

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«2D гра жанру Metroidvania у стилі PixelArt:
проектування геймплейної та візуальної складової»**

Виконав студент групи ІПЗ-22-2

ІЛЬІН Ілля Володимирович

Керівник роботи:

ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:

ВЛАСЕНКО Олег Васильович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ІЛЬІН Ілля Володимирович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: **« 2D гра жанру Metroidvania у стилі PixelArt: проєктування геймплейної та візуальної складової »,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: Об'єктом дослідження є архітектурна реалізація однокористувацької 2D гри жанру Metroidvania, розробленої на Unity з використанням C#. Предметом дослідження є принципи проєктування архітектури багаторівневої двовимірної гри з явною розділенням ролей розробників, типізованою системою подій та чистими межами залежностей між модулями.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	01.03.2026р.	01.03.2026р
2	Лобанчикова Н.М.	01.03.2026р	01.03.2026р
3	Лобанчикова Н.М.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра містить 82 сторінки друкованого тексту. У роботі наведено 20 ілюстрацій, 6 таблиць та 41 сторінку додатків. Список використаних джерел налічує 32 найменування.

Метою роботи є розробка програмного забезпечення 2D-гри жанру Metroidvania «Fiat Lux» із застосуванням модульної архітектури, слабко зв'язаних підсистем та власних інженерних рішень, що мінімізують залежність від вбудованих високорівневих патернів ігрового рушія.

Методи дослідження. При виконанні роботи застосовано методи системного аналізу для проєктування архітектури, об'єктно-орієнтоване проєктування, патерни проєктування (зокрема Publish-Subscribe через реалізацію типізованого статичного EventBus, Stack для пріоритетного керування камерою та навігації екранів), а також методи детермінованої секційної процедурної генерації лінійного ігрового світу з опціональними відгалуженнями.

Отримані результати. У результаті виконання роботи спроектовано та реалізовано відеогру «Fiat Lux» на базі рушія Unity. Розроблено гнучку архітектуру з використанням ізоляції збірок (asmdef) та подієво-орієнтованої комунікації. Реалізовано власні підсистеми подієвої шини (EventBus), просторового переміщення камери (через пріоритетний стек), навігації інтерфейсу (UIScreenStack), збереження даних та детермінованої генерації рівнів без використання стандартних компонентів-«чорних скриньок».

КЛЮЧОВІ СЛОВА: METROIDVANIA, EVENTBUS, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, UNITY, ПРОЦЕДУРНА ГЕНЕРАЦІЯ.

					ІІЗ.КР.Б-121-26-ІІЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	2D гра жанру Metroidvania у стилі PixelArt: проєктування геймплейної та візуальної складової Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.	Ільїн І.В.						3	124
Керівник	Лобанчикова Н.М.					Житомирська політехніка група ІІЗ-22-2		
Рецензент	Власенко О.В.							
Зав. каф.	Вакалюк Т.А.							

ABSTRACT

The explanatory note to the bachelor's qualification thesis consists of 82 pages of printed text. The work contains 20 illustrations, 6 tables, and 41 pages of appendices. The list of used sources includes 32 items.

The aim of the work is to develop the software for a 2D Metroidvania game "Fiat Lux" using a modular architecture, loosely coupled subsystems, and custom engineering solutions that minimize dependence on the built-in high-level patterns of the game engine.

Research methods. The work employs system analysis methods for architecture design, object-oriented design, design patterns (in particular, Publish-Subscribe through the implementation of a typed static EventBus, and Stack for camera priority control and UI screen navigation), as well as deterministic section-based procedural generation of a linear game world with optional branches.

Obtained results. As a result of the work, the video game "Fiat Lux" was designed and implemented based on the Unity engine. A flexible architecture was developed using assembly isolation (asmdef) and event-driven communication. Custom subsystems for event-driven communication (EventBus), spatial camera movement (via a priority stack), UI screen navigation (UIScreenStack), data saving, and level generation were implemented without using standard "black box" components.

KEYWORDS: METROIDVANIA, EVENTBUS, SOFTWARE ARCHITECTURE, UNITY, PROCEDURAL GENERATION.

		Ільїн І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту.....	13
1.3 Вибір архітектури та архітектурного стилю	17
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	21
Висновки до розділу 1	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ 2D ІГРОВОГО ДОДАТКУ ЖАНРУ METROIDVANIA.....	26
2.1 Варіанти використання та об'єктно-орієнтована структура системи.....	26
2.2 Архітектура підсистем та міжмодульна взаємодія	30
2.3 Реалізація ключових підсистем	35
2.4 Діаграми та алгоритми	39
2.5 Реалізація функціоналу систем	44
Висновки до розділу 2	60
РОЗДІЛ 3. ІНТЕРФЕЙС, ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ	62
3.1 Порядок встановлення та налаштування.....	62
3.2 Опис інтерфейсу користувача	65
3.3 Тестування програмного продукту	71
Висновки до розділу 3	76
ВИСНОВКИ.....	78
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	80
ДОДАТКИ.....	83

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ADR – Architecture Decision Record (запис архітектурного рішення)

API – Application Programming Interface (прикладний програмний інтерфейс)

asmdef – Assembly Definition (файл визначення збірки)

CRT – Cathode-Ray Tube (електронно-променева трубка)

FSM – Finite State Machine (кінцевий автомат)

HFSM – Hierarchical Finite State Machine (ієрархічний кінцевий автомат)

HLSL – High-Level Shader Language (високорівнева мова програмування шейдерів)

HUD – Heads-Up Display (елементи інтерфейсу, що постійно відображаються на екрані)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

PPU – Pixels Per Unit (кількість пікселів на ігрову одиницю простору)

PRNG – Pseudo-Random Number Generator (генератор псевдовипадкових чисел)

SO – ScriptableObject (серіалізований об'єкт даних в середовищі Unity)

SOLID – Single responsibility principle; Open/closed principle; Liskov substitution principle; Interface segregation principle (Принципи дизайну та розробки для програмних систем що можуть тривалий час розвиватися, розширятися і підтримуватися)

UI – User Interface (інтерфейс користувача)

URP – Universal Render Pipeline (універсальний конвеєр рендерингу)

ПЗ – програмне забезпечення

		Львів І.В.			ППЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Стрімкий розвиток індустрії відеоігор та постійне зростання вимог до якості ігрових продуктів зумовлюють необхідність застосування складних інженерних підходів при їх створенні. Розробка 2D-ігор у жанрі Metroidvania характеризується високою комплексністю завдяки наявності великого безшовного світу, складної системи прогресії гравця, розгалуженої логіки поведінки ворогів та взаємопов'язаних ігрових механік.

Більшість сучасних ігрових рушіїв, зокрема Unity, пропонують широкий спектр готових рішень, які значно пришвидшують початкові етапи розробки. Однак використання стандартних нав'язаних патернів (таких як Cinemachine для управління камерою, CharacterController для фізики чи вбудованих механізмів аддитивного завантаження сцен) часто перетворює ці компоненти на «чорні скриньки». У довгостроковій перспективі це призводить до появи прихованого зв'язування між підсистемами, ускладнює підтримку коду, унеможливорює точний контроль над ігровою фізикою та порушує принципи чистої архітектури. Зі збільшенням масштабів проекту використання глобальних станів та перехресних посилань між візуальними й логічними шарами спричиняє появу помилок, які важко відстежити та виправити.

З огляду на це, розробка програмного забезпечення відеоігор із відмовою від готових високорівневих рішень рушія на користь власноруч спроектованих підсистем є актуальною інженерною задачею. Застосування строгої ізоляції модулів (через asmdef з вимкненим автоматичним посиланням), подієво-орієнтованої комунікації через типізований статичний EventBus та явного розділення відповідальності дозволяє створити архітектуру, що легко масштабується, підтримується та тестується. Такий підхід має високу актуальність не лише в межах створення конкретного ігрового продукту, а й як демонстрація фундаментальних принципів проектування надійного програмного забезпечення.

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

Мета і завдання роботи. Метою кваліфікаційної роботи є розробка програмного забезпечення 2D-гри жанру Metroidvania під назвою «Fiat Lux» із застосуванням гнучкої модульної архітектури, слабко зв'язаних підсистем та власних алгоритмічних рішень замість стандартних високорівневих пакетів ігрового рушія.

Для досягнення поставленої мети сформульовано такі завдання:

1. Провести системний аналіз предметної області та існуючих архітектурних підходів у розробці 2D-відеоігор.

2. Спроекувати та реалізувати архітектуру проєкту з використанням модульної ізоляції збірок (asmdef) для запобігання неявному зв'язуванню між логічним, візуальним та сервісним шарами.

3. Розробити систему міжмодульної взаємодії на основі патерну Publish-Subscribe за допомогою типізованого статичного EventBus.

4. Реалізувати ключові програмні підсистеми з нуля, зокрема: систему камери на основі пріоритетного стеку, подієво-орієнтовану систему збереження/завантаження стану, систему управління екранами інтерфейсу користувача (UIScreenStack) та детерміновану процедурну генерацію простору.

5. Розробити подієво-орієнтовані шари візуалізації та звуку (PlayerAnimator, PlayerSfxHandler), які інтегруються з геймплейним шаром виключно через шину подій EventBus, без прямих посилань між збірками.

6. Провести тестування розроблених систем та оптимізацію взаємодії компонентів гри.

Об'єкт та предмет дослідження. Об'єктом дослідження є процес проєктування та розробки програмного забезпечення інтерактивних ігрових 2D-додатків. Предметом дослідження є архітектурні патерни, принципи організації коду та алгоритмічні рішення, що забезпечують слабку зв'язність і масштабованість при розробці відеогри «Fiat Lux».

Методи дослідження. При виконанні роботи застосовано комплекс методів: системний аналіз – для формування вимог до структури проєкту; об'єктно-орієнтоване проєктування – для побудови архітектури підсистем;

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

патерни проектування (зокрема Stack, Observer, Singleton) – для організації взаємодії класів; методи детермінованої секційної процедурної генерації – при розробці алгоритму побудови лінійного ігрового світу з опціональними відгалуженнями на основі сіджованого System.Random; подієве програмування – для інтеграції візуальних і звукових підсистем з геймплейним шаром, що реалізує ієрархічні кінцеві автомати, через шину подій EventBus.

Практичне значення одержаних результатів. Розроблена архітектура та реалізовані підсистеми (EventBus, Camera Framework, Room System, Save/Load, UIScreenStack) є універсальними модульними рішеннями, які можуть бути виокремлені та повторно використані при розробці інших 2D-ігор на базі рушія Unity. Створений програмний продукт «Fiat Lux» функціонує як завершений ігровий прототип, що демонструє практичне застосування глибоких інженерних принципів в ігровій індустрії та вирішує проблему залежності від «чорних скриньок» ігрових рушіїв.

Апробація. За темою кваліфікаційної роботи було опубліковано 1 тезу конференції: Ільїн І. В., Лобанчикова Н. М., Концидайло А. М. Застосування технік тест-дизайну до верифікації ігрової логіки з розгалуженою структурою станів // Матеріали Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених «Тенденції розвитку технологій в автоматизації, електроенергетиці, приладобудуванні та робототехніці» (присвячена Дню Науки). - Житомир : Державний університет «Житомирська політехніка», 2026. - С. 101. [1].

		Ільїн І.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				9
Змн.	Арк.	№ док.м.	Підпис	Дата		

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

1.1 Постановка задачі

У рамках виконання кваліфікаційної роботи необхідно спроектувати та розробити програмне забезпечення для двовимірної відеоігри в жанрі Metroidvania під робочою назвою «Fiat Lux», що відповідає сучасним принципам геймдизайну [22]. Головним напрямком розробки є створення гнучкої, масштабованої та слабко зв'язаної архітектури, що базується на власних програмних рішеннях та зводить до мінімуму використання готових високорівневих компонентів ігрового рушія Unity. Завдання полягає не лише у створенні функціонального ігрового продукту, а й у демонстрації глибокого розуміння принципів програмної інженерії, зокрема через жорстку ізоляцію модулів та побудову прозорої системи комунікації між ними. Технічний стек включає рушій Unity версії 6000.0.33f1 із Universal Render Pipeline, мову C# відповідно до стандарту .NET Standard 2.1, систему Unity Input System для обробки введення та формат JSON для персистентності ігрових даних.

Першим ключовим завданням є проектування базової архітектури з використанням механізму Assembly Definitions (asmdef) для суворої ізоляції шарів збірок. Необхідно визначити ієрархію шарів, де FiatLux.Core як фундаментальний модуль без зовнішніх залежностей, FiatLux.Contracts як шар контрактів із визначеннями інтерфейсів та структур подій, сервісні збірки FiatLux.Services.* для автономних підсистем, а також шари FiatLux.World, FiatLux.UI та FiatLux.Visual як споживачі сервісного шару. Для кожної збірки слід встановити прапор `autoReferenced: false`, що унеможливлює випадкове створення неконтрольованих міжмодульних посилань. Вектор залежностей має бути спрямований виключно знизу вгору – геймплейний шар не повинен мати посилань на візуальний або інтерфейсний, а візуальний – на геймплейний. Точкою фінального зв'язування всіх підсистем визначено збірку FiatLux.Bootstrap, що реалізує підхід Bootstrap Scene для детермінованої

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

ініціалізації менеджерів у суворо визначеному порядку при будь-якому старті Play Mode.

Для забезпечення ефективної комунікації між ізольованими шарами необхідно розробити та імплементувати власну глобальну систему обміну повідомленнями на основі патерну Publish-Subscribe – типізований статичний клас EventBus з узагальненими методами where $T : IEvent$ [3, 4]. Реалізація цієї системи вимагає розробки механізмів підписки та відписки через Delegate.Combine і Delegate.Remove, методу Publish для одноразових подій та методу PublishLatest для подій зі збереженням останнього стану (sticky-події). Необхідно передбачити метод SubscribeWithLatest, що забезпечує новоствореному підписнику негайне отримання кешованого значення для подій, опублікованих до його ініціалізації – це є критичним для компонентів, що підписуються на ActiveRoomChangedEvent після завантаження сцени. Обов'язковим є механізм автоматичного скидання стану шини через атрибут RuntimeInitializeOnLoadMethod при кожному старті Play Mode для запобігання витокам підписок у середовищі редактора.

Наступним важливим завданням є реалізація системи управління ігровим світом та генерації простору. Проєкт вимагає повної відмови від стандартного механізму аддитивного завантаження та вивантаження сцен на користь концепції одночасної активності всіх кімнат в єдиній сцені – Persistent Room System [16]. Необхідно реалізувати клас RoomManager, що підтримує реєстр кімнат типу Dictionary<string, Room> та публікує кешовану подію ActiveRoomChangedEvent з просторовими межами при зміні активної зони. Переходи між кімнатами мають виконуватись через компонент RoomTeleport із корутинною послідовністю – затемнення екрана через ScreenFader, переміщення гравця на цільову точку появи, оновлення активної кімнати та плавне відновлення зображення. Разом із цим, необхідно розробити детермінований алгоритм процедурної генерації рівнів на основі числового seed із суворим розмежуванням між чистим C#-ядром LevelLayoutCore, що не має залежностей від UnityEngine, та MonoBehaviour-адаптером LevelGenerator, що виконує інстанціювання префабів та прив'язку

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

телепортів. Алгоритм має будувати головний коридор секційно та підтримувати вертикальні відгалуження з імовірністю ForkChance. Усі параметри генерації передаються через незмінне значення LevelLayoutConfig – readonly struct – що забезпечує чисте розмежування між конфігурацією та логікою [10].

У сфері візуалізації та інтерфейсу користувача ставиться завдання розробити власну модульну систему управління камерою без застосування пакету Cinemachine. Камера має функціонувати на основі пріоритетного списку поведінок, що реалізують інтерфейс ICameraBehaviour з методом CalculateTargetPosition, який повертає Vector2 бажаної позиції. Необхідно реалізувати FollowBehaviour з мертвою зоною у світових координатах та look-ahead зміщенням на основі вектора швидкості Rigidbody2D, CutsceneBehaviour для фіксації камери з пріоритетом вище базового, а також модуль CameraShake на основі моделі trauma-decay із шумом Перліна для органічного характеру тряски. Обов'язковим є метод SnapToPixel для вирівнювання позиції до кратного значення 1/PPU, що усуває субпіксельне тремтіння. Обмеження позиції межами кімнати отримується через SubscribeWithLatest на ActiveRoomChangedEvent. Також підлягає реалізації архітектура інтерфейсу UIScreenStack на основі Stack<UIScreen>, що керує відображенням екранів через методи OnPush, OnPop, OnHide та OnReveal із відновленням фокусу введення через EventSystem.SetSelectedGameObject при кожному відкритті. Метод HandleCancel має реалізовувати системну кнопку «Назад» без участі зовнішніх контролерів.

Окремим візуальним завданням визначено створення ефекту електронно-променевої трубки у вигляді власного HLSL-шейдера у форматі ShaderLab [29]. Шейдер має реалізовувати послідовний конвеєр із семи ефектів, серед яких є бочкоподібна дисторсія UV-координат, хроматична аберація через розщеплення каналів RGB, ефект свічення розмиття на основі 9-точкового зваженого семплінгу, модуляція сканлайнами через синусоїдальну функцію, RGB-субпіксельна маска, радіальне віньєтування та корекція яскравості й контрасту. Архітектура рендерингу має реалізовуватись через двокамерний підхід – ігрова камера рендерить у внутрішній RenderTexture з роздільною здатністю 640×360

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

пікселів та режимом фільтрації Point, а окрема відображальна камера на виділеному шарі рендерить квад-меш із CRT-матеріалом поверх ігрового зображення. Параметри шейдера мають зберігатись у файл video_settings.json та відновлюватись при ініціалізації.

Геймплейний шар – кастомна 2D-фізика персонажа, бойова система, система здібностей та ієрархічні кінцеві автомати ворогів і босів – є предметом окремої роботи та безпосередньо не входить до завдань цього проєкту. Завданням у цій площині є забезпечення інтеграції геймплейного шару з візуально-звуковими та сервісними підсистемами виключно через події шини EventBus та системні контракти, зокрема Interactable, без прямих посилань між збірками. Передбачено розробку подієвого компонента PlayerAnimator, що перемикає кліпи SpriteAnimationClip у відповідь на ігрові події без посилань на клас геймплейної логіки PlayerMain, а також обробника звукових реакцій PlayerSfxHandler за аналогічним принципом.

Вагомим інженерним завданням є розробка подієво-орієнтованої системи збереження та завантаження ігрових даних. Процес серіалізації станів у формат JSON має відбуватися автономно через публікацію події CollectingSaveDataEvent, на яку автономні сервіси підписуються самостійно та записують власні дані у відповідні поля контейнера SaveData [15]. Симетрично, завантаження має виконуватись через GameLoadedEvent без прямих викликів між SaveManager та сервісами. Необхідно підтримати три незалежних слоти збереження з метаданими дати та назви сцени, а також окремий файл achievements.json для глобальних досягнень, що є стійким до видалення ігрових слотів. Передбачити поле версіонування schemaVersion у структурі SaveData для коректної обробки файлів, створених попередніми версіями програми. Такий підхід усуває пряму залежність між SaveManager та доменними сервісами, що відповідає принципу інверсії залежностей та спрощує розширення системи збереження без модифікації існуючих компонентів.

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2 Аналіз аналогів програмного продукту

Даний розділ містить опис аналогічних програмних продуктів, що наявні на ринку, із визначенням їх основних функціональних можливостей. У контексті теми кваліфікаційної роботи аналіз існуючих рішень спрямований на дослідження візуальної та звукової складових, підходів до побудови користувацького інтерфейсу (UI), управління ігровою камерою та систем картографування. Серед найбільш показових представників жанру варто детально розглянути відеоігри «Hollow Knight», «Blasphemous» та «Ori and the Blind Forest» [18, 19].

Програмний продукт «Hollow Knight» від студії Team Cherry (рис. 1.1) є однією з найбільш відомих ігор жанру Metroidvania [24]. Особливий інтерес для дослідження становить підхід розробників до візуалізації та інтерфейсу.

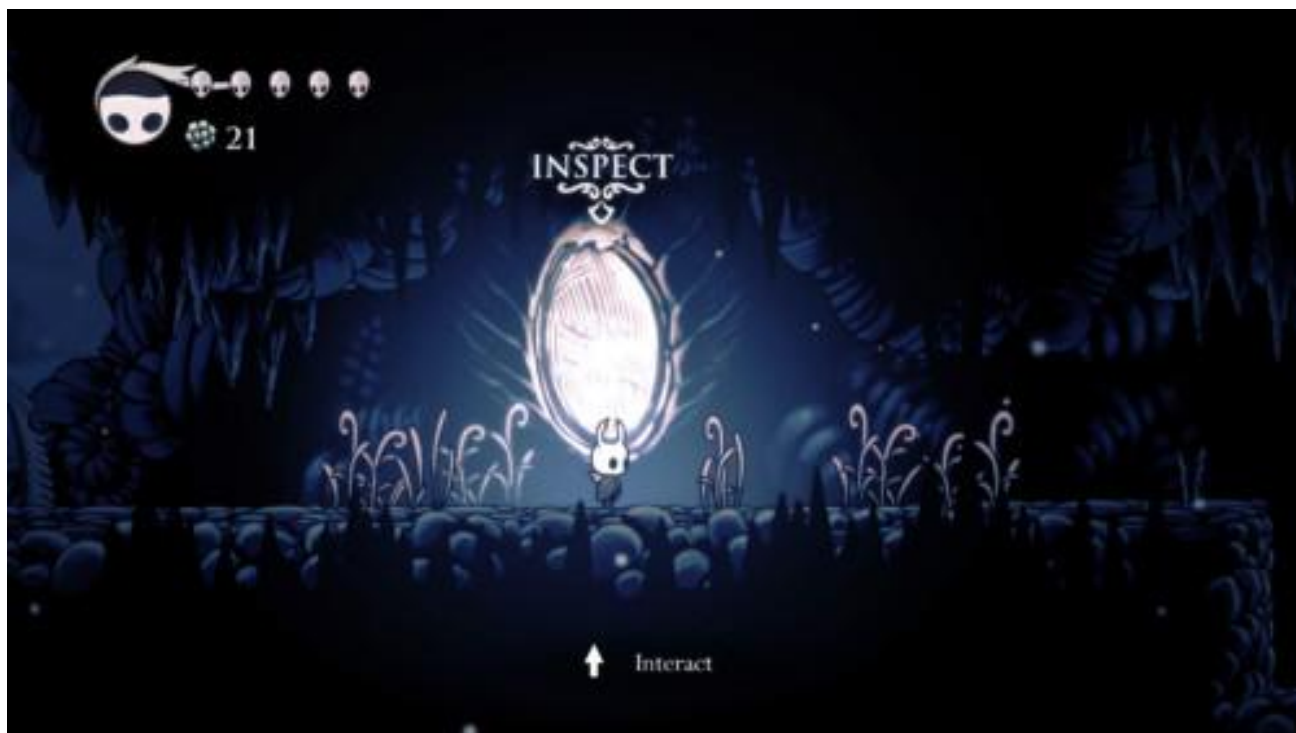


Рис. 1.1 – Ігровий процес та інтерфейс користувача гри «Hollow Knight»

Користувацький інтерфейс гри побудований за принципом мінімалізму (HUD), щоб не відвертати увагу гравця від візуального наповнення світу. Ключовою особливістю є інтерактивна система картографування. Карта не відкривається автоматично, а потребує придбання відповідних предметів (пера та пергаменту) у внутрішньоігрового торговця [21]. Це інтегрує інтерфейсний

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

елемент (карту) безпосередньо в геймплейну прогресію. Візуальна складова камери налаштована на жорстке обмеження (clamping) в межах поточного ігрового екрана, що забезпечує чітке фокусування на локальних аренах. Аудіосистема гри використовує динамічне мікшування, де музичний супровід плавно змінюється залежно від ігрової зони (біому) та напруженості подій (наприклад, перехід до бос-файту).

Іншим вагомим аналогом є програмний продукт «Blasphemous» від студії The Game Kitchen (рис. 1.2). Цей проєкт є особливо релевантним для аналізу, оскільки його візуальна складова повністю виконана у стилістиці PixelArt із застосуванням сучасних постефектів (CRT-фільтри, хроматичні аберації) [25, 29].



Рис. 1.2 – PixelArt стилістика та користувацький інтерфейс гри «Blasphemous»

Система інтерфейсів (UI) у «Blasphemous» чітко розділена на активний HUD та пасивні меню. При відкритті глобальної карти, інвентарю чи екрана досягнень, вікна повністю перекривають ігровий процес та зупиняють внутрішньоігровий час. Аудіодизайн гри підкреслює гнітючу атмосферу через акцент на низькочастотних фонових звуках та різких звукових ефектах під час бойових зіткнень. Окремої уваги заслуговує система сповіщень – розблокування

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

нових предметів чи досягнень супроводжується локальними (toast) сповіщеннями на екрані, які не переривають основний геймплей.

Ще одним класичним представником жанру є гра «Ori and the Blind Forest» від Moon Studios (рис. 1.3) [26]. Хоча візуально гра використовує 3D-моделі у 2D-площині, її підходи до управління камерою та інтеграції UI є еталонними.



Рис. 1.3 – Робота камери та візуальні ефекти у грі «Ori and the Blind Forest»

Цей програмний продукт виділяється складною системою кінематографічної камери. Ігрова камера не просто слідує за гравцем, а використовує механіку випередження (look-ahead) залежно від вектора швидкості персонажа, а також динамічно змінює масштаб (zoom) та додає ефект тремтіння (screen shake) під час масштабних подій чи отримання шкоди. Інтерфейс меню налаштувань є глибоко інтегрованим, дозволяючи користувачеві детально керувати окремими каналами аудіо (Master, Music, SFX) та візуальними ефектами.

Огляд функціоналу зазначених програмних продуктів доводить, що критично важливими елементами для ігор жанру Metroidvania є не лише геймплейні механіки, але й плавна робота камери, оптимізований користувацький інтерфейс (з підтримкою контекстних меню та глобальної карти), а також гнучка система аудіо [23].

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Нововведенням та ключовою відмінністю власної розробки («Fiat Lux») є побудова інтерфейсу та візуальних систем на основі жорсткої архітектурної ізоляції та подієво-орієнтованої комунікації. В межах роботи спроектовано систему UIScreenStack, яка керує видимістю екранів через виклик SetActive(false), зберігаючи об'єкти в пам'яті та цілісність підписок на події [4, 6]. Реалізовано власну систему камери з пріоритетним стеком поведінок (ICameraBehaviour), яка забезпечує піксельно-точне позиціонування (PixelPerfect) без використання плагінів на кшталт Cinemachine. Візуальна стилістика доповнюється кастомним CRT-шейдером із динамічним керуванням ефектами через меню налаштувань, а незалежна архітектура аудіо та системи досягнень працює автономно завдяки власній шині подій.

1.3 Вибір архітектури та архітектурного стилю

Організація програмної архітектури є фундаментальним етапом розробки складних інформаційних систем та інтерактивних додатків. У контексті створення програмного забезпечення гри «Fiat Lux» було відхилено стандартний монолітний підхід, який зазвичай нав'язується специфікою ігрового рушія Unity. Традиційна архітектура Unity передбачає глобальну видимість усіх класів усередині одного проєкту через автоматично сформовану збірку Assembly-CSharp. У середньостроковій перспективі це призводить до появи циклічних залежностей та жорсткого зв'язування між логікою геймплею та візуальним представленням [6]. Характерним прикладом такої проблеми є ситуація, за якої компонент інтерфейсу користувача звертається до поля здоров'я напряду через посилання, а перейменування або переміщення цього поля руйнує відображення HUD. Подібна зв'язність унеможливорює паралельну розробку, ускладнює тестування окремих модулів і перетворює кодову базу на крихку структуру, де локальна зміна викликає помилки в непов'язаних частинах системи.

Для вирішення цієї архітектурної проблеми обрано модульну організацію з використанням механізму Assembly Definitions (asmdef) та суворим застосуванням правила autoReferenced: false для кожної збірки [2]. Це фізично

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

розділяє загальну кодову базу на ізольовані компільовані модулі, де компілятор відстежує дотримання задекларованих залежностей. Будь-яка спроба звернутися до типу зі збірки, яка не вказана у списку залежностей, призводить до помилки компіляції, що робить порушення архітектурних правил технічно неможливим, а не лише умовним.

В основі спроектованої системи лежить багаторівнева архітектура (Layered Architecture), яка забезпечує строгий потік даних та розмежування зон відповідальності [2]. Фундаментальним шаром є FiatLux.Core – модуль без жодних зовнішніх залежностей, що містить базові структури даних, статичні утиліти та реалізацію шини подій EventBus. Над ним розміщується шар контрактів FiatLux.Contracts, який акумулює всі інтерфейси взаємодії між модулями – зокрема Interactable та ICameraBehaviour – а також структури системних подій, що реалізують маркерний інтерфейс IEvent. Цей шар не містить жодної логіки та слугує єдиним спільним словником типів для всієї системи.

