементами стратегії та колекціонування поєднує кілька взаємопов’язаних напрямів: розвиток персонажів, командне планування, випадкове отримання нових героїв, автоматизовані бої, проходження етапів і накопичення ресурсів. Саме через таку багатокомпонентність цей жанр є зручним для демонстрації навичок проєктування ігрового застосунку, реалізації процедурної генерації, роботи з даними, інтерфейсом і системою збереження прогресу.

1.2 Аналіз існуючих ігрових застосунків жанру Fantasy RPG

На сучасному ринку ігор існує значна кількість рольових застосунків, у яких поєднуються фантазійний сетинг, розвиток персонажів, командні бої та колекціонування героїв. Такі ігри можуть відрізнятися стилем графіки, складністю бойової системи, кількістю режимів і підходом до прогресії, однак їх об’єднує спільна ідея: користувач поступово збирає персонажів, покращує їх і використовує в різних бойових ситуаціях [7; 8; 9; 10].

Для аналізу було обрано кілька відомих ігрових застосунків, які мають близькі жанрові ознаки до «Project Mebius»: AFK Arena, RAID: Shadow Legends, Summoners War та Epic Seven. Їх розгляд дозволяє визначити основні підходи до побудови системи героїв, командного бою, розвитку персонажів і нагород.

AFK Arena — це рольова гра з елементами idle-механік, у якій користувач збирає героїв, формує команду та проходить етапи кампанії. Бої в грі відбуваються переважно автоматично, тому гравець більше уваги приділяє підготовці складу, розміщенню персонажів і покращенню їхніх характеристик. У грі також реалізовано систему пасивного накопичення ресурсів, завдяки якій користувач може отримувати нагороди навіть після певної перерви в грі [7].

Перевагою AFK Arena є простий і зрозумілий ігровий цикл. Користувач отримує героя, додає його до команди, проходить бій, одержує ресурси та використовує їх для подальшого розвитку. Такий підхід добре підтримує довготривалу прогресію і не вимагає постійного ручного керування кожним боєм. До недоліків можна віднести залежність розвитку від випадкового отримання сильних героїв і поступове уповільнення прогресу на складніших етапах. Приклад інтерфейсу гри AFK Arena наведено на рисунку 1.1.

Рис. 1.1 Інтерфейс гри AFK Arena

RAID: Shadow Legends є прикладом більш складної колекційної RPG. У грі представлено велику кількість персонажів, які відрізняються фракціями, рідкістю, ролями, уміннями та спорядженням. Користувач формує команди для проходження кампанії, підземель, битв із босами та інших режимів. Значну роль відіграє не лише рівень героя, а й правильний вибір екіпірування, поєднання персонажів і використання їхніх здібностей [8].

Сильною стороною RAID: Shadow Legends є глибока система розвитку персонажів. Гравець має багато варіантів для покращення команди, тому може змінювати стратегію залежно від режиму або типу ворогів. Водночас така кількість систем робить гру складнішою для нового користувача. Для повного розуміння механік потрібно враховувати багато параметрів, що може ускладнювати перше знайомство із застосунком. Приклад ігрового екрана RAID: Shadow Legends подано на рисунку 1.2.

Рис. 1.2 Ігровий екран RAID: Shadow Legends

Summoners War також належить до ігор, побудованих навколо колекціонування персонажів і командних боїв. У ній користувач отримує істот, розвиває їх, покращує характеристики та використовує в різних режимах. Важливою особливістю є те, що користь персонажа не завжди визначається тільки його рідкістю. Деякі герої нижчого рангу можуть бути ефективними завдяки своїм умінням, ролі в команді або правильному поєднанню з іншими персонажами [9].

Перевагою Summoners War є помітна роль командної стратегії. Гравець має враховувати характеристики персонажів, їхні здібності та сумісність у складі. Це робить гру більш гнучкою, оскільки різні герої можуть залишатися корисними на різних етапах проходження. Недоліком є те, що для ефективного розвитку потрібно багато часу на покращення персонажів і підбір оптимальних комбінацій. Приклад інтерфейсу Summoners War показано на рисунку 1.3.

Рис. 1.3 Інтерфейс гри Summoners War

Epic Seven вирізняється серед розглянутих ігор більш виразною візуальною подачею. У грі велика увага приділяється зовнішньому вигляду персонажів, анімаціям, бойовим ефектам і загальному стилю. Кожен герой сприймається не лише як набір характеристик, а як окремий персонаж із власним образом. Це підсилює інтерес до колекціонування, оскільки користувач отримує не тільки нову бойову одиницю, а й нового візуально впізнаваного героя [10].

Перевагою Epic Seven є якісне поєднання рольової системи, колекціонування та художнього оформлення. Гра добре показує, що для Fantasy RPG важливими є не тільки числові характеристики, а й атмосфера, стиль персонажів і подача бою. Водночас сильна залежність від візуальної частини потребує значної кількості графічних матеріалів, анімацій і художньої роботи. Приклад ігрового екрана Epic Seven наведено на рисунку 1.4.

Рис. 1.4 Ігровий екран Epic Seven

Для порівняння основних можливостей розглянутих ігор із «Project Mebius» сформовано таблицю 1.1.

Таблиця 1.1

Порівняльна характеристика існуючих ігрових застосунків та проєкту «Project Mebius»

Застосунок	**Основні риси**	**Переваги**	**Обмеження або особливість**
AFK Arena	Fantasy RPG, idle-прогрес, герої, команда, автобій	Зрозумілий цикл проходження, пасивне отримання ресурсів	Прогрес значною мірою залежить від отриманих героїв
RAID: Shadow Legends	Колекційна RPG, фракції, спорядження, різні режими	Глибока система розвитку персонажів	Велика кількість параметрів ускладнює перше знайомство
Summoners War	Колекціонування істот, командні бої, розвиток персонажів	Важливу роль має підбір команди, а не лише рідкість героя	Потребує тривалого розвитку й підбору комбінацій

Продовження таблиці 1.1

Порівняльна характеристика існуючих ігрових застосунків та проєкту «Project Mebius»

Застосунок	**Основні риси**	**Переваги**	**Обмеження або особливість**
Epic Seven	Fantasy RPG, герої, сюжет, анімації, бойові ефекти	Виразний стиль персонажів і якісна подача бою	Сильна залежність від графічних ресурсів та анімацій
Project Mebius	Fantasy RPG, герої, фракції, кампанія, башта, автобій	Процедурна генерація героїв, ворогів, етапів і нагород	Побудований як власний ігровий застосунок із акцентом на варіативності

Порівняння показує, що розглянуті ігри мають спільну жанрову основу. У кожній із них присутні герої, система розвитку, командні бої, нагороди та поступове ускладнення проходження. Водночас кожен застосунок має власний акцент. AFK Arena більше орієнтована на пасивний прогрес і швидкі ігрові сесії. RAID: Shadow Legends зосереджується на глибокому розвитку персонажів. Summoners War робить важливим правильний підбір команди. Epic Seven виділяється візуальною подачею героїв і боїв.

«Project Mebius» використовує основні жанрові принципи, характерні для таких ігор, але має власну особливість. У застосунку реалізовано процедурну генерацію героїв, ворогів, етапів і нагород. Завдяки цьому ігровий процес не обмежується лише заздалегідь підготовленими персонажами або однаковими бойовими ситуаціями. Користувач отримує різні варіанти героїв, проходить етапи з новими ворогами та накопичує ресурси для розвитку команди.

Отже, аналіз існуючих ігрових застосунків показав, що для Fantasy RPG з елементами стратегії та колекціонування важливими є система персонажів, командна взаємодія, розвиток героїв, випадкове отримання нових одиниць, нагороди, режими проходження та збереження прогресу. У «Project Mebius» ці елементи реалізовано через систему заклику, бібліотеку героїв, формацію,

фракційні синергії, автоматизований бій, кампанію, башту, AFK-нагороди та процедурне формування ігрового контенту.

1.3 Визначення функціональних і нефункціональних вимог до ігрового застосунку

Після аналізу жанру Fantasy RPG та розгляду існуючих ігрових застосунків можна перейти до визначення вимог до «Project Mebius». На цьому етапі важливо описати не лише загальну ідею гри, а й конкретні можливості, які має підтримувати застосунок. Оскільки гра побудована навколо колекціонування героїв, проходження боїв і розвитку команди, вимоги повинні охоплювати всі основні частини ігрового циклу [11].

У «Project Mebius» користувач взаємодіє з кількома основними екранами: головним меню, хабом, бібліотекою героїв, екраном команди, картою кампанії, режимом башти, бойовим екраном, завданнями та налаштуваннями. Кожен із цих екранів виконує окрему роль, але всі вони пов’язані між собою. Наприклад, герой, отриманий через систему заклику, має з’явитися в бібліотеці, після цього його можна додати до формації, використати в бою, а потім розвивати за отримані ресурси.

Окреме місце у вимогах займає процедурна генерація. У грі вона використовується для створення героїв, ворогів, нагород і бойових етапів. Це дозволяє формувати різні ігрові ситуації без повного ручного опису кожного героя або кожного ворога. Для застосунку такого жанру це важливо, оскільки користувач очікує різноманіття персонажів, фракцій, ролей і бойових умов.

Функціональні вимоги до ігрового застосунку «Project Mebius» наведено в таблиці 1.2.

Таблиця 1.2

Функціональні вимоги до ігрового застосунку «Project Mebius»

Позначення	Вимога	Короткий опис
ФВ-1	Головне меню	У застосунку має бути головне меню з можливістю почати гру, перейти до налаштувань, переглянути титри або завершити роботу.
ФВ-2	Ігровий хаб	Після запуску гри користувач має переходити до головного ігрового екрана, з якого доступні основні розділи.
ФВ-3	Система ресурсів	У грі мають використовуватися ресурси для заклику героїв, тренування та покращення персонажів.
ФВ-4	Система заклику героїв	Користувач має отримувати нових героїв за допомогою ігрових ресурсів.
ФВ-5	Генерація героя	Під час створення героя мають визначатися його ім’я, роль, фракція, рідкість, характеристики, здібності та візуальні ознаки.
ФВ-6	Бібліотека героїв	Користувач має переглядати отриманих героїв, їхній рівень, зірковість, роль, фракцію, характеристики та стан.
ФВ-7	Сортування і фільтрація	У бібліотеці мають бути засоби впорядкування героїв за силою, рідкістю, роллю, фракцією або станом.
ФВ-8	Формування команди	Користувач має додавати героїв до бойової формації перед проходженням етапів.
ФВ-9	Обмеження формації	У команді має бути не більше 5 героїв: 2 позиції передньої лінії та 3 позиції задньої лінії.
ФВ-10	Фракційні синергії	Якщо в команді є кілька героїв однієї фракції, система має надавати додаткові бонуси.
ФВ-11	Кампанія	Користувач має проходити етапи кампанії з поступовим збільшенням складності.
ФВ-12	Режим башти	У грі має бути окремий режим, у якому складність зростає з кожним наступним рівнем.
ФВ-13	Генерація ворогів	Для бойового етапу система має створювати групу ворогів відповідно до режиму, етапу та сили команди.
ФВ-14	Боси та елітні вороги	На окремих етапах мають з’являтися сильніші вороги або боси.

Продовження таблиці 1.2

Функціональні вимоги до ігрового застосунку «Project Mebius»

Позначення	Вимога	Короткий опис
ФВ-15	Автоматизований бій	Бій між командою користувача та ворогами має відбуватися автоматично за правилами бойової системи.
ФВ-16	Бойові здібності	Герої мають використовувати базові атаки та спеціальні здібності відповідно до своєї ролі.
ФВ-17	Командна суперздібність	У бою має бути передбачена спільна дія команди після накопичення необхідного заряду.
ФВ-18	Результат бою	Після завершення бою система має визначати перемогу або поразку та показувати результат користувачу.
ФВ-19	Система нагород	Після перемоги користувач має отримувати золото, досвід, матеріали або інші ресурси.
ФВ-20	AFK-нагороди	Застосунок має нараховувати ресурси за час відсутності користувача з обмеженням накопичення.
ФВ-21	Завдання	У грі мають бути завдання, за виконання яких користувач отримує додаткові нагороди.
ФВ-22	Тренування героїв	Користувач має підвищувати рівень героя за відповідні ресурси.
ФВ-23	Підвищення зірковості	У грі має бути можливість підвищення зірковості героя до встановленої межі.
ФВ-24	Налаштування	Користувач має змінювати гучність, швидкість бою, анімації та інші параметри гри.
ФВ-25	Збереження прогресу	Застосунок має зберігати героїв, ресурси, формацію, проходження кампанії, башти, завдання та налаштування.

Функціональні вимоги показують, які дії має підтримувати застосунок. Основними для «Project Mebius» є заклик і генерація героїв, формування команди, проходження боїв, отримання нагород і розвиток персонажів. Саме ці дії створюють основний цикл гри: користувач отримує героя, додає його до команди, проходить етап, отримує ресурси й використовує їх для подальшого посилення складу.

Нефункціональні вимоги описують технічні та якісні характеристики роботи застосунку. Вони потрібні для того, щоб гру можна було не тільки реалізувати, а й перевірити за конкретними параметрами. Для «Project Mebius» важливими є версія середовища розробки, платформа запуску, роздільна здатність, збереження даних, обмеження бойової системи та стабільність роботи. Нефункціональні вимоги наведено в таблиці 1.3.

Таблиця 1.3

Нефункціональні вимоги до ігрового застосунку «Project Mebius»

Позначення	**Вимога**	**Короткий опис**
НФВ-1	Середовище розробки	Проєкт реалізується в Unity 6000.4.2f1.
НФВ-2	Мова програмування	Основна програмна логіка реалізується мовою C#.
НФВ-3	Платформа запуску	Застосунок має запускатися в Unity Editor та у Windows x64-збірці.
НФВ-4	Роздільна здатність	Базова роздільна здатність інтерфейсу — 1920×1080 пікселів.
НФВ-5	Орієнтація екрана	Основна орієнтація екрана — горизонтальна.
НФВ-6	Мова інтерфейсу	Основні меню, кнопки, повідомлення, бойові записи та назви розділів подаються українською мовою.
НФВ-7	Схема команди	У бойовій формації використовується 5 позицій: 2 передні та 3 задні.
НФВ-8	Кількість ворогів	На одному бойовому етапі формується від 3 до 5 ворогів.
НФВ-9	Босові етапи	Бос з’являється на кожному десятому етапі кампанії або башти.

Продовження таблиці 1.3

Нефункціональні вимоги до ігрового застосунку «Project Mebius»

Позначення	**Вимога**	**Короткий опис**
НФВ-10	Межа бою	Автоматизована симуляція бою має завершуватися не пізніше ніж після 160 раундів.
НФВ-11	Максимальна зірковість	Максимальна зірковість героя — 7.
НФВ-12	Максимальний рівень	Максимальний рівень героя не перевищує 100.
НФВ-13	Обмеження AFK-нагород	Накопичення AFK-нагород обмежується 12 годинами.
НФВ-14	Формат збереження	Дані користувача зберігаються у форматі JSON.
НФВ-15	Місце збереження	Файл збереження розміщується в локальній директорії застосунку, доступній через Application.persistentDataPath.
НФВ-16	Повнота завантаження	Після перезапуску гри мають відновлюватися герої, ресурси, формація, прогрес кампанії, прогрес башти, завдання та налаштування.
НФВ-17	Час переходу між екранами	Перехід між основними екранами має виконуватися орієнтовно до 1 секунди за стандартного сценарію використання.
НФВ-18	Локальні ресурси	Графічні, звукові та інтерфейсні ресурси мають завантажуватися з локальних папок проєкту.
НФВ-19	Стабільність	Під час стандартного проходження застосунок не повинен аварійно завершувати роботу.
НФВ-20	Перевірка основних механік	Генерація героїв, генерація ворогів, формація, бій, нагороди, AFK-нарахування та збереження мають перевірятися тестовими сценаріями.

У таблиці 1.3 наведено вимоги, які можна перевірити під час тестування. Наприклад, схема формації перевіряється кількістю доступних позицій у команді, межа бою — максимальною кількістю раундів симуляції, а збереження —

відновленням героїв, ресурсів і прогресу після повторного запуску гри. Такі вимоги роблять опис застосунку точнішим і дають змогу оцінити, чи відповідає реалізація поставленим умовам.

Окремо слід виділити вимоги до збереження даних. Для гри з колекціонуванням персонажів це одна з основних частин, оскільки користувач поступово накопичує героїв, ресурси та результати проходження. Якщо ці дані не відновлюються після перезапуску, порушується сама логіка прогресії. Тому в «Project Mebius» передбачено локальне JSON-збереження, яке має містити основні дані користувача.

Також важливими є обмеження, пов’язані з балансом гри. Кількість героїв у команді, кількість ворогів, максимальна зірковість, межа рівня, босові етапи та обмеження AFK-нагород потрібні для того, щоб ігровий процес залишався контрольованим. Без таких обмежень гра могла б створювати некоректні ситуації, наприклад надто довгий бій або надмірне накопичення ресурсів без участі користувача.

Отже, визначені функціональні та нефункціональні вимоги описують основні можливості «Project Mebius» і конкретні умови роботи застосунку. Вони охоплюють систему героїв, процедурну генерацію, бойову формацію, кампанію, башту, автоматизований бій, нагороди, розвиток персонажів, інтерфейс і збереження прогресу. Надалі ці вимоги використовуються для вибору засобів реалізації та проєктування структури ігрового застосунку.

1.4 Інформаційна модель предметної галузі

Інформаційна модель предметної галузі потрібна для того, щоб визначити, які основні дані використовуються в ігровому застосунку та як вони пов’язані між собою. Для «Project Mebius» така модель охоплює користувача, героїв, ворогів, ресурси, команду, бойові етапи, нагороди, завдання та збереження прогресу. Усі ці елементи не існують окремо, оскільки кожна дія користувача впливає на подальший стан гри.

Основою ігрового процесу є профіль користувача. У ньому зберігається загальний прогрес: кількість ресурсів, список отриманих героїв, поточна формація, пройдений етап кампанії, прогрес у башті, виконані завдання та налаштування. Сам користувач не створює обліковий запис через сервер, оскільки застосунок працює локально. Дані зберігаються у файлі, тому після повторного запуску гри користувач може продовжити проходження з попереднього стану.

Однією з головних сутностей інформаційної моделі є герой. Герой має ім’я, роль, фракцію, рівень, зірковість, набір характеристик, здібності та візуальні ознаки. У грі герой виступає не просто окремим об’єктом у колекції, а бойовою одиницею, яку можна додати до команди, покращити та використати під час проходження етапів. Через це дані героя мають бути пов’язані з бібліотекою персонажів, формацією, бойовою системою та системою розвитку.

Характеристики героя визначають його ефективність у бою. До них належать запас здоров’я, атака, захист, швидкість, магічна сила, лікування, шанс критичного удару та ухилення. На основі цих параметрів розраховується загальна сила героя, а також його поведінка під час автоматизованого бою. Наприклад, герой із високим запасом здоров’я та захистом краще підходить для передньої лінії, а персонажі з високою атакою або магічною силою доцільніші на задній лінії.

Фракції та ролі героїв також є важливими елементами моделі. Роль визначає призначення героя в команді: танк, воїн, лучник, маг, цілитель, убивця або підтримка. Фракція потрібна для поділу персонажів за групами та розрахунку

синергій. Якщо в команді є кілька героїв однієї фракції, система може надати додаткові бонуси. Таким чином, інформація про фракцію впливає не лише на опис героя, а й на бойові показники всієї команди.

Формація описує склад команди, яку користувач відправляє в бій. У «Project Mebius» команда складається максимум із п’яти героїв: дві позиції передньої лінії та три позиції задньої лінії. Таке обмеження потрібне для того, щоб користувач не просто додавав усіх доступних героїв, а обирав потрібний склад. Формація пов’язана з бібліотекою героїв, оскільки до неї можна додати лише тих персонажів, які вже є в колекції користувача.

Ворог є окремою сутністю, яка використовується під час бойових етапів. Для нього зберігаються назва, тип, рівень, характеристики, роль у бою та ознака звичайного, елітного або босового противника. Вороги генеруються залежно від режиму, етапу, складності та сили команди користувача. Завдяки цьому бойові ситуації можуть відрізнятися між собою, а складність поступово зростає.

Бойовий етап об’єднує команду користувача, групу ворогів, режим проходження та майбутню нагороду. У грі передбачено два основні напрями проходження: кампанія та башта. Кампанія використовується для послідовного просування користувача, а башта є додатковим режимом із поступовим підвищенням складності. Кожен етап має номер, складність, набір ворогів і результат, який визначається після завершення бою.

Бій у моделі є процесом взаємодії між героями користувача та ворогами. Під час бою враховуються характеристики учасників, швидкість, атаки, спеціальні здібності, критичні удари, ухилення, лікування та командна суперздібність. Після завершення бою система визначає перемогу або поразку. У разі перемоги користувач отримує нагороду, а прогрес кампанії або башти оновлюється.

Нагороди пов’язують бойову систему з розвитком користувача. Після проходження етапу гравець може отримати золото, кристали, сувої, досвід героїв, матеріали для тренування або есенцію для підвищення зірковості. Ці ресурси зберігаються в гаманці користувача та використовуються в інших частинах гри.

Наприклад, золото й матеріали потрібні для тренування, а сувої та кристали — для отримання нових героїв.

Окремим елементом інформаційної моделі є завдання. Вони використовуються для того, щоб спрямовувати користувача до основних дій гри: заклику героїв, проходження боїв, збору нагород, тренування персонажів та розвитку команди. Завдання мають стан виконання, цільове значення та нагороду. Після виконання умови користувач може отримати додаткові ресурси.

Система збереження об’єднує всі основні дані в одну структуру. До неї входять герої, ресурси, формація, прогрес кампанії та башти, завдання, налаштування й час останнього виходу з гри. Останній параметр потрібний для розрахунку AFK-нагород. Якщо користувач певний час не запускав гру, система може нарахувати ресурси за час відсутності, але з установленим обмеженням накопичення.

Основні сутності інформаційної моделі «Project Mebius» наведено в таблиці 1.4.

Таблиця 1.4

Основні сутності інформаційної моделі ігрового застосунку «Project Mebius»

Сутність	**Призначення**	**Основні дані**
Користувач	Зберігає загальний стан гри	Ресурси, герої, формація, прогрес, налаштування
Герой	Є основною колекційною та бойовою одиницею	Ім’я, роль, фракція, рівень, зірковість, характеристики, здібності
Характеристики	Визначають силу героя або ворога	Здоров’я, атака, захист, швидкість, магічна сила, лікування, критичний удар, ухилення
Фракція	Об’єднує героїв у групи та впливає на синергії	Назва фракції, бонуси, колір або стилістична ознака
Роль	Визначає призначення персонажа в бою	Танк, воїн, лучник, маг, цілитель, убивця, підтримка
Формація	Описує склад бойової команди	2 передні позиції, 3 задні позиції, список вибраних героїв
Ворог	Використовується як противник у бою	Назва, тип, рівень, характеристики, ознака елітного ворога або боса

Продовження таблиці 1.4

Основні сутності інформаційної моделі ігрового застосунку «Project Mebius»

Сутність	Призначення	Основні дані
Етап	Описує окремий рівень кампанії або башти	Номер, режим, складність, вороги, можлива нагорода
Бій	Визначає результат зіткнення команди з ворогами	Учасники, раунди, дії, результат, бойовий журнал
Нагорода	Передає користувачу ресурси після успішної дії	Золото, кристали, сувої, досвід, матеріали, есенція
Завдання	Фіксує додаткові цілі для користувача	Назва, умова, прогрес виконання, нагорода
Збереження	Відновлює стан гри після перезапуску	Дані користувача, час останнього входу, налаштування

На рисунку 1.5 подано узагальнену інформаційну модель ігрового застосунку «Project Mebius».

