

НАЗАРЕНКО В. А.

РОЗРОБКА ІГРОВИХ ДОДАТКІВ

Навчальний посібник

для здобувачів першого (бакалаврського) рівня вищої освіти
галузі знань F «Інформаційні технології»
спеціальностей F2 «Інженерія програмного забезпечення»
та F3 «Комп’ютерні науки»

Київ

2026

УДК 004.42:794(075.8)
Н 19

Рекомендовано до видання рішенням вченої ради
Національного університету біоресурсів і природокористування України
(Протокол № 4 від 22 Жовтня 2026 року)

Рецензенти:

Касаткін Д. Ю., кандидат педагогічних наук, доцент, завідувач кафедри комп'ютерних систем, мереж та кібербезпеки, НУБіП України.

Афтанділянц Є. Г., доктор технічних наук, професор, Національна академія наук України, провідний науковий співробітник відділу спеціальних сталей і сплавів Фізико-технологічного інституту металів і сплавів НАН України.

Кузнєцов Є. В., доктор філософії, науковий співробітник (ML\CV), Katholieke Universiteit Leuven, Leuven, Belgium.

Н 19 Розробка ігрових додатків : навчальний посібник / В. А. Назаренко. Київ : НУБіП України, 2026. 318 с.

ISBN 978-617-8962-37-1

Під час підготовки навчального видання використано інструменти штучного інтелекту для технічного структурування, мовного редагування та форматування авторських матеріалів (ChatGPT (зокрема Codex), Gemini, Grammarly). Зміст, методичні рішення, добір прикладів і відповідальність за видання належать автору.

Зміст навчального посібника відповідає програмі навчальної дисципліни «Розробка ігрових додатків». Посібник призначено для студентів, викладачів, розробників-початківців і всіх, хто опановує геймдизайн, програмні системи ігрових рушіїв, інтерактивний та імерсивний дизайн.

УДК 004.42:794(075.8)

© Назаренко В. А., 2026
© НУБіП України, 2026

ВІДОМОСТІ ПРО АВТОРА



Назаренко Володимир Анатолійович
Доктор філософії, доцент кафедри комп'ютерних систем, мереж та кібербезпеки Національного університету біоресурсів і природокористування України. Викладає дисципліни «Дизайн інтерактивних систем», «Інтерактивна комп'ютерна графіка», «Теорія розпізнавання образів», «Основи програмування», «Паралельні та розподілені обчислення», «Візуальний дизайн (відеоігри та графіка)», «Розробка імерсивних додатків», «Розробка ігрових додатків». Наукові інтереси пов'язані з HCI, UX/UI та Product Design, ігровими й імерсивними системами, IoT, UAV, розумними містами, зеленими технологіями, AI, ML, CV, DS і NLP. Автор понад 30 наукових праць, зокрема монографії та статей, індексованих у Scopus.
Електронна адреса: volodnz@nubip.edu.ua.
https://linkedin.com/in/volodymyrnazukr

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ **6**
ПЕРЕДМОВА **7**
PREFACE **9**
РОЗДІЛ 1. КОНЦЕПЦІЯ, ІГРОВІ МЕХАНІКИ ТА ВИРОБНИЧИЙ ПРОЦЕС **10**
Тема 1.1. Вступ до розробки ігрових додатків: індустрія, ролі, pipeline, артефакти 10
Тема 1.2. Дизайн-мислення та дослідження гравця 16
Тема 1.3. Ігрові механіки, MDA, core loop та балансування 24
Тема 1.4. Документування концепції та порівняльний аналіз ігор 9
Тема 1.5. Наративний дизайн та інтерактивна драматургія 15
Питання для самоперевірки до розділу 1 20
Практичні завдання до розділу 1 21
Висновки до розділу 1 21
Графічні матеріали до розділу 1 21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ІГРОВОГО СВІТУ ТА ВІЗУАЛЬНА СИСТЕМА **24**
Тема 2.1. Дизайн рівнів і навігація: blockout, темп, прогресія та метрики 24
Тема 2.2. Візуальний дизайн ігрових застосунків: арт-пайплайн, UI/HUD, VFX та анімація 33
Тема 2.3. Game UI/UX foundations: HUD, diegetic UI, меню та доступність 42
Тема 2.4. Materials & Shaders I: PBR та стилізація 53
Тема 2.5. Game FX Design: particles, timing та читабельність 64
Тема 2.6. Lighting, Post-FX & Mood: readability, exposure, fog та color grading 73
Тема 2.7. Materials & Shaders II: full-screen FX, distortion, outlines, SDF та stylized motion 81
Тема 2.8. Production-ready пайплайн 2D/3D асетів, ригінгу та анімації 90
Питання для самоперевірки до розділу 2 95
Практичні завдання до розділу 2 95
Висновки до розділу 2 95
Графічні матеріали до розділу 2 96
РОЗДІЛ 3. ВЗАЄМОДІЯ, GAME FEEL ТА МУЛЬТИМОДАЛЬНИЙ **99**
Тема 3.1. Game feel та Input Systems: responsiveness, feedback loops, MDA 99
Тема 3.2. Character Controller & Camera: movement, game feel та virtual cinematography 114
Тема 3.3. Архітектура систем взаємодії: input - state - feedback 123
Тема 3.4. Audio for Interaction: SFX layers, spatialization та parameter-driven audio 131
Тема 3.5. Animation Systems & State: blend trees, root motion, notifies та interrupts 139
Тема 3.6. Onboarding, difficulty та UX telemetry 147
Тема 3.7. Імерсивний дизайн, XR та тактильний зворотний зв’язок 155
Питання для самоперевірки до розділу 3 159
Практичні завдання до розділу 3 160
Висновки до розділу 3 160
Графічні матеріали до розділу 3 160
РОЗДІЛ 4. ПРОГРАМНІ СИСТЕМИ, РУШІЇ ТА ПРОДУКТИВНІСТЬ **163**
Тема 4.1. Ігрові програмні системи: архітектура, патерни, ECS та інструменти 163

Тема 4.2. Розробка в Unity та Unreal Engine: прототипування, ітерації й інтеграція контенту ... 172
Тема 4.3. Штучний інтелект і data-driven gameplay systems ... 179
Тема 4.4. Контроль версій, налагодження та build pipeline ... 184
Тема 4.5. Performance & Optimization: profiling, budgets, LOD, instancing та latency ... 189
Тема 4.6. Мережеві ігрові системи: авторитет, реплікація та latency compensation ... 197
Тема 4.7. Процедурна генерація контенту та інструменти для команди ... 201
Питання для самоперевірки до розділу 4 ... 205
Практичні завдання до розділу 4 ... 206
Висновки до розділу 4 ... 206
Графічні матеріали до розділу 4 ... 206
РОЗДІЛ 5. ТЕСТУВАННЯ, ВИРОБНИЦТВО, РЕЛІЗ І ПОРТФОЛІО ... 209
Тема 5.1. Тестування, аналітика та підготовка до релізу: QA, user research, telemetry, build і publishing ... 209
Тема 5.2. Командний проєкт: механіка, ігровий цикл і прототип рівня ... 218
Тема 5.3. Мудборд, storyboard і візуальна комунікація ігрового бачення ... 224
Тема 5.4. Фінальний vertical slice, трейлер і pitch deck ... 228
Тема 5.5. Гейміфікація навчального процесу та мотивація команди ... 233
Тема 5.6. Академічна доброчесність, ліцензування, портфоліо і кар'єрні траєкторії ... 236
Тема 5.7. Бізнес-моделі, publishing, локалізація та Live Ops ... 240
Тема 5.8. Серйозні ігри, симуляції та оцінювання навчального результату ... 245
Питання для самоперевірки до розділу 5 ... 248
Практичні завдання до розділу 5 ... 249
Висновки до розділу 5 ... 249
Графічні матеріали до розділу 5 ... 250
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ... 252
ДОДАТОК А. ВХІДНА ДІАГНОСТИКА ТА ІНДИВІДУАЛЬНА ТРАЄКТОРІЯ ... 256
ДОДАТОК Б. ШАБЛОН MINI-GDD ТА КОНКУРЕНТНОГО АНАЛІЗУ ... 257
ДОДАТОК В. ШАБЛОН ДОКУМЕНТА ДИЗАЙНУ РІВНЯ ... 260
ДОДАТОК Г. ШАБЛОНИ INTERACTION, VFX, AUDIO ТА ANIMATION BREAKDOWN ... 262
ДОДАТОК Д. ШАБЛОН ЗВІТУ З ПРОФІЛЮВАННЯ ТА ОПТИМІЗАЦІЇ ... 264
ДОДАТОК Е. ФІНАЛЬНИЙ VERTICAL SLICE, ТРЕЙЛЕР І PITCH DECK ... 265
ДОДАТОК Ж. ШАБЛОН ПОСТМОРТЕМУ ТА САМООЦІНЮВАННЯ ... 267
ДОДАТОК З. КРИТЕРІЇ ОЦІНЮВАННЯ ТЕМАТИЧНИХ І ПРОЄКТНИХ РОБІТ ... 268
ДОДАТОК И. ОПЕРАЦІЙНІ ЧЕК-ЛИСТИ ДЛЯ ЩОДЕННОЇ РОБОТИ ... 273
ГЛОСАРІЙ ... 277

СПИСОК УМОВНИХ СКОРОЧЕНЬ

Таблиця 1

Основні скорочення, використані у навчальному посібнику

СКОРОЧЕННЯ	ЗНАЧЕННЯ
AI	Artificial Intelligence - штучний інтелект
API	Application Programming Interface - програмний інтерфейс застосунку
AR	Augmented Reality - доповнена реальність
CPU	Central Processing Unit - центральний процесор
DCC	Digital Content Creation - програмні засоби створення цифрового контенту
ECS	Entity Component System - сутнісно-компонентна архітектура
FPS	Frames Per Second - кількість кадрів за секунду
FTUE	First-Time User Experience - перший користувацький досвід
GDD	Game Design Document - документ ігрового дизайну
GPU	Graphics Processing Unit - графічний процесор
HUD	Head-Up Display - ігровий інтерфейс поверх сцени
HCI	Human-Computer Interaction - взаємодія людини з комп’ютером
LDD	Level Design Document - документ дизайну рівня
LOD	Level of Detail - рівень деталізації
MDA	Mechanics-Dynamics-Aesthetics - механіки, динаміки, естетика/досвід
PBR	Physically Based Rendering - фізично обґрунтований рендеринг
QA	Quality Assurance - забезпечення якості
RNG	Random Number Generator - генератор випадкових чисел
RTPC	Real-Time Parameter Control - керування аудіопараметрами у реальному часі
SFX	Sound Effects - звукові ефекти
UI	User Interface - користувацький інтерфейс
UX	User Experience - користувацький досвід
VCS	Version Control System - система контролю версій
VFX	Visual Effects - візуальні ефекти
VR	Virtual Reality - віртуальна реальність
XR	Extended Reality - узагальнена назва імерсивних технологій

ПЕРЕДМОВА

Сучасна розробка ігрових додатків є міждисциплінарною галуззю, у якій поєднуються програмна інженерія, геймдизайн, комп'ютерна графіка, анімація, звук, UX/UI, психологія взаємодії та виробничий менеджмент. Ігровий продукт не можна звести лише до програмного коду або набору візуальних асетів. Він формується як цілісна інтерактивна система, у якій правила, дії гравця, простір, аудіовізуальна подача та зворотний зв'язок працюють узгоджено.

Навчальний посібник створено на основі трьох частин авторських методичних рекомендацій, лекційних матеріалів, а також практичних і лабораторних завдань з дисципліни «Розробка ігрових додатків». Матеріал перероблено в єдину структуру, доповнено поясненнями, таблицями, шаблонами, критеріями оцінювання, рекомендаціями щодо графічних матеріалів та прикладами виробничих артефактів.

Мета навчального посібника - сформувати у студентів системне розуміння повного циклу створення гри: від дослідження аудиторії, формулювання ідеї та визначення core loop до побудови керування, взаємодій, рівнів, UI/HUD, візуальних і звукових систем, архітектури проєкту, профілювання, тестування, підготовки vertical slice, релізу та портфоліо.

Методичний підхід ґрунтується на принципі «від гіпотези до перевіреного прототипу». Кожна тема поєднує теоретичний матеріал із практичним результатом: схемою, таблицею параметрів, коротким документом, тестовою сценою, робочою механікою або демонстраційним build. Такий формат дозволяє використовувати посібник як під час аудиторних занять, так і для самостійної роботи.

Особлива увага приділяється вимірюваній якості. Замість абстрактного формулювання «гра працює добре» студент повинен фіксувати критерії: час до першого фідбеку, FPS і frame time, кількість помилок, зрозумілість підказок, відсоток успішного виконання завдання, стабільність build, відсутність критичних дефектів, відповідність візуальному та технічному бюджету.

Посібник орієнтований на використання Unity та Unreal Engine 5, але більшість принципів є рушій-незалежними. Вони стосуються постановки задачі, системного мислення, модульної архітектури, прототипування або створення прототипів, плейтесту, читабельності, доступності, керування ризиками та командної комунікації.

Навчальний матеріал посібника містить п'ять розділів:

Розділ 1 розкриває поняття ігрового застосунку, індустрії та платформ, дизайн-мислення, дослідження гравця, ігрові механіки, MDA, core loop, документацію та виробничий pipeline.

Розділ 2 присвячено дизайну рівнів, арт-пайплайну, UI/HUD, матеріалам і шейдерам, VFX, освітленню та формуванню цілісної візуальної мови гри.

Розділ 3 аналізує game feel, input systems, контролер персонажа і камеру, архітектуру взаємодій, анімацію, аудіо, онбординг, складність, телеметрію та імерсивний дизайн.

Розділ 4 висвітлює архітектуру ігрових програмних систем, патерни, ECS, роботу в Unity/Unreal, AI, контроль версій, збірки, профілювання й оптимізацію.

Розділ 5 охоплює QA, плейтест, аналітику, командний production, vertical slice, фінальну презентацію, реліз, гейміфікацію навчання, академічну доброчесність і кар'єрні траєкторії.

Кожен розділ навчального посібника містить структурований навчальний матеріал за темами, таблиці, графічні матеріали з рекомендаціями щодо їх підготовки, питання для самоперевірки, практичні завдання та висновки. У додатках подано шаблони документів, чек-листи й глосарій, які можна безпосередньо застосовувати у навчальному або командному проєкті.

Усі розділи (1–5), додатки та глосарій підготовлено В. А. Назаренком.

Як працювати з навчальним посібником

Кожен розділ посібника починається з мети, очікуваних результатів і ключових понять, а завершується питаннями для самоперевірки, практичними завданнями та висновками.

Сірі графічні блоки є місцями для авторських схем, анотованих скріншотів або діаграм. Під кожним блоком наведено назву рисунка та коротку рекомендацію щодо його підготовки.

Для командної та індивідуальної роботи доцільно використовувати шаблони з додатків і фіксувати кожну ітерацію у форматі «гіпотеза - зміна - вимірюваний результат». (табл. 1)

PREFACE

Modern game application development is an interdisciplinary field that combines software engineering, game design, computer graphics, animation, audio, UX/UI, interaction psychology, and production management. A game is not merely a codebase or a collection of visual assets; it is an integrated interactive system in which rules, player actions, space, audiovisual presentation, and feedback operate coherently.

This textbook consolidates three parts of the author's methodological recommendations together with lecture, practical, and laboratory materials for the course "Game Application Development". The content has been reorganized into five chapters and expanded with explanations, tables, templates, assessment criteria, figure recommendations, and production-oriented artifacts.

The main objective is to build a systematic understanding of the complete game-development cycle: audience research, concept definition, mechanics and core loop, interaction and game feel, level and visual design, software architecture, profiling, testing, vertical-slice production, release preparation, and portfolio presentation.

The methodological approach follows a hypothesis-to-prototype logic. Each topic results in an observable artifact: a diagram, parameter table, concise design document, test scene, playable mechanic, or demonstrable build. The material can therefore support both classroom instruction and independent project work.

Unity and Unreal Engine 5 are used as principal implementation contexts, while the core principles remain engine-independent: player-centered design, modular architecture, iteration, playtesting, readability, accessibility, measurable quality, and risk-aware production.

РОЗДІЛ 1. КОНЦЕПЦІЯ, ІГРОВІ МЕХАНІКИ ТА ВИРОБНИЧИЙ ПРОЦЕС

Мета розділу: сформувати системне бачення гри як інтерактивного програмного продукту та виробничого процесу. Основний акцент зроблено на взаємозв'язку аудиторії, платформи, правил, core loop, досвіду гравця, документації та ітеративного pipeline.

На початку розробки найбільшу цінність має не кількість контенту, а здатність команди швидко зменшувати невизначеність. Для цього формулюють проблему й цінність, перевіряють core mechanic, створюють lightweight artifacts, проводять короткі плейтести та фіксують рішення.

Матеріал, представлений у розділі, поєднує game design, design thinking і production. Такий підхід допомагає перейти від загальної ідеї до керованого прототипу, у якому кожна функція має мету, власника, параметри, критерії якості та доказ працездатності.

Після вивчення розділу здобувач набуває компетентності:

- пояснювати складові ігрового застосунку та вплив платформи на дизайн;
- описувати ролі команди, етапи production pipeline і ключові артефакти;
- застосовувати дизайн-мислення й дослідження гравця до формування концепції;
- визначати механіки, правила, dynamics, aesthetics, core loop і meta loop;
- створювати Mini-GDD, vision, pillars, persona, user stories і конкурентний аналіз;
- планувати прототип, vertical slice, milestones та критерії успіху.

Ключові терміни: ігровий застосунок, game design, design thinking, persona, MDA, core loop, meta loop, mechanic, production pipeline, prototype, vertical slice, GDD.

Тема 1.1. Вступ до розробки ігрових додатків: індустрія, ролі, pipeline, артефакти

Мета теми: сформувати у студентів цілісне уявлення про сучасну розробку ігрових застосунків як міждисциплінарний процес, визначити базову термінологію, ролі в команді, типові етапи (pipeline) та артефакти (документи/прототипи), які використовуються в індустрії.

Очікувані результати навчання

- вміти пояснити, чим відрізняється «ігровий застосунок» від інших типів ПЗ, та назвати ключові компоненти гри;
- описати базовий production pipeline: ідея -> дослідження -> прототип -> вертикальний зріз -> контент -> альфа/бета -> реліз;
- відрізнити ролі: game designer, programmer, technical artist, UI/UX, artist, QA, producer;

- знати мінімальний набір проєктних артефактів: vision/pillars, mini-GDD, core loop, user stories, asset list, backlog, тест-план;
- побудувати просту діаграму core loop та зробити mini-GDD на 1 сторінку.

Ключові терміни: ігровий цикл (game loop), механіка (mechanic), правило (rule), динаміка (dynamics), естетика (aesthetics), петля взаємодії (interaction loop), системність (systems design), прототип, вертикальний зріз (vertical slice), ігровий рушій (engine), артефакт, пайплайн, ітерація, UX в іграх.

1.1.1 Що таке «ігровий застосунок», з чого він складається

Ігровий застосунок - це інтерактивна програмна система, у якій користувач (гравець) взаємодіє з правилами, механіками, контентом та зворотним зв’язком (візуальним/звуковим/тактильним) у межах певної мети або фантазії. На відміну від утилітарних систем, гра створює цінність через досвід: виклик, відкриття, історію, самовираження, майстерність, соціальну взаємодію.

Кожну гру зручно розглядати як поєднання трьох шарів: 1 - правила та симуляція (системний шар); 2 - взаємодія та інтерфейс (HCI/UX шар); 3 - аудіовізуальна подача (арт/звук/наратив) (рис. 1.1).

Три шари ігрового застосунку є наступними

Правила визначають, що дозволено, що заборонено та як працює система гри.

Взаємодія перетворює правила на дії гравця через систему керування, інтерфейс і зворотний зв'язок.

Подача робить взаємодію зрозумілою, виразною та емоційно відчутною.

1.1.2 Індустрія та типи продуктів

Ігрова індустрія включає мобільні, ПК та консольні ігри, веб-ігри, VR/AR, симулятори, серйозні ігри (serious games) для освіти/медицини/промисловості.

Навіть у межах одного жанру існують різні бізнес-моделі: premium, free-to-play, підписка, рекламна монетизація, DLC. Платформа визначає вимоги до продуктивності, типу контролера, розміру екрана, сесійності, інструментів тестування та способу дистрибуції (табл. 1.1; рис. 1.2).

Таблиця 1.1

Платформа як драйвер дизайн-рішень

Платформа	Типові контролери	Контекст використання	Ключові обмеження
Mobile	Touch, gyro	короткі сесії, «на ходу»	батарея, пам’ять, DPI, store policies
PC	keyboard/mouse, gamepad	довші сесії, моди	різноманітне залізо, графічні налаштування
Console	gamepad, motion	вітальня, великий екран	сертифікація, fixed hardware
VR	6DoF controllers, hand tracking	занурення, фізика руху	motion sickness, FPS, latency



Рис. 1.1. Три рівні ігрового застосунку (правила/взаємодія/подача)



Рис. 1.2. Основні елементи ігрового дизайну



Рис. 1.3. Ітеративний pipeline розробки гри

1.1.3 Production pipeline: від ідеї до релізу

У спрощеному вигляді процес створення ігрового застосунку можна подати як ітеративну послідовність, де кожна ітерація зменшує невизначеність (ризики) і додає доказів того, що продукт «працює» (fun, performance, clarity).

Ключові етапи реалізації наступні

1. Ідея та бачення: короткий pitch, цільова аудиторія, платформа, «чому це цікаво».
2. Дослідження: аналіз конкурентів, референсів, гравців, ринку, технічних ризиків.
3. Прототип: перевірка ядра (1-2 механіки) без «дорогого» контенту.
4. Вертикальний зріз: маленький, але «якісний» фрагмент, що відображає фінальну якість.
5. Виробництво контенту: рівні, персонажі, UI, звук, VFX, наратив.
6. Альфа/бета: стабілізація, баланс, оптимізація, QA, збір фідбеку.
7. Реліз і підтримка: патчі, аналітика, live-ops (за потреби) (рис. 1.3).

1.1.4 Ролі та взаємодія в команді

Сучасні ігри створюються командами, де ролі можуть бути як вузькими (UI artist, gameplay programmer), так і гібридними (technical designer, tech artist). Для навчального курсу важливо розуміти «відповідальність за результат», а не лише назву ролі (табл. 1.2).

Таблиця 1.2

Ролі у створенні ігрового застосунку (узагальнено)

Роль	**Що робить (коротко)**	**Типові артефакти**
Game Designer	формує правила, механіки, баланс, досвід гравця	core loop, GDD, схематика, таблиці параметрів
Gameplay Programmer	реалізує механіки, керування, логіку, AI	код, unit tests, профайлинг
Technical Artist	місток між арт і кодом: шейдери, пайплайн, інструменти	матеріали, VFX графи, exporters
UI/UX Designer	екрани, навігація, інтеракції, онбординг	wireframes, UI flow, прототипи
3D/2D Artist	персонажі, оточення, концепт	асети, style guide, moodboard
QA	пошук дефектів, регресія, тест-плани	bug reports, тест-кейси
Producer	планування, ризики, комунікація, пріоритети	roadmap, backlog, ретроспективи

1.1.5 Документація та артефакти: «мінімальний набір» для студента

Документація в ігровій розробці має служити двом цілям: 1 - узгодити бачення; 2 - зменшити помилки комунікації. У навчальному контексті важливо навчитися створювати «легкі» документи, які справді використовуються в роботі.

Мінімальний набір артефактів для будь-якого студентського проєкту містить

Vision та Pillars (3–5 принципів гри, які не можна порушувати).

Цільова аудиторія, платформа та контекст використання.

Core Loop діаграма (що гравець робить знову і знову).

Mini-GDD (1–2 сторінки): жанр, механіки, правила, прогресія, UI, арт-напрям.

Backlog задач (Trello/Jira/Notion) з пріоритетами.

Asset list (список асетів) та naming convention.

План тестування (5–10 базових тест-кейсів) (рис. 1.4).



Рис. 1.4. Ключові компоненти оформлення розробки ігор

1.1.6 Інструменти: мінімальний робочий набір

Інструменти обираються не «бо модно», а під задачі. Для старту достатньо мінімального стеку, який дозволяє швидко створювати прототипи та фіксувати результати.

Рекомендовані програми та сервіси

1. Ігровий рушій: Unity (LTS) або Unreal Engine 5 (актуальна версія).
2. IDE: Visual Studio / Rider (Unity), Visual Studio / Rider (UE Стата), або VS Code (для скриптів).
3. Графіка/3D: Blender; для 2D - Krita або GIMP; для UI - Figma.
4. Контроль версій: Git (GitHub/GitLab) або Perforce (за бажанням для UE).
5. Планування: Trello / Notion / Jira (основи).
6. Діаграми: Miro / FigJam / draw.io (diagrams.net).
7. Звук: Audacity; інтеграція - FMOD (опційно) (табл. 1.3).

1.1.7 Часті помилки новачків і як їх уникати

Таблиця 1.3

Типові помилки та «виправлення»

Помилка	Виправлення (що робити)
«Велика ідея без прототипу»	Спочатку перевірте 1 механіку. 30 хв прототипу цінніші за 30 сторінок тексту.
«Контент замість системи»	Поки core loop не працює, не робіть 20 рівнів і 100 предметів.
«Немає критеріїв успіху»	Формалізуйте: що означає «працює» (FPS, 1 хв fun, 0 критичних багів).
«Файли без структури»	Naming convention: папки та версії. Це заощадить тижні.
«Команда без ролей»	Навіть у 2–3 людей ролі мають бути явними. Визначте відповідального за інтеграцію.

Питання для самоперевірки

1. Назвіть три шари ігрового застосунку та поясніть, як вони взаємодіють.
2. Поясніть різницю між прототипом і вертикальним зрізом.
3. Що таке core loop і чому він важливіший за кількість контенту на старті?
4. Які 3 артефакти/документи ви створите, якщо завтра почнете робити гру в команді з 3 людей?
5. Які обмеження платформи (mobile/PC/console/VR) найсильніше впливають на дизайн?

Практичний кейс до теми (час виконання 60-90 хв)

«Mini-GDD та Core Loop для улюбленої гри»

Оберіть одну відому гру (будь-яку), яку ви добре знаєте. Ваше завдання - швидко «розпакувати» її дизайн як продукт, виділивши ядро та мінімальний набір артефактів.

Формат здачі: 1 PDF або 1 Google Doc (1-2 сторінки) та 1 діаграма core loop (Miro/FigJam/draw.io).

Визначте платформу, контролер, контекст використання (2-3 речення).

Сформулюйте Vision гри в 1 реченні та 3-5 Pillars.

Опишіть core loop одним реченням, а потім намалюйте діаграму (3–7 кроків із петлею).

Виділіть 5 ключових механік (дієслово та об’єкт).

Складіть mini-список артефактів: 10 задач (backlog) та 10 асетів (asset list).

Опційно (бонус): який прототип ви зробили б першим за 2 години?

Порада: оцініть, що саме викликає «fun» у грі (виклик, освоєння, відкриття, історія, соціальність), і підкресліть це в Pillars.

Рекомендовані джерела та матеріали

1. GDC Vault (вибрані доповіді про дизайн і розробку): https://www.gdcvault.com/
2. Unity Manual та Learn: https://docs.unity3d.com/Manual/index.html; https://learn.unity.com/
3. Unreal Engine Documentation та Learning: https://dev.epicgames.com/documentation/; https://dev.epicgames.com/community/learning
4. MDA Framework (огляд): https://users.cs.northwestern.edu/~hunicke/MDA.pdf
5. Game Programming Patterns (безкоштовна книга): https://gameprogrammingpatterns.com/
6. Itch.io - приклади інді-проєктів та джемів: https://itch.io/

Тема 1.2. Дизайн-мислення та дослідження гравця

Мета теми: сформувати у студентів орієнтований на гравця підхід до створення ігрової концепції, навчити збирати й інтерпретувати дані про аудиторію, формулювати проблему, генерувати альтернативи, швидко створювати прототип (прототипувати) та перевіряти гіпотези.

1.2.1 Дизайн-мислення у контексті розробки ігор

Дизайн-мислення доцільно розглядати не як окрему творчу вправу, а як керований ітеративний процес зменшення невизначеності. На початку проєкту команда має багато припущень: хто буде грати, чому ця людина повернеться до гри, яка механіка створить потрібну емоцію, чи платформа відповідає контексту використання. Кожна наступна дія повинна перетворювати припущення на перевірені знання.

Для ігрового застосунку особливо важливо відокремити авторське бачення від реального досвіду гравця. Автор може вважати механіку очевидною, а рівень - простим, однак плейтест часто показує іншу картину. Тому дизайн-мислення починається з емпатії та спостереження, а завершується не «ідеальним документом», а перевіреним прототипом і рішенням щодо наступної ітерації.

Класична модель Empathize - Define - Ideate - Prototype - Test у геймдеві працює нелінійно. Після тесту команда може повернутися до формулювання проблеми, змінити цільову аудиторію, спростити core loop або відмовитися від дорогої функції. Такий відступ не є помилкою; навпаки, він свідчить, що ризик було виявлено раніше, ніж розпочалося масштабне виробництво контенту.

Продуктове мислення доповнює дизайн-мислення трьома запитаннями: чому гравець має обрати цей продукт; що саме він робитиме більшу частину часу; як команда реалізує цей досвід у межах наявних ресурсів. Вдале рішення повинно одночасно бути бажаним для аудиторії, технічно здійсненним і життєздатним для команди або замовника (рис. 1.5) (табл. 1.4).

1.2.2 Гравець, персона та контекст використання

Persona - це не вигаданий «середній користувач», а компактна модель поведінки цільової групи. Для навчального проєкту достатньо однієї основної та однієї додаткової персони. У профілі потрібно фіксувати не декоративні біографічні деталі, а характеристики, що впливають на дизайн: платформу, досвід у жанрі, тривалість сесії, контролер, мотивацію, типові бар'єри, ставлення до складності, доступність і очікуваний темп.

Контекст гри визначає не менше ніж жанр. Мобільна гра у транспорті потребує коротких сесій, швидкого відновлення стану й читабельного touch UI. Кооперативна PC-гра передбачає соціальну координацію, голосовий зв'язок, довші сесії та більшу толерантність до складного керування. VR-досвід обмежений фізичним простором, комфортом, частотою кадрів і ризиком кіберхвороби.

Карта емпатії структурує спостереження за чотирма площинами: що гравець робить і говорить; що він бачить у середовищі; що він думає й відчуває; які в нього болі та бажані результати. Важливо не підмінювати дані оцінкою розробника. Формулювання «гравець неуважний» є слабким; доцільніше зафіксувати спостереження: «3 з 5 учасників не помітили prompt у правому нижньому куті протягом перших 20 секунд».

Для курсового проєкту persona має бути пов'язана з рішеннями, які можна перевіряти. Якщо основна аудиторія має невеликий досвід у жанрі, потрібні поступове розкриття систем, зрозумілі підказки та можливість повторити навчальну дію. Якщо продукт орієнтований на досвідчених гравців, доцільно передбачити remapping, скорочення tutorial, глибші параметри та швидкий доступ до складних систем (рис. 1.6; табл. 1.5).

Таблиця 1.4

Етапи дизайн-мислення та їхні результати у геймдеві

Етап	Ключове запитання	Типові дії	Артефакт
Empathize	Хто наш гравець і в якому контексті він грає?	інтерв'ю, спостереження, аналіз відгуків, карта емпатії	persona, pain points, контекст сесії
Define	Яку проблему або незадоволену потребу ми вирішуємо?	синтез даних, формулювання HMW, визначення критеріїв успіху	problem statement, design challenge
Ideate	Які альтернативи можуть забезпечити потрібний досвід?	brainstorming, brainwriting, SCAMPER, mind map, sketchstorming	набір концепцій, пріоритети
Prototype	Який найдешевший спосіб перевірити головний ризик?	paper prototype, greybox, fake-door, scripted sequence	тестовий прототип
Test	Що насправді зрозумів і відчув гравець?	плейтест, спостереження, інтерв'ю після сесії, telemetry	висновок і план ітерації

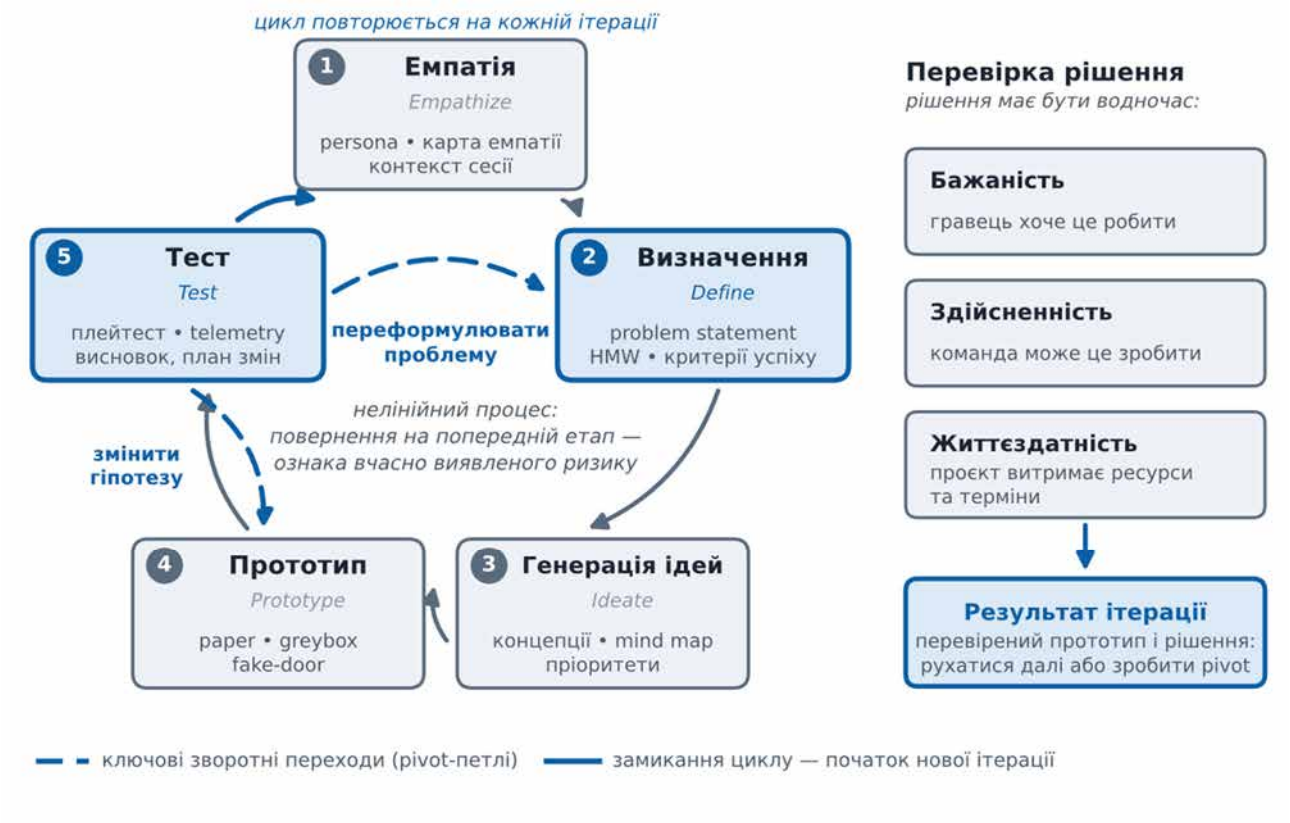


Рис. 1.5. Цикл дизайн-мислення для ігрового проєкту

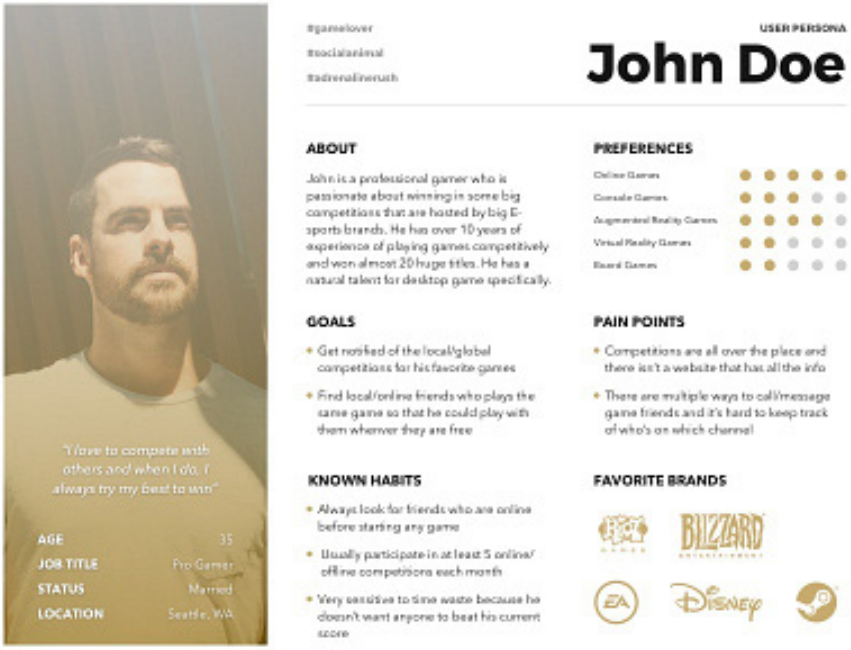


Рис. 1.6. Профіль гравця та контекст ігрової сесії

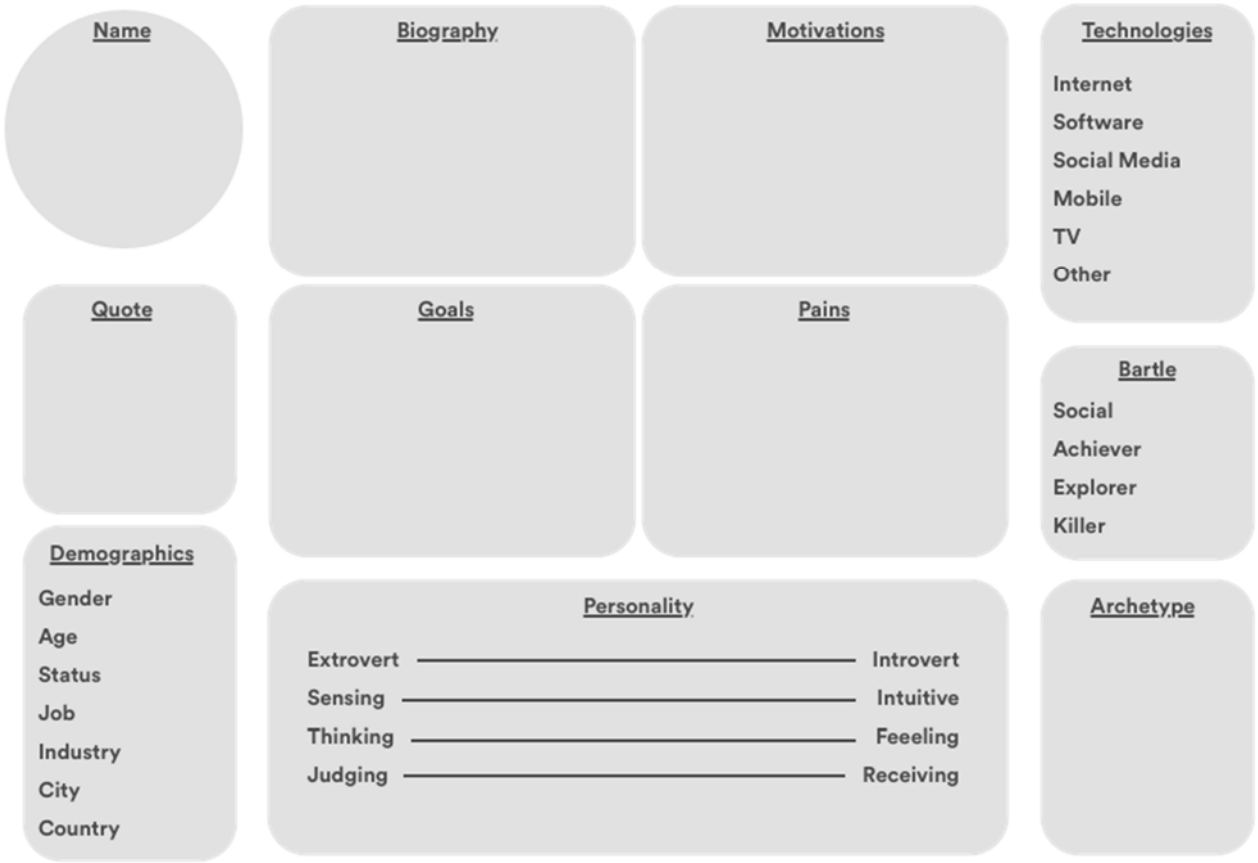


Рис. 1.6. Профіль гравця та контекст ігрової сесії

Таблиця 1.5

Структура персони гравця

Компонент персони	**Що фіксуємо**	**Як впливає на дизайн**
Платформа і контролер	PC/gamepad, mobile/touch, console, VR	схема input, safe areas, розмір UI, latency
Досвід у жанрі	новачок, середній, експерт	onboarding, складність, термінологія
Тривалість сесії	2-5 хв, 20-40 хв, 1-3 год	структура loop, checkpoints, save system
Мотивація	майстерність, відкриття, історія, соціальність, творчість	нагороди, прогресія, контент
Бар’єри	час, моторні/зорові обмеження, перевантаження, страх помилки	accessibility, assist, читабельність
Критерій цінності	що має відчути або досягти гравець	pillars, метрики плейтесту

1.2.3 Методи дослідження та якість даних

Якісні методи пояснюють, чому виникає поведінка, а кількісні - наскільки вона поширена. Інтерв'ю, спостереження й diary study дають контекст і мову гравця. Аналітика, логи подій, воронки та time-to-complete показують масштаб проблеми. Найкращий результат дає triangulation - зіставлення кількох джерел.

Інтерв'ю доцільно будувати навколо конкретного досвіду, а не гіпотетичних уподобань. Запитання «Чи хотіли б ви складну систему крафту?» провокує бажану відповідь. Запитання «Розкажіть про останню гру, де ви регулярно крафтили; що саме змушувало вас повертатися до цієї системи?» дає поведінкові дані.

Co-design корисний, коли потрібно зрозуміти ментальну модель інтерфейсу або рівня. Учасник може розмістити картки HUD, зібрати бажаний flow меню або накреслити ідеальний шлях через локацію. Card sorting допомагає перевірити групування предметів, налаштувань, квестів. Diary study доречний для ігор із довгою прогресією, де емоції змінюються протягом кількох днів.

Дані плейтесту слід збирати за однаковим сценарієм. Перед початком визначають ціль, завдання, критерії успіху та спосіб фіксації результатів. Під час сесії фасилітатор не навчає, якщо це не передбачено протоколом; він спостерігає за діями, паузами, помилками та коментарями. Після завершення варто провести коротке інтерв'ю, але не дозволяти самооцінці замінити фактичну поведінку (табл. 1.6).

Таблиця 1.6

Методи дослідження гравця

Метод	Коли застосовувати	Результат	Типова помилка
Напівструктуроване інтерв'ю	перед концептом або після сесії	мотиви, ментальні моделі, мова аудиторії	навідні та гіпотетичні запитання
Контекстне спостереження	для реального середовища гри	перешкоди, контекст, поведінкові патерни	втручання у процес
Co-design	для UI, flow, рівня, функцій	варіанти структури та пріоритетів	сприймати ескіз як фінальний дизайн
Card sorting	для меню, інвентарю, категорій	природне групування інформації	занадто багато карток
Diary study	для довготривалого досвіду	зміни мотивації, фрустрації, звички	нечіткі правила запису
Telemetry	у працездатному прототипі	частота подій, funnel, час, помилки	логувати все без гіпотези
A/B тест	для двох конкретних варіантів	порівняльний результат	змінювати кілька факторів одночасно

1.2.4 Формулювання проблеми та критерії успіху

Після збору даних команда формулює problem statement. Сильне формулювання описує цільову групу, контекст, спостережуваний бар'єр і бажаний результат, але не нав'язує конкретне рішення. Наприклад: «Нові

гравці кооперативного прототипу не розуміють, коли доступна взаємодія, через що пропускають ключові об’єкти в перші 2 хвилини».

Формат How Might We переводить проблему в поле ідей: «Як ми можемо зробити доступність взаємодії помітною, не перевантажуючи екран постійними підказками?». Після цього можна порівнювати альтернативи: підсвітка, зміна курсора, короткий звук, spatial prompt, анімація об’єкта, контекстна підказка або їх комбінація.

Критерії успіху формулюють під час створення прототипу (прототипування). Для наведеного прикладу це може бути: 80% нових учасників знаходять і використовують ключовий об’єкт без усної підказки; перший фідбек з’являється до 100 мс; prompt не перекриває критичний HUD; кількість помилкових активацій не перевищує 1 на сесію (рис. 1.7).



Рис. 1.7. Зв’язок «спостереження - проблема - гіпотеза - метрика»

1.2.5 Генерація ідей і вибір концепції

На етапі ideation необхідно розрізняти дивергентне й конвергентне мислення. Спочатку команда створює багато варіантів без попередньої критики. Після цього ідеї групують, уточнюють і оцінюють за такими критеріями: цінність для гравця, технічний ризик, вартість контенту, час реалізації, відповідність pillars і можливість швидкої перевірки.

Brainwriting зменшує домінування найактивніших учасників: кожен записує кілька ідей, після чого аркуші або цифрові картки передаються іншим для подальшого розвитку. Sketchstorming змушує показати дію, простір і фідбек, а не обмежуватися абстрактними словами. Mind map корисна для декомпозиції центральної фантазії на механіки, контент, ризики та залежності.

SCAMPER допомагає модифікувати наявну механіку: Substitute - замінити ресурс або дію; Combine - об’єднати дві системи; Adapt - перенести патерн з

іншого жанру; Modify - змінити масштаб, швидкість або правило; Put to another use - використати елемент в іншій ролі; Eliminate - прибрати зайве; Rearrange - змінити порядок і причинність.

Для вибору не варто покладатися лише на голосування. Доцільно використовувати матрицю impact/effort, dot voting як попередній фільтр, а далі - короткий feasibility review із програмістом, художником і дизайнером. Найперспективнішою є не найбільша ідея, а та, яка перевіряє найважливіше припущення з мінімальними витратами (табл. 1.7) (рис. 1.8) (рис. 1.9).

Таблиця 1.7

SCAMPER для розвитку ігрової ідеї

SCAMPER	Запитання для ігрової механіки	Приклад
Substitute	Що можна замінити: ресурс, контролер, ціль?	замість боєприпасів використати час або світло
Combine	Які дві дії можуть створити нову динаміку?	рух + будівництво, діалог + ритм
Adapt	Який патерн з іншого жанру доречний?	deckbuilding-прогресія у тактичній грі
Modify	Що станеться при зміні масштабу/темпу?	короткі сесії замість довгих рівнів
Put to another use	Чи може об’єкт мати другу функцію?	щит як платформа або джерело світла
Eliminate	Що можна прибрати без втрати fantasy?	зайвий ресурс або дубльований HUD
Rearrange	Що зміниться при іншому порядку?	спочатку нагорода, потім ризик; зворотний tutorial

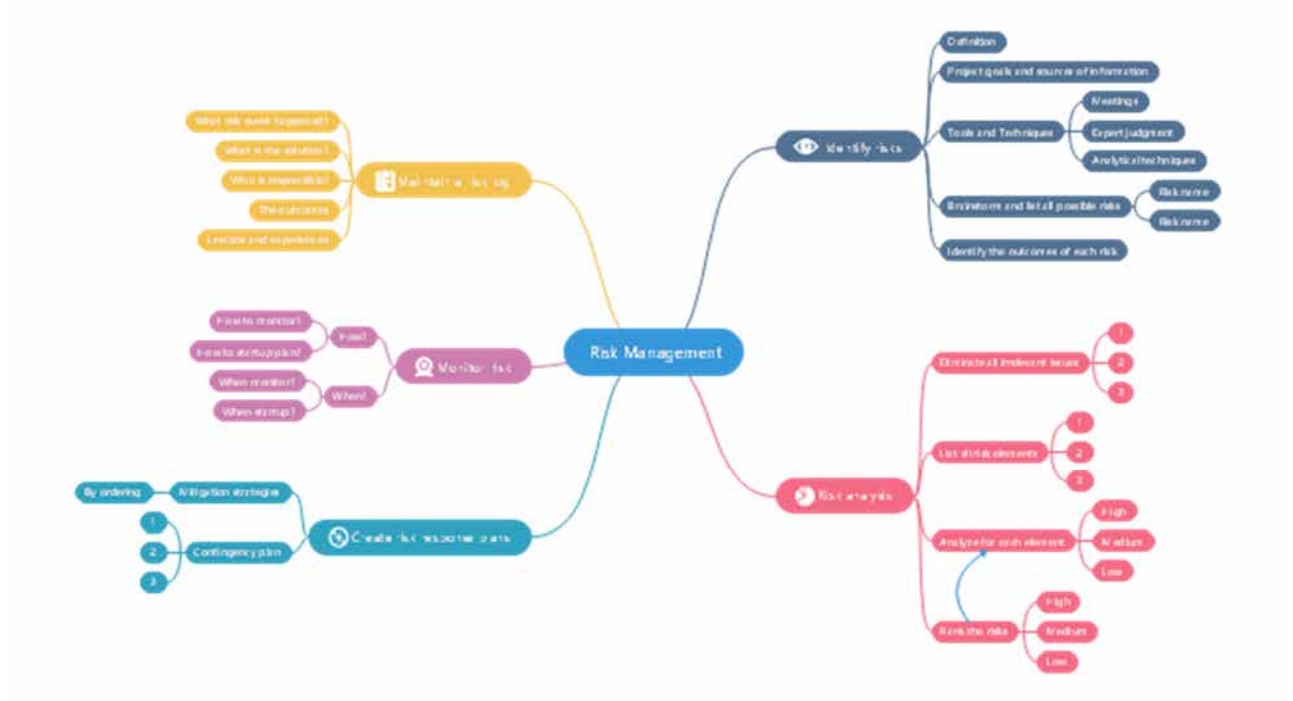


Рис. 1.8. Mind map функцій і ризиків ігрової концепції

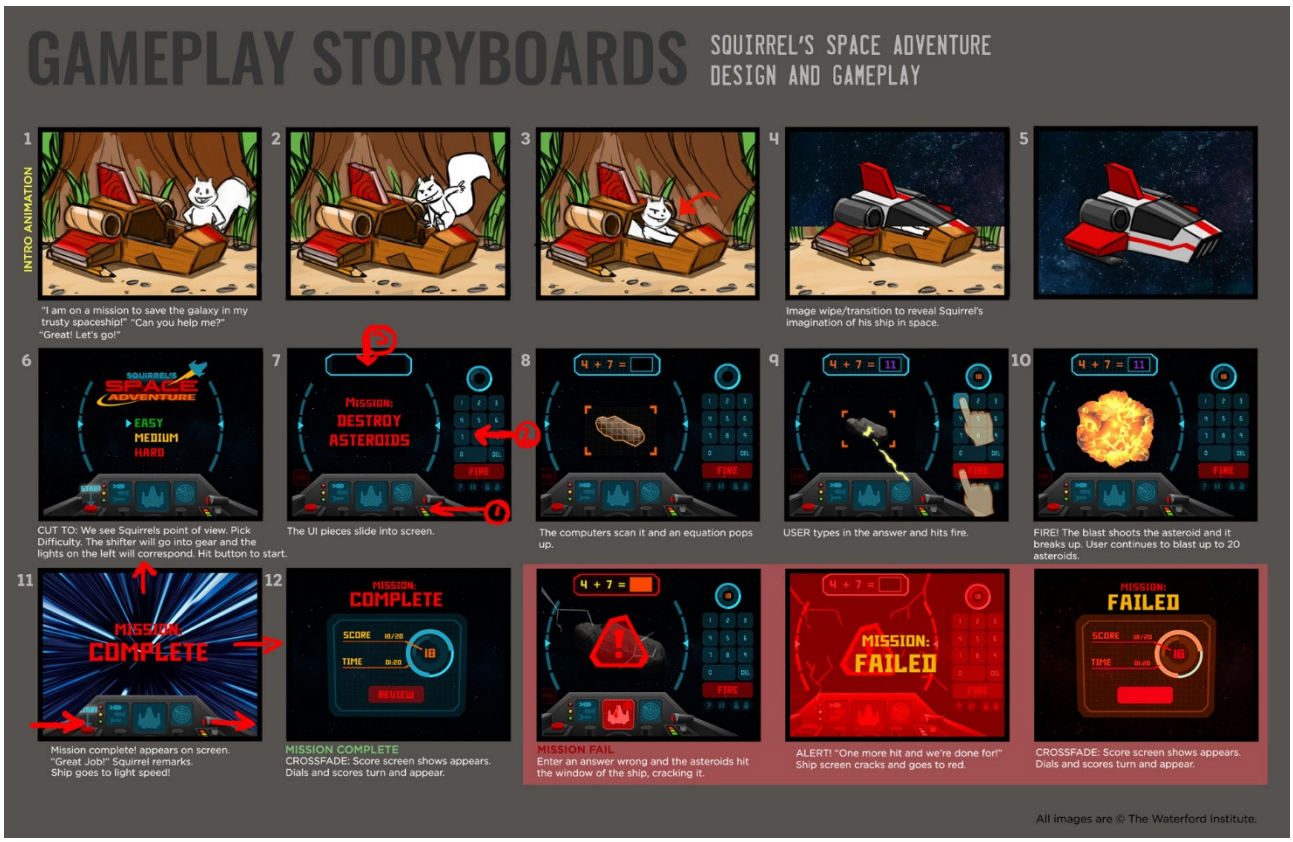


Рис. 1.9. Storyboard першої ігрової сесії

1.2.6 Прототипування і тестування гіпотез

Прототип має відповідати на одне головне запитання. Paper prototype перевіряє правила, economy або flow; greybox - навігацію, масштаби й темп; scripted prototype - емоційний beat; technical spike - ризик рушія, шейдера, мережі або продуктивності. Змішування всіх цілей робить результат дорогим і нечитабельним.

Перед тестом варто сформувати карту: гіпотеза - зміна - метрика - поріг рішення. Наприклад: «Якщо додати 150 мс anticipation перед атакою ворога та контрастний telegraph, частка ударів, які вдалося уникнути, зросте щонайменше з 35% до 60%». Після тесту команда приймає рішення: залишити, змінити, відкласти або відмовитися.

Зворотний зв'язок не слід перетворювати на голосування за бажання. Гравець є експертом у власному досвіді, але не зобов'язаний бути дизайнером рішення. Команда аналізує не лише пропозицію «зробіть кнопку більшою», а причину: кнопку не помітили, вона була недоступною, її стан нечитабельний або дія не відповідала очікуванню.

Міні-практикум «Гіпотеза за 60 хвилин»:

Оберіть одну проблему у знайомій грі або власному прототипі. Створіть problem statement, сформулюйте HMW, згенеруйте щонайменше 12 ідей, оберіть одну за рівнем impact/effort, зробіть paper/wireframe-прототип і проведіть тест із 2 учасниками. Результат подайте у форматі «спостереження - гіпотеза - прототип - дані - рішення» на 1–2 сторінках.

Тема 1.3. Ігрові механіки, MDA, core loop та балансування

Мета теми: сформувати у студентів розуміння того, як «ігрові механіки» та «правила» перетворюються на досвід гравця через ігровий цикл (game loop), а також навчити базових підходів до аналізу MDA (Mechanics-Dynamics-Aesthetics) і первинного балансування (difficulty/economy/progression) на прикладах реальних ігор.

Очікувані результати навчання

- вміти визначати механіку як «дієслово та об’єкт та умова та зворотний зв’язок» та відрізняти механіки від контенту;
- описувати core loop і meta loop гри у вигляді короткого тексту, діаграми або псевдокоду;
- аналізувати гру через MDA: від механік -> до динамік -> до естетики/емоційного ефекту;
- виділяти параметри для балансування (швидкість, шкода, вартість, дроп, таймінги, RNG) та пропонувати коригування;
- планувати простий плейтест і збір даних/телеметрії для перевірки гіпотез щодо балансу та «game feel».

Ключові терміни: механіка (mechanic), правило (rule), система (system), ігровий цикл (game loop), core loop, meta loop, MDA (Mechanics-Dynamics-Aesthetics), балансування (balancing), складність (difficulty), економіка (economy), прогресія (progression), RNG (random number generation), telegraphing (попереджувальні сигнали), hit-stop / hit-scan, time-to-fun, time-to-fail, retention loop.

1.3.1 Що таке ігрова механіка: від правил до дії гравця

Ігрова механіка - це не «фіча» і не «контент», а відтворюване правило взаємодії, яке визначає: 1) що може зробити гравець; 2) за яких умов; 3) як система реагує; 4) яку інформацію/зворотний зв’язок отримує гравець. Практичне правило опису: механіка - це дієслово, об’єкт, умова та зворотний зв’язок.

Наприклад, «стрибати» - це дієслово, але механіка повністю формується, коли задані параметри (висота, прискорення, крива, coyote time), умови (на землі/в повітрі/на стіні) та зворотний зв’язок (анімація, звук, камера, частинки, хітбокси). Саме сукупність цих елементів формує «feel» і навчання гравця (рис. 1.10; табл. 1.8).

Зазвичай думають про механіки, як про «атомарні блоки», з яких складаються системи (комбат, пересування, економіка, крафт, соціальні взаємодії). З’єднання механік створює більші структури - «системи», а взаємодія систем у часі дає динаміку.

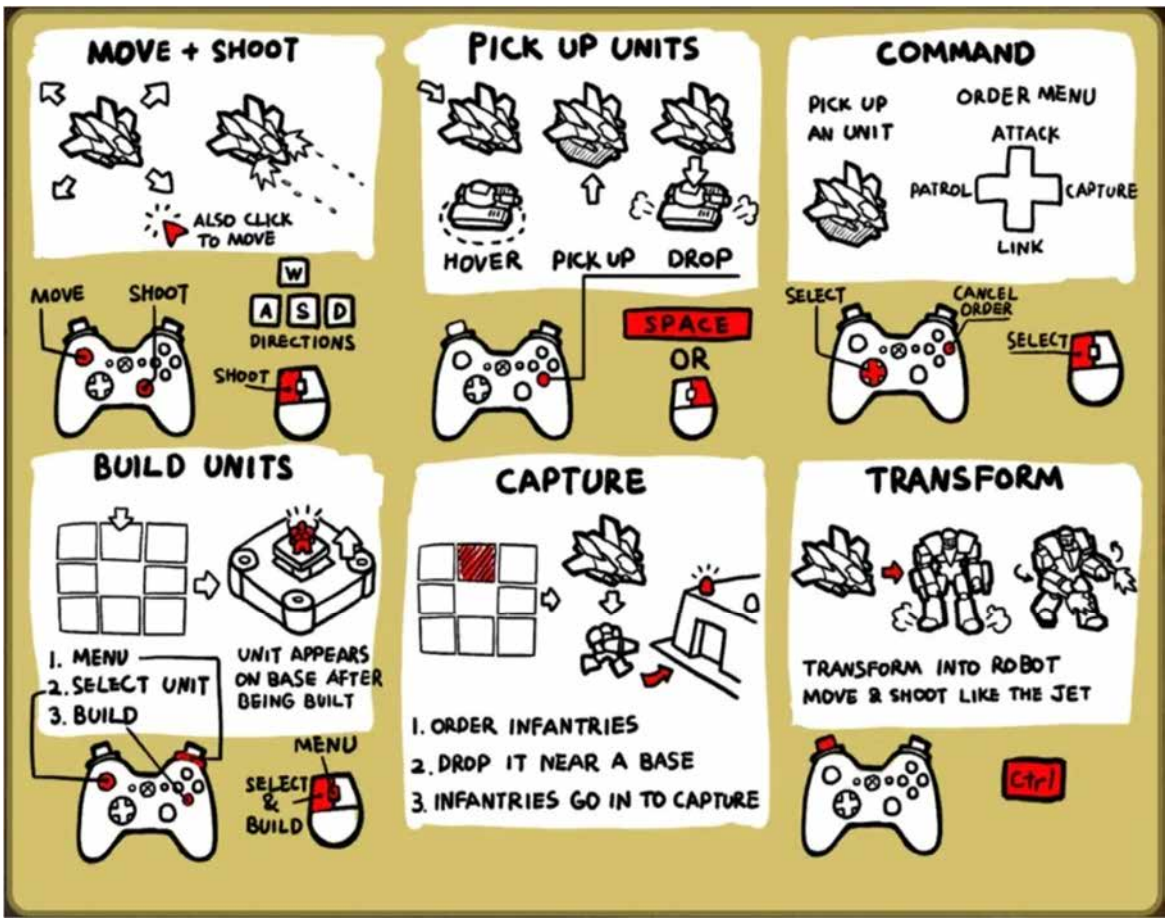




Рис. 1.10. Головні елементи ігор

Таблиця 1.8

Механіка як структурний елемент дизайну

Механіка	**Дія/об'єкт/умова**	**Фідбек (що бачить/чує гравець)**	**Ключові параметри для тюнінгу**
Стрибок	натиснути Jump; якщо grounded	анімація, звук, камера; траєкторія	висота, гравітація, coyote, буфер
Даш	натиснути Dash; якщо cooldown=0	ривок, трейл, FOV, SFX	тривалість, швидкість, cooldown, i-frames
Постріл	натиснути Fire; якщо ammo>0	muzzle flash, recoil, hit marker	ROF, spread, recoil, шкода, dps
Лут	підійти; натиснути Interact	підсвітка, звук, UI-підказка	радіус, час підбору, рідкість, RNG

1.3.2 Game Loop: core loop, session loop та meta loop

Game loop - це повторювана послідовність дій і реакцій системи, яка утримує гравця у грі та формує навички. Найчастіше виокремлюють: (1) core loop - серцевина геймплею (10–60 секунд); (2) session loop - цикл однієї сесії (кілька хвилин); (3) meta loop - довгострокова прогресія (години/тижні).

