**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ІГРОВОГО ЗАСТОСУНКУ НА БАЗІ UNITY З МОЖЛИВІСТЮ РОЗГОРТАННЯ ТА СУПРОВОДУ НА PLAY МАРКЕТІ»**

Виконав: студент 4 курсу, групи 6.1212-пі-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
С.С. Пічаєвський
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

**ЗАТВЕРДЖУЮ**
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Пічаєвському Сергію Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка мобільного ігрового застосунку на базі Unity з можливістю розгортання та супроводу на Play Маркеті

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та вибір засобів розробки.
2. Проєктування архітектури ігрового застосунку.
3. Практична реалізація мобільного застосунку.
4. Тестування, розгортання та супровід на Play Маркеті.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 09.12.2025 р.

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 09.01.2026 | |
| | | | |
| 2. | Збір вихідних даних. | 16.01.2026 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 13.02.2026 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 03.04.2026 | |
| | | | |
| 5. | Розробка третього та четвертого розділу. | 18.05.2026 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 25.05.2026 | |
| | кваліфікаційної роботи бакалавра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 15.06.2026 | |

Студент _____ С.С. Пічаєвський
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

## Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка мобільного ігрового застосунку на базі Unity з можливістю розгортання та супроводу на Play Маркеті»: 69 с., 5 рис., 1 табл., 10 джерел, 2 додатки.

ВІЗУАЛЬНА НОВЕЛА, ЕМУЛЯЦІЯ ОС, ІГРОВИЙ РУШІЙ, МОБІЛЬНИЙ ЗАСТОСУНОК, ПАТЕРНИ ПРОЄКТУВАННЯ, РОЗГОРТАННЯ ЗАСТОСУНКУ, СУПРОВІД ПЗ, ФАЙЛОВА СИСТЕМА, GOOGLE PLAY, UNITY.

Об’єкт дослідження – процес розробки мобільних ігрових застосунків з нестандартними мета-механіками на базі ігрового рушія Unity.

Мета роботи: проєктування, розробка та виведення на ринок мобільного геймдеву масштабованого ігрового застосунку в жанрі візуальної новели з інтегрованими метамеханіками (емуляція ОС) на базі принципів SOLID, як готового до комерційного релізу та супроводу продукту в Google Play.

Методи дослідження – об’єктно-орієнтований аналіз та проєктування для побудови архітектури додатку, метод моделювання для створення абстрактної моделі файлової системи та процесів ОС всередині гри, а також емпіричні методи для тестування розроблених модулів та взаємодії користувача з інтерфейсом.

У результаті роботи спроєктовано та реалізовано мобільний ігровий застосунок із розширеною архітектурою на базі принципів SOLID, що забезпечує гнучке масштабування та повне декуплювання компонентів систем. Шляхом об’єктно-орієнтованого моделювання успішно реалізовано ізольовану файлову систему та механізми емуляції операційної системи як ключові метамеханіки ігрового процесу. Надалі передбачається інтеграція додаткових сюжетних модулів, розширення пулу інструментів внутрішньоігрової ОС, а також покращення продуктивності рендерингу графічного інтерфейсу та оптимізація споживання пам’яті пристрою.

# SUMMARY

Bachelor's qualifying paper "Development of a Mobile Gaming Application Based on Unity with the Possibility of Deployment and Maintenance on the Play Market": 69 pages, 5 figures, 1 table, 10 references, 2 supplements.

VISUAL NOVEL, OS EMULATION, GAME ENGINE, MOBILE APPLICATION, DESIGN PATTERNS, APPLICATION DEPLOYMENT, SOFTWARE MAINTENANCE, FILE SYSTEM, GOOGLE PLAY, UNITY.

Research object is the process of developing mobile game applications with non-standard meta-mechanics based on the Unity game engine.

The aim of the work is to design, develop, and launch onto the mobile gamedev market a scalable game application in the visual novel genre with integrated meta-mechanics (OS emulation) based on SOLID principles, as a product ready for commercial release and support in Google Play.

Research methods include object-oriented analysis and design for building the application architecture, the modeling method for creating an abstract model of the file system and OS processes within the game, as well as empirical methods for testing the developed modules and user interface interaction.

As a result of the work, a mobile game application with an extended architecture based on SOLID principles was designed and implemented, ensuring flexible scaling and complete decoupling of system components. Through object-oriented modeling, an isolated file system and operating system emulation mechanisms were successfully implemented as key gameplay meta-mechanics. Future plans include the integration of additional story modules, expanding the toolset of the in-game OS, as well as improving graphical user interface rendering performance and optimizing device memory consumption.

**ЗМІСТ**

Завдання на кваліфікаційну роботу……..2
Реферат ..... 4
Summary ..... 5
Вступ..... 8
1 Аналіз предметної області та вибір засобів розробки..... 11
1.1 Аналіз ринку мобільних ігор та особливостей жанру візуальних новел..... 11
1.2 Дослідження концепції мета-ігор та імітації операційних систем у геймдизайні..... 12
1.3 Огляд ігрового рушія Unity та екосистеми C# для мобільної розробки..... 14
1.4 Формування вимог до застосунку та застосування принципів SOLID у проєктуванні ігор ..... 16
2 Проєктування архітектури ігрового застосунку ..... 21
2.1 Проєктування глобальної шини подій (EventBus) для забезпечення слабкої зв’язності компонентів ..... 21
2.2 Застосування патерна Command та системи перехоплювачів (Interceptors) для логіки мета-шару..... 23
2.3 Проєктування абстракції файлової системи та управління додатками емульованої ОС..... 24
2.4 Архітектура збереження стану гри (SaveManager) на основі інтерфейсу ISaveable..... 26
2.5 Розробка структури обробки діалогів та управління ігровими подіями (Facts)..... 27
3 Практична реалізація мобільного застосунку ..... 29
3.1 Реалізація візуального інтерфейсу (UI) емульованої ОС та її додатків ..... 29

3.2 Розробка системи візуалізації персонажів (VNCharacterManager) 32
3.3 Інтеграція підсистеми аудіосупроводу (AudioManager) ..... 34
3.4 Інтеграція налаштувань гри (SettingsManager) ..... 35
4 Тестування, розгортання та супровід на платформі Google Play ..... 38
4.1 Стратегія локального тестування та підготовка релізної збірки застосунку ..... 38
4.2 Налаштування середовища Google Play Console та підготовка метаданих ..... 39
4.3 Пайплайн розгортання: від внутрішнього тестування до відкритого релізу ..... 40
Висновки ..... 42
Перелік посилань ..... 45
Додаток А Фрагменти вихідного коду базових архітектурних модулів застосунку ..... 46
А.1 Реалізація глобальної шини подій (EventBus) ..... 46
А.2 Реалізація обробника команд (CommandProcessor) та системи перехоплювачів ..... 47
А.3 Реалізація абстрактної моделі файлової системи емульованої ОС ... 49
А.4 Архітектура збереження стану гри (SaveManager та ISaveable) ..... 52
Додаток Б Візуальне представлення графічного інтерфейсу застосунку ..... 56
Б.1 Програмна реалізація системи управління візуальним інтерфейсом (UI) ..... 56
Б.2 Програмна реалізація системи візуалізації персонажів візуальної новели ..... 59
Б.3 Програмна реалізація підсистеми аудіосупроводу ..... 62
Б.4 Програмна реалізація підсистеми управління налаштуваннями гри 67

# ВСТУП

Сьогодні індустрія мобільних ігор розвивається настільки стрімко, що гравців вже важко здивувати класичними підходами. Жанр візуальних новел, який завжди тримався на сильному сюжеті, зараз потребує інновацій для утримання аудиторії. Саме тому розробники все частіше звертаються до мета-механік – наприклад, імітації інтерфейсу справжнього смартфона чи операційної системи безпосередньо всередині гри. Це набагато цікавіше та зручніше для занурення в історію, адже гравець отримує унікальний досвід взаємодії з «телефоном у телефоні»: читає чужі повідомлення, шукає приховані файли та реагує на ігрові сповіщення так, ніби це відбувається в реальному житті. Зростаючий попит на такі нестандартні та глибоко імерсивні проєкти обумовлює високу актуальність розробки нашого застосунку.

**Мета роботи** полягає у проєктуванні та розробці масштабованого мобільного ігрового застосунку в жанрі візуальної новели з інтегрованими мета-механіками (емуляція ОС) з використанням принципів SOLID для подальшого розгортання та супроводу на платформі Google Play.

Досягнення поставленої мети передбачає послідовну реалізацію наступних **завдань**:

- проаналізувати предметну область та існуючі аналоги на ринку мобільних ігор;
- дослідити інструментарій розробки та обґрунтувати вибір ігрового рушія Unity;
- спроєктувати гнучку та масштабовану архітектуру застосунку на основі глобальної шини подій;
- розробити абстрактну модель файлової системи для емульованої оперативної системи всередині гри;
- реалізувати надійну систему збереження та завантаження поточного стану гри;

– протестувати готові модулі, оцінити взаємодію користувача з інтерфейсом та розгорнути застосунок на платформі Google Play.

**Об’єкт дослідження** – процес розробки мобільних ігрових застосунків з нестандартними мета-механіками на базі ігрового рушія Unity.

**Предмет дослідження** – методи, засоби та патерни проєктування (EventBus, Command, State), що забезпечують масштабованість, слабку зв’язність компонентів та імітацію середовища операційної системи в межах мобільного застосунку.

**Методи дослідження** включають об’єктно-орієнтований аналіз та проєктування для побудови архітектури додатку, метод моделювання для створення абстрактної моделі файлової системи та процесів ОС всередині гри, а також емпіричні методи для тестування розроблених модулів та взаємодії користувача з інтерфейсом.

Для досягнення необхідного технічного результату було обрано ігровий рушій Unity та мову програмування C#. Такий вибір зумовлений тим, що Unity надає потужні вбудовані інструменти для роботи з багатошаровим UI, а C# дозволяє реалізувати строгу об’єктноорієнтовану архітектуру за принципами SOLID. Щоб забезпечити незалежність мікро-додатків усередині гри, замість монолітного коду було використано архітектурні патерни Command та кастомну шину подій EventBus. Для збереження складного стану файлової системи та прогресу гравця використовується серіалізація даних у формат JSON. Завдяки такому підходу отримано не просто гру, а гнучку систему, яку легко підтримувати, масштабувати та оновлювати в майбутньому.

Практичне значення роботи полягає в тому, що розроблена архітектура може бути використана як надійний шаблон для створення інших мобільних застосунків зі складною внутрішньою логікою.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку посилань та 2 додатків. Загальний обсяг роботи становить 69 сторінок. Додатки викладено на 24 сторінках. У переліку посилань 10 використаних джерел.

Перший розділ присвячений аналізу предметної області, розгляду концепції мета-ігор та вибору інструментарію розробки. У другому розділі наведено детальне проєктування архітектури ігрового застосунку, проілюстровано роботу шини подій та системи перехоплювачів команд. В третьому розділі описано безпосередню практичну реалізацію інтерфейсу віртуальної ОС, діалогової системи та логіки збереження прогресу. Четвертий розділ містить інформацію про тестування застосунку, його збірку та етапи розгортання в маркетплейсі Google Play. В додатках наведено фрагменти програмної реалізації ключових модулів системи.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

## 1.1 Аналіз ринку мобільних ігор та особливостей жанру візуальних новел

Сучасний ринок мобільних додатків є найпотужнішим сегментом світової ігрової індустрії. Завдяки повсюдному поширенню смартфонів та постійному зростанню їхньої обчислювальної потужності, мобільні ігри давно вийшли за межі примітивних таймкілерів. Сьогодні користувачі очікують від мобільних проєктів глибокого ігрового досвіду, складної драматургії та якісного візуального виконання, які раніше були доступні лише на персональних комп'ютерах або спеціалізованих консолях. У цьому контексті жанр візуальних новел (Visual Novels) переживає справжній ренесанс, адаптуючись до нових реалій споживання контенту.

Візуальна новела – це жанр відеоігор, у якому ігровий процес здебільшого будується навколо текстової історії, доповненої статичними або анімованими зображеннями, звуковим супроводом та розгалуженою системою виборів. Історично цей жанр зародився і набув найбільшої популярності в Японії, проте в останні роки він став абсолютно глобальним феноменом. Сучасна аудиторія Google Play демонструє стабільний попит на сюжетно орієнтовані ігри, оскільки формат мобільного пристрою ідеально підходить для споживання такого контенту: телефон завжди під рукою, взаємодія з екраном за допомогою свайпів є інтуїтивно зрозумілою, а читання тексту з екрана смартфона є звичною повсякденною дією.

Разом із тим, зростання популярності жанру призвело до перенасичення ринку. Щодня в магазинах додатків з'являються десятки нових новел, більшість з яких використовують абсолютно ідентичні механіки: послідовне натискання на екран для прокручування тексту та періодичний вибір одного з кількох варіантів відповіді. Стандартні лінійні або навіть розгалужені наративні структури більше не здатні самостійно гарантувати успіх проєкту.