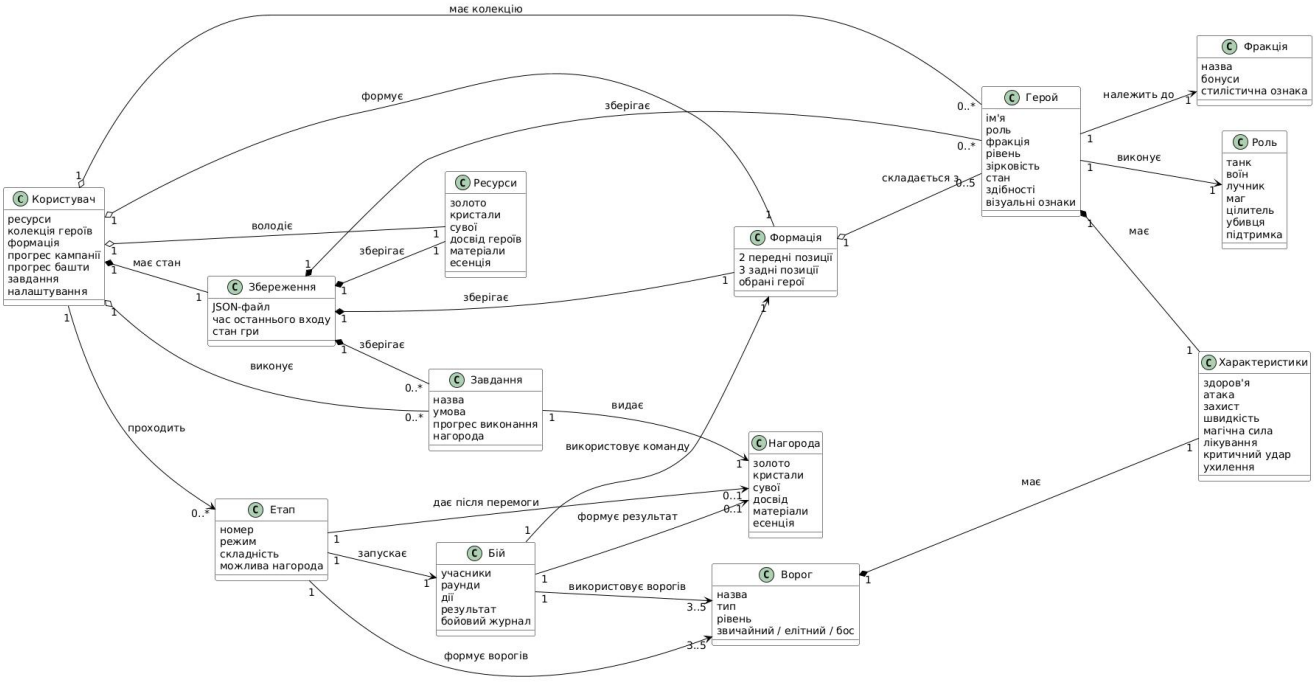


Рис. 1.5 Інформаційна модель ігрового застосунку «Project Mebius»

У цій моделі центральним елементом є користувач, оскільки саме з його прогресом пов’язані всі інші дані. Користувач має набір героїв і ресурсів, формує команду, проходить етапи, бере участь у боях, отримує нагороди та виконує

завдання. Герої пов’язані з характеристиками, ролями та фракціями, а формація визначає, які саме герої беруть участь у бою. Етапи кампанії та башти пов’язані з ворогами, боєм і нагородами.

Така інформаційна модель дозволяє зрозуміти внутрішню логіку застосунку ще до опису програмної реалізації. Вона показує, які дані потрібно зберігати, які сутності взаємодіють між собою та як формується основний ігровий цикл. Користувач отримує героя, додає його до команди, проходить етап, бере участь у бою, отримує нагороду, розвиває персонажів і продовжує проходження. Усі ці дії відображаються у збережених даних.

Отже, інформаційна модель «Project Mebius» складається з взаємопов’язаних сутностей, які описують героїв, ворогів, ресурси, формацію, бойові етапи, нагороди, завдання та прогрес користувача. Така структура є основою для подальшого вибору засобів реалізації, проєктування архітектури застосунку та опису програмних модулів у наступних розділах.

2 ЗАСОБИ ТА МЕТОДИ РЕАЛІЗАЦІЇ ІГРОВОГО ЗАСТОСУНКУ

2.1 Середовище розробки Unity

Для реалізації ігрового застосунку було обрано середовище розробки Unity. Це середовище є зручним для створення інтерактивних ігор, оскільки поєднує редактор сцени, роботу з графічними ресурсами, інтерфейсом користувача, аудіо, програмною логікою та налаштуваннями проєкту. У межах розробки Unity використовується як основна платформа, що забезпечує запуск гри, обробку дій користувача, відображення екранів і взаємодію між окремими програмними модулями.

Проєкт реалізовано у версії Unity 6000.4.2f1. Ця версія дає змогу працювати з 2D-графікою, створювати інтерфейсні елементи, підключати локальні ресурси та формувати збірку для персонального комп'ютера. Для гри в жанрі Fantasy RPG це важливо, оскільки застосунок містить декілька взаємопов'язаних частин: головне меню, ігровий хаб, бібліотеку героїв, екран команди, карту проходження, бойову сцену, систему нагород, завдання та налаштування [12].

Структура Unity-проєкту побудована за стандартним принципом. Основні матеріали розміщуються в каталозі Assets, де зберігаються сцени, скрипти, зображення, фони, спрайти, іконки, аудіофайли, шрифти та інші ресурси. Такий підхід дозволяє розділити програмну логіку й візуальні матеріали, що полегшує роботу з проєктом. Окремі папки використовуються для персонажів, ворогів, інтерфейсу, фонів, іконок та звукового супроводу.

Основна сцена виконує роль точки запуску. Після старту застосунку програмна логіка створює головний інтерфейс, налаштовує Canvas, завантажує необхідні ресурси та відкриває початковий екран. Значна частина інтерфейсу формується програмно, а не вручну в редакторі. Це зручно для такого типу гри, оскільки потрібно підтримувати багато екранів: меню, заклик героїв, бібліотеку, формацію, кампанію, башту, бій, завдання, налаштування та титри.

Для побудови інтерфейсу використовується система Unity UI. Основою є Canvas із базовою роздільною здатністю 1920×1080 пікселів. На ньому розміщуються панелі, кнопки, текстові блоки, іконки ресурсів, зображення героїв, бойовий журнал і повідомлення про результат дій. Такий формат дозволяє подати значну кількість інформації та водночас зберегти зрозумілу структуру екрана [13].

Окрему роль відіграє робота з графічними ресурсами. Оскільки застосунок виконано у 2D pixel fantasy стилістиці, у ньому використовуються фони, спрайти героїв і ворогів, іконки ресурсів, елементи інтерфейсу та декоративні зображення. Unity дає змогу завантажувати ці матеріали з локальних папок проєкту та використовувати їх під час створення різних екранів. Завдяки цьому інтерфейс, персонажі, карта та бойова сцена мають єдине візуальне спрямування.

Середовище Unity також зручне для перевірки роботи гри під час розробки. Застосунок можна запускати безпосередньо в редакторі й одразу тестувати основні сценарії: відкриття меню, генерацію героя, перегляд бібліотеки, формування команди, запуск бою, нарахування нагород і збереження даних. Це спрощує пошук помилок, оскільки зміни в коді або ресурсах можна перевіряти без створення окремої збірки після кожного редагування.

Програмна частина проєкту реалізована мовою C# у вигляді скриптів. Основний модуль відповідає за запуск гри, створення інтерфейсу, перехід між екранами, генерацію героїв і ворогів, роботу бойової системи, нагороди, завдання та збереження прогресу. Така структура підходить для локального ігрового застосунку, оскільки всі дані й ресурси знаходяться в межах одного Unity-проєкту та не потребують серверної частини.

Ще однією перевагою Unity є можливість створення Windows-збірки. Це дозволяє запускати гру не лише в редакторі, а й як окремий застосунок на персональному комп’ютері. У налаштуваннях проєкту передбачено роботу у Windows x64-середовищі, що відповідає обраній платформі запуску.

Отже, Unity було обрано як основне середовище розробки через його можливості для створення 2D-ігор, роботи з інтерфейсом, локальними ресурсами, C#-скриптами, тестуванням і підготовкою збірки для Windows. Використання

цього середовища дало змогу поєднати в одному проєкті меню, ігрові екрани, систему героїв, бойову логіку, нагороди, налаштування та збереження прогресу.

2.2 Мова програмування C#

Для реалізації програмної логіки ігрового застосунку використано мову програмування C#. Вона є основною мовою написання скриптів у Unity та добре підходить для створення ігор, у яких потрібно описувати поведінку об’єктів, обробляти дії користувача, зберігати дані та керувати взаємодією між різними частинами програми. У межах розробки C# використовується для реалізації ігрових моделей, генераторів, сервісів, бойової системи, інтерфейсу та збереження прогресу [15].

C# належить до мов із підтримкою об’єктно-орієнтованого програмування. Це зручно для гри жанру Fantasy RPG, оскільки в ній є багато сутностей зі своїми властивостями та поведінкою. Наприклад, герой має ім’я, роль, фракцію, рівень, зірковість, характеристики та здібності. Ворог також має власні параметри, рівень складності та тип. Окремо зберігаються ресурси користувача, нагороди, формація, завдання та налаштування. Завдяки класам ці дані можна описати структуровано, а потім використовувати в різних частинах програми.

У проєкті програмна логіка організована через набір класів і сервісів. Частина класів відповідає за зберігання даних, а інша частина — за виконання певних дій. Наприклад, окремі моделі описують героя, ворога, характеристики, нагороди, ресурси та збереження. Генератори використовуються для створення героїв, ворогів і нагород. Сервіси відповідають за формацію, фракційні синергії, завдання, кампанію, бій і збереження даних. Такий підхід дозволяє розділити відповідальність між різними частинами коду.

Однією з важливих переваг C# є строга типізація. Для ігрового застосунку це корисно, оскільки помилки в типах даних можна швидше виявити під час розробки. Наприклад, кількість ресурсів має зберігатися як числове значення, назва героя — як текст, а стан персонажа або тип екрана — як окрема категорія. У грі використовуються переліки для опису екранів, типів ресурсів, режимів

фільтрації та інших фіксованих значень. Це робить код зрозумілішим і зменшує кількість помилок, пов’язаних із неправильним написанням текстових значень.

C# також зручна для реалізації процедурної генерації. У розробленому застосунку генерація використовується для створення героїв, ворогів, нагород і бойових етапів. Під час створення героя програма визначає його фракцію, роль, зірковість, характеристики, ім’я, здібності та візуальні ознаки. Для ворогів формується рівень, тип, сила, кількість у групі та ознака звичайного, елітного або босового противника. Завдяки використанню методів і класів логіку генерації можна винести в окремі частини програми, не змішуючи її з інтерфейсом або бойовою системою.

Бойова система також реалізована засобами C#. Вона працює як автоматизована симуляція, у якій беруть участь герої користувача та згенеровані вороги. Під час бою враховуються характеристики учасників, швидкість, атаки, критичні удари, ухилення, лікування, енергія та спеціальні здібності. Програма визначає порядок дій, обчислює результат кожного ходу та формує підсумок бою. Така логіка потребує чітких розрахунків, тому використання C# є зручним для опису правил бою та перевірки результатів.

Окремо C# використовується для роботи з інтерфейсом користувача. У грі є багато екранів, між якими користувач переходить під час проходження: меню, хаб, бібліотека героїв, команда, кампанія, башта, бій, завдання та налаштування. За допомогою скриптів створюються кнопки, текстові блоки, панелі, списки героїв, повідомлення, бойовий журнал і вікна результатів. Під час натискання кнопки відповідний метод змінює стан гри або відкриває інший екран. Завдяки цьому інтерфейс не є статичним, а постійно оновлюється залежно від дій користувача.

Мова C# також застосовується для збереження й завантаження прогресу. У застосунку потрібно зберігати список героїв, ресурси, формацію, прогрес кампанії, башти, завдання та налаштування. Для цього дані об’єднуються у структуру збереження, після чого записуються у локальний JSON-файл. Під час наступного запуску програма читає ці дані та відновлює попередній стан гри. Для колекційної

RPG це важливо, оскільки користувач поступово накопичує героїв і ресурси, тому втрата прогресу зробила б гру некоректною.

Ще однією перевагою використання C# є можливість поступово розширювати програмну логіку. У разі потреби до гри можна додавати нові ролі героїв, фракції, типи ворогів, нагороди, завдання або режими проходження. Якщо структура класів побудована правильно, такі зміни не потребують повного переписування всієї програми. Наприклад, новий тип нагороди або нову роль героя можна додати до відповідних моделей і сервісів, після чого використати їх у генерації, інтерфейсі та бойовій системі.

У межах розробки C# взаємодіє з компонентами Unity. Скрипти можуть створювати об’єкти інтерфейсу, змінювати зображення, оновлювати текст, реагувати на натискання кнопок, запускати анімації та працювати з локальними ресурсами. Саме завдяки цій взаємодії програмна логіка поєднується з візуальною частиною гри. Наприклад, після генерації героя його дані не тільки зберігаються в колекції, а й відображаються в бібліотеці у вигляді картки з характеристиками, фракцією, роллю та зображенням.

Отже, C# було обрано як основну мову програмування через її підтримку об’єктно-орієнтованого підходу, строгу типізацію, зручну роботу з класами, методами, колекціями та інтеграцію з Unity. За допомогою цієї мови реалізовано основні системи гри: моделі даних, процедурну генерацію, формацію, фракційні бонуси, бойову логіку, інтерфейс, нагороди, завдання та збереження прогресу.

2.3 Засоби створення інтерфейсу та візуального оформлення

Інтерфейс користувача є важливою частиною ігрового застосунку, оскільки через нього користувач переходить між основними екранами, переглядає героїв, формує команду, запускає бої, отримує нагороди та змінює налаштування. Для гри жанру Fantasy RPG інтерфейс має не тільки виконувати службову функцію, а й підтримувати загальну атмосферу ігрового світу. Тому під час розробки враховувалася як зручність користування, так і відповідність візуального оформлення fantasy/pixel-art стилістиці.

Для побудови інтерфейсу використано засоби Unity UI. Основою інтерфейсної частини є Canvas, на якому розміщуються кнопки, панелі, текстові блоки, іконки ресурсів, зображення героїв, елементи карти, бойовий журнал та інші компоненти. Такий підхід дозволяє створювати окремі екрани гри без використання сторонніх бібліотек. У застосунку передбачено головне меню, хаб, екран заклику героїв, бібліотеку, формацію команди, карту кампанії, башту, бойовий екран, завдання, налаштування та титри [13].

Базова роздільна здатність інтерфейсу становить 1920×1080 пікселів. Це дає можливість розмістити на екрані достатню кількість інформації, не перевантажуючи користувача. Наприклад, у бібліотеці героїв потрібно показати список персонажів, їхні основні характеристики, роль, фракцію, зірковість і стан. На бойовому екрані потрібно відобразити команди, ворогів, повідомлення про дії, результат бою та отримані нагороди. Тому інтерфейс має бути не тільки декоративним, а й достатньо функціональним.

Для текстових елементів використовується TextMeshPro. Цей інструмент забезпечує кращу якість відображення тексту порівняно зі стандартними текстовими компонентами Unity. У грі є багато написів українською мовою: назви кнопок, описи героїв, бойові повідомлення, дані про ресурси, назви фракцій, ролі, завдання та налаштування. Через це читабельність тексту має велике значення. Якщо текст буде дрібним або нечітким, користувачу буде складно сприймати характеристики героїв і результати бою [14].

Усі основні елементи інтерфейсу мають єдину логіку побудови. Кожен екран складається з фону, заголовка, основної інформаційної області та кнопок навігації. Наприклад, у головному меню користувач бачить назву гри та основні кнопки запуску. У хабі розміщуються переходи до ключових розділів. У бібліотеці основну частину займають картки героїв, а на екрані формації — позиції передньої та задньої лінії. Така повторювана структура допомагає користувачу швидше орієнтуватися в застосунку.

Візуальне оформлення побудоване навколо 2D pixel fantasy стилістики. У проєкті використовуються фони, спрайти персонажів, зображення ворогів, іконки

ресурсів, декоративні панелі та елементи інтерфейсу. Такі ресурси зберігаються локально в папках проєкту та завантажуються під час роботи гри. Завдяки цьому застосунок не потребує підключення до зовнішніх сервісів для відображення основних графічних матеріалів.

Фони використовуються для створення атмосфери різних частин гри. Окремі зображення можуть застосовуватися для меню, хабу, кампанії, бойового режиму або інших екранів. Це дозволяє візуально розділити основні стани застосунку. Наприклад, бойовий екран має відрізнятися від головного меню або бібліотеки героїв, оскільки виконує іншу функцію і повинен привертати увагу до персонажів та ворогів.

Спрайти героїв і ворогів використовуються для візуального представлення бойових одиниць. Для колекційної RPG це важливо, оскільки герой має сприйматися не тільки як набір характеристик, а і як окрема ігрова одиниця. У застосунку зовнішній вигляд персонажів пов’язаний із процедурною генерацією: система може обирати різні візуальні варіанти залежно від параметрів героя. Це робить колекцію різноманітнішою й підтримує інтерес до отримання нових персонажів.

Іконки ресурсів застосовуються для швидкого сприйняття ігрової економіки. Користувач повинен легко розрізняти золото, кристали, сувої, матеріали, досвід героїв та інші ресурси. Якщо всі ресурси відображати лише текстом, інтерфейс стає важчим для сприйняття. Іконки дозволяють швидше зрозуміти, яку саме нагороду отримано або який ресурс потрібен для певної дії.

Окрему увагу приділено кнопкам і навігації. У грі користувач часто переходить між екранами, тому кнопки мають бути зрозумілими та розміщуватися в логічних місцях. Наприклад, кнопки повернення до хабу, запуску бою, заклику героя або підтвердження дії повинні бути помітними. Для покращення сприйняття в інтерфейсі використовуються прості візуальні ефекти натискання, які дають користувачу зрозуміти, що дія була виконана.

У бойовому екрані інтерфейс виконує особливо важливу роль. Він має показувати не лише персонажів і ворогів, а й перебіг бою. Для цього

використовується бойовий журнал, у якому фіксуються атаки, здібності, критичні удари, лікування, поразка ворогів і результат битви. Такий журнал допомагає користувачу зрозуміти, чому бій завершився перемогою або поразкою, навіть якщо сама битва проходить автоматично.

У налаштуваннях користувач може змінювати параметри, які впливають на сприйняття гри. До них належать гучність музики та звукових ефектів, швидкість бою, анімації, ефекти трясіння екрана або показ бойових чисел. Наявність таких налаштувань робить застосунок зручнішим, оскільки різні користувачі можуть мати різні вимоги до швидкості й візуальної насиченості гри.

Важливо, що інтерфейс у застосунку не існує окремо від програмної логіки. Коли користувач отримує героя, дані одразу мають відобразитися в бібліотеці. Якщо герой додається до формації, зміни повинні бути помітні на екрані команди. Після перемоги в бою оновлюється кількість ресурсів і прогрес проходження. Саме тому інтерфейс постійно взаємодіє з моделями даних, генераторами, бойовою системою та збереженням.

Отже, для створення інтерфейсу та візуального оформлення використано Unity UI, TextMeshPro, локальні графічні ресурси, спрайти, фони, іконки та елементи pixel-art стилістики. Таке поєднання дозволило побудувати україномовний інтерфейс, який підтримує основні ігрові дії: заклик героїв, перегляд бібліотеки, формування команди, проходження боїв, отримання нагород, виконання завдань і зміну налаштувань.

2.4 Методи процедурної генерації в ігрових застосунках

Процедурна генерація в іграх використовується тоді, коли частину контенту потрібно створювати автоматично, а не прописувати кожен елемент вручну. Це можуть бути персонажі, вороги, нагороди, рівні, характеристики, зовнішні ознаки або окремі події. Для ігор жанру Fantasy RPG такий підхід є особливо корисним, оскільки в них зазвичай потрібна велика кількість героїв, різні групи противників і поступове ускладнення проходження [4; 5; 6].

У розроблюваному застосунку процедурна генерація використовується в кількох основних частинах гри. Вона застосовується під час створення героїв, ворогів, нагород, етапів кампанії та башти. Також генерація впливає на візуальні ознаки персонажів, завдяки чому герої можуть відрізнятися не тільки характеристиками, а й зовнішнім виглядом. Це дозволяє зробити гру різноманітнішою без ручного створення кожного окремого варіанта.

Генерація не повинна бути повністю випадковою. Якщо всі параметри обирати без обмежень, гра може створити занадто сильного героя на початку або, навпаки, надто складних ворогів для поточного етапу. Тому в таких системах випадковість поєднується з правилами. Спочатку задаються допустимі межі, а вже в них система обирає конкретні значення. Саме такий підхід використовується в грі для героїв, ворогів і нагород.

Під час створення героя система формує кілька груп даних. До першої групи належать загальні відомості: ім’я, стать, раса, фракція, роль і зірковість. До другої групи входять бойові характеристики: здоров’я, атака, захист, швидкість, магічна сила, лікування, критичний удар і ухилення. Окремо визначаються здібності, пасивна особливість, ультимативна дія та візуальні параметри. У результаті герой має повний набір даних, який можна використовувати в бібліотеці, формації, бойовій системі та збереженні.

У грі передбачено кілька ролей героїв: танк, воїн, лучник, маг, цілитель, убивця та підтримка. Роль впливає на те, як персонаж поводиться в бою. Наприклад, танк краще підходить для передньої лінії, оскільки має витримувати більше пошкоджень. Цілитель або персонаж підтримки частіше корисний на задній лінії, де він може допомагати команді довше залишатися в бою. Завдяки цьому герой не є просто випадковим набором чисел, а має певне призначення.

Фракції також впливають на ігрову логіку. Герой може належати до однієї з кількох фракцій, і це використовується під час формування команди. Якщо в складі є кілька персонажів однієї фракції, команда може отримати додаткові бонуси. Тому під час генерації фракція стає не лише описовою характеристикою, а й частиною стратегічної системи.

Окремо реалізовано випадкове отримання героїв через систему заклику. У ній герої різної зірковості мають різні шанси появи. Звичайні персонажі трапляються частіше, а найрідкісніші — значно рідше. Це потрібно для того, щоб зберегти цінність сильних героїв і водночас дати користувачу змогу регулярно поповнювати колекцію.

Таблиця 2.1

Ймовірність отримання героя залежно від зірковості

Зірковість героя	**Ймовірність отримання**
1 зірка	50 %
2 зірки	25 %
3 зірки	13 %
4 зірки	7 %
5 зірок	3 %
6 зірок	1,5 %
7 зірок	0,5 %

Такий розподіл робить систему заклику більш керованою. Користувач найчастіше отримує базових героїв, але має шанс відкрити рідкісного персонажа. При цьому цінність героя визначається не тільки зірковістю. На практиці важливими залишаються роль, фракція, характеристики та місце в команді.

Генерація ворогів працює за іншим принципом. Вороги створюються не для постійного зберігання в колекції, а для конкретного бойового етапу. Система враховує номер етапу, режим проходження, складність і приблизну силу команди користувача. На одному етапі формується від трьох до п’яти противників. Частина етапів може містити елітних ворогів або боса. Бос з’являється на кожному десятому етапі, що дозволяє періодично підвищувати складність проходження.

Під час генерації ворога задаються його назва, тип, рівень, характеристики та ознака звичайного, елітного або босового противника. Чим далі просувається користувач, тим сильнішими стають вороги. Це потрібно для того, щоб складність

не залишалася однаковою протягом усієї гри. Якщо користувач розвиває героїв, противники також мають поступово посилюватися.

Процедурна генерація використовується і для нагород. Після перемоги користувач отримує ресурси, які потрібні для подальшого розвитку. Це може бути золото, кристали, сувої, досвід героїв, матеріали для тренування або есенція для підвищення зірковості. Розмір нагороди залежить від режиму, складності етапу та типу ворогів. Наприклад, після босового етапу нагорода має бути більшою, ніж після звичайного бою.

Нагороди пов’язують бойову систему з прогресом користувача. Якщо після бою гравець не отримує корисних ресурсів, проходження швидко втрачає сенс. Тому генерація нагород повинна підтримувати основний цикл гри: бій, перемога, отримання ресурсів, розвиток героїв, проходження складнішого етапу.