Хороший loop має: зрозумілу мету, швидкий «time-to-fun», читані сигнали (telegraphing), справедливі умови помилки, та винагороду (інформаційну, емоційну або матеріальну) (рис. 1.11; табл. 1.9).

Таблиця 1.9

Приклади core loop для різних жанрів

Жанр/приклад	**Core loop (1 речення)**	**Що модифікують для ігрового балансу**
Platformer (Celeste)	рух -> стрибок/даш -> уникнути загрози -> чекпойнт/перезапуск	таймінги, coyote, швидкість, хазард
Roguelite (Hades)	бій -> лут/вибір апгрейду -> наступна кімната -> бос	dps, дроп, вороги, криві складності
Puzzle (Portal)	спостерігати -> гіпотеза -> дія з механікою -> перевірка -> просування	подача підказок, складність, темп
Sandbox (Minecraft)	добути -> створити -> побудувати/дослідити -> покращити інструменти	економіка ресурсів, ризик, прогресія
PvP Social (Among Us)	спостерігати/виконувати -> підозрювати -> дискусія -> голосування	ролі, таймінги, інформація, карти

Практична порада: коли ви плануєте нову гру, починайте з core loop і доведіть його до стану «приємно повторювати». Контент (рівні, зброя, персонажі) має сенс лише тоді, коли loop уже працює.

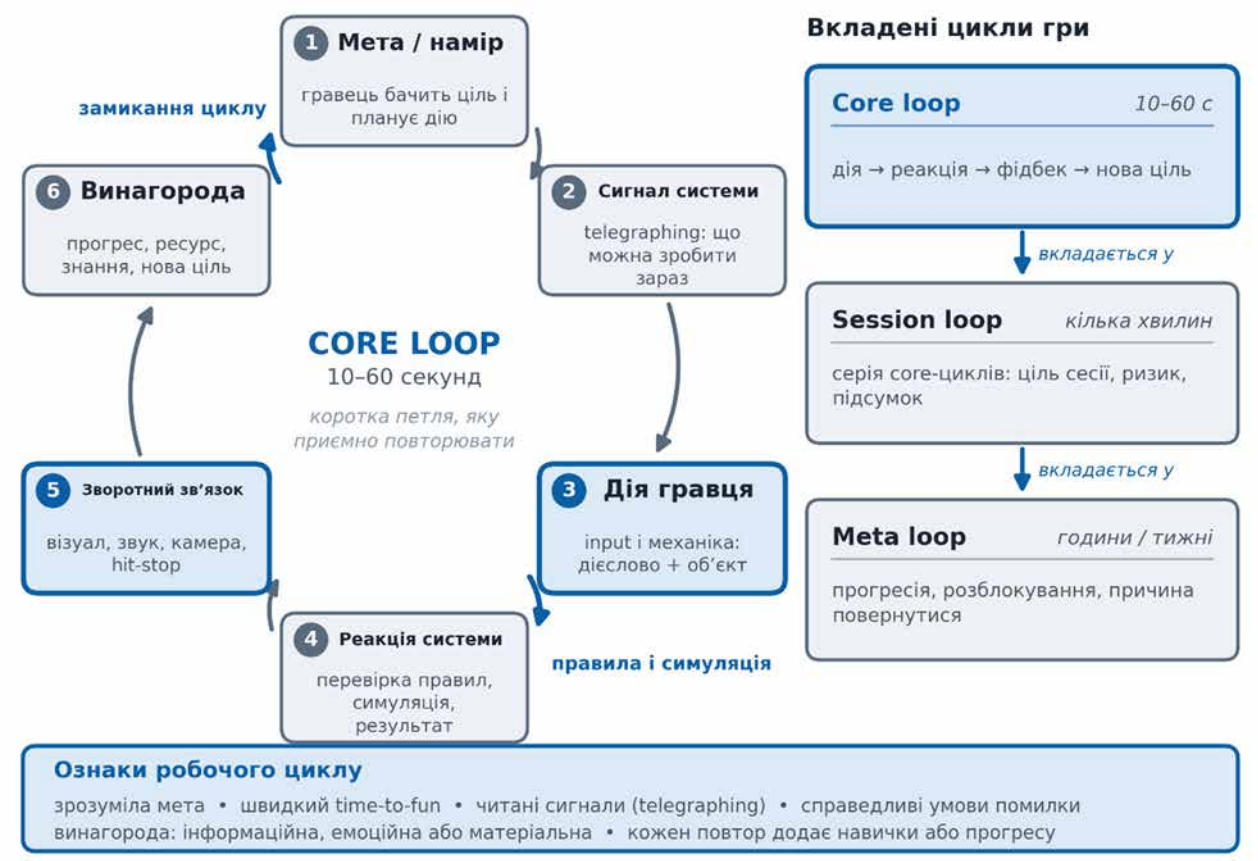


Рис. 1.11. Дизайн Ігрового циклу

1.3.3 MDA (Mechanics-Dynamics-Aesthetics): як механіки стають досвідом

Фреймворк MDA допомагає пов'язати те, що ви проєктуєте як розробник (mechanics), з тим, що виникає у грі як поведінка системи (dynamics), і тим, що відчуває гравець (aesthetics). З погляду дизайнера шлях зазвичай йде «від механік до естетики», а з погляду гравця - навпаки: він спершу відчуває емоцію/враження, а потім поступово розуміє механіки (рис. 1.12) (табл. 1.10).

Таблиця 1.10

Приклад MDA-розбору (фрагмент) для кількох ігор

Гра	Mechanics (функція)	Dynamics (дії)	Aesthetics (що відчуває гравець)
Tetris	падіння тетроміно; лінії очищаються; швидкість росте	оптимізація простору; ризик помилки; темп	напруга, потік, майстерність
Hades	атаки/деш; вибір boon; рандом кімнат	побудови (builds), ризик/нагорода, адаптація	сила, прогрес, «ще один забіг»
Among Us	ролі; завдання; саботаж; голосування	соціальна дедукція, обман, коаліції	підозра, гумор, драма
Minecraft	ресурс -> крафт; виживання; будівництво	експерименти, «свої цілі», творчість	відкриття, свобода, спокій/небезпека



Рис. 1.12. Базова логіка механіки: будь-яка механіка описується як послідовність

Важливо: MDA - не «табличка заради таблички». Це інструмент перевірки причинно-наслідкових зв'язків: які саме механіки створюють бажану динаміку, і чи призводить вона до потрібної естетики. Якщо естетика не та, найчастіше треба змінювати не «арт», а правила/таймінги/фідбек.

1.3.4 Балансування: складність, економіка, прогресія

Балансування - це керування параметрами системи так, щоб гра була справедливою, цікавою та зрозумілою. Баланс не означає «все однаково сильне»; баланс означає «вибір має сенс», а поразка - «навчальна і чесна».

Типові області балансу: (1) складність (вороги, хазард, таймінги, чекпойнти); (2) економіка (ціни, дроп, валюти, апгрейди); (3) прогресія (крива росту сили, відкриття контенту, темп навчання) (рис. 1.13; табл. 1.11).

Таблиця 1.11

«Складові» балансу та як їх перевіряти

Складова	Приклади параметрів	Що спостерігати (метрики/ознаки)	Швидкі тести
Складність	HP ворога, DPS, швидкість, кількість хазардів, чекпойнти	частота смертей, час проходження, фрустрація	3 плейтести та аналіз «де помирають»
Економіка	ціни, дроп%, ліміт ресурсів, апгрейди	дефіцит/надлишок, «обов'язкові» покупки	таблиця економіки та симуляція 10 хв
Прогресія	XP, unlocks, power curve, scaling	занадто швидкий snowball, застій	графік росту сили/складності
RNG	шанси дропу, крит, спавн	відчуття несправедливості, розкид	перевірка на 1000 прогонів (скрипт/таблиця)

Найкращий баланс - це баланс, який підтверджений плейтестом. На ранніх етапах достатньо «ручного» тестування і простих таблиць. Пізніше додають телеметрію: які кнопки натискають, де помирають, де виходять з гри, скільки часу витрачають на екрани/рівні (рис. 1.14).



Рис. 1.13. Крива складності: навчання -> виклик -> майстерність



Рис. 1.14. Логіка вкладеності циклів ігрового дизайну

1.3.5 Як моделювати механіки: діаграми, стани, псевдокод

Щоб механіки було легко обговорювати в команді (дизайнер-програміст-арт-QA), їх варто фіксувати у 3 формах: (1) короткий опис (1-3 речення); (2) діаграма (flowchart / state machine); (3) псевдокод або таблиця станів.

Мінімальний state machine для бойової механіки може містити: Idle -> Aim -> Attack -> Recovery -> (Hit/Cancel) -> Idle. Для UI/інтеракції додають стани Validating/Loading/Error/Success (рис. 1.15).

Принцип - атака розбивається на керовані стани з чіткими переходами та часовими вікнами. Це дозволяє окремо налаштовувати телеграфування, влучання, hitbox, cancel rules і recovery frames. Логіка state machine - у кожному стані дозволений окремий набір дій, таймерів і перевірок. Переходи контролюються подіями, тривалістю стану або умовами системи. Наприклад: hitbox активується лише у стані Active, а повторний input attack може бути заблокований до завершення Recovery (рис. 1.16). У практичних роботах дозволяється використовувати Miro/FigJam/draw.io для діаграм і Google Sheets/Excel для таблиць балансу. Для прототипування - Unity/Unreal/Godot, або навіть паперовий прототип, якщо мета - перевірити loop.

1.3.6 Приклади та міні-розбори (case studies)

У цьому підпункті наведено короткі приклади аналізу, які зручно повторювати на практиці. Завдання студента - не переписувати опис, а навчитися виділяти: (a) механіку, (b) цикл, (c) MDA, (d) важелі балансу.

Celeste (платформер): фокус на рух/даш; навчання через рівневі «уроки»; баланс через чекпойнти та таймінги.

Hades (roguelite): core loop - кімната -> нагорода -> вибір; баланс - через буни, вартість ресурсів, модифікатори складності.

Portal (головоломка): механіка порталу та поступове ускладнення; естетика відкриття; баланс між підказками та темпом.

Overwatch (геройський шутер): читабельність VFX/UI; баланс ролей; cooldown та здібності персонажів.

Minecraft (sandbox): системність ресурсів; прогресія інструментів; баланс ризику/нагороди через біоми та мобів (рис. 1.17).

Формула сприйняття: контакт зі зброєю -> системний результат -> багатоканальний фідбек -> інтерпретація гравцем. Якісний combat feedback поєднує синхронність, читабельність і емоційний акцент.

Приклад: меч влучає у броньованого ворога - короткий hit pause, металевий impact sound, іскри, зменшення HP, реакція ворога на stagger. Цей набір каналів разом формує відчуття достовірного контакту та підсилює читабельність бойової системи (табл. 1.12).

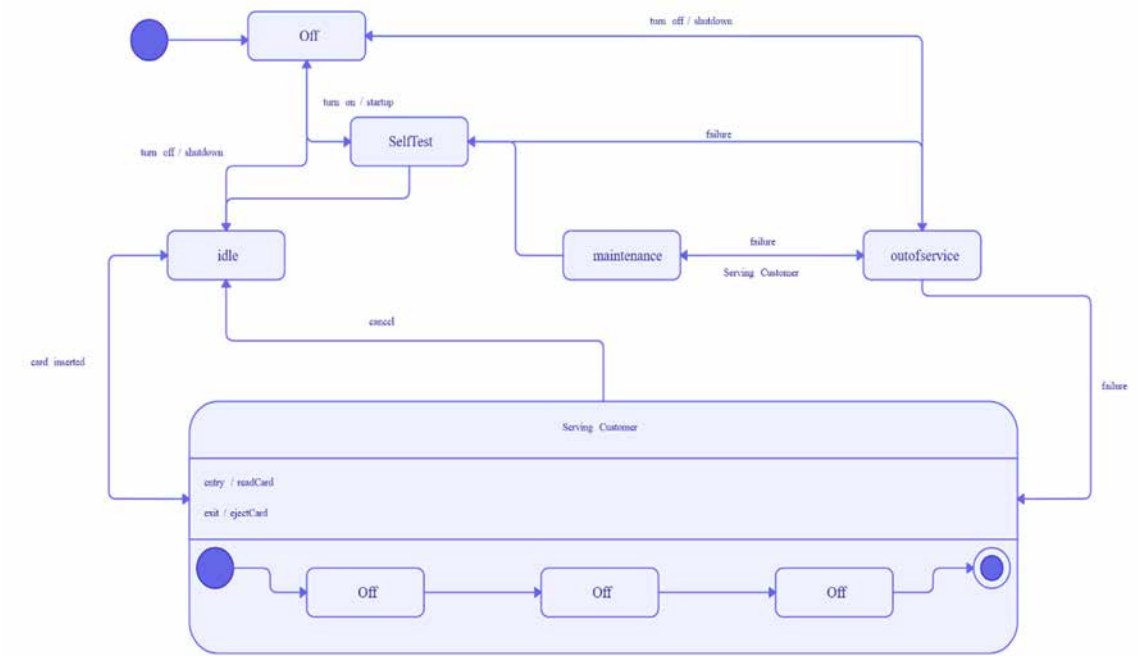


Рис. 1.15. Приклад state machine для атаки

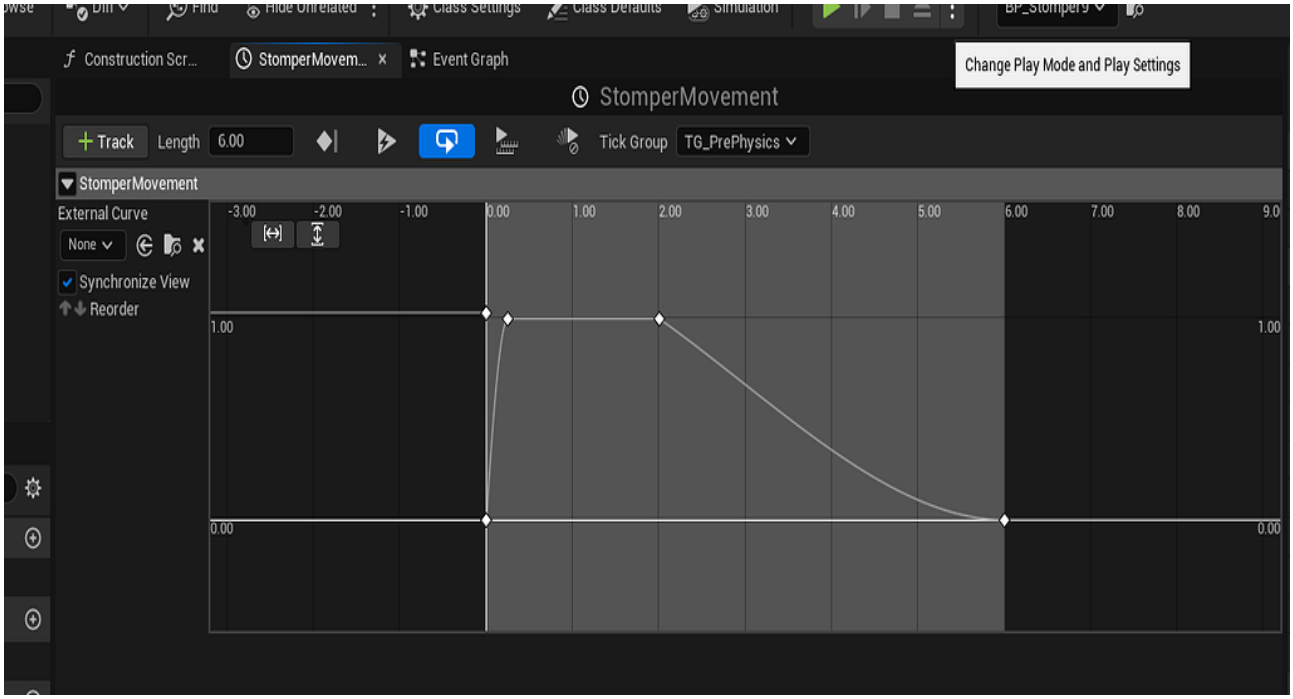


Рис. 1.16. Приклад часової структури атаки гравця або ШІ



Рис. 1.17. Фідбек: анімація удару, звук контакту, ефект попадання

Hades II (roguelite) зручний як приклад того, що складність може бути простором налаштувань, а не однією кривою. (a) Механіка: звичайні атака, спеціальна атака та закляття мають «омега»-варіанти, які витрачають ману (Magick), тож кожна дія отримує не лише таймінг, а й ресурсну ціну; до цього додаються ривок, пастка-закляття та гекси. (b) Цикл: кімната → бій → вибір нагороди (бун, ресурс, обмін) → перехід далі, а після поразки — повернення на Перехрестя, вкладення ресурсів у постійні покращення та новий забіг. (c) MDA: механіки вибору буна породжують динаміку складання білда й керування ризиком, а естетика — це майстерність, відкриття комбінацій і наратив, що розгортається саме між забігами. (d) Важелі балансу: рідкість і синергія бунів, вартість ресурсів, довжина маршруту, карти Арканів, які гравець обирає в межах ліміту, та «Клятви» — модифікатори, якими гравець свідомо підвищує тиск заради кращих нагород.

Portal 2 (головоломка) демонструє протилежний тип балансу — балансування темпу подачі знань. (a) Механіка: два портали на портальних поверхнях плюс збереження інерції, а далі гелі, тунелі, мости й лазери. (b) Цикл: камера → зчитування простору → гіпотеза → перевірка → розв'язок → нова камера. (c) MDA: мінімум правил породжує динаміку експерименту, а естетика — «ага-момент» відкриття, підсилений гумором і наративом. (d) Важелі балансу: одна нова ідея на камеру з подальшою варіацією та комбінацією, візуальні підказки (біла поверхня як дозволена зона порталу), низька ціна помилки (час і перезапуск, а не покарання) та дозована вербальна підказка.

Overwatch (геройський шутер) показує, що в багатокористувацькій грі читабельність є частиною балансу, а не оформленням. (a) Механіка: кожен герой — це набір здібностей із кулдаунами, ресурс ультимейту та чітка роль (танк, шкода, підтримка), тому «дієслово» гравця завжди прив’язане до ролі й ресурсу часу. (b) Цикл: вибір героя та підготовка → зіткнення за точку або супровід → розмін кулдаунами й ультимейтами → перегрупування чи зміна героя після респауну. (c) MDA: кулдауни та контрпіки породжують динаміку координації й розміну темпом, а естетика — це змагання, товариськість і драматичні командні моменти. (d) Важелі балансу: числові параметри здібностей, тривалість кулдаунів, вартість ультимейту, правила складу команди, режимні модифікатори на кшталт перків та предметів окремих режимів, а також силуети героїв, кольорове кодування команд, обмежений «шум» VFX і голосові репліки ультимейтів, які працюють як telegraphing. Порівняння трьох прикладів дає практичний висновок для навчального проєкту: у Hades II більшу частину балансу налаштовує сам гравець, у Portal 2 — темп подачі нової інформації, в Overwatch — числа та читабельність сигналів. У всіх трьох випадках працює та сама формула сприйняття (контакт → системний результат → багатоканальний фідбек → інтерпретація гравцем): якщо гравець не встигає прочитати сигнал, дія відчувається несправедливою навіть за коректно підібраних чисел.

1.3.7 Часті помилки в дизайні механік та балансі

Таблиця 1.12

Типові помилки та «антидоти» (рекомендовані дії)

Помилка	Як виправити (антидот)
«Занадто багато механік одразу»	Зменшити до 1 core mechanic та 1 підтримуючої; решту відкласти в backlog.
«Контент замість loop»	Спершу довести core loop до стану «приємно повторювати», потім додавати рівні/зброю.
«Невидимі правила»	Додати telegraphing: підсвітка, звук, анімація, читабельні стани та підказки.
«RNG без компенсації»	Додати pity-систему/гарантію/обмеження розкиду; пояснити шанс гравцю.
«Складність стрибає»	Побудувати криву складності; додати проміжні навчальні кроки та безпечні зони.
«Баланс без тестування»	З швидкі плей-тести та таблиця параметрів; фіксувати, де гравці губляться/помирають.

Питання для самоперевірки

1. Дайте визначення ігрової механіки у формі «дієслово + об’єкт + умова + фідбек». Наведіть 2 приклади.
2. Поясніть різницю між core loop, session loop та meta loop. Який з них ви прототипуєте першим і чому?
3. Опишіть MDA на прикладі однієї гри: які механіки ведуть до яких динамік і яку естетику це створює?
4. Назвіть щонайменше 5 «важелів» балансування (параметрів), які можна змінювати без створення нового контенту.
5. Що таке telegraphing і як він впливає на відчуття справедливості у грі?

Практичний кейс «Розбір механік, loop та баланс-патч» (час виконання 90-120 хв)

Оберіть дві гри: 1 - гру, яка вам подобається; 2 - гру, яка вам не подобається (або має очевидні проблеми з балансом/loop). Ваше завдання - зробити компактний, але професійний розбір і запропонувати коригування (баланс-патч) без додавання нового контенту.

Для кожної гри: сформулюйте core loop одним реченням і намалюйте діаграму (5-9 кроків з петлею).

Виділіть 5–8 ключових механік у форматі «дієслово та об’єкт» і додайте умову/фідбек (по 1 рядку).

Зробіть MDA-таблицю (мінімум 3 механіки → 3 динаміки → 3 естетики).

Визначте 3 проблеми балансу/досвіду (наприклад: надто сильна зброя, провали складності, несправедливий RNG, економіка ламає прогресію).

Запропонуйте баланс-патч: змініть 5-10 параметрів (таймінги, DPS, cooldown, ціни, drop%, HP, швидкості) та поясніть очікуваний ефект.

Опційно (бонус): створіть маленьку таблицю-симуляцію в Google Sheets/Excel (10 хв гри або 5 боїв) і покажіть, як змінюються результати після патчу.

Порада: використовуйте діаграми (Miro/FigJam/draw.io) та таблиці (Google Sheets) - це найшвидший шлях до «дизайну, що читається програмістом».

Рекомендовані програми та сервіси

1. Діаграми та state machines: Miro, FigJam, draw.io (diagrams.net).
2. Таблиці балансу та симуляції: Google Sheets / Microsoft Excel.
3. Прототипування механік: Unity, Unreal Engine 5, Godot (мінімальний прототип).
4. Аналітика/нотатки: Notion, Obsidian; для трекінгу задач - Trello/Jira (основи).
5. Корисні калькулятори/довідники: будь-які прості DPS/TTK-калькулятори (саморобні в Sheets).

Рекомендовані джерела та матеріали

1. MDA Framework (Hunicke, LeBlanc, Zubek): https://users.cs.northwestern.edu/~hunicke/MDA.pdf
2. GDC Vault - доповіді про механіки, loop і баланс: https://www.gdcvault.com/
3. Game Feel (Steve Swink) - про «відчуття» керування та фідбек (книга).
4. Game Mechanics: Advanced Game Design (Adams & Dormans) - про моделювання і баланс (книга).
5. Game Programming Patterns (безкоштовно): https://gameprogrammingpatterns.com/
6. GMTK (Game Maker’s Toolkit) - короткі відео розбори механік: https://www.youtube.com/@GMTK
7. Extra Credits (архів) - базові концепти дизайну: https://www.youtube.com/@ExtraCredits

Тема 1.4. Документування концепції та порівняльний аналіз ігор

Мета теми: навчити перетворювати ігрову ідею на компактний набір узгоджених артефактів: vision, pillars, persona, core loop, mini-GDD, таблицю конкурентного аналізу, backlog, asset list і критерії прототипу.

1.4.1 Від ідеї до керованого набору артефактів

Документація в ігровому проєкті має не накопичувати текст, а зменшувати помилки комунікації. Кожен документ повинен мати власника, призначення, дату оновлення і зв’язок із build. Якщо опис механіки не відповідає реалізації, джерелом істини є не старий GDD, а актуальна система параметрів і рішення команди, зафіксоване в change log.

Vision формулює продукт одним реченням: хто грає, що робить і яку унікальну емоцію отримує. Pillars - це 3–5 принципів, які допомагають ухвалювати рішення. Наприклад, «рух важливіший за стрільбу», «кожна небезпека телеграфується», «кооперація без голосового чату має бути можливою». Якщо функція суперечить pillars, її переглядають незалежно від того, скільки часу вже витрачено.

Mini-GDD для навчального проєкту доцільно обмежити 2-6 сторінками. Він включає логлайн, платформу, аудиторію, pillars, core loop, ключові механіки, умови успіху/невдачі, прогресію, короткий опис UI, арт-напрям, технічні ризики та scope першого прототипу. Документ має читатися за 10 хвилин і давати команді однакове бачення (табл. 1.13; рис. 1.18).

Таблиця 1.13

Мінімальний набір артефактів ігрового проєкту

Артефакт	**Призначення**	**Мінімальний зміст**	**Ознака готовності**
Vision + logline	узгодити сутність продукту	аудиторія, дія, цінність/емоція	зрозуміло за 20-30 секунд
Pillars	фільтр рішень	3-5 принципів із коротким поясненням	кожен принцип впливає на дизайн
Persona	прив’язати рішення до гравця	платформа, досвід, мотивація, бар’єри	є перевірювані наслідки для UX
Core loop	показати повторювану діяльність	3-7 кроків і зворотна петля	видно дію, нагороду й прогрес
Mini-GDD	об’єднати концепцію	механіки, правила, прогресія, UI, арт, ризики	команда може почати прототип
Backlog	керувати роботою	задача, пріоритет, власник, критерій готовності	є MVP і залежності
Asset list	контролювати контент	тип, формат, статус, власник, ліцензія	відсутні «невидимі» асети
Test plan	визначити якість	сценарії, метрики, пороги, середовище	тест повторюваний

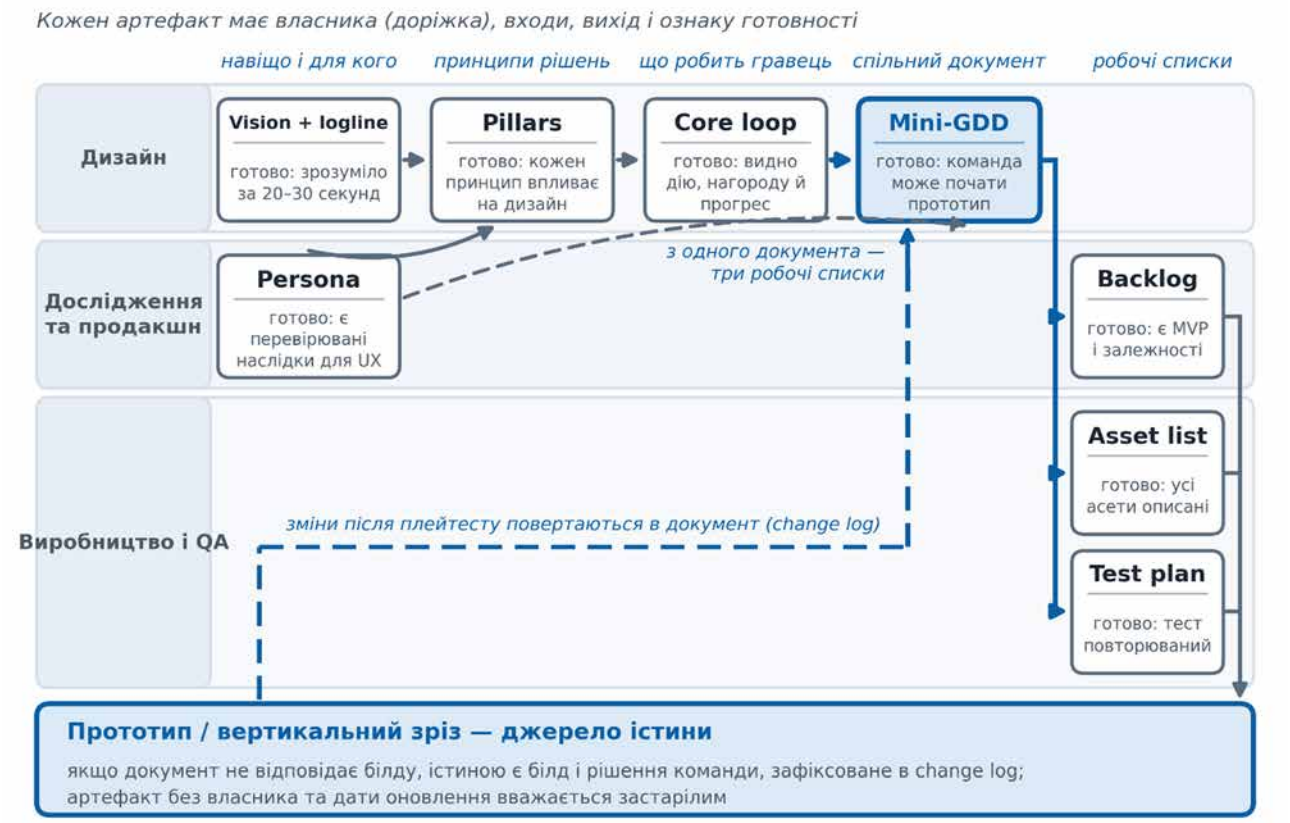


Рис. 1.18. Карта взаємозв’язку артефактів ігрового проєкту

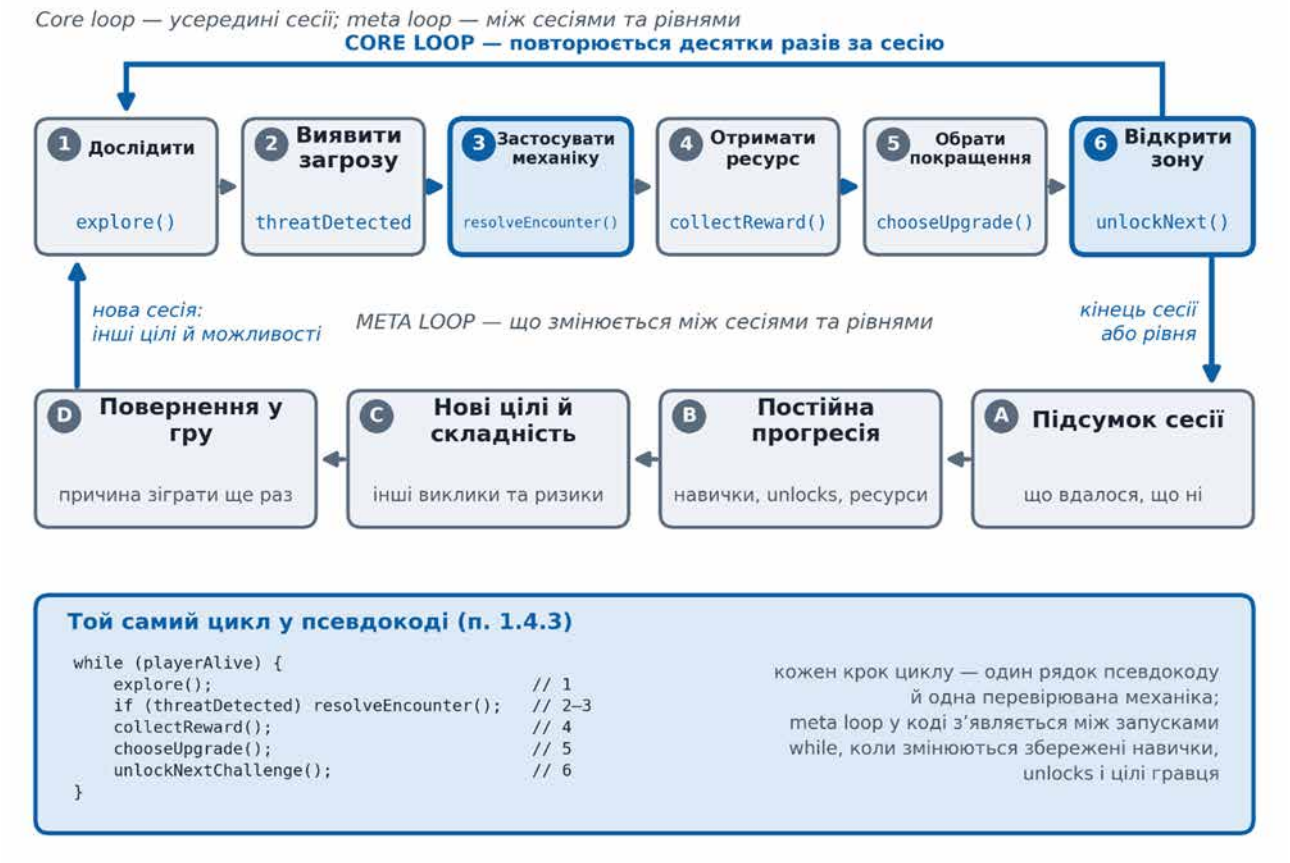


Рис. 1.19. Core loop та meta loop навчального ігрового проєкту

1.4.2 Порівняльний аналіз і референси

Конкурентний аналіз потрібен не для копіювання, а для розуміння очікувань жанру, стандартів якості та вільних ніш. Варто обирати щонайменше три референси: прямий конкурент, продукт із сильним окремим рішенням і контрастний приклад, що демонструє інший підхід.

Порівняння має бути симетричним: однакові критерії для всіх ігор. Зручними критеріями є платформа і контролер, core loop, тривалість сесії, модель прогресії, рівень складності, onboarding, UI/HUD, візуальна мова, монетизація, доступність, технічні вимоги та сильні/слабкі сторони.

Референс слід розкладати на рішення. Формулювання «зробити як у грі X» не дає команді чіткої інструкції. Корисніше зазначити: «використати контекстний prompt, що з’являється лише в радіусі взаємодії; підказка містить іконку поточного binding; після трьох успішних повторів prompt зникає».

Окремо варто фіксувати anti-reference - приклад, як не слід робити. Негативний досвід часто точніше виявляє дизайн-принцип: перевантажений HUD, нечіткі hitboxes, непослідовне керування, надто довга анімація без cancel window, відсутність feedback у разі помилки (табл. 1.14).

Таблиця 1.14

Шаблон порівняльного аналізу ігор

Критерій	Власна ідея	Референс 1	Референс 2	Референс 3
Основна fantasy				
Core loop				
Ключові механіки				
Структура рівнів				
Складність і прогресія				
UI/HUD та onboarding				
Візуальний/звуковий стиль				
Доступність				
Сильне рішення				
Ризик для власного проєкту				

1.4.3 Core loop, сценарій першого рівня та псевдокод

Core loop описує повторювану послідовність дій і винагород. Його доцільно формулювати дієсловами: дослідити - виявити загрозу - застосувати механіку - отримати ресурс - зробити вибір щодо покращення - відкрити нову зону. Meta loop пояснює, що змінюється між сесіями або рівнями.

Сценарій першого рівня є перевіркою всієї концепції. Він повинен показати ціль, базову механіку, перший безпечний приклад, варіацію, помірний

виклик, нагороду та завершення. Якщо перший рівень потребує пояснення десяти незалежних систем, scope слід скоротити.

Псевдокод корисний як спільна мова для дизайнера і програміста. Він не зобов'язаний бути синтаксично правильним, але має показувати стани, умови, зміну ресурсів і feedback. Діаграма станів доповнює псевдокод, коли механіка має переривання, cooldown, failure або залежить від анімації (рис. 1.19) (рис. 1.20).

```
while (playerAlive) {
 explore();
 if (threatDetected) resolveEncounter();
 collectReward();
 chooseUpgrade();
 unlockNextChallenge();
}
```

Практичне завдання «Аналіз двох ігор і mini-GDD»

1. Оберіть дві гри, у яких ви маєте щонайменше 5–10 годин практичного досвіду: одну з позитивним і одну з негативним користувацьким досвідом.
2. Для кожної гри визначте жанр, платформи, контролер, апаратні обмеження, основні механіки, core loop, правила, умови успіху/невдачі та тип винагороди.
3. Створіть 2 moodboard або reference board із підписами: середовище, персонажі, UI, палітра, освітлення, камера.
4. Підготуйте порівняльний аналіз на 150–250 слів: як механіки, loop, стиль і platform constraints формують досвід.
5. Запропонуйте 2–3 зміни для покращення гри та поясніть очікуваний вплив на гравця.
6. На основі висновків створіть власний mini-GDD: logline, persona, vision, 3–5 pillars, core loop, 5 механік, перший рівень, backlog із 10 задач та asset list із 10 позицій (табл. 1.15).

Таблиця 1.15

Критерії оцінювання практичної роботи

Критерій оцінювання	**Частка**
Глибина аналізу жанру, платформи та механік	20%
Точність core loop і правил	20%
Якість порівняння та аргументації	20%
Візуальна комунікація moodboard/діаграм	15%
Цілісність mini-GDD та відповідність scope	20%
Оформлення джерел і рефлексія	5%

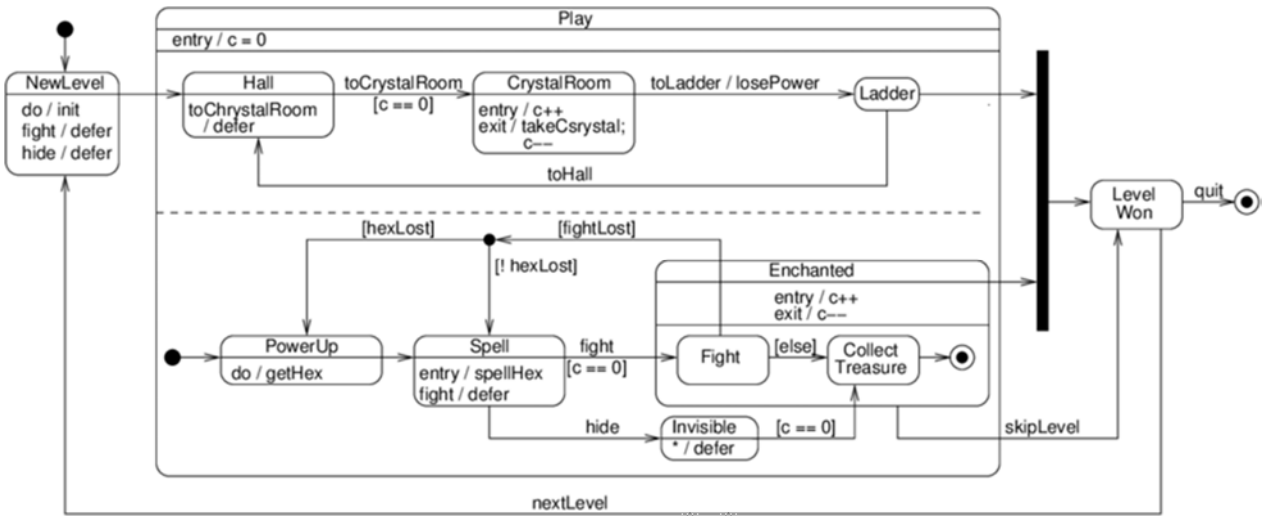


Рис. 1.20. Діаграма станів першої ключової механіки

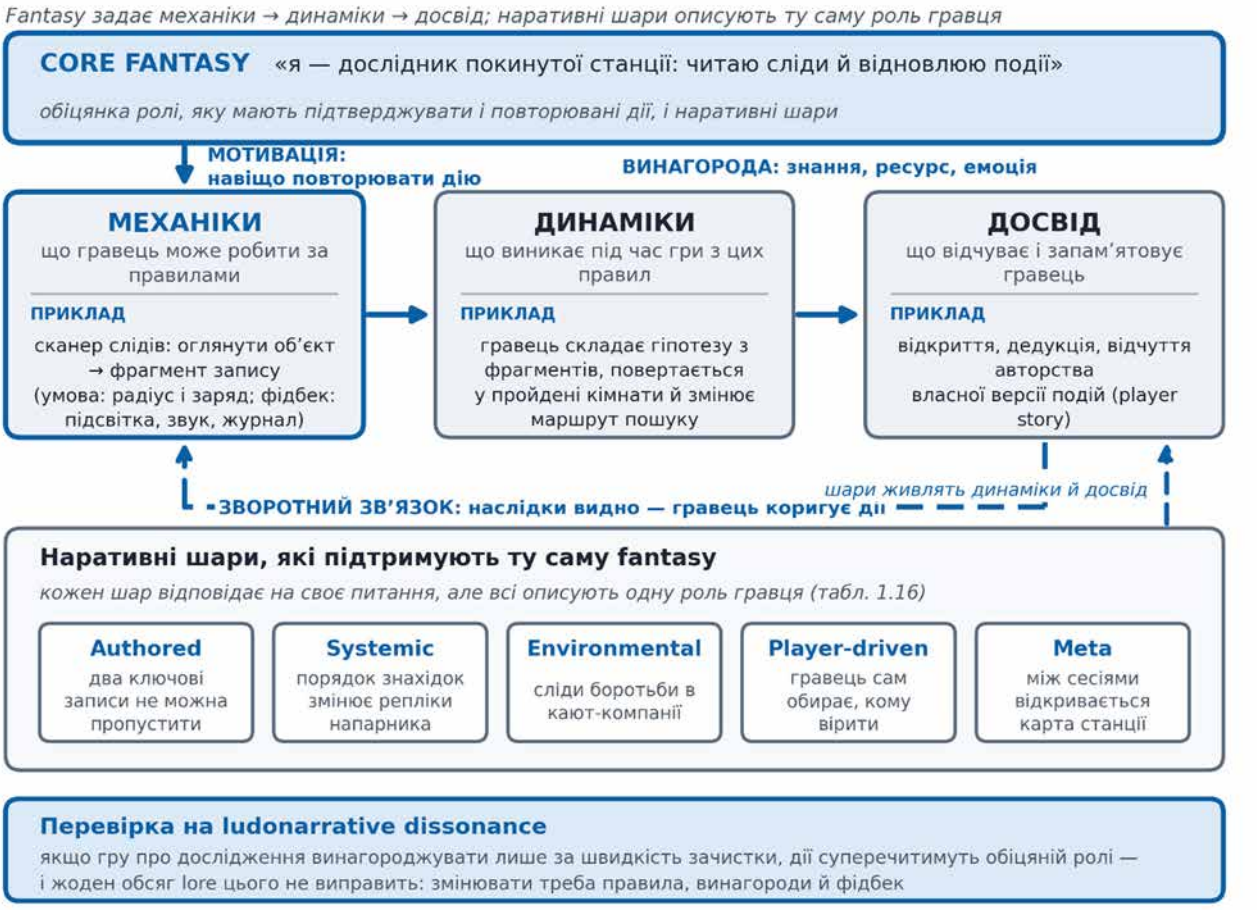


Рис. 1.21. Взаємозв'язок core fantasy, механік і наративних шарів

Тема 1.5. Наративний дизайн та інтерактивна драматургія

Мета теми: сформувати розуміння наративу як системи, що взаємодіє з механіками, простором, вибором і темпом; навчити створювати narrative pillars, beat sheet, environmental storytelling, діалогові стани та перевірний narrative prototype.

1.5.1 Наратив як система дій, наслідків і контексту

Ігровий наратив не обмежується текстом, кат-сценами або lore. Він виникає з того, що робить гравець, які правила бачить, у якому просторі рухається, які наслідки отримує і як інтерпретує події. Тому narrative design поєднує authored story, systemic story та player story.

Authored story містить заздалегідь підготовлені події, персонажів, діалоги та драматичну структуру. Systemic story виникає з взаємодії правил: конфлікт ресурсів, поведінка AI, випадкові події, комбінації механік. Player story - це особистий досвід проходження: невдалий план, несподіваний порятунок, вибір маршруту або рішення, яке гравець запам’ятав.

Наратив повинен підтримувати core fantasy. Якщо гра обіцяє роль дослідника, система має винагороджувати спостереження, інтерпретацію слідів і відкриття. Якщо fantasy пов’язана з командуванням, важливі рішення, відповідальність і зрозумілі наслідки. Сюжет, який суперечить повторюваним діям, створює ludonarrative dissonance (табл. 1.16; рис. 1.21).

Таблиця 1.16

Шари інтерактивного наративу

Шар наративу	Засіб	Питання для перевірки	Приклад артефакту
Authored	сцени, діалоги, місії	що має відбутися обов’язково?	beat sheet, script
Systemic	правила, AI, економіка, physics	які історії породжує система?	simulation map
Environmental	простір, props, світло, звук	що гравець зрозуміє без тексту?	storytelling pass
Player-driven	вибір, порядок, стиль проходження	що належить авторству гравця?	choice map
Meta-narrative	прогрес, колекції, повторні сесії	як довга петля формує сенс?	progression arc

1.5.2 Narrative pillars, тема та драматичне запитання

Narrative pillars конкретизують емоційний і смисловий напрям. Вони не повинні бути загальними словами на кшталт «епічність» або «цікавий сюжет». Корисний pillar задає рішення: «кожна перемога має видиму ціну», «світ розповідає через сліди, а не через експозицію», «діалоги короткі й залежать від останньої дії гравця».

Тема відповідає не на питання «про що сюжет», а на питання «яку ідею досліджує досвід». Драматичне запитання формує напругу: чи зможе персонаж зберегти довіру, використовуючи силу; чи варта безпека втрати автономії; чи можна відновити середовище без нового дисбалансу. Механіки повинні давати гравцеві можливість діяти в межах цього питання.

Для vertical slice достатньо одного narrative promise, одного конфлікту, двох-трьох beats і завершення, що демонструє наслідки. Великий lore без interaction не доводить якості наративного дизайну (табл. 1.17).

Таблиця 1.17

Каркас наративної концепції

Елемент	Формулювання	Перевірка
Core fantasy	ким є гравець і що регулярно робить	підтримується core loop
Theme	ідея, яку досліджує досвід	проявляється у виборах і наслідках
Dramatic question	центральна невизначеність	зберігає напругу до payoff
Narrative pillars	3-5 правил подачі	впливають на scene, dialogue, UI
Promise	що гравець очікує пережити	демонструється в перші хвилини
Payoff	емоційний або системний результат	є наслідком дій, а не випадковою сценою

1.5.3 Beat sheet, pacing і зв’язок із level design

Beat - це зміна інформації, цілі, емоції або стану. У грі beat може бути подією, encounter, відкриттям маршруту, зміною правила, вибором або короткою паузою. Beat sheet потрібно синхронізувати з gameplay pacing: після інтенсивного encounter гравцеві потрібен простір для інтерпретації наслідків.

Environmental storytelling працює через композицію, стан предметів, сліди дій, контраст «до/після», звук, освітлення та доступність простору. Важливо відрізняти clue від decoration. Clue змінює гіпотезу гравця або підказує можливу дію; decoration лише підтримує атмосферу.

Наративна інформація має мати hierarchy. Критичний факт має бути продубльований у двох каналах або інтегрований у потрібну дію. Опційний lore можна подавати через exploration. Перевантаження записками, довгими voice-over і статичними текстами часто маскує слабку постановку (табл. 1.18; рис. 1.22).

Таблиця 1.18

Приклад beat sheet для ігрової сцени

Beat	Gameplay state	Нова інформація / емоція	Простір і подача	Критерій
Hook	перша дія	обіцянка досвіду	сильний landmark / sound cue	цінність зрозуміла < 60 с
Setup	освоєння правила	контекст і мета	безпечна зона	гравець знає, що робити

Beat	Gameplay state	Нова інформація / емоція	Простір і подача	Критерій
Escalation	нове обмеження	ризик або конфлікт	стиснення простору / темпу	зміна рішення
Choice	альтернативи	цінності й наслідки	виразні affordances	вибір не декоративний
Payoff	результат	підтвердження або переворот	композиційний акцент	наслідок читається
Recovery	пауза / підготовка	рефлексія й нова ціль	відкритий простір	готовність до наступної петлі

1.5.4 Вибір, діалогові стани та наслідки

Meaningful choice змінює хоча б одну з величин: доступні дії, ресурси, відносини, інформацію, ризик, порядок подій або самоінтерпретацію гравця. Вибір між двома однаковими репліками, після якого система повертається до тієї самої сцени без видимого наслідку, є декоративним.

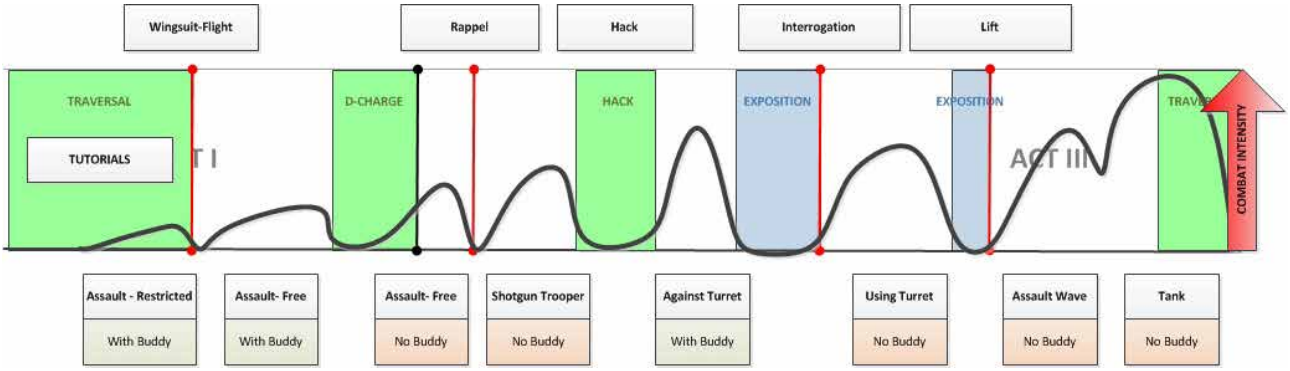
Діалог краще моделювати як систему станів і умов: хто говорить, що знає, які flags активні, які ресурси або репутація доступні, які наслідки створює репліка. Авторський текст зберігається окремо від логіки, має стабільні ID, підтримує локалізацію, subtitle timing і повторне використання.

Великі branching trees швидко комбінаційно ростуть. Тому застосовують foldback, hub-and-spoke, gated variants, systemic barks і consequence callbacks. Гравець повинен бачити не всі внутрішні flags, але розуміти причинність (табл. 1.19; рис. 1.23).

Таблиця 1.19

Патерни інтерактивного наративу

Патерн	Перевага	Ризик	Доцільне використання
Branching tree	сильна відмінність шляхів	висока вартість контенту	короткі ключові сцени
Foldback	контроль scope	ілюзія вибору	різні шляхи до спільного milestone
Hub-and-spoke	вільний порядок	слабкий темп	розслідування, exploration
State-gated variant	реактивність без великої гілки	складні flags	репліки, цілі, rewards
Systemic bark	дешевий контекстний feedback	повтори й шум	combat, navigation, companions
Delayed callback	відчуття наслідку	гравець може забути причину	довгі сесії, meta loop





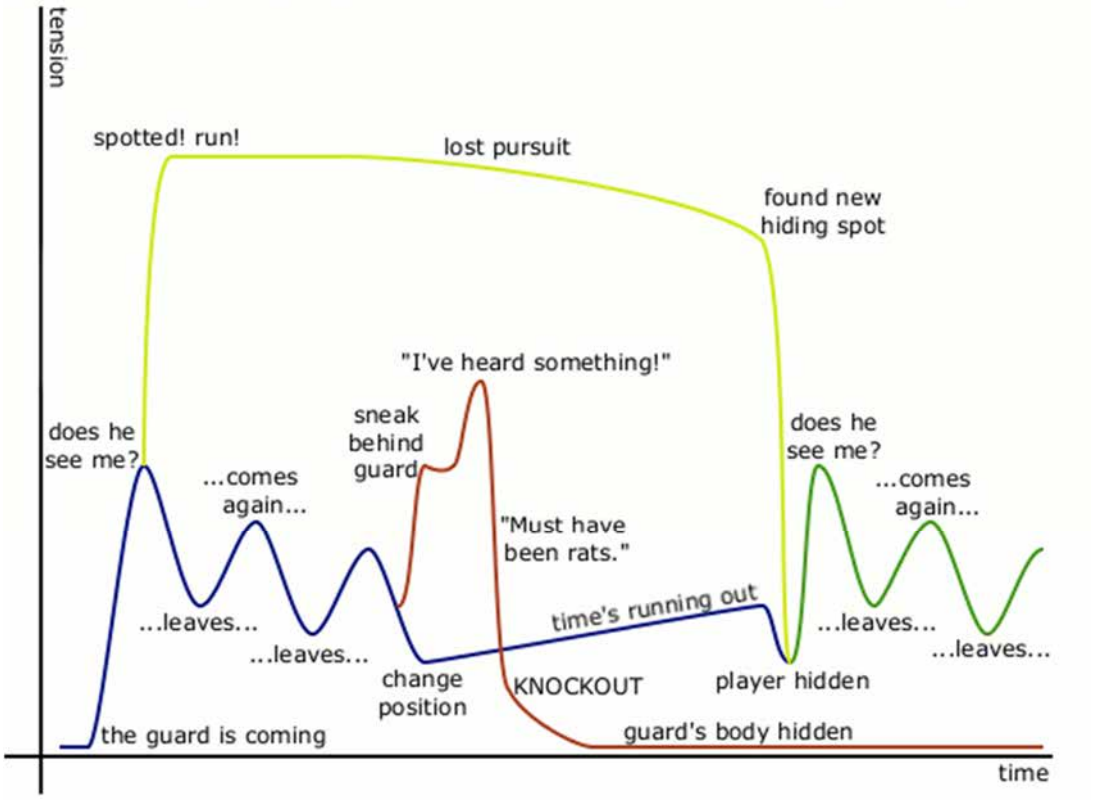
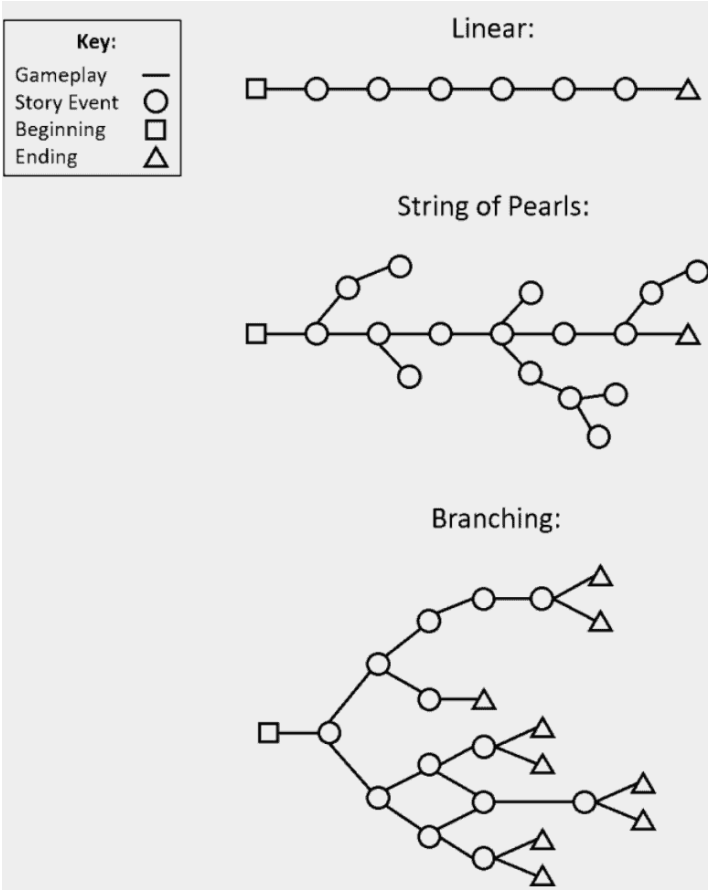


Рис. 1.22. Beat map рівня з gameplay та narrative pacing



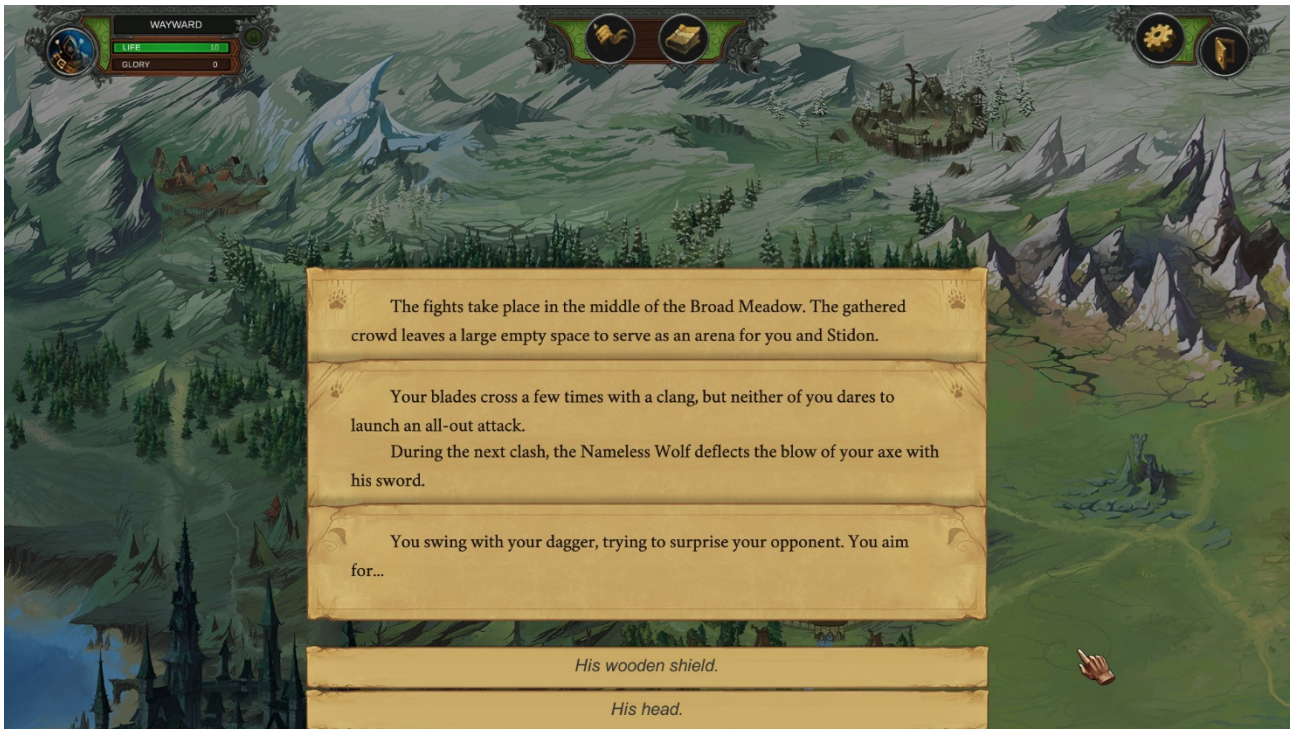


Рис. 1.23. State diagram діалогової сцени з умовами та наслідками

1.5.5 Наративний прототип і тестування

Наративний прототип може бути текстовою гілкою, storyboard, Twine-подібною схемою, greybox-сценою або playable dialogue. Його мета - перевірити зрозумілість мотивації, причинність, темп, помітність clues і відчуття agency, а не фінальну акторську подачу.

Під час тесту не варто запитувати лише: «Чи сподобався сюжет?». Корисні питання: що, на вашу думку, сталося; чому персонаж діяв так; яку ціль ви мали; які варіанти бачили; який наслідок пов’язали зі своїм рішенням; де ви втратили увагу; що очікуєте далі.

Результати фіксують як карту інтерпретацій: intended meaning, observed meaning, evidence, проблема, зміна. Непорозуміння інколи є бажаним mystery, але воно повинно зберігати цікавість, а не руйнувати мету.

Практичний кейс: narrative slice

Створіть 5-10-хвилинну greybox-сцену з hook, одним environmental clue, meaningful choice і delayed callback. Додайте beat sheet, state/flag map, список текстових ID, accessibility notes і протокол тестування для 5 учасників.

Питання для самоперевірки до розділу 1

1. Чим ігровий застосунок відрізняється від утилітарного програмного продукту?
2. Які три шари ігрового застосунку формують цілісний досвід?
3. Як платформа та контролер змінюють правила, UI, сесійність і performance budget?
4. У чому різниця між прототипом, MVP і vertical slice?
5. Які ролі потрібні для невеликої команди та хто відповідає за інтеграцію?
6. Що таке design pillar і як він допомагає контролювати scope?
7. Як сформувати persona без стереотипів і вигаданих припущень?
8. Які методи дослідження доцільні перед створенням першого прототипу?
9. Чим механіка відрізняється від контенту, функції та правила?
10. Як модель MDA пояснює шлях від правил до емоційного досвіду?
11. Які елементи має містити core loop?
12. Чим meta loop відрізняється від core loop?
13. Які параметри підлягають первинному балансуванню?
14. Що означає time-to-fun і як його перевірити?
15. Які документи є мінімально достатніми для команди з 3 осіб?
16. Як конкурентний аналіз допомагає відрізнити продукт, а не копіювати референси?
17. Що таке Definition of Done для механіки?
18. Як організувати цикл «гіпотеза - прототип - тест - рішення»?
19. Які ризики потрібно перевірити перед виробництвом великої кількості контенту?
20. Як перетворити нечітку ідею на вимірюваний план vertical slice?

Практичні завдання до розділу 1

Завдання 1. Аналіз двох ігор

Оберіть одну гру з позитивним і одну з негативним досвідом. Порівняйте жанр, платформу, контролер, механіки, core loop, правила, UI, стиль і причини власної оцінки. Додайте два moodboard та висновок 200-300 слів.

Завдання 2. Mini-GDD і core loop

Сформуйте logline, persona, 3-5 pillars, core/meta loop, п’ять механік, scope прототипу, десять задач backlog і десять асетів. Представте core loop як діаграму й псевдокод.