Користувачі стають більш вибагливими: їм потрібен вищий рівень імерсивності, залученості та інтерактивності. Щоб виділитися серед конкурентів, розробникам необхідно шукати нові шляхи взаємодії з гравцем, виходити за рамки класичного «читання з картинками» та впроваджувати інноваційні ігрові механіки, які органічно доповнюватимуть сюжет.

Крім того, аналіз ринку виявляє ще одну суттєву проблему – низьку технічну якість багатьох сучасних мобільних проєктів. У гонитві за швидким прибутком багато студій та незалежних розробників обирають шлях найшвидшого релізу на шкоду якості коду. Використання монолітної архітектури, жорстка зв'язність компонентів (tight coupling) та ігнорування базових принципів проєктування призводять до того, що такі додатки стає неможливо підтримувати, розширювати чи ефективно оновлювати після виходу в реліз.

У довгостроковій перспективі на платформі Google Play виграють ті проєкти, де інженерна якість ставиться вище за швидкість розробки. Створення чистого, масштабованого та легко читабельного коду на основі принципів SOLID є не просто правилом хорошого тону, а бізнес-необхідністю. Тільки гнучка архітектура дозволяє безболісно інтегрувати ті самі інноваційні механіки – наприклад, складну імітацію внутрішньоігрової операційної системи, – гарантуючи при цьому стабільну роботу додатку на тисячах різних моделей Android-пристроїв. Отже, сучасна візуальна новела має бути не лише захоплюючою історією, але й бездоганно спроєктованим програмним продуктом.

## 1.2 Дослідження концепції мета-ігор та імітації операційних систем у геймдизайні

Поняття «мета-гра» (meta-game) у сучасному геймдизайні описує проєкти, ігровий процес яких виходить за межі традиційного віртуального простору, змушуючи гравця взаємодіяти з грою на вищому рівні – через

руйнування так званої «четвертої стіни» (fourth wall breaking) та інтеграцію елементів реального світу або суміжних програмних систем. Одним із найяскравіших прикладів успішної реалізації такого мета-наративу в індустрії є проєкти на кшталт “Doki Doki Literature Club!”, де критично важливою частиною геймплею та сюжетного прогресу є маніпуляція файлами у кореневій папці гри на персональному комп’ютері користувача. Подібні ARG-механіки (Alternate Reality Game) створюють глибокий ефект іммерсивності, перетворюючи базові технічні дії користувача на інструмент розв’язання наративних та логічних загадок.

Проте спроба прямого перенесення такого досвіду на мобільні платформи (в екосистему Android чи iOS) стикається із серйозними бар’єрами в контексті користувацького досвіду (UX). На відміну від десктопних систем, де робота з файловим менеджером є звичною та швидкою, мобільні операційні системи мають більш закриту архітектуру, а навігація через сторонні системні провідники є громіздкою, неінтуїтивною для масового гравця і, що найголовніше, повністю руйнує ігрову атмосферу. Користувачі мобільних пристроїв зазвичай не мають мотивації залишати ігровий застосунок задля ручного пошуку прихованих файлів у реальній пам’яті смартфона, що робить класичний десктопний підхід неефективним у мобільному сегменті.

Для вирішення цієї суперечності в сучасному геймдизайні застосовується концепція імітації або повної емуляції операційної системи безпосередньо всередині ігрового рушія. Це дозволяє реалізувати повноцінний дієгетичний інтерфейс (diegetic interface) – елемент UI, який не просто слугує технічною оболонкою, а є невіддільною частиною художнього, сюжетного та фізичного всесвіту гри. У контексті розроблюваного проєкту емоційне сприйняття та взаємодія з інтерфейсом віртуальної ОС підсилюється наративним обґрунтуванням: система позиціонується як таємничий цифровий артефакт, до повного дослідження та розгадки якого в процесі розвитку сюжету має прийти головний герой.

Впровадження внутрішньоігрової ОС дозволяє реалізувати складні мета-механіки в межах єдиного безшовного простору застосунку. Основними

функціональними компонентами (мікро-додатками) такої системи, що формують геймплей, виступають:

- *провідник (файлова система)*: оскільки робота з файлами є базою для інтеграції ARG-елементів, внутрішній провідник стає головним інструментом геймплею – гравець отримує легкий, швидкий і контрольований доступ до пошуку «секретів», зашифрованих документів, логів та прихованих системних файлів, що безпосередньо впливають на хід розслідування та сюжет, не залишаючи при цьому гру;
- *модуль візуальної новели*: традиційний текстово-наративний шар, який виступає основним драйвером сюжету і взаємодіє з операційною системою через генерацію подій та архітектурних тригерів;
- *внутрішньоігровий магазин*: елемент мета-ігри, що забезпечує ігрову прогресію, дозволяючи купувати або відкривати нові утиліти, розширення та можливості для глибшого аналізу віртуального артефакту.

Таким чином, імітація операційної системи у геймдизайні мобільних застосунків повністю вирішує проблему взаємодії користувача з файловою структурою. Вона робить складні мета-текстуальні та ARG-механіки динамічними та доступними, зберігаючи високу залученість гравця і перетворюючи інтерфейс із суто утилітарного інструменту на центральний елемент геймплею та наративу.

## 1.3 Огляд ігрового рушія Unity та екосистеми C# для мобільної розробки

Вибір інструментарію для створення сучасного мобільного ігрового застосунку є критичним етапом, що визначає не лише швидкість розробки, але

й подальшу життєздатність, продуктивність та можливості масштабування проєкту. На сьогоднішній день Unity є одним із найпопулярніших та найпотужніших кросплатформних ігрових рушіїв на ринку, який де-факто став стандартом для мобільної розробки (Android та iOS).

Головною перевагою Unity в контексті розробки візуальної новели з елементами емуляції операційної системи є його розвинена підсистема для створення користувацьких інтерфейсів (Unity UI / uGUI). Оскільки ігровий процес базується на взаємодії з віртуальними вікнами, іконками, текстовими панелями та системними сповіщеннями, рушій надає гнучкі інструменти (Canvas, RectTransform, CanvasGroup) для побудови багатошарових, адаптивних та оптимізованих інтерфейсів. Крім того, компонентно-орієнтована архітектура Unity (Entity-Component-System у своєму базовому наближенні GameObject-Component) дозволяє зручно розділяти візуальну частину застосунку (View) від його логіки.

Фундаментом для реалізації програмної логіки в екосистемі Unity виступає мова програмування C#. Це сучасна, строго типізована, об’єктно-орієнтована мова, розроблена компанією Microsoft, яка ідеально підходить для створення складних архітектурних рішень. На відміну від скриптових мов, що часто використовуються в спеціалізованих рушіях для візуальних новел (наприклад, Ren’Py), C# надає розробнику повний контроль над пам’яттю, типізацією та життєвим циклом об’єктів.

Екосистема C# забезпечує розробника інструментарієм, необхідним для побудови чистої архітектури (Clean Architecture) [1]. До ключових переваг використання C# у мобільній розробці належать:

– *підтримка високорівневих абстракцій*: використання інтерфейсів (interfaces), абстрактних класів та узагальнень (generics) дозволяє проєктувати гнучкі системи, які легко розширювати без модифікації існуючого коду [3];
– *подієво-орієнтована модель*: вбудована підтримка делегатів та подій (events) робить C# ідеальною мовою для реалізації патернів, таких як

EventBus, що є критично важливим для забезпечення слабкої зв’язності (loose coupling) між мікро-додатками віртуальної ОС та ядром гри;

– *екосистема бібліотек*: широкий вибір готових рішень, зокрема інструментів для серіалізації даних (наприклад, Newtonsoft.Json), що суттєво спрощує роботу зі збереженням прогресу користувача, налаштуваннями та конфігураціями діалогів.

Завдяки синергії графічних можливостей Unity та потужної об’єктно-орієнтованої бази C#, розробник отримує змогу застосовувати класичні патерни проєктування (Command, State, Factory) та дотримуватися принципів SOLID [2, 4]. Це перетворює мобільну гру з монолітного скрипта на модульний, стабільний програмний продукт, готовий до довгострокового супроводу, тестування та інтеграції нового контенту на платформі Google Play.

## 1.4 Формування вимог до застосунку та застосування принципів SOLID у проєктуванні ігор

Для забезпечення високої якості кінцевого програмного продукту на етапі проєктування було сформовано чіткий перелік вимог. Їх доцільно розділити на функціональні, що стосуються безпосередньо ігрового процесу, та нефункціональні, які визначають технічні обмеження та стандарти якості архітектури.

Схематично склад **функціональних вимог** та сценаріїв взаємодії акторів із системою представлено на рисунку 1.1.

У межах проєктування виділено дві ключові дійові особи (актори):

– гравець – основний користувач, який безпосередньо взаємодіє з ігровим інтерфейсом симулятора та керує ігровим процесом;

– система – фоновий автоматизований актор, який відповідає за внутрішні обчислення, збереження станів та автоматичне керування економічними тригерами.

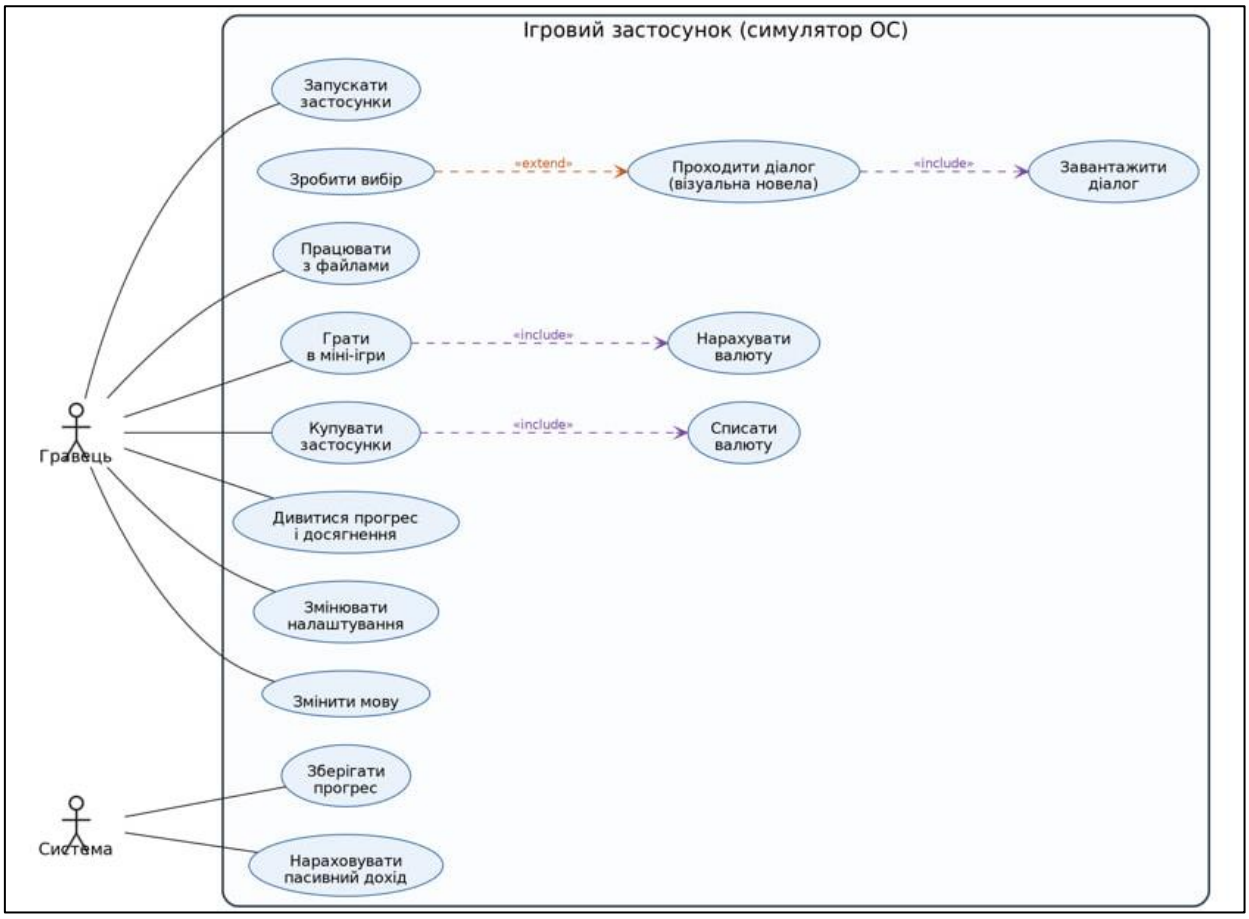



Рисунок 1.1 – Діаграма прецедентів мобільного ігрового застосунку

Взаємодія актора «Гравець» із межами системи охоплює широкий спектр прецедентів, пов’язаних як із суто геймплейними метамеханіками, так і з утилітарним налаштуванням додатку. До базових варіантів використання належать: «Запускати застосунки», «Працювати з файлами» (що ілюструє взаємодію з ізольованим провідником віртуальної ОС), «Дивитися прогрес і досягнення», а також менеджмент конфігурацій – «Змінювати налаштування» та «Змінити мову».