Наступним рівнем виступає шар сервісів FiatLux.Services.*, що об'єднує автономні підсистеми – AudioManager (управління звуковим супроводом), SaveManager (файлове введення-виведення збережень у форматі JSON), InventoryService (облік предметів), MapDiscoveryService (фіксація досліджених зон) та AchievementService (обробка глобальних досягнень). Кожен сервіс має право посилатися лише на шари Core та Contracts, що унеможливорює появу зворотних або горизонтальних залежностей між сервісами. Вище за ієрархією розташовані просторовий шар FiatLux.World, відповідальний за систему кімнат та процедурну генерацію рівня, та геймплейні шари FiatLux.Gameplay.Player і FiatLux.Gameplay.Enemies, що реалізують математику ігрового процесу. Вершиною цієї структури є візуальний шар FiatLux.Visual та інтерфейсний FiatLux.UI. Геймплейний стан передається до візуальної частини виключно через публікацію подій, що повністю виключає можливість некоректної зміни бізнес-логіки з боку графічного інтерфейсу. Точкою фінального зв'язування всіх підсистем є збірка FiatLux.Bootstrap, якій надані посилання на всі шари та яка відповідає за детерміновану ініціалізацію в рантаймі.

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

Центральним архітектурним патерном взаємодії між ізольованими модулями обрано Publish-Subscribe, реалізований через власну систему – типізований статичний клас EventBus з узагальненими методами з обмеженням where T : IEvent [3, 4]. При виборі рішення порівнювались три підходи. Стандартні UnityEvent вимагають серіалізованого зв'язування через інспектор рушія, що створює неявні залежності між об'єктами та унеможлиблює зв'язок між збірками з різних шарів. Канали подій на основі ScriptableObject усувають частину цих проблем, але вимагають створення окремого ресурсного файлу для кожного типу події та не забезпечують суворої типізації на рівні компіляції. Власна реалізація EventBus через Dictionary<Type, Delegate> забезпечує строгу типізацію, нульові накладні витрати на рефлексію при виклику та повну незалежність від інспектора. Важливою перевагою є реалізація механізму PublishLatest, що зберігає останнє значення кожного типу події. Це забезпечує коректну ініціалізацію компонентів, що активуються вже після фактичної публікації події – зокрема, CameraController отримує межі активної кімнати через SubscribeWithLatest одразу при включенні, не чекаючи наступного переходу. Для запобігання витокам підписок між ігровими сесіями реалізовано автоматичне скидання стану шини через атрибут RuntimeInitializeOnLoadMethod при кожному старті Play Mode.

Важливим стратегічним рішенням є свідомо відмова від нав'язаних інструментальних патернів Unity (Unity-imposed patterns) – CharacterController, Cinemachine та вбудованого Animator у ролі контролера поведінки. Використання цих пакетів перетворює процес розробки на конфігурування закритих алгоритмів, що функціонують як «чорні скриньки» та обмежують інженерний контроль над циклом виконання. Натомість для управління камерою реалізовано власний CameraController на основі пріоритетного списку поведінок, що реалізують інтерфейс ICameraBehaviour. Цей підхід відповідає принципу відкритості-закритості, де для додавання нової поведінки камери достатньо реалізувати інтерфейс та зареєструвати об'єкт у контролері без будь-яких змін у наявному коді. Аналогічна логіка застосована до компонента PlayerAnimator,

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

який прослуховує події шини та самостійно перемикає анімаційні кліпи `SpriteAnimationClip`, не маючи жодного посилання на клас `PlayerMain`.

Окремим архітектурним рішенням є застосування паттерну «Адаптер» у підсистемі процедурної генерації рівнів. Алгоритм компонування ігрового простору цілеспрямовано виокремлено в чистий C#-клас `LevelLayoutCore`, що не містить жодного посилання на простір імен `UnityEngine`. Конфігурація передається через незмінне значення-тип `LevelLayoutConfig` (readonly struct), а результат повертається як незмінний об'єкт `LevelLayoutResult` з колекціями `ReadOnlyList`. `MonoBehaviour`-клас `LevelGenerator` виступає єдиним адаптером між інспектором рушія та алгоритмом. Він збирає серіалізовані поля, ініціює обчислення в ядрі та виконує Unity-специфічну роботу з отриманим результатом. Таке розмежування гарантує можливість запуску та перевірки алгоритму поза ігровою сценою та робить зміни в параметрах генератора безпечними для архітектурної цілісності системи [10].

Для системи інтерфейсу користувача обрано паттерн стека з класом `UIScreenStack` на основі `Stack<UIScreen>`. Це рішення вирішує проблему порядку навігації між екранами та відновлення стану – при поверненні назад завжди активується саме той екран, що був відкритий перед поточним, незалежно від послідовності викликів. Кожна операція `Push` та `Pop` публікує відповідну подію через `EventBus`, що дозволяє зовнішнім підсистемам реагувати на зміну стану інтерфейсу без прямих посилань на стек.

Обраний архітектурний стиль підтримується підходом `Bootstrap Scene` з використанням ієрархії `DontDestroyOnLoad`. При запуску програми єдина стартова сцена ініціалізує всі глобальні сервіси в суворо визначеному порядку до завантаження будь-якої ігрової локації. Це усуває клас недетермінованих помилок, пов'язаних із зверненням до неініціалізованого сервісу в методі `Awake` геймплейного об'єкта. Для зручності розробки реалізовано інструмент `EditorSceneAutoBootstrap`, який автоматично завантажує `Bootstrap`-сцену при запуску `Play Mode` з будь-якої іншої сцени редактора, гарантуючи коректну ініціалізацію незалежно від робочого контексту розробника.

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Головним інструментальним середовищем для розробки програмного продукту обрано кросплатформний ігровий рушій Unity версії 6000.0.33f1 [5]. Вибір обґрунтовано порівняльним аналізом доступних альтернатив. Рушій Godot 4, незважаючи на відкритий вихідний код та невибагливість до апаратних ресурсів, не забезпечує механізму Assembly Definitions, необхідного для фізичної ізоляції модулів на рівні компіляції, а його GDScript-екосистема значно поступається C# за зрілістю бібліотечного оточення. Рушій Unreal Engine 5 орієнтований переважно на тривимірні проекти із фотореалістичним рендерингом, вимагає суттєво більших апаратних ресурсів і тривалого часу перекомпіляції шейдерів, що є неприйнятним для проекту з акцентом на пікселізовану двовимірну графіку. Unity 6000.0.33f1 у поєднанні з URP забезпечує оптимальний баланс між архітектурною гнучкістю та ефективністю двовимірного рендерингу. Висока стабільність цієї збірки, глибока оптимізація алгоритмів відтворення спрайтів та підтримка розширеного керування архітектурою через систему Assembly Definitions визначили її як єдине доцільне рішення в контексті поставлених інженерних завдань.

Основною мовою програмування є C# відповідно до стандарту .NET Standard 2.1 [12, 14]. Вибір мови обґрунтовано не лише загальною підтримкою з боку рушія, а й конкретними мовними конструкціями, що безпосередньо використовуються в архітектурі проекту. Узагальнені типи (generics) з обмеженням where T : IEvent є основою реалізації типізованої шини подій EventBus, де компілятор перевіряє коректність усіх підписок на етапі компіляції, унеможливаючи помилки типів у рантаймі. Делегати та Delegate.Combine застосовуються для побудови мультикаст-ланцюгів обробників без виділення додаткових об'єктів у керованій купі. Структури зі специфікатором readonly, зокрема LevelLayoutConfig та внутрішній тип FVec2, забезпечують передачу незмінних конфігураційних даних між шарами без ризику побічних змін. Вирази switch, доступні починаючи з C# 8.0, використовуються в алгоритмі розміщення

		Львів І.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

кімнат для компактного опису геометричних перетворень залежно від напрямку переходу. Строга типізація мови, підтримка дженериків та оптимізований збирач сміття дозволяють реалізувати складні архітектурні патерни – зокрема шину подій та ієрархічні кінцеві автомати – без суттєвих втрат продуктивності.

Для управління процесом візуалізації застосовано Universal Render Pipeline (URP) [27]. Вбудований рендерний конвеєр Unity (Built-in Pipeline) не підтримує ефективного сумісного використання Pixel Perfect Camera та кастомних ShaderLab-матеріалів без значних доопрацювань, а також поступається URP за швидкістю пакетної обробки двовимірних спрайтів при наявності кількох активних матеріалів. У рамках обраного пайплайну реалізовано двокамерну архітектуру рендерингу – ігрова камера відображає сцену у внутрішню RenderTexture із роздільною здатністю 640×360 пікселів та режимом фільтрації Point, що унеможливорює лінійну інтерполяцію між пікселями і зберігає піксельну чіткість зображення, а окрема відображальна камера на виділеному шарі накладає CRT-постефект поверх кадру через квад-меш із кастомним HLSL-шейдером у форматі ShaderLab [29]. Параметр 16 PPU (Pixels Per Unit) встановлено глобально для всього атласу спрайтів, що гарантує ідеальне піксельне вирівнювання ігрових об'єктів на сцені без субпіксельного зміщення. Шейдер CRT реалізує послідовний конвеєр із семи ефектів – бочкоподібна дисторсія, хроматична аберація, ефект свічення, сканлайни, субпіксельна маска, віньєтування та корекція яскравості – і керується параметрами, що зберігаються у файлі video_settings.json та відновлюються при кожній ініціалізації.

Обробка введення даних реалізована за допомогою оновленого пакета Unity Input System [28]. На відміну від застарілої системи, що базується на синхронному опитуванні стану пристрою в кожному кадрі, новий пакет реалізує асинхронну подієву модель на основі файлу InputActionAsset з описом дій та прив'язок до конкретних фізичних кнопок. Конфігурація охоплює дві схеми – Keyboard і Gamepad, – між якими система автоматично перемикається при підключенні або від'єднанні пристрою без перезапуску гри. Такий підхід є критичним для забезпечення мінімальної затримки введення в бойовій системі

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

та для коректного динамічного оновлення підказок щодо керування у HUD-інтерфейсі. Управління конфігураціями рівнів, шаблонами генерації та бібліотеками ресурсів реалізовано через ScriptableObject, що відповідає принципам data-driven design. На відміну від MonoBehaviour-компонентів, ScriptableObject існує як окремий файл-ресурс проєкту і не прив'язаний до жодного об'єкта на сцені, що дозволяє використовувати єдиний екземпляр шаблону кімнати RoomTemplate для всіх інстанцій відповідного типу без дублювання даних. Збереження ігрового прогресу забезпечується серіалізацією у текстовий формат JSON за допомогою вбудованого JsonUtility [15]. Порівняно зі сторонніми бібліотеками Newtonsoft.Json та System.Text.Json, JsonUtility не використовує рефлексію під час серіалізації, що знижує ризик помилок при АОТ-компіляції у режимі IL2CPP та не вносить додаткових залежностей у проєкт.

Редагування вихідного коду здійснюється в середовищі Visual Studio Code (VSC) із встановленим розширенням C# Dev Kit [32]. VSC забезпечує підсвітку синтаксису C#, автодоповнення на основі мовного сервера OmniSharp, навігацію по символах між збірками та вбудований термінал для виконання команд Git. Інтеграція з Unity реалізується через офіційне розширення Unity для VSC, яке налаштовує автоматичну регенерацію файлів рішення (.sln) та проєктів (.csproj) при зміні структури збірок asmdef. Версйонування вихідного коду та ресурсів забезпечується системою Git версії 2.30 та вище з розміщенням репозиторію на GitLab-інстанції університету. Файл .gitignore налаштовано з виключенням директорій Library, Temp та Build, що містять виключно автоматично згенеровані дані рушія та не мають цінності для версйонування.

Для створення графічних ресурсів у стилі Pixel Art використовується Aseprite [30] та Smack Studio [31].

Щодо вимог до апаратного забезпечення, розроблений програмний продукт оптимізовано для стабільної роботи на системах середнього та базового рівнів продуктивності. Мінімальні вимоги передбачають наявність процесора архітектури x64 з тактовою частотою не менше 2.0 ГГц та 8 ГБ оперативної пам'яті. Зазначений обсяг пам'яті зумовлений особливостями реалізованої

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

системи Persistent Room System, яка утримує повний масив кімнат ігрового рівня в оперативній пам'яті для забезпечення миттєвих переходів між зонами без звернення до диску. Відеоадаптер має підтримувати API DirectX 11 або OpenGL 4.5 для коректної роботи алгоритмів URP та рендерингу постефектів CRT-шейдера. Вільний обсяг дискового простору у розмірі не менше 2 ГБ необхідний для розміщення виконуваного файлу гри, бібліотек рушія та файлів персистентних даних гравця. Детальні мінімальні та рекомендовані вимоги у табличній формі наведено в підрозділі 3.1 цієї роботи.

Висновки до розділу 1

У першому розділі було проведено системний аналіз предметної області та визначено ключові завдання для розробки програмного забезпечення двовимірної гри в жанрі Metroidvania під назвою «Fiat Lux». Сформульовано перелік технічних задач, які вимагають проєктування слабко зв'язаної архітектури, створення кастомної моделі двовимірної фізики персонажа, розробки систем управління простором камери та детермінованої генерації рівнів.

Аналіз існуючих аналогів дозволив дослідити найефективніші ігрові механіки, просторові рішення та підходи до організації штучного інтелекту ворогів. На основі проведеного порівняння обґрунтовано необхідність відмови від стандартних високорівневих компонентів ігрового рушія, які приховують логічні обчислення, на користь повністю прозорих, власноруч спроектованих систем.

Вибір архітектурного стилю, що базується на модульній ізоляції шарів за допомогою asmdef та подієво-орієнтованій комунікації через власну шину EventBus, забезпечує високий рівень безпеки розробки, легке масштабування та жорстке розмежування ігрової логіки й візуального представлення.

Обґрунтовано вибір інструментальних засобів, серед яких ключовими є рушій Unity, URP для оптимізації рендерингу, нова система управління введенням Input System, об'єкти ScriptableObject для управління конфігураціями

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

та текстовий формат JSON для збереження даних. Сформовані вимоги до апаратного забезпечення підтверджують ефективність та оптимізованість розроблених інженерних алгоритмів.

		Ільїн І.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ 2D ІГРОВОГО ДОДАТКУ ЖАНРУ METROIDVANIA

2.1 Варіанти використання та об'єктно-орієнтована структура системи

Проектування програмного забезпечення для сучасних інтерактивних систем зі складною внутрішньою логікою вимагає початкового моделювання функціональних вимог та визначення меж взаємодії між користувачем і системою [21]. У рамках розробки відеогри «Fiat Lux» ключовим інструментом аналізу поведінки системи на високому рівні абстракції є побудова моделей варіантів використання Use-Case (рис. 2.1). Цей підхід дозволяє формалізувати вимоги до ігрового процесу, визначити ролі дійових осіб (акторів) та зафіксувати послідовності виконання операцій, які повинна забезпечувати програма. Варіанти використання є також основою для формування тестових сценаріїв, де кожен основний хід виконання та альтернативна гілка безпосередньо відповідають тест-кейсу в плані тестування системи.

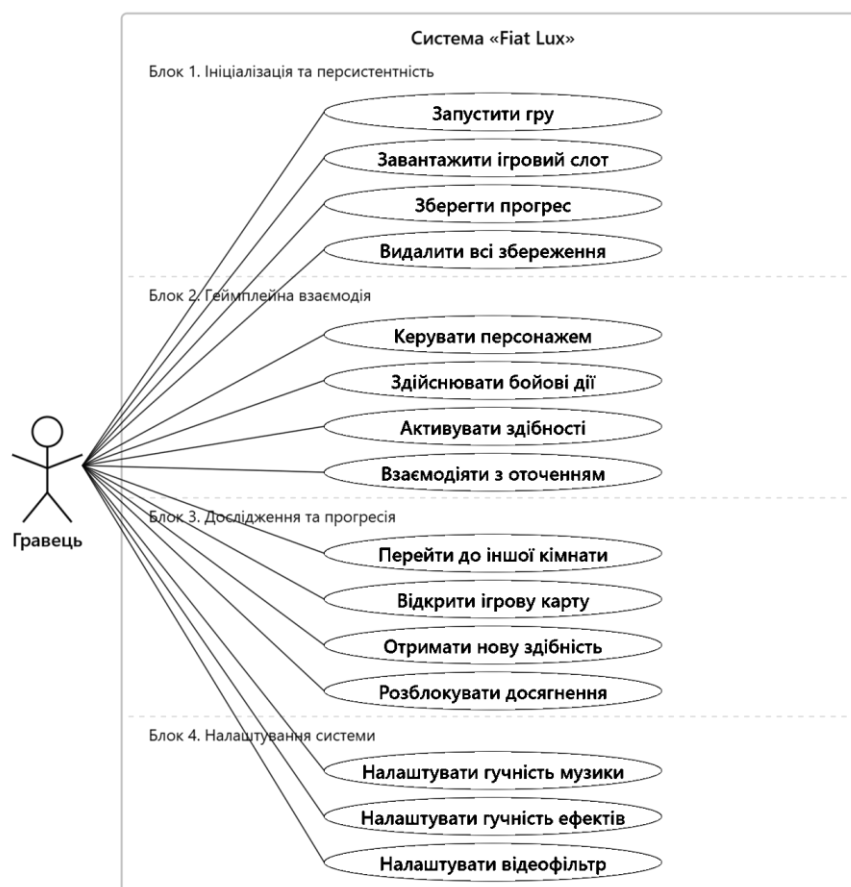


Рис. 2.1 – Діаграма варіантів використання системи

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				26
Змн.	Арк.	№ докум.	Підпис	Дата		

Головним зовнішнім актором у системі є Гравець. Його взаємодія з програмним продуктом охоплює чотири функціональних блоки, які разом формують повний спектр варіантів використання.

Перший блок стосується управління процесом ініціалізації та персистентності даних. Сюди входять варіанти використання «Запустити гру», «Завантажити наявний ігровий слот» та «Видалити слот збереження». Програма має забезпечувати коректну обробку цих запитів через відповідні інтерфейсні меню, ініціюючи зчитування та десеріалізацію файлів конфігурації. Основний хід виконання варіанту «Завантажити наявний ігровий слот» передбачає вибір слота в меню, перевірку існування файлу `save_slot_N.json`, десеріалізацію об'єкта `SaveData`, відновлення сцени за полем `sceneName` та публікацію події `GameLoadedEvent`. Альтернативна гілка – відсутній або пошкоджений файл слота – завершується показом повідомлення про помилку без виходу з меню. Для нового профілю генерується унікальний `seed` рівня, що зберігається у `SaveData` та гарантує відтворюваність процедурно згенерованого світу при повторному завантаженні.

Другий функціональний блок описує безпосередню геймплейну взаємодію в просторі ігрового світу. Основними варіантами є «Керувати переміщенням персонажа», «Здійснювати бойові дії», «Активувати здібності» та «Взаємодіяти з об'єктами оточення». Система опрацьовує події введення в реальному часі через `Unity Input System` із подієвими зворотними викликами. Варіант «Взаємодіяти з об'єктами оточення» реалізовано через інтерфейс `Interactable`. Гравець ініціює взаємодію, система виконує `Physics2D.OverlapCircleAll` із фільтрацією по шару та перевіряє наявність компонента, що реалізує інтерфейс. Це дозволяє однаково обробляти скрині, точки збереження та тригери здібностей без знання конкретного типу об'єкта.

Третій блок охоплює варіанти використання, пов'язані з дослідженням ігрового світу та прогресією. Сюди входять «Перейти до іншої кімнати», «Відкрити ігрову карту», «Отримати нову здібність» та «Розблокувати досягнення». Варіант «Перейти до іншої кімнати» ініціюється при перетині

		Львів І.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Лобанчикова Н.М.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

колайдера гравця з тригерною зоною RoomTeleport та реалізується як корутинна послідовність із затемненням екрана, переміщенням персонажа та оновленням активної кімнати через RoomManager. Варіант «Відкрити ігрову карту» завантажує екран MapScreen у UIScreenStack та відображає всі зареєстровані кімнати, позначаючи відвідані через MapDiscoveryService. Варіант «Розблокувати досягнення» є системним і не ініціюється гравцем напряму – AchievementService відстежує виконання умов через підписку на ігрові події та автономно фіксує результат у глобальному файлі achievements.json.

Четвертий блок стосується налаштування параметрів системи. Варіанти «Налаштувати гучність музики», «Налаштувати гучність ефектів» та «Налаштувати параметри відеофільтру» реалізуються через відповідні екрани налаштувань у UIScreenStack. AudioManager зберігає значення гучності через JSON-серіалізацію, CRTRenderer – через окремий файл video_settings.json. Обидва відновлюють налаштування при наступному запуску в методі Awake до будь-яких рендеринг-операцій.

Внутрішня об'єктно-орієнтована структура системи проєктується на основі принципів SOLID, з особливим акцентом на інверсії залежностей та принципі відкритості-закритості [7]. На відміну від традиційного для Unity підходу, де логіка та візуалізація змішуються всередині монолітних MonoBehaviour-класів, у розроблюваній системі впроваджено чітке розподілення на три категорії, а саме об'єкти даних, об'єкти логіки та компоненти представлення.

Об'єкти даних представлені незмінними або строго контрольованими структурами. SaveData – серіалізований клас, що містить поля для всіх підсистем, які потребують персистентності. LevelLayoutConfig – readonly struct, що передає параметри генератору рівнів без можливості модифікації після створення. LevelLayoutResult – незмінний об'єкт із колекціями IReadOnlyList, що повертає результат алгоритму генерації. RoomTemplate – ScriptableObject, що визначає тип кімнати, розмір у тайлах, конфігурацію виходів та посилання на префаб. Використання ScriptableObject для шаблонів кімнат відповідає підходу

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

data-driven design, де дизайнер може модифікувати параметри рівня без зміни коду та без повторної компіляції збірок.

Об'єкти логіки реалізовано переважно як чисті C#-класи без успадкування від MonoBehaviour. LevelLayoutCore виконує всі обчислення генерації рівня без залежностей від UnityEngine. FollowBehaviour та CutsceneBehaviour реалізують інтерфейс ICameraBehaviour та інкапсують алгоритми слідування та фіксації камери відповідно. CameraShake реалізує trauma-based модель тремтіння з шумом Перліна як ізолюваний обчислювальний об'єкт.

Компоненти представлення – MonoBehaviour-класи – відповідають лише за зв'язок з рушієм та делегують логіку об'єктам нижніх шарів. CameraController підтримує список ICameraBehaviour та делегує обчислення позиції поведінки з найвищим пріоритетом. LevelGenerator збирає параметри з інспектора, передає їх у LevelLayoutCore та виконує Unity-специфічну роботу з отриманим результатом.

Об'єктна структура базується на суворому контрактному програмуванні через шар FiatLux.Contracts. Будь-яка сутність, що потребує взаємодії з іншими модулями, описується відповідним інтерфейсом. Interactable дозволяє логічному модулю гравця викликати методи взаємодії без знання конкретного типу об'єкта. ICameraBehaviour з методами CalculateTargetPosition та OnActivated дозволяє CameraController комбінувати довільні поведінки без зміни власного коду. IEvent є маркерним інтерфейсом, що визначає контракт системи обміну повідомленнями – будь-яка структура або клас, що реалізує цей інтерфейс, автоматично стає учасником шини подій. Сервісні класи InventoryService, MapDiscoveryService та AchievementService функціонують як автономні обчислювальні блоки – вони оперують виключно власним внутрішнім станом та взаємодіють із зовнішніми системами лише через підписку на події та публікацію результатів, не маючи жодних прямих посилань на класи геймплею або компоненти інтерфейсу.

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

2.2 Архітектура підсистем та міжмодульна взаємодія

Архітектурна організація програмного продукту «Fiat Lux» базується на концепції жорсткого розмежування зон видимості коду на рівні компіляції. Для реалізації цієї концепції весь простір проєкту було розділено на ізольовані модулі за допомогою механізму Assembly Definitions (asmdef) і суворого відключення автоматичного створення посилань шляхом встановлення прапора `autoReferenced: false` для кожної збірки. Це архітектурне рішення повністю унеможливорює ситуацію, за якої розробник може випадково створити перехресне або циклічне посилання між компонентами різних рівнів, перетворюючи архітектуру на монолітну структуру, яку важко рефакторити.

Вектор залежностей між створеними збірками спрямований виключно знизу вгору. Нижній, найбільш фундаментальний шар системи представлений збіркою `FiatLux.Core`. Цей модуль не має жодних зовнішніх залежностей і містить у собі базові типи даних, статичні утиліти та безпосередню реалізацію підсистеми обміну повідомленнями. Над ним розташовується шар `FiatLux.Contracts`, який посилається лише на базовий модуль і містить усі структури подій, що реалізують маркерний інтерфейс `IEvent`, а також загальносистемні інтерфейси персистентності та взаємодії.

Проміжний рівень займають сервісні збірки, об'єднані під простором імен `FiatLux.Services.*`. Сюди входять автономні підсистеми, серед них є `AudioManager` (управління звуковим супроводом), `SaveManager` (файлове введення-виведення збережень), `InventoryService` (облік предметів), `MapDiscoveryService` (фіксація досліджених зон мапи) та `AchievementService` (обробка глобальних досягнень). Ці сервіси мають право посылатися лише на шари контрактів та ядра.

Верхній рівень архітектури займають модулі логіки світу (`FiatLux.World`) та безпосереднього геймплею (`FiatLux.Gameplay.Player` і `FiatLux.Gameplay.Enemies`). Візуальний шар (`FiatLux.Visual`) та шар користувацького інтерфейсу (`FiatLux.UI`) винесені в окремі збірки, які мають доступ лише до необхідних сервісних модулів через контрольовані залежності.

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

Пряма взаємодія або виклик методів із геймплейних модулів у візуальні чи інтерфейсні шари суттєво обмежені та контролюються через систему asmdef-залежностей. Точкою збірки проєкту виступає модуль FiatLux.Bootstrap, який володіє посиланнями на всі підсистеми і відповідає за їхнє фінальне зв'язування в рантаймі.

Структура залежностей між збірками проєкту наведена у таблиці 2.1.

Таблиця 2.1

Структура залежностей проєкту Fiat Lux

Збірка	Залежності	Призначення
FiatLux.Core	-	EventBus, базові утиліти, маркерний інтерфейс IEvent
FiatLux.Contracts	Core	Структури подій, інтерфейси IInteractable, ICameraBehaviour, класи SaveData, SaveSlotInfo
FiatLux.Services.SaveLoad	Core, Contracts	SaveManager – файловий I/O JSON-слотів
FiatLux.Services.Audio	Core, Contracts	AudioManager – канали Music/SFX, JSON-серіалізація гучності
FiatLux.Services.Inventory	Core, Contracts	InventoryService – облік предметів
FiatLux.Services.Map	Core, Contracts	MapDiscoveryService – фіксація відкритих кімнат
FiatLux.Services.Achievement	Core, Contracts	AchievementService – глобальний файл achievements.json
FiatLux.Services.Camera	Core, Contracts	CameraController, ICameraBehaviour, FollowBehaviour, CutsceneBehaviour, CameraShake, CRTRenderer
FiatLux.World	Core, Contracts, Services.*	RoomManager, RoomTeleport, LevelGenerator, LevelLayoutCore, LevelLayoutConfig
FiatLux.UI	Core, Contracts, Services.*	UIScreenStack, UIScreen та всі HUD-в'юхи
FiatLux.Visual	Core, Contracts, Services.*	PlayerAnimator, PlayerSfxHandler – підписуються на події, не мають посилань на Gameplay
FiatLux.Gameplay.Player	Core, Contracts	PlayerMain, кастомна фізика, бойова система, HFSM гравця
FiatLux.Gameplay.Enemies	Core, Contracts	HFSM ворогів, спавнери, боси
FiatLux.Bootstrap	Всі вищезазначені	Єдина точка входу; ініціалізує сервіси в детермінованому порядку

Оскільки прямі посилання між більшістю модулів відсутні, міжмодульна взаємодія та синхронізація станів реалізована за допомогою кастомної подієвої шини – типізованої статичної шини подій з узагальненими методами where T :

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

IEvent. Вибір дженерик-реалізації дозволив ізолювати списки підписників для кожного конкретного типу події на рівні виконання, що забезпечує швидкість диспетчеризації повідомлень за час $O(n)$, де n – кількість підписників на конкретну подію (у межах проєкту цей показник зазвичай не перевищує 10 об'єктів) [9, 17]. Подієва шина функціонує виключно в головному потоці Unity (Main Thread), що усуває необхідність у ресурсомістких блокуваннях потоків (thread-safety локах) і гарантує відсутність додаткових алокацій пам'яті, окрім передачі копії самої структури дані події.