Завдання 3. Концептуальний плейтест

Створіть paper/digital prototype однієї механіки, визначте 3 гіпотези, проведіть щонайменше 5 коротких тестів, зафіксуйте метрики й рішення щодо наступної ітерації.

Висновки до розділу 1

1. Якісна концепція гри виникає не з обсягу опису, а з узгодженості аудиторії, core fantasy, правил, feedback і виробничих обмежень. Документація повинна підтримувати рішення та оновлюватися разом із прототипом.
2. Core loop є центром раннього етапу проєкту. Поки повторювана дія не читається, не дає очікуваного досвіду й не має керованих параметрів, масштабування контенту створює борг. Ітеративний pipeline дозволяє перевіряти найризикованіші припущення раніше й дешевше. (табл. 1.20)

Графічні матеріали до розділу 1

Таблиця 1.20

Перелік рекомендованих графічних матеріалів до розділу 1

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
1.1	Три рівні ігрового застосунку (правила/взаємодія/подача)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.2	Основні елементи ігрового дизайну	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.3	Ітеративний pipeline розробки гри	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
1.4	Ключові компоненти оформлення розробки ігор	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.5	Цикл дизайн-мислення для ігрового проєкту	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
1.6	Профіль гравця та контекст ігрової сесії	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.7	Зв’язок «спостереження - проблема - гіпотеза - метрика»	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
1.8	Mind map функцій і ризиків ігрової концепції	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.9	Storyboard першої ігрової сесії	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.10	Головні елементи ігор	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.11	Дизайн Ігрового циклу	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
1.12	Базова логіка механіки: будь-яка механіка описується як послідовність	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
1.13	Крива складності: навчання -> виклик -> майстерність	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.14	Логіка вкладеності циклів ігрового дизайну	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
1.15	Приклад state machine для атаки	Використати UML-подібну діаграму станів із переходами, умовами та пріоритетами; окремо позначити стани помилки та переривання.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
1.16	Приклад часової структури атаки гравця або ШІ	Подати серію з 3-5 кадрів anticipation – impact – recovery, додати часову шкалу та короткі примітки щодо читабельності й бюджету.
1.17	Фідбек: анімація удару, звук контакту, ефект попадання	Подати серію з 3-5 кадрів anticipation – impact – recovery, додати часову шкалу та короткі примітки щодо читабельності й бюджету.
1.18	Карта взаємозв’язку артефактів ігрового проєкту	Подати як swimlane-діаграму відповідальностей або карту артефактів із зазначенням власника, входів, виходів і критеріїв готовності.
1.19	Core loop та meta loop навчального ігрового проєкту	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
1.20	Діаграма станів першої ключової механіки	Використати UML-подібну діаграму станів із переходами, умовами та пріоритетами; окремо позначити стани помилки та переривання.
1.21	Взаємозв’язок core fantasy, механік і наративних шарів	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.22	Beat map рівня з gameplay та narrative pacing	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
1.23	State diagram діалогової сцени з умовами та наслідками	Використати UML-подібну діаграму станів із переходами, умовами та пріоритетами; окремо позначити стани помилки та переривання.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІГРОВОГО СВІТУ ТА ВІЗУАЛЬНА СИСТЕМА

Мета розділу: розділ присвячений простору, композиції, UI/HUD і візуальним технологіям, які перетворюють правила на читабельний та емоційно виразний світ. Рівень, камера, матеріали, світло, VFX і графічний інтерфейс розглядаються як узгоджена система комунікації.

Візуальна якість не дорівнює максимальній деталізації. Для ігрового застосунку важливі hierarchy, silhouette, contrast, signposting, стабільність руху та відповідність технічному бюджету. Кожен ефект має пояснювати стан, напрям, небезпеку, результат або настрій.

Тематика побудована від blockout і маршруту до art pipeline, UI, PBR, шейдерів, VFX, освітлення та постобробки. Всі рішення перевіряються в контексті цільової камери, gameplay та performance.

Після вивчення розділу здобувач набуває компетентності:

- створювати blockout, critical path, beat map і систему signposting;
- планувати арт-пайплайн, naming, import settings і visual budgets;
- проєктувати HUD, меню, diegetic UI, feedback states та accessibility;
- пояснювати PBR, матеріали, текстури, шейдерні параметри й стилізацію;
- будувати VFX за фазами anticipation, impact і recovery;
- налаштовувати освітлення, exposure, fog, color grading і readability pass;
- оцінювати overdraw, shader complexity, LOD і продуктивність візуальних систем.

Ключові терміни: level design, blockout, critical path, signposting, art pipeline, HUD, diegetic UI, PBR, shader, VFX, lighting, post-processing.

Тема 2.1. Дизайн рівнів і навігація: blockout, темп, прогресія та метрики

Мета теми: сформувати у студентів системне розуміння дизайну рівнів як інженерно-дизайнерської дисципліни, що поєднує просторове мислення, геймплейні системи та UX-навігацію. Навчити базовому процесу blockout/greybox, побудові темпу і прогресії, проєктуванню ризику/нагороди та застосуванню метрик і плейтестів для ітераційного покращення рівня.

Очікувані результати навчання

- вміти пояснити відмінність між level design, encounter design, mission design та world design; визначити роль навігації й читабельності;
- побудувати blockout (greybox) малого рівня у рушії (Unity/UE5) та оформити базовий LDD (Level Design Document);
- спроєктувати критичний шлях (critical path), альтернативні маршрути та точки інтересу (POI) з використанням принципів signposting;
- керувати темпом (pacing) через ритм викликів/пауз, інтенсивність, «beats» та динаміку ресурсів;

- задати прогресію складності та навчання механікам (tutorialization), використовуючи «safe->challenge->mastery»;
- застосувати ризик/нагороду (risk/reward) для опційних зон і виборів гравця; збалансувати нагороди;
- визначити й зібрати мінімальний набір метрик рівня (час проходження, смерті, фрустраційні точки, траєкторії руху) та зробити 1–2 ітерації після плейтесту.

Ключові терміни: Level design, blockout/greybox, critical path, optional path, encounter, gating, signposting, readability, landmarks, affordances, line of sight (LoS), choke point, funneling, pacing, beat chart, difficulty curve, onboarding, risk/reward, resource economy, telegraphing, metrics, heatmap, telemetry, playtest, iteration.

2.1.1 Роль дизайну рівнів у створенні ігрового досвіду

Дизайн рівнів (Level Design) - це проєктування ігрового простору, який керує поведінкою гравця, підсилює механіки та забезпечує послідовний досвід (темп, складність, емоції). Рівень - це не лише «красива локація», а функціональна структура: траєкторії, видимість, правила проходження, точки ризику та винагороди, а також навігаційні сигнали.

У сучасній розробці часто розділяють суміжні підзадачі:

• World Design - логіка та структура світу (біоми, маршрути, зв’язки локацій).

• Level Design - простір окремого рівня/сцени, темп, розміщення контенту.

• Encounter Design - постановка бою/головоломки/ситуації (вороги, умови, ресурси).

• Mission/Quest Design - цілі, тригери, квести, сценарії.

Навігація і читабельність - «UX каркас» рівня: гравець має розуміти куди йти, що є важливим, які дії можливі, і який наступний крок. Якщо навігація провалена, навіть цікава механіка не спрацює (рис. 2.1).

2.1.2 Blockout (greybox): швидке створення прототипів (прототипування) простору

Blockout (або greybox) – це «чорнова» версія рівня з простих примітивів (куби/площини), без фінального арту. Мета blockout – перевірити функцію рівня: масштаби, читабельність, темп, видимість, маршрути, стрибки, бої, навігацію.

Принцип: спочатку – геометрія та геймплей; потім – декор і деталізація.

Типовий цикл роботи:

1) Ціль рівня (емоція, механіка, навчання, сюжетний «beat»).
2) План/карта зверху (2D sketch): critical path та optional paths.
3) Blockout у рушії: кімнати/коридори/арени, висоти, укриття.
4) Навігаційні сигнали: грубі landmark-и, світлові плями, «рамки» входів.
5) Плейтест 10–15 хв.: рух, читабельність, помилки.
6) Ітерація: скорочення, розширення, перенесення POI, зміна LoS.

Важливо: blockout не має бути «ідеальним». Його сила – швидкість змін і перевірка гіпотез (рис. 2.2) (табл. 2.1).

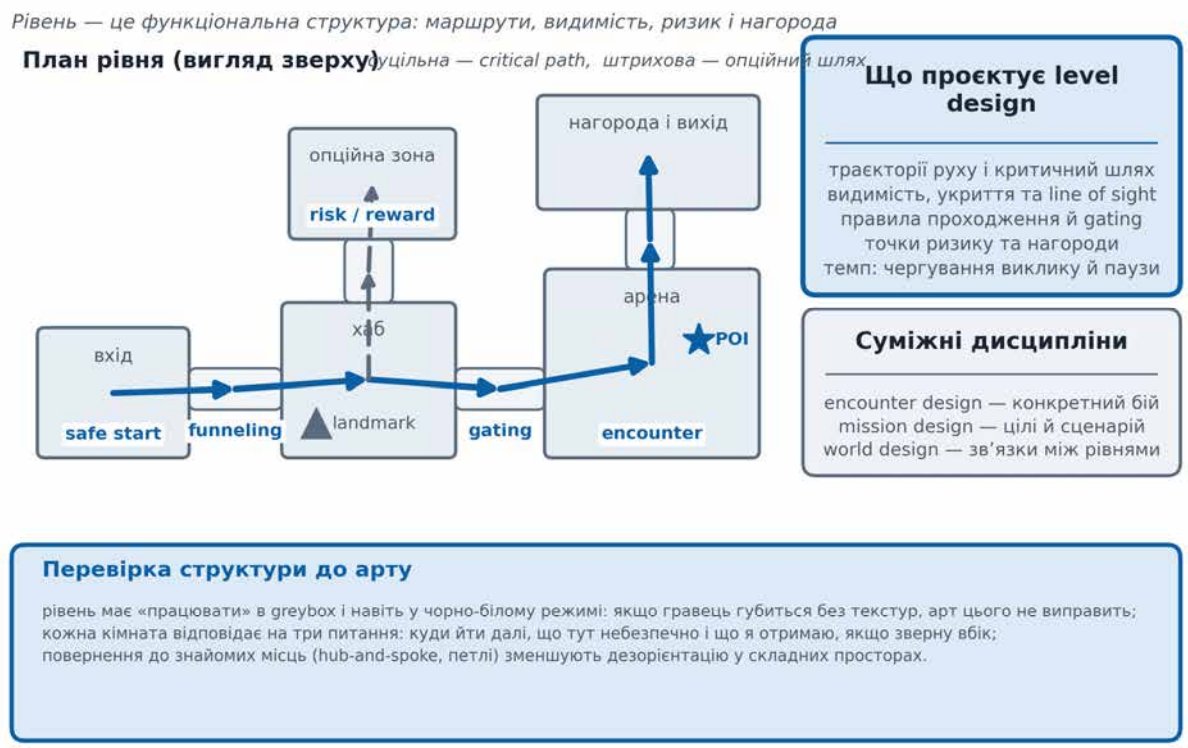


Рис. 2.1. Структура дизайну сучасного рівня



Рис. 2.2. Приклад blockout дизайну сучасного рівня

Таблиця 2.1

Параметри масштабу (scale) для швидкої перевірки blockout

Елемент	Що перевіряємо	Типові помилки	Швидкий тест
Прохід/ коридор	ширина, огляд, темп руху	занадто вузько/широко; «тунель» без орієнтирів	пробігти 3 рази без зупинок; чи губиться гравець
Дверний проріз	читабельність входу/виходу	вхід не видно; плутається з декором	зробити 1 landmark та світло; повторити плейтест
Арена/ кімната	LoS, укриття, дистанції бою	нема укриттів; «мертві зони»; хаос	порахувати 3 позиції ворога і 2 укриття; TTK/смерті
Вертикаль	висоти, стрибки, ризик падіння	стрибок неочевидний; невірні дистанції	поставити маркери; перевіри-ти 5 повторів без фрустрації

2.1.3 Навігація та читабельність: як «вести» гравця без тексту

Навігація в рівні – це система підказок, що допомагає гравцю будувати ментальну карту та ухвалювати рішення. Головна мета – зменшити когнітивне навантаження: гравець думає про виклик/стратегію, а не про «де вихід».

Ключові принципи мають бути наступними

Signposting – навмисні сигнали (світло, колір, контраст, рух, звук), що вказують напрям

Landmarks – помітні орієнтири (вежа, дерево, статуя, яскравий банер), що «якорять» простір

Affordances – підказка дії через форму: сходи «кажуть»: піднімайся; поручні «кажуть»: стій тут

Funneling – м'яке спрямування через геометрію (звуження, рамки, перспективні лінії)

Читабельність силуету – вороги/об'єкти мають швидко впізнаватися (особливо у бою)

Практичні рекомендації: рівень має «працювати» навіть у чорно-білому режимі та без текстур: якщо гравець губиться у greybox – арт це не врятує. Для складних просторів використовуйте «петлі» і повернення до знайомих місць (hub-and-spoke), щоб зменшити ризик дезорієнтації (рис. 2.3).

2.1.4 Темп (pacing): ритм викликів, пауз і «beats»

Темп (pacing) - це керування інтенсивністю подій у часі. Навіть у «повільних» іграх темп існує: це чергування мікрорішень, відкриттів, помилок і винагород.

Поняття «beat» - це одиниця події, що змінює стан гравця: новий ворог, коротка головоломка, ресурс, сюжетний момент, панорама, checkpoint. Зручний підхід - beat chart: таблиця/графік, де ви плануєте послідовність «тихо -> напруга -> пік -> розрядка».

Це особливо важливо на рівнях із боєм: після інтенсивної сутички гравцю потрібен простір, щоб «переварити» подію, зібрати ресурси, оглянутися.

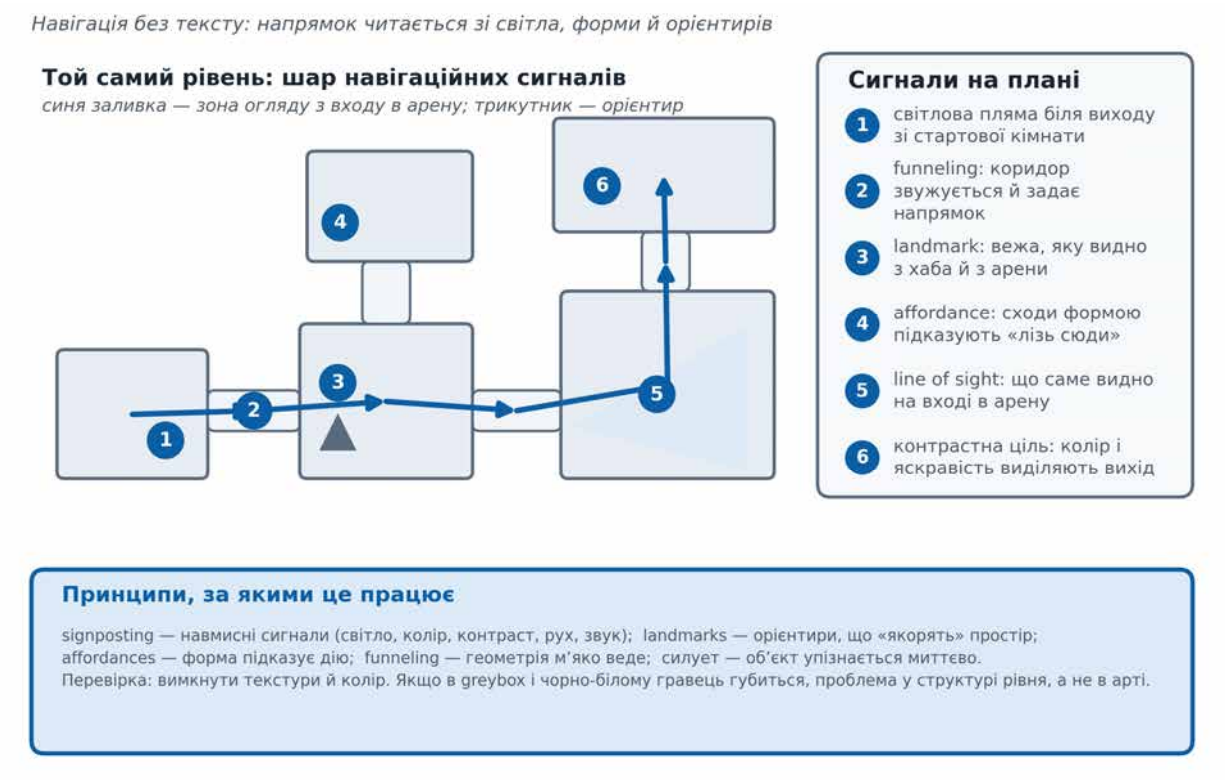


Рис. 2.3. Приклад дизайну шляху гравця для сучасного рівня

Інструменти керування темпом є наступними

Розмір і форма просторів (вузько – швидко/напружено; широко – оглядово/пауза).

LoS (лінії огляду): заздалегідь виявлена загроза – підготовка; раптовість – шок.

Ресурси (HP/боєприпаси/таймери): дефіцит підсилює напругу.

Контрольні точки та безпечні зони.

Сюжетні/візуальні паузи: панорама, NPC, ambient звук.

Головний ризик - «монотонність»: однакова інтенсивність протягом 10 хв призводить до втоми або нудьги (табл. 2.2).

Таблиця 2.2

Спрощений beat chart для 8-10 хвилин рівня (приклад)

Beat #	**Подія/ситуація**	**Інтенсивність (1-5)**	**Навігаційний сигнал**	**Мета/навчання**
1	Вхід та landmark та коротка пауза	1	світло на двері	орієнтація, контекст
2	Мікро-перешкода/стрибок	2	лінія/рамка проходу	перевірка керування
3	Перший слабкий ворог	3	звук та видимий спавн	навчити telegraphing
4	Лут/ресурс та безпечна зона	1	контрастний контейнер	винагорода, відпочинок
5	Арена (2-3 вороги)	4	фокус-об'єкт в центрі	позиціонування/укриття

Beat #	Подія/ситуація	Інтенсивність (1-5)	Навігаційний сигнал	Мета/навчання
6	Коридор-розрядка та панорама	2	landmark вдалині	переорієнтація
7	Міні-бос/пазл	5	чітка «сцена» та музика	пік, перевірка навички
8	Вихід та нагорода	1	яскравий вихід	закріплення, завершення

2.1.5 Прогресія: навчання, складність та «крива майстерності»

Прогресія в рівні – це не лише «складніше з часом», а й кероване навчання. Найпоширеніша структура: показати (safe) -> перевірити (challenge) -> ускладнити і комбінувати (mastery).

Прийоми tutorialization (навчання без підручника):

- Безпечна демонстрація механіки: гравець бачить, що можна зробити без штрафів.
- Єдина нова змінна за раз: не вводьте одразу нового ворога, нову пастку й нове правило.
- Ясний фідбек: звук/анімація/частинки, що підтверджують правильність дії.
- Повтор у новому контексті: та сама механіка, але в іншому просторі або з іншим таймінгом.

Крива складності повинна враховувати психологію: гравець легше приймає зростання виклику, якщо в нього є відчуття контролю і справедливості (readability та telegraphing та відновлення після помилки) (рис. 2.4).



Рис. 2.4. Приклад дизайну візуальної прогресії шляху гравця

2.1.6 Ризик/нагорода: опційні шляхи та дизайн виборів

Ризик/нагорода (risk/reward) створює цікаві рішення: гравець добровільно обирає складніший шлях заради ресурсу, сюжету або «брендингу майстерності».

Форми ризику можуть бути такі:

- ***Бойовий:*** більше ворогів/менше укриттів.
- ***Платформінг:*** стрибок з падінням.
- ***Часовий:*** таймер або переслідування.
- ***Ресурсний:*** витрата ключового ресурсу (HP/боєприпаси/ключ).

Форми винагороди наступні

- **Функціональна:** апгрейд, зброя, лікування.
- **Інформаційна:** лор, підказка, короткий кат-сцен.
- **Емоційна:** панорама, «секрет», колекційний предмет.

Правило гарного дизайну: нагорода має відповідати ризику.

Якщо секрет складний, але дає дрібницю – гравці перестануть досліджувати. Якщо ризик мінімальний, але нагорода велика – ламається баланс прогресії. Зручний інструмент – матриця ризик/нагорода (4 квадранти), яка допомагає швидко оцінити чи є вибір чесним і чи не створює «обов'язковий» маршрут (табл. 2.3).

Таблиця 2.3

Матриця ризик/нагорода (приклад інтерпретації)

Квадрант	Що це означає	Рішення дизайнера
Низький ризик / Низька нагорода	стандартний шлях, «базова» цінність	норма; не перевантажувати
Низький ризик / Висока нагорода	маршрут стає обов'язковим	зменшити нагороду або додати ризик/умову
Високий ризик / Низька нагорода	фрустрація, «не варто»	підсилити нагороду або спростити ризик
Високий ризик / Висока нагорода	опційний виклик для майстерності	ідеально для секретів/опційних боїв

2.1.7 Метрики рівня: що вимірювати і як робити висновки

Метрики – це спосіб перейти від суб'єктивного «норм/не норм» до гіпотез, які можна перевірити. У великих проєктах використовують телеметрію та heatmaps, але для навчальної роботи достатньо простих таблиць і протоколу тестування.

Мінімальний набір метрик для рівня має містити:

- Time-to-complete (TTC): час проходження рівня або сегмента.
- Deaths per segment: де і скільки разів гравці помирають.
- Fail reason: причина помилки (не побачив, не зрозумів, не встиг).
- Path choice: який маршрут обрали (critical/optional).
- Confusion markers: місця, де гравець зупиняється, крутиться або шукає.
- Resource delta: скільки HP/боєприпасів витрачено/знайдено.

Інтерпретація: метрики мають відповідати на поставлені запитання. Наприклад: «Чи працює signposting?» -> дивимось зупинки та помилки навігації. «Чи справедливий бій?» -> дивимось смерті, час бою, витрату ресурсів.

Головне – робити ітерації. Навіть 1 ітерація після 3 плейтестів часто підвищує якість на рівень вище, ніж додавання нового контенту (табл. 2.4).

Таблиця 2.4

Шаблон протоколу плейтесту (мінімальний)

Тестер	**TTC (хв)**	**Смерті (де?)**	**Зупинки/плутанина**	**Опційні зони (так/ні)**	**Коментар (1-2 речення)**
Гравець A	__	__	__	__	__
Гравець B	__	__	__	__	__
Гравець C	__	__	__	__	__

2.1.8 Документація рівня: LDD, flow, чек-листи

Зазначимо, що навіть у маленьких командах документація економить час. Для рівнів достатньо «легкого» LDD на 1-2 сторінки, який містить:

• Мета рівня (емоція, навчання, сюжетний beat).
• Список ключових механік, які використовуються.
• Топ-даун схема: critical path та optional.
• Encounter list (послідовність боїв/пазлів).
• Параметри (ресурси, складність, нагороди).
• Метрики і план плейтесту.

Чек-лист «читабельності» (швидко):

• Чи видно ціль/вихід з першої значущої позиції?
• Чи є 1-2 landmarks на сегмент?
• Чи зрозуміло, де безпечно, а де небезпечно (telegraphing)?
• Чи є короткі паузи після піків інтенсивності?
• Чи відповідає нагорода ризику в опційних зонах?

Практичний кейс до теми (120-180 хв): «Blockout рівня та плейтест та ітерація»

Завдання: створити у рушії (Unity або Unreal Engine 5) невеликий рівень (8–12 хв проходження), який демонструє керування навігацією, темпом та ризиком/нагородою.

Вимоги до рівня наступні та містять:

- критичний шлях від старту до виходу (мінімум 3 сегменти);
- 2 опційні відгалуження з різним ризиком і нагородою;
- щонайменше 1 «пік» інтенсивності (бій або складна перешкода);
- 1-2 паузи/зони відпочинку;
- принаймні 2 landmarks і мінімум 3 елементи signposting світло/контраст/рамки/звук).

Вимоги для здачі завдання:

1 сторінка LDD (PDF або Google Doc).

Топ-даун схема (скетч або діаграма з critical/optional path).

Заповнений протокол плейтесту на 3 тестерах (Таблиця 3.4).

Список змін після плейтесту (мінімум 5 правок) і коротке обґрунтування.

Підказка: почніть з blockout і 1 простого «біта». Коли він працює - масштабуйте. Не витрачайте час на арт, доки не пройдете 3 плейтести.

Критерії оцінювання:

Читабельність і навігація (гравці не губляться).

Темп (є ритм, піки і паузи).

Ризик/нагорода (опційні зони мають сенс).

Якість ітерації (після плейтесту рівень покращується).

Питання для самоперевірки

1. Поясніть різницю між level design і world design. Які артефакти створює level designer?
2. Що таке blockout/greybox і чому його потрібно робити до фінального арту?
3. Назвіть 5 прийомів signposting і наведіть приклади того, як ними вести гравця.
4. Що таке pacing і як beat chart допомагає уникати монотонності?
5. Опишіть шаблон прогресії Safe->Challenge->Mastery на прикладі однієї механіки.
6. Які метрики ви зберете для перевірки: (а) навігації, (б) складності, (в) ризик/нагорода?
7. Наведіть приклад ситуації «низький ризик/висока нагорода» і як ви б виправили дизайн.

Рекомендовані програми та сервіси

1. Unity (LTS) або Unreal Engine 5 – blockout і прототипування рівня. ProBuilder (Unity) або Modeling Mode (UE5) – швидке створення геометрії.
2. Blender – для простих модульних форм (опційно).
3. Miro / FigJam / draw.io – схеми level flow, critical path, beat chart.
4. Google Sheets / Excel – таблиці метрик, баланс ресурсів.
5. Trello / Notion – список задач і фіксація ітерацій.
6. OBS або будь-який screen recorder – запис плейтестів (дуже корисно для аналізу).

Рекомендовані джерела та матеріали

1. Level Design Book (безкоштовний довідник): https://book.leveldesignbook.com/
2. GDC Vault (level design/pacing talks): https://www.gdcvault.com/
3. World of Level Design (практичні гайди): https://www.worldofleveldesign.com/
4. An Architectural Approach to Level Design (C. Totten) – книга.

5. Level Up! The Guide to Great Video Game Design (S. Rogers) – книга.
6. The Art of Game Design (J. Schell) – про дизайн як систему рішень.
7. Game Accessibility Guidelines (навігація/читабельність): https://gameaccessibilityguidelines.com/

Тема 2.2. Візуальний дизайн ігрових застосунків: арт-пайплайн, UI/HUD, VFX та анімація

Мета теми: сформувати у студентів цілісне розуміння ролі візуального дизайну у виробництві ігрових застосунків та надати практичні навички побудови арт-пайплайну, створення UI/HUD, базових VFX і анімації з урахуванням стилістичної цілісності, зручності для гравця, технічних обмежень і продуктивності (performance budgets)

Очікувані результати навчання

- вміти пояснювати складові візуального стилю гри (art direction) та роль «style guide»;
- описувати арт-пайплайн від концепту до інтеграції в рушій (engine-ready assets);
- проєктувати UI/HUD під конкретний жанр та платформу з урахуванням UX, читабельності й доступності;
- планувати та оцінювати VFX і анімацію як системи фідбеку (game feel) та оптимізації;
- оформлювати базову документацію: Art Bible / UI Spec / VFX list / Animation list; застосовувати підхід «Iterate with constraints»: тестування -> правки -> повторне тестування.

Ключові терміни: Art Direction, Style Guide (Art Bible), Moodboard, Reference Board, Key Art, Visual Hierarchy, Readability, Contrast, UI Kit, HUD, Diegetic UI, Non-diegetic UI, Feedback Loop, VFX (Particles, Decals, Shaders), Animation Principles, Rig, Skeleton, State Machine, Blend Tree, Timing, Anticipation, Squash & Stretch, Silhouette, LOD, Texture Atlas, Draw Calls, Batching, Overdraw, Frame Budget, Accessibility (Color Blindness, Text Scaling).

2.2.1 Візуальний дизайн як «мова» гри: що він вирішує

Візуальний дизайн у грі – це не «красиво/некрасиво», а функціональна мова, яка одночасно виконує наступні функції

• пояснює правила (що можна/не можна робити),
• показує пріоритети (що важливе прямо зараз),
• керує увагою (композиція, контраст, рух),
• формує емоцію і жанр (tone, mood, темп),
• підтримує інтерфейс і читабельність (UX-цілі).

Ключова ідея: візуальна система – це частина ігрових систем, а не «обгортка». Візуальний дизайн має відповідати геймплею: швидким бойовим іграм потрібна максимальна читабельність і різкі контрасти; атмосферним

пригодам – сильні силуети, глибина, «м'які» переходи; казуальним – прості форми, зрозумілий UI і мінімум шуму (рис. 2.5).

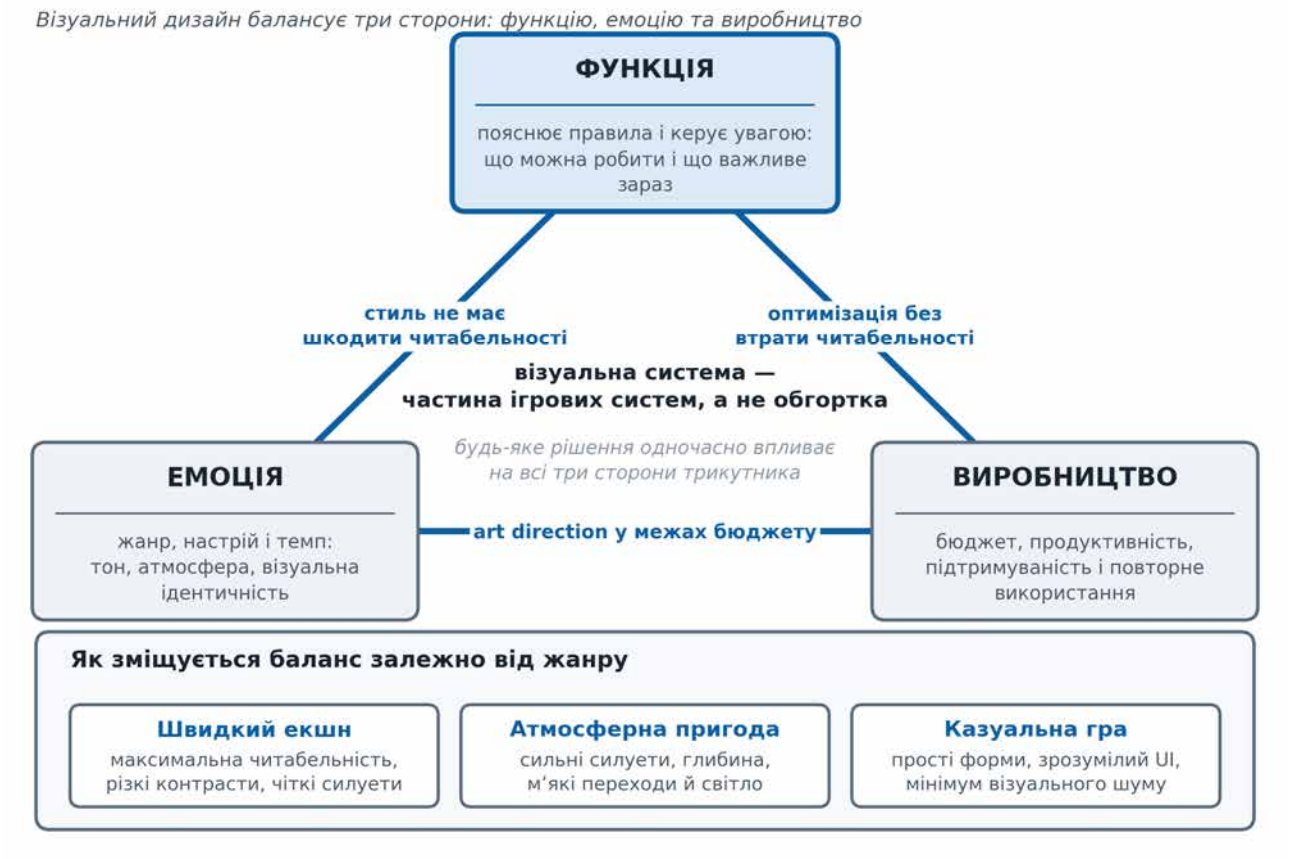


Рис. 2.5. «Трикутник» візуального дизайну гри

2.2.2 Art Direction і Style Guide: як забезпечити цілісність

Art direction – це набір свідомих рішень: форма, колір, матеріали, освітлення, типографіка, композиція, а також те, чого ми не робимо. Найпоширеніша проблема студентських проєктів: «змішали 5 стилів», бо немає єдиного правила.

Мінімальний Style Guide (Art Bible) для студентського проєкту

Style Guide у малому проєкті може бути 3-6 сторінок, але має містити

- Moodboard (10-20 референсів: форма, колір, світло);
- Palette (основні/допоміжні кольори та правила контрасту);
- Typography (шрифти і правила використання);
- Shape language (силуети: округле/кутове, товсті/тонкі лінії);
- Materials (як виглядають метал/камінь/тканина у вашому стилі);
- UI tone (плоский/об'ємний, неон/пастель, «пластик/скло» тощо);
- Do/Don't (3–5 прикладів «так/ні»).

Референси потрібні для: швидкого пояснення стилю команді; уникнення випадкових рішень; контролю якості під час ітерацій

Важливо: не «копіювати картинку», а копіювати принцип: ритм, силует, колірні відношення, правила освітлення (рис. 2.6).



Рис. 2.6. Процес візуального дизайну гри

2.2.3 Арт-пайплайн: від концепту до інтеграції в рушій

Арт-пайплайн - це маршрут активу (asset) від задуму до робочого стану в рушії. Без пайплайну ви отримуєте: хаос у файлах, різні формати, конфлікти версій, «не те світло», «не той шрифт», «VFX лагує».

Основні етапи пайплайну є наступними та мають містити

- Концепт/референси (що робимо і чому).
- Блокінг (прості форми для перевірки пропорцій).
- Production asset (2D/3D, текстури, UI екрани, VFX).
- Технічна підготовка (формати, імпорт, оптимізація).
- Інтеграція (підключення у сцені/UI-системі).
- QA та ітерації (читабельність, FPS, баги, узгодження стилю).

Ролі та відповідальності (навіть у малій команді)

Навіть якщо один студент робить все, корисно мислити ролями: арт-директор, UI дизайнер, VFX-артіст, аніматор, тех-арт (інтеграція/оптимізація) (табл. 2.5; 2.6).

Таблиця 2.5

Приклад пайплайн-ролей і артефактів

Роль/ функція	**Вхід**	**Вихід (deliverable)**	**Критерій «готово»**
Art Direction	референси, жанр	Style Guide	є палітра, шрифт, правила Do/Don't
2D/3D Art	style guide	assets (2D/3D)	правильні масштаби, неймінг, чисті файли

Роль/ функція	Вхід	Вихід (deliverable)	Критерій «готово»
UI/UX	сценарії	UI kit, макети екранів	читабельність, контраст, відповідність жанру
VFX	список подій/фідбеку	VFX список та ефекти	бюджет, читабельність, мінімум шуму
Animation	список станів	кліпи, машин станіва	плавні переходи, timing, відсутні «ривки»
Tech Art	assets	налаштування імпорту/LOD	стабільний FPS, контроль draw calls

Файлова структура, неймінг, версійність: Базове правило: імена мають пояснювати призначення

Таблиця 2.6

Мінімальний стандарт неймінгу

Тип	Шаблон	Приклад
UI екрани	UI_Screen_Name_v01	UI_Screen_Inventory_v03
Іконки	UI_Icon_Action_Size	UI_Icon_Attack_64
3D модель	SM_ObjectName	SM_Crate_A
Матеріал	M_ObjectName	M_Crate_Wood
Текстура	T_Object_Channel_Size	T_Crate_BaseColor_2048
VFX	VFX_EventName	VFX_Hit_Sparks
Анімація	AN_Character_State	AN_Hero_Run

Рекомендація: використовувати Git LFS або Perforce (для великих бінарних файлів), або дисциплінований Google Drive та правила версій (але краще VCS).

2.2.4 UI/HUD: принципи, читабельність, доступність

UI/HUD у грі – це система, яка повинна бути

читабельною (readability),
передбачуваною (consistency),
швидкою (гравець «зчитує» за 200-500 мс),
доступною (контраст, масштабування, дальтонізм),
не перевантаженою (information diet).

Ієрархія і контраст

Показуйте гравцю лише те, що потрібно «зараз»

• критичне (HP, загроза, ціль),
• контекстне (підказка, взаємодія),
• довідкове (ресурси, час) (табл. 2.7).

Таблиця 2.7

Checklist UI/HUD (швидка перевірка)

Критерій	Питання	Ознака проблеми
Контраст	текст читається на фоні?	«зливається», треба придивлятися
Ієрархія	що головне видно відразу?	Все однаково важливе
Консистентність	однакові правила іконок?	Одна дія – різні стилі
Мінімалізм	чи можна прибрати 20%?	Перевантаження, шум
Відгук	є фідбек на дію?	Гравець «не розуміє, спрацювало чи ні»
Доступність	масштаб/кольори ок?	Дрібний текст, лише колір як сигнал

Diegetic vs Non-diegetic UI

Diegetic UI існує у світі гри (табло на шоломі, екран пристрою).
Non-diegetic UI поверх екрану (HUD, меню).
Spatial UI в 3D просторі (маркер над об’єктом) (рис. 2.7).

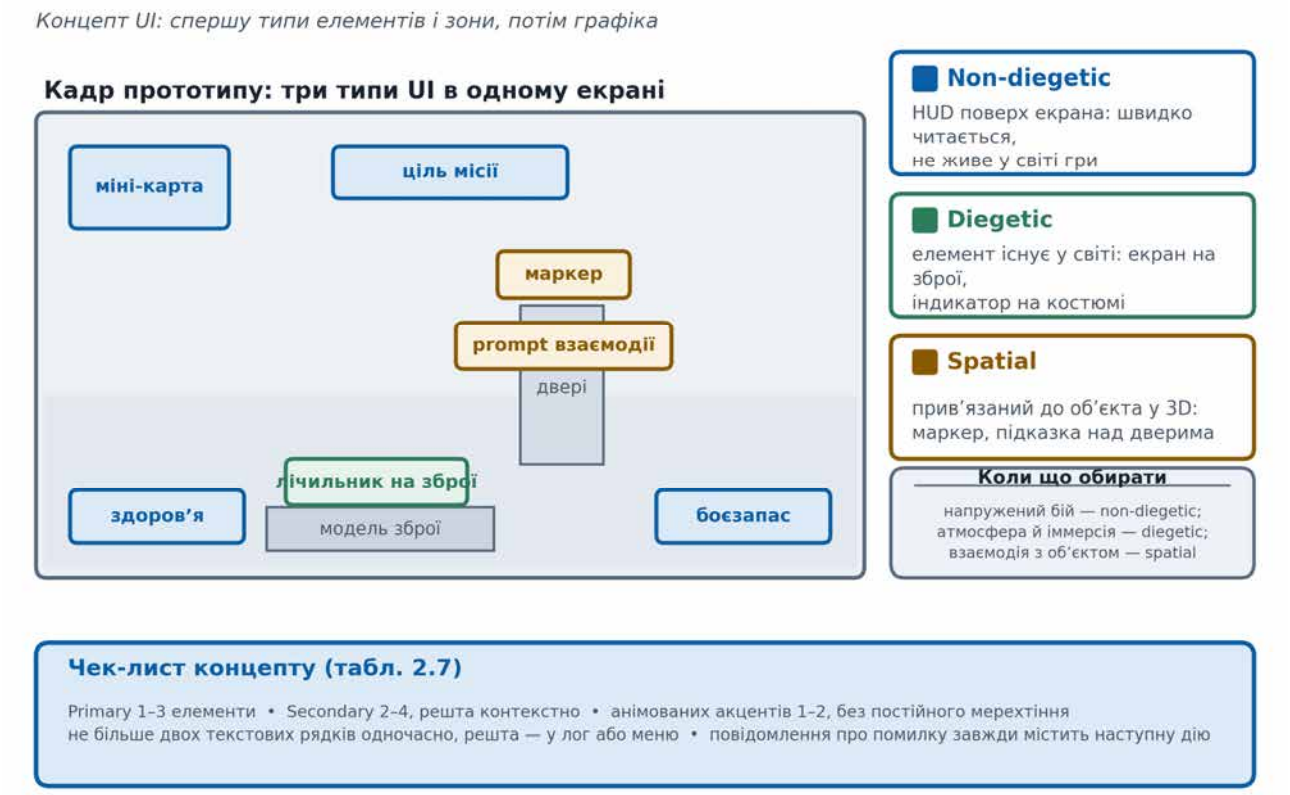


Рис. 2.7. Приклад концепт-дизайну UI для гри

2.2.5 UI Production: від wireframe до UI Kit і впровадження

Послідовність наступна

Wireframe (структура без стилю).
High-fidelity mockup (візуальний стиль).
UI Kit (кнопки, поля, типографіка, компоненти, стани).

Інтерактивний прототип (Figma/аналог).

Впровадження в рушій (Unity UI / UI Toolkit, Unreal UMG).

Тестування (на різних роздільностях, мовах, масштабах).

«Data-driven UI»

UI має бути керованим даними (HP, ammo, cooldown), а не «намальованим назавжди». Це зменшує помилки і прискорює ітерації.

Локалізація і масштабування передбачає

довші рядки тексту;

масштаб UI для мобільних/моніторів;

мінімальний розмір шрифту для читабельності.

2.2.6 VFX як система фідбеку: зрозуміло, дешево, красиво

VFX (частинки, спалахи, сліди, шейдер-ефекти) - це «мовні сигнали», якими можуть бути

• удар потрапив/не потрапив;

• критична втрата;

• небезпечна зона;

• лікування/баф/дебаф;

• взаємодія.

Принципи VFX наступні

Читабельність важливіша за красу, особливо в бою.

Таймінг: ефект має збігатися з моментом події.

Кодування сигналів: колір/форма/звук/анімація (не лише колір!).

Контроль шуму: не «сніг із частинок» на екрані (табл. 2.8).

Таблиця 2.8

Приклад бюджетів VFX (орієнтовно)

Платформа	Ціль FPS	Обмеження VFX (приклад)	Ризики
Mobile	30-60	менше овердроу, короткі ефекти	перегрів, просідання кадрів
PC (mid)	60	контроль draw calls/шейдерів	«шум» у бою
VR	90	дуже строгі бюджети, мінімум пост-ефектів	нудота через лаг

2.2.7 Анімація: game feel, стани, плавність і контроль

Анімація в іграх - це не «мультфільм», а керування станами і відгук на ввід. Ключовими задачами є наступні

• відчуття ваги і швидкості;

• зрозумілість атак/ухилів (телеграф);

• плавні переходи між станами;

• синхронізація з геймплейною логікою.

12 принципів анімації, адаптованих до ігор: Для ігор найважливіші: timing, anticipation, arcs, staging, follow-through, exaggeration (обережно), solid poses

У бойових іграх - anticipation і читабельний силует критичні.

State Machine і Blend Tree

Анімація живе у машинному стані: Idle -> Walk/Run -> Jump -> Attack -> Hit -> Death. Переходи мають умови (швидкість, кнопка, стан у повітрі), а також правила «чи можна їх перервати» (рис. 2.8).

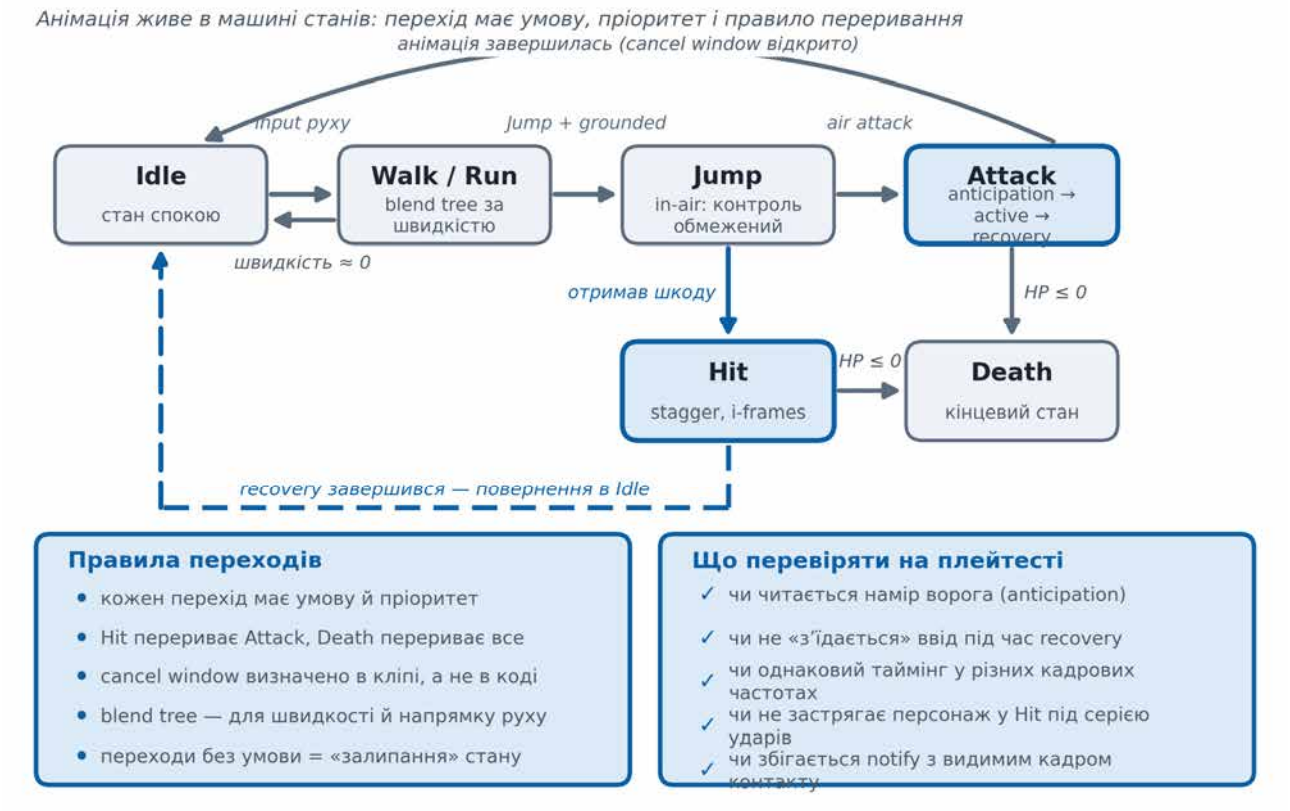


Рис. 2.8. Приклад машин станів персонажа

Інтеграція з геймплеєм

Hitbox/frames: коли саме «працює» удар (active frames).

Input buffering: гравець натискає трохи раніше, а гра запам'ятовує це.

Root motion vs in-place: рішення залежить від жанру.

2.2.8 Інтеграція та оптимізація: як не зламати FPS і читабельність

Оптимізація арту

• LOD для 3D, атласи текстур, обмеження розмірів текстур.

• Обережно з прозорістю (overdraw).

• Уніфікація матеріалів для batching.

Оптимізація UI

• Не малювати 10 оверлеїв одночасно.

• Стежити за перерисовкою canvas (Unity) або складністю UMG (Unreal).

• Тестувати на мінімальному цільовому пристрої.

Оптимізація VFX і анімації

• Вимірювати: скільки ефектів одночасно, як вони впливають на кадр.

• Компресія анімації, правильні частоти ключів, видалення «шуму».

2.2.9 Документація і контроль якості: що має бути в команді

Для малого проєкту достатньо наступного мінімального набору

Art Bible (стиль, палітра, приклади);

UI Spec (екрани, компоненти, правила масштабування);

VFX list (подія -> ефект -> бюджет -> пріоритет);

Animation list (стани, кліпи, правила переходів);

QA checklist (читабельність, контраст, FPS, консистентність).

Пам'ятайте: документація - це інструмент швидкої синхронізації, а не «папери заради паперів».

Питання для самоперевірки

1. Чим art direction відрізняється від «просто красивої графіки»?
2. Назвіть 6 елементів мінімального Style Guide.
3. Які 6 етапів арт-пайплайну ви виділите для студентського проєкту?
4. Чому неймінг активів і правила версійності є критичними?
5. Що таке ієрархія UI та чому «все важливе» = «нічого не важливе»?
6. Поясніть різницю diegetic / non-diegetic / spatial UI. Наведіть 3 приклади VFX як фідбеку (подія → ефект).
7. Що таке overdraw і чому прозорі VFX часто «вбивають» FPS?
8. Для чого потрібні state machine та blend tree в анімації?
9. Які 3 правила оптимізації арту ви застосуєте першими?
10. Як перевірити доступність UI для дальтонізму та користувачів із малим екраном?
11. Які мінімальні документи потрібні, щоб команда не «розсипалася» стилістично?

Практичний кейс (час виконання 120-180 хв): «Візуальна вертикальне демо (Vertical Slice)»

Мета: створити маленький «зріз» гри (1 екран/сцена), де узгоджені арт-стиль, UI/HUD, 1-2 VFX і 1 базова анімаційна схема.

Оберіть один сценарій (або запропонуйте свій)

• (A) «Арена» (top-down або 3rd person): HP, stamina, 2 здібності, ворог, удар.

• (B) «Платформер»: HP, колекційні предмети, стрибок, шкода від пастки.

• (C) «Виживання»: індикатори голоду/холоду, лут, взаємодія.

Кроки виконання

Міні-Style Guide (1 сторінка): палітра, 1–2 шрифти, 6 референсів, Do/Don't.

UI wireframe та high-fi макет: HUD та 1 меню/пауза.

UI Kit (мінімум): кнопка (3 стани), панель, іконка, tooltip.

VFX: 1 ефект «Hit/Impact» та 1 ефект «Ability/Interact» (з правилами читабельності).

Анімація: схема стейтів (Idle/Move/Action/Hit), опис переходів і таймінгу.

Бюджет/обмеження: 3 правила оптимізації (текстури/прозорість/UI).

Міні-тест: 2 людини відповідають: «Що важливе на екрані?», «Що сталося після натискання?».

Що необхідно для здачі завдання

1 стор. Style Guide

2-3 екрани UI (HUD та меню)

VFX list (2 ефекти: подія -> опис -> бюджет -> пріоритет)

Animation list та state diagram

1 стор. висновків: що було незрозуміло тестерам і що виправили

Критерії оцінювання

Цілісність стилю (style consistency)

Читабельність UI/HUD

Ясність фідбеку через VFX/анімацію

Дотримання обмежень продуктивності

Якість документації і аргументації рішень

Рекомендовані програми та сервіси

1. Дизайн/графіка: Figma, Adobe Photoshop/Illustrator або GIMP/Inkscape, Krita
2. 3D/текстури: Blender, Substance 3D Painter (або аналоги), Material Maker
3. 2D sprite/піксель-арт: Aseprite (або Piskel)
4. 2D анімація: Spine 2D (або DragonBones)
5. Рушії: Unity (UI / VFX Graph), Unreal Engine (UMG / Niagara)
6. Управління: Trello/Jira, Notion, Miro
7. Версійність: Git та LFS, Perforce (за потреби)
8. Доступність: Contrast checkers, color-blind simulators

Рекомендовані джерела та матеріали

1. Game UI Pro (серія збірок з практичними кейсами UI для ігор)
2. Jesse Schell, The Art of Game Design (розділи про досвід гравця і фідбек)
3. Jenifer Tidwell, Designing Interfaces (патерни UI, ієрархія, читабельність)
4. Alan Watt, 3D Computer Animation (основи анімації та постановки)
5. Офіційні документації Unity/Unreal (UI, VFX, анімація).

Корисні посилання (необхідно вставити в документ, як активні URL)

1. Unity Manual (UI): https://docs.unity3d.com/Manual/UISystem.html
2. Unity VFX Graph: https://docs.unity3d.com/Packages/com.unity.visualeffectgraph@latest
3. Unreal UMG: https://dev.epicgames.com/documentation/en-us/unreal-engine/umg-ui-designer
4. Unreal Niagara: https://dev.epicgames.com/documentation/en-us/unreal-engine/niagara-visual-effects
5. Blender Manual: https://docs.blender.org/manual/en/latest/
6. Figma Learn: https://help.figma.com/
7. WebAIM Contrast Checker: https://webaim.org/resources/contrastchecker/
8. Color Blindness Simulator (info): https://www.color-blindness.com/coblis-color-blindness-simulator/

Тема 2.3. Game UI/UX foundations: HUD, diegetic UI, меню та доступність

Мета теми: сформувати у студентів системне розуміння UI/UX у відеоіграх як частини інтерактивного дизайну та «мови зворотного зв'язку» (feedback language); навчити проєктувати читабельний HUD, вибирати між diegetic/non-diegetic/spatial UI, будувати онбординг і меню налаштувань, закладати основи доступності (accessibility) та формувати мінімальний план UX-телеметрії для перевірки гіпотез.

Очікувані результати навчання

- вміти пояснювати роль UI/UX як системи інформування, контролю та довіри (trust) у грі;
- розрізняти diegetic, non-diegetic та spatial UI та обґрунтовувати вибір під жанр і ситуації;
- проєктувати HUD з коректною інформаційною ієрархією, читабельністю у русі та адаптацією під різні екрани;
- застосовувати UX-патерни (стани, помилки, підтвердження, microinteractions) для зменшення «тертя»;
- будувати онбординг (micro-tutorial, progressive disclosure) і механізм контекстних підказок без спаму;
- визначати базові вимоги доступності: масштабування UI, субтитри, режими дальтонізму, ремапінг керування;
- проєктувати структуру options menu та збереження налаштувань (persistence) на рівні даних;
- підготувати мінімальний набір UX-метрик та подій телеметрії для перевірки гіпотез і виявлення проблем.

Ключові терміни: UI; UX; HUD; diegetic UI; non-diegetic UI; spatial UI; readability; information hierarchy; visual priority; affordance; signifiers; feedback; microinteraction; input remapping; accessibility (a11y); subtitles/captions; color-blind modes; safe area; DPI scaling; onboarding; tutorialization; progressive disclosure; dynamic hint; UX telemetry; events; funnel; retention; friction; error prevention; persistence; UI states.

2.3.1 UI/UX у відеоіграх: для чого це потрібно і що саме «ламається» без UX

У відеоіграх UI не є «декором поверх геймплею». Це операційний інтерфейс, який постійно відповідає на три запитання гравця: 1 - що відбувається? 2- що я можу зробити? 3- що буде, якщо я це зроблю?

UX (user experience) охоплює повний шлях взаємодії: від першого запуску і розуміння керування до прийняття рішень під тиском часу та довіри до системи (напр., чи чесно працює урон/хітбокси, чи зрозуміло повідомляють-ся стани, чи є контроль над помилками).

З практичної точки зору UI/UX у грі виконує функції: інформування (стан світу), управління (команди/налаштування), орієнтації (навігація, карта, цілі), навчання (онбординг), та регуляції емоцій (нагорода, напруга, розрядка). Коли UX поганий, гравець відчуває не складність, а «тертя»: невпевненість,

випадковість, незрозумілу поразку. Базовий інженерно-дизайнерський принцип теми: UI має зменшувати когнітивне навантаження та підвищувати передбачуваність. Для цього ми працюємо з ієрархією інформації, контрастом, таймінгами, станами і чіткою відповіддю системи на дію (табл. 2.9).

Таблиця 2.9

Типові «UX-поломки» та їх наслідки

Симптом	**Ймовірна причина**	**Наслідок для гравця**	**Швидке виправлення**
Не розуміє, що сталося	слабкий/несинхронний фідбек	відчуття «рандому»	дати 1-й фідбек: звук/спалах/текст
Не бачить важливе	немає ієрархії/пріоритетів	помилки в бою/таймінгах	визначити 1-3 ключові елементи HUD
Плутає кнопки/дії	погані signifiers/іконки	повільні рішення	текст/піктограми + підказки контексту
Меню «важке»	занадто багато пунктів	втома/вихід з гри	групувати, прогресивно розкривати, дефолти

2.3.2 Diegetic, non-diegetic та spatial UI: коли що працює

Класифікація UI за «місцем існування» в ігровому світі допомагає ухвалювати рішення щодо стилю, іммерсії та читабельності.

Diegetic UI - елементи інтерфейсу, що існують у світі гри і сприймаються персонажем (наприклад, індикатор здоров’я на костюмі). Non-diegetic UI - класичний HUD поверх екрану, який не є частиною світу. Spatial UI - UI, прив’язаний до об’єктів/простору (напр., підказка над дверима).

Поширений компроміс: у напружених жанрах (шутери, rogue-lite) часто потрібен non-diegetic HUD для швидких рішень, а diegetic елементи використовують як «підсилювач атмосфери». В адвенчурах і хоррах diegetic UI може бути домінантним, але лише якщо не шкодить швидкості читання (табл. 2.10).

Таблиця 2.10

Порівняння типів UI

Тип UI	**Плюси**	**Мінуси**	**Де застосовувати**
Diegetic	висока іммерсія; стиль	ризик поганої читабельності	хорор, immersive sim, адвенчура
Non-diegetic (HUD)	швидкість; контроль; стабільність	може «ламати» атмосферу	екшн, шутер, sport, rogue-lite
Spatial	контекстність; менше шуму	перевантаження при натовпі об’єктів	interactions, loot, навчання

Короткі кейси:

Dead Space - приклад diegetic UI (показники на костюмі), що підсилюють хорор-іммерсію.

Overwatch - приклад максимально читабельного HUD, де швидкість рішень важливіша за «реалістичність».
Portal / Monument Valley - приклади навчання через середовище і мінімалістичний UI.

2.3.3 Читабельність HUD: ієрархія, контраст, ритм та «інформаційна економіка»

HUD читається не в статичному скріншоті, а у русі. Тому важливі: контраст у русі, «шум» сцени, швидкість погляду і час на прийняття рішення. Найчастіша помилка студентських прототипів - показувати все одразу.

Практичне правило: визначте 1–3 «перші» елементи HUD (Primary), 2–4 «другі» (Secondary) і решту сховайте у контекст/меню. Primary HUD має бути видимим без фіксації погляду довше 200–300 мс.

Використовуйте ієрархію: розмір, вага шрифту, контраст, позиція (кут екрану), відстань до центру і «рух» (анімовані акценти). Пам’ятайте про safe areas (особливо під консоллю/TV) та про UI scaling під різні DPI (табл. 2.11; рис. 2.9).

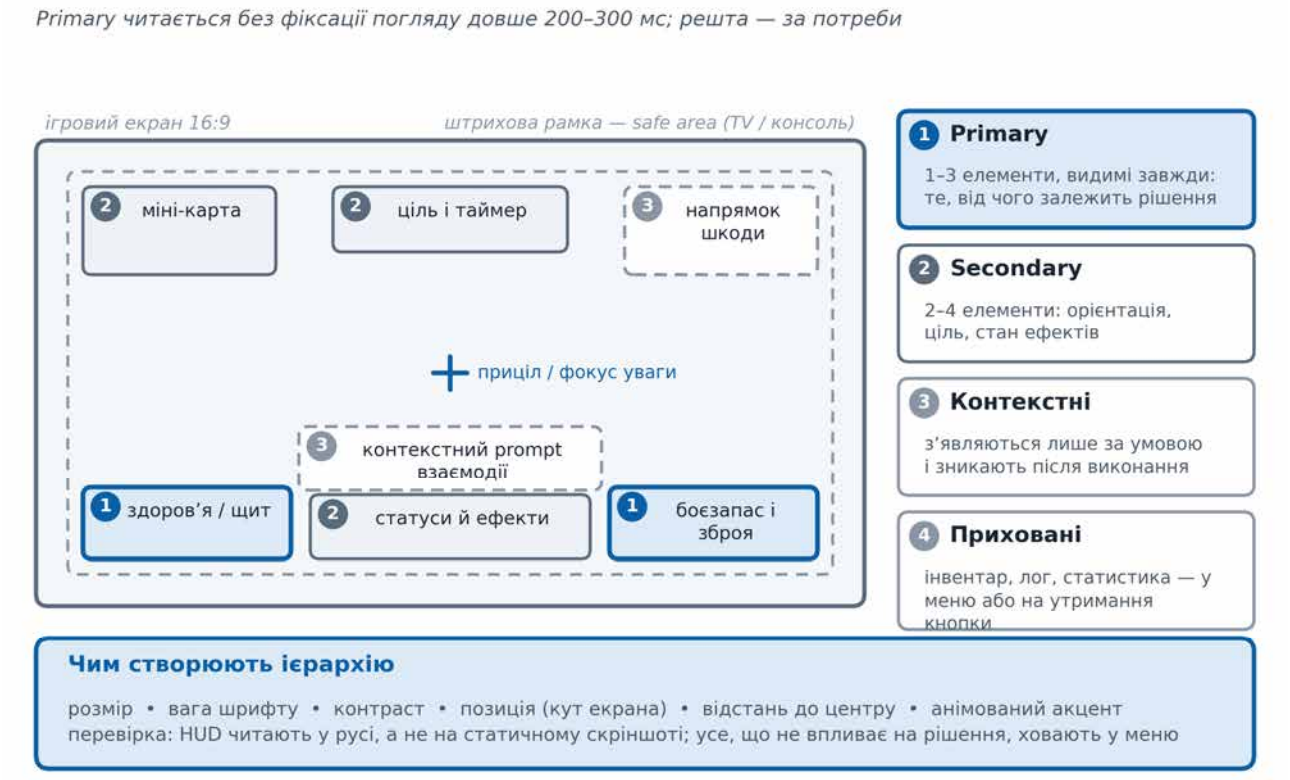


Рис. 2.9. Ієрархія інформації HUD (приклад)

Таблиця 2.11

Швидкі правила читабельності HUD

Правило	Навіщо	Швидка перевірка
Primary HUD - мінімум елементів	зменшує когнітивне навантаження	покрийте руками екран: чи все ще зрозуміло?
Контраст у русі	реальний фон завжди «шумний»	запишіть відео 10 с і перегляньте на 1.25x

Правило	Навіщо	Швидка перевірка
Одна інтонація - один сигнал	запобігає плутанині	урон!= нагорода (візуально/звуково)
Уникати дрібного тексту	читання під навантаженням	тест: 1080p/720p, 2 м від монітора
Анімація як акцент, не як шум	підтримує увагу	1-2 анімовані акценти на сцену

2.3.4 UX-патерни та microinteractions: стани, помилки, підтвердження

Microinteractions - це дрібні взаємодії, що формують відчуття якості: натискання кнопки, перемикання вкладки, підтвердження дії, повідомлення про помилку. У іграх microinteractions також працюють як «тонкий фідбек», який підсилює game feel.