Специфіку побудови ігрової логіки та зв’язків між підсистемами деталізовано через базові UML-відношення (include та extend).

*Наративний шар (візуальна новела)*: варіант використання «Проходити діалог (візуальна новела)» обов’язково включає («include») прецедент «Завантажити діалог» для ініціалізації текстових скриптів. Водночас, прецедент «Зробити вибір» розширює («extend») процес проходження діалогу, оскільки вибір є ситуативним і варіативним елементом геймплею.

*Ігрова економіка*: прецедент «Грати в міні-ігри» передбачає обов'язкове включення («include») зв'язку «Нарахувати валюту» для забезпечення ігрової прогресії. У свою чергу, прецедент «Купувати застосунки» у внутрішньоігровому магазині пов'язаний відношенням «include» із прецедентом «Списати валюту», що фіксує транзакційну логіку мета-шару.

Актор «Система» функціонує автономно і взаємодіє з прецедентами «Зберігати прогрес» (забезпечення збереження стану через серіалізацію даних) та «Нараховувати пасивний дохід», що реалізує довгострокові механіки утримання користувача в межах симулятора операційної системи.

До ключових **нефункціональних вимог** розроблюваного застосунку належать:

- *висока продуктивність та оптимізація*: застосунок має функціонувати без падінь частоти кадрів (зависань) та гарантувати плавний рендеринг UI навіть на бюджетних мобільних пристроях минулих поколінь – це вимагає жорсткого контролю за виділенням пам'яті (Garbage Collection) та оптимізації життєвого циклу об'єктів;
- *абсолютна автономність*: усі ігрові процеси, включно з системою збереження прогресу, логікою діалогів та роботою імітованої ОС, повинні повноцінно функціонувати в офлайн-режимі без жодної потреби у підключенні до мережі Інтернет;
- *гнучкість та масштабованість (Scalability)*: архітектура має бути побудована таким чином, щоб інтеграція нового контенту (додаткових міні-ігор, сюжетних гілок чи нових додатків усередині віртуальної ОС) відбувалася виключно шляхом написання нового коду – існуючі фундаментальні класи мають залишатися незмінними для запобігання регресійним багам.

Реалізація настільки суворих вимог до масштабованості унеможливлює використання стихійного написання коду або створення монолітних структур із жорсткою зв'язністю (tight coupling), що часто зустрічається в інді-розробці. Для побудови надійної архітектури в основу проєкту було покладено патерни

проєктування та **п’ять фундаментальних принципів** об’єктно-орієнтованого програмування – SOLID. Особливості практичного застосування цих рішень та їх безпосередню програмну реалізацію в розробленому застосунку доцільно розглянути детальніше.

*Принцип єдиної відповідальності* (Single Responsibility Principle, SRP). Кожен клас у системі вирішує виключно одну задачу. Наприклад, клас SaveManager відповідає лише за серіалізацію та десеріалізацію поточного стану гри, тоді як управління візуалізацією героїв новели делеговано окремому VNCharacterManager. Це унеможливлює формування неконтрольованих «супер-класів» (God Objects).

*Принцип відкритості/закритості* (Open/Closed Principle, OCP). Програмні сутності відкриті для розширення, але закриті для модифікації. Впровадження системи додатків та команд віртуальної ОС базується на абстракціях. Для додавання нової механіки чи утиліти розробнику достатньо створити новий клас, що реалізує базовий контракт, без внесення жодних змін до ядра ОС чи обробника команд.

*Принцип підстановки Лісков* (Liskov Substitution Principle, LSP). Об’єкти базових класів можуть бути безболісно замінені об’єктами похідних класів. У контексті абстрактної файлової системи застосунку будь-який об’єкт типу «Файл» (текстовий документ, зображення чи виконуваний скрипт емульованої ОС) коректно обробляється провідником без необхідності ручної перевірки його конкретного типу. Це гарантує стабільну поведінку системи при введенні нових форматів ігрових файлів.

*Принцип розділення інтерфейсу* (Interface Segregation Principle, ISP). Замість створення глобальних інтерфейсів, систему розбито на вузькоспеціалізовані контракти. Наприклад, об’єкти, стан яких потрібно зберігати, успадковують лише компактний інтерфейс ISaveable. Це позбавляє компоненти необхідності залежати від методів, які вони фактично не використовують у своїй логіці.

*Принцип інверсії залежностей* (Dependency Inversion Principle, DIP). Модулі верхнього рівня не залежать від модулів нижнього рівня; обидва рівні

залежать від абстракцій. Реалізація глобальної шини подій (EventBus) дозволила повністю декуплювати (ізолювати) підсистеми одна від одної. Модуль сповіщень (NotificationManager) не має прямих посилань на логіку файлової системи – він лише підписується на абстрактні події та реагує на них.

Такий архітектурний підхід дозволив створити надійний технологічний фундамент. Завдяки строгому дотриманню SOLID розширення проєкту є передбачуваним і безпечним, а код залишається чистим і легко підтримується на всіх етапах життєвого циклу ПЗ.

# 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ІГРОВОГО ЗАСТОСУНКУ

## 2.1 Проєктування глобальної шини подій (EventBus) для забезпечення слабкої зв’язності компонентів

Архітектура розробленого ігрового застосунку побудована на модульних принципах із застосуванням шини подій та централізованих менеджерів. Структурна схема класів, що ілюструє взаємозв'язки між основними компонентами системи, представлена на рисунку 2.1.

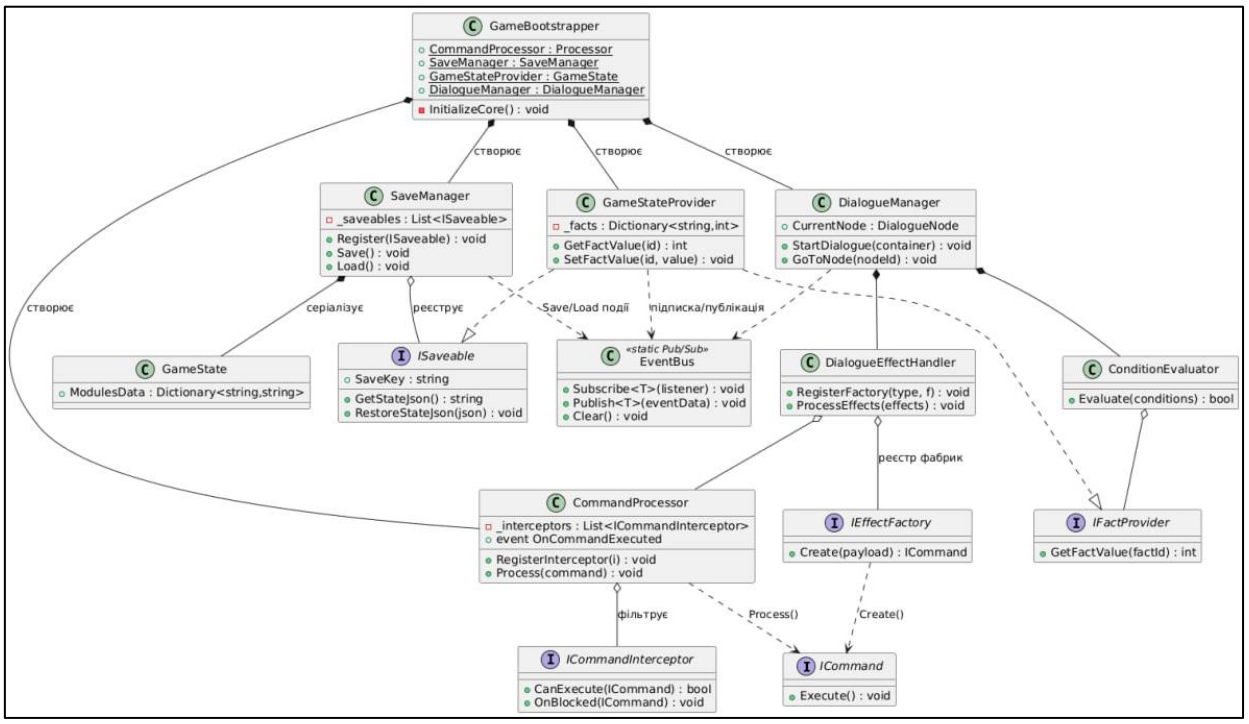



Рисунок 2.1 – Діаграма класів архітектури застосунку

Однією з найпоширеніших проблем при розробці комплексних ігрових застосунків є висока зв’язність (tight coupling) між різними підсистемами. У класичній монолітній архітектурі взаємодія компонентів часто реалізується через прямі посилання. Наприклад, подія отримання нового повідомлення у віртуальній ОС вимагає, щоб системний контролер безпосередньо викликав методи модуля візуального інтерфейсу (UI) та підсистеми аудіо

(AudioManager) для відтворення звуку. Це порушує принцип інверсії залежностей (Dependency Inversion Principle) і призводить до того, що зміна або видалення одного компонента викликає ланцюгову реакцію помилок у всьому проєкті.

Для уникнення цієї проблеми та забезпечення слабкої зв’язності (loose coupling) в архітектуру застосунку було інтегровано глобальну шину подій (EventBus). Цей архітектурний патерн базується на парадигмі «видавець-підписник» (Publish-Subscribe) і виступає єдиним централізованим посередником для обміну повідомленнями між усіма мікро-додатками та системами гри.

Основний принцип роботи EventBus полягає в наступному:

– видавець (Publisher) створює об’єкт події (наприклад, FileDeletedEvent) і передає його в шину, абсолютно нічого не знаючи про те, хто саме оброблятиме цю інформацію;
– шина (Bus) приймає подію та маршрутизує її;
– підписники (Subscribers) заздалегідь реєструються в шині для прослуховування конкретного типу подій – отримавши подію, вони виконують власну інкапсульовану логіку (наприклад, оновлюють іконку в провіднику).

Програмну реалізацію EventBus виконано мовою C# з активним використанням узагальнень (Generics) та делегатів (Action<T>). Структурно шина подій являє собою статичний клас (або синглтон), який містить словник (Dictionary), де ключем виступає тип події (System.Type), а значенням – список підписаних на неї делегатів.

Такий підхід дозволив повністю ізолювати ігрову логіку візуальної новели від системних утиліт віртуальної ОС. Компоненти існують незалежно один від одного, що критично важливо для масштабування проєкту: нові додатки можуть бути додані у віртуальний смартфон шляхом простої підписки на існуючі системні події без жодного втручання в ядро застосунку.

Фрагмент програмного коду, що демонструє базову реалізацію менеджера подій та механізм підписки, наведено у Додатку А.1.

## 2.2 Застосування патерна Command та системи перехоплювачів (Interceptors) для логіки мета-шару

У межах розробки мета-шару застосунку (емуляції віртуальної ОС) виникає потреба в гнучкому та масштабованому управлінні діями користувача. Взаємодія гравця з файловою системою або віртуальними додатками (відкриття, копіювання, видалення об’єктів) генерує велику кількість транзакцій, які потребують уніфікованої обробки. Для вирішення цього завдання було імплементовано поведінковий патерн Command.

Патерн Command дозволяє інкапсулювати будь-який запит користувача у вигляді окремого об’єкта, що реалізує базовий інтерфейс ICommand із методом Execute(). Такий підхід забезпечує повне відокремлення об’єкта, який ініціює операцію (наприклад, кнопка UI), від об’єкта, який знає, як саме її виконати (наприклад, ядро файлової системи).

Особливістю віртуальної операційної системи як ігрової механіки є необхідність ситуативного контролю за виконанням команд. За сюжетом візуальної новели гравець не повинен мати змоги випадково або навмисно видалити критично важливі системні файли чи застосунки, необхідні для подальшого проходження гри. Хардкодування таких перевірок безпосередньо в класах команд або в UI призвело б до порушення принципу єдиної відповідальності (SRP) та сильного зачеплення коду.

Для вирішення проблеми маршрутизації та блокування команд архітектуру було розширено за допомогою системи перехоплювачів (Interceptors). Було створено інтерфейс ICommandInterceptor, що містить метод CanExecute (для перевірки дозволу на виконання) та OnBlocked (для обробки відмови, наприклад, виведення системного сповіщення гравцю про неможливість видалення файлу).

Центральним вузлом обробки виступає клас CommandProcessor. Під час надходження нової команди процесор спочатку ітерує через масиви зареєстрованих перехоплювачів.

У системі реалізовано два рівні перехоплення:

– глобальні перехоплювачі (_globalInterceptors): аналізують абсолютно всі команди в системі (наприклад, блокування будь-якого вводу під час відтворення критичної сюжетної катсцени);
– типізовані перехоплювачі (_typedInterceptors): прив’язані до конкретного типу команди (наприклад, перехоплювач, що реагує виключно на DeleteFileCommand і перевіряє рівень доступу гравця до конкретного файлу).

Якщо хоча б один перехоплювач повертає негативний результат, виклик методу Execute() скасовується. Завдяки цій архітектурі додавання нових ігрових умов або обмежень відбувається шляхом створення нового класу-перехоплювача без внесення жодних змін в існуючий код ядра, що повністю задовольняє принцип відкритості/закритості (OCP).