Ще одним напрямом є генерація візуальних ознак. Для колекційної RPG важливо, щоб герої відрізнялися не лише параметрами, а й зовнішністю. У грі для цього використовуються візуальні варіанти, які можуть впливати на вибір спрайта, кольору або окремих деталей образу. Це робить персонажів більш різноманітними й допомагає користувачу сприймати їх як окремі ігрові одиниці.

Основні напрями використання процедурної генерації наведено в таблиці 2.2.

Таблиця 2.2

Напрями використання процедурної генерації в ігровому застосунку

Напрям генерації	**Що формується**	**Для чого використовується**
Герої	Ім’я, роль, фракція, зірковість, характеристики, здібності, зовнішні ознаки	Для поповнення колекції користувача
Вороги	Кількість, тип, рівень, характеристики, ознака елітного ворога або боса	Для створення бойових етапів
Етапи	Номер, режим, складність, група ворогів, можлива нагорода	Для проходження кампанії та башти
Нагороди	Золото, кристали, сувої, досвід, матеріали, есенція	Для розвитку героїв і команди

Продовження таблиці 2.2

Напрями використання процедурної генерації в ігровому застосунку

Напрям генерації	Що формується	Для чого використовується
Візуальні ознаки	Спрайт, колір, окремі елементи вигляду	Для різноманітності персонажів

Для правильної роботи процедурної генерації важливо, щоб усі створені об’єкти мали однакову структуру даних. Наприклад, кожен герой повинен мати характеристики, роль, фракцію та рівень, незалежно від того, які саме значення були згенеровані. Це потрібно для того, щоб бібліотека могла показати героя, бойова система могла використати його в бою, а система збереження могла записати його у файл.

Те саме стосується ворогів і нагород. Ворог повинен мати дані, потрібні для бою, а нагорода — ресурси, які можна додати до гаманця користувача. Якщо структура буде різною для кожного випадку, інші частини програми не зможуть коректно працювати з такими даними. Тому генерація тісно пов’язана з інформаційною моделлю застосунку.

У грі також передбачені обмеження, які не дозволяють генерації виходити за межі логіки балансу. Команда складається максимум із п’яти героїв, бій триває не більше 160 раундів, зірковість героя обмежена сьомим рівнем, максимальний рівень не перевищує 100, а AFK-нагороди накопичуються не більше ніж за 12 годин. Такі обмеження потрібні для того, щоб гра залишалася стабільною та передбачуваною.

Отже, процедурна генерація в ігровому застосунку використовується для створення героїв, ворогів, етапів, нагород і візуальних ознак. Вона дає змогу урізноманітнити проходження, підтримати колекціонування персонажів і зменшити потребу в ручному створенні кожного елемента. У поєднанні з обмеженнями та збереженням даних така система стає однією з основних частин ігрової логіки.

2.5 Засоби збереження даних та організації ресурсів проєкту

Для ігрового застосунку з елементами колекціонування важливо не лише створити героїв, провести бій або нарахувати нагороду, а й зберегти результат цих дій. Користувач поступово накопичує персонажів, ресурси, проходить етапи кампанії та башти, змінює формацію і налаштування. Якщо ці дані не будуть збережені, основна логіка прогресу втрачає зміст. Тому в розробленій грі передбачено локальне збереження стану користувача.

У застосунку не використовується серверна база даних або онлайн-авторизація. Усі дані зберігаються локально на пристрої користувача. Такий підхід підходить для одиночної гри, у якій основний прогрес пов’язаний із локальним проходженням, а не з мережевою взаємодією. Збереження виконується у форматі JSON, оскільки цей формат є зручним для запису структурованих даних і подальшого відновлення стану гри.

Файл збереження розміщується в локальній директорії, доступній через Application.persistentDataPath. Це стандартний шлях Unity для зберігання даних, які мають залишатися доступними між запусками застосунку. У межах гри використовується файл project_mebius_save.json, у якому зберігаються основні дані користувача. Під час запуску програма перевіряє наявність цього файлу. Якщо збереження існує, дані завантажуються. Якщо файл відсутній, створюється початковий стан гри [16; 17].

Структура збереження охоплює дані, необхідні для продовження проходження. До них належать ресурси користувача, список отриманих героїв, поточна формація, прогрес кампанії, прогрес башти, завдання, налаштування та час останнього виходу з гри. Останній параметр потрібний для розрахунку AFK-нагород. Якщо користувач певний час не запускав гру, система може визначити тривалість відсутності та нарахувати відповідні ресурси з урахуванням обмеження до 12 годин.

Основні дані, які зберігаються в застосунку, наведено в таблиці 2.3.

Таблиця 2.3

Основні дані, що зберігаються в ігровому застосунку

Група даних	Що зберігається	Призначення
Ресурси	золото, кристали, сувої, досвід героїв, матеріали, есенція	Використовуються для заклику, тренування та покращення героїв
Герої	список отриманих персонажів, рівень, зірковість, характеристики, роль, фракція, стан	Дають змогу відновити колекцію користувача після перезапуску гри
Формація	позиції героїв у передній і задній лінії	Визначає склад команди для наступного бою
Кампанія	поточний пройдений етап	Дозволяє продовжити основне проходження
Башта	поточний рівень режиму башти	Зберігає прогрес у додатковому режимі
Завдання	стан виконання, прогрес і отримання нагород	Дає змогу продовжити виконання ігрових цілей
Налаштування	гучність, швидкість бою, анімації, візуальні ефекти	Відновлює параметри, обрані користувачем
Час останнього входу	дата і час завершення попередньої сесії	Використовується для розрахунку AFK-нагород

Збереження даних пов’язане з більшістю ігрових систем. Наприклад, після заклику нового героя він додається до колекції та має бути записаний у файл збереження. Після перемоги в бою змінюється кількість ресурсів і прогрес проходження. Якщо користувач змінив склад команди, формація також повинна бути збережена. Такий зв’язок дозволяє уникнути ситуації, коли окрема дія виконана в інтерфейсі, але не відображається після повторного запуску гри.

Для коректної роботи збереження важливо, щоб дані мали зрозумілу структуру. Тому в програмній логіці використовуються окремі класи для ресурсів, героїв, формації, завдань, налаштувань і загального стану гри. Такий підхід спрощує запис у JSON-файл і подальше відновлення. Після завантаження система отримує не окремий набір випадкових значень, а повний стан гри, з яким можуть працювати інші модулі.

Окреме значення має організація ресурсів проєкту. У грі використовується велика кількість локальних матеріалів: фони, спрайти героїв, зображення ворогів, іконки, елементи інтерфейсу, аудіофайли та шрифти. Вони розміщені в структурі Unity-проєкту та завантажуються під час роботи застосунку. Завдяки цьому гра не потребує підключення до зовнішніх сервісів для відображення основних графічних матеріалів.

Основні групи ресурсів наведено в таблиці 2.4.

Таблиця 2.4

Організація основних ресурсів ігрового проєкту

Група ресурсів	**Приклади вмісту**	**Використання**
Фони	зображення для меню, хабу, кампанії, бою та інших екранів	Формують візуальну атмосферу гри
Спрайти героїв	зображення або аркуші анімацій персонажів	Використовуються в бібліотеці, формації та бою
Спрайти ворогів	зображення противників і босів	Відображають ворогів у бойових етапах
Іконки	ресурси, ролі, нагороди, кнопки, службові позначення	Допомагають швидко розпізнавати елементи інтерфейсу
UI-елементи	панелі, кнопки, декоративні рамки	Використовуються для побудови екранів гри
Аудіо	музика, звукові ефекти, сигнали інтерфейсу	Підтримують атмосферу та реакцію на дії користувача
Шрифти	декоративні та інтерфейсні шрифти	Використовуються для назв, кнопок і текстових блоків

Усі ці ресурси мають бути впорядковані, оскільки від цього залежить зручність подальшої роботи з проєктом. Якщо зображення, аудіо та інтерфейсні елементи зберігати без структури, їх буде складно використовувати в коді та підтримувати під час розширення гри. Тому ресурси поділено за призначенням: окремо фони, окремо персонажі, вороги, іконки, інтерфейс, аудіо та шрифти.

Під час роботи застосунку програмна логіка звертається до цих ресурсів для створення екранів і відображення ігрових об’єктів. Наприклад, у меню

використовується один набір фонових зображень, у бібліотеці — спрайти героїв і елементи карток, у бою — зображення героїв, ворогів, бойовий журнал та іконки нагород. Така організація дозволяє використовувати одні й ті самі ресурси в різних частинах гри без дублювання.

Важливо, що графічні ресурси пов’язані з процедурною генерацією. Під час створення героя система може призначити йому певний візуальний варіант, який потім використовується в бібліотеці та бойовому екрані. Це дає можливість поєднати дані персонажа з його зовнішнім виглядом. Для ворогів також використовуються окремі зображення, що дозволяє розрізняти звичайних противників, елітних ворогів і босів.

Аудіоресурси застосовуються для покращення сприйняття гри. Звукові ефекти можуть супроводжувати натискання кнопок, отримання нагород, запуск бою або інші дії. Музика використовується для створення загальної атмосфери. При цьому в налаштуваннях користувач може змінювати гучність, що робить гру зручнішою для різних умов використання.

Організація ресурсів також важлива для підготовки збірки. Оскільки всі необхідні файли розміщені в межах Unity-проєкту, під час створення Windows-версії вони можуть бути включені до складу застосунку. Це дозволяє запускати гру як окрему програму без додаткового завантаження матеріалів із мережі. Для локального ігрового застосунку такий підхід є простим і надійним.

Отже, збереження даних і організація ресурсів є важливими частинами реалізації. Локальне JSON-збереження дозволяє відновлювати прогрес користувача після перезапуску гри, а впорядкована структура ресурсів забезпечує роботу інтерфейсу, персонажів, ворогів, фонів, іконок, аудіо та шрифтів. Разом ці засоби підтримують цілісність гри й дають змогу поєднати програмну логіку з візуальним та звуковим оформленням.

2.6 Допоміжні засоби підготовки, перевірки та збирання проєкту

Окрім основного середовища розробки, мови програмування, інтерфейсу, процедурної генерації та збереження даних, у проєкті використано допоміжні

засоби. Вони потрібні для підготовки ресурсів, перевірки працездатності основних механік і створення збірки гри. Такі інструменти не є окремими режимами для користувача, але вони важливі для самого процесу розробки.

У Unity-проєктах доводиться працювати не тільки з кодом, а й з великою кількістю ресурсів: зображеннями, спрайтами, фонами, іконками, аудіофайлами та шрифтами. Якщо такі матеріали мають неправильні параметри імпорту, вони можуть некоректно відображатися в грі або погіршувати якість pixel-art стилю. Тому для підготовки частини графічних матеріалів використано редакторський скрипт ProjectMebiusAssetPreparation.cs [18].

Цей скрипт призначений для налаштування окремих параметрів зображень, які використовуються в грі. Для pixel-art ресурсів важливо вимкнути згладжування, не використовувати mipmaps і зберігати чіткість пікселів. Також важливо, щоб зображення, які надалі можуть оброблятися програмно, були доступними для читання в Unity. Автоматизована підготовка ресурсів зменшує кількість ручних дій і ризик того, що частина спрайтів буде налаштована інакше, ніж інші.

Окремим допоміжним засобом є скрипт збирання проєкту ProjectMebiusBuild.cs. Його призначення полягає у підготовці Windows-збірки гри. У межах такого процесу Unity створює виконуваний файл застосунку та необхідні службові файли для запуску на персональному комп’ютері. Для розроблюваного застосунку передбачено створення Windows x64-збірки, що відповідає обраній платформі запуску [19].

Наявність окремого build-скрипта є зручною, оскільки процес збирання можна виконувати не лише вручну через меню Unity Editor, а й у більш упорядкованому вигляді. Це корисно тоді, коли перед збиранням потрібно перевірити ресурси або дотриматися однакових параметрів збірки. У результаті розробник отримує готову версію гри, яку можна запустити окремо від редактора Unity.

Для перевірки основних механік використовується редакторський smoke-тест ProjectMebiusSmokeTest.cs. Smoke-тест не замінює повне тестування гри, але

дає змогу швидко перевірити, чи працюють ключові частини застосунку після змін у коді або ресурсах. Такий підхід є корисним, оскільки гра складається з багатьох пов’язаних систем: генерації героїв, формації, ворогів, бою, нагород, завдань і збереження.

У межах smoke-тесту перевіряються базові сценарії, без яких застосунок не може працювати коректно. Наприклад, система повинна створювати героїв із потрібними параметрами, автоматично формувати команду, генерувати ворогів, запускати бій, формувати журнал подій, нараховувати нагороди та виконувати збереження з подальшим відновленням даних. Якщо одна з таких перевірок завершується помилкою, це означає, що відповідний модуль потрібно переглянути.

Основні допоміжні засоби, використані в проєкті, наведено в таблиці 2.5.

Таблиця 2.5

Допоміжні засоби підготовки, перевірки та збирання проєкту

Засіб	Призначення	Практичне використання
ProjectMebius AssetPreparation.cs	Підготовка графічних ресурсів	Налаштування параметрів імпорту pixel-art зображень і спрайтів
ProjectMebiusBuild.cs	Створення збірки гри	Підготовка Windows x64-версії застосунку
ProjectMebius SmokeTest.cs	Швидка перевірка основних механік	Перевірка генерації, формації, бою, нагород, завдань і збереження
Unity Editor	Основне середовище роботи з проєктом	Запуск гри, перевірка сцен, ресурсів, інтерфейсу та скриптів
Локальні ресурси Unity	Матеріали для візуального й звукового оформлення	Завантаження фонів, спрайтів, іконок, аудіо та шрифтів під час роботи гри

Допоміжні інструменти також пов’язані з вимогами до стабільності проєкту. Наприклад, якщо після зміни генерації героя перестає працювати бойова система, це можна швидше помітити через тестовий сценарій. Якщо неправильно

підготовлений спрайт не відображається в інтерфейсі, його параметри можна виправити ще на етапі підготовки ресурсів. Якщо збірка не створюється, це показує проблему ще до демонстрації готової версії гри.

Варто зазначити, що такі засоби не змінюють ігровий процес для користувача напряму. Користувач не бачить smoke-тест або скрипт підготовки ресурсів у самій грі. Проте ці частини впливають на якість розробки, оскільки допомагають перевіряти, чи основні системи працюють після внесення змін. Для проєкту з процедурною генерацією це особливо важливо, адже помилка в одному модулі може вплинути одразу на кілька інших частин гри.

Отже, допоміжні засоби підготовки, перевірки та збирання проєкту забезпечують більш упорядковану роботу з Unity-застосунком. Скрипт підготовки ресурсів допомагає підтримувати єдині параметри графічних матеріалів, build-скрипт використовується для створення Windows-версії, а smoke-тест дозволяє швидко перевірити основні механіки гри. Разом ці засоби доповнюють основну програмну реалізацію та спрощують подальшу підтримку і розвиток застосунку.

3 ПРОЄКТУВАННЯ, РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

3.1 Загальна архітектура ігрового застосунку

Архітектура ігрового застосунку побудована так, щоб усі основні частини гри працювали в межах одного Unity-проєкту. Застосунок не використовує серверну частину або зовнішню базу даних, тому генерація ігрового контенту, збереження прогресу, робота інтерфейсу, обробка боїв і керування ресурсами виконуються локально. Такий підхід є зручним для одиночної гри, де користувач проходить кампанію, збирає героїв і розвиває команду без потреби в мережевому підключенні.

Основна програмна логіка зосереджена у файлі ProjectMebiusMVP.cs. У ньому описано моделі даних, генератори, сервіси, бойову систему, систему збереження та створення інтерфейсу. Хоча файл є великим за обсягом, внутрішньо його можна поділити на кілька логічних частин. Одні класи відповідають за зберігання даних, інші - за створення героїв і ворогів, ще інші - за проведення бою, нарахування нагород, виконання завдань і відновлення прогресу після запуску гри.

Після запуску застосунку Unity відкриває основну сцену, а програмна логіка створює ігровий інтерфейс. Для цього формується Canvas із базовою роздільною здатністю 1920×1080, підключаються текстові елементи, кнопки, панелі, фони, спрайти та інші ресурси. Далі користувач потрапляє до головного меню або до основного хабу, з якого можна перейти до заклику героїв, бібліотеки, формації, кампанії, башти, бою, завдань і налаштувань.

Архітектуру застосунку можна умовно поділити на чотири рівні: рівень даних, рівень сервісів, рівень керування грою та рівень інтерфейсу. Рівень даних містить моделі героїв, ворогів, характеристик, ресурсів, нагород, формації, завдань і збереження. Рівень сервісів виконує основну обробку: генерує героїв і ворогів, формує нагороди, керує кампанією, перевіряє формацію, розраховує

синергії, проводить бій і зберігає дані. Рівень керування грою поєднує ці частини між собою, а рівень інтерфейсу відображає поточний стан гри користувачу.

На рисунку 3.1 наведено загальну архітектуру ігрового застосунку.

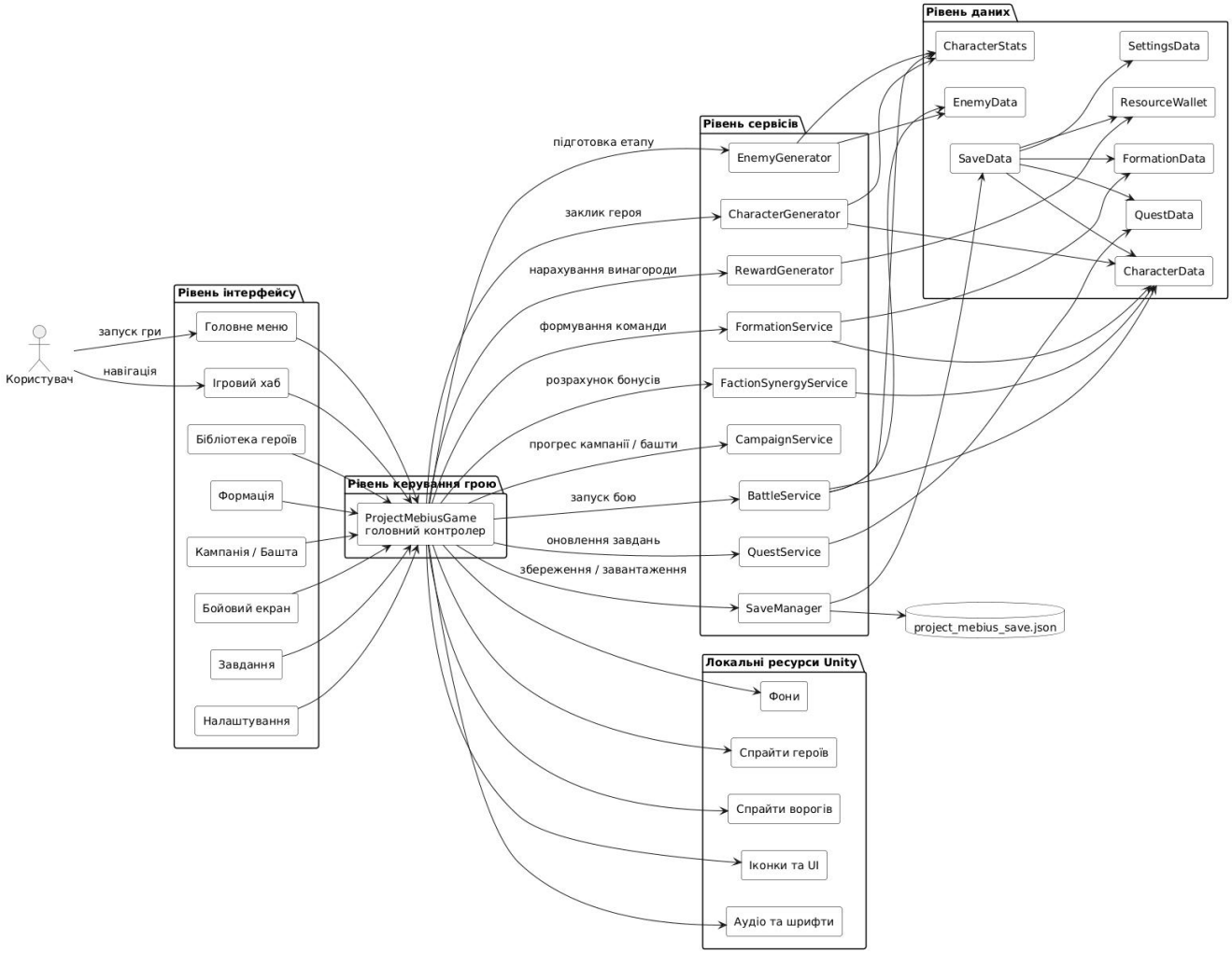


Рис. 3.1 Загальна архітектура ігрового застосунку «Project Mebius»

Центральним елементом архітектури є головний контролер гри. Він відповідає за запуск застосунку, створення початкового стану, перехід між екранами, оновлення інтерфейсу та виклик потрібних сервісів. Наприклад, коли користувач натискає кнопку заклику героя, контролер звертається до генератора персонажів, додає нового героя до збережених даних, оновлює ресурси та відображає результат на екрані. Коли користувач запускає бій, контролер передає формацію та згенерованих ворогів у бойовий сервіс, після чого отримує результат і нараховує нагороду.

Рівень даних є основою для всіх інших частин. Для героя зберігаються ім’я, роль, фракція, рівень, зірковість, характеристики, здібності, стан і візуальні ознаки. Для ворога зберігаються назва, тип, рівень, характеристики та ознака звичайного, елітного або босового противника. Окремо описуються ресурси користувача, формація, завдання, налаштування та повний стан збереження. Завдяки цьому різні системи працюють з однаковими даними і не дублюють їх.

Сервіси відповідають за основні правила гри. CharacterGenerator створює нових героїв за заданими параметрами та ймовірностями. EnemyGenerator формує групи ворогів для кампанії або башти. RewardGenerator визначає нагороди після перемоги. CampaignService працює з етапами проходження. FormationService перевіряє склад команди й не дозволяє додавати до формації некоректних персонажів. FactionSynergyService розраховує бонуси від фракцій. BattleService проводить автоматизований бій. QuestService оновлює завдання, а SaveManager відповідає за запис і читання JSON-файлу.

Основні модулі архітектури наведено в таблиці 3.1.

Таблиця 3.1

Основні програмні модулі ігрового застосунку

Модуль	**Призначення**	**Приклад використання**
ProjectMebiusGame	Керує запуском гри, екранами та основним станом застосунку	Перехід між меню, хабом, бібліотекою, боєм і налаштуваннями
CharacterGenerator	Створює героїв із випадковими, але обмеженими параметрами	Заклик героя через систему summon
EnemyGenerator	Формує ворогів для бойових етапів	Створення групи противників для кампанії або башти
RewardGenerator	Розраховує нагороди після проходження	Нарахування золота, досвіду, матеріалів або есенції
FormationService	Перевіряє та формує бойову команду	Розміщення 2 героїв спереду та 3 героїв позаду
FactionSynergyService	Визначає бонуси за поєднання героїв однієї фракції	Додавання бонусу до характеристик команди
BattleService	Виконує автоматизовану симуляцію бою	Розрахунок атак, лікування, критичних ударів і результату

Продовження таблиці 3.1

Основні програмні модулі ігрового застосунку

Модуль	Призначення	Приклад використання
QuestService	Відстежує виконання завдань	Оновлення прогресу після бою або заклику героя
SaveManager	Зберігає та завантажує прогрес користувача	Запис і читання локального JSON-файлу

Інтерфейсний рівень працює разом із сервісами, але не виконує самостійних розрахунків бойової логіки або генерації. Його завдання - показати користувачу поточний стан гри та передати дію до відповідного модуля. Наприклад, бібліотека відображає список героїв, але не створює їх сама. Екран бою показує результат симуляції, але сам розрахунок виконується в бойовому сервісі. Такий поділ робить структуру зрозумілішою і зменшує кількість дублювання в коді.

Система збереження пов’язана майже з усіма частинами архітектури. Після отримання героя, зміни формації, проходження бою, виконання завдання або зміни налаштувань стан гри має бути збережений. Для цього використовується локальний JSON-файл, у якому містяться герої, ресурси, прогрес кампанії, прогрес башти, завдання, формація та налаштування. Під час наступного запуску застосунок читає ці дані й відновлює попередній стан.