Рекомендується мислити UI як машину станів: Normal, Hover/Focus, Pressed, Disabled, Loading, Success, Error. Кожен стан має однозначний фідбек: зміна кольору/яскравості, звук, мікро-анімація, інколи - короткий текст.

Антипатерни: зайві модальні вікна, підтвердження на кожен крок, приховані помилки, повідомлення без дії (no next step) (табл. 2.12).

Таблиця 2.12

Станова модель UI-елемента (мінімальний набір)

Стан	Як виглядає	Який фідбек	Примітка
Normal	базовий вигляд	немає або легкий ambient	має бути читабельним на фоні
Hover/Focus	підсвітка/обводка	тихий звук/анім. 80-150 мс	важливо для gamepad/keyboard
Pressed	вдавлювання/scale 0.95	click sound + короткий flash	підтверджує прийняття команди
Disabled	знижена контрастність	без звуку	пояснити причину блокування
Error	контрастний сигнал	помітний звук + текст дії	дати шлях виправлення

2.3.5 Масштабування, адаптивність, safe areas: Unity та Unreal на практиці

Адаптивний UI - вимога навіть для навчальних прототипів: гравець може запускати гру у різних роздільностях, віконному режимі, на моніторі з різним DPI або на TV. Для консолей додається safe area.

Unity: для UGUI використовуйте Canvas Scaler (Scale With Screen Size), anchors і TextMeshPro. UI Toolkit - сучасніший підхід, але UGUI залишається поширеним у прототипах. Уникайте абсолютних позицій без anchors.

Unreal: UMG підтримує DPI Scaling (Project Settings), anchors у Canvas Panel, а також масштабування шрифтів. Тестуйте HUD на 16:9, 21:9, 4:3, а також на мінімальній «навчальній» роздільній здатності 1280x720 (табл. 2.13).

Таблиця 2.13

Швидкі правила адаптивного UI (Unity/UE)

Проблема	Рішення	Перевірка
Елемент «пливе»	anchors + layout groups	перемкніть 16:9 -> 21:9
Текст не читається	TextMeshPro/Font size scaling	720p + відео у русі
UI перекриває важливе	safe area, margins	увімкніть safe area preview
Меню не керується gamepad	focus navigation / selectable order	пройдіть меню без миші

2.3.6 Онбординг і навчання: micro-tutorial, progressive disclosure, dynamic hints

Онбординг - це не «туторіал текстом», а кероване відкриття правил гри. Ідеальна модель: гравець робить дію, отримує фідбек, бачить наслідок і повторює це у складнішому контексті. Класичний приклад - Portal, де рівень «вчить» через простір.

Progressive disclosure означає: показувати інформацію лише тоді, коли вона потрібна. Dynamic hints мають базуватись на умовах (гравець застряг, багато разів помилився, не бачить ціль) і не перетворюватися на спам (рис. 2.10; табл. 2.14).



Рис. 2.10. Цикл онбордингу (практична модель)

Таблиця 2.14

Шаблон логіки dynamic hint (анти-спам)

Сигнал	Умова	Підказка	Cooldown
Застряг	N спроб без прогресу (напр., 20-40 с)	короткий hint + піктограма кнопки	60-120 с
Ігнорує UI	не відкрив меню/інвентар при потребі	одноразовий pop-up	один раз
Помилки керування	часті натискання не тієї кнопки	нагадування + remap prompt	120 с

2.3.7 Доступність (Accessibility): мінімальний стандарт для навчального прототипу

Доступність (a11y) - це не «опція для великих студій», а базова якість взаємодії. Для навчального прототипу достатньо реалізувати мінімальний набір: масштаб UI, субтитри/підписи, ремапінг керування, режим для дальтонізму або хоча б налаштування контрасту.

Правило: якщо елемент передає інформацію лише кольором, він є недоступним. Додайте форму, іконку, текст, патерн або звук (табл. 2.15).

Таблиця 2.15

Мінімальний чек-лист доступності (прототип)

Категорія	Мінімальна вимога	Як перевірити
UI scaling	масштаб 80-130%	перемкнути і пройти меню
Subtitles	on/off + розмір	тест сцени з діалогом/подіями
Remapping	перепризначення 2-3 дій	змінити клавішу і зіграти
Color reliance	не лише колір	увімкнути симуляцію дальтонізму (якщо є)
Motion/flash	обмеження інтенсивності	перевірити VFX на яскравість/частоту

2.3.8 Options menu та persistence: архітектура налаштувань без хаосу

Options menu - це «контракт» з гравцем: що я можу налаштувати і чи збережеться це завтра. Для прототипу важливо: дефолти мають бути розумними, зміни - видимими одразу, а збереження - надійним.

Практична структура: Gameplay (камера/чутливість), Controls (remap), Audio (mix), Video (quality), Accessibility (a11y). З технічного боку - модель даних (SettingsData) + система застосування (Apply) + система збереження (Save/Load).

У Unity часто використовують PlayerPrefs для прототипів, але для системності краще мати один JSON-файл. В Unreal - SaveGame об’єкт, або конфіги (ini) для окремих налаштувань (рис. 2.11; табл. 2.16).

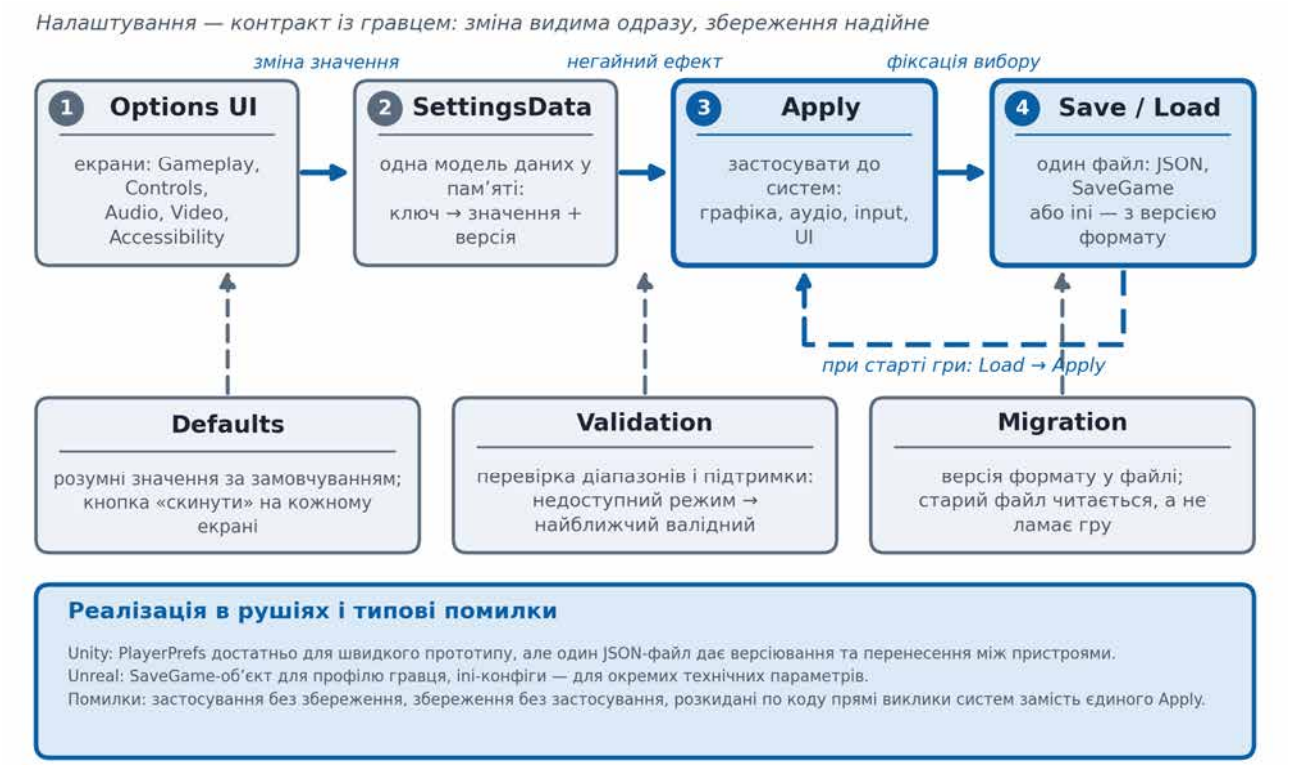


Рис. 2.11. Пайплайн налаштувань (SettingsData -> Apply -> Save)

Таблиця 2.16

Приклад структури даних налаштувань

Група	Параметр	Тип	Коментар
Gameplay	Sensitivity	float	камера/миша/стік
Controls	JumpKey	string	remap, валідація конфліктів
Audio	MasterVolume	float	0..1 + логарифмічна крива
Video	Resolution	string	підтримка перезапуску (за потреби)
Accessibility	UIScale	float	80..130%

2.3.9 UX telemetry: як вимірювати UX, а не «відчувати інтуїтивно»

UX-телеметрія у навчальному курсі потрібна не для «аналітики заради аналітики», а для перевірки гіпотез: чи зрозуміло гравцю керування, чи працює онбординг, де він застрягає, які опції змінює.

Мінімальний набір: події входу/виходу, старт рівня, смерть/фейл, успіх, відкриття меню, зміна налаштувань, показ підказки і реакція на неї. Важливо: збирати події етично, без персональних даних, і лише те, що допомагає покращити UX (табл. 2.17).

Таблиця 2.17

Мінімальна схема UX-подій для прототипу

Подія	Коли логувати	Поля	Навіщо
session_start	перший кадр/меню	platform, build, time	база для аналізу
tutorial_hint_shown	показали підказку	hint_id, context	чи спрацьовує онбординг
tutorial_hint_accepted	виконали дію	hint_id, time_to_action	ефективність підказки
options_changed	змінили налаштування	setting, from, to	які опції потрібні
fail_state	смерть/помилка	reason, location	де застрягають

2.3.10 Міні-пайплайн UI/UX тестування: плейтест, фіксація проблем, ітерації

UI/UX не «пишеться один раз». Його потрібно плейтестити так само, як механіки. Мінімальний цикл: сформулювати гіпотезу -> внести зміни -> перевірити на 3–5 людях -> зафіксувати проблеми -> повторити.

Протокол плейтесту для UI має включати: сценарії (що зробити), метрики (час/помилки), спостереження (де дивиться, що не розуміє) та коротке інтерв'ю (що було незрозуміло/що сподобалося).

Для курсу важливо вести невеликий backlog UI/UX задач і пріоритезувати: блокери (не можна пройти) -> критичне тертя -> косметика (рис. 2.12; табл. 2.18).



Рис. 2.12. Цикл поліпшення UI/UX (measure -> fix -> verify)

Таблиця 2.18

Міні-чеклист плейтесту UI/UX

Крок	Що перевіряємо	Як фіксуємо
1	чи бачить ціль/стан (Primary HUD)	відео + 2 нотатки
2	чи розуміє помилку/фейл	скрін + текст
3	чи знаходить меню/опції	час до знаходження
4	чи працює remap/scale	тест 1 зміни
5	чи спрацьовують підказки	time_to_action

Питання для самоперевірки

1. Поясніть різницю між UI та UX у контексті відеоігри.
2. Наведіть приклади diegetic та non-diegetic UI і поясніть їхні компроміси.
3. Що таке інформаційна ієрархія HUD? Як визначити Primary/Secondary елементи?
4. Чому читабельність потрібно перевіряти у русі, а не на скріншоті?
5. Які стани UI-елемента ви вважаєте мінімально необхідними і чому?
6. Поясніть принцип progressive disclosure на прикладі онбордингу.
7. Які умови роблять dynamic hints «спамом» і як цього уникати?
8. Назвіть мінімальний набір функцій доступності для навчального прототипу.
9. Чому не можна передавати важливу інформацію лише кольором? Опишіть пайплайн налаштувань: SettingsData -> Apply -> Save/Load.
10. Що таке UX-телеметрія і які події є мінімально достатніми для прототипу?
11. Складіть короткий протокол плейтесту UI/UX на 10 хвилин.

Практичний кейс до теми (120-180 хв): «Mini HUD + Options + Onboarding Hint + Telemetry»

Мета кейсу - створити маленький, але «production-minded» фрагмент UI/UX для вашого прототипу: читабельний HUD, меню налаштувань з мінімальними a11y-функціями та один механізм онбордингу (dynamic hint), а також спроєктувати мінімальну телеметрію для перевірки якості UX.

Сценарії виконання (оберіть один або комбінуйте)

Сценарій A (базовий): HUD + пауза + options (UI scale, subtitles on/off, master volume).

Сценарій B (поглиблений): додайте remapping для 2 дій (Jump, Interact) і тест на конфлікти.

Сценарій C (імерсивний): один diegetic елемент (напр., індикатор HP на об’єкті персонажа) + класичний HUD.

Покроковий план виконання завдання

- Створіть прототип HUD: HP (бар/число), ресурс (ammo/energy), таймер або ціль (objective). Визначте Primary/Secondary.

- Додайте microinteractions: стани кнопок (hover/pressed/disabled), звук кліку, короткий анімаційний акцент.
- Зробіть паузу/меню: Resume, Options, Quit (для прототипу - повернення у головне меню).
- Options: UI Scale (80-130%), Master Volume (0-100%), Subtitles (On/Off). За бажанням - Remap 2 дій.
- Onboarding: створіть 1 dynamic hint (напр., якщо гравець 20 с не взаємодіє з об'єктом - показати підказку). Додайте cooldown.
- Telemetry (мінімум): лог подій у файл/консоль: session_start, options_changed, hint_shown, hint_accepted, fail_state.
- Проведіть міні-плейтест на 1–2 людях: чи вони знаходять опції та чи розуміють підказку. Зафіксуйте 3 спостереження.

Що необхідно для здачі завдання:

- коротке відео 30-60 с з демонстрацією HUD, Options і підказки;
- 2-4 скріншоти: HUD у русі, меню Options, приклад підказки;
- README (1 стор.): що зроблено, які рішення UI/UX прийняті і чому;
- лог/скрін телеметрії або таблиця подій (мінімум 5 подій).

Критерії оцінювання

- читабельність HUD у русі та правильна ієрархія (0-4);
- якість microinteractions і фідбеку (0-3);
- логічна структура Options + працездатне збереження (0-4);
- dynamic hint без спаму (умови + cooldown) (0-3);
- мінімальна телеметрія і здатність зробити висновок (0-3);
- оформлення здачі (відео/README) (0-3).

Рекомендовані програми та сервіси:

Unity: UGUI, TextMeshPro, Canvas Scaler, UI Toolkit (за бажанням), Localization package (опційно).

Unreal Engine 5: UMG, Common UI (опційно), Enhanced Input (для remap), SaveGame.

Дизайн UI: Figma (wireframes, компоненти), Miro (user flow).

Графіка: Krita/GIMP/Inkscape; іконки: Icons8 (з ліцензією), Feather/Lucide (open).

Тестування: OBS (запис плейтесту), короткі Google Forms для фідбеку.

Організація: Trello/Notion (backlog UI/UX задач).

Рекомендовані джерела та матеріали

1. **Unity Manual:** UI (UGUI) та Canvas Scaler (офіційна документація).
2. **Unity Manual:** TextMeshPro (офіційна документація).
3. **Unreal Engine Documentation:** UMG UI Designer (офіційна документація).
4. **Unreal Engine Documentation:** DPI Scaling / UI Scaling (офіційна документація).
5. Game Accessibility Guidelines (основні рекомендації).
6. W3C WCAG Quick Reference (базові принципи доступності).
7. **GDC:** доповіді з UI/UX і readability (добірка).
8. **Game Maker's Toolkit:** відео про UI/UX та навчання через рівні (Portal).

Додаток до теми. Шаблони для швидкої роботи з UI/UX у прототипі:

Нижче наведені короткі шаблони, які студент може використати як «швидкий стандарт» у командній роботі. Їх мета - за 10-15 хвилин зафіксувати правила UI, щоб уникнути хаотичного змішування стилів та полегшити ітерації.

A) UI Style Guide (шаблон на 1 сторінку)

UI Style Guide - це короткий документ/сторінка в Notion або Figma, де визначено: шрифти, розміри, кольори, відступи, правила іконок та таймінги анімацій. У навчальному прототипі достатньо мінімального набору (табл. 2.19).

Таблиця 2.19

Шаблон UI Style Guide

Розділ	**Що визначити**	**Приклад/підказка**
Typography	2-3 розміри, 2 ваги	Title 28-32, Body 18-20, Caption 14-16
Colors	Primary/Secondary + Error/Success	перевірити контраст на фоні гри
Spacing	базовий крок (8/12)	усі відступи кратні кроку
Buttons	стани + таймінги	hover 100 ms, pressed 80 ms, disabled без звуку
Icons	стиль і товщина	одна бібліотека, без міксу стилів
Audio UI	звук кліку/помилки	тихо, коротко, без «дратування»
Accessibility	UI scale, subtitles	мінімальний набір у Options

B) User Flow та сценарії меню (шаблон)

User Flow - це маршрут гравця між екранами: головне меню -> гра -> пауза -> опції -> повернення. Навіть у прототипі він має бути однозначним і не створювати «петель» або тупиків (рис. 2.13; табл. 2.20).

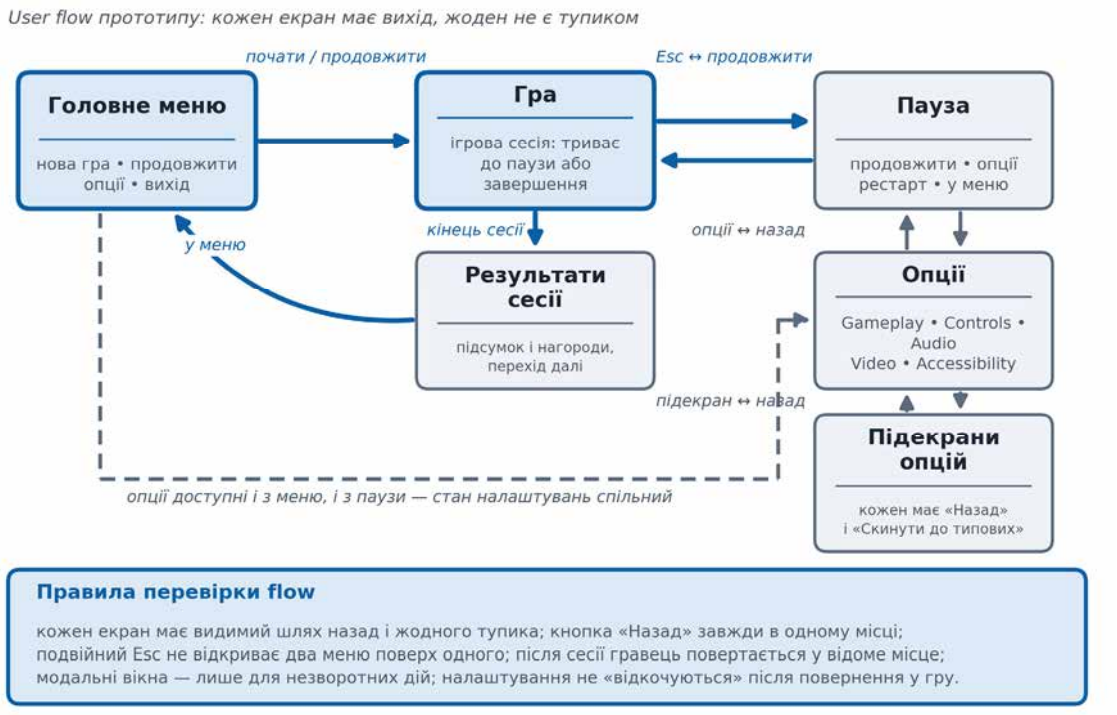


Рис. 2.13. User Flow (мінімальний для прототипу)

Таблиця 2.20

Перелік екранів та обов'язкові елементи

Екран	Обов'язкові елементи	Типові помилки
Main Menu	Start/Continue, Options, Quit	немає виходу/повернення
HUD	Primary метрики + 1 контекстний блок	зайві дрібні елементи, текст
Pause	Resume, Options, Main Menu	втрата фокусу з gamepad
Options	Audio, Controls, UI scale, Subtitles	зміни не застосовуються одразу
Game Over	Retry, Main Menu, коротка причина	порожній екран без next step

C) HUD budget: ліміти, щоб не перевантажувати екран

HUD budget - це домовленість, скільки елементів може бути на екрані і скільки з них можуть бути анімованими. Ліміти допомагають підтримувати читабельність і не «з'їдати» увагу гравця (табл. 2.21).

Таблиця 2.21

Приклад HUD budget для навчального прототипу

Параметр	Рекомендація	Коментар
Primary HUD елементів	1-3	HP/Ammo/Timer або ціль
Secondary елементів	2-4	контекстно або з fade-in/out
Анімованих акцентів	1-2	без постійного мерехтіння
Текстових рядків одночасно	до 2	решта - у log/меню
Сигналів помилки	короткі + actionable	завжди давати «що зробити»

Міні-вправа: «UI Review на 60 секунд»:

- Зробіть скрін/відео 10 с вашого HUD у русі.
- За 60 секунд відповідайте: що таке Primary? Що відволікає? Де контраст слабкий?
- Складіть 3 правки: одну про ієрархію, одну про фідбек, одну про адаптивність.
- Внесіть правки і повторіть тест (до/після).

Тема 2.4. Materials & Shaders I: PBR та стилізація

Мета теми: надати студентам практичне та теоретичне підґрунтя для створення матеріалів і шейдерів у Unity та Unreal Engine 5, з фокусом на базовий PBR-пайплайн і поширені прийоми стилізації. У результаті опрацювання теми студент(ка) має розуміти, як текстури й параметри матеріалу впливають на читабельність у русі та імерсивність, уміти збирати керовані (parameterized) матеріали, а також реалізувати два практичні ефекти: **dissolve** та **fresnel/outline** (аналог лабораторної роботи L6)

Очікувані результати навчання

- пояснювати принципи PBR (metallic/roughness workflow), роль BRDF та енергозбереження;
- розрізняти типи текстур і коректно налаштовувати color space (sRGB/Linear), normal maps та packing;
- порівнювати матеріальний пайплайн Unity (URP/HDRP) і UE5 (Master Material + Material Instances);
- будувати матеріали як параметризовані системи (tunables) та створювати пресети без дублювання шейдерів;
- застосовувати базові прийоми стилізації: toon/ramp shading, rim lighting, outlines, контроль контрасту;
- реалізувати **dissolve shader** з edge highlighting та керованими параметрами;
- реалізувати **fresnel/outline shader** (rim + один із методів контуру) та пояснити вибір методу;
- оцінювати вартість шейдера (інструкції/overdraw/варіанти) і виконувати первинну оптимізацію;
- оформлювати короткий shader breakdown (скрін/діаграма вузлів + список параметрів + правила використання).

Ключові терміни: Material, shader, PBR, BRDF, albedo/base color, metallic, roughness, specular, normal map, AO, emissive, texture sampling, tiling/UV, mipmaps, sRGB/Linear, tangent space, vertex color, mask packing, master material, material instance, shader graph, node, parameter, keyword/variant, toon shading, ramp, fresnel, rim light, outline (inverted hull / post-process / screen-space), dissolve, noise, threshold, edge width, overdraw, shader complexity, draw call, LOD, batching

2.4.1 Матеріал і шейдер: що саме виконується на GPU

Матеріал у рушії - це набір параметрів (текстури та числові значення) і прив’язка до шейдера. Шейдер - програма, що виконується на GPU і визначає, як об’єкт виглядає при освітленні, у тіні, в умовах постобробки та як він реагує на додаткові ефекти (емісія, прозорість, дисторсія тощо).

Критичний принцип для production-підходу: **один Master Shader/Material -> багато Material Instances/Variants**. Тобто ми не копіюємо шейдери «під кожну модель», а задаємо параметри, які можна швидко змінювати (tunables) (рис. 2.14).

2.4.2 Основи PBR: чому metallic/roughness - «мова матеріалів»

PBR (Physically Based Rendering) - набір моделей і практик, що забезпечують стабільний вигляд матеріалів у різних умовах освітлення. У навчальному пайплайні найзручніше використовувати metallic/roughness workflow, де:

Base Color (Albedo) - «власний» колір матеріалу без освітлення;

Metallic - чи поводиться поверхня як метал (0..1);

Roughness - мікрошорсткість, що керує розмиттям відбиття (0..1);

Normal - мікрорельєф (впливає на світлотінь і спекуляр);

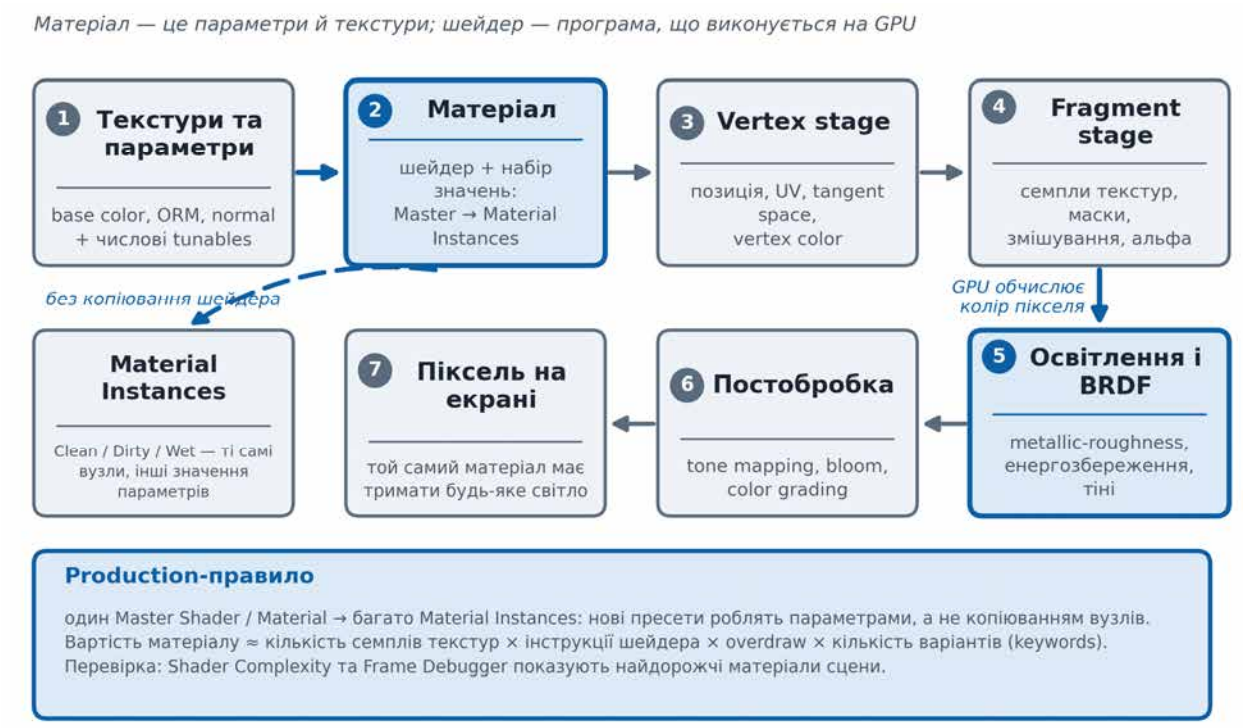


Рис. 2.14. Узагальнена схема: від матеріалу до пікселя (pipeline спрощено)

Mesh/Vertices -> Vertex Shader (позиції, UV, normal, color) -> Rasterization -> Pixel/Fragment Shader
-> Sample Textures + Evaluate Lighting (PBR/Toon) + Apply Effects (emissive/outline/dissolve) -> Framebuffer

AO (Ambient Occlusion) - затемнення у заглибленнях (обережно з подвійним затемненням).

PBR не гарантує «фотореалізму», але гарантує **узгодженість: якщо матеріал правильно налаштований, він не «ламається» під час зміни** світла або HDR (табл. 2.22).

Таблиця 2.22

Типи текстур PBR і практичні правила використання

Карта	**Діапазон/ Space**	**Типові помилки**	**Практичні поради**
Base Color	sRGB	вписують тіні/бліки в албедо	не малюйте освітлення в албедо; контролюйте насиченість
Metallic	Linear	«сірі метали» без логіки	метал зазвичай 0 або 1; змішані лише для бруду/фарби
Roughness	Linear	занадто глянцево або занадто матово	перевіряйте на різних HDRI; робіть пресети
Normal	Linear (special)	неправильний імпорт/інверсія каналу	перевіряйте tangent space; не в sRGB
AO	Linear	подвійне затемнення	використовуйте делікатно; не «вбивайте» форму
Emissive	Linear	надмірний bloom	обмежуйте значення; робіть нічний/денний пресети

2.4.3 Color space, mipmaps, packing: технічні деталі, що «ламають» матеріали

Навіть хороший шейдер може виглядати погано через неправильні налаштування імпорту текстур. Ключова відмінність: **sRGB** - для «кольору», **Linear** - для «даних». Параметричні карти (metallic, roughness, masks) майже завжди мають бути в Linear.

Пакування масок (mask packing) економить пам'ять і семпли

- R канал - AO, G - Roughness, B - Metallic (або інший прийнятий стандарт);
- завжди документуйте, що саме лежить у каналах;
- перевіряйте mipmaps: дрібні деталі можуть «плисти», якщо шум/маска занадто контрастні (рис. 2.15).

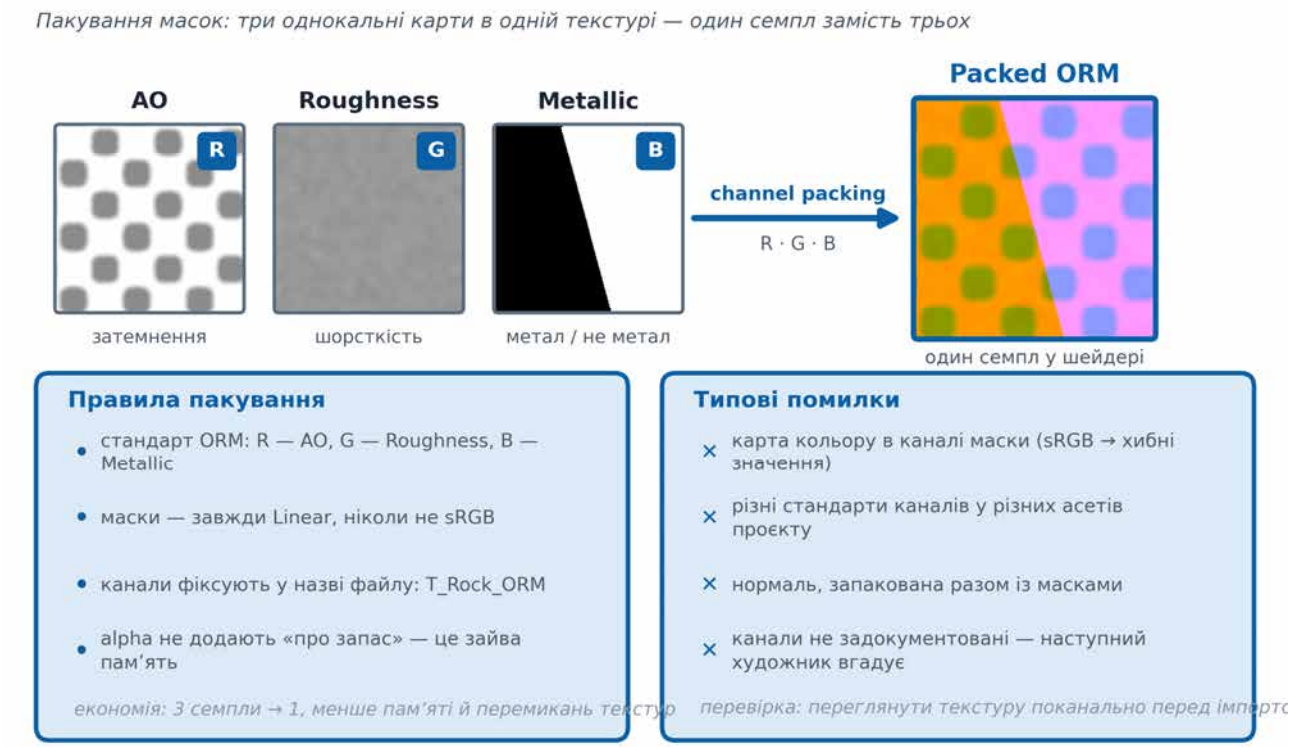


Рис. 2.15. Приклад пакування масок (ORM)

R: Occlusion (AO)
G: Roughness
B: Metallic
A: (опційно) Height/Detail mask

2.4.4 Unity vs Unreal: матеріальний пайплайн, інстанси, варіанти

Unity та Unreal мають схожі концепції, але різні «шляхи користувача». У Unity важливо враховувати обраний рендер-пайплайн (Built-in/URP/HDRP) та відповідні шейдери. У UE5 базовий підхід – Master Material + Material Instances, що дуже зручно для швидкого налаштування вигляду (табл. 2.23).

Таблиця 2.23

Порівняння підходів Unity та UE5 (для навчальних прототипів)

Аспект	Unity (URP/HDRP)	Unreal Engine 5	Коментар
Базова модель	Shader + Material	Master Material + Instances	в UE інстанси – стандарт де-факто
Візуальний редактор	Shader Graph	Material Editor	обидва node-based
Варіанти	Keywords/Variants	Static Switch/Parameters	контролюйте кількість варіантів
Профілювання	Frame Debugger/Profiler	Shader Complexity/Insights	оцінюйте overdraw і інструкції
Контент	URP/HDRP пакети	Megascans/Material Functions	UE має сильну екосистему матеріалів

2.4.5 Стилізація: керування формою, контрастом і «мовою матеріалів»

Стилізація – це не «вимкнути PBR», а переосмислити, як матеріал кодує форму і важливість. Підходи: toon/ramp shading, контроль спекуляру, обмежена палітра, rim light як читабельність, outline як графічний контур, емісія як акцент. Для навчальних кейсів зручно аналізувати Guilty Gear: Strive (виражена toon-естетика) та Fortnite (узгоджені матеріали + читабельність) (табл. 2.24).

Таблиця 2.24

Прийоми стилізації та коли вони корисні

Прийом	Навіщо	Ризик	Як уникати
Ramp/Toon	чіткі переходи світло/тінь	плоскість	додайте rim + AO/деталі
Rim/Fresnel	читабельність силуету	пластикова «обводка»	керуйте інтенсивністю і масками
Outline	графічний стиль, виділення	мерехтіння/артефакти	вибір методу + антиаліасинг
Emissive accents	акценти/навігація	пересвіт + bloom	обмежуйте значення, робіть пресети
Specular shaping	підкреслення форми	надто глянцево	керуйте roughness, clamp

2.4.6 Shader Graph / Material Editor: мінімальний набір навичок

Node-based шейдери зручні для швидких ітерацій. Мінімальний набір навичок, необхідний у семестрі:

- Parameters: float, color, texture, toggle/switch; групування параметрів;
- UV та tiling/offset; sample textures; нормалі та перетворення;
- Masking: smoothstep, saturate, remap, clamp;
- Логіка: add/multiply/lerp, step, power, one-minus;

• Preview/debug: виведення окремих масок у emissive для діагностики;
• Variant hygiene: не створювати десятки непотрібних keywords/статичних switch без причини (рис. 2.16).

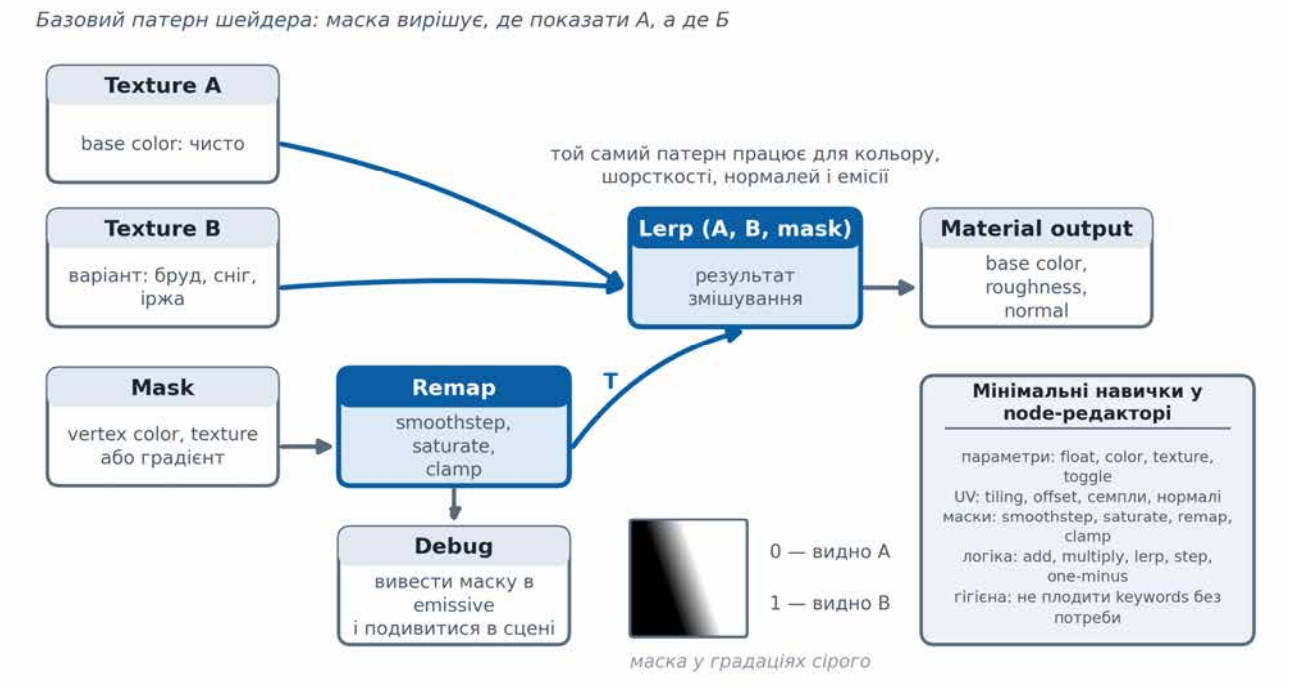


Рис. 2.16. Базовий патерн «маска керує змішуванням»

result = lerp(A, B, mask)
mask = saturate(remap(input))

2.4.7 Dissolve: шум + поріг + edge highlight (керований ефект)

Dissolve - класичний ефект для телепортації, появи/зникнення, розпаду ворогів. Суть: беремо noise/mask, порівнюємо з параметром threshold і відсікаємо пікселі. Для читабельності додаємо edge band (обводка) навколо межі (рис. 2.17).

Рекомендовані параметри (tunables) для dissolve

Threshold (0..1) – основний контролер появи/зникнення.

EdgeWidth (0.01..0.2) – ширина смуги краю.

EdgeColor / EdgeIntensity – колір і яскравість обводу.

NoiseTiling – масштаб шуму (залежить від масштабу моделі).

Optional: Directional dissolve (по Y/по нормалі) для контрольованих ефектів.

Чеклист «читабельного dissolve» (engine-ready)

Край (edge) має бути видимим на фоні (контраст + bloom контроль).

Використовуйте clip/alpha test там, де це доречно (але пам’ятайте про aliasing).

Не робіть шум надто дрібним – mipmaps можуть спричинити «пульсацію».

Для прозорих матеріалів оцінюйте overdraw (особливо у VFX сценах).

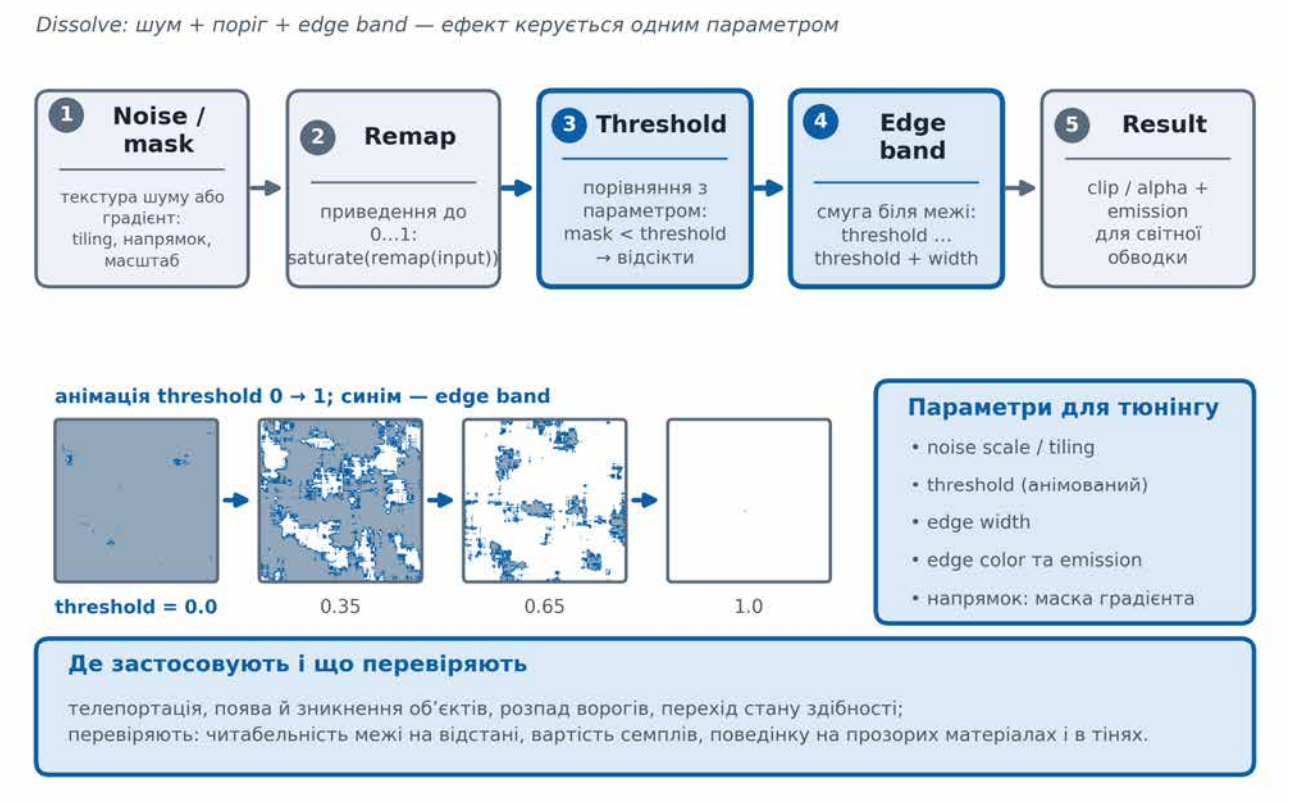


Рис. 2.17. Логіка dissolve (узагальнено)

*noise = sample(noiseTex, UV * tiling)*
cut = noise – threshold
alpha = step(0, cut) # що залишилось
edge = smoothstep(-w, 0, cut) # смуга краю
color = lerp(edgeColor, baseColor, alpha)

2.4.8 Fresnel / Outline: rim light і методи контуру

Fresnel (rim lighting) – ефект, що підсилює силует за рахунок збільшення яскравості на краях, коли нормаль поверхні перпендикулярна до напрямку погляду. Він корисний для читабельності персонажа та інтерактивних об'єктів. Outline – окремий прийом для графічного контуру або підсвітки (рис. 2.18).

Методи outline (виберіть один для лабораторної залежно від рушія/пайплайну)

- Inverted Hull (геометричний): дубль меша з розширенням по нормалі, рендер задніх граней.
- Post-process (екранний): outline за depth/normal буферами, добре для багатьох об'єктів.
- Screen-space (shader): використання нормалей/глибини (обмежено в Shader Graph без пост-ефектів) (табл. 2.25).

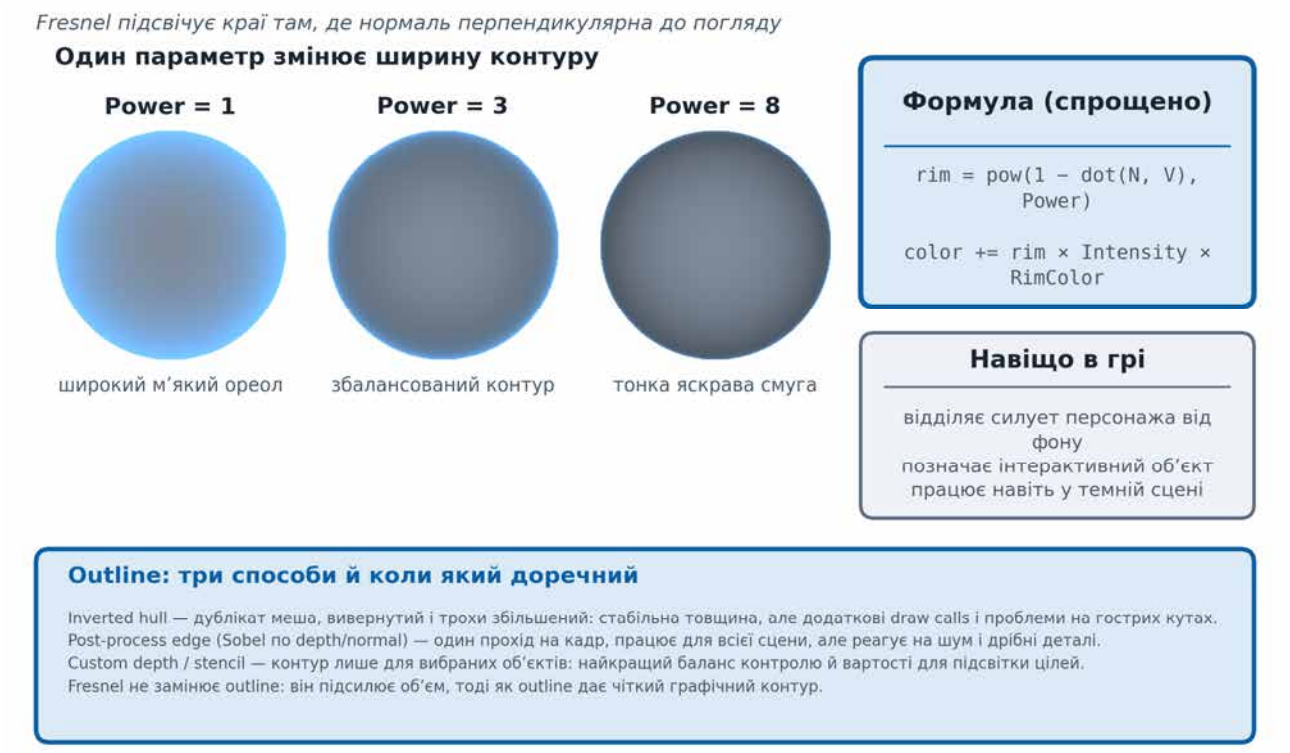


Рис. 2.18. Fresnel (спрощено)

view = normalize(CameraPos – WorldPos)
f = pow(1 – saturate(dot(normalize(N), view)), power)
rim = f * rimColor * rimIntensity

Таблиця 2.25

Порівняння методів outline

Метод	Переваги	Недоліки	Рекомендація для курсу
Inverted Hull	простий, контроль товщини	подвоює геометрію, проблеми на тонких деталях	добре для 1-3 ключових об'єктів
Post-process	масштабується на багато об'єктів	потрібен PP налаштування, іноді артефакти	краще для UE5 або Unity HDRP/URP з PP
Screen-space	без дублю геометрії	обмеження точності/доступу до буферів	використовуйте обережно

2.4.9 Продуктивність: shader cost, overdraw, варіанти та «бюджети»

Шейдери часто стають «невидимою» причиною просідання FPS. Навіть у прототипі корисно мислити бюджетами: скільки семплів текстур, скільки умовних гілок, чи є прозорість (overdraw), скільки варіантів (shader variants).

Типові оптимізаційні правила для навчального vertical slice

- Уникайте зайвих прозорих матеріалів у масових об'єктах (прозорість = overdraw).
- Пакуйте маски (ORM) і зменшуйте кількість texture samples.
- Виносьте важкі ефекти у окремі матеріали лише для «герой-об'єктів».

- Контролюйте static switches/keywords – варіанти множаться експоненційно.
- Перевіряйте Shader Complexity / Frame Debugger: виявляйте найбільш «дорогі» матеріали (табл. 2.26).

Таблиця 2.26

Типові помилки матеріалів і як їх виправляти

Симптом	Ймовірна причина	Як перевірити	Як виправити
Матеріал «пластиковий»	низька roughness, завеликий spec	HDRI test, різні кути	підніміть roughness, обмежте spec, додайте detail
Normal map «перевернутий»	інверсія Y/не той формат	тест зі світлом збоку	змініть імпорт/інвертуйте канал
Мерехтіння шуму dissolve	занадто дрібний noise + mipmaps	віддалення камери	збільште масштаб шуму, використайте smoother noise
Outline рве на тонких деталях	inverted hull на дрібній геометрії	тест на thin meshes	перейдіть на post-process або обмежте outline
FPS падає у VFX сцені	прозорість + багато шарів	overdraw/complexity view	переведіть у masked, спростіть матеріал, зменшіть шарування

2.4.10 Production-ready матеріали: структура, іменування, пресети, документація

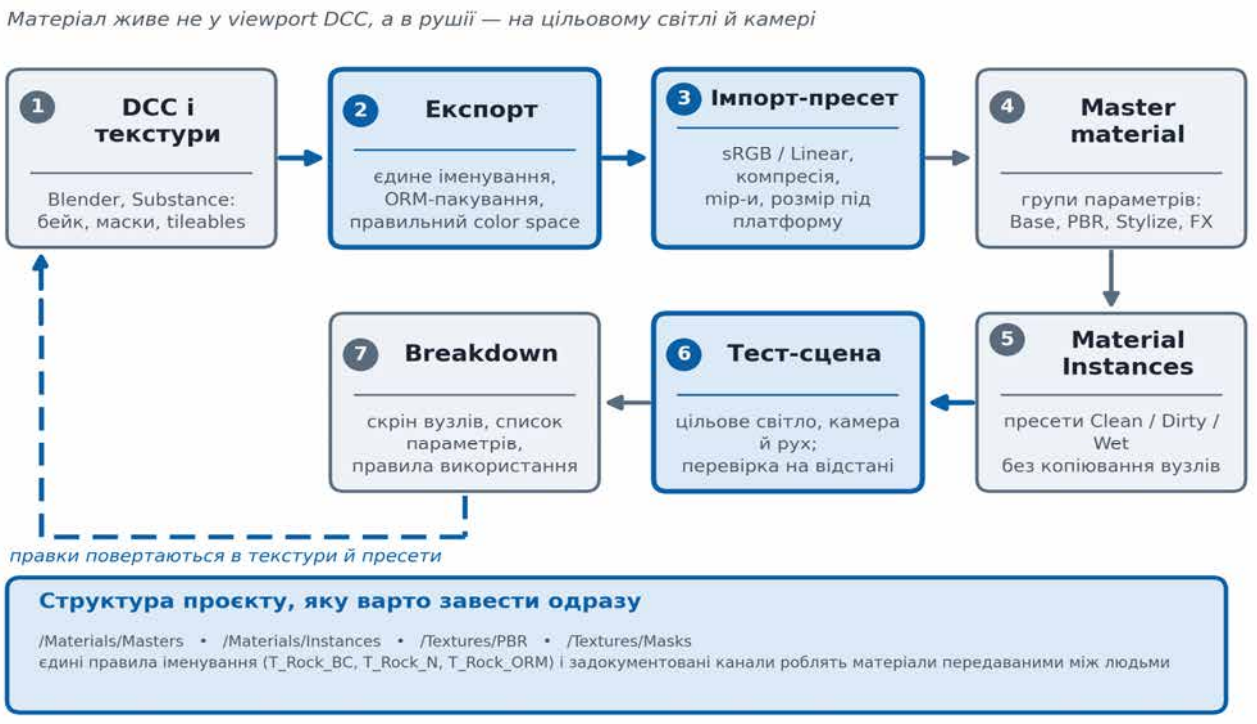


Рис. 2.19. Міні-пайплайн матеріалу (від DCC до рушія)

DCC (texture/shapes) -> Export textures (sRGB/Linear) + packing -> Import settings -> Master Material
-> Create Instances/Presets -> Apply to meshes -> Lighting check (HDRI) -> Performance check -> Commit (VCS)

Для команди важливо, щоб матеріали були підтримувані. Мінімальні правила

1. Master materials/shaders з чіткими групами параметрів (Base, PBR, Stylize, FX).
2. Material Instances/Variants для різних пресетів (наприклад: Clean/Dirty/Wet).
3. Єдині правила іменування текстур і пакування каналів (ORM).
4. Папки: /Materials/Masters, /Materials/Instances, /Textures/PBR, /Textures/Masks.
5. Короткий breakdown: скрін вузлів + список параметрів + приклади використання(рис. 2.19).

Питання для самоперевірки

1. Що таке матеріал і чим він відрізняється від шейдера?
2. Поясніть, чому PBR забезпечує узгодженість матеріалів за різних умов освітлення.
3. Чому Base Color у sRGB, а Metallic/Roughness – у Linear?
4. Наведіть приклад, як неправильно налаштований normal map «ламає» матеріал.
5. Що таке mask packing і навіщо використовується ORM?
6. Опишіть різницю Master Material та Material Instance (або Unity-еквіваленти).
7. Назвіть 3 прийоми стилізації та їхній вплив на читабельність.
8. Яка логіка dissolve-ефекту (noise + threshold + edge) і які параметри для цього потрібні?
9. Що таке Fresnel/rim light і для чого він корисний в іграх?
10. Порівняйте методи outline: inverted hull та post-process.
11. Що таке overdraw і чому прозорість часто дорога?
12. Назвіть 3 правила «production-ready» організації матеріалів у проєкті.

Практичний кейс (120-180 хв): «Two Shader Graphs Pack» (Dissolve + Fresnel/Outline)

Завдання: створити два керовані шейдери/матеріали у вибраному рушії (Unity або UE5) та продемонструвати їх застосування на 2-3 об’єктах у тестовій сцені. Результат має бути параметризований (tunables), мати пресети та коротку документацію.

Підготовка тестової сцени

• Створіть просту сцену (1 кімната або арена) з базовим освітленням (HDRI або 2-3 джерела).

• Додайте 2-3 моделі: персонаж/об’єкт, проп і простий куб/сфера для тестів.

• Створіть базовий PBR матеріал (Base+Normal+ORM) і переконайтеся, що він виглядає стабільно.

Завдання 1: Dissolve material (мінімум параметрів, максимум читабельності)

• Реалізуйте noise-based dissolve з параметрами Threshold, EdgeWidth, EdgeColor/Intensity, NoiseTiling.

• Додайте edge highlighting (смуга) і переконайтеся, що межа читабельна в русі.

• Зробіть 2 пресети: «Soft Dissolve» та «Sharp Sci-Fi Dissolve».

• Додайте керування Threshold (анімація, timeline/sequence або debug slider).

Завдання 2: Fresnel + Outline

• Реалізуйте fresnel/rim light з параметрами RimColor, RimPower, RimIntensity.

• Додайте outline одним із методів на вибір: inverted hull (геометрія) або post-process.

• Зробіть 2 пресети: «Subtle Rim» (для читабельності) та «Highlight Outline» (для інтерактивних об'єктів).

Debug & профілювання (обов'язково)

• Увімкніть перегляд складності шейдерів/overdraw (Unity Frame Debugger/Profiler; UE Shader Complexity/Insights).

• Знайдіть 1 потенційну проблему продуктивності та запропонуйте просте виправлення (без фанатизму).

Deliverables (що здати)

• Проєкт/сцена Unity або UE5 з двома матеріалами (Dissolve, Fresnel/Outline).

Коротке відео 30-45 с: базовий PBR -> dissolve -> rim/outline у русі.

• Shader breakdown: 1 сторінка (структура вузлів, параметри, 2 пресети).

• Короткий постмортем (150-250 слів).

Критерії оцінювання

• Коректність PBR (налаштування текстур, стабільний вигляд у світлі).

• Читабельність dissolve та outline у русі (контраст, масштаб шуму, стабільність).

• Параметризація: зрозумілі tunables + пресети, без дублювання матеріалів.

• Охайність пайплайну: іменування, папки, коротка документація.

• Урахування продуктивності: перевірка cost/overdraw і мінімальна оптимізація.

Рекомендовані програми та сервіси

1. Unity LTS (URP/HDRP): Shader Graph, Material Inspector, Frame Debugger, Profiler.
2. Unreal Engine 5: Material Editor, Material Instances, Shader Complexity, Unreal Insights.
3. Текстури/матеріали: ArmorPaint, Material Maker, Substance 3D (за можливості), Quixel Mixer.
4. Ресурси для тесту: Poly Haven (HDRI/матеріали), Kenney (CC0), Quixel Megascans (UE).

5. Захоплення відео: OBS, Unity Recorder, UE Movie Render Queue.

Рекомендовані джерела та матеріали

Офіційна документація Unity щодо Shader Graph, матеріалів URP/HDRP та імпорту текстур; документація Unreal Engine 5 щодо Material Editor, Material Instances і параметрів. Для розуміння базових принципів шейдерів рекомендовано The Book of Shaders та навчальні матеріали щодо PBR workflow (metallic/roughness). Для стилізації – розбори кейсів Guilty Gear: Strive та Fortnite (узгодженість матеріалів і читабельність).

Тема 2.5. Game FX Design: particles, timing та читабельність

Мета теми: сформувати у студентів практичні навички проєктування і реалізації ігрових візуальних ефектів (VFX) як частини системи відгуку та «game feel», з фокусом на таймінг (anticipation/impact/recovery), читабельність у русі, подієву інтеграцію (event-driven FX) та базові бюджети продуктивності. У результаті опрацювання теми студент(ка) зможе створювати керовані FX-пакети для здібності/атаки (ability) у Unity (VFX Graph/Particle System) або UE5 (Niagara), під'єднати їх до подій взаємодії/анімації та оцінити вплив ефектів на FPS

Очікувані результати навчання

- пояснювати роль VFX у game feel та в системі «input -> state -> feedback»;
- проєктувати ефекти за моделлю anticipation/impact/recovery і підбирати таймінги під жанр;
- будувати VFX як модульні системи (emitters, bursts, ribbons/trails, decals, distortion) та створювати пресети;
- забезпечувати читабельність ефектів: силует, контраст, масштаб, колірна мова, шарування;
- реалізовувати event-driven FX: запуск від input/state/animation notify/overlap/hit;
- налаштовувати просту VFX-«мову» для гри (категорії, правила кольору, інтенсивності, тривалості);
- використовувати Niagara або VFX Graph для створення ability FX + hit reaction;
- оцінювати продуктивність: overdraw, particle count, bounds/culling, LOD, shader cost;
- виконувати базову оптимізацію: зменшення шарів, спрощення матеріалів, бюджетування;
- оформлювати короткий VFX breakdown (стек емітерів, параметри, правила використання, скрін/діаграма).

Ключові терміни: VFX, particle system, emitter, spawn rate, burst, lifetime, velocity, size over life, color over life, ribbon/trail, sprite sheet, flipbook, GPU particles, Niagara, VFX Graph, module, parameter, event-driven, impact frame, anticipation, recovery, hit-stop, screen shake, camera impulse, decal, distortion,

bloom, overdraw, bounds, culling, LOD, pooling, shader complexity, readability, telegraph, feedback language

2.5.1 VFX як частина game feel: «зрозуміти -> відчути -> повірити»

VFX в іграх не існує «сам по собі»: він або підсилює дію, або створює ефект. Коли гравець взаємодіє, він очікує підтвердження: що дія спрацювала, де саме, з якою силою, і який результат. Тому VFX треба проєктувати як частину контурів відгуку: UI/VFX/Audio/камера/анімація.