Фрагмент програмної реалізації обробника команд наведено у Додатку А.2.

## 2.3 Проєктування абстракції файлової системи та управління додатками емульованої ОС

Центральним елементом мета-гри у розробленому застосунку виступає віртуальний провідник (Explorer) – інструмент, що дозволяє гравцеві взаємодіяти з файлами та каталогами емульованої операційної системи. Оскільки ця механіка є ключовою для сюжетного прогресу, виникла потреба у проєктуванні власної абстрактної моделі файлової системи, яка працювала б ізольовано від реальної файлової системи мобільного пристрою гравця.

При проєктуванні ієрархічних структур (якою є класична файлова система з вкладеними папками) найчастіше застосовується структурний патерн Composite (Компонувальник). У такому підході кожен вузол типу «Папка» містить колекцію посилань на свої дочірні елементи. Проте у

контексті мобільної розробки на рушії Unity цей підхід має суттєві недоліки: глибока рекурсивна серіалізація складних дерев об’єктів у формат JSON є ресурсомісткою та часто призводить до переповнення стека або витоків пам’яті.

З метою оптимізації продуктивності та спрощення процесу збереження стану, було прийнято рішення відмовитися від патерна Composite на користь плоскої реляційної моделі даних. Усю структуру емульованої файлової системи побудовано навколо єдиного базового класу FileNode. Цей клас описує будь-який об’єкт у системі (як директорію, так і кінцевий файл) і містить унікальний ідентифікатор (Id), а також ідентифікатор батьківської директорії (ParentId).

Типізація об’єктів здійснюється за допомогою переліку (enum) FileType, який визначає поведінку вузла (текстовий документ, зображення, папка або невідомий системний файл). Крім того, клас FileNode інкапсулює логіку доступу, містячи прапорці блокування (IsLocked) та дані для перевірки паролів доступу.

Управління життєвим циклом об’єктів FileNode делеговано менеджеру файлової системи (FileSystemManager). Замість побудови складного дерева у пам’яті, менеджер зберігає всі існуючі вузли у плоскому словнику (Dictionary<string, FileNode>), де ключем виступає Id файлу. Таке архітектурне рішення забезпечує ряд переваг:

– пошук будь-якого файлу або папки відбувається за час O(1);
– формування вмісту конкретної папки реалізується швидким LINQ-запитом, що фільтрує всі існуючі вузли за параметром ParentId;
– серіалізація всієї файлової системи для збереження прогресу зводиться до конвертації одного лінійного списку в JSON.

Інтеграція FileSystemManager з іншими системами віртуальної ОС (наприклад, візуальним інтерфейсом провідника) відбувається виключно через глобальну шину подій (EventBus) та команди. Менеджер підписаний на системні події (CreateFileEvent, DeleteFileEvent, UpdateFileContentEvent) і

реагує на них зміною стану внутрішнього словника. Операції видалення директорій обробляються рекурсивно, гарантуючи коректне очищення пам’яті від дочірніх об’єктів, після чого менеджер генерує зворотну подію FileSystemUpdatedEvent, сповіщаючи UI про необхідність оновлення екрана.

Фрагменти програмної реалізації моделі даних файлової системи та її менеджера наведено у Додатку А.3 [7].

## 2.4 Архітектура збереження стану гри (SaveManager) на основі інтерфейсу ISaveable

У мобільних іграх надійне збереження прогресу користувача є критично важливим аспектом, оскільки життєвий цикл застосунку на ОС Android може бути непередбачувано перерваний системою (наприклад, під час вхідного дзвінка, згортання програми або нестачі оперативної пам’яті). Класичним, але хибним підходом у мобільній розробці є створення єдиного глобального класу-менеджера, який напряму збирає дані з усіх ігрових об’єктів для їх збереження. Це грубо порушує принцип відкритості/закритості (OCP) та принцип єдиної відповідальності (SRP), перетворюючи систему збереження на непідтримуваний моноліт, який потребує постійної модифікації при кожному додаванні нового ігрового контенту.

Для вирішення цієї архітектурної проблеми систему збереження розробленого застосунку було побудовано на основі принципу розділення інтерфейсу (ISP) та інверсії залежностей (DIP). Фундаментом підсистеми виступає компактний інтерфейс ISaveable, який визначає єдиний контракт для будь-якого компонента, що потребує збереження свого стану. Інтерфейс вимагає реалізації трьох елементів: унікального ідентифікатора (SaveKey), методу генерації поточного стану у форматі JSON (GetStateJson) та методу відновлення стану з JSON-рядка (RestoreStateJson).

Головний контролер збережень – SaveManager – функціонує як незалежний координатор. Він не має жодних прямих посилань на конкретні

класи логіки (наприклад, провідник або систему діалогів). Натомість модулі самостійно реєструються в менеджері, передаючи посилання на себе як на абстракцію ISaveable під час власної ініціалізації.

Процес збереження та завантаження повністю декупльовано від основної ігрової логіки завдяки використанню глобальної шини подій (EventBus). SaveManager підписаний на події SaveGameEvent та DeleteSaveEvent. Отримавши сигнал про необхідність збереження, менеджер ітерує масив усіх зареєстрованих об’єктів, збирає їхні серіалізовані дані за допомогою бібліотеки Newtonsoft.Json та формує єдиний об’єкт GameState [7]. Фізичний запис даних на пристрій виконується у файл save.json за системним шляхом Application.persistentDataPath. Цей підхід гарантує доступність даних в офлайн-режимі та їх захист на рівні операційної системи.

Така архітектура забезпечує абсолютну масштабованість проєкту: щоб додати збереження прогресу для нової міні-гри чи застосунку віртуальної ОС, розробнику достатньо реалізувати інтерфейс ISaveable у новому класі. Вносити зміни у вихідний код SaveManager не потрібно.

Фрагменти програмної реалізації системи збереження наведено у Додатку А.4 [7, 8].

## 2.5 Розробка структури обробки діалогів та управління ігровими подіями (Facts)

Оскільки розроблюваний застосунок належить до жанру візуальної новели, ключовим елементом геймплею є подача сюжету через розгалужену систему діалогів. Нелінійність оповіді вимагає надійної архітектури для відстеження рішень гравця, управління станом ігрового світу та динамічної маршрутизації сюжетних гілок. Для вирішення цього завдання було розроблено модульну підсистему, яка базується на концепції ігрових фактів (Facts) та графовій структурі діалогів.

Управління ігровими подіями базується на фактах, які являють собою кількісні показники, що зберігають поточний стан прогресу: рівень відносин із персонажами, наявність знайдених доказів у віртуальній ОС або ключові сюжетні вибори. За управління фактами відповідає клас GameStateProvider, який імплементує інтерфейс IFactProvider та інтегрований у загальну систему збереження ISaveable. Зміна стану фактів відбувається виключно через глобальну шину подій (наприклад, ModifyFactEvent). Такий підхід гарантує, що будь-який компонент гри може вплинути на сюжет, не маючи прямого посилання на менеджер стану, що усуває жорстку зв’язність коду.

Дані діалогів, у свою чергу, спроєктовано за принципом вузлів. Базовим контейнером виступає клас DialogueContainer, який містить перелік об’єктів DialogueNode. Кожен вузол інкапсулює вичерпну інформацію про поточну репліку: ідентифікатор спікера, локалізований текст, перелік можливих виборів гравця, автоматичні розгалуження сюжету та побічні ефекти. Наприклад, таким ефектом може бути виклик консольної команди емульованої ОС. Вся ця структура повністю серіалізується, що дозволяє зберігати сценарії у форматі JSON і редагувати їх без втручання у вихідний код проєкту.

Центральним контролером підсистеми обробки виступає DialogueManager. З метою дотримання принципу єдиної відповідальності (SRP), менеджер не займається самостійною перевіркою сюжетних умов. Цю задачу делеговано окремому модулю – ConditionEvaluator, який через інтерфейс IFactProvider отримує доступ до актуальних значень ігрових фактів. Під час переходу до нового вузла DialogueManager зберігає попередній стан у стек історії для забезпечення можливості зворотного відмотування діалогу. Далі модуль перевіряє автоматичні розгалуження через ConditionEvaluator і формує список доступних виборів для гравця, приховуючи або блокуючи певні варіанти відповіді залежно від наявних фактів.

Завдяки такому розділенню логіки, де дані відокремлені від обробника, а оцінка умов ізольована від системи збереження, забезпечується висока масштабованість наративної складової застосунку.

# 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

## 3.1 Реалізація візуального інтерфейсу (UI) емульованої ОС та її додатків

Візуальний інтерфейс користувача (UI) у розроблюваному застосунку виконує не лише функцію управління, але й виступає головним інструментом імерсивності, оскільки він імітує реальне середовище мобільної операційної системи.

Для забезпечення коректного відображення інтерфейсу на всьому спектрі сучасних Android-пристроїв було застосовано компонент Canvas Scaler [5]. Базову роздільну здатність екрана зафіксовано на рівні 1080x1920 пікселів. Режим масштабування налаштовано на Scale With Screen Size із параметром Match, що дорівнює 0 (повна прив’язка до ширини екрана). Такий підхід є критично важливим, оскільки сучасні мобільні пристрої мають різні співвідношення сторін і відрізняються переважно за висотою. Прив’язка виключно до ширини дозволяє уникнути візуальних зламів та деформації елементів керування, зберігаючи оригінальні пропорції дизайну. Головний екран емульованої системи, що містить іконки застосунків та панель стану, наведено на рисунку 3.1.

Архітектуру управління візуальними вікнами побудовано за модульним принципом для забезпечення слабкої зв’язності. Фундаментом для всіх екранів виступає абстрактний клас BaseWindow. Він автоматично вимагає наявності компонента CanvasGroup, що дозволяє контролювати прозорість і блокування натискань. Завдяки інтеграції бібліотеки DOTween, відкриття та закриття вікон (Open та Close) супроводжується плавними анімаціями зміни альфа-каналу, що підвищує загальну якість користувацького досвіду (UX) [6].

Центральним контролером життєвого циклу інтерфейсу є UIManager. Він відповідає за ініціалізацію, збереження стану запущених додатків та

перемикання між ними. Для усунення жорстких залежностей менеджер взаємодіє з іншими підсистемами виключно через глобальну шину подій, підписуючись на OpenWindowEvent, GoHomeEvent та CloseAllWindowsEvent. При відкритті нового вікна попереднє не знищується, а деактивується або приховується, що значно знижує навантаження на систему збору сміття (Garbage Collector) і заощаджує ресурси мобільного пристрою.

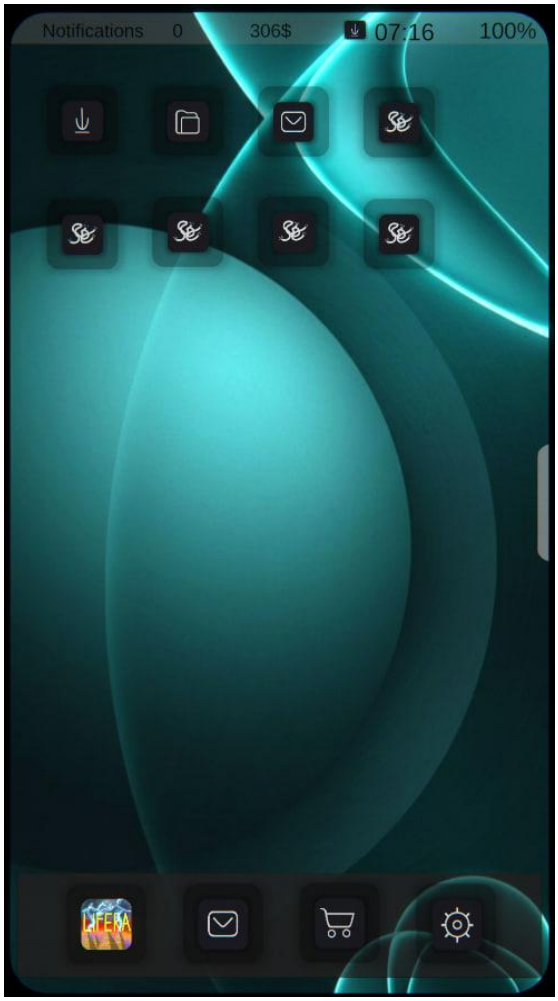



Рисунок 3.1 – Головний екран (робочий стіл) віртуальної ОС

Окрему увагу приділено зберіганню метаданих застосунків. Замість жорсткого кодування (hardcoding) інформації у скриптах вікон, було використано патерн Data-Driven Design на базі ScriptableObject. Було створено клас WindowId, який зберігає локалізовані ключі назв, іконки, ціни, скриншоти та імітовані відгуки користувачів для кожного віртуального додатка. Це дозволило повністю відокремити візуальні дані від програмної логіки.

Найбільш комплексно використання цих даних розкрито в інтерфейсі віртуального магазину додатків, де динамічно генеруються списки доступних для завантаження програм на основі об’єктів WindowId. Візуалізацію цього застосунку наведено на рисунку 3.2.