Для забезпечення гнучкої взаємодії між підсистемами в реалізації EventBus передбачено кілька механізмів обміну повідомленнями. Базовим способом є метод Publish(T e), який використовується для одноразового сповіщення всіх зареєстрованих підписників про виникнення події. Такий підхід застосовується для подій, стан яких не потребує збереження після обробки, наприклад PlayerDiedEvent.

Для подій, що відображають поточний стан системи, реалізовано механізм кешування останнього повідомлення через метод PublishLatest(T e). Після публікації система зберігає останній екземпляр події, що дозволяє новим компонентам отримати актуальний стан без очікування наступного оновлення. Подібний механізм використовується для подій HealthChangedEvent та ActiveRoomChangedEvent.

Отримання останнього збереженого стану підтримується двома способами. Метод SubscribeWithLatest(Action<T> callback) дозволяє компоненту підписатися на майбутні зміни та одночасно отримати останнє закешоване значення відразу після реєстрації. Такий підхід використовується елементами інтерфейсу користувача, які можуть створюватися або активуватися після запуску гри. Крім того, метод TryGetLatest(out T latestEvent) надає можливість одноразового безпечного зчитування поточного стану під час ініціалізації компонента без створення постійної підписки на подію.

Для запобігання витокам пам'яті в середовищі редактора Unity при багаторазовому перезапуску режиму тестування (Play Mode) в EventBus

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				32
Змн.	Арк.	№ докум.	Підпис	Дата		

інтегровано механізм автоматичного очищення статичних списків підписників. За допомогою атрибута [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.SubsystemRegistration)] викликається внутрішній метод Clear(), який гарантує повне скидання стану шини до моменту спрацьовування методів Awake будь-яких об'єктів на сцені.

Важливим елементом архітектури є організація життєвого циклу програми та детермінізм ініціалізації систем, що реалізовано через підхід Bootstrap Scene. У проєкті створено єдину стартову сцену, яка містить об'єкт Bootstrap із точкою входу. При запуску програми цей модуль динамічно створює ієрархію менеджерів сервісів, які позначаються компонентом DontDestroyOnLoad, що запобігає їх знищенню при зміні просторових локацій. Для зручності розробки в інспекторі налаштовано кастомний інструмент EditorSceneAutoBootstrap, який автоматично завантажує сцену ініціалізації в рантаймі, навіть якщо розробник натиснув кнопку запуску, перебуваючи на будь-якій іншій технологічній чи геймплейній сцені. Це гарантує, що всі глобальні сервіси (зокрема SaveManager та AudioManager) завжди будуть ініціалізовані коректно і в строго визначеному порядку.

Структура даних, що підлягає персистентності, визначається серіалізованим класом SaveData у шарі контрактів FiatLux.Contracts.Data. Клас позначено атрибутом [Serializable] для сумісності з JsonUtility. Поля розподілено на п'ять логічних груп відповідно до підсистем, що їх заповнюють через подію CollectingSaveDataEvent. Опис полів наведено у таблиці 2.2.

Таблиця 2.2

Структура класу SaveData

Поле	Тип	Значення за замовч.	Призначення
Метадані файлу			
version	string	"1.0"	Версія формату для міграції при оновленнях
timestampUtc	long	-	Час збереження у тіках UTC (DateTime.UtcNow.Ticks)

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				33
Змн.	Арк.	№ докум.	Підпис	Дата		

Чекпоінт та рівень			
levelSeed	int	-	Seed генератора рівня; той самий seed відтворює ідентичний світ
sceneName	string	-	Назва активної сцени для завантаження
roomId	string	-	Ідентифікатор кімнати точки появи
checkpointX	float	-	X-координата точки появи у світових одиницях
checkpointY	float	-	Y-координата точки появи у світових одиницях
Розблоковані здібності			
dashUnlock	bool	false	Ривок (Dash)
chainUnlock	bool	false	Ланцюг (Chain)
bowUnlock	bool	false	Лук (Bow)
magnetPushUnlock	bool	false	Магнітний поштовх
magnetPullUnlock	bool	false	Магнітний притяг
fireUnlock	bool	false	Вогняна здібність
iceUnlock	bool	false	Льодяна здібність
elementalPresetUnlock	bool	false	Елементальний пресет здібностей
activePreset	int	0	Активний пресет (приведення до enum AbilityPreset)
Магнітні покращення			
magnetAbsorbUnlock	bool	false	Магнітне поглинання снарядів
magnetAccumulateUnlock	bool	false	Магнітне накопичення заряду
magnetFrugalUnlock	bool	false	Знижений витрата снарядів
projectileReserve	int	-	Загальний резерв снарядів (магазин + запас)
Характеристики персонажа			
maxHealth	int	6	Базове максимальне здоров'я
maxEnergy	int	6	Базова максимальна енергія
syringeCount	int	3	Кількість лікувальних шприців
healthMaxBonus	float	0	Бонус до максимального здоров'я від апгрейдів

energyMaxBonus	float	0	Бонус до максимальної енергії від апгрейдів
healPowerBonus	float	0	Бонус до ефективності лікування
Колекції			
inventoryItems	List<string>	порожній	Ідентифікатори предметів інвентарю
discoveredRooms	List<DiscoveredRoomEntry>	порожній	Відкриті кімнати з координатами центру
activatedCheckpoints	List<string>	порожній	Ідентифікатори активованих чекпоінтів

Вкладений серіалізований клас DiscoveredRoomEntry описує одну відкриту кімнату та використовується підсистемою MapDiscoveryService для відображення дослідженого графа на ігровій карті.

Таблиця 2.3

Структура класу DiscoveredRoomEntry

Поле	Тип	Призначення
roomId	string	Унікальний ідентифікатор кімнати, відповідає полю Room.RoomId
centerX	float	X-координата центру кімнати у світовому просторі
centerY	float	Y-координата центру кімнати у світовому просторі

Використання двох окремих float-полів замість Vector2 зумовлено обмеженням JsonUtility – серіалізатор Unity не підтримує тип Vector2 безпосередньо у вкладених класах без атрибуту [Serializable]. Обчислювана властивість Center відновлює Vector2 з цих полів у рантаймі без накладних витрат на зберігання.

2.3 Реалізація ключових підсистем

У процесі розробки програмного продукту особлива увага приділялася проєктуванню автономних підсистем, які взаємодіють між собою без утворення жорстких зв'язків. Одним із найважливіших архітектурних рішень стала реалізація подієво-орієнтованої підсистеми збереження та завантаження ігрових даних (Save/Load System). У класичних підходах клас-менеджер збережень зазвичай має прямі посилання на всі сервіси, які потребують персистентності, що

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				35
Змн.	Арк.	№ докум.	Підпис	Дата		

призводить до порушення принципу відкритості/закритості (Open/Closed Principle) при додаванні нових модулів. Для уникнення цієї проблеми у проєкті реалізовано підхід, за якого SaveManager відповідає виключно за файлове введення-виведення даних у форматі JSON за допомогою вбудованого інструментарію JsonUtility [15]. Будь-який сервіс, чий стан необхідно зберегти (наприклад, інвентар або карта), підписується на подію CollectingSaveDataEvent. Під час ініціалізації збереження менеджер публікує цю подію, а всі підписники автономно записують свої дані у виділені блоки загального об'єкта SaveData, після чого серіалізований стан записується у відповідний файл слота. Процес завантаження функціонує у зворотному порядку – після зчитування файлу публікується кешована (sticky) подія ApplyingSaveDataEvent. Оскільки методи ініціалізації об'єктів на сцені (Awake та OnEnable) виконуються в недетермінованому порядку, застосування завантажених даних може викликати конфлікти. Для розв'язання цієї проблеми впроваджено патерн буферизації _pendingApply у класі GameManager, який затримує обробку події завантаження до моменту повного завантаження сцени (спрацьовування OnSceneLoaded), що гарантує готовність усіх компонентів типу MonoBehaviour до прийому нових даних. Окремо реалізовано збереження прогресу здобутків через AchievementService, який записує стан у незалежний файл achievements.json, що робить розблоковані досягнення глобальними та стійкими до видалення локальних ігрових слотів гравця.

Проєктування інтерфейсу користувача (UI) потребувало відмови від традиційного для ігрових рушіїв підходу до керування видимістю вікон шляхом виклику методу SetActive(false). Фізична деактивація ігрових об'єктів повністю зупиняє життєвий цикл MonoBehaviour, що автоматично розриває всі підписки на системні події, спричиняючи десинхронізацію даних між логікою гри та візуальним відображенням (наприклад, неможливість оновити смужку здоров'я на прихованому екрані паузи). Для усунення цього недоліку розроблено архітектуру UIScreenStack на основі стекової структури Stack<UIScreen>, що керує відображенням екранів через методи OnPush, OnPop, OnHide та OnReveal

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				36
Змн.	Арк.	№ докум.	Підпис	Дата		

із відновленням фокусу введення через EventSystem при кожному відкритті [13]. При відкритті нового вікна (виклик методу Push) система переводить попередній екран у прихований стан через SetActive(false), зберігаючи об'єкт в пам'яті. Таким чином, усі підписки залишаються активними та продовжують отримувати оновлення від EventBus. Водночас елементи системи Heads-Up Display (смужки здоров'я, енергії, мінімапа) функціонують як незалежні always-on компоненти поза межами стеку, реагуючи на зміни ігрового стану автономно. Такий підхід передбачає певне збільшення стаціонарного використання оперативної пам'яті та холості виклики методу Update, проте це свідомий інженерний компроміс, який повністю усуває проблеми з втратою стану інтерфейсу та забезпечує безшовну навігацію.

Функціонування ігрового простору організовано на основі кастомної системи Persistent Room System, яка замінює стандартні механізми аддитивного завантаження сцен. У жанрі Metroidvania традиційне завантаження рівнів створює асинхронні шви (async seams) під час просторових переходів та унеможливляє глобальне опитування об'єктів для систем на кшталт мінімапи, оскільки неактивні зони вивантажуються з пам'яті. У розробленій архітектурі всі кімнати ігрового світу існують у межах однієї сцени і перебувають у стані постійної активності. Клас RoomManager відстежує поточне місцеперебування гравця та при його зміні публікує кешовану подію ActiveRoomChangedEvent. Для оптимізації продуктивності вороги та інтерактивні об'єкти, що перебувають поза межами активної кімнати, самостійно переходять у неактивний стан завдяки підписці на цю подію. Переміщення між зонами контролюється класом RoomTeleport, який опрацьовує колізії гравця та ініціює корутину переходу. Цей алгоритм включає візуальне затемнення екрана через оверлей ScreenFader, миттєве переміщення координат гравця на нову точку появи (spawn point) та оновлення меж ігрової камери без необхідності зупиняти основний потік виконання для зчитування даних із жорсткого диска.

Управління просторовим переміщенням камери розроблено без використання популярного плагіна Cinemachine, оскільки його внутрішні

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				37
Змн.	Арк.	№ докум.	Підпис	Дата		

алгоритми функціонують як «чорна скринька» та безпосередньо конфліктують із пайплайном рендерингу для CRT-ефектів (зокрема з компонентом PixelPerfectCamera). Розроблена підсистема CameraController базується на застосуванні пріоритетного стеку поведінок, кожна з яких реалізує інтерфейс ICameraBehaviour. У системі наявні різноманітні алгоритми руху, серед яких FollowBehaviour для стеження за гравцем із випереджальним зміщенням (look-ahead) на основі вектора руху, CutsceneBehaviour для фіксації фокусу на специфічних координатах світу під час ігрових подій, а також ShakeModifier для адитивного розрахунку ефекту тремтіння камери (camera shake) із затуханням (trauma-decay). Математичний пайплайн обчислення позиції камери виконується виключно в методі LateUpdate і складається з чіткої послідовності кроків. Спочатку визначається базова цільова позиція від активної поведінки з найвищим пріоритетом, потім до неї додається вектор тремтіння, після чого результат обмежується межами поточної активної кімнати (clamp). Останнім етапом розрахунку є функція піксельно-точного вирівнювання (PixelPerfectSnap) на основі значення PPU (Pixels Per Unit), що гарантує відсутність ефекту тремтіння субпікселів (sub-pixel jitter) на фінальному етапі рендерингу.

Процедурна генерація рівнів реалізована як детермінований секційний алгоритм, що збирає лінійний ігровий світ із готових шаблонів кімнат (RoomTemplate). Логіка розділена на два шари, де чисте C#-ядро LevelLayoutCore, що оперує власними математичними типами без залежності від UnityEngine, та MonoBehaviour-адаптер LevelGenerator, що інстанціює префаби. Алгоритм будує головний коридор посекційно. Є вступний сегмент (поява -> лінійні кімнати -> кімната ривка -> точка збереження), далі по одній тематичній секції на кожну здібність (кілька бойових кімнат-філлерів, опційне відгалуження вгору з імовірністю ForkChance, кімната-нагорода зі здібністю, точка збереження), і термінальна кімната боса. Така структура гарантує зв'язність світу та коректний порядок отримання здібностей за побудовою, без потреби в окремій перевірці досяжності. Вибір шаблонів і параметри секцій визначаються

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

сіджованим System.Random, що забезпечує повну детермінованість – один seed завжди дає ідентичний світ.

Візуальний та аудіосупровід розроблено з урахуванням концепції повної ізоляції. Замість вбудованого об'єкта Animator, який жорстко зв'язує логічні параметри з візуальними переходами за допомогою рядкових ключів (string-keyed API), реалізовано кастомний компонент PlayerAnimator. Він прослуховує шину подій (наприклад, PlayerDirectionEvent) та самостійно перемикає кліпи типу SpriteAnimationClip без жодних посилань на клас логіки PlayerMain. Створення атмосфери ретро-візуалізації забезпечується спеціалізованим CRT-шейдером, написаним засобами мови HLSL у форматі ShaderLab. Шейдер застосовується до RenderTexture роздільною здатністю 640×360 пікселів та імітує оптичні спотворення, такі як сканлайни, хроматичні аберації, бочкоподібна дисторсія та віньєтування. Звукова підсистема контролюється класом AudioManager, який використовує об'єкти AudioLibrary (ScriptableObject) для зберігання конфігурацій звуків і відтворює аудіо через пряме управління каналами AudioSource. Це дозволило відмовитися від ресурсомістких налаштувань AudioManager та імплементувати гнучку логіку логарифмічного регулювання гучності виключно програмними засобами.

2.4 Діаграми та алгоритми

Для унаочнення архітектурних рішень, описаних у попередніх підрозділах, у даному розділі наведено графічні моделі двох типів – структурна діаграма шарів збірок та діаграми послідовностей для критичних алгоритмів системи.

Архітектурна організація збірок проєкту унаочнюється діаграмою шарів (рис. 2.2). На діаграмі видно, що вектор залежностей спрямований виключно знизу вгору. Нижні шари FiatLux.Core та FiatLux.Contracts не мають жодних вихідних залежностей, тоді як верхній шар FiatLux.Bootstrap посилається на всі підсистеми одночасно та є єдиною точкою зв'язування. Шари FiatLux.World, FiatLux.UI та FiatLux.Visual розташовані на одному рівні і залежать лише від сервісного шару та контрактів – але не від геймплейного шару. Саме ця

		Льїн І.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Лобанчикова Н.М.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

властивість унеможлиблює пряме звернення до класів PlayerMain або HFSM з візуального чи інтерфейсного коду на рівні компіляції.

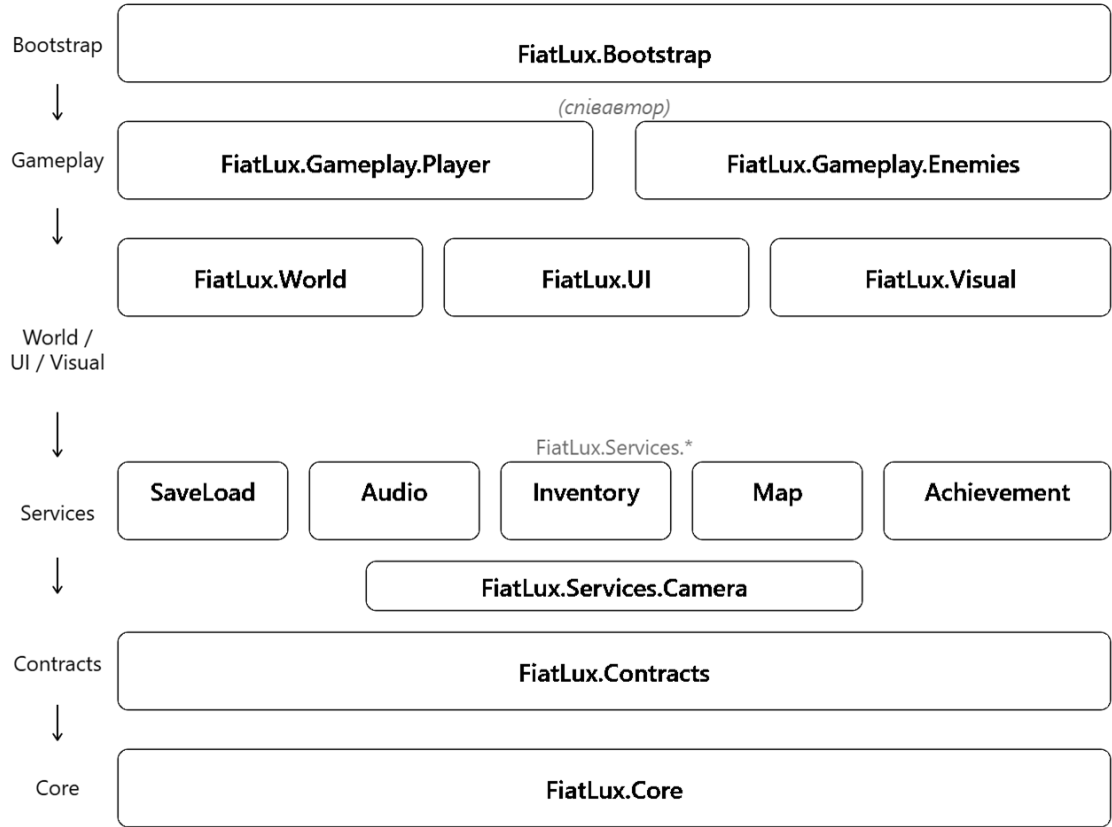


Рис. 2.2 – Діаграма шарів Assembly Definitions

Для візуалізації динаміки взаємодії об'єктів та розуміння послідовності виконання операцій у розробленій архітектурі було спроектовано діаграми послідовностей для найбільш критичних інженерних процесів. Одним із таких процесів є алгоритм подієво-орієнтованого збереження ігрового прогресу. На відміну від лінійного виклику методів, цей алгоритм базується на принципах інверсії управління [7].

Алгоритм збереження стану ініціюється викликом методу Save() у класі SaveManager. Після цього менеджер створює порожній об'єкт-контейнер SaveData і публікує системну подію CollectingSaveDataEvent, передаючи цей контейнер як аргумент. Усі автономні сервіси, такі як InventoryService, MapDiscoveryService та AchievementService, які попередньо підписалися на цю подію в методі OnEnable, отримують сповіщення. Кожен із сервісів синхронно

записує свої дані у відповідні поля контейнера. Оскільки подія обробляється в межах одного кадру на головному потоці (Main Thread), SaveManager гарантовано отримує повністю заповнений об'єкт SaveData після завершення диспетчеризації події. Завершальним кроком алгоритму є серіалізація контейнера у формат JSON за допомогою JsonUtility та запис згенерованого тексту у файл локального слота за допомогою методів простору імен System.IO. Такий алгоритм дозволяє додавати нові системи, що потребують збереження, без жодних модифікацій коду самого менеджера збережень (рис. 2.3).

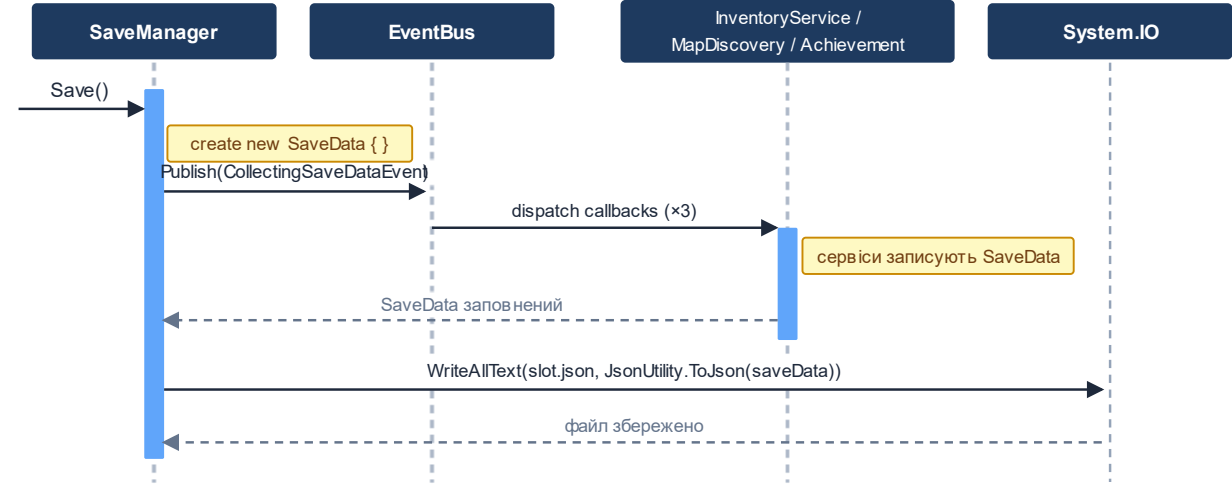


Рис. 2.3 – Діаграма послідовності подієво-орієнтованого збереження прогресу

Послідовність операцій ілюструється діаграмою послідовностей (рис. 2.3), на якій наглядно видно розподіл відповідальності між компонентами та гарантія заповнення SaveData у межах одного кадру.

Іншим ключовим алгоритмом, що визначає логіку управління простором, є послідовність переходу гравця між ігровими зонами (кімнатами) у системі Persistent Room System. Цей алгоритм повністю усуває необхідність звернення до стандартного менеджера сцен Unity (SceneManager). Процес ініціюється, коли колайдер гравця перетинає тригерну зону об'єкта RoomTeleport. Алгоритм переходу реалізовано у вигляді співпрограми (Coroutine), яка виконує дії у визначеній часовій послідовності. Спочатку викликається затемнення екрана через компонент ScreenFader. Після досягнення повної непрозорості екрана компонент переміщує об'єкт гравця на цільові координати (spawn point) нової кімнати. Наступним кроком є виклик методу SetActiveRoom(targetRoom) у класі

RoomManager. Менеджер оновлює внутрішній стан і публікує кешовану подію `ActiveRoomChangedEvent`. На цю подію миттєво реагує клас `CameraController`, який зчитує нові просторові межі та застосовує їх для обмеження вектора руху камери (`clamp`). Водночас усі об'єкти ворогів, які перебувають поза межами нової кімнати, обробляють цю ж подію та переводять себе в неактивний стан для економії обчислювальних ресурсів. Завершується алгоритм поступовим відновленням прозорості екрана (`Fade In`) (рис. 2.4).

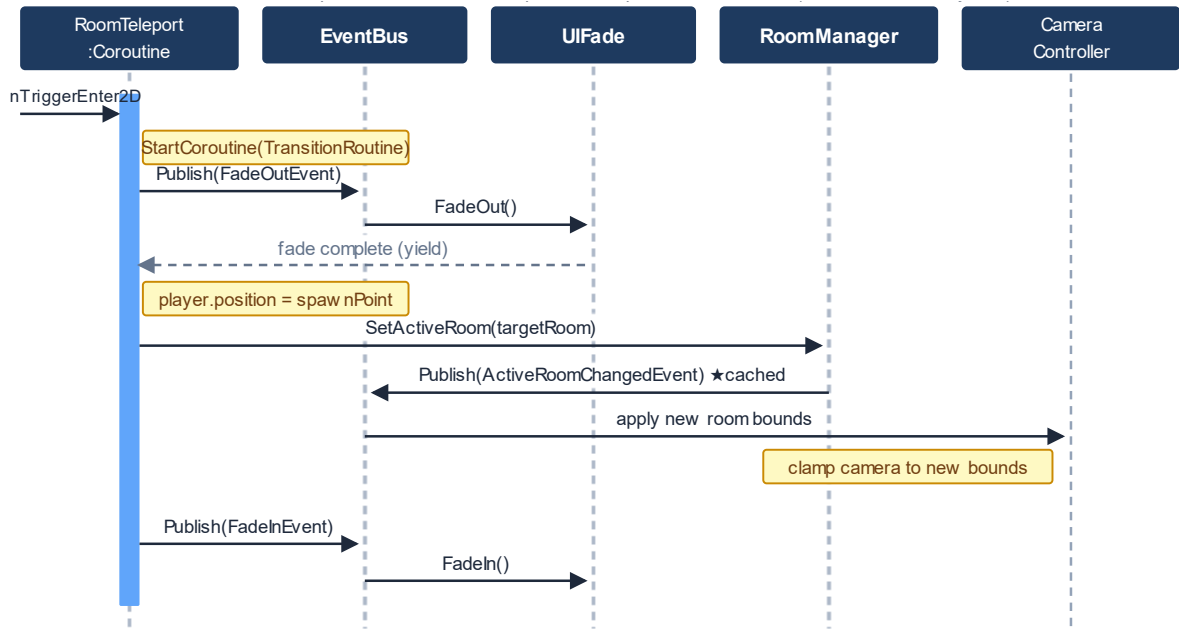


Рис. 2.4 – Діаграма послідовності переходу між ігровими кімнатами

Діаграма послідовності (рис. 2.4) відображає критичну роль механізму `PublishLatest` у забезпеченні синхронізації камери та переходу гравця без явних викликів між компонентами.

Алгоритм генерації працює послідовно. На вхід подається `seed` (ініціалізує `System.Random`) та набір шаблонів кімнат. Спочатку розміщується стартова кімната в початку координат, далі – вступний сегмент із кімнатою ривка. Потім для кожної здібності генерується окрема секція – набір бойових кімнат-філлерів (кількість у заданому діапазоні), з імовірністю `ForkChance` – вертикальне відгалуження з бічними кімнатами-нагородами, далі кімната розблокування здібності та точка збереження. Завершується генерація розміщенням кімнати боса в кінці коридору. Результатом є масив структур `GeneratedRoomData` з

координатами та парами зв'язків (горизонтальні left<->right, вертикальні lower<->upper), що передається адаптеру LevelGenerator для інстанціювання (рис. 2.5).

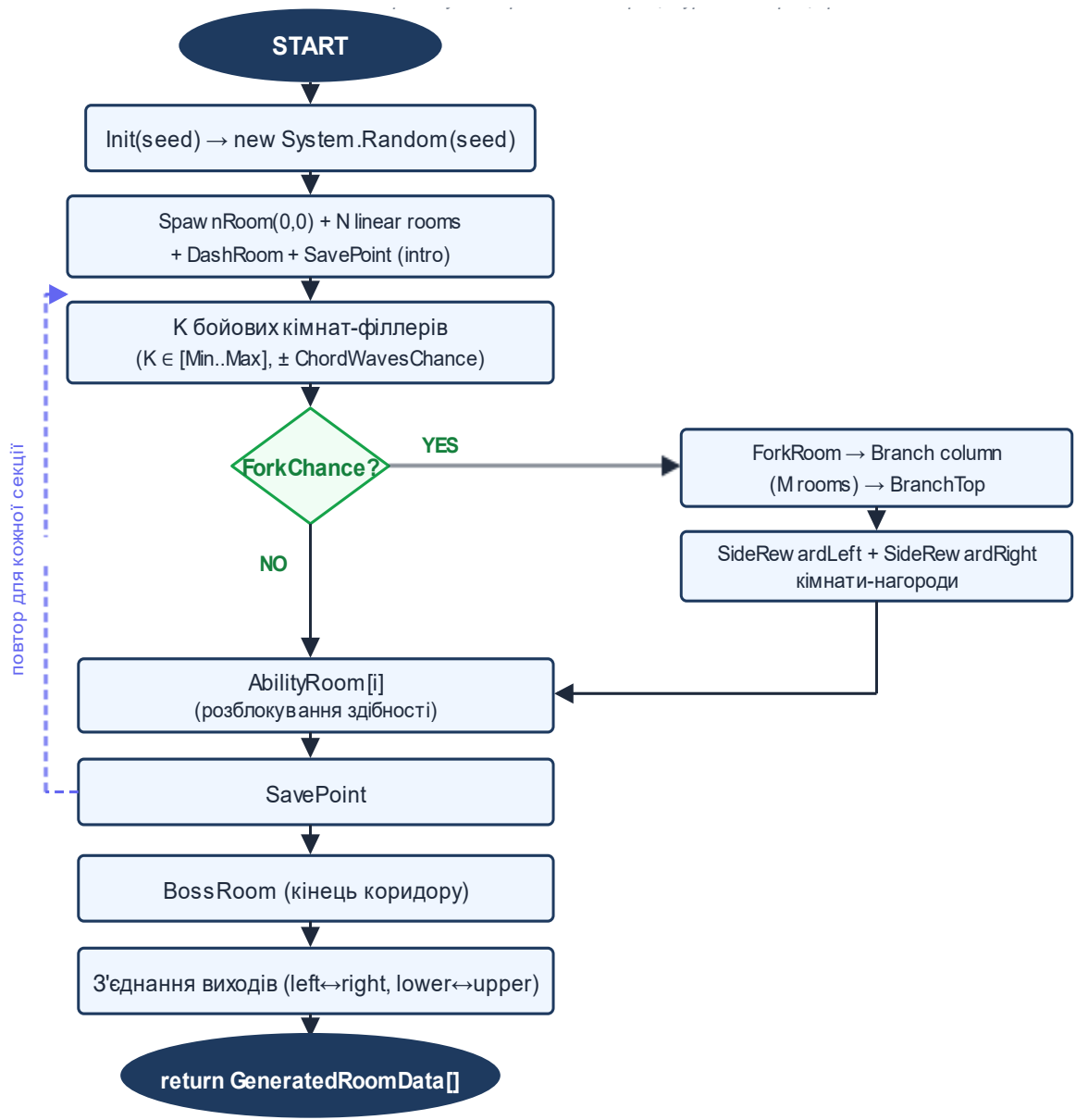


Рис. 2.5 – Блок-схема алгоритму процедурної генерації рівня

Блок-схема алгоритму (рис. 2.5) показує структуру детермінованої генерації – лінійні кімнати чергуються з вертикальними відгалуженнями відповідно до імовірностей, що запрограмовані у LevelLayoutConfig, а кінцевим результатом є масив GeneratedRoomData з повністю означеними зв'язками між кімнатами.