Практична рамка для курсу, де кожен ефект відповідає на 4 питання

• Що сталося? (тип події: удар/підбір/активація/промах)

• Де сталося? (позиція/напрямок/зона)

• Наскільки сильно? (інтенсивність/масштаб/тривалість)

• Що робити далі? (підказка для наступної дії: cooldown, stun, зона небезпеки) (рис. 2.20)

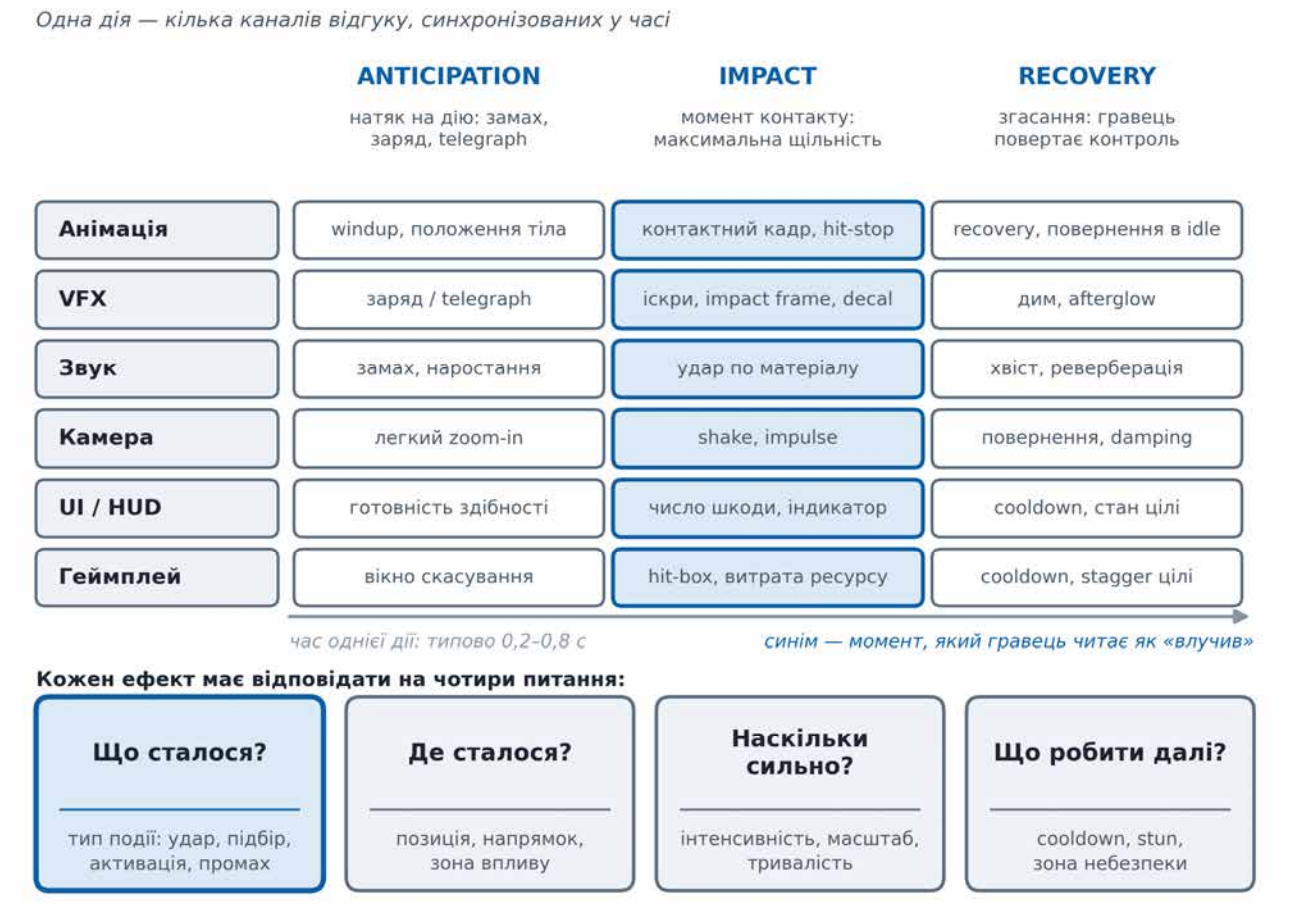


Рис. 2.20. Пакет відгуку (feedback stack) для однієї дії

[Input/Action] -> [State] -> {Animation + VFX + Audio + UI + Camera} -> [Perception] -> [Next decision]

2.5.2 Таймінг: anticipation / impact / recovery як «мова» ефектів

Класична трійка anticipation/impact/recovery описує, як гравець «зчитує» дію в часі. Навіть простий постріл має підготовку (спалах/підсвітка), момент удару (flash/hit spark), і завершення (дим/затухання) (табл. 2.27).

Таблиця 2.27

Орієнтовні таймінги (грубо, для прототипів)

Елемент	Діапазон	Навіщо	Порада
Anticipation	50-200 ms	попередження/наведення	коротко; не блокуйте читабельність сцени
Impact	1-3 кадри (16-50 ms)	пік відчуття	робіть найконтрастнішим; синхрон з SFX
Recovery	150-600 ms	затухання/шлейф	не тягніть; додавайте повітря/дим делікатно
Persistent	1-5 s	зони, аури, стат. Ефекти	обмежуйте; використовуйте LOD і culling

***Практична порада*:** у бойових іграх impact часто читається як «1 кадр істини».

2.5.3 Читабельність у русі: масштаб, контраст, силует, шарування

Читабельність – ключова вимога до VFX. Гравець дивиться не на стоп-кадр, а на сцену в русі. Тому ефект має бути зрозумілим з першого погляду і не перекривати важливу інформацію.

Правила читабельного VFX (мінімальний чек-лист)

• Силует: ефект повинен підкреслювати дію, а не перетворювати її на «білу хмару».

• Контраст: яскравість/колір/форма повинні відрізнятися від фону (тестуйте на темному й світлому тлі).

• Масштаб: FX має відповідати силі дії; «малий удар» не повинен виглядати як вибух.

• Шарування: не більше 3-5 візуальних шарів у одному моменті impact (інакше шум).

• Колірна мова: різні типи подій мають різні сімейства кольорів/форм (damage/heal/buff).

• Тривалість: коротко і по суті; залишайте сцені «дихати» (табл. 2.28).

Таблиця 2.28

«VFX мова» для прототипу (приклад системи)

Категорія	Колір/форма	Тривалість	Призначення
Hit (damage)	теплий/іскри/трикутні форми	100-250 ms	підтвердити удар
Crit	яскравіший + кільце/слеш	150-300 ms	виділити важливість
Heal	холодний/м’які частки/кола	300-800 ms	позитивний ефект
Interact	нейтральний/outline/pulse	200-500 ms	підказка взаємодії
Danger zone	червона площа/пульсація	1-5 s	телеграф небезпеки

2.5.4 Частинки як конструктор: емітери, bursts, trails, decals, distortion

Частинки – універсальний інструмент, але їх легко перевантажити. Для навчальних кейсів достатньо 4–6 модулів, які комбінуються у пакети: (1) flash/spark, (2) debris/dust, (3) trail/ribbon, (4) smoke/afterglow, (5) decal/mark, (6) distortion (опційно) (рис. 2.21).

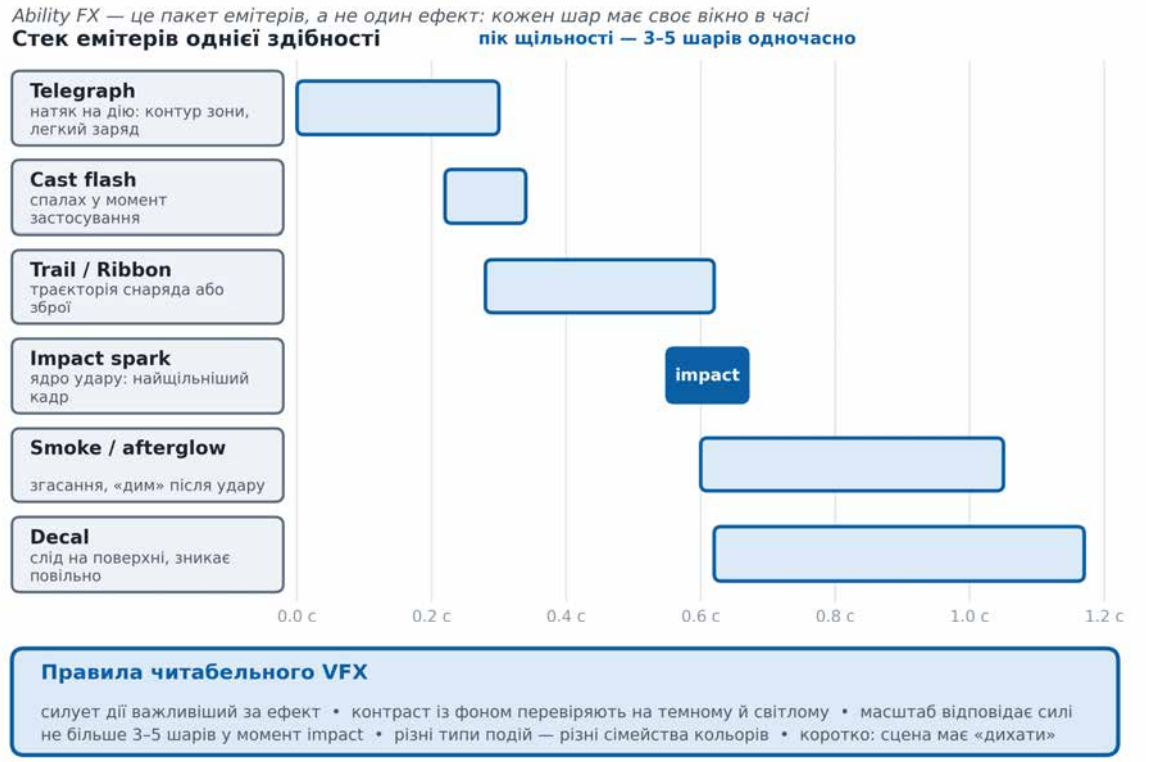


Рис. 2.21. Стек емітерів для ability FX (приклад)

AbilityFX = { Telegraph (optional) -> Cast Flash -> Trail/Ribbon -> Impact Spark -> Smoke/Afterglow -> Decal }

2.5.5 Unity VFX Graph vs UE5 Niagara: ключові концепції для старту

І Unity VFX Graph, і UE5 Niagara – node-based системи для VFX. У навчальному семестрі важливо освоїти базові принципи, а не всі модулі одразу: spawn, initialize, update, output/render (табл. 2.29).

Таблиця 2.29

Карта відповідностей (спрощено)

Концепт	Unity VFX Graph	UE5 Niagara	Коментар
Система	VFX Graph Asset	Niagara System	контейнер для емітерів
Емітер	Context/Spawner+Initialize+Update	Niagara Emitter	логіка генерації часток
Модулі	Blocks/Operators	Modules/Stacks	вузли налаштування поведінки
Параметри	Exposed Properties	User Parameters	tunables для дизайну
Події	Event context (limited)	Event Handlers	у Niagara події потужніші
Профілювання	Profiler/Frame Debugger	Niagara Debugger/Insights	оцінка вартості FX

2.5.6 Event-driven FX: запуск від подій, станів, анімаційних notifies

Production-підхід: ефект запускається не «вручну з HUD», а від конкретної події геймплею. Наприклад: OnAbilityCast, OnHitConfirmed, OnInteractSuccess. Це робить VFX масштабованим і керованим.

Джерела подій (типові)

- Input/state: натиснув кнопку, стан перейшов у Casting/Attacking.
- Animation events/notifies: момент удару або контакт з землею.
- Collision/overlap: потрапляння/вхід у зону.
- Gameplay events: крит, блок, промах, статус-ефект (рис. 2.22).

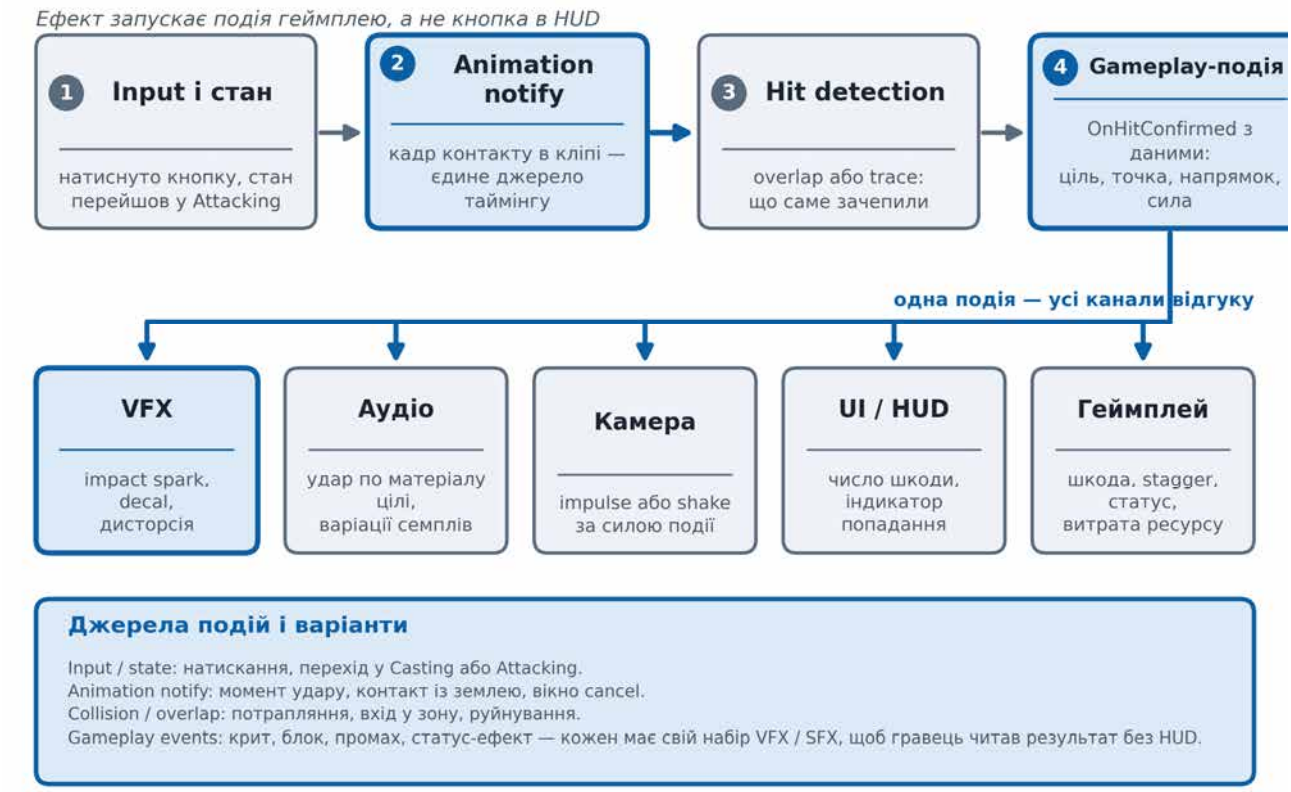


Рис. 2.22. Подієвий ланцюг для удару (узагальнено)

Input Attack -> State Attacking -> Anim Notify ‘ImpactFrame’ -> Spawn VFX + Play SFX + Camera impulse
-> Apply damage -> (if crit) Spawn Crit VFX

2.5.7 Підсилення impact: hit-stop, камера, екранні ефекти (без зловживання)

Багато «відчуття» виникає не з частинок, а з синхронізації: коротка пауза (hit-stop), мікро-шейк камери, екранний флеш або хроматична аберація (дуже обережно). Це має бути дозовано і залежати від інтенсивності події (табл. 2.30).

Таблиця 2.30

Підсилювачі impact і коли їх застосовувати

Техніка	Ефект	Ризик	Рекомендація
Hit-stop (20-80 ms)	відчуття сили	може дратувати	тільки на важливих ударах/критах
Camera impulse	кінетика/вага	motion sickness	малий імплітудний імпульс, clamp
Screen flash	підтвердження	пересвіт	коротко, низька інтенсивність
Slow motion (0.1-0.3 s)	драма/кульмінація	ламає темп	дуже рідко, як нагорода

2.5.8 Бюджети та продуктивність: overdraw, bounds, culling, LOD, pooling

VFX найчастіше «вбиває» продуктивність через прозорість (overdraw) і надмірну кількість часток. Тому навіть у студентському прототипі вводимо прості бюджети: скільки часток одночасно, скільки шарів прозорості, скільки активних систем у кадрі.

Практичні бюджети для вертикального зрізу (орієнтовно)

1 ability FX у кадрі: 1-3 емітери impact + 1 trail + 1 afterglow (без «салюта»).

Макс. 2-3 активні великі ефекти одночасно (інакше перевіряйте FPS).

Уникайте великих прозорих площин перед камерою; робіть їх короткими або masked. Контролюйте bounds (інакше culling працюватиме погано) та використовуйте LOD залежно від відстані.

Pooling: не створюйте/не знищуйте системи щокадру; використовуйте пул об’єктів (табл. 2.31).

Таблиця 2.31

Швидкі «діагнози» проблем VFX

Симптом	Ймовірна причина	Що перевірити	Швидке рішення
FPS падає в FX сцені	overdraw/прозорість	shader complexity/overdraw view	менше шарів, masked, менший розмір
FX видно «звідусіль»	погані bounds	culling/debug bounds	налаштуйте bounds або fixed bounds
Ефект «шумить»	забагато шарів	скрін/відео у русі	зменшити емітери, підняти контраст ключових
Непередбачуваний запуск	FX не від події	лог подій/нотіфаїв	перейти на event-driven запуск

2.5.9 Workflow: VFX Breakdown, пресети, параметризація, контроль версій

VFX потрібно робити як «пакет», а не як один емітер. Мінімальний VFX breakdown має містити

• Список емітерів (Telegraph/Cast/Impact/Trail/Afterglow/Decal).

• Параметри (інтенсивність, масштаб, кольорова тема, тривалість, кількість часток).

• Правила використання (на які події під’єднувати, як масштабувати під силу).

• 2-3 пресети (наприклад: light/medium/heavy або fire/ice/arcane).

• Тест-сцена та відео демонстрації у русі (рис. 2.23).



Рис. 2.23. Приклад параметризації (tunables)

Intensity (0..1)
Scale (0.5..2.0)
ColorTheme (Fire/Ice/Arcane)
ImpactBrightness (0..5)
TrailLength (0..1)
ParticleBudget (Low/Med/High)

2.5.10 Production-ready FX для вертикального зрізу: критерії готовності

Стислий чек-лист готовності VFX для інтегрованого проєкту

Ефект читається за 1-2 секунди і не «заливає» екран.

Є чіткий таймінг anticipation/impact/recovery; impact синхронізований зі SFX/анімацією.

FX запускається від події (input/state/notify/hit), а не «вручну» в UI».

Є параметри і пресети; один VFX asset може обслуговувати кілька варіантів. Перевірено продуктивність (overdraw/particle count) і зроблено хоча б 1 оптимізацію.

Є короткий breakdown + тест-сцена + 30-60 с демонстрація.

Питання для самоперевірки

1. Що таке VFX у контексті game feel і чому він не може існувати «окремо від геймплею»?
2. Поясніть модель anticipation/impact/recovery на прикладі удару або здібності.
3. Які 3-5 правил забезпечують читабельність ефектів у русі?
4. Чому шарування прозорих частинок часто створює проблеми з FPS (overdraw)?
5. Назвіть базовий стек емітерів для ability FX (telegraph/cast/impact/trail/afterglow/decal).
6. У чому різниця між VFX Graph і Niagara на рівні концепцій?
7. Що означає event-driven FX і які джерела подій найчастіші?
8. Коли доречно застосовувати hit-stop і чому ним легко зловживати?
9. Що таке bounds/culling і як неправильні bounds можуть «ламати» продуктивність або видимість?
10. Навіщо потрібні пресети та параметризація ефектів?
11. Назвіть 3 «діагнози» проблем VFX і швидкі рішення (табл. 6.5).
12. Які критерії «production-ready» VFX для вертикального зрізу?

Практичний кейс (120-180 хв): «Ability FX + Hit Reaction» (Niagara / VFX Graph)

Мета: створити один повноцінний FX-пакет для здібності (ability) з чітким telegraph -> cast -> impact -> recovery, під’єднати запуск до подій (input/state/animation notify або hit event), і продемонструвати читабельність та мінімальну оптимізацію. Пакет має включати trail/ribbon і коротку реакцію на влучання (hit spark + післясвітіння).

Сценарії (оберіть один або реалізуйте 2 для бонусу)

• Сценарій A (Melee slash): коротка атака з trail на зброї + impact spark + dust.

• Сценарій B (Projectile): постріл/снаряд з trail + hit impact + decal/mark на поверхні.

• Сценарій C (AOE zone): телеграф у колі (danger zone) + вибух + afterglow на землі.

Кроки виконання завдання

Задайте таймінг: anticipation (телеграф) -> impact (пік) -> recovery (затухання).

• Зберіть стек емітерів: Telegraph/CastFlash/Trail/Ribbon/Impact/ Smoke/ Afterglow/Decal (за потреби).

Під'єднайте запуск: input -> state -> (notify/hit) -> spawn FX; для projectile – on collision.

• Зробіть мінімальну параметризацію: Intensity, Scale, ColorTheme, TrailLength.

• Перевірте читабельність на 2 фонах (світлий/темний) і в русі камери.

• Перевірте продуктивність (overdraw/particle count). Зробіть 1 оптимізацію і зафіксуйте зміни.

• Оформіть VFX breakdown: список емітерів + параметри + пресети + правила використання.

Що необхідно здати

• Проєкт/сцена Unity або UE5 з FX пакетом і тестом (1-2 сценарії).

Відео 30-60 с: telegraph -> cast -> impact -> recovery + hit reaction у русі.

• VFX breakdown (1-2 стор.): емітери, таймінг, параметри, 2 пресети.

• Короткий постмортем (150-250 слів): що зробили для читабельності та продуктивності.

Критерії оцінювання

• Читабельність: ефект зрозумілий і не «заливає» екран.

• Таймінг: чітко видно anticipation/impact/recovery; impact синхронний з подією.

• Архітектура: запуск від подій (event-driven), а не ручні тригери без логіки.

• Якість FX пакета: є trail/ribbon, є hit reaction, є afterglow/decay.

• Продуктивність: проведено перевірку та виконано щонайменше 1 оптимізацію.

• Документація: breakdown зрозумілий, параметри описані, є пресети.

Рекомендовані програми та сервіси

1. Unity LTS: VFX Graph (URP/HDRP), Particle System (для простих задач), Profiler, Frame Debugger.
2. Unreal Engine 5: Niagara System/Emitter, Niagara Debugger, Shader Complexity view, Unreal Insights.
3. Матеріали/текстури для FX: flipbooks (Kenney/CC0), власні спрайти (Krita/GIMP), noise textures (Poly Haven).
4. Захоплення відео: OBS, Unity Recorder, UE Movie Render Queue (MRQ).
5. Документація: Notion/Google Docs; diagrams.net для простих схем.

Рекомендовані джерела та матеріали

Офіційна документація Unity щодо VFX Graph/Particle System та профілювання; документація Unreal Engine 5 щодо Niagara, матеріалів і перегляду Shader Complexity. Для практики VFX-мови і читабельності корисні розбори GDC/80 Level про «VFX readability», а також аналізи ефектів у Destiny 2 та League of Legends як прикладів чіткої VFX-семантики.

Тема 2.6. Lighting, Post-FX & Mood: readability, exposure, fog та color grading

Мета теми: сформувати у студентів практичне розуміння світла як інструмента геймдизайну (читабельність, навігація, темп, напруга), навчити виконувати mood pass та readability pass для ігрової сцени. Вміти використовувати постобробку (exposure, tone mapping, color grading, bloom, DOF, vignette), атмосферу (fog/volumetrics), і документувати рішення як частину production-пайплайну. Результатом є production-ready lighting setup для невеликої локації/арени, інтегрований з gameplay.

Очікувані результати навчання

- Вміти пояснювати, як світло керує увагою, настроєм і читабельністю;
- розділяти задачі mood pass і readability pass та виконувати їх поетапно;
- налаштовувати базову світлову схему (key/fill/rim; practicals; emissives);
- керувати exposure і уникати «washed out»/«crushed blacks»;
- застосовувати fog/volumetrics для глибини і навігації без втрати читабельності;
- налаштовувати color grading (LUT/curves/temperature) під цільовий стиль;
- використовувати post-FX з бюджетом продуктивності (bloom/DOF/SSR/film grain);
- оцінювати та фіксувати legibility under motion (камера в русі, бої, VFX);
- профілювати та оптимізувати lighting (baked vs dynamic; Lumen vs baked; shadow budgets);
- оформлювати lighting breakdown і технічні налаштування сцени.

Ключові терміни: Key light, fill light, rim light, practical lights, exposure (auto/manual), EV, tone mapping, ACES, HDR, color grading, LUT, temperature/tint, saturation, contrast, bloom, vignette, chromatic aberration, depth of field (DOF), motion blur, fog, volumetrics, god rays, shadow cascades, lightmap, baked lighting, GI, Lumen, reflection captures, probe, post-process volume, render pipeline (URP/HDRP), performance budget.

2.6.1 Світло як геймдизайн: навігація, читабельність, ритм сцени

Світло — це система навігації й «режисура» для гравця. Воно виконує наступне

- підсвічує цілі та шляхи;
- створює контраст між інтерактивним/неінтерактивним;
- задає емоційний тон (спокій/тривога/небезпека);
- підтримує ритм (quiet -> tension -> release) (табл. 2.32).

Таблиця 2.32

Функції світла у геймплеї

Функція	Як робиться	Приклад
Навігація	«світловий коридор», маяк, контраст	вихід/двері підсвічені теплим
Читабельність	силуети, rim light, обмеження шуму	ворог читається на фоні
Атмосфера	fog/volumetrics, колір, практичні джерела	ангар/коридор/ніч
Темп	зміна яскравості/кольору по зонах	«безпечна» зона світліша

Кейс-референси: Inside (контроль силуетів), Control (атмосфера+контраст), кіно Mad Max: Fury Road (color scripting).

2.6.2 Два проходи: Mood pass vs Readability pass

Mood pass – це підхід до створення настрою та художньої ідентичності. Readability pass – забезпечити, щоб у русі/бої все було читабельним (рис. 2.24).

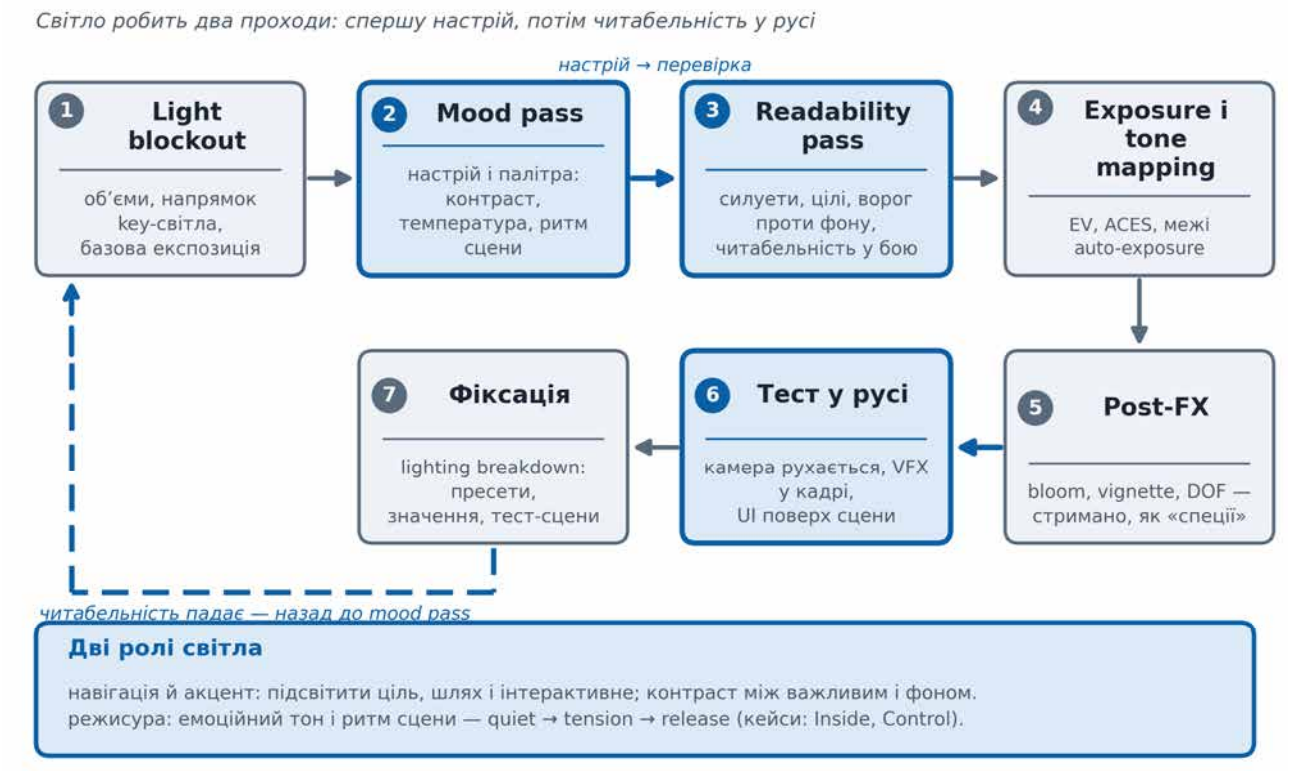


Рис. 2.24. Поетапний пайплайн

Blockout -> Base Lighting -> Mood Pass -> Readability Pass -> Polish -> Profile/Optimize (табл. 2.33)

Таблиця 2.33

Ознаки «готовності» кожного проходу

Прохід	Вище пріоритет	«Готово», коли…
Mood pass	настрій/історія/стиль	є цільова палітра і ключові акценти
Readability pass	геймплей/силуети	вороги/цілі/шлях читаються в русі

2.6.3 Базова світлова схема: key/fill/rim + practicals + emissives

Навіть у stylized сцені працюють наступні базові принципи

• Key - головне джерело (сонце/лампа/портал)
• Fill - заповнення, щоб не «провалювались» деталі
• Rim - контур для силуетів
• Practicals - «реальні» лампи/неон/екрани
• Emissives - контрольований «самосвіт» (табл. 2.34)

Таблиця 2.34

Практичні правила (швидко)

Правило	Навіщо	Помилка
1 ключовий напрям	узгодженість тіней	«хаос світла»
Rim на гравця/ворога	силуети	персонаж зливається
Practicals як «маяки»	навігація	всі лампи однакові

2.6.4 Exposure & Tone Mapping: уникнути «washed out» і «crushed blacks»

Експозиція - це часта причина «поганої» картинки

- auto exposure може «плавати» і руйнувати контроль;
- Надто темно - втрата деталей; надто світло - «мило» (табл. 2.35; рис. 2.25).

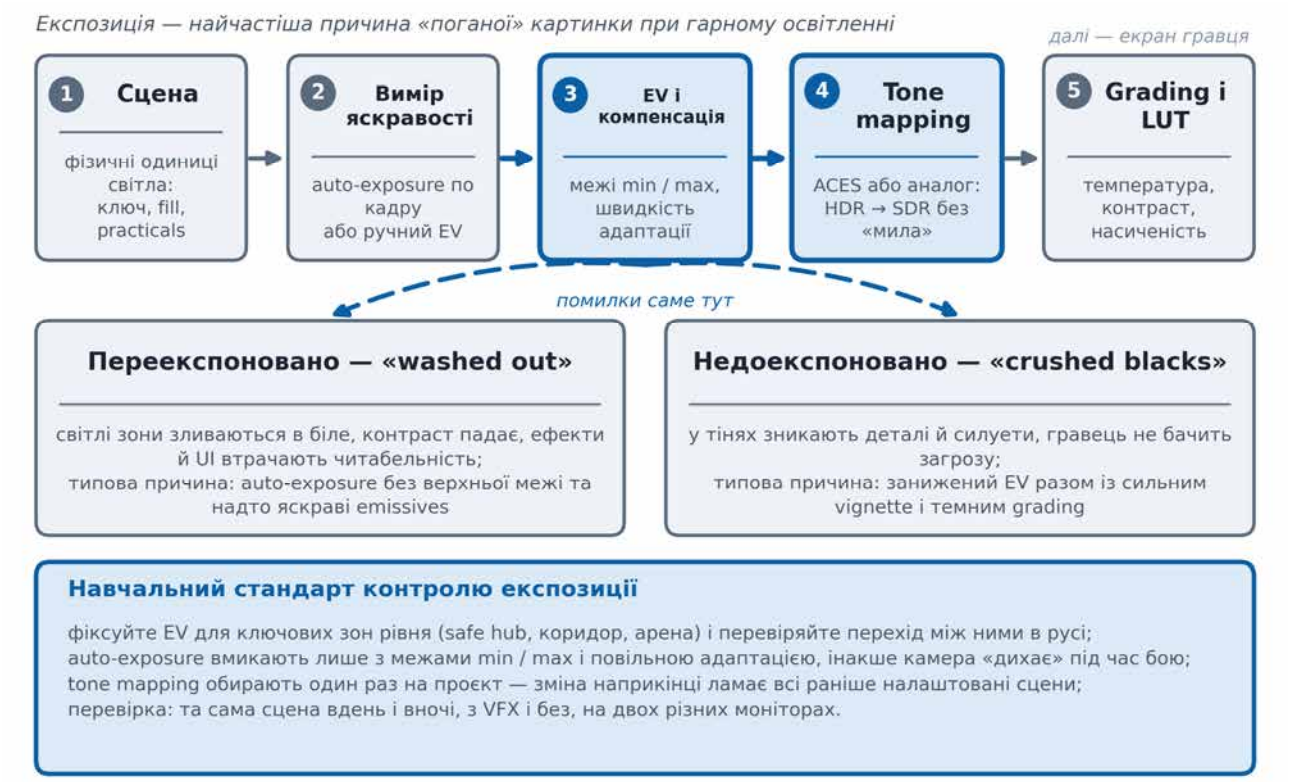


Рис. 2.25. EV логіка (узагальнено)

Таблиця 2.35

Контроль експозиції (навчальний стандарт)

Підхід	Коли	Рекомендація
Manual exposure	арена/сцена з фіксованим світлом	стабільна читабельність
Auto exposure	відкриті простори/різкі переходи	обмежити діапазон + плавність

Too bright -> lower exposure / reduce bloom / clamp highlights
Too dark -> lift shadows / add fill / reduce contrast

2.6.5 Fog & Volumetrics: глибина, масштаб, «повітря», але без втрати читабельності

-глибину (атмосферна перспектива),
-акценти (god rays),
-настрій (пил/дим/туман).
Але fog легко «вбиває» читабельність (табл. 2.36).

Таблиця 2.36

Fog правила безпечного використання

Правило	Навіщо	Помилка
менше fog біля гравця	читабельність HUD/VFX	«молоко» в кадрі
fog по дальності	глибина	все однаково сіро
локальні volumetrics	акцент	глобальна густа завіса

2.6.6 Color Grading: палітра, температура, LUT/curves, стилізація

Color grading – це «підпис» кожної сцени

температура (тепло/холод), контраст (драма), насиченість (stylized/realistic), виділення акцентних кольорів (цілі/небезпека) (табл. 2.37) (рис. 2.26).

Таблиця 2.37

«Колір як навігація»

Елемент	Колір	Сенс
шлях/вихід	теплий/жовтий	безпечно/ціль
небезпека	червоний	шкода/загроза
інтеракція	синій/бірюза	активний об’єкт

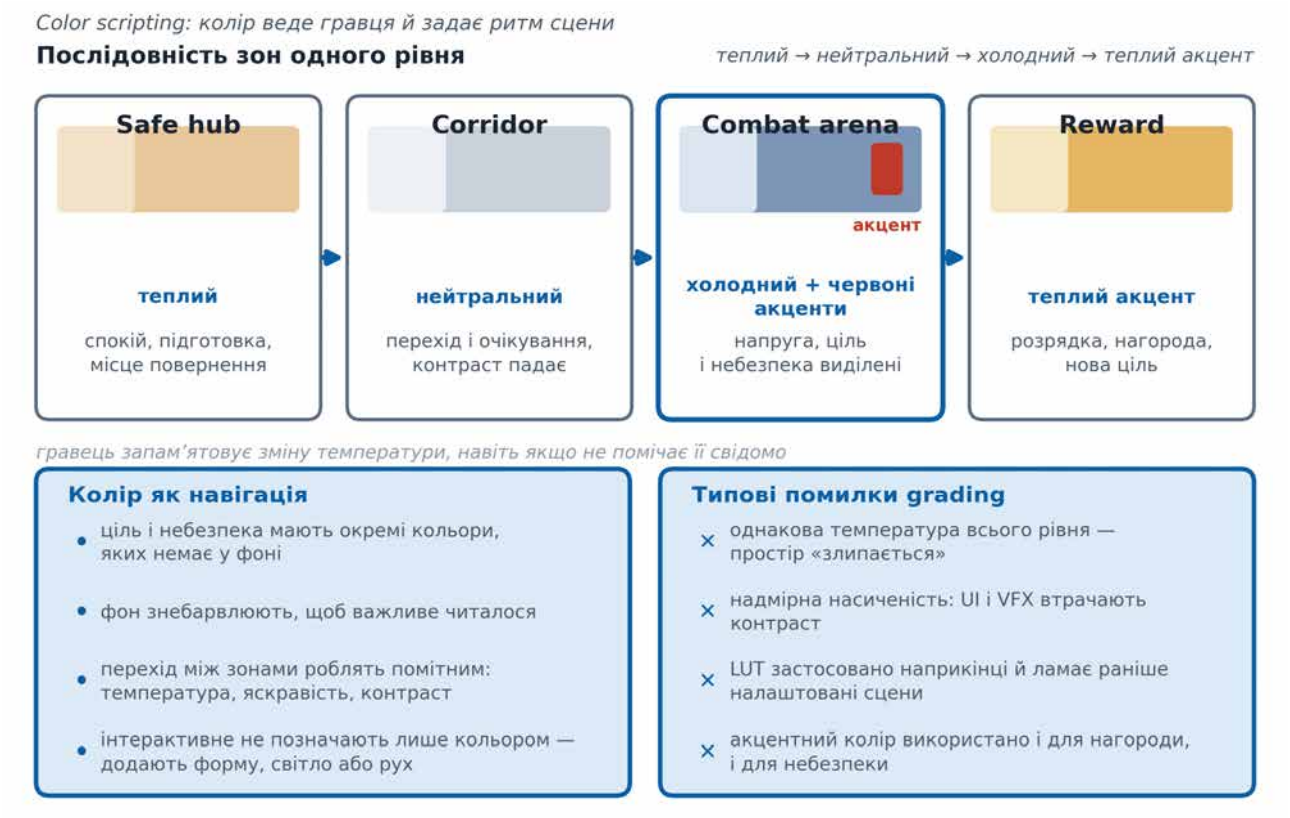


Рис. 2.26. Color scripting по зонах (приклад)

Safe hub (warm) -> Corridor (neutral) -> Combat arena (cold + red accents) -> Reward (warm highlight)

2.6.7 Post-FX набір для ігор: bloom, vignette, DOF, motion blur – коли і навіщо

Post-FX – це «спеції». Якщо переборщити, то UX буде поганим (табл. 2.38).

Таблиця 2.38

Доречність post-FX

Ефект	Дає	Використовувати	Ризик
Bloom	«світіння», стиль	неон/емісиви	«пересвіт», мило
Vignette	фокус уваги	хоррор/напруга	«тунельний» зір
DOF	кінематографічність	катсцени	в геймплеї шкодить
Motion blur	відчуття швидкості	авто/екшен	нудота/a11y

2.6.8 Readability under motion: тестування у русі, в бою, з VFX

Світло має працювати не в скріншоті, а в геймплеї

камера рухається -> силуети мають читатися;

VFX додають шум -> потрібно обмежити фон/блум;

• UI має бути контрастним до сцени (табл. 2.39; рис. 2.27).

Таблиця 2.39

Тест-сцени для читабельності

Тест	Як	Що перевіряємо
Біг по маршруту	20-30 с	шлях/перешкоди
Бій 3-5 ворогів	30-60 с	силуети/хіти
Ефекти здібностей	spam VFX	«каша» чи ні

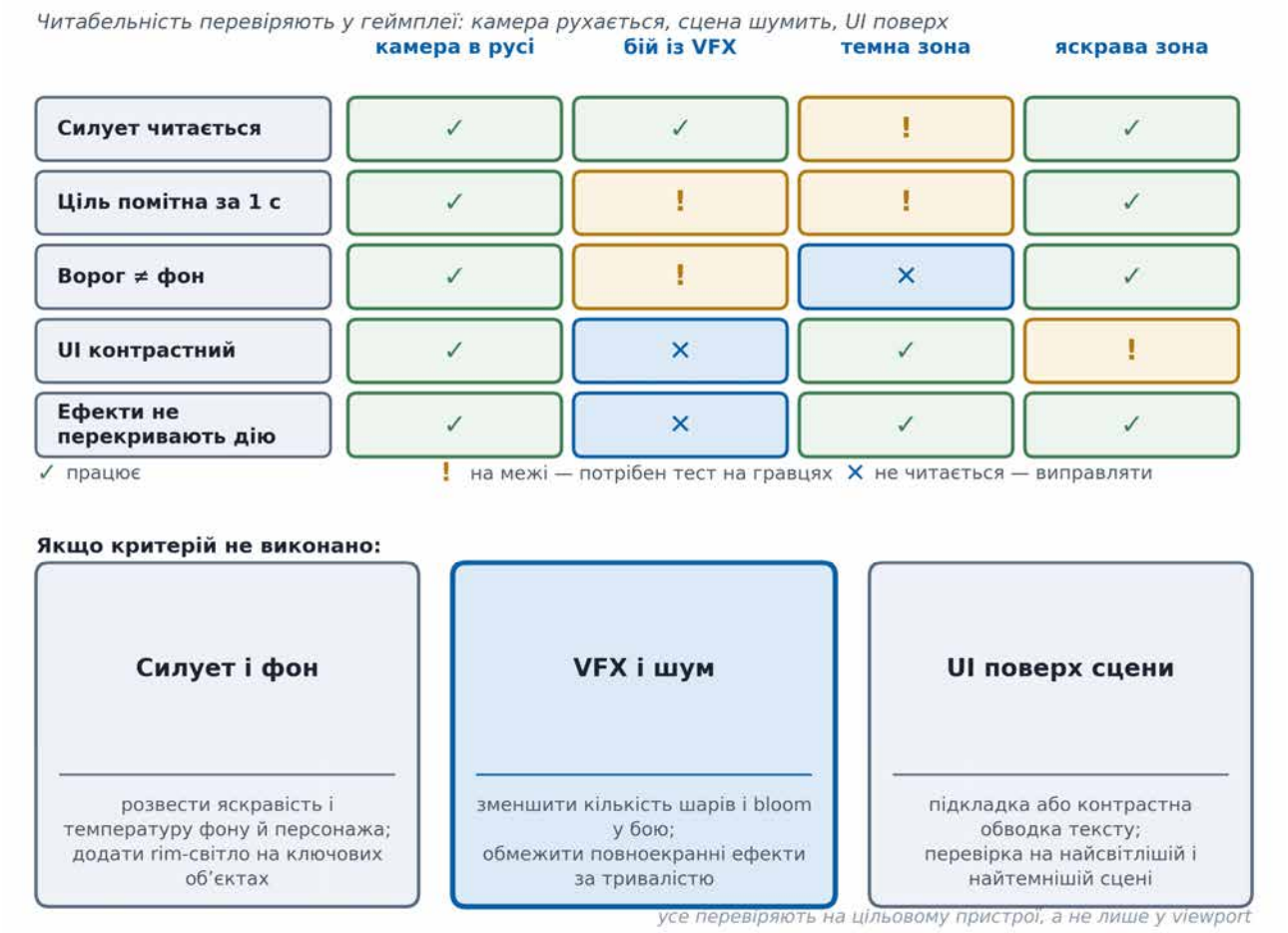


Рис. 2.27. Readability checklist (спрощено)

Silhouette readable? -> Yes/No
Target highlight clear? -> Yes/No
Gameplay cues > mood noise? -> Yes/No

2.6.9 Performance & Optimization: baked vs dynamic, shadow budgets, Lumen/URP/HDRP

Світло дороге тому використовуємо навчальний принцип

-спочатку сцена працює,

-потім оптимізуємо, зберігаючи вигляд (табл. 2.40).

Таблиця 2.40

Типові важкі речі

Джерело навантаження	Симптом	Рішення
багато dynamic lights	FPS падає	обмежити, bake, cull
високі тіні	spikes	shadow distance/cascades
volumetrics всюди	GPU «горить»	локально, менше якості
постефекти	blur/bloom	зменшити інтенсивність/quality

Корисна порада: у UE5 для прототипу часто ок, але Lumen + Volumetrics + багато lights - це ризик.

2.6.10 Production-ready lighting: чек-лист готовності + документація

Lighting готовий для vertical slice, якщо є

mood (палітра/настрій) і він стабільний;
readability: ворог/ціль/шлях читаються в русі;
експозиція не «плаває» або контрольована;
fog не перетворює сцену на «молоко»;
post-FX не шкодить UX (особливо DOF/motion blur);
є профілювання: ключові проблеми виправлені;
оформлено lighting breakdown.

Питання для самоперевірки

1. Які геймдизайнерські функції світла?
2. Чим відрізняється mood pass від readability pass?
3. Опишіть key/fill/rim і де вони застосовуються в грі.
4. Які проблеми створює auto exposure і як їх обмежити?
5. Як fog може покращити глибину і як - зруйнувати читабельність?
6. Як працює color grading як інструмент навігації?
7. Які post-FX найчастіше шкодять UX і чому?
8. Чому «скріншот красивий» ≠ «геймплей читається»?
9. Які 3 тест-сценарії треба прогнати для readability under motion?
10. Назвіть 3 ключові джерела падіння FPS через lighting.
11. Які компроміси baked vs dynamic lighting?
12. Назвіть 5 пунктів production-ready чек-листа.

Практичний кейс до теми(час виконання: 120-180 хв)

«Mood + Readability Lighting Pass для арени/кімнати»

Завдання: зробити lighting для маленької локації (кімната/коридор/арена) у Unity (URP/HDRP) або UE5 так, щоб:

• сцена мала чіткий настрій,

• геймплей був читабельний в русі.

Вхідні умови

• Blockout/сцена: 1 кімната + 1 коридор (або арена 20×20 м).

• Є 1-2 інтерактивні об’єкти + 1 «ціль/вихід».

• Є хоча б 1 VFX (з попередніх тем) або простий тест-спел.

Сценарії (A/B/C - оберіть один)

• A (Horror/Тривога): холодна палітра + теплі «маяки», туман/пил.

• B (Sci-fi/Неон): emissives + bloom, контроль експозиції.

• C (Natural/Daylight): сонце + fill, м’яка атмосфера, читабельні силуети.

Кроки виконання завдання

• Base lighting: 1 key + 1-2 practicals, без постефектів.

• Mood pass: палітра, fog/volumetrics (обережно), акценти (маяки).

• Exposure control: manual або auto з обмеженням діапазону + плавність.

• Color grading: температура/контраст/насиченість, виділення цілі/шляху.

• Post-FX мінімум: bloom/vignette (за потреби), без «перебору».

• Readability pass: rim на персонажа/ворога, контраст цілей, приглушити шум.

• Readability tests: маршрут 20-30 с + бій/рух + VFX spam.

• Performance pass: shadow distance, кількість dynamic lights, volumetrics quality.

• Документація: 6-10 скрінів «до/після» + таблиця налаштувань.

Що необхідно здати

• Проєкт сцени (Unity/UE5) з lighting setup.

• Відео 45-90 с: прогін маршруту + демонстрація цілі/інтеракції.

• Lighting breakdown (1-2 стор.):

• палітра/настрій,

• ключові джерела світла (таблично),

• exposure, fog, post-FX,

• список оптимізацій.

• 2 скріншоти «Mood» + 2 скріншоти «Readability» (у русі - кадри).

Критерії оцінювання

• Mood: настрій чіткий і стабільний, є художня логіка.

• Readability: шлях/ціль/силуети читаються в русі.

• Exposure: немає «плаваючої» картинки або вона контрольована.

• Post-FX: доречні, не шкодять UX.

• Performance: є хоча б 2 оптимізації з відчутним ефектом.

• Документація: зрозуміло, що і чому налаштовано.

Рекомендовані програми та сервіси

1. Unity URP/HDRP: Volume system, Color Adjustments, Bloom, Fog, Post-processing, Lightmapping.
2. UE5: Post Process Volume, Lumen, Exponential Height Fog, Volumetric Fog, Color Grading, Reflection Captures.
3. Профілювання: Unity Profiler/Frame Debugger; UE Stat commands/Unreal Insights.

4. Референси: PureRef (для moodboard), Pinterest/ArtStation (обережно з ліцензіями).

Рекомендовані джерела та матеріали

1. Unity Manual: URP/HDRP lighting, Volumes, post-processing, lightmapping.
2. Unreal Engine Docs: Lumen, Post Process Volume, fog/volumetrics, shadow optimization.
3. GDC/NoClip: розбори art direction і lighting pass.
4. Кіно/фото-референси: color scripting (Mad Max: Fury Road), силуети (Inside), атмосфера (Control).

Тема 2.7. Materials & Shaders II: full-screen FX, distortion, outlines, SDF та stylized motion

Мета теми: поглибити практичні навички студентів у створенні просунутих матеріалів і шейдерних ефектів, що безпосередньо впливають на game feel, читабельність та стиль: full-screen/post FX, distortion/heat haze, toon outlines, SDF-ефекти, stylized motion (halftone, posterization, linework). Навчити обирати правильний підхід (material vs post-process), будувати ефекти event-driven (від gameplay подій), контролювати параметри (tunables), та профілювати шейдери й бюджет рендеру. Результат: 1-2 advanced shader ефекти, інтегровані у прототип/vertical slice.

Очікувані результати навчання

- вміти пояснювати різницю між матеріальними та екранними (full-screen) ефектами;
- реалізувати distortion (refraction/screen-space) для VFX та ударів;
- створити toon outline (edge detection або normal-based) та керувати товщиною/маскою;
- застосувати SDF (signed distance field) для UI/масок/плавних форм/контурів;
- зробити posterization/halftone/scanline як стилізацію або «стан» (low HP);
- прив’язувати шейдери до подій через параметри (damage flash, freeze, slow);
- організувати tunables: intensity, duration, falloff, thresholds;
- уникати типових артефактів: shimmer, banding, jaggies, ghosting;
- профілювати шейдери: overdraw, instruction count, render passes;
- документувати shader breakdown: граф/нод-лист/параметри/призначення.

Ключові терміни: Full-screen effect, post-process, screen-space, UV, normal, depth, edge detection, Sobel, outlines, toon shading, ramp, posterization, halftone, dithering, refraction, distortion, chromatic aberration, SDF (signed distance field), smoothstep, threshold, mask, blend, render pass, shader variants, overdraw, GPU cost, MRT, stencil, custom depth, render feature, material instance, parameter collection.

2.7.1 Advanced shaders у production: коли потрібні і як не «вбити» продуктивність

Просунутий шейдер у першу чергу має бути функціональним

• підсилює читабельність (outline, highlight, damage flash);
• підтримує mood/стиль (halftone, toon);
• додає «удар» від дії (distortion, hit flash);

Не розростається в хаос: 1 ефект -> 1 задача -> контроль параметрів (табл. 2.41).

Таблиця 2.41

Типові цілі advanced shaders

Ціль	Ефект	Де застосувати
Читабельність силуетів	outline	вороги/інтерактивні
Відчуття удару	distortion + flash	melee/projectile hits
Стан гравця	full-screen tint/halftone	low HP, poison, slow
Стиль	posterize/toon	stylized проєкти

2.7.2 Material vs Full-screen FX: правильний вибір

Зазначимо, що інколи ефект краще зробити матеріалом (локально), інколи - екранним (глобально) (табл. 2.42; рис. 2.28).

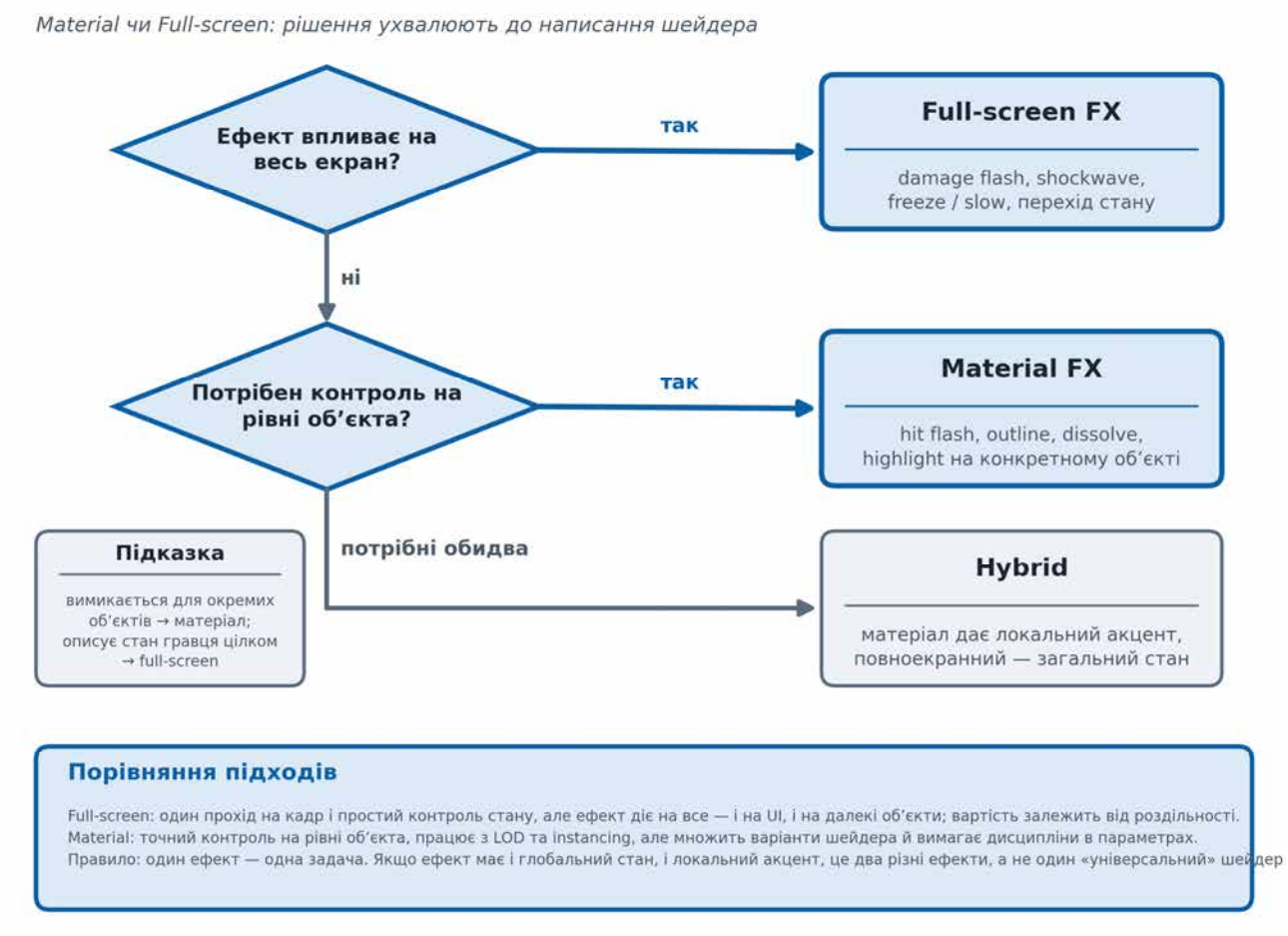


Рис. 2.28. Decision flow (спрощено)

Effect affects whole screen? -> Yes: Full-screen
Needs per-object control? -> Yes: Material
Needs both? -> Hybrid (material + post)

Таблиця 2.42

Порівняння підходів

Параметр	Material (локально)	Full-screen (глобально)
Зона впливу	окремі об’єкти	весь кадр
Продуктивність	залежить від кількості об’єктів	залежить від роздільності/пасів
Типові кейси	outline на ворога, dissolve	damage flash, shockwave, vignette
Ризики	overdraw, варіанти	banding, blur, постійна GPU вартість

2.7.3 Full-screen FX: damage flash, shockwave, freeze/slow states

Повний екран(Full-screen FX) часто прив’язують до станів

«отримав шкоду» -> короткий flash + vignette;
«вибух» -> shockwave ring + distortion;
«заморозка/slow» -> синій tint + posterize/edge (рис. 2.29).

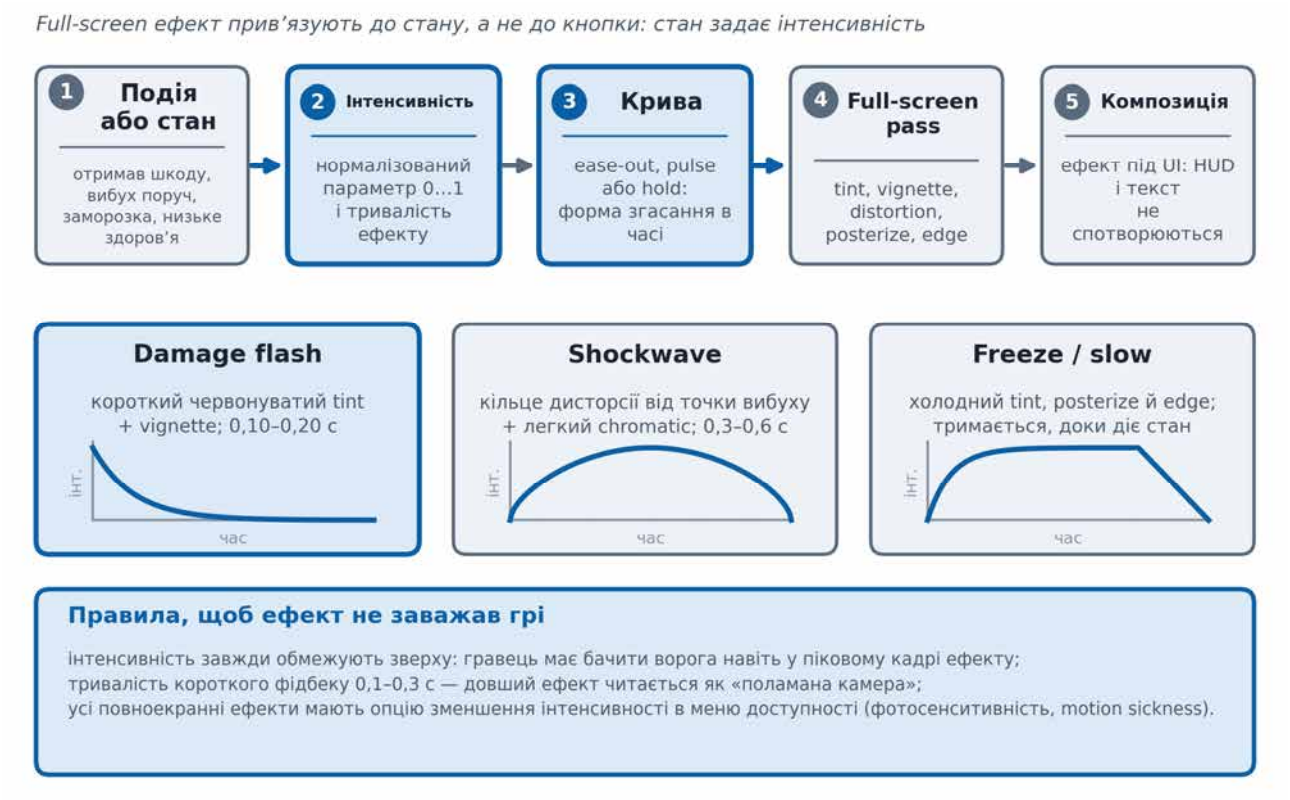


Рис. 2.29. Шаблон full-screen ефекту

Color grade / tint + mask (radial/noise) + time curve -> blend to screen (табл. 2.43)

Таблиця 2.43

Tunables

Параметр	Сенс	Типове значення
Intensity	сила ефекту	0..1
Duration	тривалість	0.1-0.6 s
Falloff	затухання	smoothstep
Center	центр удару	screen UV / world-to-screen

2.7.4 Distortion/Refraction: heat haze, ударні хвилі, магія

Distortion підсилює impact, але легко дає артефакти.

Підходи, які необхідно використовувати

• screen-space UV offset (найчастіше),
• refraction через normal map/noise,
• radial shockwave (табл. 2.44; рис. 2.30).

Таблиця 2.44

Правила distortion без артефактів

Правило	Навіщо	Помилка
шум/нормалі дрібного масштабу	органічність	«пластикова» хвиля
clamp інтенсивність	стабільність	«ламання» кадру
короткий час	читабельність	distortion «вічний»

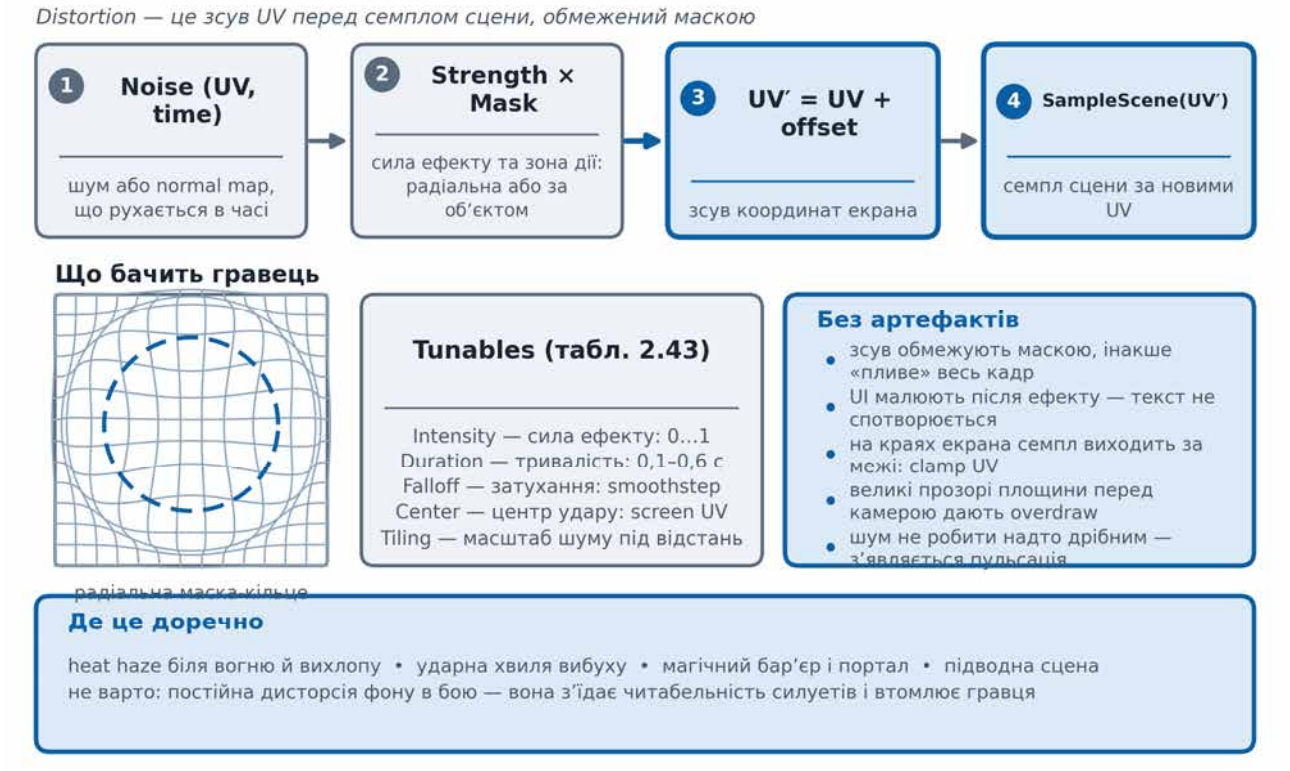


Рис. 2.30. Distortion формула (узагальнено)

UV’ = UV + (Noise(UV, t) * Strength * Mask)
Color = SampleScene(UV’)

2.7.5 Toon outlines: edge detection vs normal-based vs custom depth

Outline – ключовий ефект читабельності в stylized і навіть realistic (акцент) (табл. 2.45; рис. 2.31).