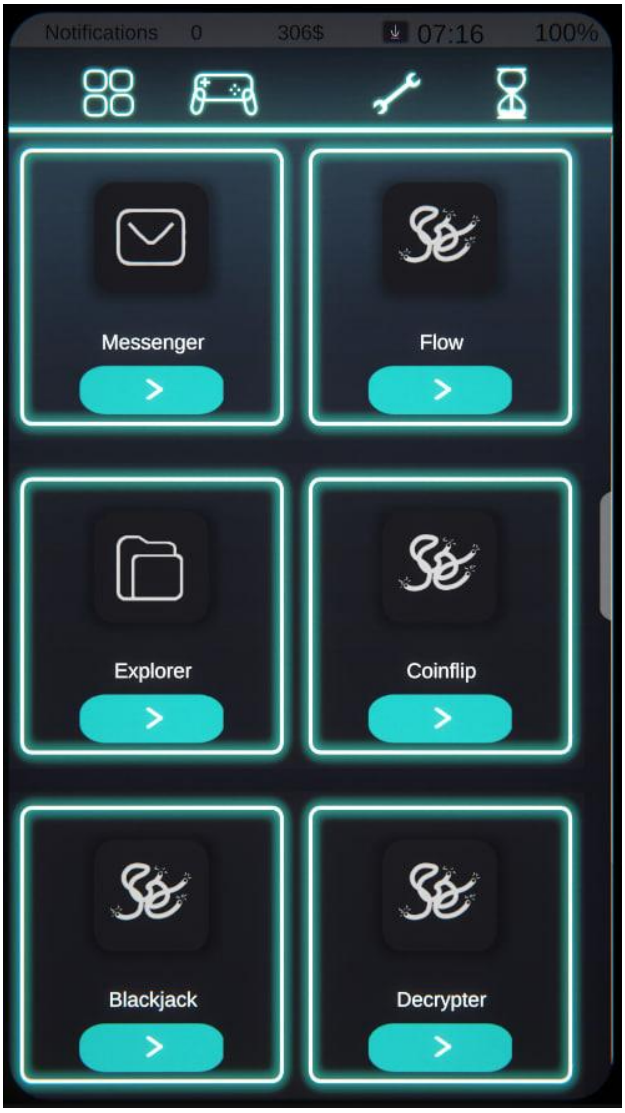



Рисунок 3.2 – Інтерфейс віртуального магазину додатків

Така архітектура графічного інтерфейсу гарантує стабільну частоту кадрів, відсутність витоків пам’яті при тривалих ігрових сесіях та дозволяє легко додавати нові віртуальні застосунки шляхом створення нових об’єктів WindowId у редакторі Unity.

Фрагменти коду реалізації системи управління візуальним інтерфейсом наведено у Додатку Б.1.

## 3.2 Розробка системи візуалізації персонажів (VNCharacterManager)

У жанрі візуальних новел графічне представлення дійових осіб та їхня реакція на хід діалогу є ключовим інструментом передачі емоцій та утримання уваги гравця. Для забезпечення гнучкої взаємодії між текстовою підсистемою та візуальним інтерфейсом було розроблено модульну систему візуалізації персонажів. Головною архітектурною вимогою до цього вузла було повне відокремлення логіки відображення від логіки обробки сюжету (DialogueManager). Для цього використано глобальну шину подій, що забезпечило дотримання принципу інверсії залежностей та єдиної відповідальності.

Центральним компонентом управління є клас VNCharacterManager, який ініціалізується на ігровій сцені та контролює життєвий цикл графічних об’єктів (префабів) персонажів. Менеджер утримує колекцію активних дійових осіб у словнику, що дозволяє швидко отримувати доступ до потрібного об’єкта за його унікальним ідентифікатором. Виклик нового персонажа на екран відбувається асинхронно через перехоплення події ShowCharacterEvent. Отримавши подію, менеджер визначає одну з трьох базових позицій для спавну (ліва, центральна або права частина екрана), інстанціює об’єкт та завантажує відповідний графічний спрайт із ресурсів застосунку.

За безпосередню анімацію та візуальний стан кожного окремого героя відповідає клас VNCharacterView. Оскільки під час діалогу на екрані може перебувати декілька персонажів одночасно, виникла потреба у створенні механізму фокусування уваги користувача на активному спікері. Цей механізм також реалізовано подієво-орієнтованим підходом: менеджер підписаний на подію DialogueNodePlayedEvent, яка генерується щоразу, коли система діалогів переходить до нової репліки. Отримавши ідентифікатор поточного спікера, VNCharacterManager ітерує всіх активних персонажів, викликаючи метод фокусування для того, хто говорить, та метод розфокусування для всіх інших.

Анімації станів реалізовано за допомогою бібліотеки DOTween, що забезпечує високу продуктивність та плавність переходів.

Стан фокусування збільшує масштаб персонажа, повертає його до початкової позиції, відновлює повну непрозорість та встановлює білий колір для оригінального відображення спрайта.

Стан розфокусування, навпаки, зменшує масштаб героя, зміщує його вбік на задану відстань, знижує альфа-канал та накладає сірий відтінок. Такий підхід інтуїтивно зрозуміло скеровує погляд гравця та створює ефект глибини простору.

Приклад роботи системи під час активного діалогу наведено на рисунку 3.3.




Рисунок 3.3 – Візуалізація персонажа під час діалогу

Фрагменти вихідного коду менеджера та контролера візуалізації наведено у Додатку Б.2.

### 3.3 Інтеграція підсистеми аудіосупроводу (AudioManager)

Звуковий супровід є невіддільною частиною занурення гравця у віртуальне середовище, особливо у жанрі візуальних новел, де аудіальний канал передачі інформації працює в синергії з текстовим наративом. Для управління всіма звуковими потоками в грі було розроблено централізовану підсистему AudioManager. Головним архітектурним завданням при її проєктуванні було забезпечення повної ізоляції від інших ігрових систем: модулі діалогів чи застосунки віртуальної ОС не повинні мати прямих посилань на аудіоконтролер. Це було успішно реалізовано через підключення менеджера до глобальної шини подій. Компоненти системи лише генерують структури даних, такі як PlayAudioEvent або StopAudioEvent, передаючи ідентифікатори потрібних треків, а менеджер самостійно обробляє ці запити.

Для гнучкого налаштування мікшування та гучності аудіосистему розділено на чотири незалежні канали, кожен з яких використовує окремий компонент AudioSource: фонова музика (BGM), звукові ефекти (SFX), системні звуки інтерфейсу (UI) та амбієнтні шуми (Ambient) [6]. База даних звуків реалізована за патерном Data-Driven Design через об’єкт ScriptableObject (AudioDatabaseSO), що дозволяє геймдизайнеру додавати нові аудіофайли, вказувати їхній тип та базову гучність без втручання у програмний код.

Особливу увагу приділено плавності звукових переходів, що є критичним для забезпечення якісного користувацького досвіду (UX). Замість різкого переривання фонових треків при зміні сюжетних сцен, у системі імплементовано механізм кросфейду (crossfade). Завдяки інтеграції бібліотеки DOTween, гучність поточного треку поступово знижується до нуля, після чого запускається новий аудіофайл із плавним наростанням гучності до цільового рівня. Такий підхід усуває неприємні звукові артефакти («клацання») при зміні музики. Для системних звуків інтерфейсу реалізовано процедурну варіативність: при кожному відтворенні звуку натискання на елементи UI його

висота (pitch) випадковим чином змінюється в діапазоні від 0.9 до 1.1. Це запобігає ефекту механічного повторення та робить взаємодію з віртуальною ОС більш природною.

Додатково реалізовано підтримку динамічних плейлистів для фонової музики. Ця функція працює на базі корутин (Coroutines) і дозволяє передавати масив ідентифікаторів треків для їх послідовного відтворення. Система автоматично вираховує тривалість кожного аудіофайлу, враховує час, необхідний на кросфейд, і динамічно перемикає треки із можливістю зациклення всього списку.

Фрагменти вихідного коду менеджера аудіосупроводу наведено у Додатку Б.3.

### 3.4 Інтеграція налаштувань гри (SettingsManager)

Забезпечення високого рівня користувацького досвіду (UX) та інклюзивності ігрового застосунку вимагає гнучкої системи конфігурації. Гравці повинні мати можливість адаптувати під себе рівень звуку, візуальні ефекти, параметри тексту та тактильний відгук (вібрацію). Для реалізації цього функціоналу було розроблено модульну підсистему управління налаштуваннями.

З метою дотримання принципу єдиної відповідальності (SRP), архітектуру підсистеми було розділено на три логічні рівні: контейнер даних, менеджер стану та контролер логіки.

Базовим контейнером виступає клас SettingsRecord, який містить виключно набір конфігураційних полів та їхні значення за замовчуванням. Такий підхід дозволяє легко серіалізувати дані у формат JSON без ризику збереження зайвих залежностей.

Повний перелік реалізованих параметрів налаштувань та їх призначення наведено у таблиці 3.1.

Таблиця 3.1 – Перелік та призначення конфігураційних параметрів застосунку

| **Категорія** | **Назва параметра** | **Тип даних** | **Опис та призначення** |
|---|---|---|---|
| Аудіо | MasterVolume | float | Загальний рівень гучності всього застосунку |
| Аудіо | MusicVolume | float | Гучність фонової музики (BGM) |
| Аудіо | SFXVolume | float | Гучність звукових ефектів (SFX) |
| Аудіо | UIVolume | float | Гучність системних звуків інтерфейсу |
| Аудіо | EnableClickSound | float | Перемикач відтворення звуку при натисканні на елементи UI |
| Керування | EnableGyroMenu | bool | Дозвіл на використання гіроскопа в головному меню |
| Керування | EnableGyroVN | bool | Дозвіл на паралакс-ефекти від гіроскопа у візуальній новелі |
| Геймплей | SkipMiniGames | bool | Можливість автоматичного пропуску міні-ігор для фокусу на сюжеті |
| Наратив | EnableTextAnimation | bool | Увімкнення ефекту поступового друку тексту реплік |
| Наратив | TextAnimationSpeed | bool | Швидкість анімації друку тексту |
| Наратив | DialogueFontSize | float | Розмір шрифту в діалогових вікнах (інклюзивне налаштування) |
| Тактильність | VibrationEnabled | bool | Загальний перемикач тактильного відгуку пристрою |
| Тактильність | UIVibrationEnabled | bool | Дозвіл на вібрацію при взаємодії з інтерфейсом |
| Тактильність | MiniGameVibrationEnabled | bool | Дозвіл на вібрацію під час проходження міні-ігор |
| Тактильність | VibrationStrength | float | Коефіцієнт інтенсивності тактильного відгуку |

Управління життєвим циклом даних делеговано класу SettingsState, який реалізує базовий інтерфейс ISaveable. Це дозволяє підсистемі налаштувань автоматично інтегруватися в загальну архітектуру збереження стану гри

(SaveManager) без написання додаткового сполучного коду, що повністю відповідає принципу відкритості/закритості (OCP).

Безпосередню обробку запитів від користувацького інтерфейсу виконує контролер SettingsManager. Окремої уваги заслуговує його взаємодія з компонентом AudioMixer рушія Unity [6]. Оскільки аудіомікшер Unity працює з логарифмічною шкалою гучності (у децибелах, від -80 дБ до 0 дБ), а повзунки в інтерфейсі користувача повертають лінійні значення (від 0.0 до 1.0), контролер виконує динамічну математичну конвертацію за формулою логарифма за основою 10.

Якщо лінійне значення падає нижче мінімального порогу, звук повністю глушиться (-80 дБ). Ця трансформація інкапсульована в методі SetMixerVolume, що робить управління звуком стабільним і передбачуваним для гравця.

Фрагменти програмної реалізації менеджерів стану та налаштувань наведено у Додатку Б.4.

# 4 ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА СУПРОВІД НА ПЛАТФОРМІ GOOGLE PLAY

## 4.1 Стратегія локального тестування та підготовка релізної збірки застосунку

Перед безпосереднім розгортанням ігрового застосунку на платформі Google Play необхідно провести фінальне локальне тестування та виконати строгу конфігурацію проєкту в середовищі Unity. Після перевірки всіх ігрових механік та оптимізації пам’яті в режимі Play Mode розробник переходить до налаштувань збірки (Player Settings). Першим критичним кроком є налаштування ідентифікаторів. Для коректної ідентифікації застосунку в екосистемі Android задається унікальний Application ID у форматі зворотного доменного імені (наприклад, com.companyname.appname). Також налаштовується система версіонування: параметр Version визначає публічну версію гри (наприклад, 1.0.0), яку бачать користувачі, тоді як Bundle Version Code – це цілочисельний внутрішній ідентифікатор, який обов’язково має інкрементуватися при кожному новому завантаженні оновлення в консоль розробника.

Важливим етапом підготовки є візуальне оформлення білда. У налаштуваннях встановлюється головна іконка застосунку (Default Icon), яка адаптується під різні роздільні здатності екранів мобільних пристроїв, та налаштовується завантажувальний екран (Splash Image). Це необхідно для дотримання стандартів платформи щодо користувацького досвіду (UX).