2.5 Реалізація функціоналу систем

У п'ятому підрозділі детально описується реалізація п'яти ключових програмних підсистем проєкту з наведенням фрагментів коду, що є найбільш характерними з точки зору застосованих архітектурних рішень. Повна демонстрація класів наведена у відповідних додатках.

Центральним елементом архітектури міжмодульної взаємодії є типізована статична шина подій у просторі імен FiatLux.Core.Events. Контракт події визначається порожнім маркерним інтерфейсом IEvent – будь-який клас або структура, що реалізує цей інтерфейс, може бути опублікована або оброблена через шину без реєстрації в центральному реєстрі:

```
namespace FiatLux.Core.Events
{
    public interface IEvent { }
}
```

Внутрішньо шина підтримує два статичних словників. `_handlers` типу `Dictionary<Type, Delegate>` для зберігання підписників та `_lastValues` типу `Dictionary<Type, object>` для кешування останніх значень подій. Ключем в обох словниках є `System.Type` конкретного типу події, що забезпечує ізоляцію обробників по типу на рівні виконання.

Метод `Subscribe` комбінує нового обробника з наявним делегатом через `Delegate.Combine`, що дозволяє необмежену кількість підписників на один тип події:

```
public static void Subscribe<T>(Action<T> handler) where T : IEvent
{
    if (handler == null) return;
    var type = typeof(T);
    if (_handlers.TryGetValue(type, out var existing))
        _handlers[type] = Delegate.Combine(existing, handler);
    else
        _handlers[type] = handler;
}
```

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Лобанчикова Н.М.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

Симетричним є метод `Unsubscribe`, який видаляє конкретний обробник через `Delegate.Remove`. Якщо після видалення ланцюг делегатів стає порожнім, ключ повністю видаляється зі словника – це запобігає накопиченню мертвих записів при активному перестворенні об'єктів:

```
public static void Unsubscribe<T>(Action<T> handler) where T : IEvent
{
    if (handler == null) return;
    var type = typeof(T);
    if (!_handlers.TryGetValue(type, out var existing)) return;
    var updated = Delegate.Remove(existing, handler);
    if (updated == null) _handlers.Remove(type);
    else
        _handlers[type] = updated;
}
```

Метод `Publish` отримує делегат зі словника та викликає його через безпечне приведення типу `as Action<T>`. Виклик обгорнутий у блок `try-catch`: виняток одного підписника фіксується через `Debug.LogException` і не перериває ланцюг виклику решти підписників:

```
public static void Publish<T>(T evt) where T : IEvent
{
    if (!_handlers.TryGetValue(typeof(T), out var handler)) return;
    try { (handler as Action<T>)?.Invoke(evt); }
    catch (Exception e) { Debug.LogException(e); }
}
```

Для подій, що відображають поточний стан системи, реалізовано механізм `PublishLatest`. Він зберігає останній екземпляр події у словнику `_lastValues` перед викликом `Publish`, що дозволяє новим підписникам отримати актуальний стан без очікування наступного оновлення:

```
public static void PublishLatest<T>(T evt) where T : IEvent
{
    _lastValues[typeof(T)] = evt;
    Publish(evt);
}
```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

Цей механізм є критичним для `ActiveRoomChangedEvent`: `CameraController` підписується на неї через `SubscribeWithLatest` і миттєво отримує межі поточної кімнати навіть при ініціалізації після першої публікації. Метод `SubscribeWithLatest` реєструє обробника стандартним чином, а потім негайно передає йому кешоване значення, якщо воно вже існує:

```
public static void SubscribeWithLatest<T>(Action<T> handler) where T :
    IEvent
{
    if (handler == null) return;
    Subscribe(handler);
    if (_lastValues.TryGetValue(typeof(T), out var val))
    {
        try { handler((T)val); }
        catch (Exception e) { Debug.LogException(e); }
    }
}
```

Для одноразового зчитування поточного стану без постійної підписки реалізовано метод `TryGetLatest`. Він використовується під час ініціалізації компонентів, яким потрібен поточний стан, але не потрібні майбутні оновлення:

```
public static bool TryGetLatest<T>(out T evt) where T : IEvent
{
    if (_lastValues.TryGetValue(typeof(T), out var val))
    {
        evt = (T)val; return true;
    }
    evt = default; return false;
}
```

Метод `ClearLatestCache` очищає лише словник `_lastValues`, залишаючи всі підписки активними. Він є критичним для системи збереження. Перед застосуванням завантажених даних кеш скидається, щоб уникнути ситуації, коли новий підписник отримує застарілий стан попередньої сесії:

```
public static void ClearLatestCache() => _lastValues.Clear();
```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Лобанчикова Н.М.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

Окремо існує метод Clear, який повністю скидає обидва словники. Його не слід викликати вручну в методах Awake – це знищить уже зроблені підписки. Повне очищення при старті Play Mode виконується автоматично методом AutoReset, позначеним атрибутом RuntimeInitializeOnLoadMethod зі значенням SubsystemRegistration. Це гарантує повне скидання стану шини до моменту спрацьовування методів Awake будь-яких об'єктів на сцені:

```
[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.SubsystemRegistration)]
private static void AutoReset()
{
    _handlers.Clear();
    _lastValues.Clear();
}
```

Підсистема генерації ігрового простору є найбільш показовим прикладом застосування паттерну «Адаптер» у проєкті. Вся математика компонування кімнат зосереджена в чистому C#-класі LevelLayoutCore, який не містить жодного посилання на простір імен UnityEngine. Клас LevelGenerator є єдиним MonoBehaviour-адаптером між інспектором рушія та алгоритмом. Точкою входу є метод GenerateLevelLayout(int seed), який викликається із GameManager – ніколи із Start – щоб уникнути гонки з OnSceneLoaded:

```
public void GenerateLevelLayout(int seed)
{
    var config = BuildConfig();
    var result = _core.Generate(config, seed);

    SpawnRoomsInHierarchy(result);
    LinkRoomTeleports(result);
    SaveLayoutToJSON(result);

    StartCoroutine(SetInitialRoomDeferred("room_1"));
    EventBus.Publish(new LevelGeneratedEvent(result.RoomCount));
}
```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

Конфігурація передається в алгоритм у вигляді незмінного значення LevelLayoutConfig – readonly struct із константою RoomPadding = 0.5f та усіма параметрами, які ініціалізуються єдиний раз у конструкторі. Метод BuildConfig збирає всі серіалізовані поля інспектора в один виклик конструктора – це єдина точка зв'язку між шарами MonoBehaviour та чистим C#:

```
private LevelLayoutConfig BuildConfig() => new LevelLayoutConfig(
    spawnTemplate:          _spawnTemplate,
    dashRoomTemplate:        _dashRoomTemplate,
    savePointTemplate:        _savePointTemplate,
    bossTemplate:            _bossTemplate,
    linearEnemyTemplates:     _linearEnemyTemplates,
    chordWavesTemplates:     _chordWavesTemplates,
    treasureAbilityTemplates: _treasureAbilityTemplates,
    forkTemplates:           _forkTemplates,
    branchMidTemplates:       _branchMidTemplates,
    branchTopTemplates:       _branchTopTemplates,
    sideBranchTemplates:      _sideBranchTemplates,
    sideRewardLeftTemplates:  _sideRewardLeftTemplates,
    sideRewardRight:          _sideRewardRight,
    minIntroLinearRooms:      _minIntroLinearRooms,
    maxIntroLinearRooms:      _maxIntroLinearRooms,
    minLinearRoomsPerSection: _minLinearRoomsPerSection,
    maxLinearRoomsPerSection: _maxLinearRoomsPerSection,
    minBranchLength:          _minBranchLength,
    maxBranchLength:          _maxBranchLength,
    forkChance:                _forkChance,
    chordWavesChance:         _chordWavesChance,
    tileSize:                  _tileSize);
```

Результат роботи алгоритму представлено незмінним об'єктом LevelLayoutResult із трьома колекціями, доступними лише для читання. Зв'язки між кімнатами зберігаються як кортежі з іменованими полями – (left, right) для

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

горизонтальних і (lower, upper) для вертикальних переходів, що робить код адаптера самодокументованим:

```
public sealed class LevelLayoutResult
{
    public readonly IReadOnlyList<GeneratedRoomData> Rooms;
    public readonly IReadOnlyList<(GeneratedRoomData left,
GeneratedRoomData right)> HorizontalLinks;
    public readonly IReadOnlyList<(GeneratedRoomData lower,
GeneratedRoomData upper)> VerticalLinks;
    public int RoomCount => Rooms.Count;
}
```

Метод Generate ініціалізує System.Random із переданим значенням seed та очищає всі внутрішні стани перед запуском. Використання System.Random замість UnityEngine.Random є свідомим архітектурним рішенням – System.Random є детермінованим на всіх платформах незалежно від глобального стану рушія, тому той самий seed завжди породжує ідентичне розташування кімнат – необхідна умова для відтворення рівня після завантаження збереження:

```
public LevelLayoutResult Generate(LevelLayoutConfig config, int seed)
{
    _cfg = config;
    _rng = new System.Random(seed);
    _layout.Clear();
    _horizontalLinks.Clear();
    _verticalLinks.Clear();
    _occupied.Clear();
    _usedLeftRewards.Clear();
    _lastLinearTemplate = null;
    // ...
}
```

Для внутрішніх геометричних обчислень у LevelLayoutCore визначено дві власні readonly struct: FVec2 (двовимірний вектор із полями x та y) та FRect (прямокутник із методом Overlaps). Оскільки обидва типи є значенневими, вся геометрія розміщення виконується без виділення пам'яті в керованій купі та без

		Льїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

жодних посилань на UnityEngine. Розміщення кімнат виконується методом PlaceNextTo, який забезпечує вирівнювання підлог при горизонтальних переходах:

```
private FVec2 PlaceNextTo(GeneratedRoomData a, RoomTemplate b, Direction
dir)
{
    float halfW_A = a.sizeInTiles.x * _cfg.TileSize * 0.5f;
    float halfH_A = a.sizeInTiles.y * _cfg.TileSize * 0.5f;
    float halfW_B = b.roomSizeInTiles.x * _cfg.TileSize * 0.5f;
    float halfH_B = b.roomSizeInTiles.y * _cfg.TileSize * 0.5f;
    float floorY = a.worldPosition.y - halfH_A;

    return dir switch
    {
        Direction.Right => new FVec2(
            a.worldPosition.x + halfW_A + LevelLayoutConfig.RoomPadding +
halfW_B,
            floorY + halfH_B),
        Direction.Top => new FVec2(
            a.worldPosition.x,
            a.worldPosition.y + halfH_A + LevelLayoutConfig.RoomPadding +
halfH_B),
        _ => throw new System.ArgumentOutOfRangeException()
    };
}
```

Метод GetLinearOnly забезпечує відсутність однакових шаблонів підряд через відстеження `_lastLinearTemplate`. Алгоритм будує рівень послідовно – стартова кімната у початку координат, вступний сегмент із лінійними кімнатами та кімнатою ривка, далі по одній секції на кожен елемент масиву `TreasureAbilityTemplates`, завершуючи кімнатою боса. У вступному сегменті ChordWaves-кімнати виключені навмисно – гравець ще не має ривка. З імовірністю `ForkChance` будується вертикальне відгалуження через `TrySideBranch`.

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

Після завершення алгоритму LevelGenerator виконує Unity-специфічну роботу. SpawnRoomsInHierarchy інстанціює префаби під спільний контейнер, тимчасово деактивований для уникнення передчасних Awake-викликів. Після інстанціювання контейнер активується, і кожен Room-компонент реєструється у RoomManager:

```
private void SpawnRoomsInHierarchy(LevelLayoutResult result)
{
    var container = new GameObject("GeneratedRooms");
    container.SetActive(false);

    foreach (var data in result.Rooms)
    {
        var go = Instantiate(data.template.roomPrefab,
                             data.worldPosition,
                             Quaternion.identity,
                             container.transform);

        go.name = data.roomId;
        var room = go.GetComponent<Room>();
        room.RoomId = data.roomId;
        room.CacheTeleports();
        data.runtimeInstance = room;
    }
    container.SetActive(true);
}
```

LinkRoomTeleports обходить кортежі зв'язків та прив'язує компоненти RoomTeleport у двох напрямках для кожної пари. Незв'язані двері деактивуються через Seal:

```
private void LinkRoomTeleports(LevelLayoutResult result)
{
    foreach (var (left, right) in result.HorizontalLinks)
    {
```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Лобанчикова Н.М.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        LinkTeleport(left, Direction.Right, right);
        LinkTeleport(right, Direction.Left, left);
    }
    foreach (var (lower, upper) in result.VerticalLinks)
    {
        LinkTeleport(lower, Direction.Top, upper);
        LinkTeleport(upper, Direction.Bottom, lower);
    }
    SealUnlinkedDoors(result);
}

private void LinkTeleport(GeneratedRoomData from, Direction exitDir,
GeneratedRoomData to)
{
    var door = from.runtimeInstance.GetTeleport(exitDir);
    var arrival = to.runtimeInstance.GetArrivalPoint(Opposite(exitDir));
    door?.LinkToRoom(to.roomId, arrival);
}

```

SaveLayoutToJSON серіалізує граф у level_graph.json для налагодження. Завершальним кроком корутина SetInitialRoomDeferred очікує один кадр – щоб усі Room-компоненти завершили реєстрацію у RoomManager – і лише після цього встановлює початкову кімнату та публікує LevelGeneratedEvent:

```

private IEnumerator SetInitialRoomDeferred(string roomId)
{
    yield return null; // Очікуємо завершення Awake усіх Room-компонентів
    RoomManager.Instance.SetInitialRoom(roomId);
}

```

Сам перехід між кімнатами обробляється компонентом RoomTeleport через корутину TeleportRoutine. Алгоритм переходу реалізує чітку послідовність – публікація RoomTransitionStartedEvent, затемнення екрана через ScreenFader, миттєве переміщення гравця на точку появи з обнуленням швидкості $\text{rb.linearVelocity} = \text{Vector2.zero}$, виклик RoomManager.SetActiveRoom – який

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

публікує кешовану подію `ActiveRoomChangedEvent` для оновлення меж камери – та відновлення прозорості:

```
private IEnumerator TeleportRoutine(Transform player)
{
    _isTeleporting = true;
    EventBus.Publish(new RoomTransitionStartedEvent());

    yield return ScreenFader.Instance.FadeOut(_fadeDuration);

    player.position = _spawnPoint.position;
    if (player.TryGetComponent<Rigidbody2D>(out var rb))
        rb.linearVelocity = Vector2.zero;

    RoomManager.Instance.SetActiveRoom(_targetRoomId);

    if (_delayBeforeFadeIn > 0f)
        yield return new WaitForSeconds(_delayBeforeFadeIn);

    yield return ScreenFader.Instance.FadeIn(_fadeDuration);
    EventBus.Publish(new RoomTransitionCompletedEvent());
    _isTeleporting = false;
}
```

Клас `CameraController` організовано як менеджер пріоритетного списку поведінок, кожна з яких імплементує інтерфейс `ICameraBehaviour`. На початку кожного кадру у методі `LateUpdate` контролер ітерує список поведінок, впорядкований за критерієм `Priority` (більш високе значення = більш високий пріоритет), та запитує у поведінки з найвищим пріоритетом розрахову позицію цільової точки спостереження. Ці вектори послідовно обробляються трьома фільтрами – `ClampToBounds`, який обмежує позицію границями поточної активної кімнати та коректно центрує вьюпорт у малих кімнатах; `CameraShake`, який додає до позиції `offset` на основі поточного рівня травми та `Perlin noise`; та `SnapToPixel`, який вирівнює позицію до кратного значення `1/PPU` (`Pixels Per Unit`), що цілком залежить від конфігурації `Pixel Perfect Camera`. На закінчення

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

розраховувана позиція присвоюється Transform камери, зберігаючи Z-координату (камера завжди перебуває на однаковій глибині в Z). Такий дизайн дозволяє з легкістю комбінувати довільні поведінки (наприклад, одночасно FollowBehaviour, CutsceneBehaviour та ShakeModifier) без модифікації коду самого контролера.

```
private void LateUpdate():
{
    Vector2 desired = _behaviours[0].CalculateTargetPosition(
        _target, transform.position, Time.deltaTime);

    if (_hasBounds)    desired = ClampToBounds(desired);
    if (_shake.IsActive) desired += _shake.GetOffset(Time.deltaTime);
    if (_pixelSnap)    desired = SnapToPixel(desired);

    transform.position = new Vector3(desired.x, desired.y,
transform.position.z);
}
```

Базовою поведінкою є FollowBehaviour з Priority 0, що реалізує мертву зону у світових координатах та look-ahead зміщення за вектором linearVelocity Rigidbody2D гравця зі згладжуванням через Vector2.Lerp. Для кат-сцен передбачено CutsceneBehaviour з Priority 10, яке витісняє FollowBehaviour і плавно переміщує камеру у фіксовану точку через Vector2.SmoothDamp. Метод ClampToBounds коректно обробляє випадок, коли кімната менша за вьюпорт замість зажиму вісь центрується. SnapToPixel вирівнює позицію до кратного значення 1/PPU, що усуває субпіксельне тремтіння при Pixel Perfect Camera.

Клас AudioManager реалізовано як Singleton із двома компонентами AudioSource (musicSource та sfxSource), які зберігаються на DontDestroyOnLoad-об'єкті та функціонують незалежно одна від одної. Клас інкапсулює три рівні керування гучністю: MasterVolume (загальна гучність прослуховувача), MusicVolume (гучність музичної дорізки) та SfxVolume (гучність звукових ефектів). Кожний рівень має відповідний boolean-флаг для миттєвого вимкнення звуку без зміни позиції слайдера (masterMuted, musicMuted, sfxMuted).

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

Конфігурація звуків зберігається у ScriptableObject-об'єкті AudioLibrary, який містить словник ідентифікаторів звуків та відповідних аудіо-клипів.

При ініціалізації у методі Awake менеджер завантажує збережені налаштування гучності з файлу та підписується на три типи подій через EventBus:

```
private void Awake()
{
    if (Instance != null) { Destroy(gameObject); return; }
    Instance = this;

    audioLibrary.Initialize();
    LoadVolumes();
    ApplyAll();

    EventBus.Subscribe<PlaySfxEvent>(OnPlaySfx);
    EventBus.Subscribe<PlayMusicEvent>(OnPlayMusic);
    EventBus.Subscribe<StopMusicEvent>(OnStopMusic);
}
```

Відтворення звукових ефектів здійснюється через подію PlaySfxEvent, яка містить ідентифікатор клипу. Обробник OnPlaySfx пошукує клип у бібліотеці та запускає його на каналі SFX через PlayOneShot, що дозволяє накладати кілька ефектів одночасно:

```
private void OnPlaySfx(PlaySfxEvent e)
{
    var clip = audioLibrary.Get(e.ClipId);
    if (clip == null) { Debug.LogWarning($"[AudioManager] SFX not found: '{e.ClipId}'"); return; }
    sfxSource.PlayOneShot(clip);
}
```

Музична композиція змінюється через подію PlayMusicEvent, яка призначає новий клип музичному каналу та запускає його відтворення:

```
private void OnPlayMusic(PlayMusicEvent e)
{

```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

```

var clip = audioLibrary.Get(e.TrackId);
if (clip == null) { Debug.LogWarning($"[AudioManager] Music not
found: '{e.TrackId}'"); return; }
musicSource.clip = clip;
musicSource.Play();
}

```

Критично важливим аспектом аудіосистеми є перцептуальна степенева крива гучності. Слайдер слухача (діапазон 0..1, лінійна позиція) перетворюється на амплітуду через степеневу функцію з показником 2.5:

```

private float ToAmplitude(float linear, bool muted)
{
    if (muted || linear <= 0f) return 0f;
    return Mathf.Pow(Mathf.Clamp01(linear), perceptualExponent); //
exponent = 2.5
}

```

Цей підхід відповідає фізіологічному сприйняттю звука людиною. Логарифмічна чутливість вуха означає, що при лінійному русі слайдера гучність змінюється нелінійно – на малих значеннях (0-30% слайдера) амплітуда зростає повільно, що дозволяє тонко керувати гучністю, а на вищих значеннях (70-100%) зростання крутіше. Формула $amplitude = linear^{2.5}$ економічніша за логарифмічну та краще відповідає очікуванням гравців при взаємодії з UI-слайдерами.

Гучність застосовується на трьох рівнях паралельно. Рівень Master встановлюється на глобальний `AudioListener.volume`, що впливає на всі звуки в сцені:

```

private void ApplyMaster() => AudioListener.volume =
ToAmplitude(_masterLinear, _masterMuted);
private void ApplyMusic() => musicSource.volume =
ToAmplitude(_musicLinear, _musicMuted);
private void ApplySfx() => sfxSource.volume =
ToAmplitude(_sfxLinear, _sfxMuted);
private void ApplyAll() { ApplyMaster(); ApplyMusic(); ApplySfx(); }

```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

Через множення (Listener volume × Music source volume) фінальна гучність музики розраховується як добуток двох рівнів, що дозволяє гравцю незалежно контролювати загальну гучність та баланс музики/ефектів.

Гравець керує гучністю через UI-слайдери, які викликають публічні методи SetMasterVolume, SetMusicVolume та SetSFXVolume. Аналогічно, методи SetMasterMuted, SetMusicMuted та SetSFXMuted дозволяють вимкнути звук без втрати позиції слайдера:

```
public void SetMasterVolume(float linear) { _masterLinear =
Mathf.Clamp01(linear); ApplyMaster(); }
public void SetMusicVolume(float linear) { _musicLinear =
Mathf.Clamp01(linear); ApplyMusic(); }
public void SetSFXVolume(float linear) { _sfxLinear =
Mathf.Clamp01(linear); ApplySfx(); }

public void SetMasterMuted(bool muted) { _masterMuted = muted;
ApplyMaster(); }
public void SetMusicMuted(bool muted) { _musicMuted = muted;
ApplyMusic(); }
public void SetSFXMuted(bool muted) { _sfxMuted = muted;
ApplySfx(); }
```

Налаштування гучності зберігаються у файлі audio_settings.json у форматі JSON, що містить три float-значення гучності та три boolean-прапорці mute:

```
[System.Serializable]
private class VolumeData
{
    public float master = 1f;
    public float music = 1f;
    public float sfx = 1f;
    public bool masterMuted;
    public bool musicMuted;
    public bool sfxMuted;
}
```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

Збереження та завантаження здійснюються через методи SaveVolumes та LoadVolumes з обробкою помилок через try-catch:

```
public void SaveVolumes()
{
    var data = new VolumeData
    {
        master = _masterLinear, music = _musicLinear, sfx = _sfxLinear,
        masterMuted = _masterMuted, musicMuted = _musicMuted, sfxMuted =
        _sfxMuted,
    };
    try { File.WriteAllText(SavePath, JsonUtility.ToJson(data, true)); }
    catch (System.Exception ex) { Debug.LogWarning($"[AudioManager] Save
failed: {ex.Message}"); }
}

private void LoadVolumes()
{
    VolumeData data = null;
    try
    {
        if (File.Exists(SavePath))
            data =
            JsonUtility.FromJson<VolumeData>(File.ReadAllText(SavePath));
    }
    catch (System.Exception ex) { Debug.LogWarning($"[AudioManager] Load
failed: {ex.Message}"); }

    data ??= new VolumeData(); // Fallback to defaults
    _masterLinear = data.master;      _musicLinear = data.music;
    _sfxLinear     = data.sfx;
    _masterMuted   = data.masterMuted; _musicMuted   = data.musicMuted;
    _sfxMuted      = data.sfxMuted;
}
```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл зберігається в каталозі `Application.persistentDataPath`, що гарантує функціонування на різних платформах. Завантаження конфігурації виконується у методі `Awake` перед першим кадром, що забезпечує активацію налаштувань гравця з моменту запуску програми.

Клас `SaveManager` не має прямих посилань на жоден із сервісів, що беруть участь у серіалізації. Метод `Save` приймає вже створений об'єкт `SaveData` від виклику та публікує `CollectingSaveDataEvent` з ним як аргументом. Сервіси `InventoryService`, `MapDiscoveryService` та `AchievementService` підписуються на цю подію в `OnEnable` і самостійно заповнюють відповідні поля контейнера. Оскільки `Publish` виконується синхронно, після повернення з методу `SaveManager` гарантовано отримує повністю заповнений об'єкт і записує його на диск:

```
public bool Save(int slot, SaveData data)
{
    EventBus.Publish(new CollectingSaveDataEvent(data));
    data.timestampUtc = DateTime.UtcNow.Ticks;
    File.WriteAllText(SlotPath(slot), JsonUtility.ToJson(data,
prettyPrint: true));
    return true;
}
```

Завантаження є симетричним. Метод `Load` зчитує JSON та повертає десеріалізований об'єкт; розподіл даних між сервісами виконується через `GameLoadedEvent`, на яку кожен сервіс підписується самостійно. Підтримується три незалежних слоти, шляхи яких формуються через `Application.persistentDataPath`.

Система навігації між екранами побудована на стандартному `Stack<UIScreen>`. Базовий клас `UIScreen` управляє видимістю через `SetActive` на власному `GameObject` та відновлює фокус системи введення через `EventSystem.SetSelectedGameObject` при кожному відкритті. Метод `Push` приховує поточний верхній екран викликом `OnHide`, поміщає новий у стек та

		Льїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

викликає на ньому OnPush; Pop витягує верхній екран і відновлює попередній через OnReveal. Кожна операція публікує відповідну подію через EventBus:

```
public void Push(UIScreen screen)
{
    if (_stack.Count > 0) _stack.Peek().OnHide();
    _stack.Push(screen);
    screen.OnPush();
    EventBus.Publish(new ScreenPushedEvent(screen.name));
}

public void Pop()
{
    var top = _stack.Pop();
    top.OnPop();
    EventBus.Publish(new ScreenPoppedEvent(top.name));
    if (_stack.Count > 0) _stack.Peek().OnReveal();
}
```

Метод HandleCancel реалізує системну кнопку «Назад». Якщо верхній екран має прапор CloseOnCancel, він закривається автоматично без участі будь-якого зовнішнього контролера. Публікація ScreenPushedEvent та ScreenPoppedEvent дозволяє зовнішнім системам реагувати на зміну стану інтерфейсу без прямої підписки на конкретний екран.

Висновки до розділу 2

У другому розділі було детально розглянуто процеси проєктування та програмної реалізації архітектури відеогри «Fiat Lux». Розроблена об'єктно-орієнтована модель спирається на принципи контрактного програмування, де ключовим фактором є розмежування логіки, візуального представлення та сервісних підсистем.

Головним архітектурним досягненням є реалізація строгої модульної системи на основі механізму Assembly Definitions (asmdef) із заборonoю автоматичного створення перехресних посилань. Це рішення дозволило повністю ізолювати візуальні компоненти від геймплейної логіки, гарантуючи

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

неможливість виникнення циклічних залежностей. Зв'язок між модулями успішно забезпечується за допомогою розробленого типізованого статичного EventBus, який підтримує механізми кешування станів (sticky events) та усуває необхідність використання серіалізованих методів в інспекторі рушія.

У ході роботи було спроектовано та реалізовано низку автономних систем, які замінюють стандартні високорівневі компоненти Unity, що працюють як «чорні скриньки». Розроблена система камери CameraController забезпечує плавне переміщення та піксельно-точне позиціонування завдяки використанню пріоритетного стеку поведінок ICameraBehaviour замість пакету Cinemachine. Реалізація UIScreenStack на основі стекової навігації зі збереженням фокусу введення забезпечила коректний порядок відображення екранів та автоматичне повернення до попереднього стану.

Замість традиційного завантаження рівнів розроблено Persistent Room System, де всі просторові зони є постійно активними, а переходи між ними контролюються подіями від RoomManager. Додатково імплементовано алгоритм детермінованої процедурної генерації ігрового графа на базі System.Random, що гарантує відтворюваність рівнів на різних платформах. Підсистема збереження прогресу успішно переведена на подієво-орієнтовані рейки, що дозволило автоматизувати серіалізацію даних у формат JSON без створення прямих залежностей між SaveManager та іншими сервісами. Комплексне застосування цих інженерних підходів дозволило створити надійну, масштабовану та оптимізовану програмну базу для подальшого розширення ігрового продукту.

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС, ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ

3.1 Порядок встановлення та налаштування

Для коректної роботи з проектом відеогри «Fiat Lux» необхідно використовувати ігровий рушій Unity версії 6000.0.33f1 із встановленим модулем Build Support для платформи Windows (IL2CPP). Керування версіями рушія здійснюється через застосунок Unity Hub. Для редагування програмного коду рекомендується використовувати JetBrains Rider або Microsoft Visual Studio 2022 з підтримкою Unity. Проект також потребує наявності системи контролю версій Git версії 2.30 або новішої.

Вихідний код проекту зберігається у Git-репозиторії. Після клонування репозиторію проект відкривається через Unity Hub із вибором кореневої директорії. Під час першого запуску Unity автоматично виконує імпорт ресурсів, відновлення залежностей і компіляцію програмного коду. Тривалість цього процесу залежить від продуктивності комп'ютера та може становити декілька хвилин.

Проект побудований на модульній архітектурі з використанням Assembly Definitions. Під час розширення функціональності необхідно дотримуватися наявної структури модулів та коректно налаштовувати міжмодульні залежності (рис. 3.1). Базові сервіси, контракти, елементи інтерфейсу та компоненти ігрового світу мають розміщуватися у відповідних підсистемах проекту. При виникненні конфліктів у системі Assembly Definitions – зокрема, помилок типу Assembly reference cycle detected або Type already defined – необхідно перевірити відповідність розміщення нових скриптів архітектурним шарам проекту та переконатися у відсутності дублікатів імен класів між збірками. У разі пошкодження або відсутності директорії Library її можна безпечно видалити - Unity автоматично відновить її при наступному відкритті проекту, що не впливає на вихідний код та налаштування.

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				62
Змн.	Арк.	№ докум.	Підпис	Дата		

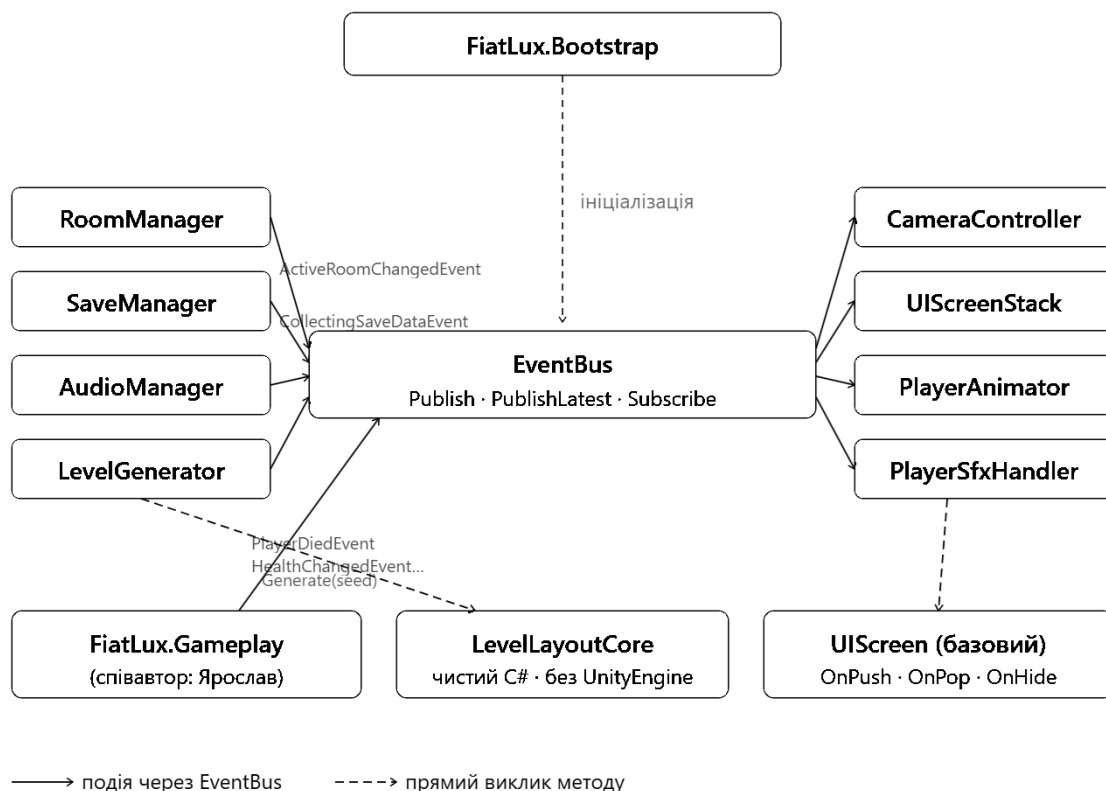


Рис. 3.1 – Діаграма компонентів

Для запуску гри використовується окрема сцена Bootstrap, яка виконує ініціалізацію глобальних сервісів та завантаження необхідної ігрової сцени. Під час роботи в редакторі Unity цей процес виконується автоматично, що дозволяє запускати тестування без додаткового налаштування середовища. Для зручності тестування окремих сцен реалізовано інструмент EditorSceneAutoBootstrap. При натисканні Play в редакторі він автоматично підміняє стартову сцену на Bootstrap, ініціалізує всі сервіси та дозавантажує поточну робочу сцену розробника. Завдяки цьому запуск із будь-якої сцени проєкту завжди дає коректно ініціалізоване середовище без ручного перемикання між сценами.

Керування в грі реалізовано за допомогою Unity Input System. Підтримуються клавіатура та геймпад. При підключенні контролера система автоматично визначає активний пристрій введення та перемикається між доступними схемами керування без необхідності перезапуску гри.

Для створення фінальної версії програмного продукту необхідно відкрити меню Build Profiles та вибрати платформу Windows x64. У списку сцен першою

має бути вказана сцена Bootstrap, після чого додаються інші сцени гри. Для фінальної збірки рекомендується використовувати скриптовий бекенд IL2CPP. Після завершення процесу компіляції формується автономний виконуваний файл, який може запускатися на цільовій системі без встановлення додаткового програмного забезпечення. Нижче наведено мінімальні та рекомендовані вимоги до апаратного забезпечення.

Таблиця 3.1

Мінімальні системні вимоги

Операційна система	Windows 10 (64-bit)
Процесор	Intel Core i3-6100 або аналогічний
Оперативна пам'ять	8 ГБ
Відеокарта	NVIDIA GeForce GTX 750 Ti або аналогічна з підтримкою DirectX 11
Вільне місце на диску	2 ГБ
Пристрій введення	клавіатура або геймпад

Таблиця 3.2

Рекомендовані системні вимоги

Операційна система	Windows 11 (64-bit)
Процесор	Intel Core i5-10400 або аналогічний
Оперативна пам'ять	16 ГБ
Відеокарта	NVIDIA GeForce GTX 1660 або аналогічна
Вільне місце на диску	4 ГБ
Пристрій введення	геймпад або клавіатура

Для контролю працездатності програмного забезпечення під час розробки використовується вбудована система журналювання Unity. Усі повідомлення про помилки, попередження та службові повідомлення відображаються у вікні Console редактора. При виникненні критичних помилок під час запуску необхідно перевірити коректність імпорту пакетів, наявність усіх сцен у конфігурації збірки та відсутність помилок компіляції у вихідному коді. У разі виникнення проблем із кешем рушія рекомендується виконати повторну генерацію службових директорій Library та Temp шляхом їх видалення з подальшим відкриттям проєкту в Unity.

Файли персистентних даних розміщуються у директорії %USERPROFILE%\AppData\LocalLow\[CompanyName]\FiatLux: save_slot_0.json, save_slot_1.json, save_slot_2.json – три незалежних слоти збереження; achievements.json – глобальні досягнення, спільні для всіх слотів; video_settings.json – параметри CRT-шейдера (рис. 3.2). У разі відсутності файлів збереження підсистеми гри автоматично створюють нові файли зі значеннями за замовчуванням. Для повного скидання прогресу користувач може скористатися відповідною функцією в меню налаштувань або видалити файли збереження вручну.

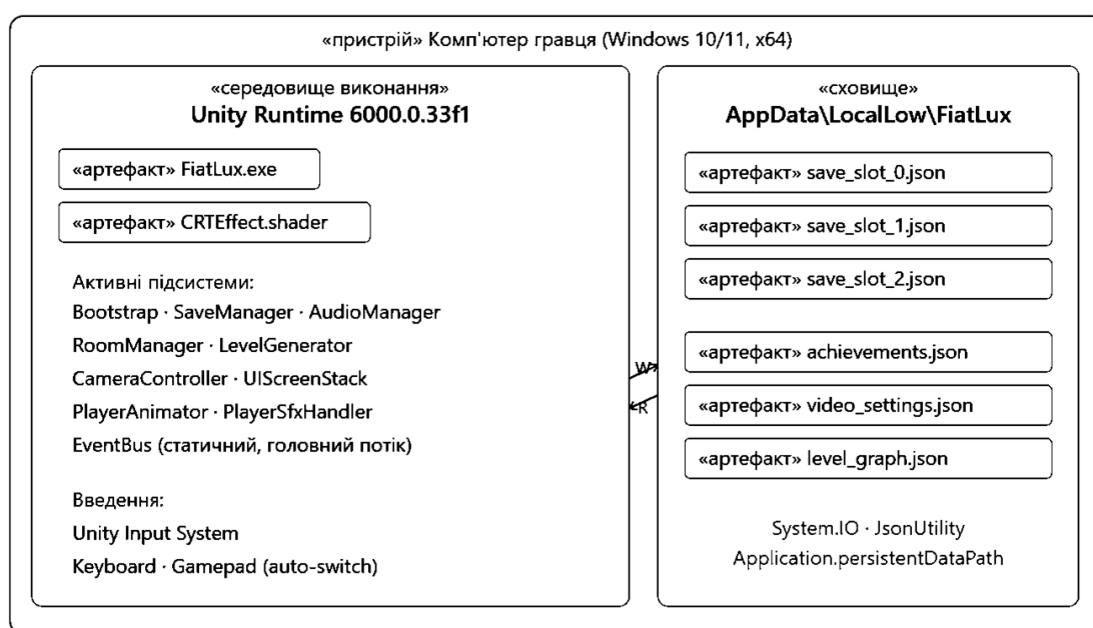


Рис. 3.2 – Діаграма розгортання гри

Дотримання наведених вимог та рекомендацій забезпечує коректне функціонування програмного забезпечення, стабільну роботу під час розробки та коректне виконання відеогри.

3.2 Опис інтерфейсу користувача

Інтерфейс користувача (UI) спроектовано з урахуванням концепції максимальної інформативності при мінімальному перекритті ігрового простору. Структурно інтерфейс поділяється на два незалежні логічні шари – систему постійного відображення стану (HUD) та систему інтерактивних екранів, керованих розробленим компонентом UIScreenStack [1, 3].

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

Шар HUD (Heds-Up Display) функціонує поза межами стеку екранів і завжди залишається на екрані. Він складається з кількох автономних підв'ю (sub-views), кожне з яких підписане на відповідні системні події через EventBus [2, 3]. У верхній лівій частині екрана постійно відображається поточний стан здоров'я і енергії персонажа, а також HealingHUDView (кількість доступних лікувальних предметів). У правій верхній частині розташовано панель активних здібностей гравця (AbilityHUDView), яка динамічно відображає розблоковані навички та відповідні їм кнопки введення, а також MiniMapView – динамічну мініатюрну структуру досліджених кімнат. Оновлення цих елементів відбувається виключно при публікації кешованих (sticky) подій від систем гравця та менеджера кімнат. Також до складу HUD входить InteractionPromptView – контекстне спливаюче вікно, що з'являється над інтерактивними об'єктами при отриманні події InteractableFocusedEvent (рис. 3.3).



Рис. 3.3 – Загальний вигляд ігрового інтерфейсу (HUD)

Система інтерактивних екранів реалізована на основі компонента UIScreenStack, що керує стеком об'єктів UIScreen. Відкриття нового екрану приховує попередній через SetActive(false), не видаляючи його зі стеку, що гарантує коректне відновлення екрану та фокусу введення при поверненні назад. Первинною точкою взаємодії користувача є головне меню гри. Воно

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

завантажується після ініціалізації всіх глобальних сервісів та надає доступ до управління слотами збереження, переходу до системних налаштувань і перегляду прогресу гравця (рис. 3.4).



Рис. 3.4 – Інтерфейс головного меню гри

З головного меню користувач має можливість перейти до комплексного меню налаштувань, яке розділене на кілька тематичних вкладок. У вкладці налаштувань аудіо реалізовано управління параметрами звуку, зокрема окреме регулювання рівнів гучності для загального міксу (Master), музичного супроводу (Music) та звукових ефектів (Sound) завдяки прямій взаємодії з підсистемою AudioManager [1] (рис. 3.5).

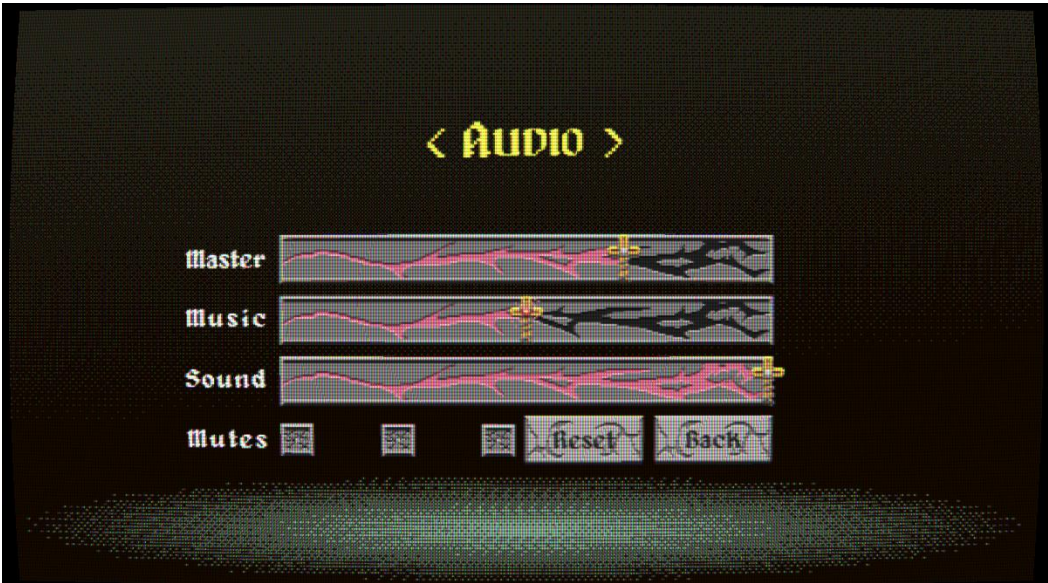


Рис. 3.5 – Вкладка налаштувань аудіо

Вкладка відео налаштувань надає користувачу прямий контроль над параметрами розробленого кастомного CRT-шейдера [26, 28]. Через інтерфейс повзунків можна динамічно регулювати інтенсивність таких візуальних ефектів, як сканлайни (Scanlines), хроматичні аберації (Chromatic Aberration), віньєтування (Vignette) та бочкоподібна дисторсія (Curvature), або повністю вимкнути дію шейдера (рис. 3.6).

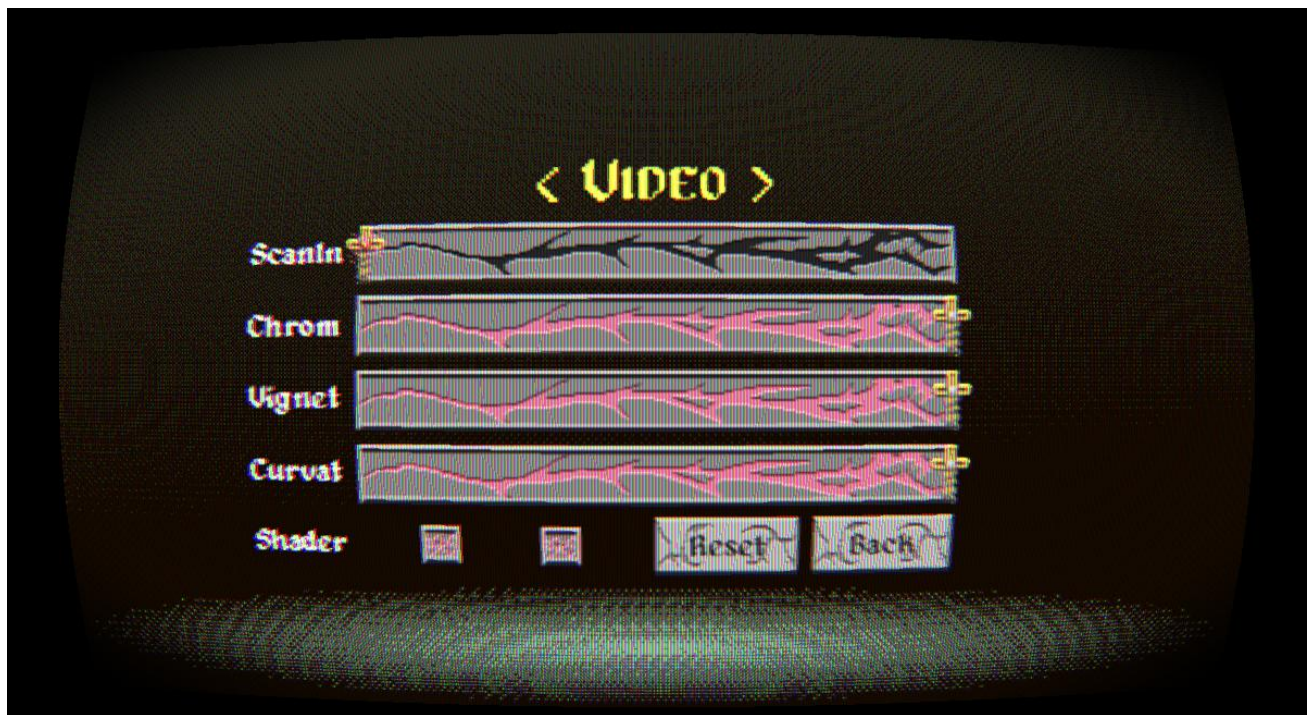


Рис. 3.6 – Вкладка відео налаштувань із параметрами CRT-шейдера

Додатково реалізовано системну вкладку налаштувань (Other), яка дозволяє виконувати критичні операції з даними гравця, зокрема повне видалення всіх слотів збережень із локальної директорії. Вкладка містить кнопку «Delete All Saves», натискання якої ініціює публікацію події через EventBus, після чого SaveManager видаляє всі файли слотів збереження з локальної директорії гравця, але не видаляє файл achievements.json. Для запобігання випадковому видаленню передбачено діалог підтвердження, який блокує взаємодію з рештою інтерфейсу через UIScreenStack до моменту явного підтвердження або скасування операції (рис. 3.7).

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

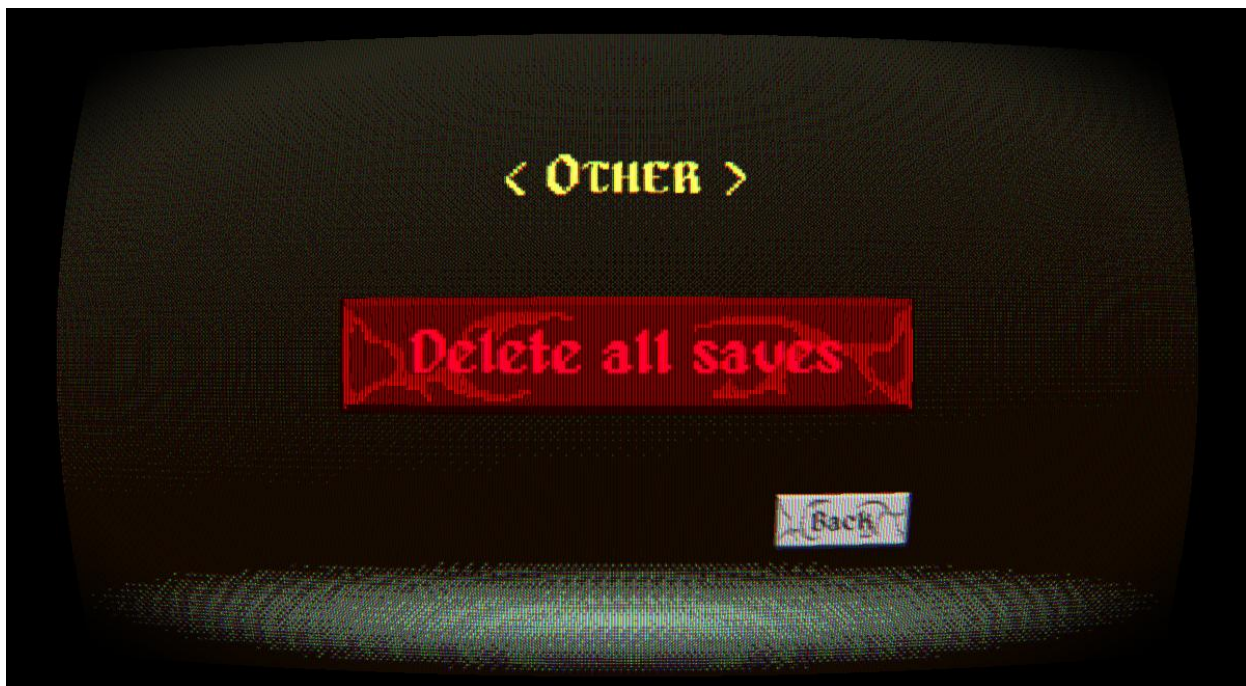


Рис. 3.7 – Системна вкладка налаштувань

Підсистема досягнень має дві форми візуального представлення – локальну та глобальну. При виконанні умови досягнення в реальному часі під час ігрового процесу (наприклад, при публікації BossDefeatedEvent) система генерує локальне сповіщення (toast) у куті екрана за допомогою UnlockToastView, не перериваючи геймплей (рис. 3.8).

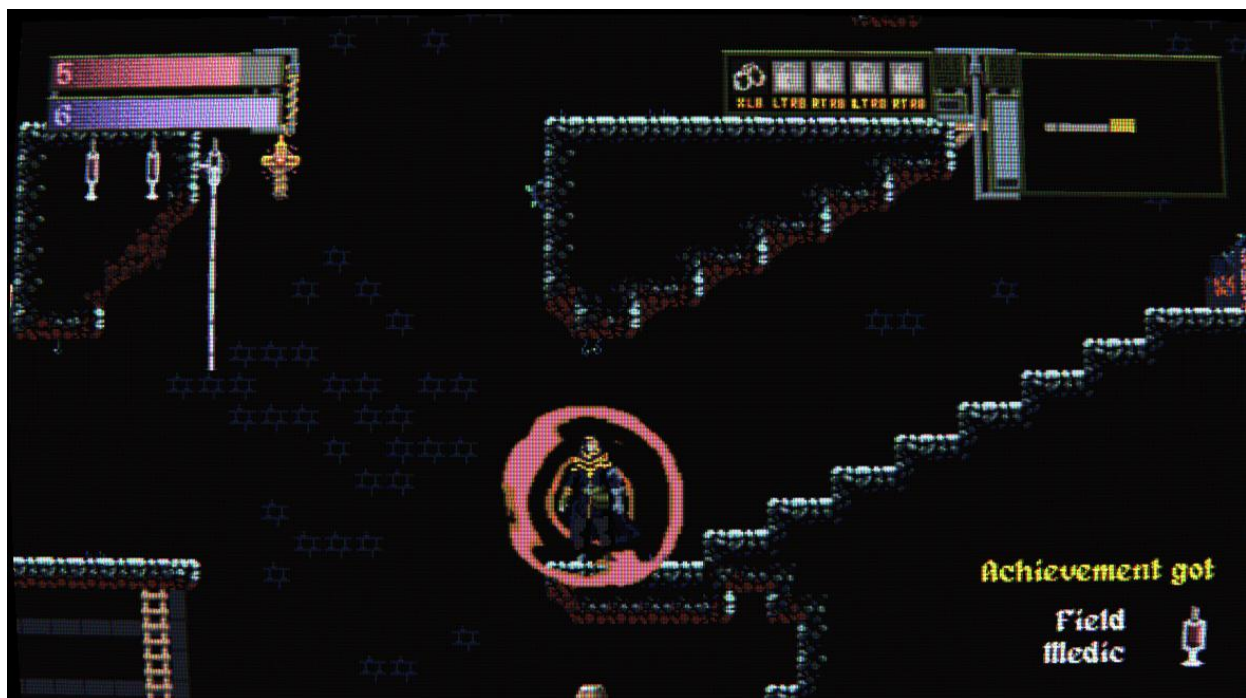


Рис. 3.8 – Локальне сповіщення про отримання досягнення

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

Для детального ознайомлення з прогресом реалізовано екран глобальних досягнень, який взаємодіє з сервісом AchievementService [1, 3]. Цей екран демонструє перелік виконаних та заблокованих завдань гравця, зчитуючи дані з незалежного файлу achievements.json [14] (рис. 3.9).



Рис. 3.9 – Екран глобальних досягнень

Безпосередньо під час ігрового процесу користувачу доступні екран паузи (PauseScreen) та екран карти (MapView). Вони функціонують як незалежні інтерфейсні вікна і викликаються окремими кнопками введення («Start» та «Select» відповідно) [27]. Екран паузи перекриває ігровий процес та зупиняє внутрішньоігровий час, надаючи доступ до опцій продовження гри або виходу в головне меню (рис. 3.10).

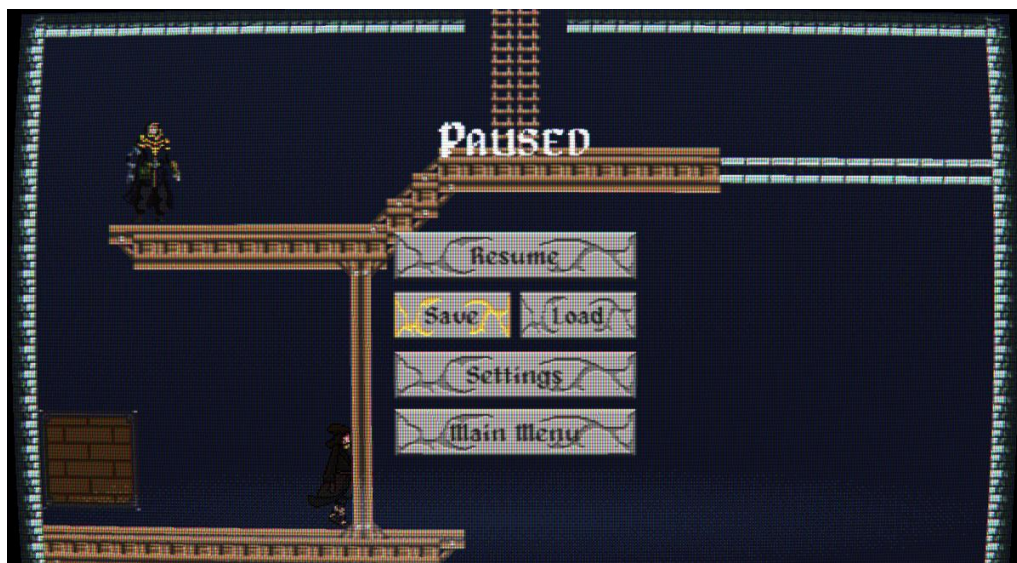


Рис. 3.10 – Інтерфейс меню паузи

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

Натомість незалежний виклик екрана карти відкриває розширену версію дослідженого графа кімнат на основі даних, отриманих із MapDiscoveryService. На відміну від мініатюри в HUD, повноекранна карта відображає усі відкриті кімнати у вигляді вузлів з'єднаного графа між локаціями. Колірне кодування вузлів дозволяє розрізнити відвідану та відкриті кімнати. Не відвідані, та повністю недосліджені кімнати не видно, це надає гравцю лише актуальну інформацію про стан дослідження ігрового світу (рис. 3.11).



Рис. 3.11 – Інтерфейс глобальної ігрової карти

3.3 Тестування програмного продукту

Для забезпечення надійності та стабільності роботи розроблених алгоритмів було проведено комплексне тестування програмного продукту. Враховуючи модульну архітектуру системи, тестування було розділено на два ключові етапи – ізольоване модульне тестування окремих компонентів та інтеграційне функціональне тестування міжмодульної взаємодії в середовищі ігрового рушія.

Ізольоване тестування було застосовано до двох фундаментальних компонентів системи – шини подій EventBus та математичного ядра генерації рівнів LevelLayoutCore. Обидва компоненти реалізовано таким чином, що їхня внутрішня логіка не залежить від підсистем Unity (MonoBehaviour,

		Льїн І.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

ScriptableObject, корутини), що відкриває можливість для верифікації в ізолюваному середовищі. Тести запускалися через вбудований у редактор Unity механізм Unity Test Framework у режимі EditMode, що дозволяє виконувати тести безпосередньо в процесі редагування без запуску ігрової сцени та без компіляції окремого Player-білду.

Для організації тестів було створено окрему збірку FiatLux.Tests з файлом конфігурації FiatLux.Tests.asmdef, в якому визначено посилання на UnityEngine.TestRunner, UnityEditor.TestRunner та цільові збірки FiatLux.World і FiatLux.Contracts. Параметр autoReferenced встановлено у значення false, що унеможливлює потрапляння тестового коду до фінальної збірки продукту. Параметр includePlatforms обмежено значенням Editor, що гарантує компіляцію тестів виключно в контексті редактора Unity.

Тестування EventBus охоплювало перевірку всіх ключових сценаріїв роботи шини подій. Перша група тестів перевіряла коректність механізму PublishLatest – при публікації кешованої події вона зберігається у внутрішньому словнику типу та повертається методом TryGetLatest. Друга група тестів симулювала проблему «пізнього підписника» – сценарій, характерний для систем з відкладеною ініціалізацією компонентів. Ініціювалася публікація кешованої події PublishLatest, після чого створювався новий підписник, який викликав патерн SubscribeWithLatest. Результати тестів підтвердили, що підписник гарантовано отримує останню кешовану подію незалежно від часу своєї ініціалізації відносно публікації. Третя група тестів перевіряла ізоляцію між різними типами подій – підписники на один тип не отримують події інших типів, а при множинних підписах кожен обробник викликається рівно один раз за виклик Publish.

Тестування LevelLayoutCore базувалося на ключовій архітектурній властивості цього компонента – детермінізмі генерації. Оскільки значення seed зберігається у структурі SaveData та використовується при завантаженні гри для відновлення стану рівня, коректність детермінізму є критичною для системи збереження прогресу. Основний тест

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

GenerateLevel_SameSeed_ProducesIdenticalLayout виконував подвійний виклик алгоритму Generate з однаковим значенням seed та порівнював отримані масиви об'єктів GeneratedRoomData за roomId, type та worldPosition кожного елемента. Додаткові тести перевіряли структурні інваріанти. Стартова кімната розміщується в координатах (0, 0), остання кімната завжди має тип BossEncounter, усі посилання між кімнатами в масиві HorizontalLinks є валідними (обидва об'єкти присутні у списку Rooms), жодні дві кімнати не займають однакову позицію у просторі.

Для ізольованого запуску тестів LevelLayoutCore використовувався метод ScriptableObject.CreateInstance<RoomTemplate>() для створення мінімальних тестових шаблонів кімнат без прив'язки до файлової системи Assets. Це дозволило верифікувати логіку алгоритму без залежності від конкретних ресурсів проєкту.

Загальна кількість модульних тестів склала 13. 6 тестів для EventBus та 7 тестів для LevelLayoutCore. За результатами виконання усі 13 тестів отримали статус успішно пройдених у вікні Unity Test Runner [8] (рис. 3.10).

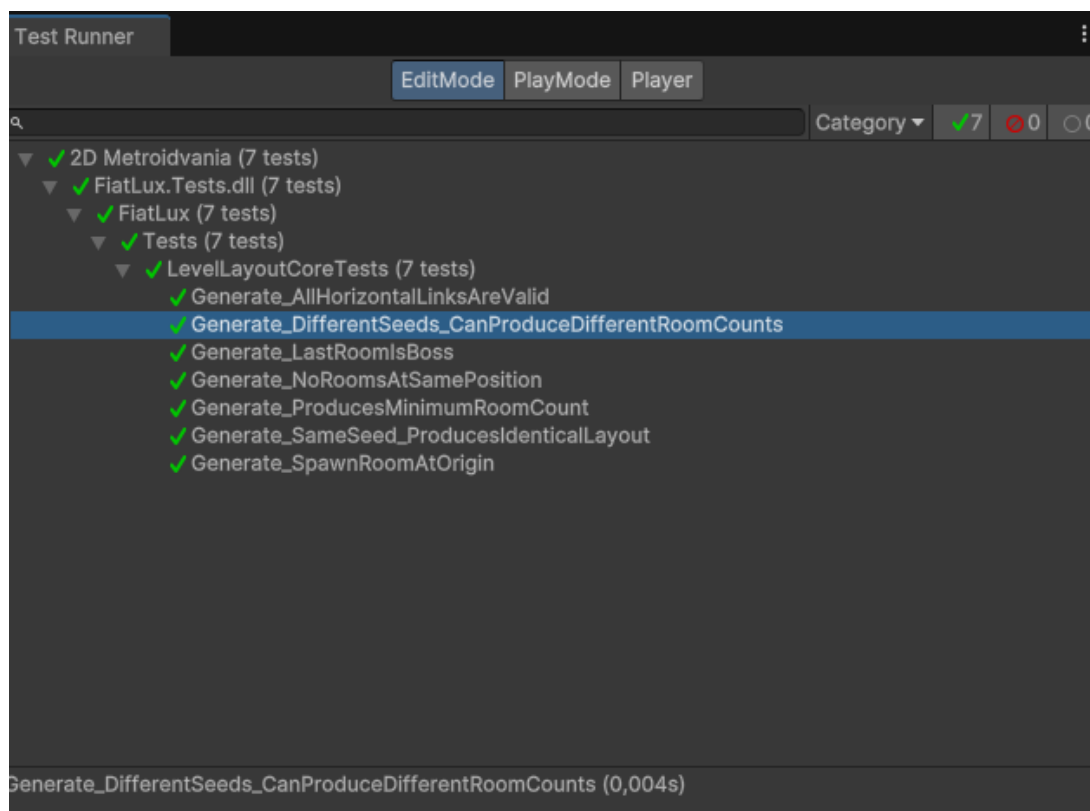


Рис. 3.12 – Результати модульного тестування у вікні Unity Test Runner

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

Інтеграційне функціональне тестування проводилося безпосередньо в середовищі Unity та фокусувалося на перевірці міжмодульної взаємодії ключових підсистем у сценарії, максимально наближеному до умов реального виконання. На відміну від модульних тестів, що верифікують ізольовану логіку окремих класів, функціональні тести перевіряють поведінку системи на рівні взаємодії між компонентами різних збірок через механізм EventBus.

Для проведення тестів використовувався повноцінний Bootstrap Scene з ініціалізованими сервісами SaveManager, AudioManager, RoomManager та CameraController. Кожен тест-кейс описує конкретну вхідну умову, очікувану реакцію системи та зафіксований фактичний результат виконання.

Таблиця 3.3

Результати функціонального тестування ключових підсистем

ID	Опис тест-кейсу	Очікуваний результат	Фактичний результат	Статус
ТС-01	Збереження прогресу: Виклик SaveManager.Save(0) під час перебування гравця в кімнаті з ідентифікатором room_05.	Подія CollectingSaveDataEvent публікується. JSON-файл слота створюється у AppData\LocalLow\FiatLux\Saves\ . У полі roomId записується значення room_05.	Файл save_slot_0.json сформовано коректно. JSON-структура валідна, поле roomId містить room_05. Перевірено ручним розбором файлу через JsonUtility.FromJson.	Успіх
ТС-02	Завантаження стану: Ініціалізація завантаження збереженого слота з головного меню на новій сцені GameScene.	GameManager публікує ApplyingSaveDataEvent лише після виклику OnSceneLoaded. Усі сервіси оновлюють стан без виникнення винятків NullReferenceException.	Сцена завантажилась коректно. InventoryService, MapDiscoveryService та AchievementService отримали подію та синхронізували власний стан. Гравець з'явився на збереженій позиції checkpointX, checkpointY.	Успіх

ТС -03	Перехід між кімнатами: Перетин Rigidbody2D-колайдера гравця з тригерним колайдером об'єкта RoomTeleport з цільовою кімнатою room_07.	Запускається корутина TeleportRoutine. Екран затемнюється через ScreenFader. RoomManager.SetActiveRoom("room_07") викликається. Публікується ActiveRoomChangedEvent (PublishLatest). CameraController оновлює просторові межі.	Екран коректно перейшов у затемнений стан. Гравець переміщений на spawn point кімнати room_07. CameraController отримав подію та застосував нові межі. Екран відновив прозорість. Загальна тривалість переходу – 0.8 сек.	Успіх
ТС -04	Ізоляція пам'яті UIScreenStack: Циклічне відкриття та закриття екрана інвентарю 20 разів поспіль.	Екрани приховуються через SetActive(false), об'єкти не знищуються. Підписки на події залишаються активними після повторного відкриття. Обсяг алокованої пам'яті залишається стабільним.	SetActive(false) викликано коректно при кожному Pop. Підписки на UIStateChangedEvent збереглися активними. За даними Unity Profiler Memory: базовий рівень ~45 MB, після 20 циклів ~45.8 MB. Витік пам'яті відсутній.	Успіх
ТС -05	Спрацювання BossEncounterEvent як PublishLatest: Вхід гравця до кімнати боса після респауну (пізній підписник).	BossHealthBarView підписується через SubscribeWithLatest після публікації події та миттєво отримує кешований стан. Смушка здоров'я боса з'являється на екрані з коректними показниками.	Патерн SubscribeWithLatest коректно доставив кешовану подію пізньому підписнику. BossHealthBarView відобразився на екрані в межах одного кадру після входу до кімнати (затримка < 17 мс при 60 FPS).	Успіх

ТС-06	Архітектурна ізоляція asmdef: Спроба додавання прямого посилання на FiatLux.Gameplay з класу PlayerAnimator (збірка FiatLux.Visual).	Компілятор Unity генерує помилку відсутності посилання на збірку до початку виконання.	Компіляція зупинилась з помилкою: «Assembly FiatLux.Visual does not reference FiatLux.Gameplay». Помилка зафіксована на етапі компіляції, що підтверджує ефективність ізоляції через autoReferenced: false у файлах .asmdef.	Успіх
-------	--	--	--	-------

Результати проведеного комплексного тестування підтверджують надійність та коректність спроектованої архітектури. Усі 13 модульних тестів та 6 функціональних тест-кейсів отримали статус успішно пройдених.

Детермінізм алгоритму генерації рівнів, підтверджений тестами LevelLayoutCore, гарантує ідентичне відтворення структури рівня при завантаженні збереженого прогресу. Це забезпечує цілісність взаємодії між системою збереження SaveManager та процедурним генератором рівнів LevelGenerator.

Механізм PublishLatest та SubscribeWithLatest шини подій EventBus продемонстрував свою ефективність у вирішенні класичної проблеми синхронізації компонентів з відкладеною ініціалізацією (тест ТС-05). Відсутність прямих посилань між компонентами різних шарів, підтверджена тестом ТС-06, унеможливорює випадкове порушення архітектурних меж на рівні компіляції.

Відсутність витоків пам'яті при циклічному відкриванні та закриванні UI-екранів (ТС-04) підтверджує коректність паттерну SetActive замість інстанціювання та знищення об'єктів. Цей підхід зберігає активними підписки на події EventBus протягом усього часу існування об'єкта, що є ключовою умовою коректної роботи event-driven HUD.

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до розділу 3

У третьому розділі було описано особливості налаштування розробленого програмного забезпечення, структуру користувацького інтерфейсу та проведено тестування основних систем гри.

Сформульовано вимоги до середовища розробки та процесу компіляції. Доведено ефективність використання інструментарію Bootstrap Scene, що автоматизує ініціалізацію базових сервісів та мінімізує ризик виникнення помилок на етапі запуску тестових сцен. Описано процедуру формування фінальної збірки продукту під операційну систему Windows.

Проведено аналіз архітектури користувацького інтерфейсу, розділеного на стаціонарний HUD та інтерактивний стек екранів UIScreenStack. Реалізований підхід керування видимістю екранів через метод SetActive дозволив вирішити проблему розриву підписок та зберегти цілісність реакції візуальних елементів на системні події EventBus. Розглянуто ключові візуальні меню: інвентар, карту, систему досягнень та паузу.

Проведене модульне та інтеграційне тестування підтвердило стабільність системи. Ізольовані тести засобами Unity Test Framework у режимі EditMode довели коректність логіки диспетчеризації шини подій та детермінізм процедурного генератора рівнів. Результати функціонального тестування в середовищі Unity, наведені у формі тест-кейсів, засвідчили безперебійну роботу системи збереження прогресу у форматі JSON, стабільність переходів між зонами за допомогою Persistent Room System та надійну ізоляцію шарів на рівні збірок asmdef.

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне інженерне завдання – спроектовано та розроблено програмне забезпечення для двовимірної відеоігри в жанрі Metroidvania під назвою «Fiat Lux» із застосуванням слабко зв'язаної архітектури. Відповідно до поставлених завдань отримано такі результати:

Проведено системний аналіз предметної області та існуючих архітектурних підходів у розробці 2D-відеоігор. Дослідження аналогів дозволило сформулювати вимоги до системи та обґрунтувати необхідність відмови від стандартних високорівневих компонентів-«чорних скриньок» ігрового рушія Unity на користь власних інженерних рішень.

Спроектовано та успішно реалізовано строгу модульну структуру програмного забезпечення з використанням механізму Assembly Definitions (asmdef). Відмова від автоматичного створення посилань гарантувала фізичну ізоляцію візуального та інтерфейсного шарів від логіки геймплею, що повністю унеможливило виникнення циклічних залежностей та значно підвищило надійність і масштабованість кодової бази.

Розроблено та впроваджено систему міжмодульної взаємодії на основі патерну Publish-Subscribe за допомогою власного типізованого статичного EventBus. Реалізовані механізми підписки, публікації та кешування станів (sticky-події) забезпечили ефективну та безпечну синхронізацію життєвого циклу об'єктів без їхнього жорсткого зв'язування в інспекторі.

Реалізовано ключові програмні підсистеми з нуля. Зокрема, створено Persistent Room System для безшовного керування простором, систему камери на основі пріоритетного стеку (ICameraBehaviour) для піксельно-точного позиціонування, архітектуру користувацького інтерфейсу UIScreenStack на основі SetActive та подієво-орієнтовану підсистему збереження даних у формат JSON. Також інтегровано детермінований алгоритм секційної процедурної генерації рівнів, що будує лінійний ігровий світ з опціональними відгалуженнями на основі сіджаного System.Random із суворим

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Лобанчикова Н.М.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

розмежуванням чистого C#-ядра LevelLayoutCore та MonoBehaviour-адаптера LevelGenerator.

Реалізовано подієво-орієнтовану інтеграцію геймплейного шару з візуально-звуковими та сервісними підсистемами. Подієвий компонент PlayerAnimator перемикає анімаційні кліпи, а обробник PlayerSfxHandler відтворює звукові реакції у відповідь на ігрові події без жодних прямих посилань на класи геймплейної логіки, що повністю зберігає ізоляцію збірок та слабку зв'язність шарів.

Проведено комплексне тестування розроблених систем та оптимізацію взаємодії компонентів гри. Результати тестування (зокрема EventBus та UIScreenStack) підтвердили надійність архітектури, відсутність витоків пам'яті та стабільність роботи підсистем при розширенні ігрового світу.

Таким чином, усі поставлені завдання виконано в повному обсязі, а мета кваліфікаційної роботи – створення програмного забезпечення гри зі слабко зв'язаною архітектурою та власними інженерними рішеннями – успішно досягнута. Отримані результати мають практичну цінність і можуть слугувати міцним фундаментом для подальшої розробки комерційних ігрових проєктів.

		Ільїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ільїн І. В., Лобанчикова Н. М., Концидайло А. М. Застосування технік тест-дизайну до верифікації ігрової логіки з розгалуженою структурою станів. Тези Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених «Тенденції розвитку технологій в автоматизації, електроенергетиці, приладобудуванні та робототехніці» (присвяченої Дню науки), м. Житомир, 13-15 травня 2026 р. – Житомир: Житомирська політехніка, 2026. – С. 24-26.
2. Baptista G., Abbruzzese F. Software Architecture with C# 12 and .NET 8. 4th ed. Birmingham : Packt Publishing, 2024. 756 p.
3. Sarcar V. Design Patterns in C#: A Hands-on Guide with Real-World Examples. 3rd ed. Berkeley: Apress, 2023. 450 p.
4. Ferrone H. Learning Design Patterns with Unity. Birmingham : Packt Publishing, 2024. 676 p.
5. Dickinson J. Unity 2022 by Example. Birmingham: Packt Publishing, 2023. 412 p.
6. Bateman J. Clean Code in C#: Refactor your legacy C# code base and improve application performance. 2nd ed. Birmingham: Packt Publishing, 2022. 466 p.
7. Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 11th ed. Berkeley: Apress, 2022. 1300 p.
8. Osherove R., Khorikov V. The Art of Unit Testing. 3rd ed. Shelter Island: Manning Publications, 2023. 360 p.
9. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. Cambridge: MIT Press, 2022. 1312 p.
10. Paris J. Procedural Content Generation for Unity Game Development. 2nd ed. Birmingham: Packt Publishing, 2022. 280 p.
11. Aversa D., Pytel D., Sposato M. Unity Artificial Intelligence Programming. 5th ed. Birmingham: Packt Publishing, 2022. 358 p.
12. ECMA International. C# Language Specification. Standard ECMA-334. 7th ed. Geneva, 2023. 532 p.

		Ільїн І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

13. Albahari J., Albahari B. C# 10 in a Nutshell: The Definitive Reference. Sebastopol: O'Reilly Media, 2022. 1064 p.
14. Microsoft. .NET Standard documentation [Електронний ресурс]. 2025. URL: <https://learn.microsoft.com/en-us/dotnet/standard/net-standard> (дата звернення: 10.05.2026).
15. Microsoft. JSON Serialization and Deserialization in .NET [Електронний ресурс]. 2025. URL: <https://learn.microsoft.com/en-us/dotnet/standard/serialization/> (дата звернення: 10.05.2026).
16. Zhang X., Li Y., Wang Z. An auxiliary development framework for lightweight RPG games based on Unity3D. Computer Animation and Virtual Worlds. 2024. Vol. 35, No. 3. DOI: 10.1002/cav.2206. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.2206> (дата звернення: 10.05.2026).
17. Folmer E. Component Based Game Development - A Solution to Escalating Costs and Expanding Deadlines? // Component-Based Software Engineering: 10th International Symposium, CBSE 2007. Berlin : Springer, 2007. P. 66-73.
18. Oliveira R., Pereira L., Clua E. A Framework for Metroidvania Games // Proceedings of SBGames 2020. Recife : SBC, 2020. P. 844-853.
19. Марчук Г. В., Кірій Д. О. Механіки та дизайн рівнів в жанрах «Metroidvania» та «Soulslike». Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. 2024. Т. 35 (74), № 1. С. 87-92.
20. Гришко Н. С., Дубровін В. І., Дейнега Л. Ю. Розробка ігрових застосунків. Прикладні питання математичного моделювання. 2025. Т. 8, № 1. С. 45-56.
21. Schell J. The Art of Game Design: A Book of Lenses. 4th ed. Boca Raton: CRC Press, 2024. 650 p.
22. Fullerton T. Game Design Workshop: A Playcentric Approach to Creating Innovative Games. 4th ed. Boca Raton: CRC Press, 2023. 560 p.
23. Adams E. Fundamentals of Game Design. 4th ed. Berkeley: New Riders, 2023. 580 p.

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

- 24.Team Cherry. Hollow Knight [Електронний ресурс]. 2026. URL: <https://www.hollowknight.com/> (дата звернення: 10.05.2026).
- 25.The Game Kitchen. Blasphemous [Електронний ресурс]. 2026. URL: <https://thegamekitchen.com/blasphemous/> (дата звернення: 10.05.2026).
- 26.Moon Studios. Ori and the Blind Forest [Електронний ресурс]. 2026. URL: <https://www.orithegame.com/> (дата звернення: 10.05.2026).
- 27.Unity Technologies. Universal Render Pipeline (URP) [Електронний ресурс]. 2026. URL: <https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@17.0/manual/> (дата звернення: 10.05.2026).
- 28.Unity Technologies. Unity Input System [Електронний ресурс]. 2026. URL: <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/> (дата звернення: 10.05.2026).
- 29.Unity Technologies. Writing custom shaders in URP [Електронний ресурс]. 2026. URL: <https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@17.0/manual/writing-shaders-urp.html> (дата звернення: 10.05.2026).
- 30.Aseprite. Aseprite Documentation [Електронний ресурс]. 2026. URL: <https://www.aseprite.org/docs/> (дата звернення: 10.05.2026).
- 31.Smack Studio. Smack Studio Official Website [Електронний ресурс]. 2026. URL: <https://smackstudio.com/> (дата звернення: 10.05.2026).
- 32.Microsoft. Visual Studio Code [Електронний ресурс]. 2026. URL: <https://code.visualstudio.com/> (дата звернення: 10.05.2026).

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Гльїн І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг класу LevelLayoutCore.cs