Таблиця 2.45

Види outline

Тип	Як працює	Плюси	Мінуси
Edge detection (Sobel)	по depth/normal	універсально	може «шуміти»
Normal-based	по різниці нормалей	стабільніше	залежить від геометрії
Custom depth/stencil	маркуємо об’єкти	контрольовано	складніше налаштування

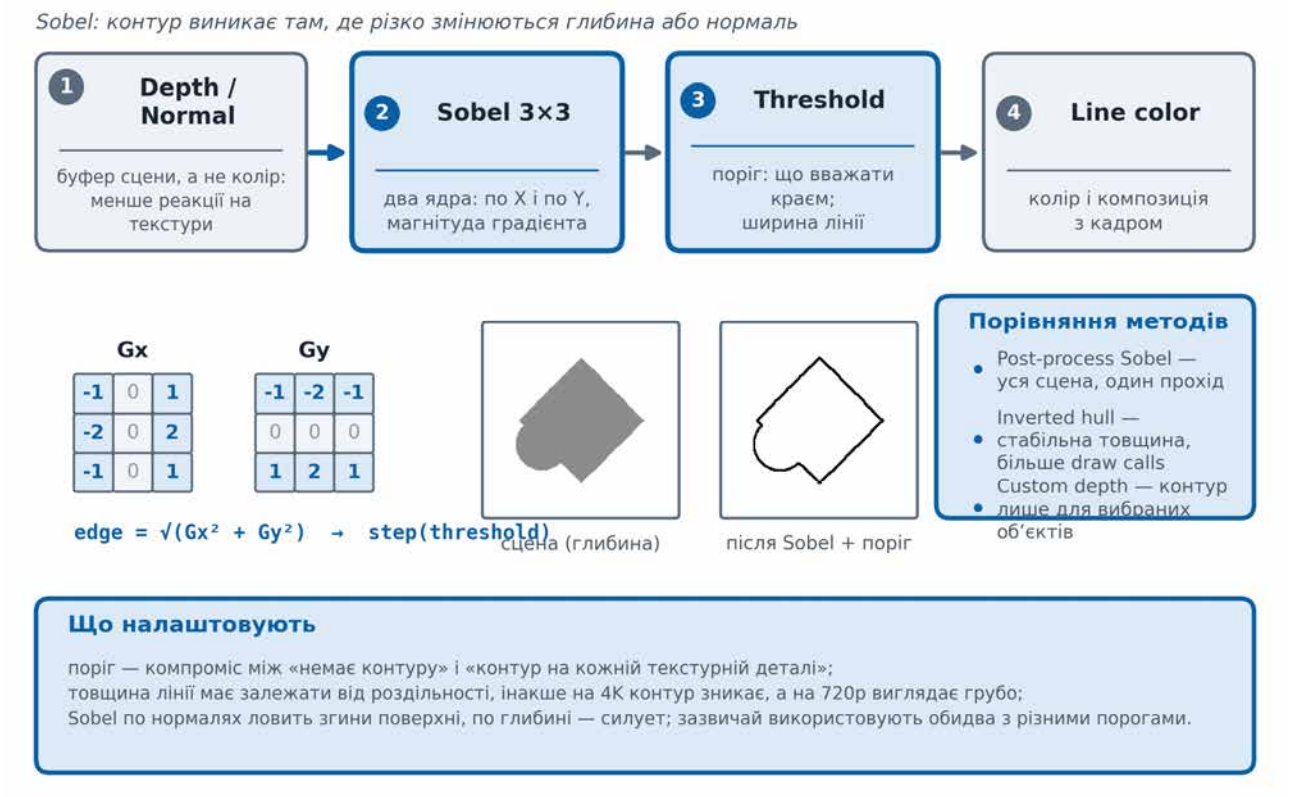


Рис. 2.31. Sobel outline (спрощено)

Edge = Sobel(depth/normal) -> threshold -> line color (табл. 2.46; рис. 2.32)

2.7.6 SDF: ідеальний інструмент для UI/масок/контурів/глоу

SDF дає: плавні контури незалежно від масштабу; прості маски (round rect, circle) та outline + glow + soft shadow

Таблиця 2.46

SDF «рецепти»

Ефект	Формула	Де застосувати
Контур	smoothstep(d)	UI, маркери
Glow	exp(-k*d)	highlight, pick-up
Маска	step(d)	health bar, рамки

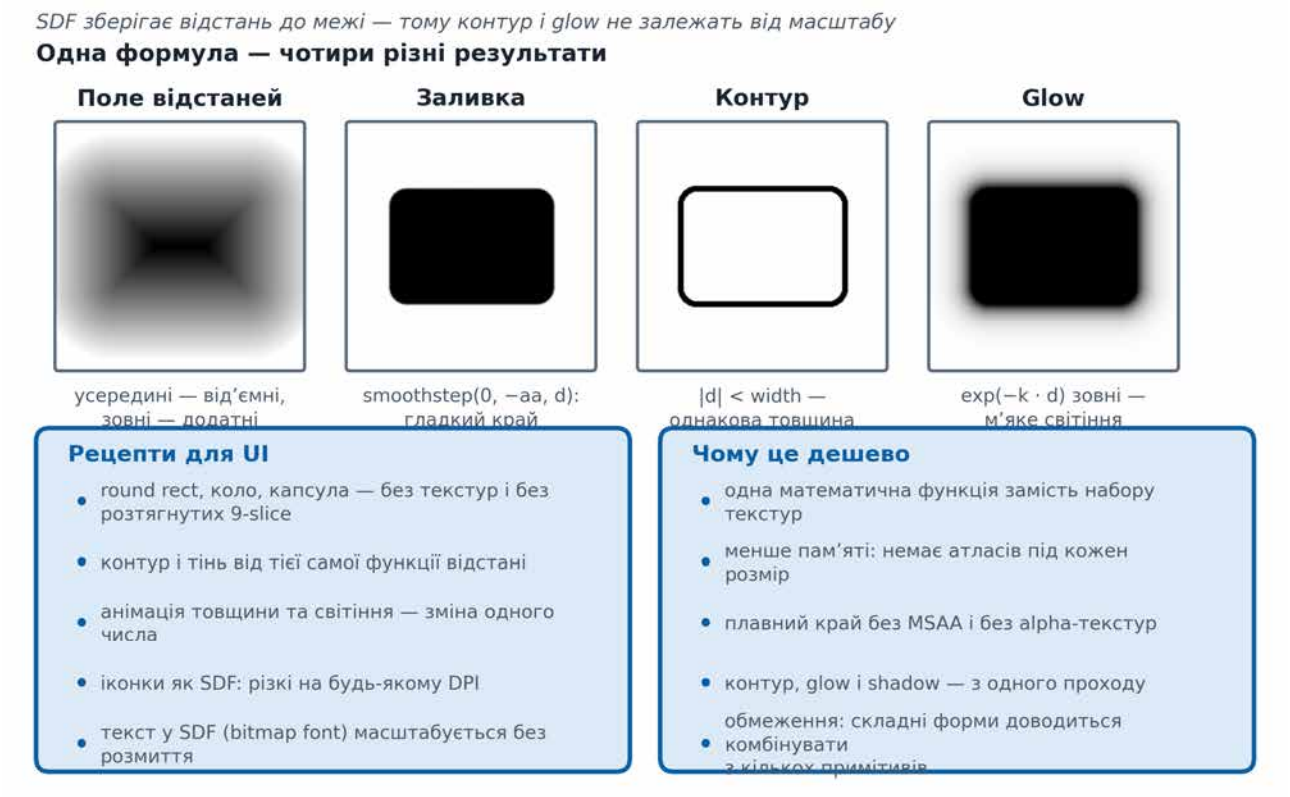


Рис. 2.32. SDF логіка (узагальнено)

d = distance(field)
alpha = 1 – smoothstep(edge0, edge1, d)

2.7.7 Stylized motion: posterization, halftone, linework, Spider-Verse підхід

Stylized motion - «стиль у русі», а не лише в статичному кадрі

• posterize shading,
• halftone dots,
• speed-line overlay,
• temporal jitter (обережно!) (табл. 2.47).

Таблиця 2.47

Що можна безпечно для гри

Ефект	Де добре	Ризик
Posterize	low HP, спец-режим	banding
Halftone	стилізація сцени	шум, мерехтіння
Lines overlay	dash/impact	забиває UI

2.7.8 Event-driven shaders: інтеграція з gameplay (damage, interact, ability)

Шейдери повинні керуватися подіями

hit -> flash на матеріалі (emission boost) + короткий outline;
interact -> highlight pulse;
ability cast -> dissolve/appear (табл. 2.48; рис. 2.33).

Таблиця 2.48

Події -> параметри

Подія	Параметр	Ефект
Damage	HitIntensity	flash + tint
Interact hover	OutlineWidth	outline pulse
Ability	Dissolve	dissolve in/out
Low HP	_ScreenTint	full-screen state

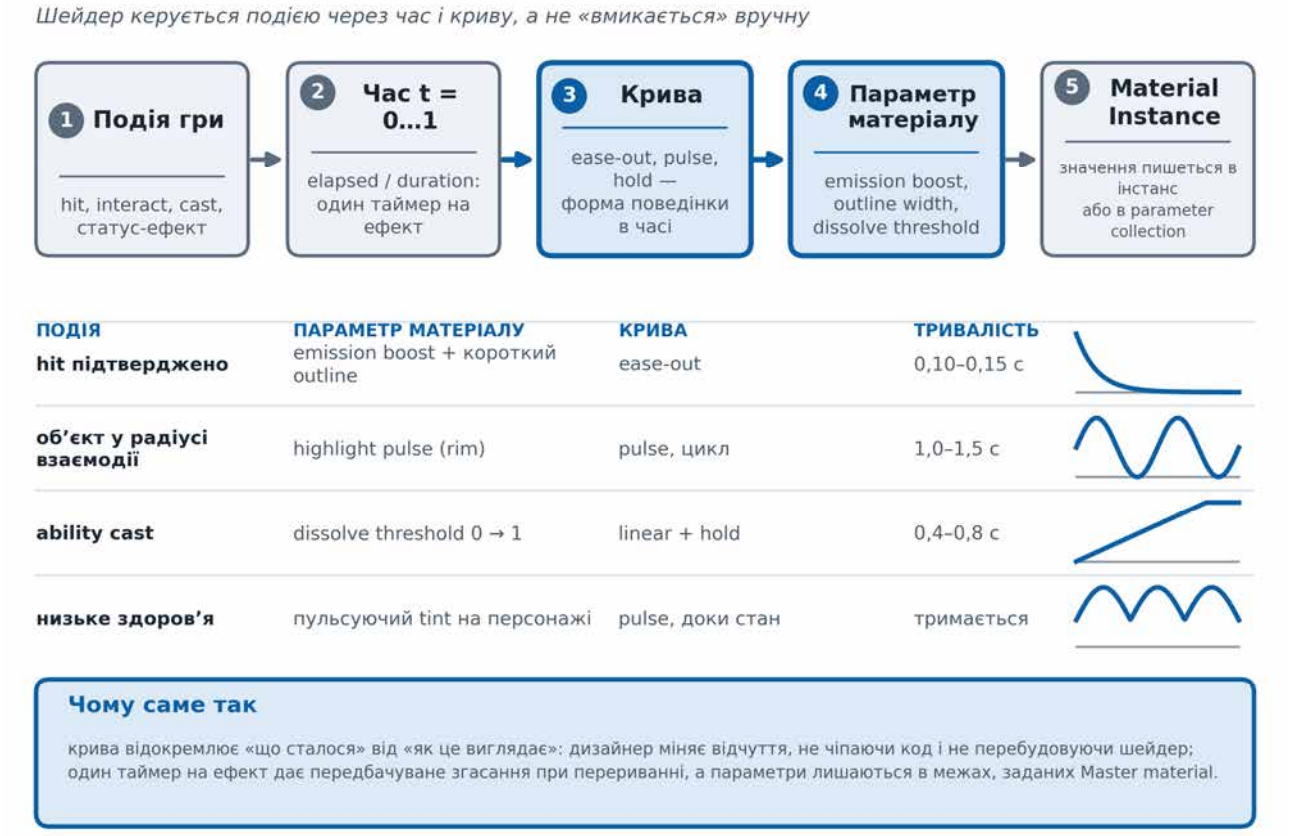


Рис. 2.33. Шаблон «подія -> curve -> shader parameter»

Event -> AnimationCurve(t) -> SetFloat/SetScalar -> Material/PostFX

2.7.9 Performance & Debug: overdraw, variants, render passes

Просунуті шейдери є дороговартісними через

• overdraw (прозорі VFX),
• full-screen pass на високій роздільності,
• багато варіантів (keywords),
• дорогі ноди (noise, loops, branching) (табл. 2.49).

Таблиця 2.49

Дебаг та оптимізація

Проблема	**Симптом**	**Рішення**
Overdraw	GPU spikes	менше прозорості, cutout
Variants	довга збірка	мінімізувати keywords
Full-screen cost	FPS падає	lower res, half-res pass
Noise дорогий	кадр «тягне»	простіший noise, LUT

2.7.10 Production-ready advanced shader - чек-лист готовності

Ефект має чітку функцію (читабельність/стан/impact);
Параметри керовані (intensity/duration/mask);
Інтеграція event-driven (не «постійно увімкнено»);
Відсутні критичні артефакти (banding/shimmer);
Є оптимізація (обмеження пасів/варіантів);
Є shader breakdown (граф/параметри/призначення).

Питання для самоперевірки

1. Коли обираємо матеріал, а коли full-screen FX?
2. Які 3 типові full-screen ефекти у gameplay?
3. Як працює distortion у screen-space?
4. Чому distortion потрібно обмежувати по часу та інтенсивності?
5. Назвіть 3 підходи до toon outline і їх різницю.
6. Що таке SDF і чому він корисний для UI/масок?
7. Як прив’язати шейдер до gameplay події (event-driven)?
8. Які tunables потрібні майже завжди?
9. Що таке overdraw і чому він критичний для VFX?
10. Який чек-лист «production-ready» для advanced shader?

Практичний кейс (час виконання: 120-180 хв)

«Advanced Shader Pack» (1 Full-screen + 1 Object Shader)

Мета: зробити два ефекти:

• Full-screen state FX (damage/low HP/shockwave)

• Object effect (outline або SDF-based highlight або distortion material)

Сценарії (оберіть один набір)

• A (Combat readability): outline ворога + damage flash full-screen.

• B (Ability impact): shockwave distortion full-screen + dissolve material на об’єкті.

• C (Stylized mode): halftone/posterize full-screen + SDF highlight marker.

Кроки виконання завдання

• Створіть full-screen FX з параметрами: intensity, duration, falloff, (опц.) center.

• Додайте time curve (ease in/out) і event trigger (damage/ability).
• Створіть object shader: outline або SDF highlight (pulse) або distortion.
• Підключіть event-driven: hover/interact/hit.
• Додайте мінімальний debug UI: слайдери intensity/duration/threshold.
• Протестуйте в русі + з VFX/lighting (чи не «каша»).
• Зробіть швидку оптимізацію: зменшіть pass, clamp noise, обмежте variants.
• Оформіть shader breakdown.

Що необхідно здати

• Проєкт Unity/UE5 з двома ефектами.
• Відео 30-60 с: показ тригерів і параметрів.
• Shader breakdown (1-2 стор.):
• список параметрів,
• як запускається (події),
• схема графа/нодів (скрін або опис),
• 2 пресети (subtle / strong).
• Постмортем 150-250 слів: які компроміси, де оптимізували.

Критерії оцінювання

• функціональність (ефект реально допомагає читабельності/стану);
• керованість (tunables зрозумілі, є пресети);
• інтеграція (event-driven, без «постійного шуму»);
• якість (мінімум артефактів, гарний вигляд у русі);
• продуктивність (є хоча б 2 оптимізації);
• документація (breakdown чіткий і відтворюваний).

Рекомендовані програми та сервіси

1. Unity: Shader Graph, (HDRP/URP) Renderer Features/Full-screen passes, RenderDoc (advanced).
2. UE5: Material Editor, Post Process Materials, Custom Depth/Stencil, (опц.) Material Parameter Collection.
3. Профілювання: Unity Profiler/Frame Debugger; UE Stat GPU/Unreal Insights.
4. Референси/практика: Shadertoy, The Book of Shaders (для ідей), але адаптувати під production.

Рекомендовані джерела та матеріали

1. Unity Docs: Shader Graph, URP/HDRP post-processing, renderer features/full-screen effects.
2. Unreal Engine Docs: Post Process Materials, Custom Depth/Stencil, Materials optimization.
3. Практичні розбори stylized shading: Breath of the Wild; стилізація руху: Spider-Verse підходи (на рівні принципів).
4. GDC доповіді про readability, outlines, VFX+shader budgets.

Тема 2.8. Production-ready пайплайн 2D/3D асетів, ригінгу та анімації

Мета теми: навчити студентів організовувати повний цикл асета - від brief і blockout до імпорту, LOD, collision, animation, validation та інтеграції в build - з урахуванням художніх і технічних бюджетів.

2.8.1 Asset brief і критерії готовності

Асет починається не з моделювання, а з brief: призначення, камера, дистанція, масштаб, interaction, матеріали, анімація, collision, платформа, бюджет і власник. Без brief художник може створити красивий, але непридатний для gameplay об’єкт.

Hero asset, gameplay asset, modular kit, background prop і VFX mesh мають різні вимоги. Для об’єкта, який гравець бере в руки, важливі pivot, grip points, close-up texel density та стани пошкодження. Для далекої декорації - silhouette, LOD і дешевий матеріал.

Definition of Done повинна включати не лише фінальний вигляд, а й правильні одиниці, orientation, pivot, naming, UV, materials, collision, LOD, import preset, prefab/blueprint, test scene, source license і перевірку на target platform (табл. 2.50) (рис. 2.34).

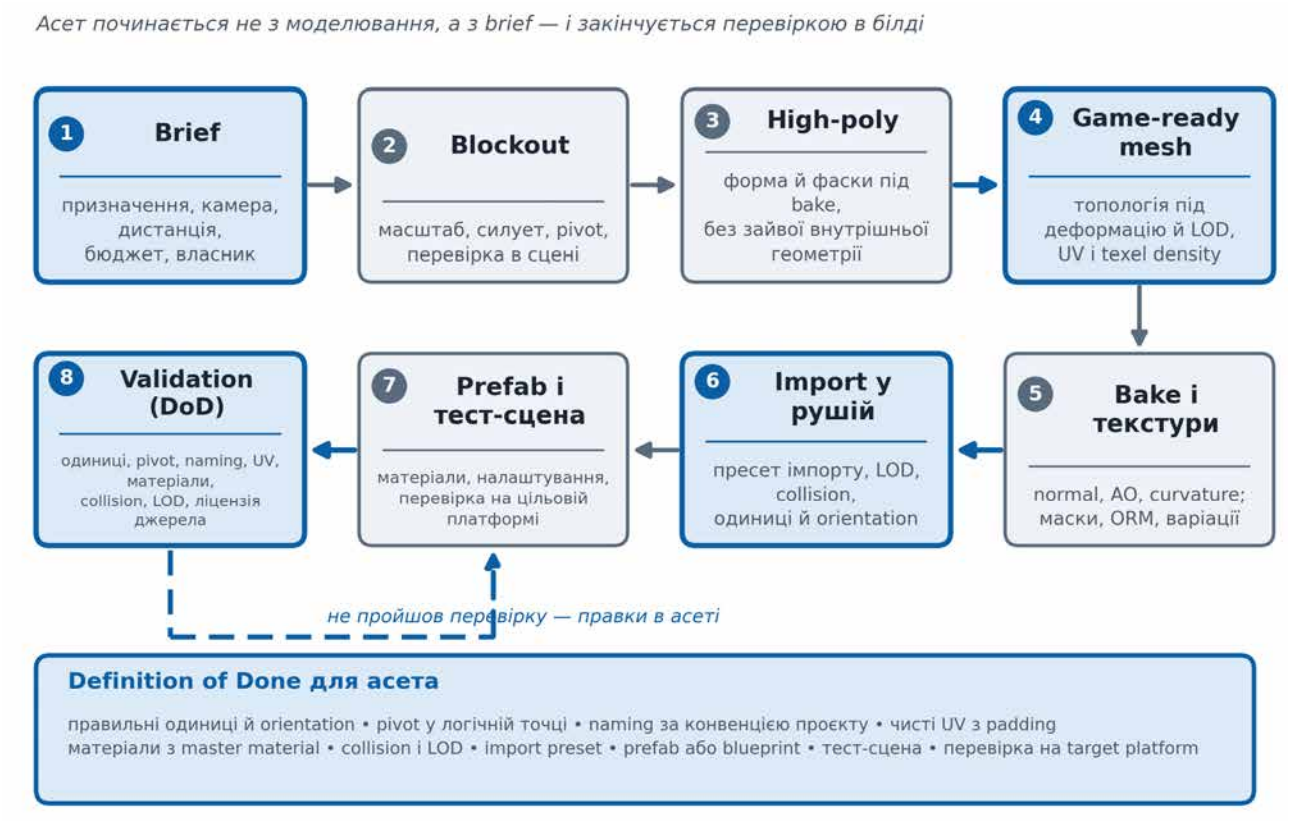


Рис. 2.34. Життєвий цикл game asset від brief до validation

Таблиця 2.50

Приклад asset brief

Поле brief	Приклад рішення	Критерій
Роль	інтерактивні двері	стани closed/open/locked/broken
Камера	third-person, 2-10 м	силует читається у русі
Scale / pivot	метри, pivot на петлі	анімація без offset
Geometry budget	25k/10k/3k triangles	LOD transition непомітний
Textures	2K atlas, shared trim	стабільна texel density
Collision	simple convex	немає snagging
Animation	open/close/impact	notify синхронний із SFX
Platform	mid-range PC / console	budget підтверджено profiler

2.8.2 Геометрія, topology, UV і baking

Топологія повинна відповідати деформації, shading і LOD, а не абстрактній «чистоті». У статичному hard-surface важливі silhouette, smoothing groups, bevel strategy і відсутність зайвої внутрішньої геометрії. У персонажах edge loops підтримують суглоби, міміку й corrective shapes.

UV-підхід обирають за типом асета: унікальні maps, trim sheets, tileable textures, atlases або virtual texturing. Texel density має бути узгодженою в межах сцени, а padding - достатнім для mip maps. Overlap допустимий лише свідомо.

Baking normal, AO, curvature та інших maps вимагає стабільної cage, triangulation і перевірки у цільовому tangent space. Артефакти, які непомітні в DCC, можуть стати видимими після компресії або під іншим освітленням (табл. 2.51) (рис. 2.35).

Таблиця 2.51

Типові дефекти 3D-асетів

Проблема	Симптом у рушії	Перевірка / виправлення
Невірний scale	physics, animation, camera працюють некоректно	єдині units і export preset
Поганий pivot	обертання/placement зі зміщенням	pivot rule у brief
Недостатній UV padding	bleeding на mip levels	padding за resolution
Tangent mismatch	шви normal map	узгоджений tangent space
Надмірна topology	memory і skinning cost	silhouette-based optimization
Відсутній LOD	GPU cost на відстані	LOD chain і screen-size test
Складний collision	CPU spikes, snagging	simple primitives / convex hulls

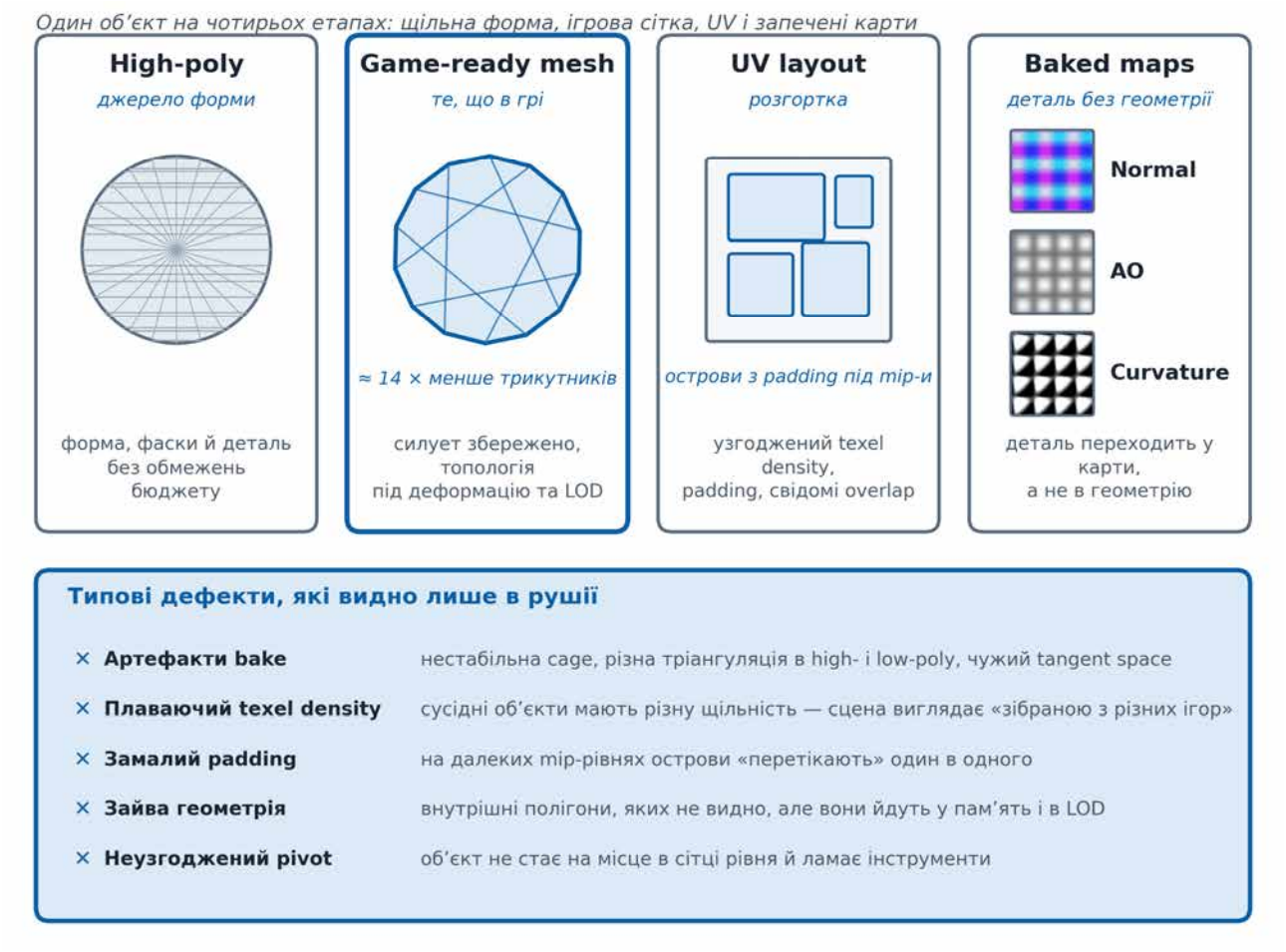


Рис. 2.35. Порівняння high-poly, game-ready mesh, UV та baked maps

2.8.3 Матеріали, текстури та варіативність

Матеріальна система має зменшувати кількість унікальних shaders і текстур, зберігаючи варіативність. Master material, material instances, masks, trim sheets, decals і vertex colors дозволяють створити багато станів без дублювання великих наборів даних.

Компресія залежить від типу карти. Normal, color, mask і HDR-дані не повинні використовувати однаковий preset. Alpha, channel packing, mip bias і streaming впливають на обсяг пам'яті та якість. Рішення перевіряють на target hardware, а не лише у viewport.

Для 2D-асетів важливі pixel density, atlas packing, pivot, slicing, 9-slice borders, filtering, color space та масштабування UI. Pixel art потребує окремих правил щодо camera та integer scaling (табл. 2.52).

Таблиця 2.52

Підготовка текстурних даних

Тип даних	Рекомендація	Типова помилка
Base color	sRGB, контроль palette	запечене освітлення
Normal	linear, normal compression	інверсія green channel
Masks	channel packing	зайві окремі textures
HDR/emissive	відповідний HDR format	banding або clipping

Тип даних	Рекомендація	Типова помилка
UI sprite	atlas, correct border/pivot	blur на масштабуванні
Pixel art	point filtering, integer scale	нерівномірні pixels
Decal	обмежена overdraw area	дорогі великі прозорі шари

2.8.4 Rigging, skinning, retargeting та animation export

Риг повинен відповідати runtime-потребам. Deformation skeleton відокремлюють від control rig, зайві helper bones не експортують. Naming, hierarchy, bind pose, root, scale і axis фіксуються у pipeline standard.

Skinning перевіряють у крайніх позах, а не лише в neutral pose. Для складних зон застосовують corrective blend shapes або pose-space deformation, але враховують витрати пам’яті та runtime. Кількість influences має відповідати платформі.

Retargeting працює надійно, коли узгоджені bone mapping, reference pose, proportions і root motion policy. Animation clips повинні мати точні ranges, loop settings, events/notifies, compression і semantic names (табл. 2.53; рис. 2.36).

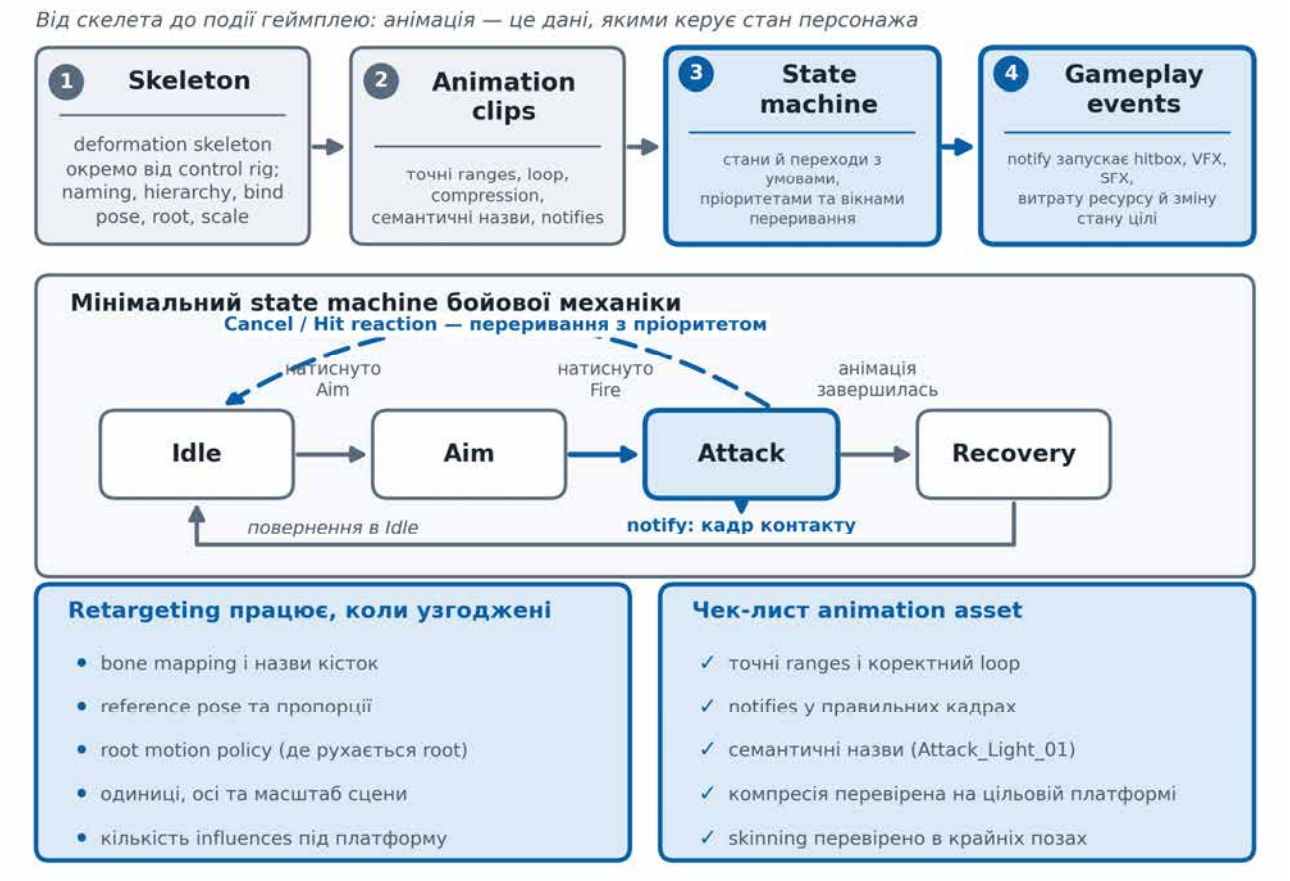


Рис. 2.36. Схема skeleton - animation clip - state machine - gameplay events

Таблиця 2.53

Чек-лист animation asset

Перевірка	Питання
Skeleton	чи стабільні hierarchy, root, names, orientation?
Skinning	чи немає collapse, candy-wrapper, sliding?
Clips	чи коректні start/end, loop, additive, root motion?
Retarget	чи зберігається contact і пропорційність руху?
Events	чи синхронні footsteps, hit windows, VFX/SFX?
Compression	чи немає jitter і втрати контакту?
Runtime	скільки bones, skinned meshes і anim layers активні?

2.8.5 Import presets, validation та автоматизація

Ручне налаштування сотень асетів призводить до нерівномірного результату. Import presets, naming rules і validation scripts повинні автоматично перевіряти scale, texture size, compression, material count, missing references, LOD, collision та заборонені формати.

Validation краще виконувати на трьох рівнях: локально під час імпорту, у test scene та в CI/build pipeline. Попередження має пояснювати проблему та спосіб її усунення. Критична помилка блокує build лише тоді, коли дійсно загрожує якості або платформі.

Source-файли, export-файли й runtime-асети потрібно розділяти. Репозиторій має містити достатні вихідні матеріали для повторного експорту, але не дублювати кеші та автоматично згенеровані дані без потреби (табл. 2.54).

Таблиця 2.54

Приклади автоматичної валідації

Validation rule	Severity	Автоматична дія
Texture > дозволеного розміру	warning/error	застосувати max size або відхилити
Невірний scale/orientation	error	зупинити інтеграцію
Material slots > budget	warning	показати cost
Missing LOD/collision	warning	створити task
Unsupported shader	error	замінити preset
Невірне ім’я	warning	запропонувати шаблон
Missing license metadata	error before release	додати в asset register

Практичний кейс: production-ready asset pack

Підготуйте modular kit або персонажа з brief, source files, naming, geometry/texture budgets, LOD, collision, materials, animation clips, import preset, validation report і test scene. Додайте before/after optimization та README з ліцензіями.

Питання для самоперевірки до розділу 2

1. Яку роль blockout відіграє у зменшенні ризиків на рівні?
2. Як critical path пов’язаний із optional paths і risk/reward?
3. Що таке beat map і як вона допомагає керувати темпом?
4. Які spatial metrics потрібно стандартизувати для конкретної механіки?
5. Як landmarks, світло, композиція та звук підтримують навігацію?
6. Чим art direction відрізняється від набору референсів?
7. Які етапи входять до production-ready арт-пайплайну?
8. Як naming convention і import presets зменшують технічний борг?
9. Коли HUD має бути постійним, контекстним або diegetic?
10. Які стани має мати інтерактивний UI-елемент?
11. Як safe area, scale і contrast залежать від платформи?
12. Що означає фізично обґрунтований матеріал?
13. Як roughness, metallic і normal map впливають на сприйняття поверхні?
14. Коли стилізація потребує відхилення від PBR і як зберегти узгодженість?
15. Чому VFX потрібно синхронізувати з gameplay event, а не лише з input?
16. Як overdraw і кількість частинок впливають на GPU?
17. Як розділити readability pass і mood pass освітлення?
18. Чому автоматична exposure може шкодити ясність gameplay?
19. Які ризики пов’язані з full-screen ефектами, distortion і outlines?
20. Як скласти візуальний budget для vertical slice?

Практичні завдання до розділу 2

Завдання 1. Blockout першого рівня

Створіть 3 top-down варіанти та 1 3D blockout-версію. Позначте critical path, landmarks, encounter zones, checkpoints, optional rewards і beat map. Проведіть плейтест без фінального арту.

Завдання 2. UI/HUD prototype

Спроєктуйте HUD і два екрани меню для keyboard/mouse та gamepad. Додайте focus states, error states, safe area, text scaling, color-blind check і user flow.

Завдання 3. World & FX slice

Підготуйте невелику сцену з PBR/stylized materials, освітленням, одним ключовим VFX, post-processing і performance budget. Представте breakdown та before/after readability pass.

Висновки до розділу 2

1. Ігровий світ є інтерфейсом. Розташування, масштаб, композиція, матеріал, світло й ефекти повинні дозволяти гравцеві бачити можливості, ризики та наслідки без перевантаження текстовими підказками.
2. Візуальний pipeline стає production-ready лише тоді, коли поєднує художню мету з технічними правилами: naming, імпорт, LOD, бюджети, профілювання та контроль якості в цільовій камері й build.

Графічні матеріали до розділу 2

Таблиця 2.55

Перелік рекомендованих графічних матеріалів до розділу 2

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
2.1	Структура дизайну сучасного рівня	Рекомендовано поєднати план зверху та перспективний greybox-кадр, позначити critical path, опційні маршрути, точки інтересу і зони ризику.
2.2	Приклад blockout дизайну сучасного рівня	Рекомендовано поєднати план зверху та перспективний greybox-кадр, позначити critical path, опційні маршрути, точки інтересу і зони ризику.
2.3	Приклад дизайну шляху гравця для сучасного рівня	Рекомендовано поєднати план зверху та перспективний greybox-кадр, позначити critical path, опційні маршрути, точки інтересу і зони ризику.
2.4	Приклад дизайну візуальної прогресії шляху гравця	Рекомендовано поєднати план зверху та перспективний greybox-кадр, позначити critical path, опційні маршрути, точки інтересу і зони ризику.
2.5	«Трикутник» візуального дизайну гри	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.6	Процес візуального дизайну гри	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
2.7	Приклад концепт-дизайну UI для гри	Додати анотований скріншот або wireframe з нумерованими зонами, ієрархією інформації, safe area та прикладами станів елементів.
2.8	Приклад машин станів персонажа	Використати UML-подібну діаграму станів із переходами, умовами та пріоритетами; окремо позначити стани помилки та переривання.
2.9	Ієрархія інформації HUD (приклад)	Додати анотований скріншот або wireframe з нумерованими зонами, ієрархією інформації, safe area та прикладами станів елементів.
2.10	Цикл онбордингу (практична модель)	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
2.11	Пайплайн налаштувань (SettingsData -> Apply -> Save)	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
2.12	Цикл поліпшення UI/UX (measure -> fix -> verify)	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
2.13	User Flow (мінімальний для прототипу)	Додати анотований скріншот або wireframe з нумерованими зонами, ієрархією інформації, safe area та прикладами станів елементів.
2.14	Узагальнена схема: від матеріалу до пікселя (pipeline спрощено)	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
2.15	Приклад пакування масок (ORM)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.16	Базовий патерн «маска керує змішуванням»	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.17	Логіка dissolve (узагальнено)	Показати порівняння матеріалів на однакових тестових об’єктах і фрагмент node graph; підписати параметри, що змінюються.
2.18	Fresnel (спрощено)	Показати порівняння матеріалів на однакових тестових об’єктах і фрагмент node graph; підписати параметри, що змінюються.
2.19	Міні-пайплайн матеріалу (від DCC до рушія)	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
2.20	Пакет відгуку (feedback stack) для однієї дії	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.21	Стек емітерів для ability FX (приклад)	Подати серію з 3-5 кадрів anticipation – impact – recovery, додати часову шкалу та короткі примітки щодо читабельності й бюджету.
2.22	Подієвий ланцюг для удару (узагальнено)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.23	Приклад параметризації (tunables)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.24	Поетапний пайплайн	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
2.25	EV логіка (узагальнено)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.26	Color scripting по зонах (приклад)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.27	Readability checklist (спрощено)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.28	Decision flow (спрощено)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.29	Шаблон full-screen ефекту	Подати серію з 3-5 кадрів anticipation – impact – recovery, додати часову шкалу та короткі примітки щодо читабельності й бюджету.
2.30	Distortion формула (узагальнено)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.31	Sobel outline (спрощено)	Показати порівняння матеріалів на однакових тестових об’єктах і фрагмент node graph; підписати параметри, що змінюються.
2.32	SDF логіка (узагальнено)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
2.33	Шаблон «подія -> curve -> shader parameter»	Показати порівняння матеріалів на однакових тестових об’єктах і фрагмент node graph; підписати параметри, що змінюються.
2.34	Життєвий цикл game asset від brief до validation	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
2.35	Порівняння high-poly, game-ready mesh, UV та baked maps	Рекомендовано поєднати план зверху та перспективний greybox-кадр, позначити critical path, опційні маршрути, точки інтересу і зони ризику.
2.36	Схема skeleton - animation clip - state machine - gameplay events	Використати UML-подібну діаграму станів із переходами, умовами та пріоритетами; окремо позначити стани помилки та переривання.

РОЗДІЛ 3. ВЗАЄМОДІЯ, GAME FEEL ТА МУЛЬТИМОДАЛЬНИЙ

Мета розділу: розділ розглядає якість взаємодії як результат узгодження вводу, станів, руху, камери, анімації, VFX, аудіо, UI та тактильного feedback. Гравець повинен швидко отримувати підтвердження команди, розуміти результат і передбачати наступний стан.

Game feel формується на рівні мілісекунд, кривих, пріоритетів, cancel windows і мікросигналів. Тому суб'єктивне відчуття потрібно перетворювати на вимірювані tunables, A/B-варіанти й сценарії тестування.

У розділі поєднано input systems, controller/camera, interaction architecture, audio, animation, onboarding, telemetry та XR. Це дозволяє бачити feedback не як набір окремих ефектів, а як спільну мову системи.

Після вивчення розділу здобувач набуває компетентності:

• оцінювати input latency і забезпечувати швидкий first feedback;
• будувати action-based input, buffering, coyote time і state-driven controller;
• налаштовувати камеру, framing, smoothing, collision та impulses;
• проєктувати interaction за схемою input – state – feedback;
• інтегрувати animation events, hit windows, interrupts і root motion;
• створювати layered, spatial і parameter-driven audio;
• проєктувати onboarding, progressive disclosure, hints і telemetry;
• враховувати VR/XR comfort, spatial UI та haptics.

Ключові терміни: game feel, responsiveness, input latency, controller, camera, interaction system, animation state, audio event, onboarding, telemetry, XR, haptics.

Тема 3.1. Game feel та Input Systems: responsiveness, feedback loops, MDA

Мета: сформувати у студентів цілісне розуміння «відчуття гри» (game feel) як інтегральної характеристики взаємодії гравця з грою; навчити проєктувати та реалізовувати низьколатентний ввід, керовані стани, зрозумілий фідбек (UI/VFX/Audio/Camera), а також організовувати тестування responsiveness у вигляді міні-пайплайну, що підтримує ітеративне покращення прототипу протягом семестру.

Очікувані результати навчання

- вміти пояснювати складові game feel та принцип «first feedback < 100 ms»;
- визначати джерела input latency (input -> update -> render -> display) і оцінювати їхній внесок у сприйняття;
- проєктувати фідбек за фазами anticipation / impact / recovery та синхронізувати канали (анімація, VFX, аудіо, UI, камера);
- застосовувати прийоми forgiveness (coyote time, buffering, jump forgiveness) для покращення керованості без втрати чесності;
- конфігурувати action-based input mapping у Unity Input System або UE Enhanced Input з підтримкою контекстів та rebinding;

- будувати micro-feedback патерни і перевіряти їх читабельність у русі;
- використовувати MDA (Mechanics-Dynamics-Aesthetics) для аналізу проблем «відчуття» та формалізації гіпотез;
- організовувати параметризацію (tunables), debug UI та швидкі A/B тести;
- виявляти типові помилки responsiveness та пропонувати виправлення;
- збирати «Responsiveness Test Bench» з 3 сценаріями та робити ітераційний звіт.

Ключові терміни: game feel; responsiveness; input latency; feedback loop; anticipation/impact/recovery; buffering; coyote time; jump forgiveness; action mapping; context; rebinding; dead zone; smoothing; state machine; event bus; micro-feedback; hit-stop; camera impulse; MDA; tunables; debug HUD; telemetry; frame time; VSync; render pipeline.

3.1.1 Поняття game feel і принцип «first feedback»

Game feel (відчуття гри) – це сукупне сприйняття гравцем керованості, точності та «фізичної правдоподібності» взаємодії з грою. Воно формується не окремою механікою, а узгодженістю багатьох компонентів: вводу, руху, анімації, камери, візуальних ефектів, звуку, UI та ритму. Коли ці компоненти зібрані в єдину систему фідбеку, гравець відчуває контроль: «я натиснув – гра відповіла – я розумію, що сталося».

Ключовий практичний орієнтир для прототипів семестру – забезпечити перший відгук (first feedback) за менш ніж 100 мс від моменту дії гравця. Важливо: це не означає, що вся дія завершується за 100 мс. Йдеться про перший підтверджувальний сигнал (мікро-рух, звук кліку, зміна пози, підсвітка, імпульс камери), який повідомляє гравцю: команда прийнята, система реагує.

Складові game feel (практична модель)

- Ввід: чистий сигнал, зрозумілі мапінги, мінімальна затримка.
- Модель руху: передбачуваність, плавність, контроль над прискоренням/гальмуванням.
- Станова логіка: коли дія дозволена/заборонена, як система це пояснює гравцю.
- Фідбек: синхронізація каналів (анімація + VFX + аудіо + UI + камера).
- Ритм: паузи/удари, темп, читабельність, керовані «піки» інтенсивності (табл. 3.1; 3.2; рис. 3.1).

Таблиця 3.1

Орієнтовні цілі часу першого фідбеку (first feedback)

Тип дії	**Перший фідбек (ціль)**	**Канали фідбеку (приклади)**	**Коментар**
Натискання кнопки UI	30-80 мс	звук/анімація кнопки/підсвітка	інтерактивність і довіра
Стрибок/ривок	50-100 мс	мікро-зміна пози/звук/камера impulse	важливо для «tight controls»

Тип дії	Перший фідбек (ціль)	Канали фідбеку (приклади)	Коментар
Атака/вміння	60-120 мс	телеграф + звук + flash + pose	impact може бути пізніше
Взаємодія (підібрати/відкрити)	60-120 мс	підсвітка/підказка/звук підтвердження	інколи краще швидше
Помилка (дія недоступна)	50-100 мс	UI повідомлення/звук відмови	фідбек важливіший за «мовчання»

Таблиця 3.2

Навчальна таблиця до теми «Game feel та Input Systems: responsiveness, feedback loops, MDA»

```
| ДІЯ ГРАВЦЯ -> СИСТЕМА -> ПЕРШИЙ ФІДБЕК -> РЕЗУЛЬТАТ |
| (натиснув) (перевірила) (мікро-сигнал) (ефект) |
+-Ціль: first feedback < 100 ms, навіть якщо результат пізніше |
+-----------------------------------------------------------+
```

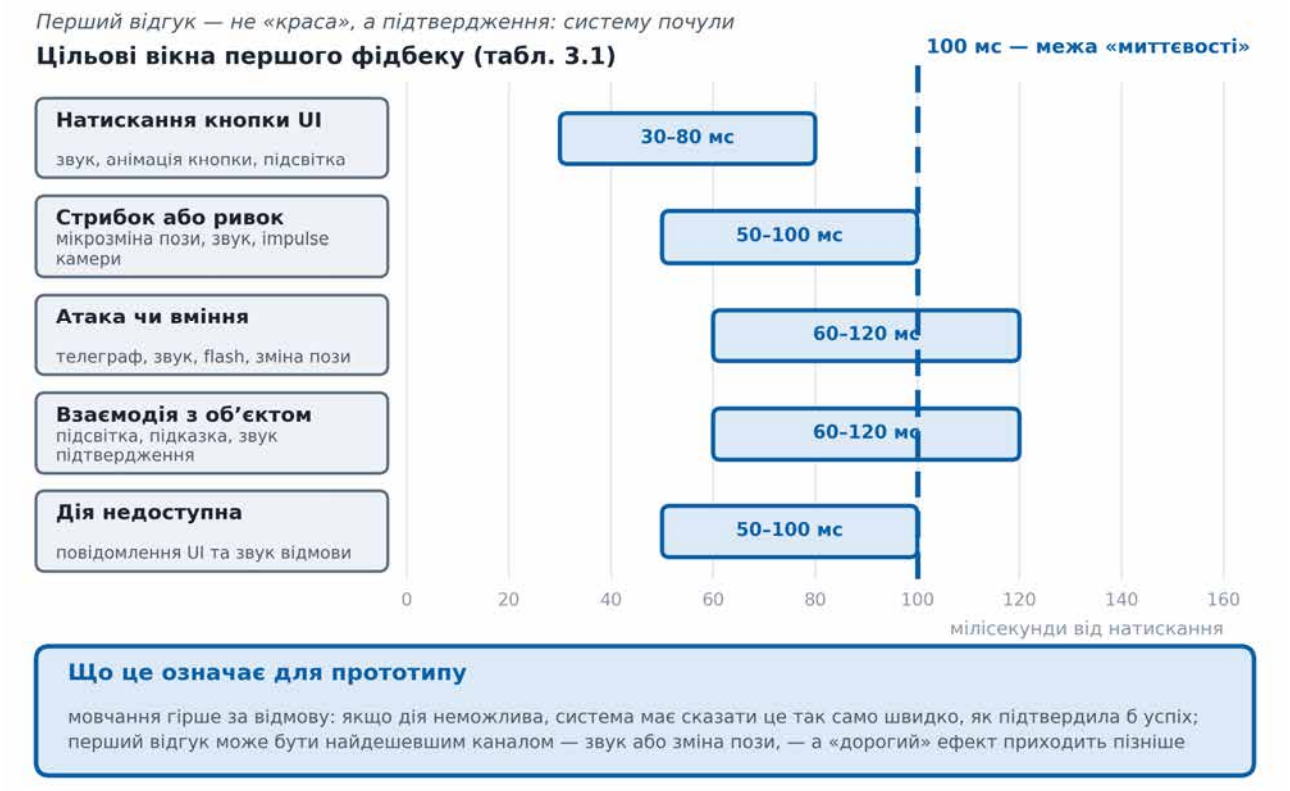


Рис. 3.1. Принцип «first feedback < 100 ms» як стандарт якості прототипу

Висновок: якщо гравець не отримує швидкого та зрозумілого першого фідбеку, він відчуває затримку, «липкість» керування або невпевненість у системі. Тому у всіх лабораторних та проєктних роботах семестру фокусуємось на вимірюваному responsiveness і на узгодженості фідбек-каналів.

3.1.2 Input latency: джерела затримки (input -> update -> render -> display)

Input latency - це час від фізичної дії гравця (натискання/рух стика/миші) до моменту, коли результат стає видимим на екрані (або відчутним через аудіо/вібрацію). Для практики важливо не лише знати, що затримка існує, а вміти розкладати її на складові та оптимізувати ті, що у наших руках.

Типовий ланцюжок затримки

• Пристрій вводу (опитування контролера, debounce кнопок).

• ОС/драйвери (черги подій, фільтри, налаштування).

• Ігровий рушій: збір подій вводу (input sampling).

• Оновлення логіки (Update/FixedUpdate/Tick) та станів.

• Симуляція/фізика (за потреби).

Рендеринг кадру (CPU -> GPU), пост-ефекти.

• Синхронізація з дисплеєм (VSync), час відгуку матриці.

Навіть без детальних вимірювань варто пам’ятати: на 60 FPS тривалість кадру становить 16,67 мс. Якщо подія вводу потрапила одразу після моменту семплінгу, вона може чекати майже цілий кадр, перш ніж буде оброблена. А якщо ще й відрендерений кадр чекає VSync - затримка збільшується (табл. 3.3; 3.4; рис. 3.2).

Таблиця 3.3

Оцінка «кадрового внеску» у latency

FPS	Тривалість кадру, мс	Очікування до обробки вводу (0-1 кадр)	Очікування до показу (0-1 кадр)	Примітка
30	33,3	0-33,3	0-33,3	часто відчутно «в’язко»
60	16,7	0-16,7	0-16,7	базовий мінімум для багатьох жанрів
90	11,1	0-11,1	0-11,1	VR/шутери виграють від цього
120	8,3	0-8,3	0-8,3	відчутно краще в контролі

Таблиця 3.4

Навчальна таблиця до теми «Game feel та Input Systems: responsiveness, feedback loops, MDA»

Input event
|
v
[Input Sample]----(waiting)---->[Update/Tick]---->[Render]----(VSync?)---->[Display]
^ ^ ^ ^ ^
момент збору черга подій логіка/стан CPU->GPU видимий кадр

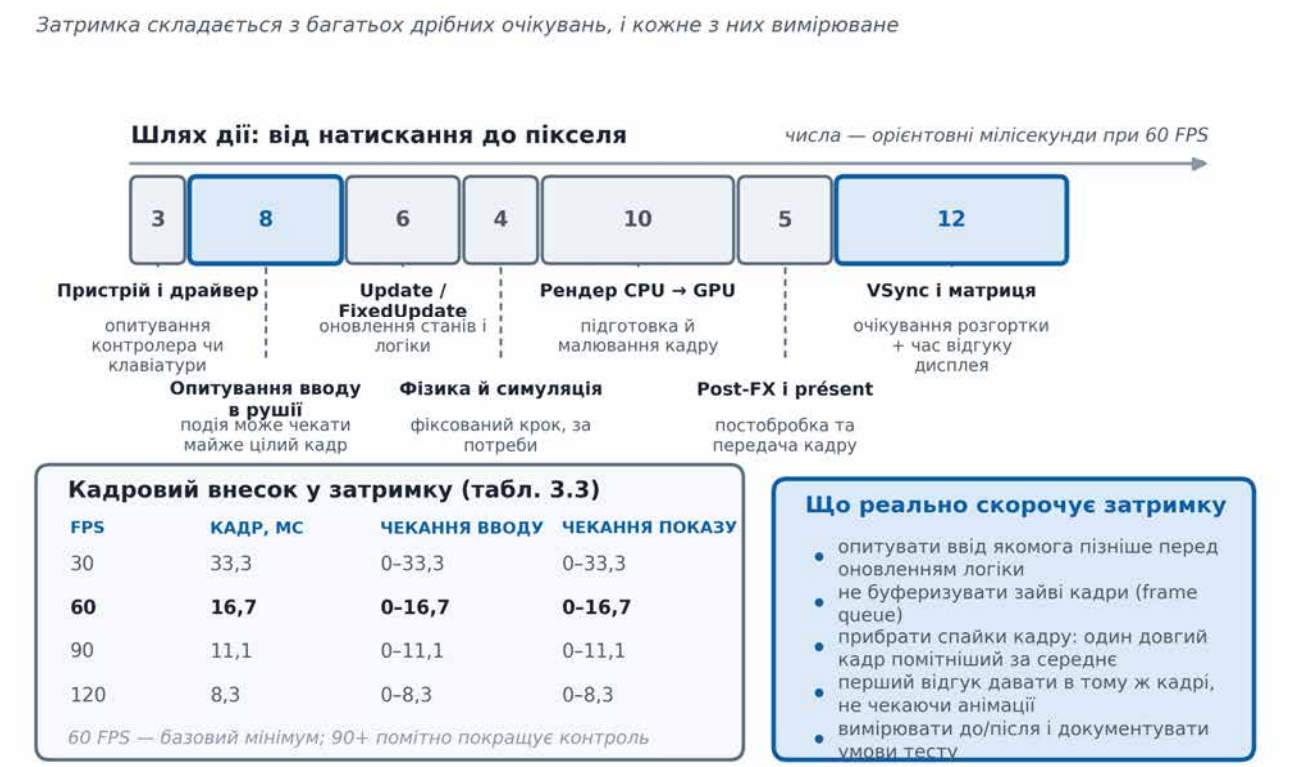


Рис. 3.2. Спрощена шкала затримки: від input event до display

Практичні рекомендації зменшення latency

• Прагніть до стабільного FPS (не піків), бо саме спайки руйнують відчуття керування.

• Уникайте зайвого smoothing на вводі та камері на етапі прототипу: краще мінімум і керовані параметри.

• Розділяйте Update/FixedUpdate: рух і ввід мають бути узгоджені (особливо в Unity).

• Перевіряйте налаштування VSync/Frame pacing під час тестів (але документуйте, у яких умовах міряли).

• Використовуйте профайлери рушія, щоб знаходити «довгі кадри» (CPU/GPU).

3.1.3 Anticipation / impact / recovery як дизайн-мова фідбеку

Одна з найкорисніших моделей для формування читабельного й «приємного» фідбеку - поділ будь-якої значущої дії на три фази: anticipation (підготовка/попередження), impact (момент дії/удару) та recovery (завершення/повернення). Це універсальна мова, яка працює як у бойових діях, так і в UI, пересуванні, взаємодіях.

Мета такого поділу - допомогти гравцю передбачити подію, відчути її силу та зрозуміти, що система повернулась у стабільний стан. Візуально та аудіально ці фази можуть бути дуже короткими, але вони мають бути відчутними (табл. 3.5; 3.6; рис. 3.3).

Таблиця 3.5

Фази фідбеку та канали (приклади)

Фаза	**Завдання для гравця**	**Візуально**	**Аудіо**	**Камера/ геймплей**
Anticipation	підготуватисьзрозуміти намір	поза, телеграф, підсвітка, заряд	наростаючий тон/свист	невеликий «wind-up», обмеження руху
Impact	зафіксувати результат	flash, hit VFX, спрайти удару	удар/клац/вибух	hit-stop, impulse, короткий shake
Recovery	повернути контроль	затухання, слід, частинки	затухаючий хвіст	вікно відновлення, cooldown

Таблиця 3.6

Навчальна таблиця до теми «Game feel та Input Systems: responsiveness, feedback loops, MDA»

Anticipation: ___/ **Impact: |*|** **Recovery: ___** **(Мета: ритм і читабельність, навіть у дуже коротких діях)**

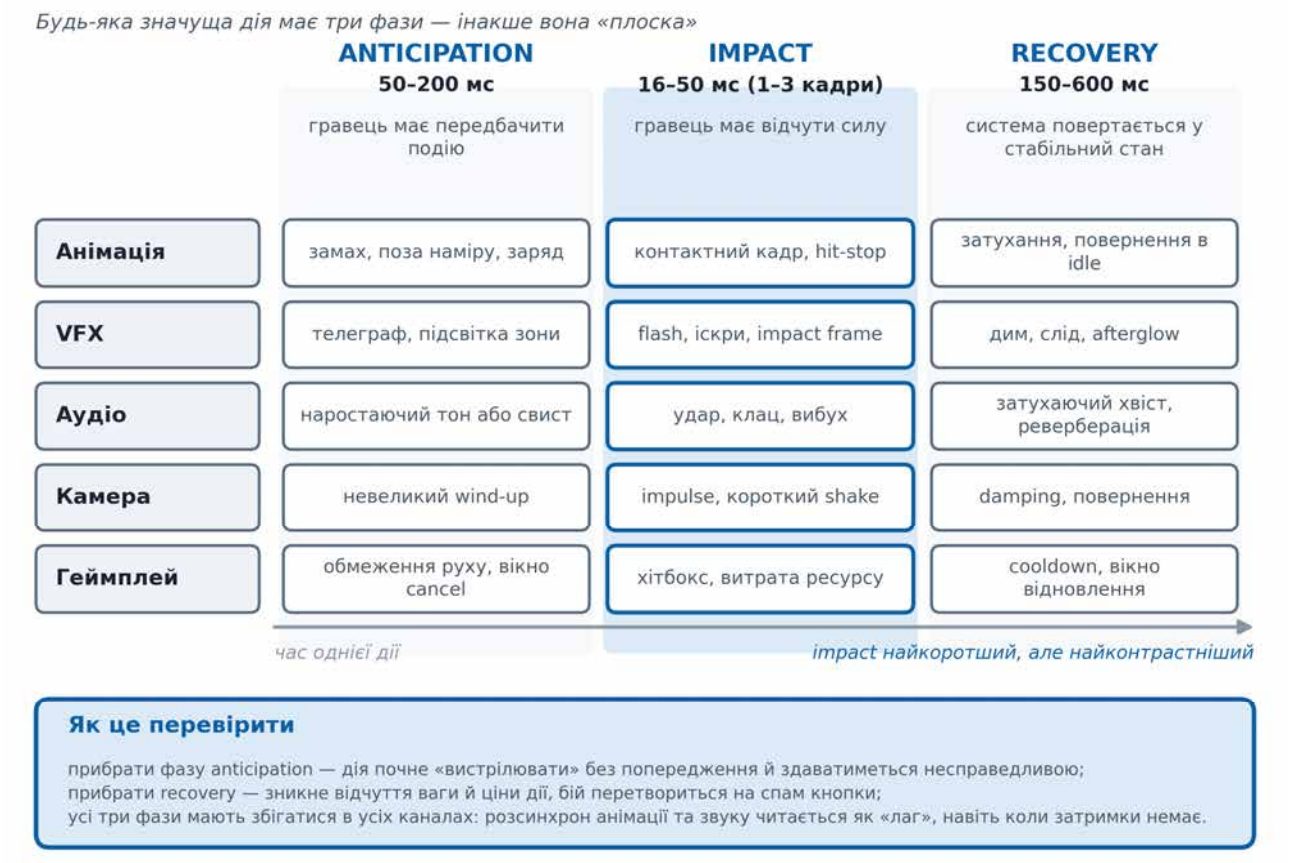


Рис. 3.3. Ритм дії у фазах anticipation/impact/recovery (схема)

Застосування моделі у прототипах

• Рух/стрибок: anticipation – підтиснення/звук; impact – відрив/камера; recovery – приземлення/пил.