Важливою особливістю архітектури є те, що основні ігрові системи пов’язані через спільний стан гри. Наприклад, результат бою впливає на ресурси, ресурси використовуються для тренування або заклику, новий герой потрапляє до бібліотеки, з бібліотеки він може бути доданий до формації, а формація використовується в наступному бою. Завдяки цьому застосунок працює як єдина система, а не як набір незалежних екранів.

Отже, загальна архітектура ігрового застосунку побудована на поєднанні моделей даних, сервісів, головного контролера, інтерфейсу та локального збереження. Такий підхід дозволяє реалізувати основні механіки Fantasy RPG: генерацію героїв і ворогів, колекціонування, формування команди, фракційні

бонуси, автоматизований бій, нагороди, завдання, режими проходження та відновлення прогресу після перезапуску гри.

Для детальнішого опису програмної структури застосунку доцільно подати діаграму класів основних модулів. Вона показує не лише загальні блоки архітектури, а й конкретні класи, які використовуються для зберігання даних, генерації ігрового контенту, проведення бою, роботи з формацією, завданнями та збереженням прогресу.

На рисунку 3.2 зображено діаграму класів основних програмних модулів ігрового застосунку.

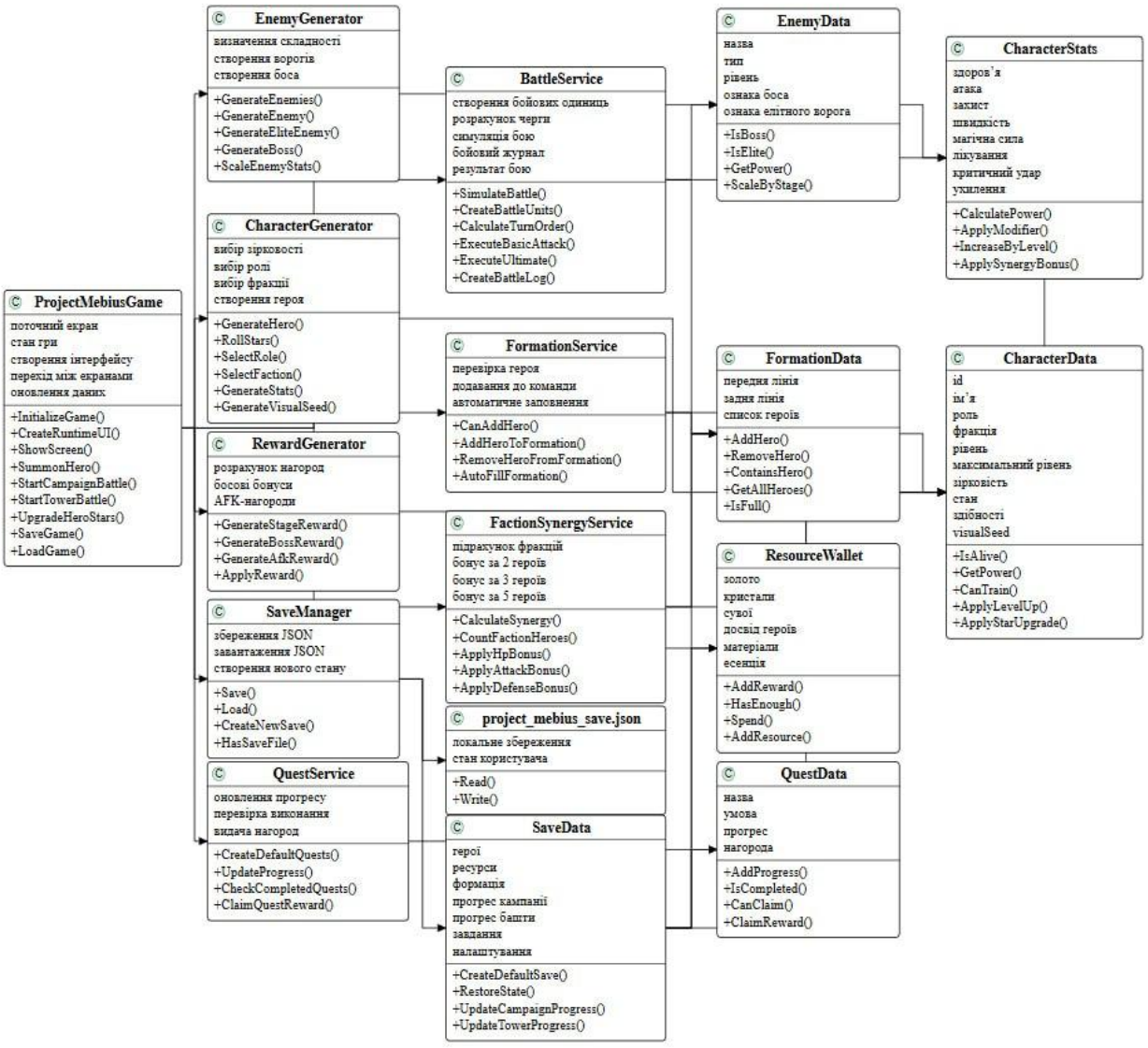


Рис. 3.2 Діаграма класів застосунку

Діаграма класів відображає поділ програмної логіки на моделі даних і сервіси. До моделей даних належать CharacterData, EnemyData, CharacterStats, ResourceWallet, FormationData, QuestData і SaveData. Вони описують основні сутності гри: героїв, ворогів, характеристики, ресурси, формацію, завдання та загальний стан збереження. Наприклад, клас CharacterData містить дані героя, а CharacterStats описує його бойові характеристики. Клас FormationData зберігає склад команди, а SaveData об’єднує всі дані, які потрібно відновити після перезапуску гри.

Окрему групу становлять сервісні класи, які виконують основну ігрову логіку. CharacterGenerator відповідає за створення нових героїв, EnemyGenerator — за формування противників, RewardGenerator — за розрахунок нагород, FormationService — за роботу з бойовою командою, FactionSynergyService — за фракційні бонуси, BattleService — за симуляцію бою, QuestService — за оновлення завдань, а SaveManager — за збереження та завантаження JSON-файлу. Центральний клас ProjectMebiusGame координує роботу цих сервісів і забезпечує взаємодію між інтерфейсом, даними та ігровими механіками.

Зв’язки між класами показують, що застосунок має єдину структуру даних. SaveData містить колекцію героїв, ресурси, формацію, завдання та налаштування. Герої й вороги пов’язані з характеристиками, формація містить до п’яти героїв, а сервіси працюють із відповідними моделями даних. Такий підхід дозволяє уникнути дублювання інформації та забезпечує узгоджену роботу генерації, бою, розвитку героїв і збереження прогресу.

3.2 Проєктування ігрового циклу та основних сценаріїв користувача

Ігровий цикл визначає послідовність дій, які користувач виконує під час взаємодії із застосунком. Для гри жанру Fantasy RPG такий цикл не обмежується лише запуском бою. Він охоплює отримання героїв, перегляд колекції, формування команди, проходження етапів, отримання нагород, розвиток

персонажів і збереження прогресу. Саме повторення цих дій створює основу довготривалого проходження.

У розробленому застосунку користувач починає роботу з головного меню. Звідти він може перейти до гри, налаштувань, титрів або завершити роботу із застосунком. Після початку гри відкривається основний хаб, який виконує роль центрального екрана. Через нього користувач переходить до заклику героїв, бібліотеки, формації, кампанії, башти, завдань, налаштувань та інших розділів. Такий підхід спрощує навігацію, оскільки всі основні можливості зібрані в одному місці.

Основний ігровий цикл можна описати як послідовність: отримання героя, додавання його до колекції, формування команди, запуск бойового етапу, отримання результату, нарахування нагород і подальше покращення героїв. Після цього користувач знову може перейти до проходження кампанії або башти. Якщо команда стає недостатньо сильною, гравець повертається до розвитку персонажів, заклику нових героїв або зміни формації.

Важливо, що користувач не зобов’язаний кожного разу проходити цикл в однаковому порядку. Наприклад, після бою він може одразу перейти до наступного етапу, якщо команда достатньо сильна. В іншому випадку він може відкрити бібліотеку, переглянути героїв, провести тренування, змінити склад команди або зібрати AFK-нагороди. Така структура робить проходження менш лінійним і дозволяє користувачу самостійно обирати подальшу дію.

На рисунку 3.2 подано загальну схему ігрового циклу застосунку.

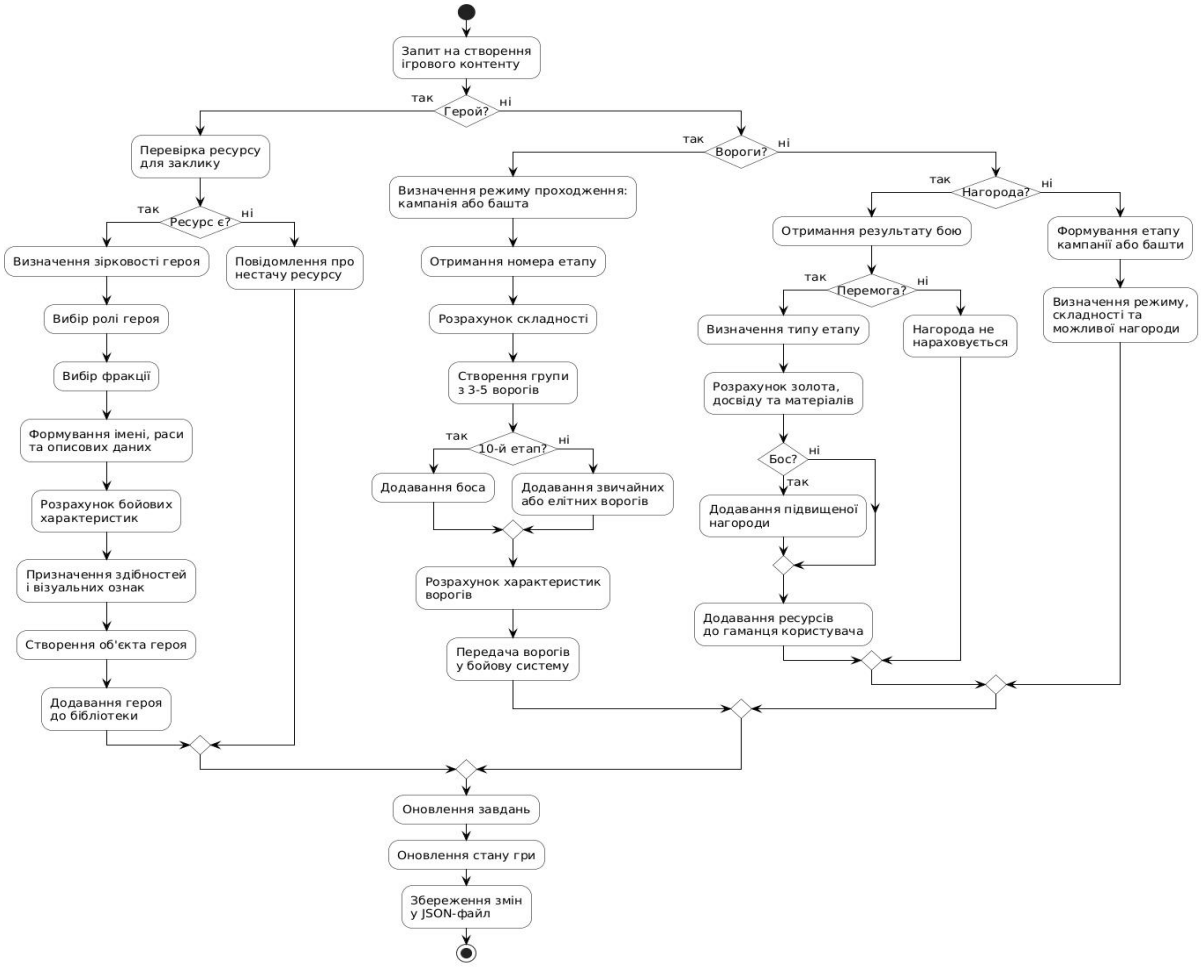


Рис. 3.2 Ігровий цикл застосунку «Project Mebius»

Першим важливим сценарієм є заклик героя. Користувач переходить до відповідного екрана, витрачає ігрові ресурси та отримує нового персонажа. Після цього герой додається до бібліотеки. Для нього зберігаються роль, фракція, зірковість, характеристики, здібності та візуальні ознаки. Якщо користувач отримав сильного або корисного героя, він може додати його до формації та використати в наступних боях.

Другим сценарієм є робота з бібліотекою героїв. У цьому розділі користувач переглядає вже отриманих персонажів, порівнює їх між собою та визначає, кого варто розвивати. Для зручності передбачено сортування й фільтрацію. Це особливо важливо в колекційній RPG, де з часом кількість героїв збільшується, і користувачу потрібно швидко знаходити персонажів потрібної ролі, фракції або сили.

Третім сценарієм є формування команди. У грі використовується формація з п’яти позицій: дві передні та три задні. Передня лінія підходить для героїв, які мають витримувати основні удари, а задня - для атакувальних персонажів, магів, цілителів або підтримки. Під час формування складу користувач також може враховувати фракційні синергії. Якщо в команді є кілька героїв однієї фракції, система надає додаткові бонуси, що впливає на результат бою.

Наступним сценарієм є проходження кампанії. Кампанія складається з послідовних етапів, складність яких поступово зростає. Перед запуском бою система формує ворогів відповідно до номера етапу та поточного режиму. Після цього запускається автоматизована бойова симуляція. Користувач не керує кожною атакою вручну, а спостерігає за перебігом бою та отримує результат на основі сили й складу команди.

Окремим режимом є башта. Вона працює як додаткове випробування, де користувач проходить рівні зі зростаючою складністю. Такий режим потрібний для урізноманітнення проходження. Якщо кампанія є основним шляхом прогресу, то башта дозволяє перевірити команду в іншому форматі та отримати додаткові нагороди.

Після завершення бою система визначає результат. У разі перемоги користувач отримує ресурси та переходить до наступного етапу. Якщо команда програла, користувач може повернутися до хабу, змінити склад формації, покращити героїв або спробувати пройти етап пізніше. Такий підхід пов’язує бойову систему з розвитком персонажів: поразка не завершує гру, а показує, що команду потрібно посилити.

Розвиток героїв є ще одним важливим сценарієм. Користувач може витрачати ресурси на тренування персонажів, підвищення рівня та зірковості. Це поступово збільшує силу команди й відкриває можливість проходити складніші етапи. У такій системі нагороди після бою мають практичне значення, оскільки вони використовуються не лише як результат перемоги, а як матеріал для подальшого прогресу.

AFK-нагороди доповнюють основний ігровий цикл. Якщо користувач певний час не запускав гру, система може нарахувати ресурси за період відсутності. При цьому накопичення має обмеження до 12 годин, щоб пасивне отримання ресурсів не порушувало баланс. Ця механіка дозволяє підтримувати розвиток команди навіть тоді, коли користувач не проходить бої постійно.

Основні сценарії користувача наведено в таблиці 3.2.

Таблиця 3.2

Основні сценарії користувача в ігровому застосунку

Сценарій	**Дії користувача**	**Результат**
Запуск гри	Відкриття застосунку, перехід із головного меню до гри	Завантаження або створення стану гри
Заклик героя	Перехід до екрана заклику, витрата ресурсів	Отримання нового героя та додавання його до бібліотеки
Перегляд бібліотеки	Відкриття списку героїв, перегляд характеристик, сортування або фільтрація	Вибір персонажів для розвитку або формації
Формування команди	Додавання героїв у передню та задню лінію	Створення бойової формації з урахуванням ролей і фракцій
Проходження кампанії	Вибір етапу, запуск бою, перегляд результату	Оновлення прогресу кампанії та отримання нагород
Проходження башти	Запуск рівня башти, участь у бою	Оновлення прогресу башти та отримання ресурсів
Розвиток героя	Вибір героя, витрата ресурсів на тренування або підвищення зірковості	Збільшення характеристик персонажа
Отримання AFK-нагород	Перехід до екрана нагород після перерви	Нарахування ресурсів за час відсутності
Виконання завдань	Виконання умов завдання, отримання винагороди	Додаткові ресурси та оновлення прогресу завдань
Зміна налаштувань	Перехід до налаштувань, зміна параметрів звуку або бою	Збереження індивідуальних параметрів користувача

Проєктування сценаріїв користувача дозволяє визначити, які екрани потрібні в застосунку та як вони мають бути пов’язані між собою. Наприклад, екран заклику пов’язаний із ресурсами й бібліотекою героїв, екран формації - з

бібліотекою та бойовою системою, а екран бою - з ворогами, нагородами, завданнями та збереженням. Завдяки цьому кожен розділ має конкретне призначення і не існує окремо від загальної логіки гри.

Важливою частиною ігрового циклу є збереження стану. Після отримання героя, зміни формації, перемоги в бою, виконання завдання або зміни налаштувань дані мають бути записані у файл збереження. Це дозволяє користувачу продовжити гру після перезапуску застосунку. Для колекційної RPG така поведінка є обов’язковою, оскільки прогрес накопичується поступово.

Отже, ігровий цикл застосунку побудований навколо послідовної взаємодії користувача з героями, командою, бойовими етапами, нагородами та розвитком персонажів. Основні сценарії охоплюють запуск гри, заклик героїв, перегляд колекції, формування формації, проходження кампанії та башти, отримання нагород, покращення персонажів і збереження прогресу. Така структура забезпечує зв’язок між усіма ключовими механіками гри та формує основу для подальшої реалізації програмних модулів.

3.3 Проєктування системи процедурної генерації героїв, ворогів і нагород

Процедурна генерація є однією з основних частин ігрової логіки. Вона використовується для того, щоб гра могла створювати різні варіанти героїв, противників, бойових етапів і нагород без ручного опису кожного окремого об’єкта. Для жанру Fantasy RPG це особливо важливо, оскільки користувач поступово збирає персонажів, проходить багато боїв і очікує, що нові герої та вороги не будуть повністю однаковими.

У застосунку генерація побудована за принципом поєднання випадковості та заданих правил. Це означає, що система не створює дані довільно, а працює в межах певних обмежень. Наприклад, герой може отримати одну із визначених ролей, належати до однієї з фракцій, мати зірковість у межах від 1 до 7 і набір характеристик, який відповідає його ролі. Такий підхід дозволяє зберегти різноманітність і водночас не порушувати логіку балансу.

Проєктування системи генерації можна поділити на три основні напрями: генерацію героїв, генерацію ворогів і генерацію нагород. Кожен із цих напрямів виконує окреме завдання, але всі вони пов’язані з основним ігровим циклом. Герої потрібні для колекції та формації користувача, вороги формують бойові ситуації, а нагороди забезпечують подальший розвиток команди.

На рисунку 3.3 подано загальну схему роботи системи процедурної генерації.

Рис. 3.3 Схема роботи системи процедурної генерації ігрового контенту

Першим етапом є генерація героя. Вона запускається під час використання системи заклику або під час створення початкового набору персонажів. Спочатку система визначає зірковість героя відповідно до встановлених імовірностей. Далі

обираються роль, фракція, раса, ім’я, базові характеристики, здібності та візуальні ознаки. Після завершення генерації герой додається до бібліотеки користувача та може бути використаний у формації.

Для героя зберігається не лише бойова інформація, а й описові параметри. До них належать ім’я, стать, раса, фракція, роль, зброя, талант, пасивна здібність, ультимативна дія та візуальний seed. Останній параметр використовується для вибору або комбінування зовнішніх ознак персонажа. Завдяки цьому герої можуть відрізнятися між собою не тільки значеннями характеристик, а й зовнішнім представленням у бібліотеці та бою.

Ролі персонажів впливають на розподіл характеристик. Наприклад, танк повинен мати більший запас здоров’я та захисту, оскільки він частіше перебуває на передній лінії. Лучник або маг можуть мати вищі атакувальні показники, але меншу витривалість. Цілитель і підтримка мають бути корисними не тільки через шкоду, а й через лікування або допоміжні ефекти. Завдяки цьому під час формування команди користувач враховує не лише загальну силу героя, а й його призначення.

Для порівняння персонажів у грі використовується показник сили героя. Він розраховується на основі основних характеристик персонажа.

Основні етапи генерації героя наведено в таблиці 3.3.

Таблиця 3.3

Етапи процедурної генерації героя

Етап	Дія системи	Результат
1	Визначення зірковості	Герой отримує рідкість від 1 до 7 зірок
2	Вибір ролі	Визначається бойове призначення персонажа
3	Вибір фракції	Герой належить до однієї з фракцій гри
4	Формування імені та описових даних	Створюється ім’я, раса, стать, зброя та інші відомості
5	Розрахунок характеристик	Визначаються здоров’я, атака, захист, швидкість та інші параметри

Продовження таблиці 3.3

Етапи процедурної генерації героя

Етап	Дія системи	Результат
6	Призначення здібностей	Герой отримує талант, пасивну та спеціальну дію
7	Вибір візуальних ознак	Призначається спрайт, колір або інші елементи зовнішності
8	Додавання до бібліотеки	6

Другим напрямом є генерація ворогів. Вона виконується перед запуском бойового етапу. На відміну від героїв, вороги не зберігаються в колекції користувача, а створюються для конкретного бою. Система враховує режим проходження, номер етапу, рівень складності та силу команди. На одному етапі формується від трьох до п’яти противників. Це дає змогу створювати бої різної щільності та поступово ускладнювати проходження.

Звичайні вороги використовуються на більшості етапів. Елітні противники можуть з’являтися на окремих рівнях і мають посилені характеристики. Босові бої розміщуються на кожному десятому етапі. Такий поділ робить структуру проходження більш зрозумілою: користувач поступово проходить звичайні бої, а через певний проміжок стикається з сильнішим ворогом.

Під час генерації ворога визначається його назва, тип, рівень, характеристики, роль у бою та ознака звичайного, елітного або босового противника. Для босів використовуються окремі назви та підвищені параметри. Завдяки цьому користувач бачить, що бій із босом відрізняється від звичайного етапу не лише числовою складністю, а й подачею.

Загальну логіку генерації ворогів наведено в таблиці 3.4.

Таблиця 3.4

Особливості генерації ворогів

Тип противника	Умови появи	Особливості
Звичайний ворог	Більшість етапів кампанії та башти	Має базові характеристики відповідно до складності

Продовження таблиці 3.4

Особливості генерації ворогів

Тип противника	Умови появи	Особливості
Елітний ворог	Окремі посилені етапи	Має вищі бойові показники порівняно зі звичайним противником
Бос	Кожен десятий етап	Має підвищену силу, окрему назву та складніший бій
Група ворогів	Перед кожним боєм	Містить від 3 до 5 противників

Третім напрямом є генерація нагород. Вона виконується після перемоги в бою або під час нарахування AFK-ресурсів. Нагорода може містити золото, кристали, сувої, досвід героїв, матеріали для тренування та есенцію. Ці ресурси потрібні для подальшого розвитку персонажів, підвищення зірковості та отримання нових героїв.

Нагорода залежить від режиму проходження та складності етапу. Звичайні етапи дають базовий обсяг ресурсів, а босові бої мають приносити ціннішу винагороду. Такий підхід підтримує логіку прогресу: чим складніший етап проходить користувач, тим більшою має бути користь від перемоги. Це також мотивує повертатися до розвитку команди, якщо поточна сила героїв уже недостатня для наступних боїв.

AFK-нагороди працюють окремо від звичайних бойових винагород. Вони нараховуються за час відсутності користувача, але мають обмеження до 12 годин. Це обмеження потрібне для того, щоб пасивне накопичення ресурсів не порушувало баланс гри. Користувач може отримати певну кількість матеріалів після повернення, але активне проходження кампанії та башти все одно залишається важливою частиною прогресу.

Усі згенеровані дані мають бути сумісними з іншими системами. Герой повинен відображатися в бібліотеці, додаватися до формації, брати участь у бою та зберігатися у файлі. Ворог повинен мати характеристики, які може використати бойова система. Нагорода повинна додаватися до ресурсів користувача. Саме

тому генерація пов’язана не тільки з випадковим вибором, а й зі структурою даних усього застосунку.

Важливо, що генерація не працює ізольовано. Вона пов’язана з інтерфейсом, збереженням, бойовою системою та системою завдань. Наприклад, після отримання нового героя інтерфейс показує результат заклику, збереження записує героя до колекції, а завдання можуть оновити прогрес за дію заклику. Після перемоги бойова система передає результат, генератор нагород формує винагороду, а ресурси користувача оновлюються.