```

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

namespace FiatLux.World
{
    /// <summary>
    /// Pure C# layout algorithm. Has zero dependency on UnityEngine or
    MonoBehaviour.
    /// Accepts a <see cref="LevelLayoutConfig"/> and a seeded <see
    cref="System.Random"/>;
    /// returns a <see cref="LevelLayoutResult"/> containing room
    positions and link pairs.
    ///
    /// Separation rationale (ADR): keeping layout logic in plain C#
    makes it
    /// fully unit-testable, re-runnable without a scene, and clearly
    owned by
    /// the architecture layer rather than the Unity object graph.
    /// </summary>
    public sealed class LevelLayoutCore
    {
        // — private state (reset on every Generate call)

        private LevelLayoutConfig _cfg;
        private System.Random _rng;

        private readonly
        List<GeneratedRoomData> _layout =
        new();
        private readonly List<(GeneratedRoomData lower, GeneratedRoomData
        upper)> _verticalLinks = new();
        private readonly List<(GeneratedRoomData left, GeneratedRoomData
        right)> _horizontalLinks = new();
        private readonly
        List<FRect> _occupied =
        new();

        private RoomTemplate _lastLinearTemplate;
        private readonly List<RoomTemplate> _usedLeftRewards = new();

        // — public API

        /// <summary>

```

```

        /// Runs the procedural layout algorithm and returns the complete
result.
        /// Deterministic: the same <paramref name="seed"/> always
produces the same layout.
        /// </summary>
        /// <param name="config">
        ///     All template arrays and tuning parameters - collected by
the MonoBehaviour adapter.
        /// </param>
        /// <param name="seed">
        ///     Integer seed forwarded from <c>SaveData</c>; stored so the
level can be
        ///     reconstructed identically after a load.
        /// </param>
        /// <returns>
        ///     A <see cref="LevelLayoutResult"/> with the flat room list
and two link lists
        ///     (horizontal left↔right, vertical lower↔upper) ready for
teleport wiring.
        /// </returns>
public LevelLayoutResult Generate(LevelLayoutConfig config, int
seed)
{
    _cfg = config;
    _rng = new System.Random(seed);

    _layout.Clear();
    _verticalLinks.Clear();
    _horizontalLinks.Clear();
    _occupied.Clear();
    _usedLeftRewards.Clear();
    _lastLinearTemplate = null;

    // 1. Spawn room at world origin
    var prev = CreateRoom(config.SpawnTemplate, FVec2.Zero,
"room_1");
    _layout.Add(prev);

    // 1b. Intro linear rooms before the dash ability (no chord
waves - player has no dash yet)
    int introCount = RangeInclusive(_cfg.MinIntroLinearRooms,
_cfg.MaxIntroLinearRooms);
    for (int j = 0; j < introCount; j++)
    {
        var tml = GetLinearOnly();
        var pos = PlaceNextTo(prev, tml, Direction.Right);
        var room = CreateRoom(tml, pos, $"Room_Intro_{j}");
        _layout.Add(room);
        _horizontalLinks.Add((prev, room));
        prev = room;
    }
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

point        // 1c. Dash ability room, followed immediately by a save
              if (config.DashRoomTemplate != null)
              {
                  var dashPos = PlaceNextTo(prev, config.DashRoomTemplate,
Direction.Right);
                  var dash    = CreateRoom(config.DashRoomTemplate,
dashPos, "Room_Dash");
                  _layout.Add(dash);
                  _horizontalLinks.Add((prev, dash));
                  prev = dash;

                  if (config.SavePointTemplate != null)
                  {
                      var savePos = PlaceNextTo(prev,
config.SavePointTemplate, Direction.Right);
                      var save    = CreateRoom(config.SavePointTemplate,
savePos, "Room_DashSave");
                      _layout.Add(save);
                      _horizontalLinks.Add((prev, save));
                      prev = save;
                  }
              }

              // 2. One section per remaining ability unlock
              for (int i = 0; i < config.TreasureAbilityTemplates.Length;
i++)
              {
                  // Linear / chord filler rooms
                  int linearCount =
RangeInclusive(_cfg.MinLinearRoomsPerSection,
_cfg.MaxLinearRoomsPerSection);
                  for (int j = 0; j < linearCount; j++)
                  {
                      var tmp1 = GetRandomLinearTemplate();
                      var pos  = PlaceNextTo(prev, tmp1, Direction.Right);
                      var room = CreateRoom(tmp1, pos,
$"Room_Linear_{i}_{j}");
                      _layout.Add(room);
                      _horizontalLinks.Add((prev, room));
                      prev = room;
                  }

                  // Optional fork: main line continues right, vertical
branch grows upward
                  if (_cfg.ForkChance > 0f
                      && NextDouble() < _cfg.ForkChance
                      && _cfg.ForkTemplates.Length > 0
                      && _cfg.BranchTopTemplates.Length > 0)
                  {

```

		Льїн І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        var forkTpl = Pick(_cfg.ForkTemplates);
        var forkPos = PlaceNextTo(prev, forkTpl,
Direction.Right);
        var fork = CreateRoom(forkTpl, forkPos,
$"Room_Fork_{i}");
        _layout.Add(fork);
        _horizontalLinks.Add((prev, fork));

        BuildUpwardBranch(fork, i);

        prev = fork;
    }

    // Ability unlock room
    var abilityTpl = config.TreasureAbilityTemplates[i];
    var abilityPos = PlaceNextTo(prev, abilityTpl,
Direction.Right);
    var ability = CreateRoom(abilityTpl, abilityPos,
$"Room_Ability_{i}");
    _layout.Add(ability);
    _horizontalLinks.Add((prev, ability));
    prev = ability;

    // Save point after each ability
    if (config.SavePointTemplate != null)
    {
        var savePos = PlaceNextTo(prev,
config.SavePointTemplate, Direction.Right);
        var save = CreateRoom(config.SavePointTemplate,
savePos, $"Room_Save_{i}");
        _layout.Add(save);
        _horizontalLinks.Add((prev, save));
        prev = save;
    }
}

// 3. Boss room at the end of the main corridor
var bossPos = PlaceNextTo(prev, config.BossTemplate,
Direction.Right);
var boss = CreateRoom(config.BossTemplate, bossPos,
"Room_Boss");
_layout.Add(boss);
_horizontalLinks.Add((prev, boss));

return new LevelLayoutResult(
    new List<GeneratedRoomData>(_layout),
    new List<GeneratedRoomData,
GeneratedRoomData>(_horizontalLinks),
    new List<GeneratedRoomData,
GeneratedRoomData>(_verticalLinks));
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

// — branch builders

```

    /// <summary>
    /// Builds a vertical column above <paramref name="fork"/>, then
attempts side branches
    /// (left reward + right reward) off the column's topmost room.
    /// </summary>
    private void BuildUpwardBranch(GeneratedRoomData fork, int
forkIndex)
    {
        int length      = RangeInclusive(_cfg.MinBranchLength,
_cfg.MaxBranchLength);
        var prevInBranch = fork;

        // Optional mid rooms (need Bottom + Top exits)
        for (int k = 0; k < length - 1 &&
_cfg.BranchMidTemplates.Length > 0; k++)
        {
            var midTpl = Pick(_cfg.BranchMidTemplates);
            var midPos  = PlaceNextTo(prevInBranch, midTpl,
Direction.Top);
            var mid     = CreateRoom(midTpl, midPos,
$"Room_Branch_{forkIndex}_{k}");
            _layout.Add(mid);
            _verticalLinks.Add((prevInBranch, mid));
            prevInBranch = mid;
        }

        // Top cap room
        var topTpl = Pick(_cfg.BranchTopTemplates);
        var topPos  = PlaceNextTo(prevInBranch, topTpl,
Direction.Top);
        var top     = CreateRoom(topTpl, topPos,
$"Room_BranchTop_{forkIndex}");
        _layout.Add(top);
        _verticalLinks.Add((prevInBranch, top));

        // Side reward branches off the top cap
        TrySideBranch(top, Direction.Right, _cfg.SideRewardRight,
forkIndex);
        TrySideBranch(top,
Direction.Left, PickLeftReward(), forkIndex);
    }

    /// <summary>
    /// Attempts to attach one connector room + one reward room in
direction <paramref name="dir"/>
    /// from <paramref name="top"/>. Silently skipped if either
template is null or space is occupied.