Для забезпечення безпеки та підтвердження авторства застосунку генерується криптографічний ключ підпису (Keystore) через вбудований інструмент Keystore Manager. Цей файл разом із паролями доступу надійно зберігається на локальному носії, оскільки втрата ключа безповоротно унеможливлює випуск будь-яких подальших оновлень для гри в Google Play.

На фінальному етапі підготовки, згідно з сучасними вимогами платформи Google, проєкт компілюється не у застарілий формат APK із додатковими файлами OBB, а в єдиний оптимізований архів Android App Bundle (.aab) [9]. Цей формат містить увесь скомпілюваний код та ігрові ресурси, дозволяючи серверам Google Play автоматично генерувати та доставляти користувачам оптимізовані APK-файли, що містять лише ті дані, які відповідають архітектурі конкретного мобільного пристрою. Отриманий файл .aab є кінцевим артефактом, готовим до завантаження на сервери магазину додатків.

## 4.2 Налаштування середовища Google Play Console та підготовка метаданих

Процес публікації мобільного застосунку вимагає створення та налаштування офіційного облікового запису в середовищі Google Play Console [9]. Для отримання доступу до панелі управління розробника необхідно пройти процедуру верифікації особи та сплатити одноразовий реєстраційний внесок. Цей етап є обов’язковим і надає пожиттєвий доступ до інструментарію публікації, збору аналітики та управління фінансовими транзакціями. Після успішної реєстрації створюється новий проєкт застосунку, для якого розпочинається етап конфігурації сторінки магазину.

Підготовка метаданих є критично важливим кроком як для внутрішніх алгоритмів ранжування маркетплейсу (ASO – App Store Optimization), так і для правильної презентації ігрового продукту кінцевому користувачу. До базового пакета інформації входить створення локалізованої назви гри, короткого та повного описів, а також завантаження графічних ресурсів: адаптивної іконки, серії репрезентативних скриншотів геймплею та головного банера. Відповідно до сучасних вимог безпеки платформи Google, обов’язковою умовою є надання валідного вебпосилання на політику конфіденційності (Privacy

Policy). Розробник також зобов’язаний заповнити декларацію безпеки даних (Data Safety), де деталізується інформація про те, які саме користувацькі дані збирає застосунок, чи передаються вони третім особам та чи зашифровані вони під час передачі.

Наступним обов’язковим кроком перед релізом є отримання офіційного вікового рейтингу. Для цього в консолі реалізовано інтегроване анкетування за системою Міжнародної коаліції щодо вікових рейтингів (IARC). Розробник має надати вичерпні відповіді на запитання щодо наявності у грі сцен насильства, ненормативної лексики, специфічного контенту, а також внутрішньоігрових покупок. На основі результатів цього опитування застосунку автоматично присвоюються регіональні вікові рейтинги, такі як PEGI для європейського ринку або ESRB для Північної Америки. Ці рейтинги не лише інформують користувачів, але й безпосередньо впливають на доступність гри в різних країнах та можливість її просування на платформі.

### 4.3 Пайплайн розгортання: від внутрішнього тестування до відкритого релізу

Згідно з сучасними політиками платформи Google Play, публікація нового ігрового застосунку відбувається не миттєво, а проходить через суворий багатоетапний пайплайн розгортання [10]. Першим етапом після завантаження скомпільованого файлу є внутрішнє тестування. Воно призначене для швидкої перевірки базової стабільності білда у вузькому колі довірених осіб та розробників. На цьому етапі перевіряється відсутність критичних помилок при запуску на різних конфігураціях пристроїв, що дозволяє оперативно вносити виправлення без ризику негативного впливу на публічну репутацію ігрового продукту.

Наступним, найбільш суворим етапом є закрите тестування [10]. Для нових облікових записів розробників платформа висуває жорсткі вимоги щодо

його проходження: застосунок має безперервно тестуватися визначеною кількістю користувачів протягом щонайменше 14 днів. При цьому алгоритми Google Play ретельно аналізують метрики залученості тестувальників. Правилами платформи забороняється формальне проходження етапу, тобто просте встановлення гри з подальшим її видаленням або одноразовим запуском. Для успішного проходження автоматичної перевірки тестувальники повинні демонструвати органічну поведінку: періодично відкривати застосунок, взаємодіяти з ігровими механіками та завантажувати оновлення, які розробник випускає під час цього періоду. Лише за умови дотримання таких суворих правил поведінки система дозволяє системі розробника перейти до наступного кроку.

Фінальним етапом пайплайну є подача заявки на публікацію застосунку у відкритий доступ (Production). Цей перехід супроводжується ще однією детальною верифікацією з боку модераторів платформи. Під час оформлення заявки розробник зобов’язаний заповнити спеціальну форму-опитувальник, де необхідно розгорнуто описати методологію проведення закритого тестування. Зокрема, вимагається надати інформацію про те, яким чином збирався зворотний зв’язок від користувачів, які саме проблеми продуктивності чи геймдизайну були виявлені під час ігрових сесій, та які архітектурні зміни були впроваджені для їх усунення. Тільки після ретельного ручного аудиту цієї декларації та фінальної перевірки коду застосунок стає загальнодоступним для завантаження користувачами з усього світу.

# ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання: проєктування, розробка та підготовка до розгортання на платформі Google Play масштабованого мобільного ігрового застосунку в жанрі візуальної новели з інтегрованими метамеханіками (емуляція операційної системи). У ході виконання роботи було досягнуто поставленої мети та повністю виконано всі завдання дослідження.

Проведено детальний аналіз предметної області та сучасного ринку мобільних ігор, який підтвердив високий попит користувачів на імерсивні проєкти з нестандартними метамеханіками. Доведено, що класична взаємодія з реальними файловими системами мобільних пристроїв руйнує ігровий досвід, що обґрунтувало необхідність розробки внутрішньоігрової емульованої ОС (дієгетичного інтерфейсу). Вибір ігрового рушія Unity та екосистеми C# дозволив забезпечити необхідну гнучкість, кросплатформеність та високу продуктивність застосунку.

Спроєктовано та імплементовано гнучку, масштабовану архітектуру програмного продукту, в основу якої покладено принципи чистого об’єктно-орієнтованого проєктування SOLID. Для усунення жорсткої зв’язності (tight coupling) між мікро-додатками віртуальної ОС, діалоговою системою та ядром гри розроблено та впроваджено глобальну шину подій (EventBus). Це рішення дозволило декуплювати архітектурні шари та дотриматися принципу інверсії залежностей (DIP).

Розроблено абстрактну модель файлової системи на базі плоскої реляційної структури даних, що суттєво оптимізувало процеси пошуку та серіалізації в пам’яті мобільного пристрою. Для маршрутизації та обробки дій гравця у віртуальному середовищі впроваджено патерн Command із дворівневою системою перехоплювачів (Interceptors). Це забезпечило надійний захист сюжетно-важливих системних файлів від видалення без

необхідності модифікації базового коду (дотримання принципу відкритості/закритості – OCP).

Створено надійну підсистему збереження поточного стану гри (SaveManager) на основі локального формату JSON та єдиного контракту ISaveable, що ізолює логіку серіалізації від бізнес-логіки об’єктів. Також реалізовано модульну систему обробки діалогів, керовану графовою структурою вузлів та ігровими фактами (Facts), де логіка оцінки сюжетних умов суворо відокремлена від візуалізації персонажів (VNCharacterManager) та відтворення звуку (AudioManager).

Успішно розв’язано завдання практичної реалізації графічного інтерфейсу через розробку UIManager із застосуванням патерна Data-Driven Design (через об’єкти ScriptableObject) та компонента Canvas Scaler. Це гарантувало коректну адаптацію візуального простору віртуального смартфона під екрани сучасних Android-пристроїв різних пропорцій. Впроваджено гнучку підсистему конфігурації (SettingsManager), що дозволяє керувати аудіо (з логарифмічним перетворенням для AudioMixer), анімаціями тексту та тактильним відгуком.

Визначено та успішно пройдено стратегію розгортання проєкту в екосистемі Google Play Console. Проєкт оптимізовано та скомпільовано в актуальний формат Android App Bundle (.aab) із генерацією криптографічного ключа підпису (Keystore). Виконано всі бюрократичні та технічні вимоги платформи щодо оформлення метаданих сторінки застосунку, пройдено вікове анкетування IARC та розроблено поетапний план подолання закритого тестування з дотриманням політики безперервного залучення тестувальників протягом 14 днів перед випуском продукту у відкритий доступ.

Отже, розроблений мобільний застосунок є не просто функціональною грою, а комплексним програмним продуктом із чистою, модульною архітектурою. Він повністю готовий до комерційного релізу, легкого масштабування (додавання нових сюжетних гілок чи віртуальних утиліт без переписування існуючого коду) і довгострокового супроводу на ринку.

Практичне значення одержаних результатів полягає у тому, що створена архітектурна модель може бути використана як надійний шаблон для розробки інших ігрових застосунків зі складною внутрішньою логікою.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення. Київ : Фабула, 2019. 368 с.

2. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley Professional, 1994. 416 p.

3. Troelsen A., Japikse P. Pro C# 9 with .NET 5: Foundational Principles and Practices in Program Design : 10th ed. Berkeley, CA : Apress, 2021. 1237 p.

4. Nystrom R. Game Programming Patterns. Genever Benning, 2014. 354 p. URL: https://gameprogrammingpatterns.com/ (дата звернення: 14.03.2026).

5. Unity User Manual (LTS). *Unity Documentation*. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 15.03.2026).

6. Scripting API Reference. *Unity Documentation*. URL: https://docs.unity3d.com/ScriptReference/ (дата звернення: 16.03.2026).

7. Newtonsoft.Json Documentation. *Newtonsoft*. URL: https://www.newtonsoft.com/json/help/html/Introduction.htm (дата звернення: 12.03.2026).

8. Richter J. CLR via C# : 4th ed. Redmond, Wash. : Microsoft Press, 2012. 896 p.

9. Як створити й налаштувати додаток. *Google Play Console Help*. URL: https://support.google.com/googleplay/android-developer/answer/9859152 (дата звернення: 18.03.2026).

10. Як налаштувати відкрите, закрите або внутрішнє тестування. *Google Play Console Help*. URL: https://support.google.com/googleplay/android-developer/answer/9845334 (дата звернення: 20.03.2026).

# ДОДАТОК А

## Фрагменти вихідного коду базових архітектурних модулів застосунку

### А.1 Реалізація глобальної шини подій (EventBus)

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

public static class EventBus
{
   private static readonly Dictionary<Type, Delegate> _events = new
Dictionary<Type, Delegate>();

   public static void Subscribe<T>(Action<T> listener) where T : struct
   {
      var type = typeof(T);
      if (_events.ContainsKey(type))
      {
         _events[type] = Delegate.Combine(_events[type], listener);
      }
      Else
      {
         _events[type] = listener;
      }
   }

   public static void Unsubscribe<T>(Action<T> listener) where T : struct
   {
      var type = typeof(T);
      if (_events.TryGetValue(type, out var currentDel))
      {
         _events[type] = Delegate.Remove(currentDel, listener);
         if (_events[type] == null)
         {
            _events.Remove(type);
         }
      }
   }

   public static void Publish<T>(T eventData) where T : struct
```

```
    {
        var type = typeof(T);
        if (_events.TryGetValue(type, out var currentDel))
        {
            (currentDel as Action<T>)?.Invoke(eventData);
        }
    }
    public static void Clear()
    {
        _events.Clear();
    }
}
```

## А.2 Реалізація обробника команд (CommandProcessor) та системи перехоплювачів

```
using System;
using System.Collections.Generic;

public interface ICommand
{
    void Execute();
}

public interface ICommandInterceptor
{
    bool CanExecute(ICommand command);
    void OnBlocked(ICommand command);
}

public class CommandProcessor
{
    public event Action<ICommand> OnCommandExecuted;

    private readonly List<ICommandInterceptor> _globalInterceptors = new();
    private readonly Dictionary<Type, List<ICommandInterceptor>> _typedInterceptors = new();

    public void RegisterGlobalInterceptor(ICommandInterceptor interceptor)
    {
        if (!_globalInterceptors.Contains(interceptor))
            _globalInterceptors.Add(interceptor);
    }
```

```
public void RemoveGlobalInterceptor(ICommandInterceptor interceptor)
{
   _globalInterceptors.Remove(interceptor);
}

public void RegisterInterceptor(Type commandType, ICommandInterceptor interceptor)
{
   if (!_typedInterceptors.TryGetValue(commandType, out var list))
   {
      list = new List<ICommandInterceptor>();
      _typedInterceptors[commandType] = list;
   }

   if (!list.Contains(interceptor))
      list.Add(interceptor);
}

public void RemoveInterceptor(Type commandType, ICommandInterceptor interceptor)
{
   if (_typedInterceptors.TryGetValue(commandType, out var list))
   {
      list.Remove(interceptor);
   }
}

public void Process(ICommand command)
{
   foreach (var interceptor in _globalInterceptors)
   {
      if (!interceptor.CanExecute(command))
      {
         interceptor.OnBlocked(command);
         return;
      }
   }

   var type = command.GetType();
   if (_typedInterceptors.TryGetValue(type, out var interceptors))
   {
      foreach (var interceptor in interceptors)
      {
         if (!interceptor.CanExecute(command))
         {
            interceptor.OnBlocked(command);
            return;
         }
```