Отже, система процедурної генерації охоплює створення героїв, ворогів і нагород. Вона дає змогу формувати різні ігрові ситуації, підтримує колекціонування персонажів, поступове ускладнення боїв і розвиток команди. Поєднання випадковості з установленими правилами дозволяє зберегти різноманітність і водночас не виходити за межі логіки ігрового балансу.

3.4 Реалізація системи колекціонування, формування команди та розвитку героїв

Система героїв є однією з головних частин ігрового застосунку, оскільки саме через неї користувач отримує нових персонажів, переглядає їх у бібліотеці, формує бойову команду та поступово посилює склад. Для гри жанру Fantasy RPG така система має бути пов’язана не тільки з візуальним відображенням персонажів, а й з бойовою логікою, прогресом, ресурсами та збереженням даних.

У програмній реалізації герой описується як окрема модель даних. Для нього зберігаються ім’я, стать, раса, фракція, роль, рівень, максимальний рівень, зірковість, стан, характеристики, зброя, талант, пасивна здібність, ультимативна дія та візуальні параметри. Такий набір даних дозволяє використовувати одного й того самого героя в різних частинах гри: у бібліотеці, формації, бойовій системі, тренуванні, підвищенні зірковості та збереженні прогресу.

Колекціонування героїв реалізовано через систему заклику. Коли користувач витрачає відповідний ресурс, система створює нового персонажа за правилами процедурної генерації. Після цього герой додається до загального

списку персонажів користувача і стає доступним у бібліотеці. Користувач може переглянути його основні дані, порівняти з іншими героями та вирішити, чи варто додавати його до бойової команди.

Бібліотека героїв виконує роль основного екрана для роботи з колекцією. У ній користувач бачить отриманих персонажів, їхню зірковість, роль, фракцію, рівень, силу та стан. Для зручності передбачено сортування й фільтрацію. Це потрібно тому, що після кількох закликів кількість героїв поступово збільшується, і без додаткових засобів пошуку користувачу було б складно швидко знайти потрібного персонажа.

Основні дані героя, які використовуються в бібліотеці та бойовій системі, наведено в таблиці 3.5.

Таблиця 3.5

Основні дані героя в ігровому застосунку

Група даних	**Приклади параметрів**	**Використання**
Описові дані	ім’я, стать, раса, фракція, роль	Відображення героя в бібліотеці та картці персонажа
Прогрес	рівень, максимальний рівень, зірковість, стан	Тренування, підвищення зірковості, перевірка доступності героя
Бойові характеристики	здоров’я, атака, захист, швидкість, магічна сила, лікування	Розрахунок сили героя та участь у бою
Здібності	талант, пасивна дія, ультимативна дія	Використання під час автоматизованого бою
Візуальні ознаки	sprite key, visual seed, кольори, варіанти вигляду	Відображення персонажа в інтерфейсі та бою

Формування команди реалізовано через систему формації. У грі використовується п’ять позицій: дві передні та три задні. Передня лінія призначена для героїв, які мають вищу витривалість і можуть приймати на себе основну частину пошкоджень. Задня лінія більше підходить для лучників, магів, цілителів і персонажів підтримки. Така схема змушує користувача не просто обирати найсильніших героїв, а враховувати їхню роль у бою.

Для роботи з формацією використовується FormationService. Він перевіряє, чи можна додати героя до команди, чи не зайнята відповідна позиція та чи герой доступний для бою. Якщо персонаж має стан загиблого, він не повинен залишатися у формації. Це пов’язано з тим, що в бойовій системі передбачено втрату героїв після невдалих боїв, тому склад команди має постійно відповідати актуальному стану колекції.

У застосунку також передбачено автоматичне заповнення формації. Такий механізм корисний на початку гри, коли користувач ще не має великої кількості героїв або не знає, кого краще додати до команди. Система може обрати доступних персонажів за силою та роллю і розмістити їх у відповідних позиціях. Надалі користувач може змінити склад вручну.

Важливим елементом командної системи є фракційні синергії. Герої належать до різних фракцій, і якщо в команді є кілька персонажів однієї фракції, система надає додаткові бонуси. Це робить формування команди більш стратегічним, оскільки користувач має враховувати не тільки окремі характеристики героя, а й те, як він поєднується з іншими персонажами.

У грі використовується кілька рівнів фракційного бонусу. Якщо в команді є два герої однієї фракції, команда отримує бонус до здоров’я. Якщо таких героїв три, додатково посилюється атака або захист залежно від фракції. Якщо в команді п’ять героїв однієї фракції, команда отримує посилений набір бонусів і додаткову енергію на початку бою. Основні умови синергії наведено в таблиці 3.6.

Таблиця 3.6

Умови нарахування фракційних бонусів

Кількість героїв однієї фракції	**Бонус**	**Призначення**
2 герої	+5 % до здоров’я	Підвищує витривалість команди
3 герої	+10 % до атаки або захисту	Посилює бойову роль фракції
5 героїв	+8 % до здоров’я, атаки й захисту, +10 енергії	Дає найбільший командний бонус

Розвиток героїв реалізовано через тренування та підвищення зірковості. Тренування дозволяє збільшувати рівень персонажа за ресурси. Для цього використовуються досвід героїв і матеріали тренування. Після підвищення рівня збільшуються основні характеристики персонажа, а отже зростає його бойова ефективність. Така система дає користувачу змогу посилювати вже отриманих героїв, а не тільки чекати появи нових.

Максимальний рівень героя залежить від його зірковості. У грі використовується обмеження, за яким герой вищої зірковості може досягати більшого рівня. Це створює зв’язок між рідкістю, розвитком і довготривалим прогресом. Максимальні рівні для героїв різної зірковості наведено в таблиці 3.7.

Таблиця 3.7

Обмеження максимального рівня героя залежно від зірковості

Зірковість героя	**Максимальний рівень**
1 зірка	20
2 зірки	40
3 зірки	60
4 зірки	70
5 зірок	80
6 зірок	90
7 зірок	100

Підвищення зірковості використовується як окремий спосіб розвитку персонажа. Воно доступне після досягнення героєм максимального рівня для поточної зірковості. Для покращення потрібні золото та есенція. Після підвищення зірковості збільшується межа рівня і посилюються характеристики. Максимальна зірковість героя обмежена значенням 7.

Загальну послідовність розвитку героя можна описати так: користувач отримує персонажа, додає його до бібліотеки, за потреби включає до формації, проходить бої, отримує ресурси, тренує героя до максимального рівня, після чого може підвищити його зірковість. Потім цикл повторюється, але вже з вищою межею розвитку. Такий підхід підтримує довготривалу прогресію і дає користувачу зрозумілу мету для посилення команди.

На рисунку 3.4 подано діаграму взаємодії системи колекціонування, формації та розвитку героїв.

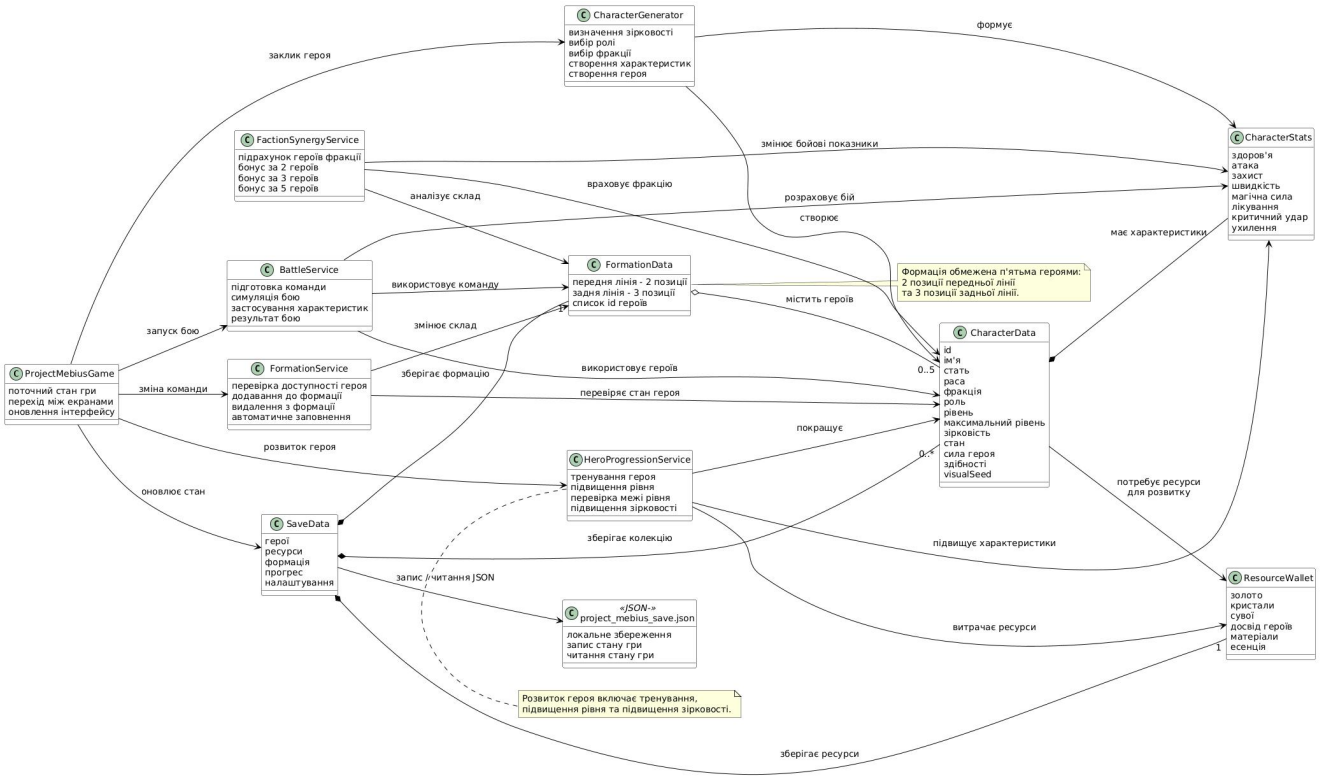


Рис. 3.4 Діаграма взаємодії системи героїв, формації та розвитку

Система колекціонування тісно пов’язана із збереженням даних. Кожен отриманий герой має бути записаний у файл збереження, інакше після перезапуску гри користувач втратить прогрес. Так само потрібно зберігати рівень героя, зірковість, стан, формацію та ресурси. Якщо користувач змінив склад команди або покращив персонажа, ці зміни повинні відновлюватися під час наступного запуску.

З бойовою системою герої пов’язані через формацію. Перед початком бою система отримує список героїв, які перебувають у команді, перевіряє їхній стан і використовує характеристики для симуляції. Після бою результат може вплинути на подальший стан героїв, ресурси користувача, виконання завдань і прогрес проходження. Тому система героїв не є окремим екраном, а працює разом з іншими частинами застосунку.

Отже, система колекціонування, формування команди та розвитку героїв забезпечує один із головних ігрових циклів. Користувач отримує персонажів, переглядає їх у бібліотеці, формує формацію, враховує ролі та фракційні бонуси,

проходить бої й витрачає ресурси на покращення. Така структура поєднує колекційну, стратегічну та рольову складові гри.

3.5 Реалізація бойової системи та режимів проходження

Бойова система є центральною частиною ігрового процесу, оскільки саме через неї перевіряється ефективність зібраної команди, роль героїв, фракційні бонуси та рівень розвитку персонажів. У застосунку бій реалізовано в автоматизованому форматі. Користувач не керує кожною дією героя вручну, а спочатку формує команду, розміщує персонажів у формації, після чого запускає бойовий етап. Далі система самостійно обчислює перебіг бою за встановленими правилами.

Такий підхід відповідає логіці idle/autobattler RPG. Основне рішення користувач приймає до початку бою: кого додати до складу, які ролі використати, чи є в команді цілитель або підтримка, чи працюють фракційні синергії, чи достатньо розвинені герої для поточного етапу. Сам бій у такій моделі є результатом підготовки команди, а не ручного керування кожною атакою.

У програмній реалізації за проведення бою відповідає модуль BattleService. Він отримує склад формації користувача, створену групу ворогів, інформацію про режим проходження та застосовує правила симуляції. На початку бою система перетворює героїв і ворогів на бойові одиниці, у яких зберігаються поточні характеристики, здоров’я, енергія, роль, стан і доступні дії.

Бій проходить по раундах. У кожному раунді система визначає порядок дій учасників. На черговість впливає швидкість персонажа, а також невеликий випадковий модифікатор. Це дозволяє уникнути повністю однакового перебігу бою навіть у випадку схожих команд. Після визначення черги кожна жива бойова одиниця виконує дію: атакує ворога, застосовує здібність, лікує союзника або активує посилену дію, якщо накопичено достатньо енергії.

Основні етапи автоматизованого бою наведено в таблиці 3.8.

Таблиця 3.8

Основні етапи автоматизованої бойової системи

Етап	Дія системи	Результат
1	Отримання формації користувача	Визначається склад героїв, які беруть участь у бою
2	Генерація або завантаження ворогів	Формується група противників для поточного етапу
3	Створення бойових одиниць	Герої та вороги перетворюються на учасників бою з поточними характеристиками
4	Розрахунок черги ходу	Порядок дій визначається швидкістю та випадковим модифікатором
5	Виконання дій	Учасники атакують, лікують, використовують здібності або накопичують енергію
6	Перевірка стану команд	Система визначає, чи залишилися живі герої або вороги
7	Завершення бою	Фіксується перемога або поразка, формується бойовий журнал
8	Оновлення прогресу	У разі перемоги нараховуються нагороди та оновлюється проходження

У бою враховуються основні характеристики персонажів: здоров’я, атака, захист, швидкість, магічна сила, сила лікування, шанс критичного удару та шанс ухилення. Під час атаки система перевіряє, чи вдалося цілі ухилитися. Якщо ухилення не спрацювало, розраховується шкода з урахуванням атаки нападника та захисту цілі. Додатково може спрацювати критичний удар, який збільшує підсумкову шкоду.

Ролі героїв впливають на їхню поведінку в бою. Танк частіше використовується для прийняття основних ударів, воїн і лучник завдають фізичної шкоди, маг використовує посилені магічні атаки, цілитель відновлює здоров’я союзників, убивця орієнтований на швидке усунення слабших цілей, а підтримка допомагає команді за рахунок додаткових ефектів. Завдяки цьому кожен герой має своє призначення, а результат бою залежить не лише від загальної сили, а й від складу команди.

Окрему роль відіграє система енергії. Під час бою герої накопичують енергію за виконані дії. Коли її достатньо, персонаж може використати сильнішу здібність. У грі також передбачено командну суперздібність, яка заряджається залежно від дій живих героїв і може посилюватися фракційними бонусами. Така механіка додає бою додаткову подію, яка може вплинути на результат у складніших ситуаціях.

Узагальнені бойові дії наведено в таблиці 3.9.

Таблиця 3.9

Основні типи бойових дій

Тип дії	**Хто виконує**	**Призначення**
Базова атака	Герої та вороги	Завдання основної шкоди цілі
Критичний удар	Герої та вороги	Збільшення шкоди за умови спрацювання критичного шансу
Ухилення	Герої та вороги	Уникнення отримання шкоди
Лікування	Цілителі або підтримка	Відновлення здоров’я союзника
Спеціальна здібність	Герої відповідної ролі	Посилена дія, пов’язана з роллю персонажа
Босова дія	Босові вороги	Посилена атака або особлива дія сильного противника
Командна суперздібність	Команда користувача	Спільна дія героїв після накопичення заряду

Під час бою система формує бойовий журнал. У ньому фіксуються основні події: атаки, критичні удари, ухилення, лікування, використання здібностей, усунення ворогів і підсумок бою. Журнал потрібний для того, щоб користувач міг зрозуміти, як саме проходила битва. Це важливо для автоматизованого бою, оскільки гравець не керує діями вручну й має бачити пояснення результату.

Бойова симуляція має обмеження за кількістю раундів. Якщо бій не завершується раніше, він припиняється після 160 раундів. Таке обмеження потрібне для захисту від ситуацій, коли жодна зі сторін не може перемогти через низьку шкоду або надмірне лікування. У такому випадку система все одно

повинна завершити симуляцію та визначити результат, щоб застосунок не зависав на одному бою.

Після завершення бою система перевіряє результат. Якщо команда користувача перемогла, оновлюється прогрес кампанії або башти, нараховуються нагороди та оновлюються завдання. Якщо команда програла, користувач повертається до хабу або екрана підготовки й може змінити формацію, покращити героїв або спробувати пройти етап пізніше. Такий підхід пов’язує бій із системою розвитку: складніші етапи змушують користувача переглядати команду та посилювати персонажів.

У застосунку також передбачено стан героя після бою. Якщо персонаж втрачає все здоров’я, він позначається як недоступний для подальшого використання й видаляється з формації. Це додає значення підготовці до бою, оскільки користувач має уважніше ставитися до складу команди та не відправляти слабких героїв на надто складні етапи.

Режими проходження поділяються на кампанію та башту. Кампанія є основним напрямом прогресу. Вона складається з послідовних етапів, складність яких поступово зростає. На певних етапах можуть з’являтися сильніші вороги або боси. Після перемоги користувач переходить далі й отримує ресурси для розвитку героїв.

Башта працює як додатковий режим. Її рівні також ускладнюються поступово, але сприймаються як окреме випробування для вже сформованої команди. Такий режим дає змогу урізноманітнити проходження і надати користувачу ще один шлях отримання ресурсів. Якщо кампанія виконує роль основної лінії просування, то башта дозволяє перевірити силу команди в іншому форматі.

Порівняння режимів проходження наведено в таблиці 3.10.

Таблиця 3.10

Характеристика режимів проходження

Режим	Призначення	Особливості
Кампанія	Основне послідовне проходження	Етапи відкриваються поступово, складність зростає, на окремих рівнях з’являються боси
Башта	Додаткове випробування для команди	Рівні проходяться окремо, складність підвищується з кожним поверхом
Босовий етап	Перевірка сили команди на складнішому рівні	Бос з’являється на кожному десятому етапі та має посилені характеристики
Повторна спроба	Можливість пройти етап після поразки	Користувач може змінити формацію або покращити героїв перед наступним боєм

На рисунку 3.5 подано узагальнену схему роботи бойової системи та режимів проходження.

Бойова система також пов’язана із системою нагород. Після перемоги користувач отримує ресурси, які можуть бути використані для заклику нових героїв, тренування або підвищення зірковості. Це підтримує основний ігровий цикл: команда проходить бій, отримує винагороду, посилюється й переходить до складнішого етапу. Якщо користувач програє, він повертається до підготовки, що також є частиною циклу.

Завдання оновлюються залежно від виконаних дій. Наприклад, проходження бою, перемога над ворогами, отримання нагород або розвиток героя можуть впливати на прогрес завдань. Таким чином, бойова система не працює окремо, а передає результати іншим частинам застосунку.

Отже, реалізація бойової системи та режимів проходження поєднує автоматизовану симуляцію, характеристики героїв і ворогів, ролі персонажів, здібності, командну суперздібність, нагороди та прогрес користувача. Кампанія і башта забезпечують різні способи проходження, а результат бою впливає на розвиток команди й подальший стан гри.

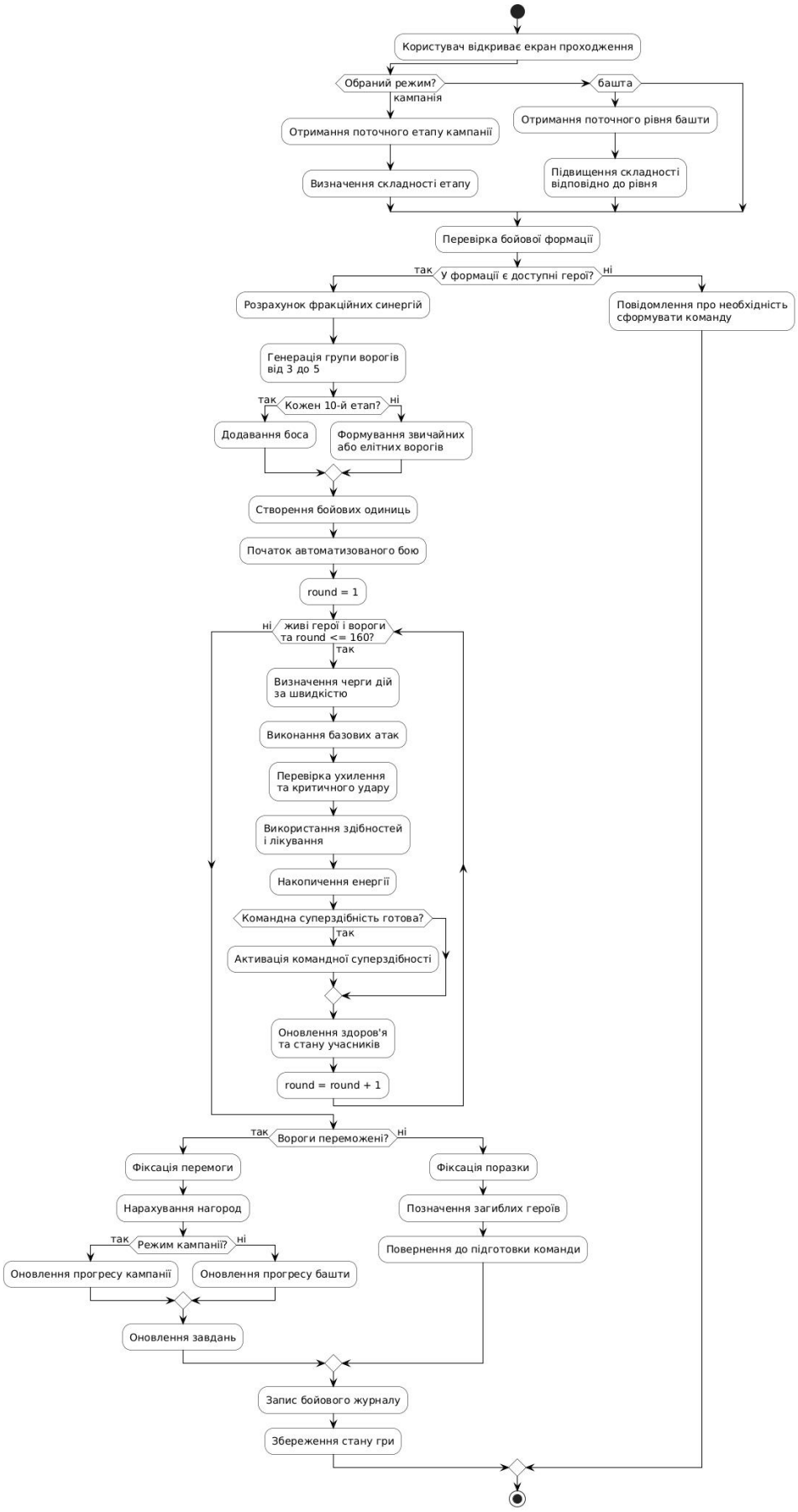


Рис. 3.5 Схема роботи бойової системи та режимів проходження

3.6 Реалізація інтерфейсу користувача та системи збереження даних

Інтерфейс користувача в ігровому застосунку реалізовано як набір взаємопов’язаних екранів, через які гравець отримує доступ до основних функцій гри. До таких екранів належать головне меню, хаб, заклик героїв, бібліотека персонажів, формація команди, карта кампанії, режим башти, бойовий екран, завдання, налаштування та титри. Кожен екран має окреме призначення, але всі вони працюють із загальним станом гри.

Після запуску застосунку створюється основний інтерфейсний Canvas. Він використовується як контейнер для всіх елементів інтерфейсу: кнопок, панелей, текстових блоків, іконок, зображень героїв, повідомлень і бойового журналу. Базова роздільна здатність інтерфейсу становить 1920×1080 пікселів, що дозволяє зручно розміщувати основні елементи на екрані та підтримувати горизонтальну орієнтацію гри.