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    /// </summary>
    private void TrySideBranch(GeneratedRoomData top, Direction dir,
                               RoomTemplate rewardTmpl, int
forkIndex)
    {
        if (_cfg.SideBranchTemplates.Length == 0 || rewardTmpl ==
null)
            return;

        var sideTmpl = Pick(_cfg.SideBranchTemplates);
        var sidePos = PlaceNextTo(top, sideTmpl, dir);

        if (!IsFree(sidePos, sideTmpl.roomSizeInTiles))
            return;

        // Probe reward position using a temporary data object (not
added to layout)
        var sideTmp = new GeneratedRoomData("_tmp", sideTmpl,
ToUnityVec(sidePos));
        var rewardPos = PlaceNextTo(sideTmp, rewardTmpl, dir);

        if (!IsFree(rewardPos, rewardTmpl.roomSizeInTiles))
            return;

        string tag = dir == Direction.Right ? "R" : "L";
        var side =
CreateRoom(sideTmpl, sidePos, $"Room_Side_{forkIndex}_{tag}");
        var reward = CreateRoom(rewardTmpl, rewardPos,
$"Room_SideReward_{forkIndex}_{tag}");

        _layout.Add(side);
        AddHorizontalLink(top, side, dir);

        _layout.Add(reward);
        AddHorizontalLink(side, reward, dir);
    }

    // — room creation & placement

```

```

    /// <summary>
    /// Allocates a <see cref="GeneratedRoomData"/> at <paramref
name="position"/>,
    /// computes its exit points from the template flags, and
registers its bounding
    /// rectangle in the collision list.
    /// </summary>
    private GeneratedRoomData CreateRoom(RoomTemplate template, FVec2
position, string roomId)
    {

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        var room = new GeneratedRoomData(roomId, template,
ToUnityVec(position));

        float halfW = template.roomSizeInTiles.x * _cfg.TileSize *
0.5f;
        float halfH = template.roomSizeInTiles.y * _cfg.TileSize *
0.5f;

        if (template.hasExitLeft)
            room.exits.Add(new ExitPoint(Direction.Left,
ToUnityVec(position + new FVec2(-halfW, 0f)),
roomId));
        if (template.hasExitRight)
            room.exits.Add(new ExitPoint(Direction.Right,
ToUnityVec(position + new FVec2( halfW, 0f)),
roomId));
        if (template.hasExitTop)
            room.exits.Add(new ExitPoint(Direction.Top,
ToUnityVec(position + new FVec2(0f, halfH)),
roomId));
        if (template.hasExitBottom)
            room.exits.Add(new ExitPoint(Direction.Bottom,
ToUnityVec(position + new FVec2(0f, - halfH)),
roomId));

        _occupied.Add(new FRect(
            position.x - halfW - LevelLayoutConfig.RoomPadding,
            position.y - halfH - LevelLayoutConfig.RoomPadding,
            (halfW + LevelLayoutConfig.RoomPadding) * 2f,
            (halfH + LevelLayoutConfig.RoomPadding) * 2f));

        return room;
    }

    // Overload for FVec2 entry point (internal helper)
    private GeneratedRoomData CreateRoom(RoomTemplate template,
UnityEngine.Vector2 position, string roomId)
        => CreateRoom(template, ToFVec2(position), roomId);

    /// <summary>
    /// Returns the world-space centre for a room of type <paramref
name="bTemplate"/>
    /// placed directly adjacent (in direction <paramref
name="exitDir"/>) to room <paramref name="a"/>.
    /// Floors are aligned: both rooms share the same bottom Y unless
the direction is vertical.
    /// </summary>
    private FVec2 PlaceNextTo(GeneratedRoomData a, RoomTemplate
bTemplate, Direction exitDir)
    {
        float halfW_A = a.sizeInTiles.x * _cfg.TileSize * 0.5f;

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

float halfH_A = a.sizeInTiles.y * _cfg.TileSize * 0.5f;
float halfW_B = bTemplate.roomSizeInTiles.x * _cfg.TileSize *
0.5f;
float halfH_B = bTemplate.roomSizeInTiles.y * _cfg.TileSize *
0.5f;

float ax      = a.worldPosition.x;
float ay      = a.worldPosition.y;
float floorY  = ay - halfH_A;

return exitDir switch
{
    Direction.Right => new FVec2(ax + halfW_A +
LevelLayoutConfig.RoomPadding + halfW_B, floorY + halfH_B),
    Direction.Left  => new FVec2(ax - halfW_A -
LevelLayoutConfig.RoomPadding - halfW_B, floorY + halfH_B),
    Direction.Top    => new FVec2(ax, ay + halfH_A +
LevelLayoutConfig.RoomPadding + halfH_B),
    Direction.Bottom => new FVec2(ax, ay - halfH_A -
LevelLayoutConfig.RoomPadding - halfH_B),
    -                  => new FVec2(ax, ay)
};
}

private bool IsFree(FVec2 center, UnityEngine.Vector2Int
sizeTiles)
{
    float halfW = sizeTiles.x * _cfg.TileSize * 0.5f;
    float halfH = sizeTiles.y * _cfg.TileSize * 0.5f;
    var candidate = new FRect(center.x - halfW, center.y - halfH,
halfW * 2f, halfH * 2f);
    foreach (var occ in _occupied)
        if (candidate.Overlaps(occ))
            return false;
    return true;
}

// — template pickers

```

```

/// <summary>
/// Picks a linear room, intentionally excluding chord-wave
rooms.
/// Used for the intro segment before the player has the dash
ability.
/// Also enforces no-consecutive-repeat to avoid visual monotony.
/// </summary>
private RoomTemplate GetLinearOnly()
{
    if (_cfg.LinearEnemyTemplates.Length == 1)
    {

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        _lastLinearTemplate = _cfg.LinearEnemyTemplates[0];
        return _lastLinearTemplate;
    }

    RoomTemplate pick;
    do { pick = Pick(_cfg.LinearEnemyTemplates); }
    while (pick == _lastLinearTemplate);

    _lastLinearTemplate = pick;
    return pick;
}

/// <summary>
/// Picks either a chord-wave room (with probability <see
cref="LevelLayoutConfig.ChordWavesChance"/>)
/// or a linear room without consecutive repeats.
/// </summary>
private RoomTemplate GetRandomLinearTemplate()
{
    if (_cfg.ChordWavesChance > 0f
        && NextDouble() < _cfg.ChordWavesChance
        && _cfg.ChordWavesTemplates.Length > 0)
        return Pick(_cfg.ChordWavesTemplates);

    return GetLinearOnly();
}

/// <summary>
/// Returns a left-branch reward template that has not been used
yet in this level,
/// guaranteeing no duplicate rewards across fork branches.
/// Returns <c>null</c> if all variants are exhausted.
/// </summary>
private RoomTemplate PickLeftReward()
{
    if (_cfg.SideRewardLeftTemplates == null ||
        _cfg.SideRewardLeftTemplates.Length == 0)
        return null;

    var available = _cfg.SideRewardLeftTemplates
        .Where(t => !_usedLeftRewards.Contains(t))
        .ToList();

    if (available.Count == 0)
        return null;

    var pick = available[_rng.Next(available.Count)];
    _usedLeftRewards.Add(pick);
    return pick;
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

// — link helpers

```
private void AddHorizontalLink(GeneratedRoomData from,
GeneratedRoomData to, Direction dir)
```

```
{
    if (dir == Direction.Right)
        _horizontalLinks.Add((from, to));
    else
        _horizontalLinks.Add((to, from));
}
```

// — math helpers (pure C# - no UnityEngine.Mathf / Vector2)

```
private int    RangeInclusive(int min, int max) => _rng.Next(min,
max + 1);
```

```
private double NextDouble()                =>
_rng.NextDouble();
```

```
private I      Pick<I>(I[] arr)          =>
arr[_rng.Next(arr.Length)];
```

```
// Thin bridge types so the algorithm never imports UnityEngine
private static FVec2  ToFVec2(UnityEngine.Vector2 v) => new
FVec2(v.x, v.y);
```

```
private static UnityEngine.Vector2 ToUnityVec(FVec2 v) => new
UnityEngine.Vector2(v.x, v.y);
```

// — bridge-local value types

```
/// <summary>Minimal 2-component float vector used internally to
avoid UnityEngine import.</summary>
```

```
private readonly struct FVec2
{
    public readonly float x, y;
    public FVec2(float x, float y) { this.x = x; this.y = y; }
    public static FVec2 Zero => new FVec2(0f, 0f);
    public static FVec2 operator +(FVec2 a, FVec2 b) => new
FVec2(a.x + b.x, a.y + b.y);
}
```

```
/// <summary>Axis-aligned rectangle used for overlap detection
(no UnityEngine.Rect dependency).</summary>
```

```
private readonly struct FRect
{
    public readonly float x, y, w, h;
    public FRect(float x, float y, float w, float h) { this.x =
x; this.y = y; this.w = w; this.h = h; }
```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        /// <summary>Returns true when this rectangle overlaps
        <paramref name="other"/> (touching edges excluded).</summary>
        public bool Overlaps(FRect other)
            => x < other.x + other.w && x + w > other.x
            && y < other.y + other.h && y + h > other.y;
    }
}

```

Додаток Б

Лістинг класу LevelGenerator.cs

```

using System.Collections;
using UnityEngine;
using FiatLux.Core.Events;

namespace FiatLux.World
{
    /// <summary>
    /// MonoBehaviour adapter for the procedural level-layout system.
    ///
    /// Responsibilities (this class only):
    /// 1. Expose all designer-facing parameters via
    <c>[SerializeField]</c> for the Unity inspector.
    /// 2. Pack those parameters into a <see cref="LevelLayoutConfig"/>
    value and
    /// delegate the entire layout algorithm to <see
    cref="LevelLayoutCore"/> – pure C#.
    /// 3. Receive the resulting <see cref="LevelLayoutResult"/> and
    perform Unity-specific work:
    /// prefab instantiation, teleport wiring, RoomManager
    initialisation, EventBus publish.
    ///
    /// What this class intentionally does NOT do:
    /// - No layout math (→ <see cref="LevelLayoutCore"/>).
    /// - No data model (→ <see cref="GeneratedRoomData"/>, <see
    cref="LevelLayoutConfig"/>).
    /// - No random seeding (seed comes from <c>GameManager</c> via
    <see cref="GenerateLevelLayout"/>).
    /// </summary>
    public class LevelGenerator : MonoBehaviour
    {
        // — Inspector: fixed structural rooms

        [SerializeField] private RoomTemplate _spawnTemplate;
        [SerializeField] private RoomTemplate _dashRoomTemplate;
        [SerializeField] private RoomTemplate _savePointTemplate;
        [SerializeField] private RoomTemplate _bossTemplate;
    }
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

// — Inspector: pool arrays

```
[SerializeField] private RoomTemplate[] _linearEnemyTemplates;  
[SerializeField] private RoomTemplate[] _chordWavesTemplates;
```

```
/// <summary>One entry per ability unlock; determines the total  
number of main-corridor sections.</summary>
```

```
[SerializeField] private RoomTemplate[]  
_treasureAbilityTemplates;  
[SerializeField] private RoomTemplate[] _forkTemplates;
```

// — Inspector: branch templates

```
[Header("Fork branch (vertical column)")]  
[SerializeField] private RoomTemplate[] _branchMidTemplates;  
[SerializeField] private RoomTemplate[] _branchTopTemplates;  
[SerializeField] private int _minBranchLength = 1;  
[SerializeField] private int _maxBranchLength = 2;  
  
[Header("Fork side branches (left and right off the top cap)")]  
[SerializeField] private RoomTemplate[] _sideBranchTemplates;  
[SerializeField] private RoomTemplate[] _sideRewardLeftTemplates;  
[SerializeField] private RoomTemplate _sideRewardRight;
```

// — Inspector: tuning

```
[SerializeField] private int _minLinearRoomsPerSection = 1;  
[SerializeField] private int _maxLinearRoomsPerSection = 3;
```

```
[Header("Intro (spawn → dash): linear rooms only, no chord  
waves")]
```

```
[SerializeField] private int _minIntroLinearRooms = 1;  
[SerializeField] private int _maxIntroLinearRooms = 2;
```

```
[SerializeField] private float _chordWavesChance = 0.3f;  
[SerializeField] private float _forkChance = 0.2f;
```

```
/// <summary>16 PPU → 1 world unit = 4 tiles → tileSize =  
0.25.</summary>
```

```
[SerializeField] private float _tileSize = 0.25f;
```

// — private runtime state

```
private readonly LevelLayoutCore _core = new LevelLayoutCore();
```

// — public API

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    /// <summary>
    /// Entry point called by <c>GameManager</c> (never from
    <c>Start</c>) to avoid
    /// a race with <c>OnSceneLoaded</c>.
    /// Packs inspector fields → <see cref="LevelLayoutConfig"/> →
    <see cref="LevelLayoutCore.Generate"/>
    /// → spawns prefabs → wires teleports → initialises RoomManager
    → publishes event.
    /// </summary>
    /// <param name="seed">
    /// Deterministic seed from <c>SaveData</c>. The same seed always
    produces the same layout.
    /// </param>
    public void GenerateLevelLayout(int seed)
    {
        Debug.Log($"[LevelGenerator] Generating with seed {seed}");

        // 1. Pack all inspector parameters into the pure-C# config
value
        var config = BuildConfig();

        // 2. Run the pure-C# layout algorithm
        var result = _core.Generate(config, seed);

        Debug.Log($"[LevelGenerator] Generated {result.RoomCount}
rooms");

        // 3. Unity-side: instantiate prefabs, wire teleports,
persist graph, notify systems
        SpawnRoomsInHierarchy(result);
        LinkRoomTeleports(result);
        SaveLayoutToJSON(result);

        StartCoroutine(SetInitialRoomDeferred("room_1"));
        EventBus.Publish(new LevelGeneratedEvent(result.RoomCount));
    }

    // — config builder

```

```

    /// <summary>
    /// Collects all serialized fields into a <see
    cref="LevelLayoutConfig"/> snapshot.
    /// This is the only place that bridges the inspector and the
    pure-C# layer.
    /// </summary>
    private LevelLayoutConfig BuildConfig() => new LevelLayoutConfig(
        spawnTemplate:         _spawnTemplate,
        dashRoomTemplate:      _dashRoomTemplate,
        savePointTemplate:     _savePointTemplate,

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

bossTemplate:                _bossTemplate,
linearEnemyTemplates:        _linearEnemyTemplates,
chordWavesTemplates:         _chordWavesTemplates,
treasureAbilityTemplates:    _treasureAbilityTemplates,
forkTemplates:               _forkTemplates,
branchMidTemplates:          _branchMidTemplates,
branchTopTemplates:          _branchTopTemplates,
sideBranchTemplates:         _sideBranchTemplates,
sideRewardLeftTemplates:     _sideRewardLeftTemplates,
sideRewardRight:             _sideRewardRight,
minIntroLinearRooms:         _minIntroLinearRooms,
maxIntroLinearRooms:         _maxIntroLinearRooms,
minLinearRoomsPerSection:    _minLinearRoomsPerSection,
maxLinearRoomsPerSection:    _maxLinearRoomsPerSection,
minBranchLength:             _minBranchLength,
maxBranchLength:             _maxBranchLength,
forkChance:                  _forkChance,
chordWavesChance:            _chordWavesChance,
tileSize:                    _tileSize);

```

```
// — Unity-side work (uses MonoBehaviour APIs)
```

```

/// <summary>
/// Instantiates a prefab for every room in the result, parents
them under a single
/// container GameObject, and caches each <see cref="Room"/>
component reference
/// back onto its <see cref="GeneratedRoomData"/>.
/// </summary>
private void SpawnRoomsInHierarchy(LevelLayoutResult result)
{
    var container = new GameObject("GeneratedRooms");
    container.SetActive(false);

    foreach (var data in result.Rooms)
    {
        var go = Instantiate(data.template.roomPrefab,
                             data.worldPosition,
                             Quaternion.identity,
                             container.transform);
        go.name = data.roomId;

        var room = go.GetComponent<Room>();
        room.RoomId = data.roomId;
        room.CacheTeleports();
        data.runtimeInstance = room;
    }

    container.SetActive(true);
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    /// <summary>
    /// Iterates the link lists from the result and calls
    /// <see cref="LinkTeleport"/> for each directed pair, then seals
any unlinked doors.
    /// </summary>
    private void LinkRoomTeleports(LevelLayoutResult result)
    {
        foreach (var (left, right) in result.HorizontalLinks)
        {
            LinkTeleport(left, Direction.Right, right);
            LinkTeleport(right, Direction.Left, left);
        }

        foreach (var (lower, upper) in result.VerticalLinks)
        {
            LinkTeleport(lower, Direction.Top, upper);
            LinkTeleport(upper, Direction.Bottom, lower);
        }

        SealUnlinkedDoors(result);
    }

    private void SealUnlinkedDoors(LevelLayoutResult result)
    {
        foreach (var data in result.Rooms)
        {
            if (data.runtimeInstance == null) continue;
            foreach (var tp in data.runtimeInstance.Teleports)
                if (string.IsNullOrEmpty(tp.TargetRoomId))
                    tp.Seal();
        }
    }

    private void LinkTeleport(GeneratedRoomData from, Direction
exitDir, GeneratedRoomData to)
    {
        if (from.runtimeInstance == null || to.runtimeInstance ==
null) return;

        var door = from.runtimeInstance.GetTeleport(exitDir);
        if (door == null)
        {
            Debug.LogWarning($"[LevelGenerator] '{from.roomId}': no
door {exitDir}");
            return;
        }

        var arrival =
to.runtimeInstance.GetArrivalPoint(Opposite(exitDir));
        if (arrival == null)

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        {
            Debug.LogWarning($"[LevelGenerator] '{to.roomId}': no
ArrivalPoint from {Opposite(exitDir)}");
            return;
        }

        door.LinkToRoom(to.roomId, arrival);
    }

    private static Direction Opposite(Direction d) => d switch
    {
        Direction.Left    => Direction.Right,
        Direction.Right   => Direction.Left,
        Direction.Top     => Direction.Bottom,
        Direction.Bottom => Direction.Top,
        -                  => d
    };

    /// <summary>
    /// Serialises the layout to
    <c>Application.persistentDataPath/level_graph.json</c>
    /// for external tooling and post-mortem debugging.
    /// </summary>
    private void SaveLayoutToJSON(LevelLayoutResult result)
    {
        var graphData = new LevelGraphData { spawnRoomId = "room_1"
};

        foreach (var room in result.Rooms)
        {
            graphData.rooms.Add(new LevelGraphData.SerializedRoom
            {
                id      = room.roomId,
                type    = room.type.ToString(),
                posX    = room.worldPosition.x,
                posY    = room.worldPosition.y,
                sizeX    = room.sizeInTiles.x,
                sizeY    = room.sizeInTiles.y
            });

            foreach (var exit in room.exits)
                graphData.links.Add(new LevelGraphData.SerializedLink
                {
                    fromRoomId = room.roomId,
                    toRoomId   = exit.linkedRoomId,
                    direction  = exit.direction.ToString(),
                    spawnPosX  = exit.worldPosition.x,
                    spawnPosY  = exit.worldPosition.y
                });
        }
    }

```

		Льїн І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        string json = JsonUtility.ToJson(graphData, prettyPrint:
true);
        string path = Application.persistentDataPath +
"/level_graph.json";
        System.IO.File.WriteAllText(path, json);
        Debug.Log($"[LevelGenerator] Layout saved to {path}");
    }

    private IEnumerator SetInitialRoomDeferred(string roomId)
    {
        yield return null;
        RoomManager.Instance.SetInitialRoom(roomId);
        Debug.Log($"[LevelGenerator] Deferred SetInitialRoom
'{roomId}'");
    }
}

    /// <summary>Published after layout generation and prefab
instantiation are complete.</summary>
    public struct LevelGeneratedEvent : IEvent
    {
        /// <summary>Total number of rooms in the generated
layout.</summary>
        public int roomCount;
        public LevelGeneratedEvent(int count) => roomCount = count;
    }
}

```

Додаток В

Лістинг класу CameraController.cs

```

using System.Collections.Generic;
using UnityEngine;
using FiatLux.Core.Events;
using FiatLux.Contracts.Events.World;

namespace FiatLux.Services.Camera
{
    /// <summary>
    /// Custom camera controller for Fiat Lux.
    /// Singleton MonoBehaviour living on the Main Camera in the
Bootstrap [Persistent] scene.
    ///
    /// Responsibilities:
    /// 1. Maintain a priority-sorted stack of <see
cref="ICameraBehaviour"/> implementations;
    /// the highest-priority behaviour drives the desired position
each frame.
    /// 2. Clamp the resulting position to the active room's camera
bounds

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    ///      received via <see cref="ActiveRoomChangedEvent"/>
(PublishLatest).
    ///      3. Apply trauma-based screen shake as a post-process offset.
    ///      4. Pixel-snap the final position to keep rendering crisp at 16
PPU.
    ///
    /// Usage - follow target:
    /// <code>
    ///      CameraController.Instance.SetTarget(player.transform,
playerRb);
    /// </code>
    ///
    /// Usage - screen shake:
    /// <code>
    ///      CameraController.Instance.AddTrauma(0.6f);
    /// </code>
    ///
    /// Usage - cutscene:
    /// <code>
    ///      var cs = CameraController.Instance.PushCutscene(targetPoint);
    ///      // ... on completion:
    ///      CameraController.Instance.PopCutscene(cs);
    /// </code>
    /// </summary>
[RequireComponent(typeof(UnityEngine.Camera))]
public class CameraController : MonoBehaviour
{
    public static CameraController Instance;
    public Transform Target => _target;

    // — Inspector: pixel perfect

```

```

[Header("Pixel Perfect")]
[SerializeField] private int    _referenceResolutionY = 360;
[SerializeField] private float  _pixelsPerUnit       = 64f;
[SerializeField] private int    _zoom                = 1;
[SerializeField] private bool   _pixelSnap           = true;

```

```

// — Inspector: follow

```

```

[Header("Follow")]
[SerializeField] private Vector2 _deadZoneHalfSize = new(1.0f,
1.0f);
[SerializeField] private Vector2 _targetOffset     = new(0f,
0.3f);
[SerializeField] private float   _followDamping   = 0.15f;
[SerializeField] private float   _lookAheadTime    = 0.12f;
[SerializeField] private float   _lookAheadSmoothing = 5f;

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

// — Inspector: shake

```
[Header("Shake")]  
[SerializeField] private float _shakeMaxMagnitude = 0.3f;  
[SerializeField] private float _shakeDecay        = 1.5f;  
[SerializeField] private float _shakeFrequency    = 25f;
```

// — Runtime state

```
private UnityEngine.Camera _cam;  
private Transform           _target;  
  
private Bounds _clampBounds;  
private bool    _hasBounds;  
  
private FollowBehaviour _followBehaviour;  
private CameraShake      _shake;  
private readonly List<ICameraBehaviour> _behaviours = new();
```

// — Lifecycle

```
private void Awake()  
{  
    if (Instance != null) { Destroy(gameObject); return; }  
    Instance = this;  
  
    _cam = GetComponent<UnityEngine.Camera>();  
    _cam.orthographicSize = _referenceResolutionY / (2f *  
_pixelsPerUnit * _zoom);  
    _shake = new CameraShake(_shakeMaxMagnitude, _shakeDecay,  
_shakeFrequency);  
  
    // FollowBehaviour is the default base behaviour - always  
    present in the stack.  
    _followBehaviour = new FollowBehaviour  
    {  
        DeadZoneHalfSize    = _deadZoneHalfSize,  
        TargetOffset        = _targetOffset,  
        FollowDamping        = _followDamping,  
        LookAheadTime        = _lookAheadTime,  
        LookAheadSmoothing   = _lookAheadSmoothing,  
    };  
    RegisterBehaviour(_followBehaviour);  
}  
  
private void OnEnable()  
{
```

		Льїн І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        // SubscribeWithLatest - receives the current room
        immediately if already set.
        EventBus.SubscribeWithLatest<ActiveRoomChangedEvent>(OnActive
RoomChanged);
    }

    private void OnDisable()
    {
        EventBus.Unsubscribe<ActiveRoomChangedEvent>(OnActiveRoomChan
ged);
    }

    private void OnDestroy()
    {
        if (Instance != this) return;
        Instance = null;
    }

    // — Public API

```

```

    /// <summary>
    /// Assigns a new follow target. Call from <c>GameManager</c> on
player spawn or respawn.
    /// Snaps the camera to the target immediately to avoid a catch-
up slide after teleport.
    /// </summary>
    /// <param name="target">Transform to follow. Pass <c>null</c> to
detach.</param>
    /// <param name="rb">Optional Rigidbody2D used for look-ahead
velocity sampling.</param>
    public void SetTarget(Transform target, Rigidbody2D rb = null)
    {
        _target = target;
        _followBehaviour.SetRigidbody(rb);
        _followBehaviour.OnActivated(transform.position);

        if (target != null)
            SnapToTarget();
    }

    /// <summary>
    /// Adds trauma to the shake system in the [0, 1] range. Values
accumulate up to 1.
    /// </summary>
    public void AddTrauma(float amount) => _shake.AddTrauma(amount);

    /// <summary>
    /// Registers an <see cref="ICameraBehaviour"/> and inserts it
into the

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    /// priority-sorted stack. The behaviour with the highest
<c>Priority</c>
    /// drives the camera position each frame.
    /// </summary>
    public void RegisterBehaviour(ICameraBehaviour behaviour)
    {
        if (_behaviours.Contains(behaviour)) return;
        behaviour.OnActivated(transform.position);
        _behaviours.Add(behaviour);
        SortBehaviours();
    }

    /// <summary>Removes a behaviour from the stack.</summary>
    public void UnregisterBehaviour(ICameraBehaviour behaviour)
    {
        _behaviours.Remove(behaviour);
    }

    /// <summary>
    /// Pushes a <see cref="CutsceneBehaviour"/> that moves the
camera to
    /// <paramref name="destination"/>. Returns the instance - pass
it back
    /// to <see cref="PopCutscene"/> when the cutscene ends.
    /// </summary>
    public CutsceneBehaviour PushCutscene(Vector2 destination, float
smoothTime = 0.4f)
    {
        var cs = new CutsceneBehaviour(destination, smoothTime);
        RegisterBehaviour(cs);
        return cs;
    }

    /// <summary>Removes a cutscene behaviour, returning control to
the follow stack.</summary>
    public void PopCutscene(CutsceneBehaviour cs)
    {
        UnregisterBehaviour(cs);
    }

    // — Event handlers

private void OnActiveRoomChanged(ActiveRoomChangedEvent evt)
{
    _clampBounds = evt.CameraBounds;
    _hasBounds = true;
    Debug.Log($"[CameraController] Bounds updated: room
'{evt.RoomId}'");
}

```

// — Update

```
private void LateUpdate()
{
    if (_target == null || _behaviours.Count == 0) return;

    Vector2 currentPos = transform.position;

    // Highest-priority behaviour drives the desired position
    (first after descending sort).
    Vector2 desired =
    _behaviours[0].CalculateTargetPosition(_target, currentPos,
    Time.deltaTime);

    // Clamp to active room bounds.
    if (_hasBounds)
        desired = ClampToBounds(desired);

    // Shake offset applied after clamping.
    if (_shake.IsActive)
        desired += _shake.GetOffset(Time.deltaTime);

    // Pixel snap (PPC will additionally align rendering on the
    GPU side).
    if (_pixelSnap)
        desired = SnapToPixel(desired);

    transform.position = new Vector3(desired.x, desired.y,
    transform.position.z);
}
```

// — Helpers

```
private void SnapToTarget()
{
    Vector2 pos = (Vector2)_target.position +
    _followBehaviour.TargetOffset;
    if (_hasBounds) pos = ClampToBounds(pos);
    if (_pixelSnap) pos = SnapToPixel(pos);
    transform.position = new Vector3(pos.x, pos.y,
    transform.position.z);
}

private Vector2 ClampToBounds(Vector2 pos)
{
    // orthographicSize is owned by the Pixel Perfect Camera -
    read it directly.
    float halfH = _cam.orthographicSize;
    float halfW = halfH * _cam.aspect;
```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		105

```

        float minX = _clampBounds.min.x + halfW;
        float maxX = _clampBounds.max.x - halfW;
        float minY = _clampBounds.min.y + halfH;
        float maxY = _clampBounds.max.y - halfH;

        // If the room is smaller than the viewport, centre rather
        than clamp.
        if (minX > maxX) { float cx = _clampBounds.center.x; minX =
        maxX = cx; }
        if (minY > maxY) { float cy = _clampBounds.center.y; minY =
        maxY = cy; }

        return new Vector2(
            Mathf.Clamp(pos.x, minX, maxX),
            Mathf.Clamp(pos.y, minY, maxY));
    }
    private Vector2 SnapToPixel(Vector2 pos)
    {
        float snap = 1f / _pixelsPerUnit;
        return new Vector2(
            Mathf.Round(pos.x / snap) * snap,
            Mathf.Round(pos.y / snap) * snap);
    }
    private void SortBehaviours()
    {
        // Descending by Priority - index 0 is always the active
        driver.
        _behaviours.Sort((a, b) => b.Priority.CompareTo(a.Priority));
    }
}

```

Лістинг класу ICameraBehaviour.cs

using UnityEngine;

namespace FiatLux.Services.Camera
{

/// <summary>
/// Contract for camera behaviour implementations (plain C#, not
MonoBehaviour).
/// <see cref="CameraController"/> drives the behaviour with the
highest <see cref="Priority"/>.