• UI кнопка: anticipation – hover; impact – click; recovery – release + коротке затухання.

• Взаємодія: anticipation – підсвітка об’єкта; impact – підтвердження; recovery – зміна стану (відкрито/підібрано).

3.1.4 Coyote time, buffering, jump forgiveness: контроль без «обману»

Forgiveness (поблажливість) – це набір прийомів, що робить керування комфортнішим, зменшує фрустрацію та підсилює відчуття майстерності. Важливо: правильна поблажливість не «грає замість гравця», а компенсує обмеження людської реакції, латентність системи та неточність сприйняття таймінгів (табл. 3.7).

Основні прийоми, які необхідно застосовувати

Coyote time – коротке вікно (наприклад 80-150 мс) після сходу з платформи, коли стрибок ще дозволено

Jump buffering – якщо гравець натиснув стрибок трохи зарано (до приземлення), команда зберігається на короткий час

Jump forgiveness / leniency – корекції, що допомагають приземленню/чіпляння (наприклад, розширення колайдера, snap-to-ground)

Таблиця 3.7

Типові параметри forgiveness (орієнтири для прототипу)

Прийом	Типові значення	Перевага	Ризик/обмеження
Coyote time	0,08-0,15 с	менше «несправедливих» падінь	не переборщити, щоб не ламати правила
Jump buffer	0,08-0,20 с	стрибок спрацьовує «як очікується»	треба керувати пріоритетами вводу
Ground snap	0-0,10 м	прибирає дрібні «підвисання»	може створити артефакти на схилах
Input dead zone (стик)	0,10-0,25	стабільність керування	занадто велика – втрата точності

Приклади: Celeste відома своєю продуманою поблажливістю (coyote time, buffer, корекції стрибка), які роблять керування надзвичайно «tight» і водночас дружнім до помилок. Hades також використовує комбінацію чітких таймінгів, анімаційних сигналів та швидкого фідбеку (impact + audio + камера), що підсилює контроль у динамічному бою.

Фрагмент 1.1. Логіка coyote time та jump buffering (псевдокод) (табл. 3.8)

Таблиця 3.8

Навчальна таблиця до теми «Game feel та Input Systems: responsiveness, feedback loops, MDA»

```
// Псевдокод (незалежний від рушія): coyote time + jump buffer
float coyoteTimer = 0;
float jumpBufferTimer = 0;

onGroundedChanged(isGrounded):
if isGrounded:
coyoteTimer = COYOTE_TIME; // скидаємо при приземленні
else:
// починаємо спадати після втрати землі

onJumpPressed():
jumpBufferTimer = JUMP_BUFFER_TIME;

update(dt):
if isGrounded:
coyoteTimer = COYOTE_TIME;
else:
coyoteTimer -= dt;

jumpBufferTimer -= dt;

if jumpBufferTimer > 0 and coyoteTimer > 0:
DoJump();
jumpBufferTimer = 0;
coyoteTimer = 0;
```

3.1.5 Input mapping у Unity Input System та UE Enhanced Input

Сучасний підхід до вводу в іграх – action-based mapping: замість прив’язки «кнопка -> дія» в коді ми описуємо «дія -> набір прив’язок (bindings)», підтримуємо різні пристрої, контексти (menus/combat) та можливість переназначення (rebinding).

Ключові поняття action-based вводу

Action (дія): Jump, Attack, Interact, Move, Look, Pause.

Binding (прив’язка): Space / Gamepad South / Mouse Left.

Action Map або Context: набір дій для конкретного режиму (Gameplay, UI, PhotoMode).

Interactions/Triggers: tap/hold/double-tap, press/release, combo.

Rebinding: зміна прив’язки користувачем із збереженням у профілі (табл. 3.9).

Таблиця 3.9

Unity Input System vs Unreal Enhanced Input (коротке порівняння)

Критерій	**Unity Input System**	**UE Enhanced Input**
Модель	Input Actions + Action Maps	Input Actions + Mapping Contexts
Контексти	перемикання Action Map	додавання/пріоритизація Mapping Context
Тригери	Interactions (tap/hold)	Triggers (hold/press/chorded)
Модифікатори	Processors (deadzone, invert)	Modifiers (deadzone, scale, invert)
Rebinding	підтримується (UI/скрипти)	можливо (з UI/системою збереження)
Типовий пайплайн	Asset -> PlayerInput -> callbacks	EnhancedInputComponent -> events

Практичний мінімум налаштування (Unity)

Створити Input Actions asset (Gameplay: Move, Look, Jump, Interact).

Налаштувати bindings для Keyboard/Mouse та Gamepad.

Додати PlayerInput компонент, обрати поведінку (Send Messages / Invoke Unity Events / C# events).

Під’єднати callbacks до контролера та систем взаємодій.

Додати простий екран Debug (показ поточних значень Move/Look).

Практичний мінімум налаштування (Unreal Engine 5)

Створити Input Actions (IA_Move, IA_Look, IA_Jump, IA_Interact).

Створити Mapping Context (IMC_Gameplay) і додати bindings для KB/M та Gamepad.

У персонажі/контролері додати EnhancedInputComponent і прив’язати події до функцій.

Додавати/видаляти Mapping Context відповідно до стану (UI vs Gameplay).

Зробити простий Debug вивід (Print String / Debug Widget) для значень вводу (табл. 3.10) (рис. 3.4).

Таблиця 3.10

Навчальна таблиця до теми «Game feel та Input Systems: responsiveness, feedback loops, MDA»

| Input Actions | --> | Context/Map | --> | Gameplay Systems |
| (Jump/Move/...) | | (Gameplay/UI) | | (Controller/Interact) |
+-----_____ Rebinding / Modifiers _____/

Рис. 3.4. Action-based ввід: розділення «дія» та «прив’язка» (схема)

3.1.6 Відгук: micro-feedback (UI / VFX / Audio / Camera)

Micro-feedback - це короткі, точні сигнали, які підтверджують дію та пояснюють її наслідки. У сучасному геймдеві micro-feedback - основний інструмент підтримки control fantasy: гравець відчуває, що він керує системою, а система «розмовляє» з ним.

Важливий принцип: фідбек має бути не «красивим сам по собі», а функціональним - читабельним у русі, зрозумілим на різних екранах, узгодженим з правилами гри і доступним для різних користувачів (табл. 3.11).

Таблиця 3.11

Патерни micro-feedback і їх функція

Ситуація	**UI**	**VFX**	**Audio**	**Camera/ feel**	**Коментар**
Натиснув Jump	іконка/індикатор стану	пил/ шлейф	короткий «whoosh»	легкий impulse	підтвердження дії
Attack hit	плаваюче число/marker	hit spark	удар	hit-stop 30-60 мс	потрібна читабельність
Interact available	prompt/ підсвітка	outline/глоу	тихий сигнал	-	не перевантажувати
Interact failed	повідомлення /іконка	короткий flash червоний	звук відмови	-	краще ніж тиша
Low HP	пульсуючий HUD	віньєтка	серцебиття	легке хитання	контроль інтенсивності

Практичний чек-лист читабельності micro-feedback

Сигнал помітний за 1 секунду під час руху?

Сигнал не конфліктує з іншими сигналами (пріоритети)?

Сигнал працює без звуку (і навпаки)?

Контраст і розмір читабельні на 2-3 роздільностях?

Є фідбек на «відмову» (дія недоступна)?

3.1.7 MDA як інструмент аналізу «відчуття» (Mechanics-Dynamics-Aesthetics)

MDA (Mechanics-Dynamics-Aesthetics) - це модель, яка допомагає формалізувати зв’язок між тим, що ми реалізуємо (механіки), як система поводиться в реальному часі (динаміки) і що відчуває гравець (естетики). Для теми game feel MDA корисна тим, що дозволяє перетворити суб’єктивне «не відчувається» на структуру проблеми та перевірювані гіпотези.

Як застосовувати MDA для проблем responsiveness

Aesthetics: сформулюйте бажане відчуття (наприклад «tight», «snappy», «weighty», «precise»)

Dynamics: опишіть поведінку системи, яку гравець бачить (час реакції, ритм, частота помилок)

Mechanics: знайдіть конкретні правила/параметри/реалізацію, що породжують динаміки (coyote time, буфер, smoothing, таймінги)

Зробіть гіпотезу: «Якщо змінити X, то зміниться Y, і естетика наблизиться до Z» (табл. 3.12).

Таблиця 3.12

Міні-аналіз MDA (приклад для «tight jump» у платформері)

Рівень	Приклад опису
Aesthetics	відчуття точного стрибка без «зради», контроль у повітрі
Dynamics	гравець рідше падає через 1-2 кадри запізнення; стрибок спрацьовує «як очікується»
Mechanics	coyote time 0,12 с; jump buffer 0,15 с; tuned gravity curve; швидкий first feedback

Кейс-спостереження (Celeste / Hades)

Celeste демонструє естетику «точність + справедливість», що досягається через механіки forgiveness і дуже швидкий фідбек. Hades демонструє «швидкість + бойова читабельність», де на перший план виходять телеграфи, ритм ударів та миттєві підтвердження дії.

3.1.8 Параметризація: tunables, debug UI, швидкі тести

Щоб системно покращувати game feel, потрібно зробити його керованим. Практично це означає: винести ключові параметри в конфігурації (tunables), створити простий debug UI та організувати швидкі тести (A/B або порівняння пресетів). Без цього команда або студент витрачає час на «перекомпіляцію почуттів» і не може повторити успішний результат (табл. 3.13).

Таблиця 3.13

Приклади tunables для контролера/камери/фідбеку

Категорія	Параметри (приклади)	Коментар
Рух	speed, accel, decel, air control	впливає на «snappy» vs «floaty»
Стрибок	jump height, gravity, coyote time, buffer	керує forgiveness
Камера	follow damping, look sensitivity, FOV	надмірний damping = «липко»
Фідбек	hit-stop ms, shake intensity, VFX scale	важливо зберігати бюджет
Input	deadzone, smoothing, sensitivity	залежить від пристрою

Рекомендований debug UI мінімум

Поточні значення Move/Look (вектор) і активна схема вводу (KBM/Gamepad).

Стан персонажа: Grounded/Airborne/Attacking/Interacting.

Таймери forgiveness: coyoteTimer, jumpBufferTimer.

FPS та frame time (мінімум - через профайлер або простий лічильник).

Перемикач 2-3 пресетів параметрів (Preset A/B/C).

Порада: якщо ви не можете швидко перемкнути пресет і відразу відчути різницю - параметри або не ті, або фідбек нечитабельний. У хорошому тестовому стенді різниця між пресетами помітна за 20-30 секунд гри.

3.1.9 Типові помилки responsiveness і як їх діагностувати

Наведено поширені причини «поганого відчуття», які регулярно трапляються у студентських прототипах. Кожна проблема має симптоми, причину та типові виправлення. Важливо: виправляйте спочатку найсильніші проблеми і завжди перевіряйте результат плейтестом (табл. 3.14).

Таблиця 3.14

Помилки responsiveness: симптоми -> причини -> рішення

Симптом	Ймовірна причина	Швидке рішення	Перевірка
Дія «спрацьовує із запізненням»	VSync/низький FPS/важкий кадр	профілювати, прибрати спайки	до/після frame time
Керування «липке»	надмірний smoothing/damping	зменшити damping, зробити tunable	A/B пресет
Стрибок «не спрацьовує»	нема buffer/coyote	додати 0,12-0,15с	рахунок невдалих стрибків
Prompt миготить/спамить	нема hysteresis/priorities	додати стабілізацію таргетингу	відео+спостереження
Удар «не відчувається»	нема impact-фази	додати hit-stop+VFX+SFX	короткий плейтест
Гравець не розуміє відмову	нема negative feedback	звук/іконка/flash	чи бачить гравець причину

Міні-алгоритм діагностики включає наступні кроки

Зафіксуйте конкретний симптом (коли саме, у якому сценарії).
Перевірте FPS і наявність спайків (профайлер).
Перевірте «first feedback»: чи є він і чи вчасний.
Визначте, у якій фазі A/I/R проблема (нема телеграфу? нема impact?).
Перевірте forgiveness-таймери (coyote/buffer).
Зробіть одну зміну, повторіть тест, задокументуйте результат.

3.1.10 Міні-пайплайн тесту відчуття гри (feel testing pipeline)

Щоб підтримувати стандарт якості протягом семестру, рекомендується використовувати повторюваний пайплайн тестування. Це мінімізує «суб'єктивні суперечки» і переводить роботу в режим інженерного покращення: виміряли -> змінили -> перевірили (табл. 3.15).

Таблиця 3.15

Feel testing pipeline (узагальнення)

Етап	Зміст
Крок 1. Сформулювати ціль відчуття	1-2 речення естетики (snappy/weighty/precise) + 1 метрика (наприклад «first feedback < 100 ms»).
Крок 2. Побудувати тестові сценарії	3 короткі сцени/режими, що стабільно відтворюють ключову взаємодію.

Етап	Зміст
Крок 3. Інструментувати	Debug UI, лог таймерів, простий лічильник FPS/кадру, перемикач пресетів.
Крок 4. Плейтест (2-3 людини)	Короткий протокол: що відчули, де застрягли, що не зрозуміли.
Крок 5. Внести 1-3 зміни	Тільки найважливіше: або latency, або фідбек, або forgiveness.
Крок 6. Перевірити «до/після»	Порівняти відео/метрики, зафіксувати висновки.
Крок 7. Зафіксувати як стандарт	Описати параметри (Preset), додати в README і коміт.

Після кожної значущої ітерації прототипу рекомендується виконувати цей пайплайн хоча б у скороченому вигляді. Так ви забезпечите стабільний розвиток проєкту й уникнете накопичення технічного та дизайн-боргу.

Питання для самоперевірки

1. Дайте визначення game feel. Які компоненти формують відчуття керування?
2. Що означає принцип «first feedback < 100 ms»? Який саме фідбек може бути «першим»?
3. Назвіть основні джерела input latency в ланцюжку input -> update -> render -> display.
4. Як FPS та VSync впливають на затримку від натискання до показу?
5. Поясніть фази anticipation/impact/recovery та наведіть приклад для стрибка або атаки.
6. Що таке coyote time та jump buffering? Які ризики надто великих значень?
7. У чому різниця між action-based input mapping і «жорсткою» прив’язкою кнопок у коді?
8. Які елементи має містити micro-feedback, щоб бути читабельним у русі?
9. Як MDA допомагає діагностувати проблеми з responsiveness?
10. Що таке tunables і навіщо робити debug UI у прототипі?
11. Наведіть 3 типові помилки responsiveness та способи їх виправлення.
12. Опишіть мінімальний feel testing pipeline: які кроки є обов’язковими?

Практичний кейс до теми (120-180 хв): «Responsiveness Test Bench»

Мета кейсу - створити невеликий тестовий стенд (test bench), який дозволяє вимірювано перевіряти шлях «input -> state -> feedback» у трьох різних сценаріях, швидко порівнювати пресети параметрів та фіксувати результати ітерацій.

Вхідні умови

Оберіть рушій: Unity (LTS) або Unreal Engine 5.

Створіть окрему сцену/рівень «TestBench_01» з мінімальною геометрією та зрозумілими орієнтирами.

Увімкніть action-based input (Unity Input System або UE Enhanced Input) і налаштуйте керування KB/M та Gamepad (табл. 3.16).

Сценарії тесту (3 обов'язкові)

Таблиця 3.16

Сценарії «Responsiveness Test Bench»

Сценарій	**Зміст**
Сценарій A: Movement & Jump	рух + стрибок з вимірюваним first feedback, coyote time і jump buffer.
Сценарій B: Attack/Ability	одна дія з фазами anticipation/impact/recovery, із hit-stop або camera impulse.
Сценарій C: Interact Prompt	взаємодія з об'єктом: підсвітка/підказка, підтвердження, фідбек відмови.

Вимоги до реалізації (мінімум)

Перший фідбек у кожному сценарії має бути очевидним та швидким (ціль - < 100 мс)

Debug UI має показувати: поточний стан, значення вводу, таймери coyote/buffer (за потреби), активний пресет

Додайте 2-3 пресети параметрів (Preset A/B/C) і можливість перемикання під час гри

Зробіть мінімум один елемент negative feedback (коли дія недоступна).

Забезпечте стабільний FPS у тестовій сцені (без «важких» пост-ефектів на старті) (табл. 3.17).

План виконання завдання (орієнтовно 150 хв)

Таблиця 3.17

Рекомендований таймплан виконання

Час	**Дія**
0-20 хв	Створення сцени TestBench, підключення вводу, базовий рух/камера.
20-55 хв	Сценарій A: рух+стрибок, додати coyote time і jump buffer, зробити debug UI.
55-90 хв	Сценарій B: дія/атака з A/I/R, додати impact-фідбек (VFX+SFX+hit-stop або camera impulse).
90-120 хв	Сценарій C: Interact prompt, підсвітка/підказка, підтвердження та відмова.
120-150 хв	Пресети A/B/C, плейтест 2-3 людини (або self-test), фіксація результатів і короткий постмортем.
150-180 хв	Полірування: вирівнювання читабельності, зменшення спаму фідбеку, стабілізація FPS.

Що необхідно для здачі завдання

1. Проєкт/сцена (Unity package або UE project/screenshot рівня).
2. Відео-демонстрація 20-40 секунд (з перемиканням пресетів).
3. README (1-2 стор.): короткий опис системи, список tunables, як запускати тест.
4. Табличка «Preset A/B/C» (параметри + короткий ефект від зміни).
5. Міні-постмортем 150-250 слів: що покращили, що не вийшло, що робитимете далі (табл. 3.18).

Критерії оцінювання зазначені у таблиці

Таблиця 3.18

Оцінювання кейсу

Критерій	**Бали**	**Опис**	**Як перевіряється**
Responsiveness	0-4	first feedback помітний і швидкий у 3 сценаріях; керованість не «липка».	візуально/відео + self-test
Фідбек A/I/R	0-4	є телеграф, impact і recovery; узгоджені канали UI/VFX/Audio/Camera.	спостереження
Архітектура і чистота	0-4	дії оформлені як actions; стани зрозумілі; немає «хаосу» у коді/графах.	перегляд проєкту
Tunables + Debug	0-4	є пресети A/B/C, debug UI, можливість швидкого порівняння.	демо
Документація	0-4	README, таблиця параметрів, короткий постмортем з висновками.	перевірка файлів

Додаткове завдання (за бажанням)

- Зробити простий лічильник latency (наближено): міряти час від натискання до появи UI індикатора.
- Додати режим «Accessibility check»: вимкнути звук або знизити контраст і перевірити, чи фідбек ще читабельний.
- Додати короткий лог подій (console) і експортувати у текстовий файл для аналізу (табл. 3.19).

Рекомендовані програми та сервіси

Таблиця 3.19

Інструменти для виконання теми

Категорія	**Рекомендації**
Unity (LTS)	Input System, Cinemachine (за потреби), Profiler, Recorder
Unreal Engine 5	Enhanced Input, Sequencer, Niagara (за потреби), Unreal Insights
Аудіо	Audacity; (опційно) FMOD або Wwise
Контроль версій	Git + Git LFS / Plastic SCM; (для UE команд) Perforce Helix Core
Захоплення відео	OBS; вбудовані Recorder/Sequencer/MRQ
Дебаг	Вбудовані профайлери; прості debug widgets/overlay

Рекомендовані джерела та матеріали

1. Unity Documentation: Input System (official docs), PlayerInput, rebinding.
2. Unreal Engine Documentation: Enhanced Input (official docs), input mapping contexts.
3. Матеріали про game feel і responsiveness: виступи GDC (добірка за ключовими словами «game feel», «input», «controls»).
4. Практичні розбори: Game Maker's Toolkit (GMTK) - теми про керування, читабельність, фідбек.
5. Додатково: Steve Swink «Game Feel» (для поглиблення), Jesse Schell «The Art of Game Design» (про MDA/MDA-like аналіз).

Коротка пам'ятка студенту до теми

1) Визнач відчуття (1-2 слова): snappy / weighty / precise.
2) Забезпеч first feedback < 100 мс у ключових діях.
3) Розклади фідбек на A/I/R і синхронізуй канали.
4) Додай forgiveness (coyote/buffer) і зроби параметрами.
5) Зроби test bench + debug UI, порівняй пресети, зафіксуй висновок.

Тема 3.2. Character Controller & Camera: movement, game feel та virtual cinematography

Мета: сформувати практичні та теоретичні засади побудови керованості (game feel) через системи руху персонажа та камери.

За результатами вивчення теми студент набуває компетентності:

- проєктувати рух як керовану систему параметрів (tunables), а не «магічні числа»;
- забезпечувати стабільний grounding, коректну роботу на схилах/сходинках, передбачувані арки стрибка;
- застосовувати прийоми forgiveness (coyote time, input buffering, jump forgiveness) без «поломки чесності»;
- збирати камеру як rig (follow/orbit/aim), керувати blending, демпфуванням, колізіями камери;
- використовувати камеру як інструмент читабельності й драматургії, не створюючи дискомфорту;
- організовувати швидкі тести «відчуття керування» і ітераційно покращувати результат.

Очікувані результати навчання

- пояснювати різницю між responsiveness та control fidelity і переводити вимоги в tunables;
- будувати цикл руху: прискорення/гальмування, grounded/air state, керування в повітрі;
- налаштовувати grounding, схили, сходинки і діагностувати типові «симптоми» (jitter, sliding, sticky corners);
- застосовувати coyote time, buffering, jump forgiveness та обґрунтовувати значення для game feel;

- збирати камеру в Unity/UE5: follow/orbit/aim, blending, collision/confiner, комфортні параметри;
- впроваджувати камерний micro-feedback (shake/impulse/FOV kick) у межах «camera budget»;
- створювати тестову сцену, 3 пресети feel та короткий постмортем за результатами плейтесту.

Ключові терміни: Character controller, kinematic vs physics, grounding, slope limit, step offset, acceleration curve, deceleration/braking, friction, air control, jump arc, time-to-apex, coyote time, input buffering, jump cut/short hop, camera rig, follow/orbit/aim camera, spring arm, confiner, damping, dead zone, blending, FOV, impulse/shake, shot language, composition, readability, motion sickness, debug overlay.

3.2.1 Вимоги до контролера: передбачуваність, читабельність, «чесні» компроміси

Контролер - це «рука» гравця. У прототипах ціль - не фізична точність, а передбачуваність і керованість. Відчуття руху визначається не «максимальною швидкістю», а кривою зміни швидкості, стабільним контактом із поверхнею та ясним переходом станів.

Важливо відрізняти поняття

Responsiveness - як швидко і однозначно гра реагує на введення.

Control fidelity - як точно гравець може «дозувати» рух (мікрокорекції, плавний поворот, контроль інерції).

Надмірне згладжування (великий damping) часто «вбиває» responsiveness. У навчальних роботах: спочатку чіткість, потім «краса» (табл. 3.20).

Таблиця 3.20

Kinematic vs Physics controller

Критерій	Kinematic (кінематичний)	Physics (rigidbody/physics)	Рекомендація
Передбачуваність	Висока	Залежить від контактів/тертя	Починати з kinematic
Колізії	Керуєте правилами	Реалістичні, але jitter-ризик	Фізику вводити після MVP
Схили/сходинки	Налаштовуються параметрами	Може «підкидати/залипати»	Для першого прототипу - kinematic
Складність	Нижча	Вища	Не ускладнювати без потреби
Реплікація/ мережа	Простіша	Складніша	У навчанні - спрощувати

3.2.2 Grounding: контакт із землею, схили, сходинки - як уникати «ковзання» і провалів

Grounding - визначення «на землі/в повітрі» + реакція на поверхню. Типові проблеми можуть бути наступними

- «втрачає землю» на дрібних нерівностях;
- ковзає по схилу або «підстрибує» на сходинках;
- має випадкові поштовхи через колізії.
- Практичний підхід: поєднати
- капсульну колізію;
- додатковий ground-check (ray/sphere/capsule cast);
- допуск sticky-to-ground (невелике «приклеювання» при малих швидкостях);
- обмеження max slope + керований step offset (рис. 3.5).

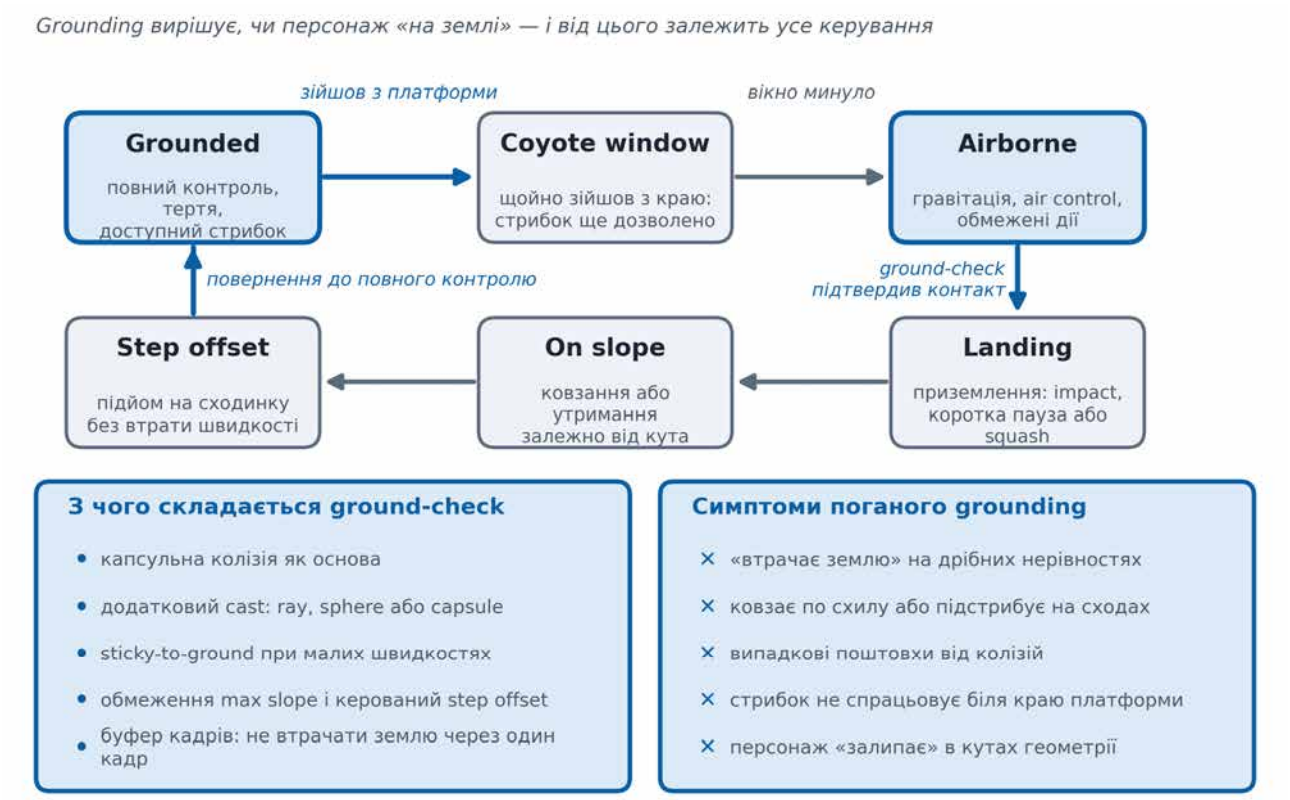


Рис. 3.5. Узагальнена схема grounding і переходів станів

```
[Input] -> [Desired velocity] -> [Ground check] -> [State: grounded/air]
      | |
      v v
      [Slope/step rules] [Air control + gravity]
      \ /
      v v
[Move & collide] -> [Update timers]
```

3.2.3 Прискорення/гальмування: криві руху, «відчуття ваги» і контроль

Гравець «відчуває» не speed, а accel/decel, turn rate, friction/braking. Рекомендація: завжди мати tunables і не «зашивати» значення (табл. 3.21).

Таблиця 3.21

Мінімальний набір tunables для руху

Група	Параметр	Навіщо	Стартовий діапазон
Рух	Max Speed	межа швидкості	2D: 3-8; 3D: під жанр
Рух	Acceleration	«набір» швидкості	2D: 15-60
Рух	Decel/Braking	гальмування	2D: 20-80
Поворот	Turn Rate	швидкість зміни напрямку	6-14
Повітря	Air Control	керування у стрибку	0.1-0.6

Практичне правило: зробіть 2-3 пресети

Arcade (швидко/чітко), Weighty (вага/інерція), Precision (контроль/без різких ривків).

3.2.4 Стрибок як дизайн-система: арка, time-to-apex, «читабельна гравітація»

Стрибок має бути передбачуваним. Зручно задавати його не «силою», а: Jump Height (висота), Time-to-Apex (час до вершини), а гравітацію виводити з цих значень (рис. 3.6)

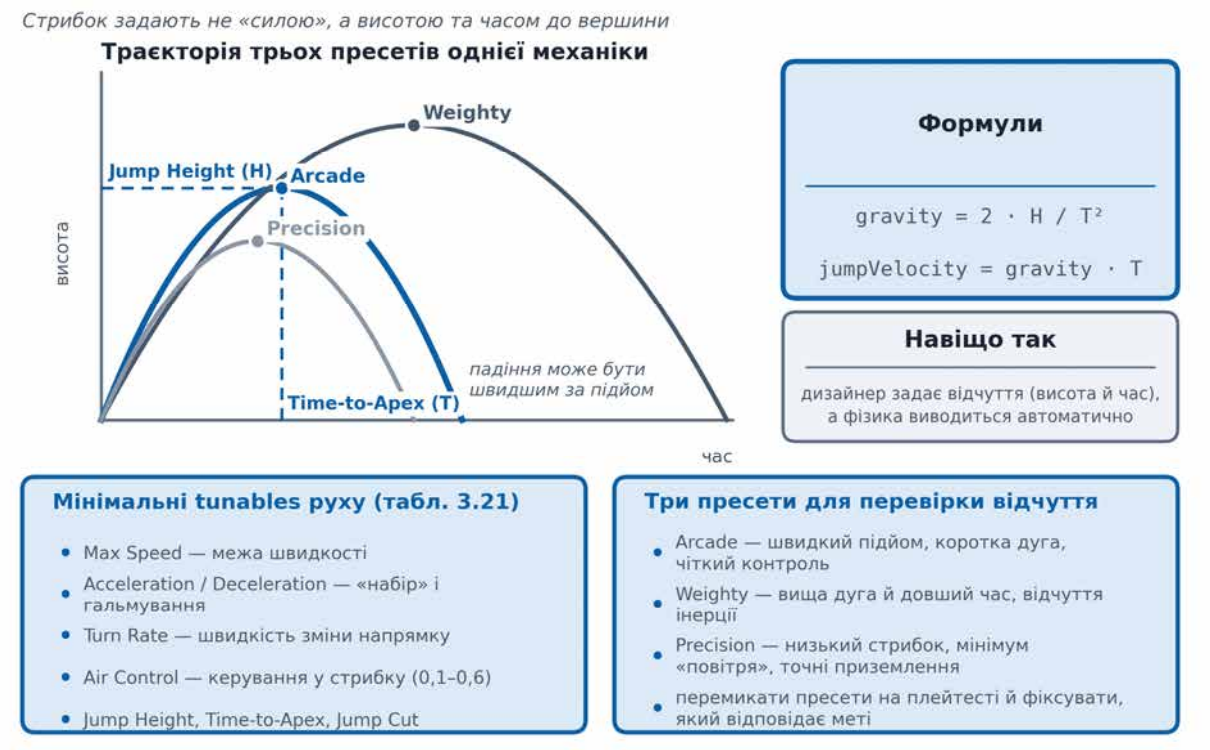


Рис. 3.6. Параметричний стрибок

Given: H = JumpHeight, T = TimeToApex

$$gravity = 2*H / (T*T) \quad (3.1)$$

$$jumpVelocity = gravity * T \quad (3.2)$$

Це дозволяє швидко робити «feel»-варіанти: короткий/різкий або плавний/повітряний.

3.2.5 Forgiveness: coyote time, buffering, jump forgiveness – чому Celeste «відчувається чесно»

Forgiveness – це допуски, які зменшують фрустрацію без «читерства».

Coyote time: після сходу з платформи ще можна стрибнути (50-120 ms).

Input buffering: натиснув стрибок до приземлення – він спрацює при контакті (50-120 ms).

Jump forgiveness / jump cut: варіативність висоти або швидше падіння при ранньому відпусканні кнопки (табл. 3.22; рис. 3.7).

Таблиця 3.22

Forgiveness – користь і ризики

Прийом	Що виправляє	Ризик	Запобіжник
Coyote time	пізній стрибок	«магія»	мале вікно + лише якщо був grounded
Buffering	ранній натиск	авто-стрибки	очищати буфер після спрацювання
Jump cut	керування висотою	різкість	плавні множники, тест комфортності

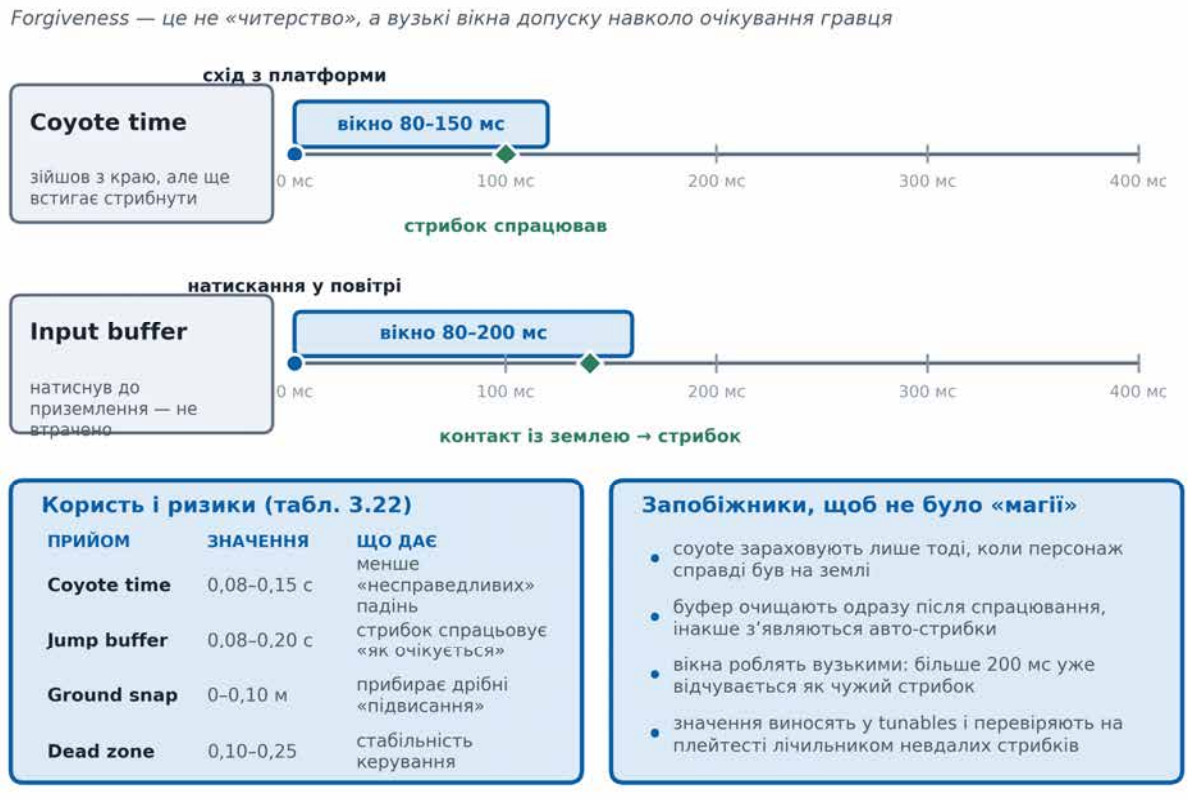


Рис. 3.7. Таймлайн coyote/buffer

Grounded – edge – Airborne
|<- coyote ->|
Airborne – landing – Grounded
|<- buffer ->|

3.2.6 Камера як інструмент навігації й читабельності: «камера не повинна боротися з гравцем»

Камера виконує 3 ролі

• навігація (де я/куди/що важливо),
• інформація (читабельність подій),
• емоція (драматургія, настрій).

Погана камера «вбиває» керованість: різкі повороти, колізійні стрибки, надмірна інерція, хаотичні шейки (табл. 3.23).

Таблиця 3.23

Типи камер

Тип	Плюси	Мінуси	Де краще
Follow	стабільно	може бути «пласко»	платформер/прототип
Orbit (3rd person)	контроль огляду	motion sickness ризик	3D екшн
Aim/OTS	точність	менше огляду	шутер/приціл
Fixed	сильна кіномова	складна навігація	хорор/пазли
Cinematic	контроль кадру	не інтерактивно	трейлер/катсцени

3.2.7 Unity Cinemachine / UE5 Camera Rigs: збірка, blending, collision

Unity (Cinemachine)

• Virtual Camera (Follow/Aim/Composer)
• Cinemachine Brain (blending)
• Confiner/Collider (обмеження)
• Noise/Impulse (фідбек)

UE5

• SpringArm + Camera (колізії/плавність)
• CameraManager/CameraStack (за потреби)
• Sequencer (кінематографія/запис) (рис. 3.8)

3.2.8 Віртуальна кіномова: композиція, FOV, кадрування, «читабельність у русі»

Прості правила для створення прототипу (прототипування)

персонаж не «впирається» у край кадру;

зона дії (вороги/цілі) - в «комфортному» полі зору;

FOV: менший = напруга/близькість; більший = швидкість/простір (але не зловживати);

різкі зміни кадру - тільки по події й з коротким blending.

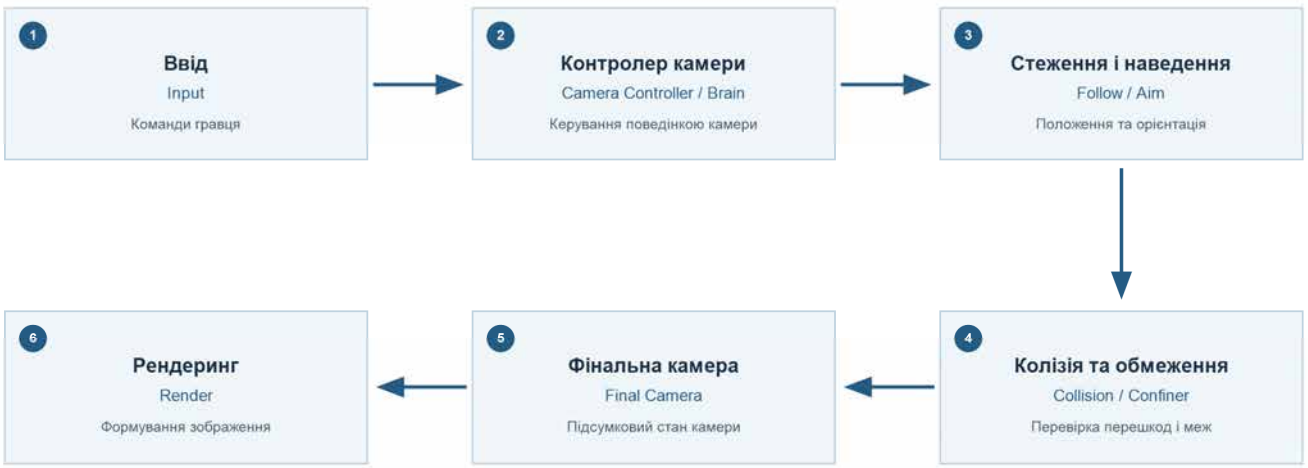


Рис. 3.8. Стек камерного ригу

Input -> Camera Controller/Brain -> (Follow/Aim) -> Collision/Confiner -> Final Camera -> Render

3.2.9 Камерний micro-feedback: shake/impulse/FOV kick і «camera budget»

Камера - канал фідбеку: удар, приземлення, dash. Але потрібен budget (ліміти), інакше буде дискомфорт (табл. 3.24).

Таблиця 3.24

Camera feedback budget (стартові правила)

Ефект	Коли доречно	Ліміт	Антипатерн
Impulse/Shake	удар/приземлення	0.05-0.18 s	постійний shake
FOV kick	dash/sprint	+3..+8 deg на 0.1-0.2 s	довга «пульсація»
Zoom-out	бій/натовп	0.2-0.6 s плавно	різкі стрибки
Vignette	low HP/status	ледь помітно	перекриває HUD

3.2.10 Типові помилки (і як лікувати) + міні-пайплайн тесту «feel»

Типові проблеми

Jitter: конфлікт update loops або колізії камери -> interpolation/узгодження update + collision tuning.

Sticky corners: капсула чіпляється -> skin width/offset, простіші колізійні меші.

Camera clipping/pumping: погана колізія spring arm/confiner -> правильні маски колізій, мінімальна дистанція, damping.

Over-smoothing: «ватність» -> менший damping, окремі профілі для режимів.

Direction mismatch: рух не від камери -> рух у просторі камери (forward/right) + ясний aim mode (рис. 3.9).



Рис. 3.9. Mini-pipeline test

Build test scene -> Add debug overlay -> Create 3 presets -> Run same route -> A/B compare -> 1 change at a time -> Verify

Питання для самоперевірки

1. Чим відрізняються responsiveness і control fidelity?
2. Які 3 причини «поганого grounding» у 3D?
3. Як step offset впливає на сходинки?
4. Чому зручніше задавати стрибок через height + time-to-apex?
5. Як coyote time зменшує фрустрацію?
6. Який антипатерн може створити input buffering?
7. Які 3 типи камер найчастіші у прототипах і чому?
8. Навіщо blending між режимами камери?
9. Коли доречний FOV kick і чому він має бути коротким?
10. Назвіть 2 причини camera jitter і 2 способи зменшення.
11. Які параметри треба показати в debug overlay для руху/камери?
12. Опишіть простий A/B тест між двома пресетами.

Практичний кейс до теми (120-180 хв): «Controller + Camera Feel Pack»

Мета: зробити «базу семестру» - керування + камера + 3 feel-пресети + тестова сцена + короткий звіт.

Сценарії (обрати 1 або 2)

1) (2D/2.5D): 3 стрибки + зона зі схилом/сходинками
2) (3D third-person): арена з перешкодами + dash/ухилення

C (Aim mode): Exploration -> Aim (OTS) з blending

Кроки для виконання завдання

• Зробіть тестову сцену з 3 ділянками: пряма/схил+сходинки/прогалини.
• Реалізуйте рух з tunables (табл. 3.2).
• Додайте forgiveness: coyote + buffer (50-120 ms).
• Зберіть камеру (follow/orbit/aim) + collision/confiner.
• Додайте 1-2 micro-feedback (імпульс приземлення, FOV kick на dash).
• Debug overlay: preset, grounded, timers, speed, FOV, shake intensity.
• Зробіть 3 пресети: Arcade / Weighty / Precision.
• Запишіть 20-40 сек на кожен пресет одним маршрутом.
• Короткий постмортем (150-250 слів).

Що необхідно здати

• Проєкт Unity або UE5 + сцена тесту
• README (1-2 стор.) з параметрами/керуванням
• 3 короткі відео/фрагменти (Arcade/Weighty/Precision)
• Міні-постмортем

Критерії оцінювання до теми

Responsiveness, стабільність grounding, читабельність камери, різниця пресетів, наявність debug overlay, охайна документація.

Рекомендовані програми та сервіси

1. Unity LTS: CharacterController/Rigidbody movement, Cinemachine, Input System, Profiler, Recorder
2. UE5: Character Movement Component, Enhanced Input, SpringArm Camera, Sequencer, Unreal Insights
3. OBS Studio (запис), DaVinci Resolve (за бажанням)
4. VCS: Git + Git LFS / Plastic SCM; Perforce (за потреби для UE-команд)
5. Docs/Diagrams: Notion/Google Docs, diagrams.net

Рекомендовані джерела та матеріали

1. Unity Manual/Docs: CharacterController, Rigidbody movement, Input System, Cinemachine
2. Unreal Engine Docs: Character Movement Component, Enhanced Input, Cameras, Sequencer
3. GDC talks (game feel, camera design, platformer controls)
4. Кейси для аналізу: Celeste (forgiveness), Hades (responsiveness + читабельність), Mario / Hollow Knight (jump feel), God of War 2018 / RE4 (camera language)

Тема 3.3. Архітектура систем взаємодії: input - state - feedback

Мета: сформувати системне уявлення про проєктування та реалізацію взаємодій у відеоіграх як керованого конвеєра «ввід -> стан -> зворотний зв'язок». Навчитися будувати масштабовану архітектуру interactables, систем таргетингу, контекстних підказок (prompts) і реакцій (UI/VFX/Audio), що підтримує ітеративну розробку, тестування та розширення без «хаосу» у коді або блюпрінтах.

Очікувані результати навчання

- пояснювати модель взаємодії «input -> state -> feedback» та застосовувати її для аналізу ігрових сценаріїв;
- проєктувати інтерфейси/компоненти взаємодії (Interactable) та повторно використовувані модулі (Interactor/Interaction Manager);
- реалізовувати таргетинг і фокус (raycast/overlap/cone/priority) та правила вибору цілі;
- створювати систему контекстних підказок (prompt system) з підтримкою ремапінгу клавіш/кнопок та доступності;
- використовувати подієві підходи (Observer, Event Bus, dispatchers) для інтеграції UI/VFX/Audio без жорстких залежностей;
- проєктувати стан-машини (FSM) для взаємодій та обробляти переривання (cancel/interrupt/lock);
- будувати data-driven взаємодії (конфіги, теги, таблиці параметрів) і забезпечувати розширення на 3+ дії;
- проводити відладку взаємодій: debug overlay, логи, тестові сцени, чек-листи;
- оформлювати коротку технічну документацію (діаграма потоку, опис компонентів, список подій).

Ключові терміни: Interaction, input action, context, affordance, interactable, interactor, focus, targeting, raycast/overlap, priority rules, prompt, diegetic prompt, state machine (FSM), cooldown, lock, interrupt, event bus, observer, dispatcher, message, decoupling, data-driven, config, gameplay tags, UX feedback, readability, accessibility.

3.3.1 Взаємодія як система: affordance -> action -> feedback

У сучасному геймдеві взаємодія розглядається як окремий «продукт усередині продукту»: вона має бути зрозумілою, надійною, повторюваною та вимірюваною. Базова формула: гравець бачить можливість (affordance), виконує дію (action) і отримує підтвердження (feedback). Якщо будь-яка ланка слабка - взаємодія відчувається «ламаючою» або «несправедливою».

Практичне правило курсу: для кожної взаємодії описуємо 3 речі, а саме

Що гравець має зрозуміти ДО натискання (візуальний сигнал/позиція/контекст)?

Що саме відбувається ПІСЛЯ натискання (стан, наслідки, обмеження)?

Яке підтвердження отримує гравець (UI/VFX/Audio/анімація/фізика)? (рис. 3.10)

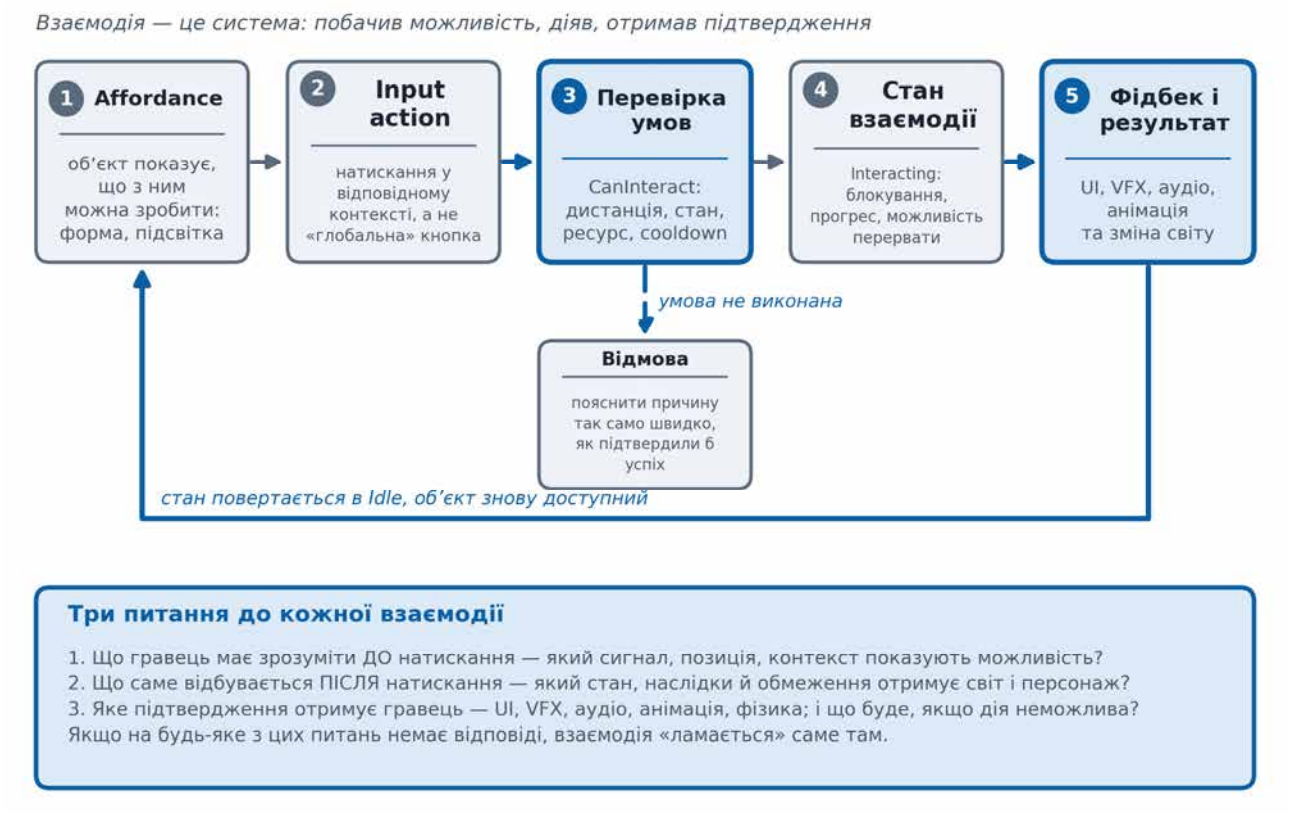


Рис. 3.10. Конвеєр взаємодії «input -> state -> feedback» (узагальнена схема)

[Input Action] -> [Interactor] -> [Target Selection] -> [Interaction State/FSM] -> [Effect/Result]
| |
v v
[Prompt UI] [Feedback: UI/VFX/Audio]

3.3.2 Архітектурні принципи: модульність, повторне використання, мінімальні залежності

Щоб взаємодії масштабувалися, важливо рознести відповідальність на модулі. Типова помилка студентських проєктів – логіка взаємодії «розмазана» між персонажем, UI і об'єктами рівня. Натомість рекомендується така структура:

• Interactor (на гравцеві): зчитує input, визначає фокус/ціль, викликає interaction API;

• Interactable (на об'єктах): описує, що можна зробити, за яких умов, і які події/результати виникають;

• Prompt System: відображає підказки незалежно від конкретної логіки об'єкта;

• Feedback Layer: під'єднання UI/VFX/Audio через події або повідомлення;

• Data/config: правила, тексти, іконки, параметри – винесені з коду (табл. 3.25).

Таблиця 3.25

Розподіл відповідальностей (рекомендована модель)

Компонент	**Відповідальність**	**Антипатерн**
Interactor	Input + вибір цілі + виклик взаємодії	UI логіка всередині Interactor
Interactable	Умови/дії/стани об’єкта	Прямі посилання на HUD/звуки
Prompt System	Показ підказки (текст/іконка/кнопка)	Підказка «зашита» в кожен об’єкт
Feedback Layer	Відгук (UI/VFX/Audio) через події	Кожна дія напряму запускає ефекти
Config/Data	Теги, тексти, параметри, баланс	Тисячі «магічних» значень у коді

3.3.3 Interactable API: інтерфейс/компонент і мінімальний контракт

Interactable – це контракт між гравцем (Interactor) і об’єктом. Мінімальний контракт повинен включати:

CanInteract (interactor) – чи дозволена взаємодія (умови, дистанція, ключ, cooldown);

GetPrompt – що показати в підказці (текст, іконка, тип натискання: tap/hold);

Interact (interactor) – виконання дії;

OnFocusEnter/OnFocusExit – події фокусу (highlight, outline, звук).

У Unity це може бути C# interface + MonoBehaviour, в UE – Blueprint Interface + Actor Component. Важливо: Interactable не «знає» про конкретний HUD – він лише віддає дані й події.

Приклад (псевдо) мінімального API

```
interface Iinteractable {
 bool CanInteract(Interactor who);
 PromptData GetPrompt(Interactor who);
 void Interact(Interactor who);
 void FocusEnter(Interactor who);
 void FocusExit(Interactor who);
}
```

3.3.4 Таргетинг і фокус: raycast/overlap/cone та правила пріоритезації

Вибір цілі (target selection) повинен бути передбачуваним. У 3D найчастіше використовується raycast з камери плюс додаткові правила. У 2D – overlap зона або «найближчий у напрямку».

Ключові параметри таргетингу: дистанція, кут, visibility, пріоритет, стабільність (щоб не «скакало») (табл. 3.26; рис. 3.11).

Таблиця 3.26

Методи таргетингу та де вони доречні

Метод	Переваги	Ризики/обмеження	Рекомендовано
Raycast (camera)	Точно, інтуїтивно для 3D	Проблеми з дрібними об’єктами	FPS/3rd-person
Overlap Sphere/Box	Стабільно, просто	Може вибирати «не те»	Платформери, loot
Cone/Frustum	Підтримує пріоритет у напрямку	Потрібні правила вибору	Бій/взаємодія в натовпі
Priority + Sticky target	Мало «миготіння»	Потрібно налаштовувати	Складні сцени

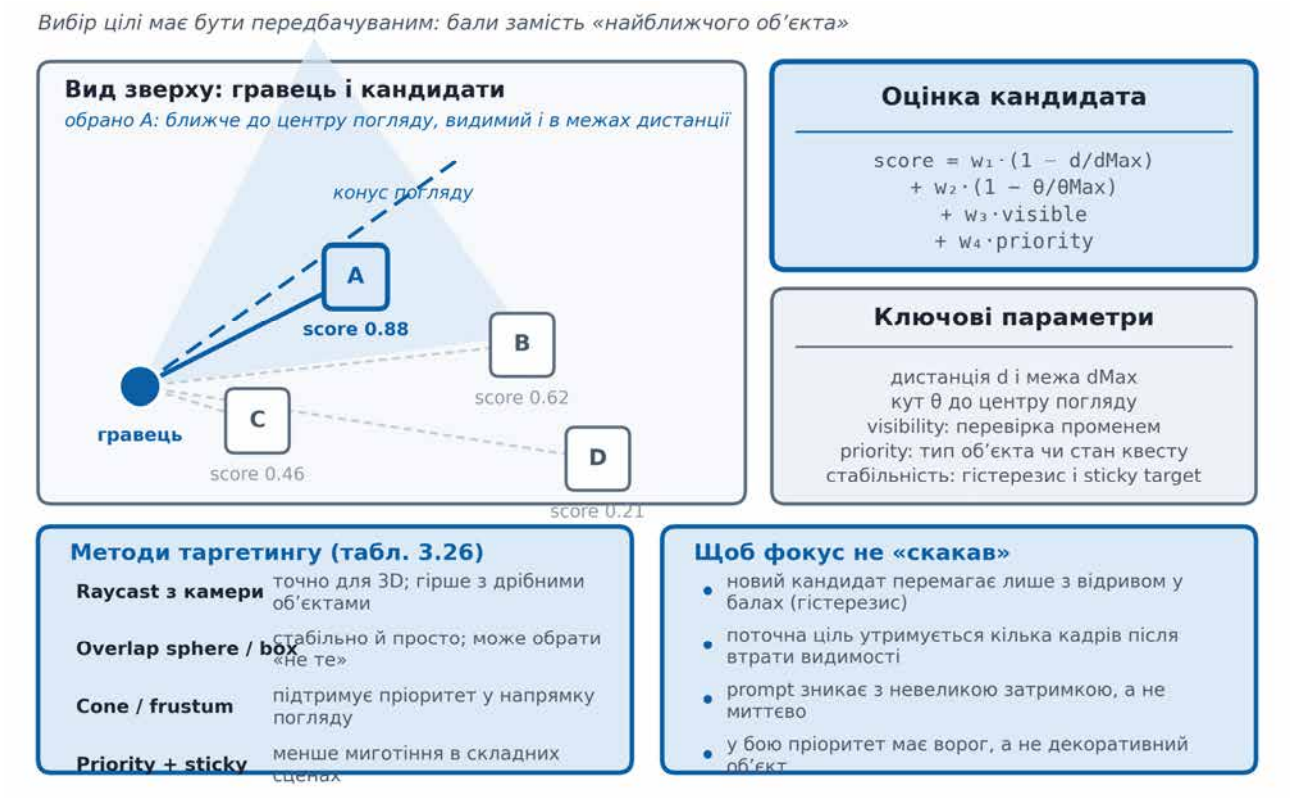


Рис. 3.11. Приклад пріоритету цілі (score-based selection)

score = w_dist * (1 – dist/maxDist) + w_angle * (1 – angle/maxAngle) + w_tag * priorityTag
choose target with max(score) and apply hysteresis (sticky) to avoid flicker

3.3.5 Prompt System: контекстні підказки без спаму та з підтримкою ремапінгу

Підказка (prompt) – це частина UX взаємодії. Вона має бути контекстною, короткою і не перетворюватися на «спам». Система prompts повинна підтримувати: текст + іконку, режим натискання (tap/hold), джерело input (клавіатура/геймпад), локалізацію, а також відображення реальної прив’язки кнопки після ремапінгу.

Практичні правила якісного prompt UI

- Показувати підказку лише для поточної цілі (focus target) і лише якщо CanInteract = true.
- Не перекривати критичні HUD елементи та не миготіти (debounce 100-200 ms).
- Текст – короткий (2-4 слова) + дія (наприклад: «E – Відкрити»).
- Підказка повинна бути читабельною в русі: контраст, розмір, фон/обводка.
- Для hold-взаємодій показувати прогрес (кільце/смужка) (табл. 3.27).

Таблиця 3.27

Шаблон PromptData (мінімальний склад)

Поле	**Тип**	**Приклад**	**Навіщо**
ActionId	string/tag	Interact/Open	зв’язок з input map
DisplayText	string	Відкрити	локалізований текст дії
Icon	sprite/texture	key_E / button_X	візуальна підказка
Mode	enum	Tap/Hold	тип виконання
HoldTime	float	0.4	для прогресу
Priority	int	10	для вирішення конфліктів

3.3.6 Події та відокремлення (decoupling): Observer, Event Bus, dispatchers

Найбільша архітектурна користь досягається тоді, коли Interactable не запускає ефекти напряму, а лише генерує подію на рівні «що сталося». UI/VFX/Audio підписуються на ці події й реагують. Це дозволяє міняти презентацію без зміни логіки.

Поширені варіанти реалізації

Unity: C# events/delegates, UnityEvent, ScriptableObject Event Channels.

UE5: Event Dispatchers, Gameplay Tags + повідомлення, Blueprint Interfaces.

Універсально: Command/Observer pattern, message queues (для складних систем) (рис. 3.12).

3.3.7 FSM для взаємодій: стани, блокування, переривання, cooldown

Щоб взаємодії не конфліктували одна з одною, потрібна мінімальна станова модель. Наприклад: Idle -> Focusing -> Interacting -> Cooldown. Для деяких дій потрібні блокування (lock) або переривання (interrupt), наприклад під час анімації (рис. 3.13).