```
            }
        }

        command.Execute();
        OnCommandExecuted?.Invoke(command);
    }
}
```

### А.3 Реалізація абстрактної моделі файлової системи емульованої ОС

```
using System;
using System.Collections.Generic;
using System.Linq;
using Newtonsoft.Json;

public enum FileType
{
    Folder,
    Text,
    Image,
    Unknown
}

[Serializable]
public class FileNode
{
    public string Id;
    public string ParentId;
    public string NameKey;
    public FileType Type;
    public string ContentPayload;
    public bool IsLocked;
    public string Password;
    public string SavedPassword;
}

public class FileSystemManager : ISaveable
{
    private Dictionary<string, FileNode> _files = new();

    public string SaveKey => "FileSystemState";
```

```
public FileSystemManager()
{
   GameBootstrapper.SaveManager.Register(this);
   EventBus.Subscribe<UpdateFileContentEvent>(OnUpdateFileContent);
   EventBus.Subscribe<CreateFileEvent>(OnCreateFile);
   EventBus.Subscribe<DeleteFileEvent>(OnDeleteFile);
}

~FileSystemManager()
{
   if (GameBootstrapper.SaveManager != null)
   {
      GameBootstrapper.SaveManager.Unregister(this);
   }
   EventBus.Unsubscribe<UpdateFileContentEvent>(OnUpdateFileContent);
   EventBus.Unsubscribe<CreateFileEvent>(OnCreateFile);
   EventBus.Unsubscribe<DeleteFileEvent>(OnDeleteFile);
}

public string GetStateJson()
{
   return JsonConvert.SerializeObject(_files.Values.ToList());
}

public void RestoreStateJson(string json)
{
   _files.Clear();
   if (!string.IsNullOrEmpty(json))
   {
      var loadedFiles = JsonConvert.DeserializeObject<List<FileNode>>(json);
      if (loadedFiles != null)
      {
         foreach (var file in loadedFiles)
         {
            _files[file.Id] = file;
         }
      }
   }

   if (!_files.ContainsKey("tutorial_sys"))
   {
      InitializeDefaultFileSystem();
   }
}

public List<FileNode> GetFilesInFolder(string folderId)
```

```
    {
        return _files.Values.Where(f => f.ParentId == folderId).ToList();
    }

    public FileNode GetFile(string fileId)
    {
        return _files.TryGetValue(fileId, out var node) ? node : null;
    }

    private void OnCreateFile(CreateFileEvent evt)
    {
        if (!_files.ContainsKey(evt.Node.Id))
        {
            _files[evt.Node.Id] = evt.Node;
            EventBus.Publish(new SaveGameEvent());
            EventBus.Publish(new FileSystemUpdatedEvent());
        }
    }

    private void OnDeleteFile(DeleteFileEvent evt)
    {
        if (_files.ContainsKey(evt.FileId))
        {
            DeleteNodeRecursive(evt.FileId);
            EventBus.Publish(new SaveGameEvent());
            EventBus.Publish(new FileSystemUpdatedEvent());
        }
    }

    private void DeleteNodeRecursive(string nodeId)
    {
        var children = GetFilesInFolder(nodeId);
        foreach (var child in children)
        {
            DeleteNodeRecursive(child.Id);
            EventBus.Publish(new FolderActionAttemptEvent { FolderId = child.Id, ActionType =
"Delete" });
        }
        _files.Remove(nodeId);
    }

    private void InitializeDefaultFileSystem()
    {
        var root = new FileNode
        {
            Id = "root",
```

```
            ParentId = "system_hidden",
            NameKey = "fs_root",
            Type = FileType.Folder
        };
        _files[root.Id] = root;

        var systemFolder = new FileNode
        {
            Id = "sys_folder",
            ParentId = "root",
            NameKey = "fs_sys",
            Type = FileType.Folder,
            IsLocked = true,
            Password = "admin"
        };
        _files[systemFolder.Id] = systemFolder;

        var tutorialFile = new FileNode
        {
            Id = "tutorial_sys",
            ParentId = "root",
            NameKey = "tutorial.sys",
            Type = FileType.Unknown,
            ContentPayload = ""
        };
        _files[tutorialFile.Id] = tutorialFile;

        EventBus.Publish(new SaveGameEvent());
    }
}
```

## А.4 Архітектура збереження стану гри (SaveManager та ISaveable)

```
using System.Collections.Generic;
using System.IO;
using Newtonsoft.Json;
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Linq;

public interface ISaveable
{
```

```
 string SaveKey { get; }
 string GetStateJson();
 void RestoreStateJson(string json);
}

public class SaveManager
{
 private readonly string _filePath;
 private readonly List<ISaveable> _saveables = new();
 private GameState _currentState;
 private bool _isWiping;

 public SaveManager()
 {
 _filePath = Path.Combine(Application.persistentDataPath, "save.json");
 EventBus.Subscribe<SaveGameEvent>(Save);
 EventBus.Subscribe<DeleteSaveEvent>(DeleteSave);
 }

 ~SaveManager()
 {
 EventBus.Unsubscribe<SaveGameEvent>(Save);
 EventBus.Unsubscribe<DeleteSaveEvent>(DeleteSave);
 }

 public void Register(ISaveable saveable)
 {
 if (!_saveables.Contains(saveable))
 {
 _saveables.Add(saveable);
 if (_currentState != null)
          {
             _currentState.ModulesData.TryGetValue(saveable.SaveKey, out var moduleJson);
             saveable.RestoreStateJson(moduleJson ?? string.Empty);
          }
       }
    }

    public void Unregister(ISaveable saveable)
    {
       _saveables.Remove(saveable);
    }

    public void ResetModules(params string[] saveKeys)
    {
       if (_currentState == null) return;
```

```
    foreach (var key in saveKeys)
    {
        if (_currentState.ModulesData.ContainsKey(key))
        {
            _currentState.ModulesData.Remove(key);
            var saveable = _saveables.FirstOrDefault(s => s.SaveKey == key);
            saveable?.RestoreStateJson(null);
        }
    }
    Save();
}

public void Load()
{
    if (File.Exists(_filePath))
    {
        var json = File.ReadAllText(_filePath);
        _currentState = JsonConvert.DeserializeObject<GameState>(json) ?? new GameState();
    }
    else
    {
        _currentState = new GameState();
    }

    foreach (var saveable in _saveables)
    {
        _currentState.ModulesData.TryGetValue(saveable.SaveKey, out var moduleJson);
        saveable.RestoreStateJson(moduleJson ?? string.Empty);
    }

    EventBus.Publish(new GameStateLoadedEvent());
}

public void Save(SaveGameEvent evt = default)
{
    if (_isWiping) return;

    if (_currentState == null) _currentState = new GameState();
    foreach (var saveable in _saveables)
    {
        _currentState.ModulesData[saveable.SaveKey] = saveable.GetStateJson();
    }

    var json = JsonConvert.SerializeObject(_currentState, Formatting.Indented);
    File.WriteAllText(_filePath, json);
}
```

```
    private void DeleteSave(DeleteSaveEvent evt)
    {
        _isWiping = true;

        if (File.Exists(_filePath))
        {
            File.Delete(_filePath);
        }

        foreach (var saveable in _saveables)
        {
            saveable.RestoreStateJson(null);
        }

        if (GameBootstrapper.LanguageState != null)
        {
            GameBootstrapper.LanguageState.Data.IsSelected = false;
        }

        _currentState = new GameState();

        EventBus.Clear();

        _isWiping = false;

        UnityEngine.SceneManagement.SceneManager.LoadScene(5);
    }
}
```

# ДОДАТОК Б

## Візуальне представлення графічного інтерфейсу застосунку

### Б.1 Програмна реалізація системи управління візуальним інтерфейсом (UI)

```
using UnityEngine;

using System.Collections.Generic;

public enum AppCategory
{
 EarlyAccess,
 MiniGames,
 Tools,
 System
}

[CreateAssetMenu(fileName = "NewWindowId", menuName = "OS/Window ID")]
public class WindowId : ScriptableObject
{
 public AppCategory Category;
 public string AppNameKey;
 public Sprite AppIcon;
 public int Price;

 [TextArea(3, 10)]
 public string DescriptionKey;
 public List<Sprite> Screenshots;

 public float MockRating = 4.5f;
 public List<string> MockComments;
}

using UnityEngine;
using DG.Tweening;

[RequireComponent(typeof(CanvasGroup))]
public abstract class BaseWindow : MonoBehaviour
{
   [field: SerializeField] public WindowId Id { get; private set; }
```

```
        protected CanvasGroup CanvasGroup;

    protected virtual void Awake()
    {
        CanvasGroup = GetComponent<CanvasGroup>();
    }

    public virtual void Open()
    {
        gameObject.SetActive(true);
        CanvasGroup.alpha = 0f;
        CanvasGroup.DOFade(1f, 0.3f);
    }

    public virtual void Close()
    {
        CanvasGroup.DOFade(0f, 0.3f).OnComplete(() => gameObject.SetActive(false));
    }
}

using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class UIManager : MonoBehaviour
{
    [SerializeField] private List<BaseWindow> _registeredWindows;

    private readonly List<BaseWindow> _runningWindows = new();
    private BaseWindow _currentActiveWindow;
    private readonly Dictionary<WindowId, BaseWindow> _windowsMap = new();

    private void Awake()
    {
        foreach (var window in _registeredWindows)
        {
            _windowsMap[window.Id] = window;
            window.gameObject.SetActive(false);
        }

        EventBus.Subscribe<OpenWindowEvent>(OnWindowOpened);
        EventBus.Subscribe<GoHomeEvent>(OnGoHome);
        EventBus.Subscribe<CloseAllWindowsEvent>(OnCloseAll);
    }

    private void OnDestroy()
```

```
{
    EventBus.Unsubscribe<OpenWindowEvent>(OnWindowOpened);
    EventBus.Unsubscribe<GoHomeEvent>(OnGoHome);
    EventBus.Unsubscribe<CloseAllWindowsEvent>(OnCloseAll);
}

private void OnWindowOpened(OpenWindowEvent evt)
{
    if (_windowsMap.TryGetValue(evt.TargetWindow, out var window))
    {
        if (_currentActiveWindow != null && _currentActiveWindow != window)
        {
            _currentActiveWindow.gameObject.SetActive(false);
        }

        if (!_runningWindows.Contains(window))
        {
            _runningWindows.Add(window);
            NotifyRunningAppsChanged();
        }

        window.Open();
        _currentActiveWindow = window;
    }
}

private void OnGoHome(GoHomeEvent evt)
{
    if (_currentActiveWindow != null)
    {
        _currentActiveWindow.Close();
        _currentActiveWindow = null;
    }
}

private void OnCloseAll(CloseAllWindowsEvent evt)
{
    OnGoHome(new GoHomeEvent());
    _runningWindows.Clear();
    NotifyRunningAppsChanged();
}

private void NotifyRunningAppsChanged()
{
    EventBus.Publish(new RunningAppsChangedEvent
    {
```