/// </summary>

public interface ICameraBehaviour
{

/// Higher value wins. Follow = 0, Cutscene = 10+.
int Priority { get; }

/// Returns the desired XY camera position in world space.

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				106
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        /// <paramref name="currentPosition"/> is needed for dead zone
        and damping.
        Vector2 CalculateTargetPosition(Transform target, Vector2
        currentPosition, float deltaTime);

        /// Called on activation – reset internal state (velocity, look-
        ahead, etc.) to avoid a positional jump.
        void OnActivated(Vector2 currentCameraPosition);
    }
}

```

Лістинг класу FollowBehaviour.cs

```

using UnityEngine;

namespace FiatLux.Services.Camera
{
    /// <summary>
    /// Default follow behaviour: world-space dead zone + look-ahead via
    Rigidbody2D velocity.
    /// Parameters are assigned by <see cref="CameraController"/> from
    its serialized fields.
    /// </summary>
    public sealed class FollowBehaviour : ICameraBehaviour
    {
        public int Priority => 0;

        // — Configuration (set by CameraController)

        /// Dead zone half-extents in world units. Camera stays still
        while the target
        /// remains inside the rectangle (currentCamPos ±
        DeadZoneHalfSize).
        public Vector2 DeadZoneHalfSize = new(1.0f, 1.0f);

        /// Offset applied to the target position (e.g. Y=0.3 shifts view
        slightly upward).
        public Vector2 TargetOffset = new(0f, 0.3f);

        /// SmoothDamp time. Lower = tighter follow.
        public float FollowDamping = 0.15f;

        /// velocity × LookAheadTime = forward offset in the direction of
        movement.
        public float LookAheadTime = 0.12f;

        /// Lerp speed for the look-ahead offset.
        public float LookAheadSmoothing = 5f;
    }
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

```

// — Internal state


---


private Rigidbody2D _rb;
private Vector2 _smoothVelocity;
private Vector2 _lookAheadOffset;

// — Public API


---



public void SetRigidbody(Rigidbody2D rb) => _rb = rb;

public void OnActivated(Vector2 currentCameraPosition)
{
    _smoothVelocity = Vector2.zero;
    _lookAheadOffset = Vector2.zero;
}

public Vector2 CalculateTargetPosition(Transform target, Vector2
currentPos, float deltaTime)
{
    Vector2 targetPos = (Vector2)target.position + TargetOffset;

    // — Look-ahead


---


    if (_rb != null)
    {
        Vector2 wantedLookAhead = _rb.linearVelocity *
LookAheadTime;
        _lookAheadOffset = Vector2.Lerp(
            _lookAheadOffset, wantedLookAhead,
            deltaTime * LookAheadSmoothing);
        targetPos += _lookAheadOffset;
    }

    // — Dead zone: camera moves only when target exits the zone


---


    float dx = targetPos.x - currentPos.x;
    float dy = targetPos.y - currentPos.y;

    Vector2 desired = currentPos;

    if (Mathf.Abs(dx) > DeadZoneHalfSize.x)
        desired.x = targetPos.x - Mathf.Sign(dx) *
DeadZoneHalfSize.x;

    if (Mathf.Abs(dy) > DeadZoneHalfSize.y)
        desired.y = targetPos.y - Mathf.Sign(dy) *
DeadZoneHalfSize.y;

```

```

// — SmoothDamp

return Vector2.SmoothDamp(
    currentPos, desired,
    ref _smoothVelocity,
    FollowDamping, float.MaxValue, deltaTime);
}
}
}

```

Додаток Г

Лістинг шейдера CRTEffect.shader

Shader "FiatLux/CRT"

```

{
    Properties
    {
        _MainTex          ("Source RT", 2D)          = "white" {}

        [Header(Scanlines)]
        _ScanlineIntensity ("Scanline Intensity",      Range(0, 1))          =
0.3
        _ScanlineCount     ("Scanline Count",          Range(100, 1080))     =
360

        [Header(RGB Mask)]
        _MaskIntensity     ("RGB Mask Intensity",      Range(0, 1))          = 0.5
        _MaskSize           ("RGB Mask Size (px)",     Range(1, 8))          = 3

        [Header(Bleed Glow)]
        _BleedIntensity    ("Bleed Intensity",        Range(0, 2))          =
0.8
        _BleedRadius       ("Bleed Radius",           Range(0, 0.01))       =
0.003
        _BleedThreshold    ("Bleed Threshold",        Range(0, 1))          =
0.4

        [Header(Chromatic Aberration)]
        _ChromaticOffset   ("Chromatic Offset",       Range(0, 0.01))       =
0.002

        [Header(Vignette)]
        _VignetteIntensity ("Vignette Intensity",     Range(0, 2))          = 0.8
        _VignetteSmoothness ("Vignette Smoothness",   Range(0.1, 1))        = 0.5

        [Header(Curvature)]
        _Curvature         ("Screen Curvature",       Range(0, 0.5))        = 0.1

        [Header(Color)]
        _Brightness        ("Brightness",             Range(0.5, 2))        = 1.2
    }
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				109
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        _Contrast          ("Contrast",          Range(0.5, 2)) =
1.05
    }

    SubShader
    {
        Tags { "RenderType"="Opaque" "Queue"="Overlay" }
        Cull Off
        ZWrite Off
        ZTest Always

        Pass
        {
            HLSLPROGRAM
            #pragma vertex   vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            struct appdata { float4 vertex : POSITION; float2 uv :
TEXCOORD0; };
            struct v2f      { float4 pos      : SV_POSITION; float2 uv :
TEXCOORD0; };

            sampler2D _MainTex;
            float4     _MainTex_TexelSize;

            float _ScanlineIntensity;
            float _ScanlineCount;
            float _MaskIntensity;
            float _MaskSize;
            float _BleedIntensity;
            float _BleedRadius;
            float _BleedThreshold;
            float _ChromaticOffset;
            float _VignetteIntensity;
            float _VignetteSmoothness;
            float _Curvature;
            float _Brightness;
            float _Contrast;

            v2f vert(appdata v)
            {
                v2f o;
                o.pos = UnityObjectToClipPos(v.vertex);
                o.uv  = v.uv;
                return o;
            }

            // Barrel distortion to simulate curved CRT screen.
            float2 ApplyCurvature(float2 uv)

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				110
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    float2 cc = uv - 0.5;
    float dist = dot(cc, cc) * _Curvature;
    return uv + cc * dist;
}

// Per-column RGB phosphor mask.
float3 RGBMask(float2 screenUV)
{
    float column = floor(screenUV.x * _ScreenParams.x /
_MaskSize);
    float idx      = fmod(column, 3.0);

    float3 mask = float3(1, 1, 1);
    if (idx < 1.0)      mask = float3(1.0, 0.5, 0.5); // R
    else if (idx < 2.0) mask = float3(0.5, 1.0, 0.5); // G
    else                mask = float3(0.5, 0.5, 1.0); // B

    return lerp(float3(1,1,1), mask, _MaskIntensity);
}

// 8-tap glow - only bright pixels bleed; diagonals weighted
half.
float3 SampleBleed(float2 uv)
{
    float3 bleed = float3(0, 0, 0);
    float r      = _BleedRadius;

    float2 offsets[8] = {
        float2( r, 0), float2(-r, 0), float2(0, r), float2(0,
- r),
        float2( r, r), float2(-r, r), float2( r,-r), float2(-
r,-r)
    };

    [unroll]
    for (int j = 0; j < 8; j++)
    {
        float3 s      = tex2D(_MainTex, uv + offsets[j]).rgb;
        float luma = dot(s, float3(0.299, 0.587, 0.114));
        s *= step(_BleedThreshold, luma);
        bleed += s * ((j < 4) ? 0.15 : 0.075);
    }

    return bleed * _BleedIntensity;
}

float4 frag(v2f i) : SV_Target
{
    // 1. Curvature
    float2 uv = ApplyCurvature(i.uv);

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				111
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (uv.x < 0.0 || uv.x > 1.0 || uv.y < 0.0 || uv.y > 1.0)
            return float4(0, 0, 0, 1);

        // 2. Chromatic aberration - R/B shifted outward from
centre
        float2 caOffset = (uv - 0.5) * _ChromaticOffset;
        float3 col = float3(
            tex2D(_MainTex, uv + caOffset).r,
            tex2D(_MainTex, uv                ).g,
            tex2D(_MainTex, uv - caOffset).b);

        // 3. Bleed glow
        col += SampleBleed(uv);

        // 4. Scanlines
        float scan = sin(uv.y * _ScanlineCount * 3.14159) * 0.5 +
0.5;

        col *= lerp(1.0, scan, _ScanlineIntensity);

        // 5. RGB phosphor mask
        col *= RGBMask(uv);

        // 6. Vignette
        float2 vc = uv - 0.5;
        float vig = smoothstep(0.0, _VignetteSmoothness,
                                1.0 - dot(vc, vc) * _VignetteIntensity);
        col *= vig;

        // 7. Brightness / contrast
        col = (col - 0.5) * _Contrast + 0.5;
        col *= _Brightness;

        return float4(col, 1.0);
    }
    ENDHLSL
}

}

FallBack Off
}

```

Лістинг класу CRTRenderer.cs

```
using UnityEngine;
```

```
namespace FiatLux.Services.Camera
{
```

```
    /// <summary>
    /// Renders the game into a low-resolution <see
    cref="RenderTexture"/> and displays it

```

		Льїн І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				112
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    /// through a fullscreen quad with the <c>FiatLux/CRT</c> post-effect
    material.
    ///
    /// Architecture: the game camera renders to an internal 640x360 RT
    (point-filtered);
    /// a dedicated display camera on a separate layer composites the CRT
    quad on top.
    /// This keeps the HUD canvas bound to a static overlay camera, not
    the moving game camera.
    /// </summary>
    [RequireComponent(typeof(UnityEngine.Camera))]
    public sealed class CRTRenderer : MonoBehaviour
    {
        public static CRTRenderer Instance { get; private set; }

        [Header("Internal Resolution")]
        [SerializeField] private int _refResX = 640;
        [SerializeField] private int _refResY = 360;

        [Header("Toggles")]
        [SerializeField] private bool _crtEnabled = true;
        [SerializeField] private bool _bleedEnabled = true;

        [Header("CRT Effect")]
        [SerializeField] private Shader _crtShader;

        [Range(0f, 1f)]
        [SerializeField] private float _scanlineIntensity = 0.3f;
        [SerializeField] private float _scanlineCount = 360f;

        [Range(0f, 0.01f)]
        [SerializeField] private float _chromaticOffset = 0.002f;

        [Range(0f, 2f)]
        [SerializeField] private float _vignetteIntensity = 0.8f;
        [Range(0.1f, 1f)]
        [SerializeField] private float _vignetteSmoothness = 0.5f;

        [Range(0f, 0.5f)]
        [SerializeField] private float _curvature = 0.1f;

        [Range(0.5f, 2f)]
        [SerializeField] private float _brightness = 1.1f;
        [Range(0.5f, 2f)]
        [SerializeField] private float _contrast = 1.0f;

        [Header("RGB Mask")]
        [Range(0f, 1f)] [SerializeField] private float _maskIntensity =
0.5f;
        [Range(1f, 8f)] [SerializeField] private float _maskSize =
3f;

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				113
Змн.	Арк.	№ докум.	Підпис	Дата		

```

[Header("Bleed Glow")]
[Range(0f, 2f)] [SerializeField] private float _bleedIntensity
= 0.8f;
[Range(0f, 0.01f)] [SerializeField] private float _bleedRadius
= 0.003f;
[Range(0f, 1f)] [SerializeField] private float _bleedThreshold
= 0.4f;

// — Runtime

```

```

private const int DisplayLayer = 31;

private UnityEngine.Camera _gameCamera;
private UnityEngine.Camera _displayCamera;
private RenderTexture _rt;
private Material _crtMaterial;
private GameObject _quad;

```

```

// — Lifecycle

```

```

private void Awake()
{
    if (Instance != null && Instance != this) {
Destroy(gameObject); return; }
    Instance = this;

    _gameCamera = GetComponent<UnityEngine.Camera>();

    LoadSettings();
    CreateRenderTexture();
    CreateMaterial();
    CreateDisplayCamera();
    CreateQuad();
}

private void OnDestroy()
{
    if (Instance == this) Instance = null;

    if (_rt != null) { _rt.Release(); Destroy(_rt); }
    if (_crtMaterial != null) Destroy(_crtMaterial);
    if (_quad != null) Destroy(_quad);
    if (_displayCamera != null)
Destroy(_displayCamera.gameObject);
}

private void Update()
{

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				114
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        UpdateShaderProperties();
        UpdateQuadAspect();
    }

    // — Setup

private void CreateRenderTexture()
{
    _rt = new RenderTexture(_refResX, _refResY, 24,
RenderTextureFormat.ARGB32)
    {
        filterMode    = FilterMode.Point,
        wrapMode      = TextureWrapMode.Clamp,
        antiAliasing  = 1,
        name          = "CRTInternalRT",
    };
    _rt.Create();
    _gameCamera.targetTexture = _rt;
    _gameCamera.cullingMask  &= ~(1 << DisplayLayer);
}

private void CreateMaterial()
{
    if (_crtShader == null)
        _crtShader = Shader.Find("FiatLux/CRT");

    if (_crtShader == null)
    {
        Debug.LogError("[CRTRenderer] CRT shader not found");
        return;
    }

    _crtMaterial = new Material(_crtShader) { mainTexture = _rt
};
}

private void CreateDisplayCamera()
{
    var go = new GameObject("CRTDisplayCamera");
    go.transform.SetParent(transform, false);

    _displayCamera = go.AddComponent<UnityEngine.Camera>();
    _displayCamera.orthographic      = true;
    _displayCamera.orthographicSize = 0.5f;
    _displayCamera.nearClipPlane    = 0.01f;
    _displayCamera.farClipPlane     = 10f;
    _displayCamera.clearFlags       =
CameraClearFlags.SolidColor;
    _displayCamera.backgroundColor  = Color.black;
    _displayCamera.cullingMask     = 1 << DisplayLayer;
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				115
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        _displayCamera.depth                = _gameCamera.depth + 1; //
composited above game camera
        _displayCamera.allowMSAA           = false;
        _displayCamera.allowHDR            = false;
    }

    private void CreateQuad()
    {
        _quad = GameObject.CreatePrimitive(PrimitiveType.Quad);
        _quad.name = "CRTDisplayQuad";
        Destroy(_quad.GetComponent<Collider>());

        _quad.transform.SetParent(_displayCamera.transform, false);
        _quad.transform.localPosition = new Vector3(0f, 0f, 1f);
        _quad.layer = DisplayLayer;

        _quad.GetComponent<MeshRenderer>().sharedMaterial =
        _crtMaterial;
        UpdateQuadAspect();
    }

    private void UpdateQuadAspect()
    {
        if (_quad == null || _displayCamera == null) return;
        float aspect = (float)Screen.width / Screen.height;
        _quad.transform.localScale = new Vector3(aspect, 1f, 1f);
    }

    // — Shader update

```

```

    private void UpdateShaderProperties()
    {
        if (_crtMaterial == null) return;

        float fx = _crtEnabled ? 1f : 0f;

        _crtMaterial.SetFloat("_ScanlineIntensity",
        _scanlineIntensity * fx);
        _crtMaterial.SetFloat("_MaskIntensity",      _maskIntensity
        * fx);
        _crtMaterial.SetFloat("_ChromaticOffset",    _chromaticOffset
        * fx);
        _crtMaterial.SetFloat("_VignetteIntensity",
        _vignetteIntensity * fx);
        _crtMaterial.SetFloat("_Curvature",          _curvature
        * fx);
        _crtMaterial.SetFloat("_BleedIntensity",     _crtEnabled &&
        _bleedEnabled ? _bleedIntensity : 0f);

        _crtMaterial.SetFloat("_ScanlineCount",      _scanlineCount);

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				116
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        _crtMaterial.SetFloat("_MaskSize",          _maskSize);
        _crtMaterial.SetFloat("_BleedRadius",       _bleedRadius);
        _crtMaterial.SetFloat("_BleedThreshold",    _bleedThreshold);
        _crtMaterial.SetFloat("_VignetteSmoothness", _vignetteSmoothness);
        _crtMaterial.SetFloat("_Brightness",        _brightness);
        _crtMaterial.SetFloat("_Contrast",          _contrast);
    }

    // — Public API

```

```

    public void SetScanlineIntensity(float v) => _scanlineIntensity =
Mathf.Clamp01(v);
    public void SetChromaticOffset(float v)  => _chromaticOffset   =
Mathf.Clamp(v, 0f, 0.01f);
    public void SetVignetteIntensity(float v) => _vignetteIntensity =
Mathf.Clamp(v, 0f, 2f);
    public void SetCurvature(float v)        => _curvature        =
Mathf.Clamp(v, 0f, 0.5f);

```

```

    public float ScanlineIntensity => _scanlineIntensity;
    public float ChromaticOffset   => _chromaticOffset;
    public float VignetteIntensity => _vignetteIntensity;
    public float Curvature         => _curvature;

```

```

    public bool CRTEnabled
    {
        get => _crtEnabled;
        set { _crtEnabled = value; UpdateShaderProperties(); }
    }

```

```

    public bool BleedEnabled
    {
        get => _bleedEnabled;
        set { _bleedEnabled = value; UpdateShaderProperties(); }
    }

```

```

    // — Persistence

```

```

[System.Serializable]
private class CRTData
{

```

```

    public bool  crtEnabled    = true;
    public bool  bleedEnabled  = true;
    public float scanline     = 0.3f;
    public float chromatic    = 0.002f;
    public float vignette     = 0.8f;
    public float curvature    = 0.1f;

```

		Львів І.В.			ІІІ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				117
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    private string CRTSavePath =>
        System.IO.Path.Combine(Application.persistentDataPath,
            "video_settings.json");

    public void SaveSettings()
    {
        var data = new CRTData
        {
            crtEnabled    = _crtEnabled,    bleedEnabled =
            _bleedEnabled,
            scanline      = _scanlineIntensity, chromatic =
            _chromaticOffset,
            vignette      = _vignetteIntensity, curvature =
            _curvature,
        };
        try { System.IO.File.WriteAllText(CRTSavePath,
            JsonUtility.ToJson(data, true)); }
        catch (System.Exception ex) {
            Debug.LogWarning($"[CRTRenderer] Save failed: {ex.Message}"); }
    }

    private void LoadSettings()
    {
        if (!System.IO.File.Exists(CRTSavePath)) return;
        try
        {
            var data =
            JsonUtility.FromJson<CRTData>(System.IO.File.ReadAllText(CRTSavePath));
            if (data == null) return;
            _crtEnabled    = data.crtEnabled;    _bleedEnabled
            = data.bleedEnabled;
            _scanlineIntensity = data.scanline;    _chromaticOffset
            = data.chromatic;
            _vignetteIntensity = data.vignette;    _curvature
            = data.curvature;
        }
        catch (System.Exception ex) {
            Debug.LogWarning($"[CRTRenderer] Load failed: {ex.Message}"); }
    }
}

```

Додаток Д

Лістинг класу SaveManager.cs

```

using System;
using System.IO;
using UnityEngine;
using FiatLux.Core;
using FiatLux.Core.Events;

```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				118
Змн.	Арк.	№ докум.	Підпис	Дата		

```

using FiatLux.Contracts.Data;
using FiatLux.Contracts.Events.Save;

namespace FiatLux.Services.SaveLoad
{
    /// <summary>
    /// Handles JSON save slot I/O and coordinates the save/load
    lifecycle via EventBus.
    ///
    /// Save flow: <see cref="Save"/> publishes <see
    cref="CollectingSaveDataEvent"/> so that
    /// services self-serialize into <see cref="SaveData"/> before it is
    written to disk -
    /// no direct coupling between SaveManager and domain services.
    ///
    /// Load flow: <see cref="OnLoadRequested"/> reads the file and
    publishes
    /// <see cref="GameLoadedEvent"/>; subscribers apply state via <see
    cref="ApplyingSaveDataEvent"/>.
    /// </summary>
    public class SaveManager : MonoBehaviour
    {
        public static SaveManager Instance;

        public const int SlotCount = 3;

        // — Lifecycle

        private void Awake()
        {
            if (Instance != null) { Destroy(gameObject); return; }
            Instance = this;

            EventBus.Subscribe<SaveRequestedEvent>(OnSaveRequested);
            EventBus.Subscribe<LoadRequestedEvent>(OnLoadRequested);
        }

        private void OnDestroy()
        {
            EventBus.Unsubscribe<SaveRequestedEvent>(OnSaveRequested);
            EventBus.Unsubscribe<LoadRequestedEvent>(OnLoadRequested);

            if (Instance == this) Instance = null;
        }

        // — Public API

        public bool HasSave(int slot) => File.Exists(SlotPath(slot));
    }

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				119
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    /// <summary>Returns metadata for the slot without fully
deserializing it.</summary>
    public SaveSlotInfo GetSlotInfo(int slot)
    {
        var path = SlotPath(slot);
        if (!File.Exists(path)) return SaveSlotInfo.Empty;
        try
        {
            var data =
JsonUtility.FromJson<SaveData>(File.ReadAllText(path));
            return data == null
                ? SaveSlotInfo.Empty
                : new SaveSlotInfo(true, data.timestampUtc,
data.sceneName);
        }
        catch (Exception ex)
        {
            Debug.LogWarning($"[SaveManager] GetSlotInfo slot {slot}
failed: {ex.Message}");
            return SaveSlotInfo.Empty;
        }
    }

    /// <summary>Reads and deserializes a slot. No events fired - raw
data only.</summary>
    public SaveData Load(int slot)
    {
        var path = SlotPath(slot);
        if (!File.Exists(path)) return null;
        try
        {
            return
JsonUtility.FromJson<SaveData>(File.ReadAllText(path));
        }
        catch (Exception ex)
        {
            Debug.LogWarning($"[SaveManager] Load slot {slot} failed:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Publishes <see cref="CollectingSaveDataEvent"/> so services
populate
    /// <paramref name="data"/>, then serializes to disk.
    /// </summary>
    public bool Save(int slot, SaveData data)
    {
        if (data == null)
        {

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				120
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Debug.LogWarning("[SaveManager] Save: data is null");
        return false;
    }

    try
    {
        EventBus.Publish(new CollectingSaveDataEvent(data));
        data.timestampUtc = DateTime.UtcNow.Ticks;
        File.WriteAllText(SlotPath(slot),
JsonUtility.ToJson(data, prettyPrint: true));
        return true;
    }
    catch (Exception ex)
    {
        Debug.LogWarning($"[SaveManager] Save slot {slot} failed:
{ex.Message}");
        return false;
    }
}

public bool DeleteSlot(int slot)
{
    var path = SlotPath(slot);
    if (!File.Exists(path)) return false;
    try { File.Delete(path); return true; }
    catch (Exception ex)
    {
        Debug.LogWarning($"[SaveManager] Delete slot {slot}
failed: {ex.Message}");
        return false;
    }
}

public int DeleteAllSlots()
{
    int deleted = 0;
    for (int i = 0; i < SlotCount; i++)
        if (DeleteSlot(i)) deleted++;

    EventBus.Publish(new SavesDeletedEvent());
    return deleted;
}

// — EventBus handlers

private void OnSaveRequested(SaveRequestedEvent e)
{
    // Intentionally empty - save is driven by GameManager, not
    by this event directly.
}

```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		121

```

        private void OnLoadRequested(LoadRequestedEvent e)
        {
            var data = Load(e.Slot);
            EventBus.Publish(new GameLoadedEvent(e.Slot, data != null,
data));
        }

// — Helpers

```

```

        private string SlotPath(int slot) =>
            Path.Combine(Application.persistentDataPath,
$"save_slot_{slot}.json");
    }
}

```

Додаток Е

Лістинг класу UIScreenStack.cs

```

using System.Collections.Generic;

using UnityEngine;
using FiatLux.Core.Events;
using FiatLux.Contracts.Events.UI;

namespace FiatLux.UI
{
    /// <summary>
    /// Stack-based UI screen navigator. Push opens a screen and
hides the previous one;
    /// Pop restores it. All transitions are broadcast via
EventBus.
    /// </summary>
    public class UIScreenStack : MonoBehaviour
    {
        public static UIScreenStack Instance;

        private readonly Stack<UIScreen> _stack = new
Stack<UIScreen>();

        public UIScreen Top    => _stack.Count > 0 ? _stack.Peek() :
null;
        public int      Count => _stack.Count;

        private void Awake()
        {
            if (Instance != null) { Destroy(gameObject); return; }
            Instance = this;
        }
    }
}

```

		Львів І.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				122
Змн.	Арк.	№ докум.	Підпис	Дата		

```
public void Push(UIScreen screen)
{
    if (screen == null) return;
    if (_stack.Count > 0) _stack.Peek().OnHide();

    _stack.Push(screen);
    screen.OnPush();

    EventBus.Publish(new ScreenPushedEvent(screen.name));
}

public void Pop()
{
    if (_stack.Count == 0) return;

    var top = _stack.Pop();
    top.OnPop();
    EventBus.Publish(new ScreenPoppedEvent(top.name));

    if (_stack.Count > 0) _stack.Peek().OnReveal();
}

/// Pops all screens.
public void Clear()
{
    while (_stack.Count > 0)
    {
        var s = _stack.Pop();
        s.OnPop();
        EventBus.Publish(new ScreenPoppedEvent(s.name));
    }
}

/// Pops the top screen if it has <see
cref="UIScreen.CloseOnCancel"/> set.
/// Returns <c>true</c> if the cancel was consumed.
public bool HandleCancel()
{
    if (_stack.Count == 0) return false;
    var top = _stack.Peek();
    if (!top.CloseOnCancel) return false;

    Pop();
    return true;
}
}
```

ЛІСТИНГ класу UIScreen.cs

```
using UnityEngine;
using UnityEngine.EventSystems;

namespace FiatLux.UI
{
    /// <summary>
    /// Base class for all UI screens managed by <see
    cref="UIScreenStack"/>.
    /// Override lifecycle methods to hook into push/pop/hide/reveal
    transitions.
    /// </summary>
    public class UIScreen : MonoBehaviour
    {
        [Tooltip("GameObject to focus in EventSystem when the screen
opens (gamepad/keyboard navigation).")]
        [SerializeField] private GameObject firstSelected;

        [Tooltip("If true, the screen closes on UI/Cancel (Escape /
gamepad Back).")]
        [SerializeField] private bool closeOnCancel = true;

        public bool CloseOnCancel => closeOnCancel;

        // — Lifecycle hooks

        public virtual void OnPush()    { gameObject.SetActive(true);
SetFocus(); }
        public virtual void OnPop()     { gameObject.SetActive(false); }
        public virtual void OnHide()    { gameObject.SetActive(false); }
        public virtual void OnReveal()  { gameObject.SetActive(true);
SetFocus(); }

        // — Helpers

        private void SetFocus()
        {
            if (firstSelected == null || EventSystem.current == null)
                return;
            EventSystem.current.SetSelectedGameObject(null);
            EventSystem.current.SetSelectedGameObject(firstSelected);
        }
    }
}
```

		Львів І.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Лобанчикова Н.М.				124
Змн.	Арк.	№ докум.	Підпис	Дата		