Для відображення тексту використовується TextMeshPro. Це важливо, оскільки застосунок містить велику кількість текстових даних українською мовою: назви кнопок, характеристики героїв, фракції, ролі, бойові повідомлення, описи нагород, завдання та налаштування. У грі користувач постійно читає інформацію про героїв, ресурси й результати бою, тому текстові елементи мають бути достатньо чіткими та зрозумілими.

Головне меню є початковим екраном застосунку. З нього користувач може розпочати гру, перейти до налаштувань, переглянути титри або завершити роботу. Після початку гри відкривається хаб. Він виконує роль центрального екрана, з якого доступні основні розділи. Така структура спрощує навігацію, оскільки користувач не переходить між екранами випадково, а повертається до одного головного центру керування.

Екран заклику героїв призначений для отримання нових персонажів. Після натискання відповідної кнопки система перевіряє наявність потрібного ресурсу, запускає генерацію героя, додає його до колекції та показує результат

користувачу. На цьому екрані важливо одразу відобразити основні дані нового героя: зірковість, роль, фракцію, ім’я та візуальне представлення. Це підсилює відчуття отримання нового персонажа і робить систему колекціонування зрозумілішою.

Бібліотека героїв використовується для перегляду колекції. На цьому екрані користувач бачить отриманих персонажів, їхній рівень, зірковість, роль, фракцію, силу та стан. Також передбачено сортування й фільтрацію, що спрощує роботу з великою кількістю героїв. Якщо користувач хоче знайти найсильнішого персонажа, героя певної ролі або фракції, він може скористатися відповідними засобами впорядкування.

Екран формації дає змогу сформувати бойову команду. У ньому відображаються дві позиції передньої лінії та три позиції задньої лінії. Користувач обирає героїв із бібліотеки й додає їх до відповідних слотів. Якщо герой недоступний або не може брати участь у бою, система не повинна залишати його у складі команди. Також на цьому екрані можна показувати інформацію про фракційні синергії, щоб користувач бачив, які бонуси отримує команда.

Бойовий екран відрізняється від інших тим, що він показує не тільки статичні дані, а й перебіг симуляції. На ньому відображаються герої, вороги, журнал дій, результат бою та отримані нагороди. Оскільки бій проходить автоматично, бойовий журнал має важливе значення. Він пояснює користувачу, які дії виконували персонажі, хто завдав шкоди, хто ухилився, хто використав здібність і чим завершився бій.

Основні екрани інтерфейсу наведено в таблиці 3.11.

Таблиця 3.11

Основні екрани інтерфейсу користувача

Екран	Призначення	Основні елементи
Головне меню	Початок роботи із застосунком	Кнопки запуску гри, налаштувань, титрів і виходу
Хаб	Центральний екран навігації	Переходи до заклику, бібліотеки, команди, кампанії, башти й завдань
Заклик героїв	Отримання нових персонажів	Кнопки заклику, результат генерації, дані нового героя
Бібліотека	Перегляд колекції героїв	Список персонажів, характеристики, сортування, фільтрація
Формація	Формування бойової команди	2 передні позиції, 3 задні позиції, обрані герої
Кампанія	Послідовне проходження етапів	Поточний етап, кнопка запуску бою, інформація про прогрес
Башта	Додатковий режим проходження	Рівень башти, запуск бою, складність
Бій	Відображення бойової симуляції	Герої, вороги, бойовий журнал, результат, нагороди
Завдання	Додаткові цілі користувача	Назва завдання, прогрес, нагорода
Налаштування	Зміна параметрів гри	Гучність, швидкість бою, анімації, візуальні ефекти

Для візуального оформлення використовуються локальні ресурси проєкту: фони, спрайти героїв і ворогів, іконки, панелі, кнопки, аудіофайли та шрифти. Такі ресурси завантажуються з папок Unity-проєкту, тому застосунок не потребує звернення до зовнішніх сервісів під час роботи. Це робить гру незалежною від мережевого підключення та спрощує запуск на персональному комп’ютері.

Окремо реалізовано налаштування користувача. Вони дозволяють змінювати гучність музики та звукових ефектів, швидкість бою, показ анімацій, трясіння екрана й відображення бойових чисел. Такі параметри не змінюють основну логіку гри, але впливають на зручність користування. Після зміни

налаштування мають зберігатися, щоб під час наступного запуску застосунок відновив обрані значення.

На рисунку 3.6 подано узагальнену схему взаємодії інтерфейсу користувача із системою збереження.

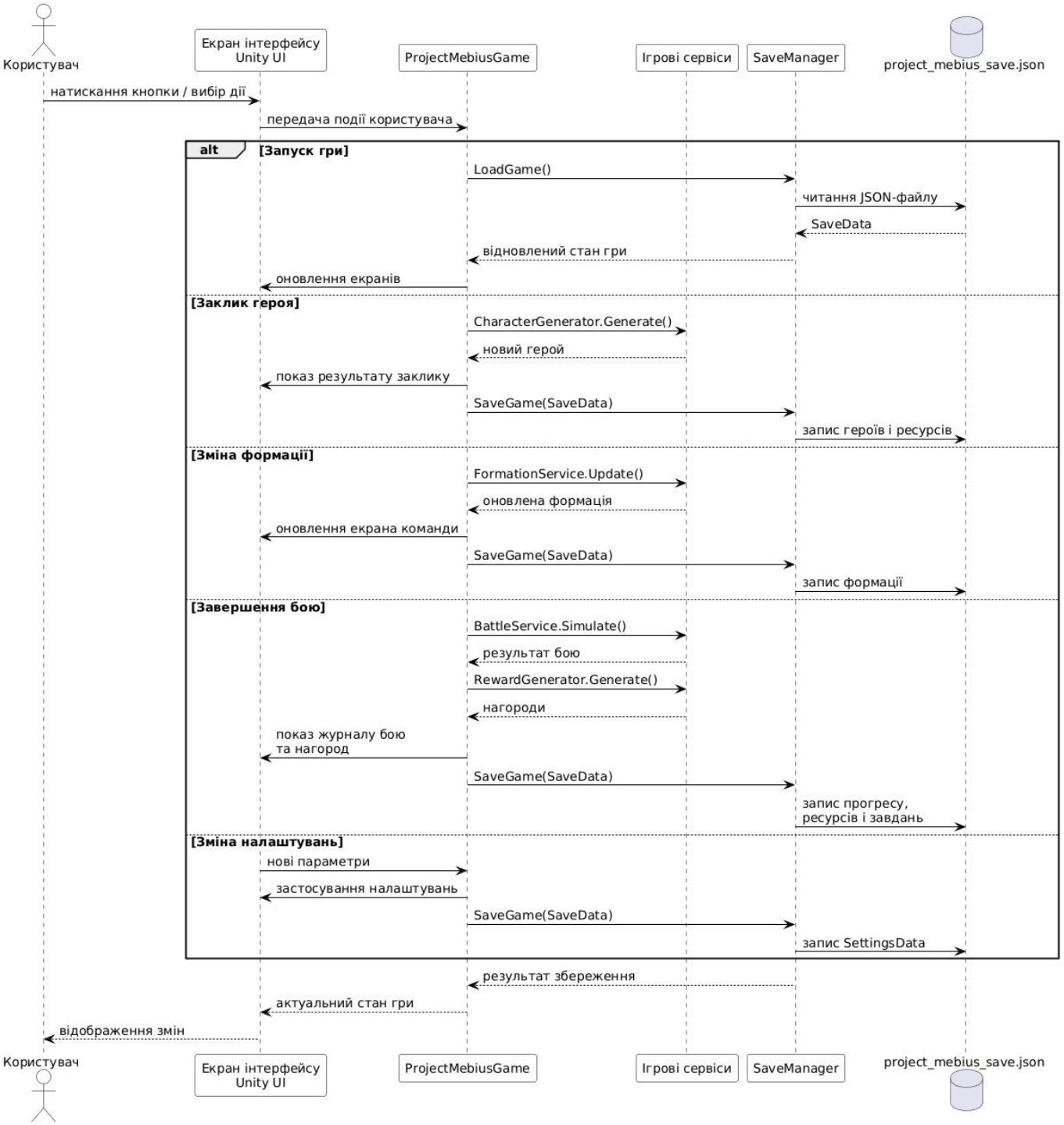


Рис. 3.6 Взаємодія інтерфейсу користувача із системою збереження даних

Система збереження потрібна для того, щоб користувач міг продовжити гру після перезапуску застосунку. У грі з колекціонуванням персонажів це особливо важливо, оскільки користувач поступово накопичує героїв, ресурси, проходить етапи, змінює формацію та виконує завдання. Якщо ці дані не зберігати, прогрес буде втрачено, а основний ігровий цикл не працюватиме коректно.

Збереження реалізовано у форматі JSON. Файл збереження розміщується в локальній директорії, доступній через Application.persistentDataPath, і має назву project_mebius_save.json. Під час запуску застосунок перевіряє, чи існує файл збереження. Якщо файл знайдено, дані завантажуються, і користувач продовжує гру з попереднього стану. Якщо файлу немає, створюється початковий стан із базовими ресурсами та стартовими героями.

До структури збереження входять усі дані, які потрібні для відновлення гри. Це ресурси користувача, список героїв, формація, прогрес кампанії, прогрес башти, завдання, налаштування та час останнього входу. Час останнього входу використовується для розрахунку AFK-нагород. Якщо користувач певний час не запускав гру, система визначає тривалість відсутності та нараховує ресурси з обмеженням до 12 годин.

Структуру даних, які зберігаються у файлі, наведено в таблиці 3.12.

Таблиця 3.12

Дані, що входять до структури збереження

Група даних	Приклади значень	Для чого використовується
Ресурси	золото, кристали, сувої, досвід героїв, матеріали, есенція	Для заклику героїв, тренування та підвищення зірковості
Герої	ім’я, роль, фракція, рівень, зірковість, характеристики, стан	Для відновлення колекції користувача
Формація	герої передньої та задньої лінії	Для запуску бою з попередньо обраною командою
Кампанія	номер пройденого етапу	Для продовження основного проходження
Башта	поточний рівень башти	Для продовження додаткового режиму
Завдання	прогрес, стан виконання, отримання нагород	Для відновлення ігрових цілей
Налаштування	гучність, швидкість бою, анімації, ефекти	Для відновлення параметрів користувача
Час останнього входу	дата й час завершення попередньої сесії	Для розрахунку AFK-нагород

Збереження виконується після важливих дій користувача. Наприклад, після отримання нового героя, зміни формації, завершення бою, нарахування нагород, виконання завдання або зміни налаштувань. Це потрібно для того, щоб дані не втрачалися, якщо користувач закриє застосунок після виконання дії. Під час наступного запуску відновлений стан має відповідати останнім змінам.

Система збереження також пов’язана з AFK-нагородами. Коли користувач виходить із гри, у збереженні фіксується час останньої сесії. Під час наступного запуску застосунок порівнює цей час із поточним і визначає, скільки ресурсів можна нарахувати. При цьому враховується обмеження до 12 годин, щоб користувач не міг отримувати необмежену кількість ресурсів без активного проходження.

Важливо, що інтерфейс і збереження працюють разом. Якщо користувач отримав героя, інтерфейс показує результат заклику, а система збереження записує нового персонажа у файл. Якщо користувач змінив команду, екран формації оновлюється, а новий склад записується до збереження. Якщо після бою отримано нагороду, її потрібно показати на екрані та додати до ресурсів у збережених даних.

Для узагальнення основних дій користувача в ігровому застосунку доцільно подати діаграму прецедентів. Вона відображає, з якими функціями взаємодіє гравець і які системні дії виконуються під час роботи застосунку. На відміну від опису окремих екранів, така діаграма показує не інтерфейсні елементи, а загальні сценарії використання гри: запуск, заклик героя, перегляд бібліотеки, формування команди, розвиток персонажів, проходження кампанії та башти, отримання нагород, виконання завдань і зміну налаштувань.

На рисунку 3.8 подано діаграму прецедентів ігрового застосунку «Project Mebius».

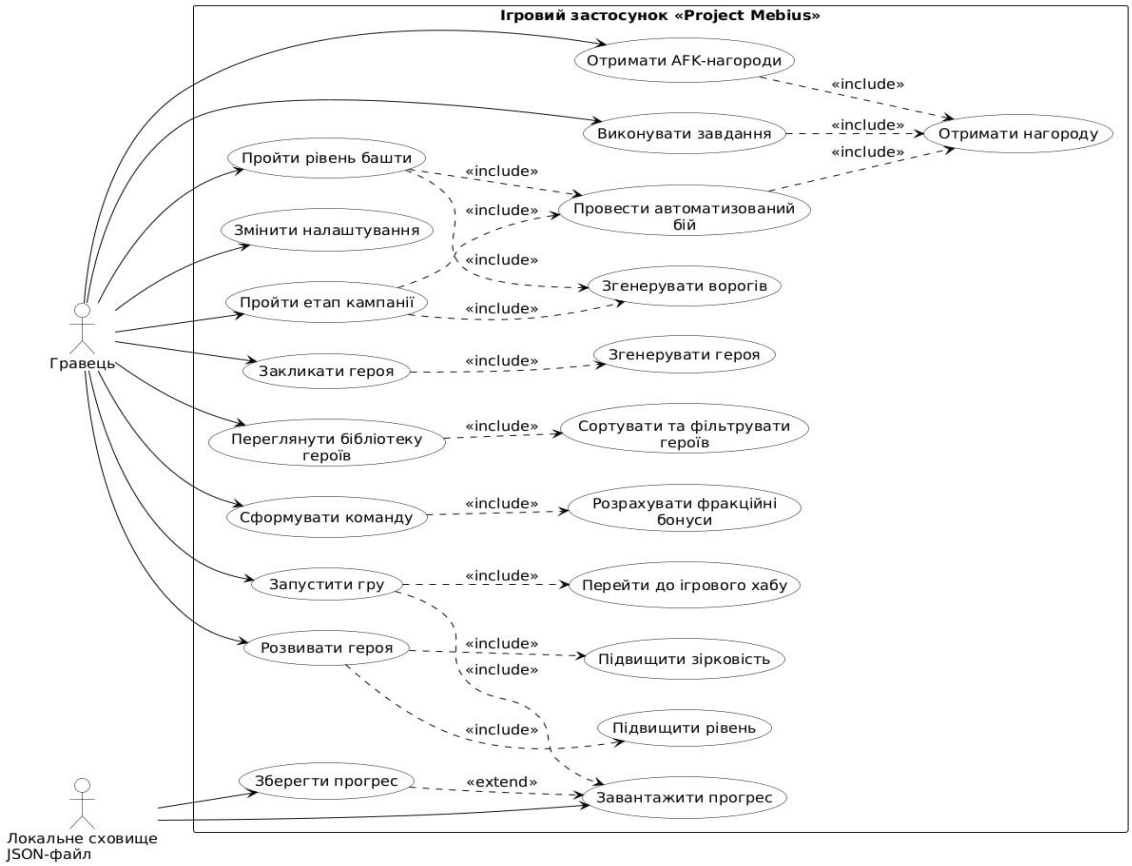


Рис. 3.8 Діаграма прецедентів ігрового застосунку «Project Mebius»

Діаграма прецедентів показує, що головним актором системи є гравець. Він запускає гру, переходить до ігрового хабу та взаємодіє з основними функціями застосунку. До таких функцій належать заклик героя, перегляд бібліотеки персонажів, формування бойової команди, розвиток героя, проходження кампанії, проходження башти, отримання AFK-нагород, виконання завдань і зміна налаштувань. Ці сценарії відповідають основному ігровому циклу, у якому користувач отримує героя, додає його до команди, проходить бойовий етап, отримує ресурси та використовує їх для подальшого розвитку.

Окремо на діаграмі відображено локальне сховище у вигляді JSON-файлу. Воно не є користувачем застосунку, але бере участь у системних сценаріях завантаження та збереження прогресу. Під час запуску гри застосунок звертається до файлу збереження, щоб відновити попередній стан. Після важливих дій, зокрема отримання героя, зміни формації, проходження бою, отримання нагород або зміни налаштувань, дані мають бути записані у файл збереження.

Зв’язки між прецедентами показують залежність основних дій від допоміжних операцій. Наприклад, заклик героя пов’язаний із генерацією персонажа, проходження кампанії або башти — з генерацією ворогів і проведенням автоматизованого бою, а бій — з отриманням нагороди. Такий підхід дозволяє показати, що функції гри не існують окремо, а працюють як частини єдиного процесу. Діаграма також підтверджує, що інтерфейс користувача пов’язаний із внутрішньою логікою застосунку та системою збереження даних.

Отже, реалізація інтерфейсу користувача та системи збереження забезпечує зв’язок між діями гравця і внутрішнім станом застосунку. Інтерфейс дає змогу користувачу переходити між екранами, переглядати героїв, формувати команду, запускати бої та змінювати налаштування. Система збереження відповідає за те, щоб усі важливі результати цих дій залишалися доступними після повторного запуску гри.

3.7 Тестування ігрового застосунку

Тестування ігрового застосунку виконувалося для перевірки того, чи працюють основні механіки гри відповідно до визначених вимог. Оскільки застосунок складається з кількох пов’язаних систем, перевіряти потрібно було не лише окремі екрани, а й повний ігровий цикл: запуск гри, створення або завантаження збереження, отримання героя, формування команди, запуск бою, отримання нагород, оновлення завдань і повторне збереження прогресу [20].

Основна увага під час тестування приділялася функціональним сценаріям, які користувач виконує найчастіше. До них належать перехід між екранами, заклик героїв, перегляд бібліотеки, додавання персонажів до формації, проходження кампанії або башти, робота бойової системи, нарахування ресурсів і зміна налаштувань. Також перевірялося, чи не втрачаються дані після перезапуску застосунку.

Тестування проводилося в середовищі Unity 6000.4.2f1. Для перевірки використовувався запуск гри в Unity Editor, а також підготовлена Windows x64-збірка. Такий підхід дозволив перевірити роботу застосунку як у режимі розробки, так і в умовах окремого запуску на персональному комп’ютері. Основні умови тестування наведено в таблиці 3.13 [12; 20].

Таблиця 3.13

Умови проведення тестування ігрового застосунку

Параметр	Значення
Середовище розробки	Unity 6000.4.2f1
Мова програмування	C#
Платформа перевірки	Unity Editor, Windows x64-збірка
Базова роздільна здатність	1920×1080 пікселів
Тип збереження	Локальний JSON-файл
Основний файл збереження	project_mebius_save.json
Основні ресурси	Локальні ресурси Unity-проєкту

Продовженнят таблиці 3.13

Умови проведення тестування ігрового застосунку

Параметр	Значення
Тип тестування	Функціональне тестування, перевірка сценаріїв, smoke-тестування

Першим етапом перевірявся запуск застосунку. Потрібно було переконатися, що після старту створюється інтерфейс, відкривається головне меню, а кнопки переходу працюють коректно. Якщо файл збереження вже існує, гра має завантажити попередній стан. Якщо файл відсутній, система повинна створити початковий набір даних, зокрема стартові ресурси, героїв і формацію.

Окремо перевірялася система заклику героїв. Під час тестування виконувалися одиничний заклик і заклик кількох героїв. Після кожної дії перевірялося, чи зменшується відповідний ресурс, чи додається новий герой до бібліотеки, чи правильно відображаються його роль, фракція, зірковість, характеристики та візуальні ознаки. Також перевірялося, чи не створюється герой із некоректними або порожніми даними.

Наступним етапом було тестування бібліотеки героїв. У цьому сценарії перевірявся список отриманих персонажів, робота сортування, фільтрації та перегляду інформації про героя. Оскільки бібліотека пов’язана із системою генерації, формацією та збереженням, важливо було переконатися, що всі отримані персонажі залишаються доступними після переходу між екранами й після перезапуску гри.

Формація перевірялася окремо. У грі передбачено п’ять позицій: дві передні та три задні. Під час тестування перевірялося, чи можна додати героя до вільної позиції, чи не додається один і той самий герой кілька разів, чи не перевищується максимальна кількість персонажів у команді. Також перевірялося автоматичне заповнення формації та видалення недоступних героїв зі складу.

Бойова система тестувалася через запуск етапів кампанії та башти. Перед боєм система повинна сформувати ворогів, підготувати бойові одиниці, застосувати характеристики героїв і ворогів, після чого провести симуляцію. Під

час тестування перевірялося, чи завершується бій перемогою або поразкою, чи формується бойовий журнал, чи не перевищується обмеження в 160 раундів і чи коректно нараховуються нагороди після перемоги.

Результати функціонального тестування наведено в таблиці 3.14.

Таблиця 3.14

Результати функціонального тестування ігрового застосунку

№	Тестовий сценарій	Очікуваний результат	Фактичний результат
1	Запуск застосунку	Відкривається головне меню, інтерфейс створюється без помилок	Виконано
2	Створення нового збереження	За відсутності файлу створюється початковий стан гри	Виконано
3	Завантаження збереження	Після перезапуску відновлюються ресурси, герої, формація та прогрес	Виконано
4	Перехід між екранами	Кнопки меню, хабу, бібліотеки, команди, бою та налаштувань працюють коректно	Виконано
5	Заклик одного героя	Створюється новий герой, ресурс витрачається, герой додається до бібліотеки	Виконано
6	Заклик кількох героїв	Створюється кілька персонажів із різними параметрами	Виконано
7	Перегляд бібліотеки	Отримані герої відображаються зі своїми характеристиками	Виконано
8	Сортування та фільтрація	Герої впорядковуються за вибраним критерієм	Виконано
9	Формування команди	До формації додається не більше 5 доступних героїв	Виконано
10	Перевірка фракційних синергій	За наявності кількох героїв однієї фракції нараховується бонус	Виконано
11	Генерація ворогів	Для етапу створюється від 3 до 5 ворогів	Виконано
12	Запуск бою	Бій проходить автоматично, формується результат і журнал подій	Виконано
13	Нарахування нагород	Після перемоги ресурси додаються до гаманця користувача	Виконано
14	Режим кампанії	Після перемоги оновлюється прогрес кампанії	Виконано

Продовження таблиці 3.14

№	Тестовий сценарій	Очікуваний результат	Фактичний результат
15	Режим башти	Після перемоги оновлюється рівень башти	Виконано
16	AFK-нагороди	Ресурси нараховуються за час відсутності, але не більше ніж за 12 годин	Виконано
17	Завдання	Після відповідних дій оновлюється прогрес завдань	Виконано
18	Налаштування	Зміна гучності, швидкості бою та візуальних параметрів зберігається	Виконано

Під час перевірки бойової системи окремо розглядалися ситуації перемоги та поразки. У разі перемоги користувач отримує нагороду, прогрес кампанії або башти оновлюється, а завдання можуть отримати нове значення прогресу. У разі поразки користувач повертається до підготовки команди. Це дозволяє перевірити, чи правильно працює зв’язок між боєм, нагородами, розвитком героїв і збереженням.

Окремий блок тестування стосувався збереження даних. Для цього після виконання кількох дій застосунок закривався і запускався повторно. Перевірялося, чи залишаються в грі отримані герої, чи зберігається кількість ресурсів, чи відновлюється формація, чи не скидаються налаштування та прогрес проходження. У результаті перевірки було підтверджено, що основні дані відновлюються після перезапуску.

Для швидкої перевірки працездатності основних механік використовується редакторський smoke-тест ProjectMebiusSmokeTest.cs. Його призначення полягає в тому, щоб швидко перевірити критичні частини гри після змін у коді або ресурсах. Такий тест не замінює повне ручне тестування, але допомагає швидко виявити помилки в основних системах.

У smoke-тесті перевіряються генерація героїв, автоматичне формування команди, генерація ворогів, масштабування складності, створення нагород, обмеження AFK-нагород, робота бойового журналу, оновлення завдань і

перевірка збереження через JSON. Якщо всі перевірки проходять успішно, у консолі Unity виводиться повідомлення про успішне завершення тесту.

Під час тестування також було перевірено роботу обмежень, визначених у попередніх розділах. Команда не повинна містити більше п’яти героїв, кількість ворогів на етапі має бути в межах від трьох до п’яти, автоматизований бій не повинен тривати більше 160 раундів, максимальна зірковість героя обмежується значенням 7, а AFK-нагороди накопичуються не більше ніж за 12 годин. Перевірка цих обмежень важлива, оскільки вони впливають на стабільність і баланс гри.

Основні перевірені обмеження наведено в таблиці 3.15.