```
            RunningApps = _runningWindows.Select(w => w.Id).ToList()
        });
    }
}
```

## Б.2 Програмна реалізація системи візуалізації персонажів візуальної новели

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

public class VNCharacterManager : MonoBehaviour
{
 [SerializeField] private VNCharacterView _testPrefab;
 [SerializeField] private Transform _posLeft;
 [SerializeField] private Transform _posCenter;
 [SerializeField] private Transform _posRight;

 private readonly Dictionary<string, VNCharacterView> _activeCharacters = new();

 private void Awake()
 {
 EventBus.Subscribe<ShowCharacterEvent>(OnShowCharacter);
 EventBus.Subscribe<DialogueNodePlayedEvent>(OnDialogueNodePlayed);
 }

 private void OnDestroy()
 {
      EventBus.Unsubscribe<ShowCharacterEvent>(OnShowCharacter);
      EventBus.Unsubscribe<DialogueNodePlayedEvent>(OnDialogueNodePlayed);
   }

   private void OnShowCharacter(ShowCharacterEvent evt)
   {
      if (!_activeCharacters.TryGetValue(evt.CharId, out var character))
      {
         var parent = GetPosition(evt.PositionId);
         character = Instantiate(_testPrefab, parent);
         _activeCharacters[evt.CharId] = character;

         character.SetSprite(evt.CharId);
         character.FadeIn();
```

```
        }
        else
        {
            character.transform.SetParent(GetPosition(evt.PositionId), false);
            character.SetSprite(evt.CharId);
        }
    }

    private void OnDialogueNodePlayed(DialogueNodePlayedEvent evt)
    {
        foreach (var kvp in _activeCharacters)
        {
            if (kvp.Key == evt.CharacterId)
            {
                kvp.Value.Focus();
            }
            else
            {
                kvp.Value.Unfocus();
            }
        }
    }

    private Transform GetPosition(string positionId)
    {
        return positionId switch
        {
            "Pos_Left" => _posLeft,
            "Pos_Center" => _posCenter,
            "Pos_Right" => _posRight,
            _ => _posCenter
        };
    }
}

using UnityEngine;
using UnityEngine.UI;
using DG.Tweening;

[RequireComponent(typeof(CanvasGroup), typeof(Image))]
public class VNCharacterView : MonoBehaviour
{
    [SerializeField] private float _unfocusOffsetX = 800f;

    private CanvasGroup _canvasGroup;
    private Image _image;
```

```
private RectTransform _rect;

private bool _isFocused;
private bool _hasState;
private float _sideMultiplier = 1f;

private void Awake()
{
    _canvasGroup = GetComponent<CanvasGroup>();
    _image = GetComponent<Image>();
    _rect = GetComponent<RectTransform>();
    _canvasGroup.alpha = 0f;
}

private void Start()
{
    _sideMultiplier = transform.parent.localPosition.x >= 0f ? 1f : -1f;
}

public void SetSprite(string spriteName)
{
    var sprite = Resources.Load<Sprite>($"Images/Novell/Persons/{spriteName}");
    if (sprite != null)
    {
        _image.sprite = sprite;
    }
}

public void FadeIn(float duration = 0.5f)
{
    _canvasGroup.DOKill();
    transform.DOKill();

    _canvasGroup.DOFade(1f, duration);
    transform.localPosition = new Vector3(0, -50, 0);
    transform.DOLocalMoveY(0, duration).SetEase(Ease.OutQuad);
    _rect.localScale = Vector3.one * 1.2f;
}

public void FadeOutAndDestroy(float duration = 0.5f)
{
    _canvasGroup.DOKill();
    _canvasGroup.DOFade(0f, duration)
        .OnComplete(() => Destroy(gameObject));
}
```

```
    public void Focus()
    {
        if (_hasState && _isFocused) return;
        _isFocused = true;
        _hasState = true;

        _canvasGroup.DOKill();
        _rect.DOKill();
        _image.DOKill();

        _canvasGroup.DOFade(1f, 0.3f);
        _rect.DOScale(1.3f, 0.3f).SetEase(Ease.OutBack);
        _rect.DOAnchorPosX(0f, 0.3f).SetEase(Ease.OutBack);
        _image.DOColor(Color.white, 0.3f);
    }

    public void Unfocus()
    {
        if (_hasState && !_isFocused) return;
        _isFocused = false;
        _hasState = true;

        _canvasGroup.DOKill();
        _rect.DOKill();
        _image.DOKill();

        _canvasGroup.DOFade(0.8f, 0.3f);
        _rect.DOScale(1.2f, 0.3f).SetEase(Ease.OutQuad);
        _rect.DOAnchorPosX(_unfocusOffsetX * _sideMultiplier, 0.3f).SetEase(Ease.OutQuad);
        _image.DOColor(new Color(0.6f, 0.6f, 0.6f, 1f), 0.3f);
    }
}
```

## Б.3 Програмна реалізація підсистеми аудіосупроводу

```
using UnityEngine;
using System;
using System.Linq;
using System.Collections;
using DG.Tweening;

public struct PlayAudioEvent
```

```
{
 public string AudioId;
 public bool Loop;
}

public struct StopAudioEvent
{
 public AudioType Type;
 public float FadeDuration;
}

public class AudioManager : IDisposable
{
 private readonly AudioDatabaseSO _database;
 private readonly AudioSource _bgmSource;
 private readonly AudioSource _sfxSource;
 private readonly AudioSource _uiSource;
 private readonly AudioSource _ambientSource;
 private readonly MonoBehaviour _runner;

 private Tween _bgmFadeTween;
 private Tween _ambientFadeTween;
 private Coroutine _playlistRoutine;

 public AudioManager(MonoBehaviour runner, AudioDatabaseSO database, AudioSource bgm,
AudioSource sfx, AudioSource ui, AudioSource ambient)
    {
        _runner = runner;
        _database = database;
        _bgmSource = bgm;
        _sfxSource = sfx;
        _uiSource = ui;
        _ambientSource = ambient;

        EventBus.Subscribe<PlayAudioEvent>(OnPlayAudio);
        EventBus.Subscribe<StopAudioEvent>(OnStopAudio);
        EventBus.Subscribe<PlayPlaylistEvent>(OnPlayPlaylist);
    }

    public void Dispose()
    {
        EventBus.Unsubscribe<PlayAudioEvent>(OnPlayAudio);
        EventBus.Unsubscribe<StopAudioEvent>(OnStopAudio);
        EventBus.Unsubscribe<PlayPlaylistEvent>(OnPlayPlaylist);
    }
```

```
    private void OnPlayAudio(PlayAudioEvent evt)
    {
        if (_database == null) return;

        var entry = _database.Entries.FirstOrDefault(e => e.Id == evt.AudioId);
        if (entry == null) return;

        switch (entry.Type)
        {
            case AudioType.BGM:
                StopPlaylist();
                CrossfadeBGM(entry.Clip, entry.DefaultVolume, evt.Loop);
                break;
            case AudioType.Ambient:
                CrossfadeAmbient(entry.Clip, entry.DefaultVolume, evt.Loop);
                break;
            case AudioType.SFX:
                _sfxSource.PlayOneShot(entry.Clip, entry.DefaultVolume);
                break;
            case AudioType.UI:
                if (!GameBootstrapper.SettingsState.Data.EnableClickSound) return;
                _uiSource.pitch = UnityEngine.Random.Range(0.9f, 1.1f);
                _uiSource.PlayOneShot(entry.Clip, entry.DefaultVolume);
                _uiSource.pitch = 1f;
                break;
        }
    }

    private void OnStopAudio(StopAudioEvent evt)
    {
        if (evt.Type == AudioType.BGM)
        {
            StopPlaylist();
            _bgmFadeTween?.Kill();
            _bgmFadeTween = _bgmSource.DOFade(0f, evt.FadeDuration).OnComplete(() =>
_bgmSource.Stop());
        }
        else if (evt.Type == AudioType.Ambient)
        {
            _ambientFadeTween?.Kill();
            _ambientFadeTween = _ambientSource.DOFade(0f, evt.FadeDuration).OnComplete(()
=> _ambientSource.Stop());
        }
    }

    private void OnPlayPlaylist(PlayPlaylistEvent evt)
```

```
{
    StopPlaylist();
    if (evt.AudioIds == null || evt.AudioIds.Length == 0) return;
    _playlistRoutine = _runner.StartCoroutine(PlaylistRoutine(evt.AudioIds, evt.Loop));
}

private void StopPlaylist()
{
    if (_playlistRoutine != null)
    {
        _runner.StopCoroutine(_playlistRoutine);
        _playlistRoutine = null;
    }
}

private IEnumerator PlaylistRoutine(string[] ids, bool loop)
{
    int index = 0;
    while (true)
    {
        var entry = _database.Entries.FirstOrDefault(e => e.Id == ids[index]);
        if (entry != null && entry.Clip != null)
        {
            bool wasPlaying = _bgmSource.isPlaying;
            CrossfadeBGM(entry.Clip, entry.DefaultVolume, false);

            float waitTime = Mathf.Max(0.1f, entry.Clip.length + (wasPlaying ? 1f : 0f) - 1f);
            yield return new WaitForSeconds(waitTime);
        }
        else
        {
            yield return null;
        }

        index++;
        if (index >= ids.Length)
        {
            if (loop) index = 0;
            else break;
        }
    }
}

private void CrossfadeBGM(AudioClip newClip, float targetVolume, bool loop)
{
    _bgmFadeTween?.Kill();
```

```
        if (_bgmSource.isPlaying)
        {
            _bgmFadeTween = _bgmSource.DOFade(0f, 1f).OnComplete(() =>
            {
                _bgmSource.clip = newClip;
                _bgmSource.loop = loop;
                _bgmSource.Play();
                _bgmFadeTween = _bgmSource.DOFade(targetVolume, 1f);
            });
        }
        else
        {
            _bgmSource.clip = newClip;
            _bgmSource.loop = loop;
            _bgmSource.volume = 0f;
            _bgmSource.Play();
            _bgmFadeTween = _bgmSource.DOFade(targetVolume, 1f);
        }
    }

    private void CrossfadeAmbient(AudioClip newClip, float targetVolume, bool loop)
    {
        _ambientFadeTween?.Kill();
        if (_ambientSource.isPlaying)
        {
            _ambientFadeTween = _ambientSource.DOFade(0f, 1f).OnComplete(() =>
            {
                _ambientSource.clip = newClip;
                _ambientSource.loop = loop;
                _ambientSource.Play();
                _ambientFadeTween = _ambientSource.DOFade(targetVolume, 1f);
            });
        }
        else
        {
            _ambientSource.clip = newClip;
            _ambientSource.loop = loop;
            _ambientSource.volume = 0f;
            _ambientSource.Play();
            _ambientFadeTween = _ambientSource.DOFade(targetVolume, 1f);
        }
    }
}
```

## Б.4 Програмна реалізація підсистеми управління налаштуваннями гри

```
using System;
using Newtonsoft.Json;

[Serializable]
public class SettingsRecord
{
   public float MasterVolume = 0.3f;
   public float MusicVolume = 0.3f;
   public float sfxVolume = 0.3f;
   public float uiVolume = 0.3f;

   public bool EnableClickSound = true;
   public bool EnableGyroMenu = true;
   public bool EnableGyroVN = true;
   public bool SkipMiniGames = false;
   public bool EnableTextAnimation = true;
   public float TextAnimationSpeed = 1f;
   public float DialogueFontSize = 50f;
   public bool VibrationEnabled = true;
   public bool UIVibrationEnabled = false;
   public bool MiniGameVibrationEnabled = true;
   public float VibrationStrength = 1f;
}

public class SettingsState : ISaveable
{
   public SettingsRecord Data = new();
   public string SaveKey => "SettingsState";

   public SettingsState()
   {
      GameBootstrapper.SaveManager.Register(this);
   }

   public string GetStateJson() => JsonConvert.SerializeObject(Data);

   public void RestoreStateJson(string json)
   {
      if (!string.IsNullOrEmpty(json))
      {
         Data = JsonConvert.DeserializeObject<SettingsRecord>(json) ?? new SettingsRecord();
```

```
        }
    }
}

using UnityEngine;
using UnityEngine.Audio;

public class SettingsManager
{
    private readonly SettingsState _state;
    private readonly AudioMixer _audioMixer;

    public SettingsManager(SettingsState state, AudioMixer mixer = null)
    {
        _state = state;
        _audioMixer = mixer;
        ApplyAudioSettings();
    }

    public void SetGyroMenuEnabled(bool value) => _state.Data.EnableGyroMenu = value;
    public void SetGyroVNEnabled(bool value) => _state.Data.EnableGyroVN = value;
    public void SetDialogueFontSize(float value) => _state.Data.DialogueFontSize = value;
    public void SetSkipMiniGames(bool value) => _state.Data.SkipMiniGames = value;
    public void SetTextAnimation(bool value) => _state.Data.EnableTextAnimation = value;
    public void SetTextSpeed(float value) => _state.Data.TextAnimationSpeed = value;
    public void SetVibrationEnabled(bool value) => _state.Data.VibrationEnabled = value;
    public void SetUIVibrationEnabled(bool value) => _state.Data.UIVibrationEnabled = value;
    public void SetMiniGameVibrationEnabled(bool value) =>
_state.Data.MiniGameVibrationEnabled = value;
    public void SetVibrationStrength(float value) => _state.Data.VibrationStrength = value;
    public void SetClickSoundEnabled(bool value) => _state.Data.EnableClickSound = value;

    public void SetMasterVolume(float value)
    {
        _state.Data.MasterVolume = value;
        SetMixerVolume("MasterVolume", value);
    }

    public void SetMusicVolume(float value)
    {
        _state.Data.MusicVolume = value;
        SetMixerVolume("MusicVolume", value);
    }

    public void SetSFXVolume(float value)
    {
```

```
        _state.Data.sfxVolume = value;
        SetMixerVolume("SFXVolume", value);
    }

    public void SetUIVolume(float value)
    {
        _state.Data.uiVolume = value;
        SetMixerVolume("UIVolume", value);
    }

    public void ApplyAudioSettings()
    {
        if (_audioMixer == null) return;

        SetMixerVolume("MasterVolume", _state.Data.MasterVolume);
        SetMixerVolume("MusicVolume", _state.Data.MusicVolume);
        SetMixerVolume("SFXVolume", _state.Data.sfxVolume);
        SetMixerVolume("UIVolume", _state.Data.uiVolume);
    }

    private void SetMixerVolume(string parameterName, float linearValue)
    {
        if (_audioMixer == null) return;

        float dbValue = linearValue > 0.0001f ? Mathf.Log10(linearValue) * 20f : -80f;
        _audioMixer.SetFloat(parameterName, dbValue);
    }

    public SettingsRecord GetData() => _state.Data;
}
```