Таблиця 3.15

Перевірка основних ігрових обмежень

Обмеження	**Очікуване значення**	**Результат перевірки**
Кількість героїв у формації	Не більше 5	Виконано
Передня лінія формації	2 позиції	Виконано
Задня лінія формації	3 позиції	Виконано
Кількість ворогів на етапі	Від 3 до 5	Виконано
Босовий етап	Кожен 10-й етап	Виконано
Максимальна тривалість бою	До 160 раундів	Виконано
Максимальна зірковість героя	7 зірок	Виконано
Максимальний рівень героя	100 рівень	Виконано
Обмеження AFK-нагород	До 12 годин	Виконано
Формат збереження	JSON	Виконано

За результатами тестування основні функціональні можливості працюють відповідно до визначених вимог. Застосунок запускається, коректно створює або завантажує збереження, дозволяє отримувати героїв, формувати команду, проходити бої, отримувати нагороди та зберігати прогрес. Виявлені під час

перевірки дрібні помилки в розміщенні інтерфейсних елементів і переходах між екранами були виправлені під час розробки.

Отже, тестування підтвердило працездатність основних модулів ігрового застосунку. Перевірені сценарії охоплюють запуск гри, інтерфейс, генерацію героїв і ворогів, формацію, бойову систему, нагороди, завдання, AFK-нарахування, налаштування та збереження даних. Це дає змогу зробити висновок, що реалізовані механіки відповідають поставленим функціональним і нефункціональним вимогам.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено ігровий застосунок «Project Mebius» в жанрі Fantasy RPG з елементами стратегії та колекціонування. У процесі виконання роботи було розглянуто особливості обраного жанру, проаналізовано існуючі ігрові рішення, визначено вимоги до застосунку, обґрунтовано вибір засобів реалізації, спроєктовано основні програмні модулі та реалізовано ключові механіки гри.

У першому розділі було проаналізовано предметну галузь ігрових застосунків жанру Fantasy RPG. Визначено, що для цього напряму характерними є розвиток персонажів, командні бої, колекціонування героїв, випадкове отримання нових одиниць, система ресурсів і поступове ускладнення проходження. Також було розглянуто існуючі ігрові застосунки, зокрема AFK Arena, RAID: Shadow Legends, Summoners War та Epic Seven. На основі аналізу встановлено, що більшість таких ігор мають схожу основу, однак відрізняються акцентами: пасивним прогресом, глибиною розвитку героїв, командною стратегією або візуальною подачею.

За результатами аналізу було сформовано функціональні та нефункціональні вимоги до розроблюваного застосунку. До основних функціональних вимог віднесено систему заклику героїв, бібліотеку персонажів, формування команди, фракційні синергії, кампанію, режим башти, автоматизований бій, нагороди, завдання, тренування, підвищення зірковості та збереження прогресу. Нефункціональні вимоги уточнено через конкретні параметри: використання Unity 6000.4.2f1, мови C#, базової роздільної здатності 1920×1080, Windows x64-запуску, JSON-збереження, обмеження формації до 5 героїв, обмеження AFK-нагород до 12 годин і максимального рівня героя до 100.

У другому розділі було обґрунтовано вибір засобів і методів реалізації. Як основне середовище розробки обрано Unity, оскільки воно дозволяє працювати з 2D-графікою, інтерфейсом, ресурсами, сценами, аудіо та програмною логікою в межах одного проєкту. Для написання основної логіки використано мову C#, яка

підходить для опису моделей даних, сервісів, генераторів, бойової системи та збереження прогресу.

Було розглянуто засоби створення інтерфейсу та візуального оформлення. У роботі використано Unity UI, Canvas, TextMeshPro, локальні графічні ресурси, спрайти героїв і ворогів, іконки, фони, аудіофайли та шрифти. Інтерфейс реалізовано українською мовою, що робить застосунок зрозумілішим для цільового користувача. Окремо описано застосування процедурної генерації, яка використовується для створення героїв, ворогів, етапів, нагород і візуальних ознак персонажів.

У третьому розділі було описано проєктування, реалізацію та тестування ігрового застосунку. Загальна архітектура побудована на поєднанні моделей даних, сервісів, головного контролера, інтерфейсу та локального збереження. Основна програмна логіка зосереджена у модулі ProjectMebiusMVP.cs, у якому реалізовано генерацію героїв і ворогів, роботу з ресурсами, формацію, фракційні бонуси, бойову систему, завдання, інтерфейс і збереження даних.

Було спроєктовано ігровий цикл, у межах якого користувач отримує героя, переглядає його в бібліотеці, додає до формації, проходить бойовий етап, отримує нагороду та використовує ресурси для подальшого розвитку. Такий цикл поєднує рольову, стратегічну та колекційну складові гри. Окремо реалізовано кампанію та башту, що дають користувачу різні способи проходження й отримання ресурсів.

У роботі реалізовано систему процедурної генерації героїв, ворогів і нагород. Герої створюються з урахуванням ролі, фракції, зірковості, характеристик, здібностей і візуальних параметрів. Вороги формуються залежно від режиму, етапу та складності, а босові противники з’являються на кожному десятому етапі. Нагороди пов’язані з результатами проходження та використовуються для розвитку персонажів і команди.

Система колекціонування дозволяє користувачу отримувати нових героїв, переглядати їх у бібліотеці, сортувати, фільтрувати та додавати до формації. Команда формується за схемою дві передні позиції та три задні позиції. Також реалізовано фракційні синергії, які надають бонуси до характеристик залежно від

кількості героїв однієї фракції в команді. Розвиток персонажів здійснюється через тренування та підвищення зірковості.

Бойова система реалізована у форматі автоматизованої симуляції. Під час бою враховуються характеристики героїв і ворогів, швидкість, атаки, критичні удари, ухилення, лікування, спеціальні здібності та командна суперздібність. Бій проходить по раундах і має обмеження до 160 раундів, що запобігає зависанню симуляції. Після перемоги користувач отримує нагороди, а прогрес кампанії або башти оновлюється.

Також реалізовано систему збереження даних у локальному JSON-файлі. У збереженні фіксуються герої, ресурси, формація, прогрес кампанії та башти, завдання, налаштування й час останнього входу. Завдяки цьому користувач може продовжити гру після повторного запуску застосунку. Час останнього входу використовується для розрахунку AFK-нагород, що доповнює основний цикл розвитку.

Проведене тестування підтвердило працездатність основних механік застосунку. Було перевірено запуск гри, створення та завантаження збереження, заклик героїв, роботу бібліотеки, формування команди, генерацію ворогів, запуск бою, нарахування нагород, роботу кампанії та башти, AFK-нагороди, завдання, налаштування та обмеження системи. Результати тестування показали, що реалізовані функції відповідають визначеним вимогам.

Практичне значення роботи полягає у створенні застосунку, який демонструє повний цикл розробки Fantasy RPG з елементами стратегії та колекціонування: від аналізу жанру й аналогів до реалізації програмних модулів і тестування. Результати можуть бути використані для подальшого розширення гри новими героями, ворогами, режимами, анімаціями та системою прогресії.

Отже, поставлену мету роботи досягнуто. Було розроблено ігровий застосунок, який поєднує процедурну генерацію, колекціонування героїв, стратегічне формування команди, автоматизовані бої, режими проходження, нагороди, розвиток персонажів, україномовний інтерфейс і локальне збереження прогресу. Основні завдання, визначені у вступі, виконано.

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

1. Гребенюк І., Золотухіна О.А. Концепція гри «Project Mebius» в жанрі Fantasy RPG з елементами стратегії та колекціонування. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.388-389.

2. Гребенюк І., Золотухіна О.А. Розробка та супровід програмного забезпечення ігрового застосунку «Project Mebius». Матеріали Міжнародної науково-практичної конференції “Міждисциплінарні наукові дослідження та перспективи їх розвитку” м. Кривий Ріг, Україна. Збірник матеріалів. (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Newzoo. Global Games Market Report 2025 : free version. Newzoo, 2025. URL: https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2025 (дата звернення: 15.05.2026).

2. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. Boca Raton : CRC Press, 2019. 652 p. DOI: https://www.taylorfrancis.com/books/mono/10.1201/b22101/art-game-design-jesse-schell (дата звернення: 15.05.2026).

3. Adams E. Fundamentals of Game Design. 3rd ed. Berkeley : New Riders, 2014. 576 p. URL: https://www.oreilly.com/library/view/fundamentals-of-game/9780133435726/ (дата звернення: 15.05.2026).

4. Shaker N., Togelius J., Nelson M. J. Procedural Content Generation in Games. Cham : Springer, 2016. DOI: 10.1007/978-3-319-42716-4 (дата звернення: 15.05.2026).

5. Hendrikx M., Meijer S., Van Der Velden J., Iosup A. Procedural Content Generation for Games: A Survey. ACM Transactions on Multimedia Computing, Communications, and Applications. 2013. Vol. 9, No. 1. P. 1–22. DOI: 10.1145/2422956.2422957 (дата звернення: 15.05.2026).

6. Farrokhi Maleki M., Zhao R. Procedural Content Generation in Games: A Survey with Insights on Emerging LLM Integration. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 2024. Vol. 20. P. 167–178. DOI: 10.1609/aiide.v20i1.31877 (дата звернення: 15.05.2026).

7. AFK Arena. Official website. Lilith Games. URL: https://afkarena.lilith.com/ (дата звернення: 15.05.2026).

8. RAID: Shadow Legends. Official website. Plarium. URL: https://raidshadowlegends.com/ (дата звернення: 15.05.2026).

9. Summoners War: Sky Arena. Official website. Com2uS. URL: https://sw.com2us.com/en/skyarena/game (дата звернення: 15.05.2026).

10. Epic Seven. Official website. Smilegate. URL: https://m.smilegate.com/en/game/epicseven.do (дата звернення: 15.05.2026).

11. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. Geneva : ISO, 2018. URL: https://www.iso.org/standard/72089.html (дата звернення: 15.05.2026).

12. Unity Technologies. Unity 6.4 User Manual. Unity Documentation. URL: https://docs.unity3d.com/6000.4/Documentation/Manual/UnityManual.html (дата звернення: 15.05.2026).

13. Unity Technologies. uGUI (Unity UI). Unity Manual 6.4. URL: https://docs.unity3d.com/6000.4/Documentation/Manual/com.unity.ugui.html (дата звернення: 15.05.2026).

14. Unity Technologies. TextMesh Pro Documentation. Unity Documentation. URL: https://docs.unity3d.com/Packages/com.unity.textmeshpro%403.2/manual/index.html (дата звернення: 15.05.2026).

15. Microsoft. C# language documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 15.05.2026).

16. Unity Technologies. Application.persistentDataPath. Unity Scripting API 6.4. URL: https://docs.unity3d.com/6000.4/Documentation/ScriptReference/Application-persistentDataPath.html (дата звернення: 15.05.2026).

17. Unity Technologies. JsonUtility. Unity Scripting API 6.4. URL: https://docs.unity3d.com/6000.4/Documentation/ScriptReference/JsonUtility.html (дата звернення: 15.05.2026).

18. Unity Technologies. Texture Import Settings window reference. Unity Manual 6.4. URL: https://docs.unity3d.com/6000.4/Documentation/Manual/class-TextureImporter.html (дата звернення: 15.05.2026).

19. Unity Technologies. Introduction to build profiles. Unity Manual 6.4. URL: https://docs.unity3d.com/6000.4/Documentation/Manual/build-profiles.html (дата звернення: 15.05.2026).

20. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering — Software testing — Part 1: General concepts. Geneva : ISO, 2022. URL: https://www.iso.org/standard/81291.html (дата звернення: 15.05.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати особливості жанру Fantasy RPG з елементами стратегії та колекціонування.
2. Провести аналіз існуючих ігрових рішень, близьких до Fantasy RPG за жанром, механіками та принципами розвитку персонажів.
3. Визначити функціональні та нефункціональні вимоги до ігрового застосунку.
4. Обґрунтувати вибір засобів реалізації програмної реалізації ігрового застосунку.
5. Спроєктувати архітектуру застосунку, ігровий цикл і основні програмні модулі.
6. Реалізувати системи процедурної генерації героїв, ворогів, етапів і нагород.
7. Реалізувати механіки колекціонування, формування команди, розвитку героїв, кампанії, башти та автоматизованого бою.
8. Реалізувати інтерфейс користувача, систему збереження даних і базові налаштування застосунку.
9. Провести тестування ігрового застосунку.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	AFK Arena	RAID: Shadow Legends	Summoners War	Epic Seven	Project Mebius
Колекціонування героїв	+	+	+	+	+
Автоматизований бій	+	-	-	+	+
Формація команди та стратегічний підбір героїв	+	+	+	+	+
Фракції / групи персонажів і командні бонуси	+	+	+	+	+
AFK-нагороди за час відсутності	+	—	—	—	+
Процедурна генерація героїв, ворогів, етапів і нагород	—	—	—	—	+
Україномовний інтерфейс	—	—	—	—	+
локальне JSON-збереження прогресу	-	-	-	-	+

4

КОНЦЕПТ ГРИ

Сетінг - темне fantasy-середовище з героями різних фракцій, магією, ворогами, босами, кампанією та баштою випробувань.

Сюжет - у світі Project Mebius користувач збирає команду героїв, які проходять бойові етапи, піднімаються Баштою Загублених Зірок і протистоять ворогам, що посилюються з кожним рівнем.

Ключові механіки

1. Заклик і колекціонування героїв;
2. Процедурна генерація героїв, ворогів, етапів і нагород;
3. Формування команди з 5 героїв: 2 передні та 3 задні позиції;
4. Фракційні бонуси для героїв однієї фракції;
5. Автоматизовані бої в кампанії та башті;
6. Розвиток героїв через рівні, зірковість і ресурси;
7. AFK-нагороди та локальне збереження прогресу.

Unique Selling Points (USP)

1. Процедурна генерація не лише героїв, а й ворогів, етапів і нагород, що робить проходження більш варіативним.
2. Україномовний інтерфейс і автономна робота без сервера з локальним JSON-збереженням прогресу.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Заклик і процедурна генерація героїв.
2. Бібліотека героїв із сортуванням та фільтрацією.
3. Формування команди та розрахунок фракційних бонусів.
4. Автоматизований бій між героями та ворогами.
5. Генерація ворогів, босів і нагород.
6. Тренування героїв, підвищення рівня та зірковості.
7. Завдання, AFK-нагороди, налаштування та локальне збереження прогресу.

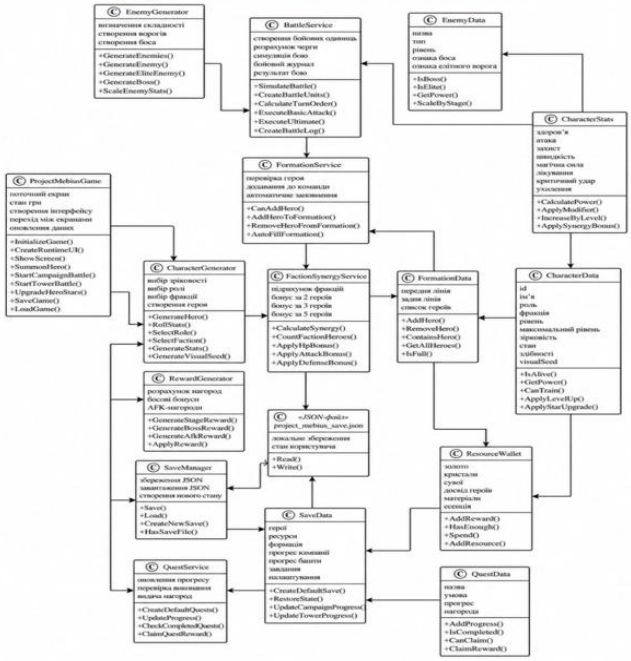
Нефункціональні вимоги:

1. Платформа запуску: Windows x64 та Unity Editor.
2. Базова роздільна здатність інтерфейсу: 1920×1080 px.
3. Максимальна зірковість героя: 7, максимальний рівень: 100.
4. Обмеження AFK-нагород: до 12 годин.
5. Формат збереження: локальний JSON-файл project_mebius_save.json.
6. Мова інтерфейсу: українська.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

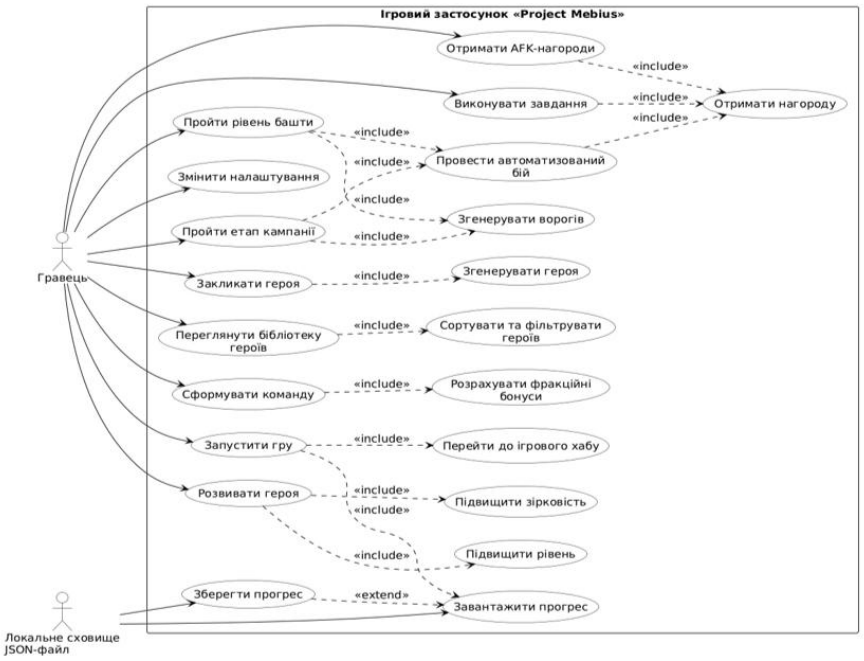
7

ДІАГРАМА КЛАСІВ



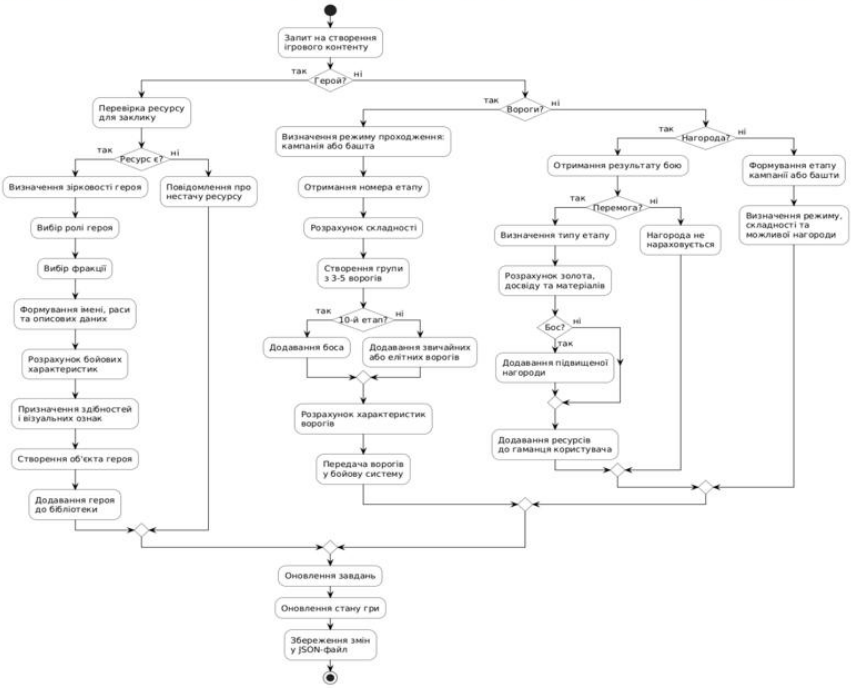
8

ДІАГРАМА ПРЕЦЕНДЕНТІВ



9

ДІАГРАМА ДІЯЛЬНОСТІ ПРОЦЕСУ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ КОНТЕНТУ



10

СТРУКТУРА ФАЙЛУ ЗБЕРЕЖЕННЯ

Файл збереження:project_mebius_save.json

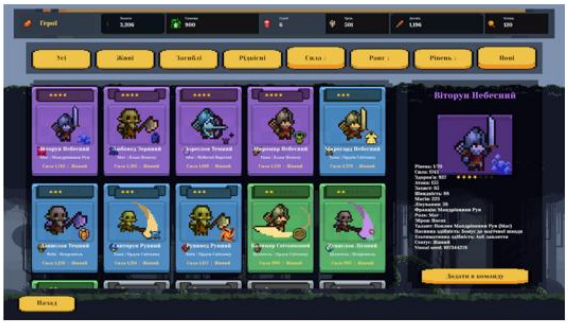
Формат: JSON

Призначення: відновлення героїв, ресурсів, формації, прогресу, завдань і налаштувань після перезапуску гри.

Поле	Тип даних	Призначення
heroes	CharacterData[]	колекція отриманих героїв
resources	ResourceWallet	ресурси користувача
formation	FormationData	поточна бойова команда
highestStage	int	найвищий пройдений етап кампанії
highestFloor	int	найвищий пройдений поверх башти
completedStages	int[]	список завершених етапів кампанії
completedFloors	int[]	список завершених поверхів башти
quests	QuestData[]	стан ігрових завдань
lastAfkClaimUtc	string	дата й час останнього AFK-нарахування у UTC
settings	SettingsData	налаштування звуку, бою та ефектів
summonCount	int	кількість виконаних закликів героїв
campaignWins	int	кількість перемог у кампанії
towerWins	int	кількість перемог у башті

11

ЕКРАННІ ФОРМИ

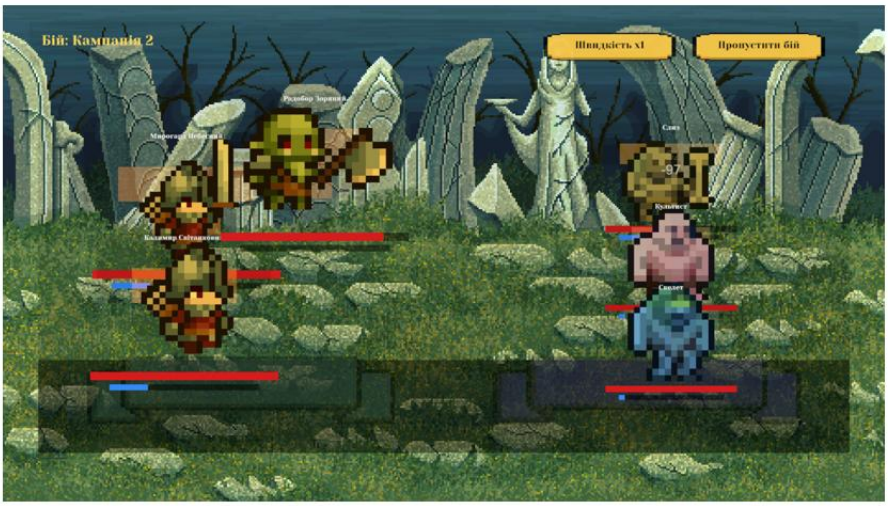


Бібліотека героїв

Вибір рівнів

12

ЕКРАННІ ФОРМИ



Сцена геймплею

13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Гребенюк І., Золотухіна О.А. Концепція гри «Project Mebius» в жанрі Fantasy RPG з елементами стратегії та колекціонування. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.388-389.

2. Гребенюк І., Золотухіна О.А. Розробка та супровід програмного забезпечення ігрового застосунку «Project Mebius». Матеріали Міжнародної науково-практичної конференції “Міждисциплінарні наукові дослідження та перспективи їх розвитку” м. Кривий Ріг, Україна. Збірник матеріалів. (подано до друку).

14

ВИСНОВКИ

1. Проаналізовано особливості жанру Fantasy RPG з елементами стратегії та колекціонування; визначено, що основою такого ігрового процесу є розвиток героїв, формування команди, проходження етапів, отримання нагород і поступове посилення персонажів.
2. Розглянуто існуючі ігрові рішення, близькі за жанром і механіками.Визначено їхні сильні сторони, обмеження та особливості, які враховано під час створення власного застосунку.
3. Сформовано функціональні та нефункціональні вимоги до ігрового застосунку, зокрема вимоги до генерації героїв, формації команди, автоматизованого бою, кампанії, башти, нагород, AFK-механіки, збереження прогресу та україномовного інтерфейсу.
4. Обґрунтовано вибір Unity 6000.4.2f1, мови C# та допоміжних засобів реалізації, які забезпечують створення 2D-гри, роботу з інтерфейсом, локальними ресурсами, програмною логікою, тестуванням і підготовкою Windows x64-збірки.
5. Спроєктовано архітектуру застосунку, ігровий цикл і основні програмні модулі, які поєднують моделі даних, генератори, сервіси, бойову систему, інтерфейс користувача та локальне збереження прогресу.
6. Реалізовано системи процедурної генерації героїв, ворогів, етапів і нагород, що забезпечує варіативність ігрового контенту.
7. Реалізовано механіки колекціонування героїв, формування команди, фракційних бонусів, розвитку персонажів, кампанії, башти та автоматизованого бою, що формує основний ігровий цикл застосунку.
8. Реалізовано україномовний інтерфейс користувача, систему збереження даних і базові налаштування застосунку.
9. Проведено перевірку основних сценаріїв та підтверджено працездатність реалізованих функцій. 15

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

Лістинг А.1 - моделі даних та структура збереження гри

Файл: Assets/Scripts/ProjectMebiusMVP.cs

```
public class CharacterStats
  {
    public int hp;
    public int attack;
    public int defense;
    public int speed;
    public float critChance;
    public float critDamage;
    public float dodgeChance;
    public float accuracy;
    public int magicPower;
    public int healingPower;
    public int power;

    public CharacterStats Clone()
    {
      return new CharacterStats
      {
        hp = hp,
        attack = attack,
        defense = defense,
        speed = speed,
        critChance = critChance,
        critDamage = critDamage,
        dodgeChance = dodgeChance,
        accuracy = accuracy,
        magicPower = magicPower,
        healingPower = healingPower,
        power = power
      };
    }
  }

public class CharacterData
  {
    public string id;
    public string heroName;
    public string gender;
    public string race;
    public string factionId;
    public string factionName;
    public string roleId;
```

```
    public string roleName;
    public int stars;
    public int level;
    public int experience;
    public int maxLevel;
    public CharacterStats stats = new CharacterStats();
    public string weaponType;
    public string talent;
    public string passiveAbility;
    public string ultimateAbility;
    public string status;
    public string createdDateUtc;
    public int visualSeed;
    public string spriteKey;
    public int bodyIndex;
    public int headIndex;
    public int hairIndex;
    public int armorIndex;
    public int weaponIndex;
    public string factionColorHex;
    public string roleColorHex;

    public bool IsAlive
    {
      get { return status == "Alive"; }
    }
  }

public class ResourceWallet
  {
    public int gold = 2500;
    public int gems = 900;
    public int scrolls = 20;
    public int training = 450;
    public int heroExp = 900;
    public int essence = 120;

    public int Get(ResourceKind kind)
    {
      switch (kind)
      {
        case ResourceKind.Gold: return gold;
        case ResourceKind.Gems: return gems;
        case ResourceKind.Scrolls: return scrolls;
        case ResourceKind.Training: return training;
```

```
            case ResourceKind.HeroExp: return heroExp;
            case ResourceKind.Essence: return essence;
            default: return 0;
        }
    }

    public void Add(RewardBundle reward)
    {
        gold += reward.gold;
        gems += reward.gems;
        scrolls += reward.scrolls;
        training += reward.training;
        heroExp += reward.heroExp;
        essence += reward.essence;
    }

    public bool Spend(ResourceKind kind, int amount)
    {
        if (amount <= 0) return true;
        if (Get(kind) < amount) return false;

        switch (kind)
        {
            case ResourceKind.Gold: gold -= amount; break;
            case ResourceKind.Gems: gems -= amount; break;
            case ResourceKind.Scrolls: scrolls -= amount; break;
            case ResourceKind.Training: training -= amount; break;
            case ResourceKind.HeroExp: heroExp -= amount; break;
            case ResourceKind.Essence: essence -= amount; break;
        }

        return true;
    }
}

public class FormationData
{
    public List<string> front = new List<string>();
    public List<string> back = new List<string>();
}

public class QuestData
{
    public string id;
    public string name;
```

```
    public string description;
    public int progress;
    public int maxProgress;
    public RewardBundle reward = new RewardBundle();
    public bool claimed;

    public bool Completed
    {
      get { return progress >= maxProgress; }
    }
  }

public class SettingsData
  {
    public float masterVolume = 0.85f;
    public float musicVolume = 0.45f;
    public float sfxVolume = 0.85f;
    public bool battleAnimations = true;
    public bool fastBattle = false;
    public bool screenShake = true;
    public bool damageNumbers = true;
    public string language = "Українська";
  }

public class SaveData
  {
    public List<CharacterData> heroes = new List<CharacterData>();
    public ResourceWallet resources = new ResourceWallet();
    public FormationData formation = new FormationData();
    public int highestStage = 0;
    public int highestFloor = 0;
    public List<int> completedStages = new List<int>();
    public List<int> completedFloors = new List<int>();
    public List<QuestData> quests = new List<QuestData>();
    public string lastAfkClaimUtc;
    public SettingsData settings = new SettingsData();
    public int summonCount;
    public int campaignWins;
    public int towerWins;
  }
```

Лістинг А.2 - генерація героя та розрахунок його характеристик

Файл: Assets/Scripts/ProjectMebiusMVP.cs

```
public class CharacterGenerator
  {
    private readonly string[] maleStarts = { "Яро", "Свято", "Миро", "Дани", "Тихо", "Любо", "Радо", "Олек", "Бори",
"Віто" };
    private readonly string[] femaleStarts = { "Зоре", "Лади", "Миро", "Яри", "Світо", "Дани", "Руни", "Астри",
"Кали", "Віри" };
    private readonly string[] ends = { "слав", "мир", "дан", "рад", "бор", "зор", "рун", "вед", "гард", "лис" };
    private readonly string[] titles = { "Зоряний", "Рунний", "Тихий", "Попелястий", "Світанковий", "Лісовий",
"Небесний", "Темний", "Мідний", "Срібний" };
    private readonly string[] races = { "Людина", "Ельф", "Дворф", "Орк", "Зоряний нащадок", "Тіньовий",
"Рунолюд" };
    private readonly System.Random random = new System.Random();

    public CharacterData GenerateHero(int? forcedStars = null)
    {
      int seed = random.Next(100000, 999999999);
      System.Random rng = new System.Random(seed);
      int stars = forcedStars.HasValue ? Mathf.Clamp(forcedStars.Value, 1, 7) : RollStars(rng);
        RoleDefinition role = ProjectMebiusDefinitions.Roles[rng.Next(ProjectMebiusDefinitions.Roles.Count)];
        FactionDefinition faction = ProjectMebiusDefinitions.Factions[rng.Next(ProjectMebiusDefinitions.Factions.Count)];
      bool male = rng.NextDouble() > 0.48f;
      string start = male ? maleStarts[rng.Next(maleStarts.Length)] : femaleStarts[rng.Next(femaleStarts.Length)];
      string end = ends[rng.Next(ends.Length)];
      string suffix = titles[rng.Next(titles.Length)];
      string heroName = start + end + " " + suffix;

      CharacterStats stats = RollStats(stars, role, rng);
      return new CharacterData
      {
        id = Guid.NewGuid().ToString("N"),
        heroName = heroName,
        gender = male ? "Чоловіча" : "Жіноча",
        race = races[rng.Next(races.Length)],
        factionId = faction.id,
        factionName = faction.name,
        roleId = role.id,
        roleName = role.name,
        stars = stars,
        level = 1,
        experience = 0,
        maxLevel = ProjectMebiusDefinitions.LevelCaps[stars],
        stats = stats,
        weaponType = role.weapon,
```

```
            talent = GenerateTalent(role, faction, rng),
            passiveAbility = role.passive,
            ultimateAbility = role.ultimate,
            status = "Alive",
            createdDateUtc = DateTime.UtcNow.ToString("o"),
            visualSeed = seed,
            spriteKey = rng.NextDouble() > 0.22 ? "soldier" : "orc",
            bodyIndex = rng.Next(0, 8),
            headIndex = rng.Next(0, 8),
            hairIndex = rng.Next(0, 10),
            armorIndex = rng.Next(0, 10),
            weaponIndex = rng.Next(0, 40),
            factionColorHex = faction.colorHex,
            roleColorHex = role.colorHex
        };
    }

    public List<CharacterData> GenerateHeroes(int count)
    {
        List<CharacterData> heroes = new List<CharacterData>();
        for (int i = 0; i < count; i++)
        {
            heroes.Add(GenerateHero());
        }
        return heroes;
    }

    private int RollStars(System.Random rng)
    {
        double roll = rng.NextDouble();
        if (roll < 0.50) return 1;
        if (roll < 0.75) return 2;
        if (roll < 0.88) return 3;
        if (roll < 0.95) return 4;
        if (roll < 0.98) return 5;
        if (roll < 0.995) return 6;
        return 7;
    }

    private CharacterStats RollStats(int stars, RoleDefinition role, System.Random rng)
    {
        float rarity = 0.78f + stars * 0.25f;
        int baseHp = Mathf.RoundToInt((260 + rng.Next(40, 120) + stars * 75) * role.hp * rarity);
        int attack = Mathf.RoundToInt((32 + rng.Next(8, 25) + stars * 13) * role.attack * rarity);
        int defense = Mathf.RoundToInt((18 + rng.Next(5, 18) + stars * 9) * role.defense * rarity);
```

```
        int speed = Mathf.RoundToInt((55 + rng.Next(0, 24) + stars * 3) * role.speed);
        int magic = Mathf.RoundToInt((18 + rng.Next(5, 20) + stars * 10) * role.magic * rarity);
        int healing = Mathf.RoundToInt((10 + rng.Next(4, 18) + stars * 8) * role.healing * rarity);

        CharacterStats stats = new CharacterStats
        {
            hp = baseHp,
            attack = attack,
            defense = defense,
            speed = speed,
            critChance = Mathf.Clamp(0.045f + stars * 0.012f + (role.id == "assassin" ? 0.12f : 0f), 0.03f, 0.38f),
            critDamage = 1.45f + stars * 0.05f + (role.id == "assassin" ? 0.25f : 0f),
            dodgeChance = Mathf.Clamp(0.025f + speed / 2200f + (role.id == "assassin" ? 0.05f : 0f), 0.01f, 0.28f),
            accuracy = Mathf.Clamp(0.82f + stars * 0.018f, 0.75f, 0.98f),
            magicPower = magic,
            healingPower = healing
        };
        stats.power = CalculatePower(stats);
        return stats;
    }

    public static int CalculatePower(CharacterStats stats)
    {
        return Mathf.RoundToInt(stats.hp * 0.18f + stats.attack * 3.2f + stats.defense * 2.4f + stats.speed * 1.7f +
stats.magicPower * 2.6f + stats.healingPower * 2.8f + stats.critChance * 280f + stats.dodgeChance * 220f);
    }

    private string GenerateTalent(RoleDefinition role, FactionDefinition faction, System.Random rng)
    {
        string[] talentWords = { "Резонанс", "Пам’ять", "Іскра", "Оберіг", "Поклик", "Печатка", "Передчуття",
"Полум’я" };
        return talentWords[rng.Next(talentWords.Length)] + " " + faction.name + " (" + role.name + ")";
    }
  }
```

Лістинг А.3 - генерація ворогів, бойових етапів і нагород

Файл: Assets/Scripts/ProjectMebiusMVP.cs

```
public class EnemyGenerator
  {
    private readonly string[] typeIds = { "slime", "skeleton", "goblin", "cultist", "dark_knight", "demon", "tower_beast",
"guardian" };
    private readonly string[] typeNames = { "Слиз", "Скелет", "Гоблін", "Культист", "Темний лицар", "Демон",
"Баштова потвора", "Вартовий" };
    private readonly string[] sprites =
```

```
    {
        "Enemies/DeathSlime", "Enemies/BlindedGrimlock", "Enemies/RedCap", "Enemies/OcularWatcher",
        "Enemies/StoneTroll", "Enemies/CrimsonSlaad", "Enemies/BrawnyOgre", "Enemies/CrushingCyclops",
        "Enemies/FungalMyconid", "Enemies/HumongousEttin", "Enemies/MurkySlaad", "Enemies/OchreJelly",
        "Enemies/ShriekerMushroom", "Enemies/SwampTroll", "Enemies/BloodshotEye"
    };
    private readonly System.Random random = new System.Random();

    public List<EnemyData> GenerateEncounter(int stage, int teamPower, bool tower)
    {
        List<EnemyData> enemies = new List<EnemyData>();
        int floorStage = Mathf.Max(1, stage);
        bool bossStage = floorStage % 10 == 0;
        int count = bossStage ? 5 : Mathf.Clamp(3 + floorStage / 10, 3, 5);

        for (int i = 0; i < count; i++)
        {
            bool boss = bossStage && i == count - 1;
            bool elite = !boss && ((floorStage % 5 == 0 && i == 0) || (floorStage >= 12 && floorStage % 3 == 0 && i == count -
1));
            enemies.Add(GenerateEnemy(floorStage, teamPower, tower, boss, elite));
        }

        return enemies;
    }

    private EnemyData GenerateEnemy(int stage, int teamPower, bool tower, bool boss, bool elite)
    {
        int seed = random.Next(1000, 99999999);
        System.Random rng = new System.Random(seed + stage * 17);
        int idx = boss ? 7 + rng.Next(0, 8) : rng.Next(typeIds.Length);
            RoleDefinition role = ProjectMebiusDefinitions.Roles[rng.Next(ProjectMebiusDefinitions.Roles.Count)];
        float chapterScale = 1f + Mathf.FloorToInt((stage - 1) / 10f) * 0.24f;
        float modeScale = tower ? 1.28f : 1f;
        float stageCurve = 0.82f + stage * 0.092f + Mathf.Pow(stage, 1.12f) * 0.012f;
        float teamPressure = Mathf.Clamp(teamPower / (6200f + stage * 900f), 0.10f, 2.15f);
        float scale = stageCurve * chapterScale * modeScale * (1f + teamPressure * 0.22f);
        if (elite) scale *= 1.45f;
        if (boss) scale *= 3.05f + Mathf.Min(0.65f, stage * 0.012f);

        int level = Mathf.Clamp(1 + stage + (tower ? stage / 3 : 0), 1, 140);
        CharacterStats stats = new CharacterStats
        {
            hp = Mathf.RoundToInt((195 + level * 36 + rng.Next(0, 110)) * role.hp * scale),
            attack = Mathf.RoundToInt((28 + level * 6 + rng.Next(0, 24)) * role.attack * scale),
```

```
            defense = Mathf.RoundToInt((15 + level * 3.4f + rng.Next(0, 14)) * role.defense * scale),
            speed = Mathf.RoundToInt((42 + rng.Next(0, 24) + level * 1.0f) * role.speed * (tower ? 1.04f : 1f)),
            critChance = Mathf.Clamp(0.035f + level * 0.0028f + (elite ? 0.025f : 0f) + (boss ? 0.07f : 0f), 0.02f, 0.36f),
            critDamage = boss ? 1.95f : (elite ? 1.62f : 1.48f),
            dodgeChance = Mathf.Clamp(0.016f + level * 0.0017f + (elite ? 0.015f : 0f), 0.01f, 0.26f),
            accuracy = Mathf.Clamp(0.77f + level * 0.0022f + (boss ? 0.035f : 0f), 0.70f, 0.99f),
            magicPower = Mathf.RoundToInt((11 + level * 4.8f) * role.magic * scale),
            healingPower = Mathf.RoundToInt((6 + level * 2.4f) * role.healing * scale)
        };
        stats.power = CharacterGenerator.CalculatePower(stats);

        string name = boss ? "Бос " + BossTitle(stage, rng) : typeNames[Mathf.Clamp(idx, 0, typeNames.Length - 1)];
        if (elite && !boss) name = "Елітний " + name;

        return new EnemyData
        {
            id = Guid.NewGuid().ToString("N"),
            enemyName = name,
            typeId = boss ? "boss" : typeIds[Mathf.Clamp(idx, 0, typeIds.Length - 1)],
            typeName = boss ? "Бос" : typeNames[Mathf.Clamp(idx, 0, typeNames.Length - 1)],
            roleId = role.id,
            roleName = role.name,
            level = level,
            stats = stats,
            elite = elite,
            boss = boss,
            spriteKey = sprites[Mathf.Abs(seed) % sprites.Length],
            visualSeed = seed,
            colorHex = boss ? "#FF4668" : EnemyColor(seed),
            scale = boss ? 1.35f : (elite ? 1.12f : 1f)
        };
    }

    private string BossTitle(int stage, System.Random rng)
    {
        string[] names = { "Зорежер", "Пожирач Рун", "Серце Безодні", "Князь Попелу", "Вартовий Порожнечі", "Око
Башти" };
        return names[(stage / 10 + rng.Next(names.Length)) % names.Length];
    }

    private string EnemyColor(int seed)
    {
        string[] colors = { "#B3F06A", "#BFC7D5", "#D58B4E", "#AD7DFF", "#77C7FF", "#FF7A66", "#E6E17B" };
        return colors[Mathf.Abs(seed) % colors.Length];
    }
```

```
  }

public class RewardGenerator
  {
    private readonly System.Random random = new System.Random();

    public RewardBundle StageReward(int stage, bool tower, bool boss)
    {
      int chapter = Mathf.Max(1, Mathf.CeilToInt(stage / 10f));
      float mode = tower ? 1.2f : 1f;
      RewardBundle reward = new RewardBundle
      {
        gold = Mathf.RoundToInt((85 + stage * 32 + chapter * 45) * mode),
        heroExp = Mathf.RoundToInt((45 + stage * 15 + chapter * 20) * mode),
        training = Mathf.RoundToInt((18 + stage * 5) * mode),
        essence = boss ? 35 + chapter * 12 : (random.NextDouble() < 0.18 ? 5 + chapter : 0),
        scrolls = boss ? 1 : (random.NextDouble() < 0.08 ? 1 : 0),
        gems = boss ? 45 + chapter * 8 : (random.NextDouble() < 0.06 ? 12 : 0),
        label = boss ? "Особлива нагорода боса" : "Нагорода етапу"
      };
      return reward;
    }

    public RewardBundle AfkReward(int highestStage, int highestFloor, TimeSpan elapsed)
    {
      double hours = Math.Min(12.0, Math.Max(0.0, elapsed.TotalHours));
      int progress = Mathf.Max(1, highestStage) + Mathf.Max(0, highestFloor / 2);
      return new RewardBundle
      {
        gold = Mathf.RoundToInt((70 + progress * 18) * (float)hours),
        heroExp = Mathf.RoundToInt((28 + progress * 7) * (float)hours),
        training = Mathf.RoundToInt((8 + progress * 2.2f) * (float)hours),
        essence = Mathf.RoundToInt((1 + progress * 0.18f) * (float)hours),
        scrolls = hours >= 6 ? 1 : 0,
        gems = hours >= 10 ? 25 : 0,
        label = "AFK-нагороди"
      };
    }
  }
```

Лістинг А.4 - формування команди та фракційні синергії

Файл: Assets/Scripts/ProjectMebiusMVP.cs

```
public class FactionSynergyService
  {
    public SynergyResult Calculate(IEnumerable<CharacterData> heroes)
    {
      SynergyResult result = new SynergyResult();
      Dictionary<string, int> counts = new Dictionary<string, int>();
      foreach (CharacterData hero in heroes)
      {
        if (hero == null || !hero.IsAlive) continue;
        if (!counts.ContainsKey(hero.factionId)) counts[hero.factionId] = 0;
        counts[hero.factionId]++;
      }

      foreach (KeyValuePair<string, int> pair in counts)
      {
        FactionDefinition faction = ProjectMebiusDefinitions.Faction(pair.Key);
        if (pair.Value >= 2)
        {
          result.hpMultiplier += 0.05f;
          result.lines.Add(faction.name + " x2: +5% здоров'я");
        }

        if (pair.Value >= 3)
        {
          if (faction.attackAtThree)
          {
            result.attackMultiplier += 0.10f;
            result.lines.Add(faction.name + " x3: +10% атаки");
          }
          else
          {
            result.defenseMultiplier += 0.10f;
            result.lines.Add(faction.name + " x3: +10% захисту");
          }
        }

        if (pair.Value >= 5)
        {
          result.hpMultiplier += 0.08f;
          result.attackMultiplier += 0.08f;
          result.defenseMultiplier += 0.08f;
          result.energyBonus += 10;
          result.lines.Add(faction.name + " x5: " + faction.fiveHeroBuff);
```

```
            }
        }

        return result;
    }
}

public class SynergyResult
{
    public float hpMultiplier;
    public float attackMultiplier;
    public float defenseMultiplier;
    public int energyBonus;
    public List<string> lines = new List<string>();

    public string Summary()
    {
        return lines.Count == 0 ? "Синергія: немає активних бонусів" : string.Join("\n", lines);
    }
}

public class FormationService
{
    public List<CharacterData> GetTeam(SaveData save)
    {
        List<CharacterData> team = new List<CharacterData>();
        foreach (string id in save.formation.front.Concat(save.formation.back))
        {
            CharacterData hero = save.heroes.FirstOrDefault(h => h.id == id);
            if (hero != null && hero.IsAlive) team.Add(hero);
        }
        return team;
    }

    public void Sanitize(SaveData save)
    {
        save.formation.front = save.formation.front.Where(id => save.heroes.Any(h => h.id == id &&
h.IsAlive)).Distinct().Take(2).ToList();
        save.formation.back = save.formation.back.Where(id => save.heroes.Any(h => h.id == id && h.IsAlive)
&& !save.formation.front.Contains(id)).Distinct().Take(3).ToList();
    }

    public bool AddHero(SaveData save, CharacterData hero)
    {
        if (hero == null || !hero.IsAlive) return false;
```

```
        Sanitize(save);
        RemoveHero(save, hero.id);
            RoleDefinition role = ProjectMebiusDefinitions.Role(hero.roleId);
        if ((role.preferredLine == "front" || role.preferredLine == "flex") && save.formation.front.Count < 2)
        {
            save.formation.front.Add(hero.id);
            return true;
        }

        if (save.formation.back.Count < 3)
        {
            save.formation.back.Add(hero.id);
            return true;
        }

        if (save.formation.front.Count < 2)
        {
            save.formation.front.Add(hero.id);
            return true;
        }

        return false;
    }

    public void RemoveHero(SaveData save, string id)
    {
        save.formation.front.RemoveAll(x => x == id);
        save.formation.back.RemoveAll(x => x == id);
    }

    public void AutoFill(SaveData save)
    {
        save.formation.front.Clear();
        save.formation.back.Clear();
        List<CharacterData> alive = save.heroes.Where(h => h.IsAlive).OrderByDescending(h => h.stats.power).ToList();
        foreach (CharacterData hero in alive)
        {
            if (save.formation.front.Count + save.formation.back.Count >= 5) break;
            AddHero(save, hero);
        }
    }
}
```

Лістинг А.5 - локальне JSON-збереження прогресу користувача

Файл: Assets/Scripts/ProjectMebiusMVP.cs

```
public class SaveManager
  {
    private string SavePath
    {
        get { return Path.Combine(Application.persistentDataPath, "project_mebius_save.json"); }
    }

    public bool HasSave()
    {
      return File.Exists(SavePath);
    }

    public SaveData Load()
    {
      if (!File.Exists(SavePath)) return null;
      try
      {
        return JsonUtility.FromJson<SaveData>(File.ReadAllText(SavePath));
      }
      catch (Exception ex)
      {
        Debug.LogWarning("Save load failed: " + ex.Message);
        return null;
      }
    }

    public void Save(SaveData data)
    {
      try
      {
        File.WriteAllText(SavePath, JsonUtility.ToJson(data, true));
      }
      catch (Exception ex)
      {
        Debug.LogWarning("Save failed: " + ex.Message);
      }
    }

    public void Reset()
    {
      if (File.Exists(SavePath)) File.Delete(SavePath);
    }
  }
```