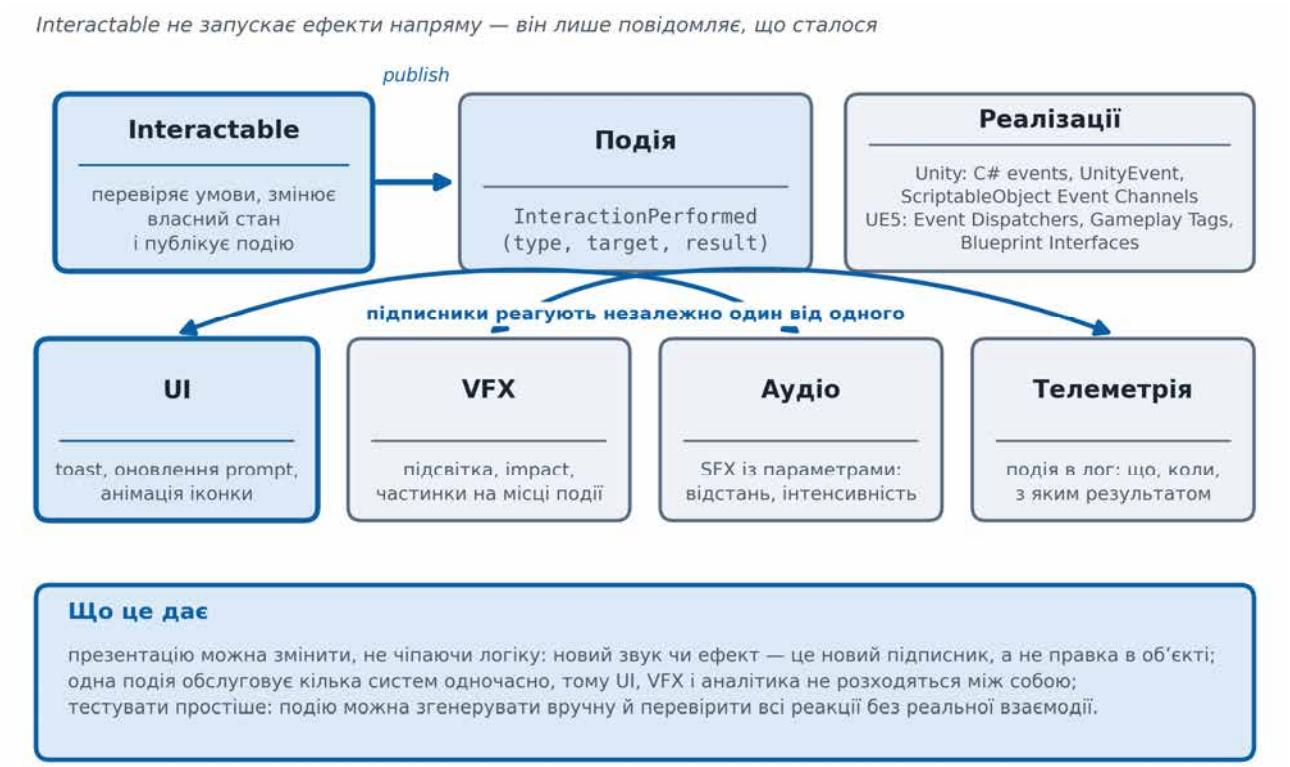


Рис. 3.12. Подієва інтеграція взаємодії та фідбеку

Interactable -> raises event «InteractionPerformed(type, target, result)»
UI listener -> shows toast / updates prompt / plays icon animation
VFX listener -> spawns highlight/impact effect
Audio listener -> plays SFX with parameters (distance, intensity)

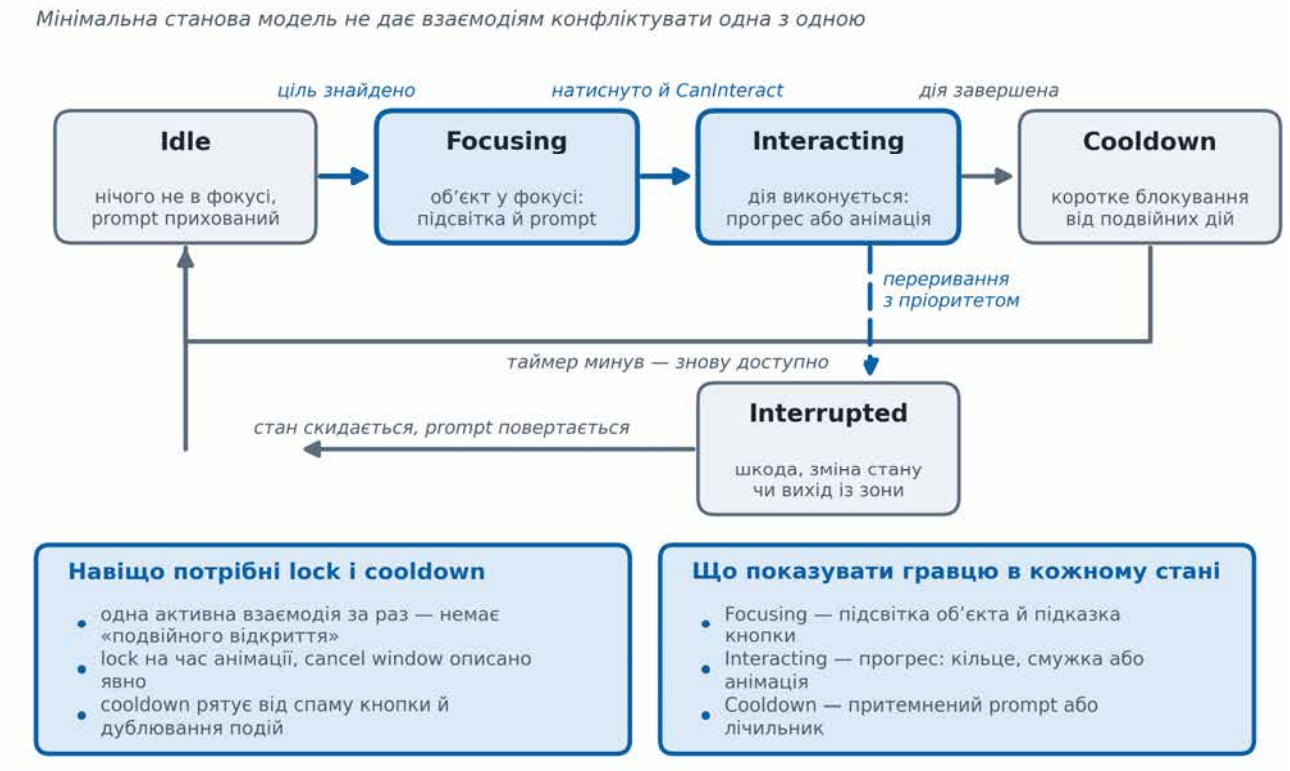


Рис. 3.13. Мінімальна FSM взаємодії (приклад)

Idle
 -> Focus(target) -> Focusing
Focusing
 -> InteractPressed -> Interacting
 -> TargetLost -> Idle
Interacting
 -> Success/Fail -> Cooldown or Idle
Cooldown
 -> timer done -> Idle

3.3.8 Data-driven взаємодії: теги, конфіги, таблиці параметрів

Data-driven підхід зменшує кількість «if-else» і спрощує розширення. Приклади: Interactable має тип (Door, Pickup, Terminal), а правила (текст, звук, час hold, вимоги) лежать у конфігах.

Практичні інструменти

• Unity: ScriptableObjects для InteractionDefinition, таблиці/CSV для балансних параметрів.

• UE5: Data Assets, Data Tables, Gameplay Tags.

• Обидва: іменування й папки для контенту + рев'ю конфігів.

3.3.9 Відладка і тестування взаємодій: debug overlay, лог, тестова сцена

Взаємодії потрібно тестувати системно, інакше помилки проявляються «випадково». Рекомендовано створити тестову сцену з 6-10 об'єктами різних типів і сценаріями (tap/hold, locked/unlocked, cooldown, multiple targets).

• Debug overlay: current target, distance, canInteract, actionId, holdProgress, state.

• Interaction log: останні 20 подій із часом і результатом.

• Чек-лист тесту: фокус працює стабільно, підказка не мигає, remap відображається, hold прогрес коректний.

3.3.10 Production-ready прототип взаємодій: критерії готовності

«Готовність» означає не ідеальну систему, а мінімально надійну і масштабовану базу. Нижче наведено стислий чек-лист готовності interaction framework.

• Є Interactor + Interactable контракт, мінімум 3 типи взаємодій.

• Таргетинг стабільний (sticky/hysteresis), немає «флікеру».

• Prompt system відображає реальні input bindings; підтримує tap/hold.

• UI/VFX/Audio під'єднані через події або диспетчери; логіка не залежить від конкретного HUD.

• FSM/стани взаємодії описані; є блокування/переривання для критичних моментів.

• Є тестова сцена, debug overlay, коротка документація та діаграма потоку.

Питання для самоперевірки

1. Поясніть модель «input -> state -> feedback» на прикладі взаємодії «відкрити двері».
2. Що таке affordance і як її підсилити без зайвого UI?
3. Які відповідальності має Interactor, а які - Interactable?
4. Назвіть мінімальний контракт Interactable API (4-5 методів/подій).
5. Які методи таргетингу використовуються найчастіше та в яких жанрах?
6. Навіщо потрібні правила пріоритету/score для вибору цілі?
7. Які вимоги до prompt system для підтримки ремапінгу та доступності?
8. Поясніть, що таке decoupling і як події допомагають інтегрувати UI/VFX/Audio.
9. Навіщо потрібна FSM для взаємодій? Наведіть 3 стани і 2 переходи.
10. Що таке data-driven взаємодії та які вигоди вони дають при масштабуванні?
11. Назвіть 3 інструменти відладки взаємодій у прототипі.
12. Які критерії «production-ready» для interaction framework у навчальному вертикальному зрізі?

Практичний кейс до теми (120-180 хв): «Interaction Framework + Prompt System»

Мета практичного кейсу - реалізувати універсальну систему взаємодій, яка підтримує мінімум 3 дії, працює стабільно в тестовій сцені та має подієву інтеграцію з UI/VFX/Audio.

Сценарії (оберіть один або реалізуйте всі три)

• Сценарій A: Door (tap) - «Відкрити/Закрити» + locked/unlocked (умова ключа або стану).

• Сценарій B: Pickup (tap) - «Підібрати» + інвентар/лічильник + короткий SFX.

• Сценарій C: Terminal (hold) - «Активувати» 0.4-0.8 с + прогрес-індикатор + VFX/звук.

Кроки виконання завдання

• Створіть Interactor на гравцеві: input, пошук цілі, виклик Interact().

• Створіть Interactable контракт (Unity interface / UE Blueprint Interface).

• Реалізуйте таргетинг (raycast або overlap) + sticky (hysteresis) для стабільності.

• Зробіть Prompt UI: текст + іконка, режим tap/hold, прогрес для hold.

Під’єднайте feedback через події: InteractionPerformed -> UI/VFX/Audio listeners.

• Додайте FSM взаємодії: Idle/Focusing/Interacting/Cooldown; обробіть cancel.

• Зробіть тестову сцену (6-10 об’єктів) та debug overlay.

• Оформіть README + діаграму потоку (рис. 4.1/4.3) та короткий постмортем.

Що необхідно здати

• Проєкт Unity або UE5 (сцена тесту + мінімум 3 interactables).

• Коротке відео 30-45 с з демонстрацією трьох взаємодій.

• README (1-2 стор.) з описом компонентів, tunables, списком подій.

• Діаграма потоку взаємодії (можна у diagrams.net).

Критерії оцінювання

• Стабільність: вибір цілі не «стрибає», підказки не мигають.

• Архітектура: є розподіл Interactor/Interactable/Prompt/Feedback, мінімум жорстких залежностей.

• UX: підказки читабельні, підтримано hold прогрес, тексти короткі.

• Розширюваність: легко додати 4-ту дію або новий тип об’єкта.

• Відладка/документація: debug overlay, лог подій, зрозумілий README та діаграма.

Рекомендовані програми та сервіси

1. Unity LTS: Input System, UI (UGUI/UI Toolkit), ScriptableObject Event Channels (опційно).
2. Unreal Engine 5: Enhanced Input, UMG, Blueprint Interfaces, Event Dispatchers, Data Tables/Assets.
3. DCC (за потреби): Blender/Krita; іконки/GUI: Kenney UI pack (CC0) або власні.
4. VCS: Git + Git LFS / Plastic SCM; для UE-команд - Perforce Helix Core.
5. Документація/діаграми: Notion або Google Docs; diagrams.net; Miro/Figma для flows.

Рекомендовані джерела та матеріали

Офіційна документація Unity та Unreal Engine 5 щодо Input System / Enhanced Input, UI систем, Blueprint Interfaces/Events, а також матеріали про патерни Observer/Command/State Machine. Для аналізу UX взаємодій рекомендовано розбір кейсів The Last of Us (контекстні підказки) та Zelda: Breath of the Wild (системні affordances).

Тема 3.4. Audio for Interaction: SFX layers, spatialization та parameter-driven audio

Мета теми: сформувати у студентів системне розуміння ролі звуку в Interaction design та game feel, навчити будувати покрокове аудіо (event-driven) у Unity або Unreal Engine 5 та застосовувати шарування (layering), просторовість (spatial audio), зони середовища (ambience/reverb) і параметризацію (RTPC/parameters) для підсилення читабельності дій, темпу та емоційного стану сцени. Результатом має стати керована аудіо-система інтеракцій, придатна для vertical slice.

Очікувані результати навчання

- вміти пояснювати, як звук впливає на game feel, темп і читабельність взаємодій;

- класифікувати SFX: one-shot, loop, tail, stinger, UI, ambience; проєктувати аудіо-фідбек у моделі anticipation -> impact -> recovery;
- будувати event-driven запуск аудіо від input/state/animation notify/collision;
- налаштовувати spatial audio (attenuation, priority, occlusion, voice limits);
- створювати ambience layers і reverb zones з коректними переходами;
- застосовувати parameters/RTPC (pitch/volume/filter) для «живого» реагування;
- організовувати базовий мікс: bus groups, headroom, баланс UI/SFX/Ambience;
- виконувати дебаг типових проблем: кліпінг, маскування, повтори, latency;
- оформлювати audio breakdown (події, параметри, правила, пресети).

Ключові терміни: SFX, ambience, stinger, loop, one-shot, tail, layering, transient, ADSR, dynamic range, headroom, clipping, bus, mix, ducking/sidechain, spatial audio, attenuation, occlusion, reverb, early reflections, RTPC/parameters, snapshot, audio event, sound cue, audio mixer/submix, latency, sample rate, bit depth, compression, limiter.

3.4.1 Позиція аудіо в Interaction design: «підтвердити», «пояснити» та «керувати увагою»

Звук у взаємодіях виконує наступні три функції

Підтвердження (confirmation) - дія «прийнята» і «відбулося».

Пояснення (meaning) - що саме відбулося (успіх/помилка/критичний стан).

Керування увагою (attention) - виділити важливе серед шуму подій (табл. 3.28; рис. 3.14).

Таблиця 3.28

Audio feedback stack (приклад навчального стандарту)

Подія	**SFX**	**UI звук**	**Space / Ambience**	**Ціль**
Натискання (input)	короткий click	confirm beep	-	«ввід прийнято»
Успішний hit	impact + tail	crit ping (опц.)	-	сила/результат
Взаємодія (pickup/door)	pickup/door	prompt confirm	-	успіх/контекст
Помилка (fail)	dull thud	error beep	-	«не можна»
Перехід зони	-	-	ambience + reverb	простір/настрій

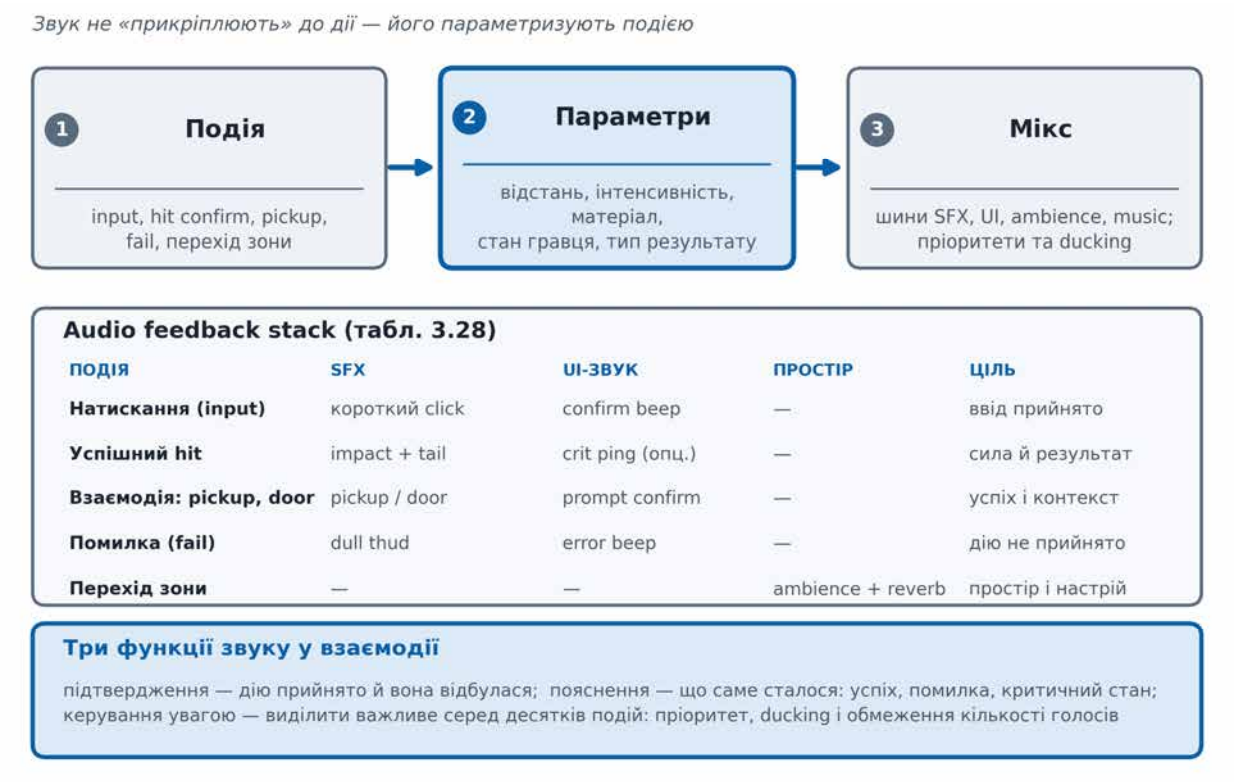


Рис. 3.14. Схема «подія -> параметри -> мікс»

Gameplay Event -> Audio Event (SFX/UI) -> Parameters (RTPC) -> Bus/Mixer -> Output

3.4.2 Аудіо-пайплайн: стандарти асетів, формати, рівні та неймінг

У навчальних прототипах головне – це дотримання дисципліни при створенні та збереженні матеріалів:

• імена (Action_Footstep_Stone_01, UI_Confirm_01, Amb_RoomTone_01);
• папки (/Audio/SFX/Actions, /Audio/UI, /Audio/Ambience);
• моно для 3D, стерео – для ambience/музики/інтерфейсу;
• headroom: не «душити» до кліпінгу (табл. 3.29).

Таблиця 3.29

Технічний мінімум (рекомендовано)

Категорія	Формат	Sample rate	Канал	Коментар
SFX (3D)	WAV/OGG	44.1/48k	Mono	коректна просторова локалізація
Ambience	OGG/WAV	48k	Stereo	луп без «шва», кросфейди
UI	WAV/OGG	44.1k	Stereo/Mono	коротко, м’яко, не дратує
VO	WAV/OGG	48k	Mono	стабільний рівень і читабельність

3.4.3 Layering: як збирати «сильний» SFX з 2-4 шарів

Більшість «приємних» звуків – це, як правило, композиція

Transient (атака): 5-30 ms - «клац»/контакт
Body (тіло): 50-200 ms – читає «матеріал»
Texture/Sweetener: деталь/характер
Tail (хвіст): простір/післязвуччя (рис. 3.15)

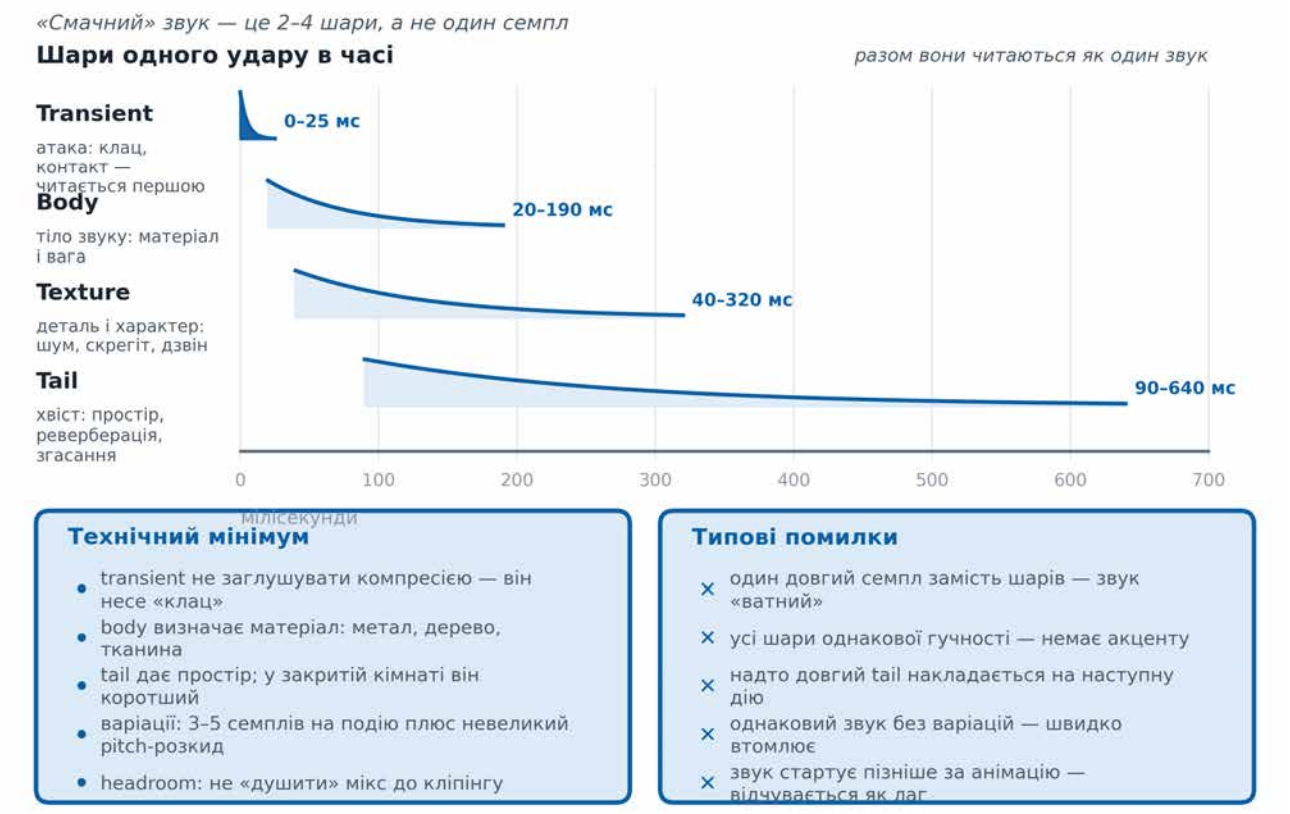


Рис. 3.15. Формула шарування

ImpactSFX = Transient + Body + (Sweetener) + Tail
Intensity -> volume/pitch blend of layers

Порада: робіть 2-3 варіанти кожного шару або додавайте random pitch ±2-6%.

3.4.4 Просторовий звук: attenuation, priority, voice limits, occlusion

Ключ до читабельності 3D аудіо – правильний радіус, крива затухання, пріоритет і обмеження голосів (табл. 3.30).

Таблиця 3.30

Мінімальний набір spatial параметрів

Параметр	Що робить	Значення прототипів	Типова помилка
Min/Max distance	межі затухання	1-3 м / 15-30 м	занадто великий Max -> «шум»
Rolloff curve	форма затухання	логарифмічна	лінійна звучить «плоско»
Priority	що важливіше	hit > steps > ambience	важливий звук «з'їдається»
Voice limit	скільки звуків одночасно	8-32 (за сценою)	без ліміту – хаос
Occlusion	стіна між джерелом і слухачем	low-pass + -3..-12 dB	різкі стрибки без згладжування

3.4.5 Ambience + Reverb zones: простір, настрій, переходи

Радимо не робити один «фон», завжди краще 2-3 шари

Room tone (тихий базовий).

Detail layer (вентиляція/птахи/механіка).

Occasional stingers (далекий удар/сирена).

Переходи між зонами мають бути плавні: crossfade 0.5-2.0 s (рис. 3.16).

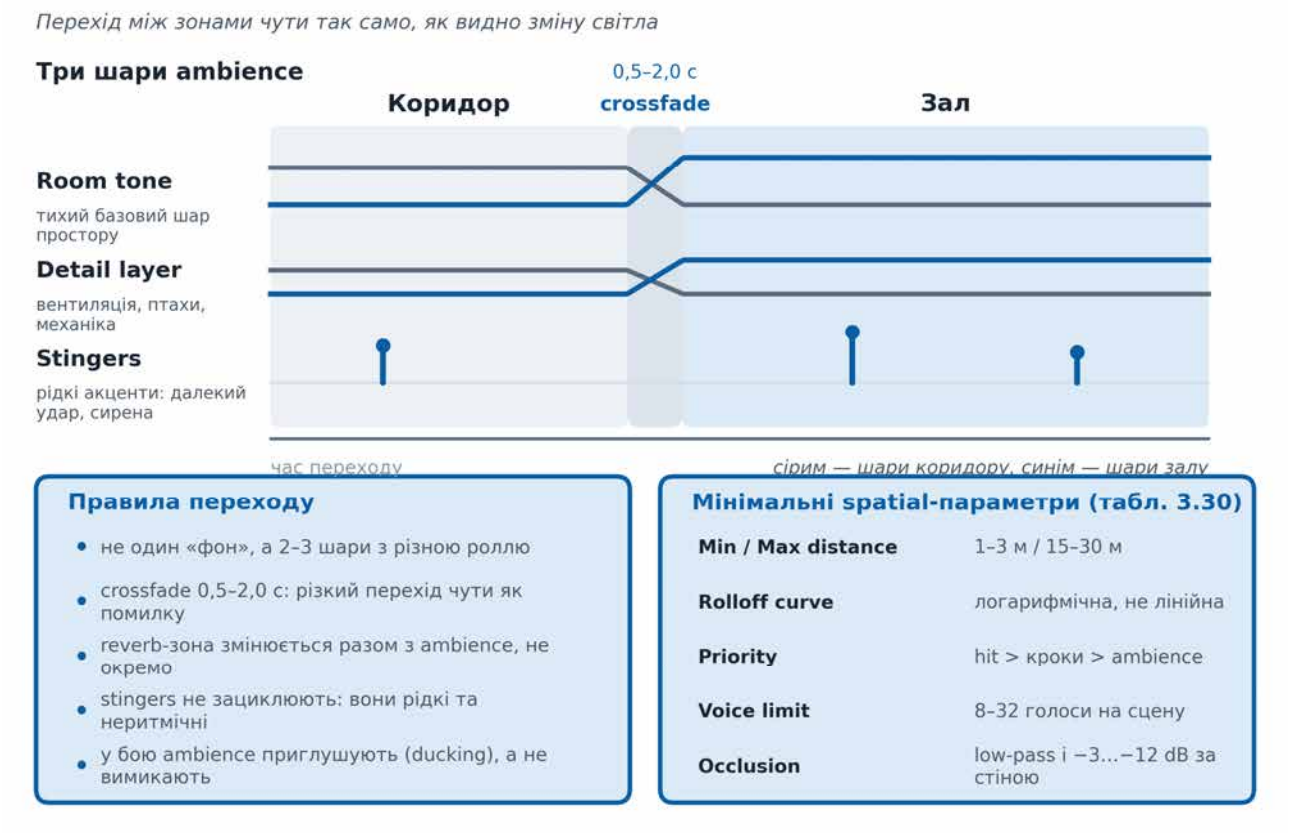


Рис. 3.16. Переходи ambience між зонами

Enter Zone A -> Snapshot A (amb+reverb)

Crossfade 0.5-2s

Enter Zone B -> Snapshot B

3.4.6 Parameter-driven audio (RTPC/parameters): звук, що «живе» разом з геймплеєм

Параметри дозволяють одному набору звуків реагувати на стан

Speed (0..1) -> частота кроків + легкий pitch

Charge (0..1) -> pitch + фільтр + додатковий електро-шар

Distance -> low-pass + volume

State enum (Normal/Combat/LowHP) -> snapshots міксу (рис. 3.17)

RTPC Charge

Pitch: 0..+300 cents

Volume: 0..+6 dB

Filter cutoff: 2k..12k

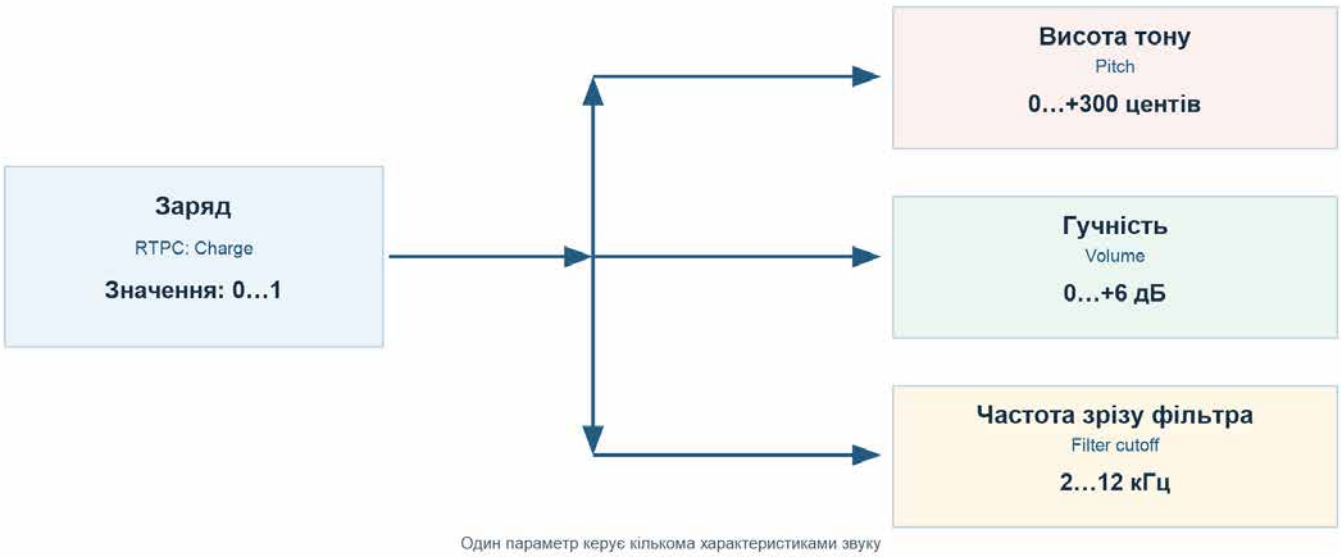


Рис. 3.17. Приклад RTPC мапінгу

3.4.7 Event-driven інтеграція: input/state/notifies/collision

Найнадійніше - це події

input accepted -> UI confirm
state entered (attack) -> anticipation
animation notify (footstep) -> step
hit confirmed -> impact
overlap success -> pickup (табл. 3.31)

Таблиця 3.31

Точки запуску аудіо

Подія	Точка	Що грає	Правило
Footstep	anim notify	step + surface	залежить від матеріалу
Hit	hit confirm	impact + tail	лише при підтвердженні
UI	button click	ui blip	тихо і коротко
Pickup	overlap success	sparkle	не дублювати

3.4.8 Базовий мікс: bus groups, headroom, маскування, ducking

Мінімальна структура може бути такою:
Master -> SFX / UI / Ambience / VO (табл. 3.32).

Таблиця 3.32

Швидкі правила міксу

Правило	**Навіщо**	**Перевірка**	**Порада**
Headroom	уникнути кліпінгу	peak meter	не «женіть» гучність
Пріоритет SFX	читабельність дій	тест у бою	hit/confirm завжди читається
UI не домінує	не дратує	серія кліків	коротко, м’яко, тихіше
Ducking ambience	«звільняє місце»	при важливих SFX	ambience -2..-6 dB
Variety	менше повторів	60с прогону	варіанти + random pitch

3.4.9 Типові помилки та дебаг

Проблеми, з якими можна зустрітися

• подвійний запуск (дві події на один звук);
• кліпінг/перегруз;
• маскування (ambience «з’їдає» hit);
• нескінченні loops без лімітів;
• «latency» через неправильну точку запуску.

Чек-лист дебагу

• лог подій: чи не запускається SFX двічі;
• voice limits + пріоритети;
mute ambience -> перевірити читабельність SFX;
• синхронізація з notify/impact frame;
• 2-6% random pitch і 2-4 варіанти асета.

3.4.10 Production-ready аудіо для vertical slice: чек-лист готовності

Чек-лист готовності включає:
- підтвердження успіх/помилка;
- удар/крок/взаємодія синхронні з анімацією/подією;
- мікс структурований, кліпінгу немає;
- spatial налаштований (mono, attenuation, priority);
- ambience має 2-3 шари + плавні переходи зон;
- є хоча б 1 RTPC/parameter;
- оформлено audio breakdown (події, параметри, правила, пресети).

Питання для самоперевірки

1. Які три основні функції звуку в інтеракції?
2. Чим відрізняються one-shot, loop, tail, stinger?
3. Чому для 3D SFX переважно потрібне mono?
4. Що таке attenuation і як обрати Max distance?
5. Навіщо потрібні voice limits і priority?
6. Що таке ambience layering і чому 2-3 шари краще за 1?

7. Як працюють reverb zones і чому важливий crossfade?
8. Що таке RTPC/parameter-driven audio? Наведіть 2 приклади.
9. Які переваги event-driven підходу?
10. Що таке headroom і чому кліпінг критично поганий?
11. Поясніть маскування у міксі й способи зменшення.
12. Які критерії «production-ready» аудіо для vertical slice?

Практичний кейс (час виконання: 120 хв)

«Interaction Audio Pack» (Event-driven + Spatial + RTPC)

Мета: реалізувати аудіо-пак для однієї ключової взаємодії/ability з трьома фазами:
anticipation -> impact -> recovery, плюс spatial, зона середовища, 1 параметр (RTPC).

Сценарії (A/B/C)

A (Melee): swing -> hit confirm -> tail + fail miss

B (Projectile): launch -> fly loop -> impact + tail + attenuation

C (Interact/UI): door/pickup start -> success -> close + UI confirm/fail

Покрокове виконання завдання

• Підготуйте 6-12 звуків (CC0/own/educational).

• Стандартизуйте неймінг і папки (/SFX/UI/Ambience). Зробіть шини: Master -> SFX/UI/Ambience(/VO).

• Реалізуйте event-driven: input/state/notify/collision.

• Налаштуйте spatial: mono, attenuation, priority, voice limit.

• Додайте reverb/ambience zone з плавним переходом (crossfade/snapshot).

Додайте 1 parameter/RTPC (Charge/Speed/State) -> pitch/volume/filter.

• Перевірте мікс: без кліпінгу; важливі події читаються.

• Додайте варіативність (2-4 варіанти або random pitch ±2-6%).

• Зробіть audio breakdown (1-2 стор.): події, параметри, правила, пресети.

Що необхідно здати

• Проєкт Unity/UE5 з інтегрованим паком.

• Відео 30-60 с з демонстрацією (звук обов’язково).

• Audio breakdown (1-2 стор.).

• Постмортем 150-250 слів.

Критерії оцінювання

• читабельність (impact/confirm/fail зрозумілі);

• синхронізація (impact frame/footstep);

• системність (event-driven, структура міксу, неймінг);

• spatial (адекватний радіус, пріоритети, без «шуму»);

• parameter реально відчутний;

• якість (без кліпінгу, без надмірно гучного UI, є variety).

Рекомендовані програми та сервіси

1. Unity LTS: AudioSource, Audio Mixer, snapshots, reverb zones; (опц.) FMOD.
2. Unreal Engine 5: Sound Cues, attenuation, submixes; (опц.) MetaSounds, Wwise/FMOD.
3. Редактори: Audacity, Reaper, Ocenaudio.
4. Планування: Notion/Trello; VCS: Git/Git LFS (або Perforce для UE команд).
5. Бібліотеки (перевіряти ліцензії): freesound, навчальні/CC0 набори, власні записи.

Рекомендовані джерела та матеріали

1. Unity Manual: Audio Mixer, Spatial Audio, Snapshots, Reverb Zones.
2. Unreal Engine Documentation: Sound Cue, Attenuation, Submixes (MetaSounds - опц.).
3. FMOD / Wwise: Events, RTPC/Parameters, Mixing, Snapshots.
4. GDC Talks про interactive audio та game mix (відбір під ваш курс).
5. Кейси: DOOM (2016) (combat mix), Forza (telemetry/parameter-driven audio).

Тема 3.5. Animation Systems & State: blend trees, root motion, notifies та interrupts

Мета: сформувати у студентів практичне та системне розуміння анімації, як частини interaction/game feel, навчити будувати керовані анімаційні стани (locomotion, interaction, combat), інтегрувати Blend Tree / Blend Space. Вміти використовувати root motion або in-place, працювати з layers/masks, налаштовувати notifies/events/curves для синхронізації (хіт-вікна, кроки, VFX/SFX), а також коректно реалізувати interrupt/priority (hit реакції, stagger, cancel windows). Результатом вивчення теми є production-ready прототип анімаційної системи, що масштабується під vertical slice.

Очікувані результати навчання

- вміти пояснювати роль анімації у читабельності, темпі та «відчутті» дії (anticipation/impact/recovery, commitment);
- будувати locomotion state machine з параметрами руху та повороту;
- застосовувати Blend Tree (Unity) або Blend Space/Anim Graph (UE);
- обирати між root motion та in-place (з урахуванням камер, колізій, мережі);
- організовувати layers/masks (верх/низ тіла, aim, additive);
- використовувати Animation Events / Notifies / Curves для синхронізації gameplay;
- реалізувати hit reactions, stagger/knockback, i-frames (за потреби), та interrupt policy;
- дебажити анімаційні переходи, уникати «pop», sliding, loop jitter;
- оптимізувати анімацію та контролювати бюджет (profiling, culling, compression);
- оформлювати animation breakdown: стани, параметри, правила переходів, пріоритети.

Ключові терміни: State machine, locomotion, idle/walk/run, Blend Tree, Blend Space, Anim Graph, Anim Blueprint, Montage, Layer, Mask, Additive, Aim Offset, Root Motion, In-place, Retargeting, IK, Foot locking, Notifies/Events, Curves, Sync markers, Transition, Interrupt, Priority, Cancel window, Commitment frames, Hit reaction, Stagger, Knockback, i-frames, Animation compression, Culling.

3.5.1 Анімація як «мова» інтеракції: читабельність, намір та commitment

Анімація - це не «косметика», а канал комунікації. Вона

- пояснює намір (anticipation: замах/підготовка),
- фіксує результат (impact: контакт/хіт),
- задає темп (recovery: повернення у контроль),
- визначає commitment frames (наскільки «серйозно» ви зобов'язали гравця до дії).

Приклади/кейси

Monster Hunter: довгі recovery + чіткі commitment frames -> «важка» зброя відчувається реальною.

Spider-Man traversal: швидкі transitions + інерція/IK -> відчуття плавності та контролю (табл. 3.33).

Таблиця 3.33

«Animation feedback stack» для події

Подія	Анімація	Notifies/Events	VFX/SFX	Ціль
Старт руху	Start/Accel	step notify	шаг/пил	контроль, темп
Стрибок	Jump start -> airborne -> land	jump/land notify	whoosh/land	читабельність
Удар	Wind-up -> active -> recovery	hit-window notify	impact + spark	сила/результат
Взаємодія	interact pose	interact notify	UI confirm	«успіх»

3.5.2 Пайплайн анімації: імпорт, ретаргет, компресія, якість

Навіть найкраща логіка зламається, якщо асети «брудні». Мінімальний навчальний стандарт включає

- неймінг: Locomotion_Run_Fwd, Combat_Attack_Light_01, HitReact_Front, Interact_Pickup;
- однакова частота (часто 30/60 fps - залежно від контенту; головне - стабільність);
- ретаргетинг: Unity Humanoid / UE5 IK Retargeter;
- компресія: без «дір», без тремтіння, з адекватним розміром (табл. 3.34).

Таблиця 3.34

Чеклист імпорту (Unity / UE5)

Пункт	Unity	UE5	Типова помилка
Skeleton/Rig	Humanoid/Generic	Skeleton asset	різні скелети без ретаргету
Root/hip	Avatar root	Root bone	sliding через некоректний root
Compression	import settings	compression settings	тремтіння кісток/поза «пливе»
Loop	loop time + match	looping + sync markers	«ривок» на шві лупа
Scale	1.0	import scale	різні масштаби -> IK/колізії ламаються

3.5.3 Locomotion state machine: параметри, transitions, start/stop

Базова locomotion-система в прототипі має бути
зрозумілою (мінімум станів, максимум читабельності),
керованою параметрами (speed, direction, grounded),
без «teleport transitions» (плавні blend-и, правильні thresholds) (рис. 3.18).

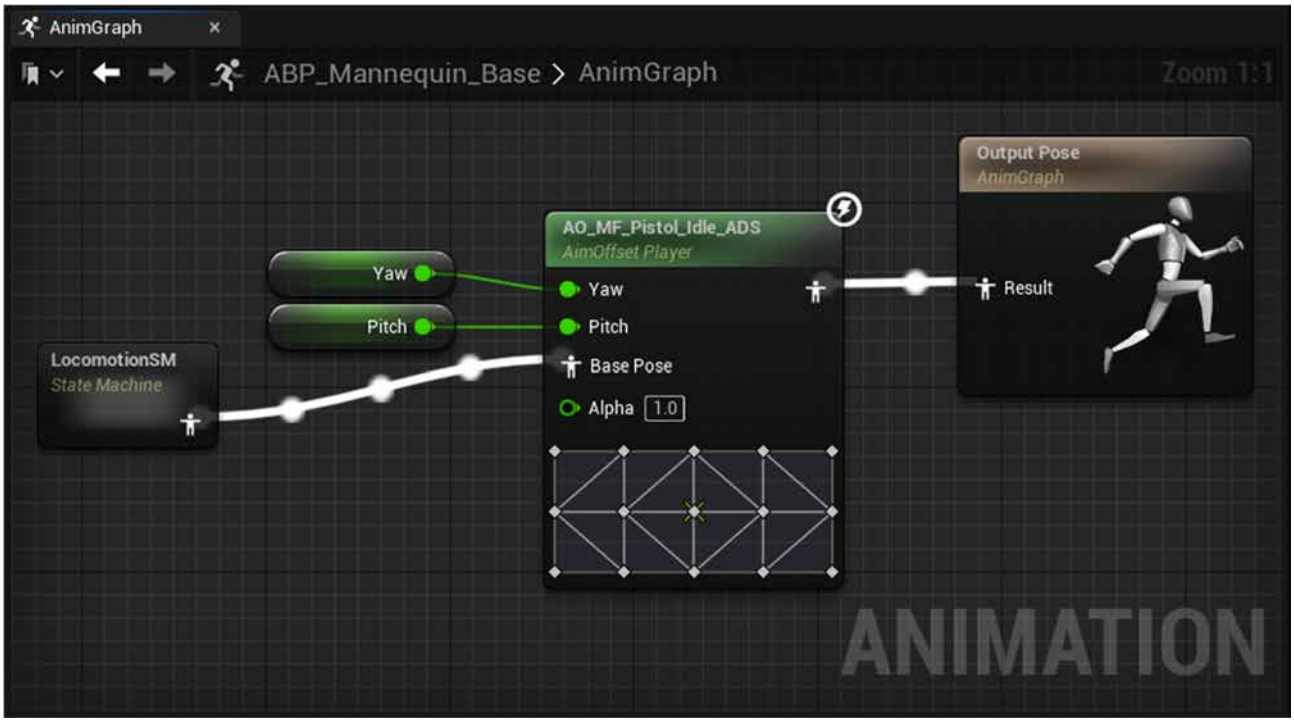


Рис. 3.18. Схема locomotion (спрощено)

```
      Idle <-> Move (Blend Tree / Blend Space)
      |
+-> Jump (Start -> Air -> Land)
      |
      +-> Interact (overlay / montage)
      |
      +-> HitReact (interrupt)
```

3.5.4 Root Motion vs In-Place: як обрати правильно

Root Motion: рух бере з анімації. In-place: рух робить контролер/фізика, анімація «підкреслює» (табл. 3.35).

Таблиця 3.35

Порівняння Root Motion та In-Place

Критерій	Root Motion	In-place
Відчуття «ваги»	дуже добре (природно)	залежить від контролера
Контроль геймплею	складніше (колізії/повороти)	простіше керувати
Мережа/реплікація	складніше	часто простіше
Системність	чудово для melee/монтажів	чудово для locomotion/FPS
Ризик sliding	менший (якщо добре зроблено)	більший (якщо поганий sync speed)

Правило для курсу (прототип)

locomotion -> in-place (швидше налаштувати, легше дебажити);
атаки/фінішери/кінематичні дії -> root motion або montage (де треба вага/точність).

3.5.5 Blend Trees / Blend Spaces + Layers/Masks + Additive

Щоб персонаж не виглядав «роботом», потрібно

Blend за speed/direction (walk/run/strafe),
Layering (верх тіла стріляє/цілиться, низ біжить),
Additive (легкі коливання, recoil, breathing),
Aim Offset / Look-at (особливо для third-person) (рис. 3.19).

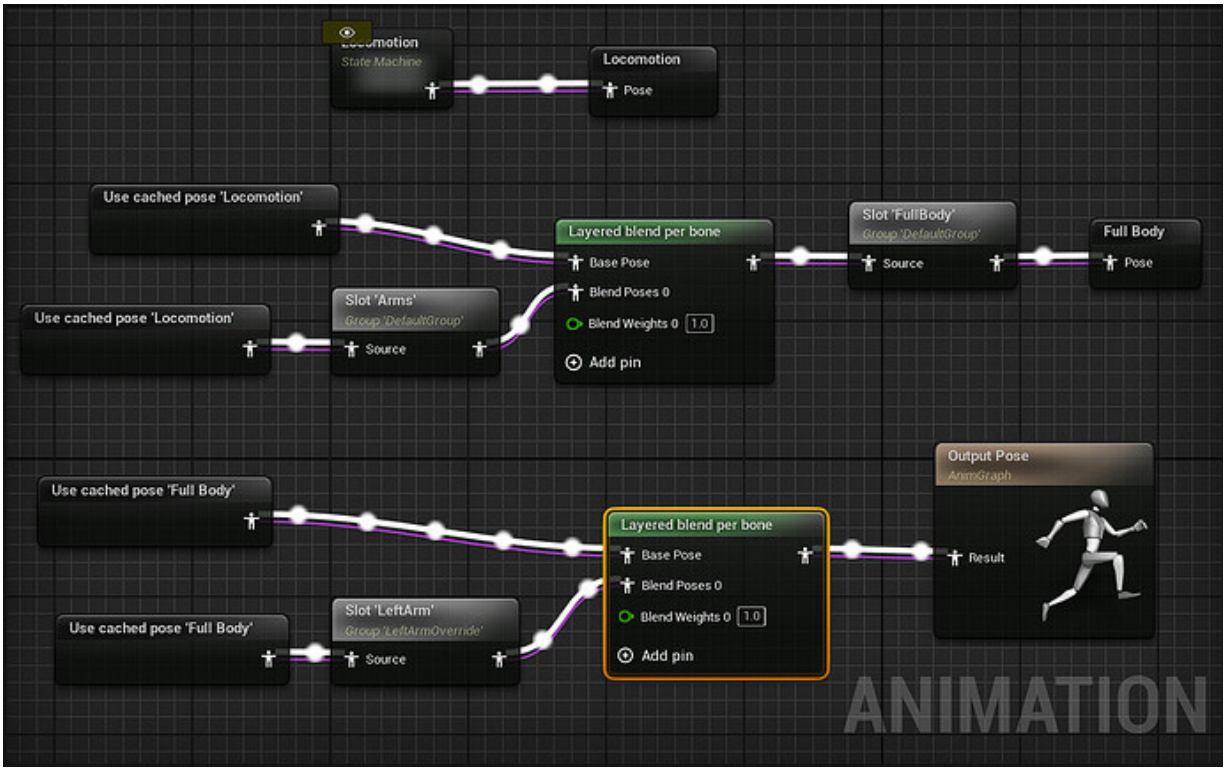


Рис. 3.19. Логіка шарів

Base Layer: Locomotion (Idle/Walk/Run)
Upper Body Layer (masked): Aim / Weapon / Interaction
Additive Layer: Breath/Recoil

3.5.6 Notifies / Events / Curves: синхронізація gameplay з анімацією

Події в анімації - «скріпка» між виглядом і логікою

Footstep notify -> SFX/VFX + surface type
Hit window -> активувати хітбокс/трейс
Weapon trail -> VFX on/off
Camera impulse -> shake
Invuln window -> i-frames (за потреби) (табл. 3.36)

Таблиця 3.36

Типові notifies (навчальна база)

Notify	Коли	Навіщо	Помилка
Footstep	контакт ноги	SFX/VFX	кроки «плаваючі», не в такт
HitStart/HitEnd	active frames	хітбокс	вікно завелике -> «читерство»
TrailOn/TrailOff	замах/удар	читабельність	Trailпостійно увімкнений
Land	приземлення	impact	відсутній landing -> «порожньо»
InterruptAllowed	cancel window	контроль	гравця «замикає» в анімації

3.5.7 Combat states: wind-up / active / recovery, commitment, cancel windows

Будь-яку бойову дію варто мислити як таймлайн

Wind-up (anticipation): читається намір, готуємо удар;
Active (impact window): момент результату;
Recovery: плата за дію, темп, ризик/нагорода (рис. 3.20).

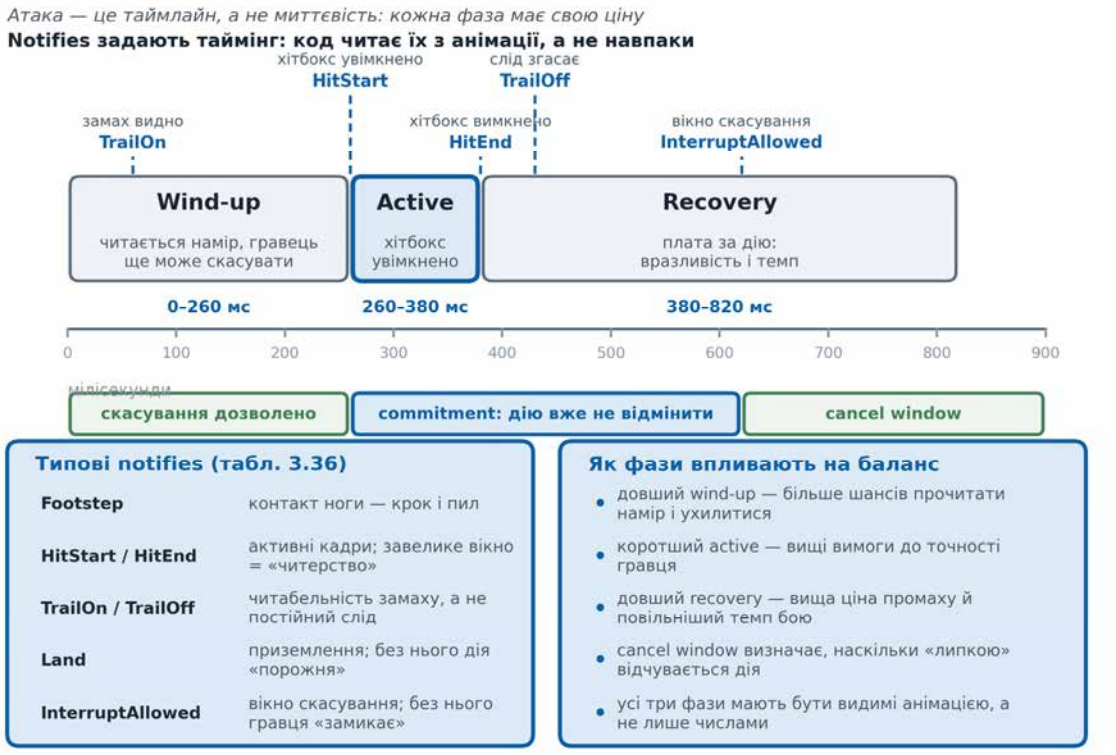


Рис. 3.20. Таймлайн атаки

[Wind-up] [Active / Hit window] [Recovery]
^ ^ ^
читабельність результат commitment
Monster Hunter – еталон «чесного» recovery.
Spider-Man – еталон «плавних» transitions і cancel-логіки для traversal/combat.

3.5.8 Hit reactions, interrupts, priority: як не зламати контроль

Hit reaction не має перетворювати гру на «stunlock simulator», проте має бути чесною. Потрібно дотримуватися таких правил:

• пріоритети (що перебиває що),
• cooldowns на реакції,
• клас реакції: light stagger / heavy stagger / knockdown,
• умови ігнорування (наприклад, i-frames або armor state) (табл. 3.37).

Таблиця 3.37

Матриця переривань (приклад)

Поточний стан	Light hit	Heavy hit	Interact	Dodge
Locomotion	interrupt	interrupt	overlay	дозволено
Attack wind-up	часто дозволено	interrupt	ні	залежить від дизайну
Attack active	рідко	interrupt	ні	за cancel window
Attack recovery	інколи	interrupt	ні	інколи
Interact	інколи	interrupt	-	інколи

3.5.9 Дебаг і оптимізація: profiling, culling, компресія, budget

Анімація може «вбити» FPS через

- надто складний граф,
- забагато шарів/IK,
- відсутність culling,
- погану компресію/надмірний розмір кліпів.

Підхід «measure -> fix -> verify» застосовуємо і тут (рис. 3.21).

Практичні поради

Unity: Animator Culling Mode, оптимізація layers, уникати зайвих параметрів/переходів.

UE5: Animation Budget (за можливості), оптимізація AnimBP, LOD для SkeletalMesh, відключення дорогих нодів на дистанції.



Рис. 3.21. Цикл оптимізації анімації

Profile -> Find hot spot (Animator/AnimBP/IK) -> Simplify -> Verify

3.5.10 Production-ready animation system: чек-лист готовності

Є стабільний locomotion (idle/walk/run + start/stop);
Є jump/land з читабельним timing;
Є interaction overlay/montage (наприклад, pickup/door);
Є combat action з wind-up/active/recovery;
Hit reactions працюють по правилах priority (без хаосу);
Notifies синхронізують SFX/VFX/hit windows;
Gараметри задокументовані (speed, direction, grounded, state);
Transitions без «pop», без sliding;
Граф не розростається: модульність і правила.

Питання для самоперевірки

1. Чому анімація є частиною interaction/game feel, а не «декором»?
2. Опишіть базову схему locomotion state machine.
3. Для чого потрібні Blend Tree / Blend Space?
4. Root motion vs in-place: ключові плюси/мінуси.
5. Що таке layers/masks і де вони критично потрібні?
6. Що таке additive animation і навіщо вона в прототипі?
7. Які типові notifies потрібні для кроків та ударів?
8. Що таке hit window і чому його не можна робити завеликим?

9. Поясніть commitment frames і cancel windows на прикладі melee.
10. Як уникати «stunlock» і водночас зберегти чесний hit reaction?
11. Назвіть 3 джерела проблем sliding/popping у переходах.
12. Які 5 пунктів «production-ready» чек-листа для vertical slice?

Практичний кейс (час виконання: 120 хв)

«Animation-Driven Controller + State & Interrupts»

Мета: зібрати керовану анімаційну систему персонажа (Unity або UE5), де locomotion + interaction + hit reaction працюють як узгоджений state machine з notifies.

Сценарії (A / B / C)

• A (Traversal): locomotion + jump/land + повороти + footstep notifies.

• B (Combat): 1 легка атака (wind-up/active/recovery) + hit window + cancel window.

• C (Interaction): interact (pickup/door) як overlay/montage + UI/SFX підтвердження.

Рекомендовано: виконати A + один із (B або C). За бонус – усі три.

Кроки виконання завдання

• Підготуйте 6-10 анімацій (можна Mixamo/інше навчальне джерело): idle, walk/run, jump start/loop/land, interact, hit react, attack.

• Налаштуйте імпорт/ретаргет (Unity Humanoid або UE Skeleton/IK Retargeter).

• Побудуйте locomotion: Blend Tree/Blend Space на speed (та direction за бажання).

Додайте jump state (Start -> Air -> Land) з коректними transitions.

• Додайте interaction overlay/montage (інтеракція не ламає locomotion повністю).

• Додайте combat action (wind-up/active/recovery) або hit reaction.

• Додайте notifies/events: footstep, land, hit start/end.

• Реалізуйте interrupt policy: hit reaction перериває locomotion/interaction за правилами.

• Додайте debug overlay: поточний state, speed, grounded, canInterrupt.

• Перевірте «читабельність»: таймінг, sliding, pop у переходах.

Що необхідно здати

• Проєкт Unity/UE5 з реалізованим контролером і анімаційною системою.

• Відео 45-90 сек: locomotion + jump + (interaction або attack) + hit reaction.

• Animation breakdown (1-2 стор.): стани, параметри, правила interrupt/priority, список notifies.

• Короткий постмортем 150-250 слів: що було складно, що поліпшили.

Критерії оцінювання

• Читабельність: стани легко «прочитати» без UI.

• Синхронізація: notifies працюють у правильні моменти.

• Керованість: параметри та transitions зрозумілі, без хаосу.

• Interrupt policy: реакції не ламають контроль і не «спамляться».

• Якість переходів: мінімум sliding/popping, стабільні blends.
• Документація: breakdown структурований і практичний.

Рекомендовані програми та сервіси

1. Unity LTS: Animator (Mecanim), Blend Trees, Animation Events, (опц.) Animation Rigging, Timeline.
2. Unreal Engine 5: Anim Blueprint, State Machine, Blend Space, Montages, Notifies, (опц.) Control Rig, IK Retargeter.
3. DCC/асети: Blender; Mixamo (для навчальних прототипів); базові mocap набори (перевіряти ліцензії).
4. Відладка/профілювання: Unity Profiler; Unreal Insights/Stat commands.
5. VCS/організація: Git/Git LFS; (опц.) Perforce для UE; Notion/Trello для чек-листів і breakdown.

Рекомендовані джерела та матеріали

1. Unity Manual: Animator, Blend Trees, Animation Events, Humanoid Rig/Avatar.
2. Unreal Engine Documentation: Animation Blueprints, Blend Spaces, Montages, Notifies, Retargeting.
3. GDC доповіді про combat animation, timing, readability, hit reactions.
4. Книги/підходи: принципи анімації (anticipation/impact/recovery), game feel як «відчуття контролю».

Тема 3.6. Onboarding, difficulty та UX telemetry

Мета: сформувати у студентів системне бачення того, як проєктувати онбординг (перші 5-15 хвилин гри), керувати складністю (difficulty/assist/options), будувати мікро-туторіали та підказки, а також збирати UX-телеметрію для валідації рішень. Навчити застосовувати принципи progressive disclosure, «teach through play», і створювати мінімальний, але production-сумісний pipeline telemetry -> аналіз -> ітерації для vertical slice.

Очікувані результати навчання

- вміти пояснювати цілі онбордингу та відмінність tutorial vs onboarding; проєктувати мікро-туторіал з 1-2 механіками («вивчив -> застосував -> закріпив»);
- застосовувати progressive disclosure та «one thing at a time»;
- визначати та керувати difficulty knobs (шкода, таймінги, assist, AI, економіка);
- реалізовувати динамічну складність (обережно) або assist-системи;
- проєктувати UX-логування: події, параметри, контекст, session;
- будувати простий дашборд/звіт (таблиця/CSV) та формувати висновки;
- уникати типових помилок (over-tutorialization, spam hints, «ложна» телеметрія);
- оформлювати onboarding + telemetry design doc (1-2 стор.) для команди.

Ключові терміни: Onboarding, tutorialization, first-time user experience (FTUE), progressive disclosure, teach through play, affordance, signposting, friction, hint system, dynamic hints, input reminder, difficulty curve, assist mode, difficulty knobs, pacing, fail-forward, retries, funnel, retention (session-level), telemetry, event logging, instrumentation, session id, timestamp, cohort, heatmap (концепт), A/B test (концепт), QA analytics.

3.6.1 Onboarding vs Tutorial: що саме ми «вчимо» у перші хвилини

Onboarding – ширше за поняття «tutorial». Він включає

• правила світу, цілі, ризики/нагороди;

• керування і базові дії;

• очікування щодо темпу, стилю, складності;

«петлю» гри (loop): зробив -> отримав фідбек -> зрозумів -> повторив.

Tutorial – це конкретний навчальний сегмент/екран/послідовність (табл. 3.38).

Таблиця 3.38

Цілі онбордингу (мінімум)

Ціль	Що має статися	Як перевірити
Розуміння цілі	«Що робити?»	гравець йде у правильному напрямку
Опанування 1-2 дій	«Як?»	виконує дію без підказки вдруге
Відчуття контролю	«Я керую»	менше випадкових помилок
Мотивація	«Хочу далі»	не виходить з гри після 3 хв

3.6.2 «Teach through play»: навчання через дію, а не через інструкцію

Працює формула

- Показати (середовище підштовхує).
- Змусити спробувати (безпечний контекст).
- Закріпити (повтор в іншій ситуації).

Кейси: Portal (покроково вводить механіки), Monument Valley (інтуїтивні підказки через рівень) (рис. 3.22).

3.6.3 Progressive disclosure: «одна нова річ за раз»

Дуже важливо не перевантажувати, а діяти наступним чином

1 нова дія/правило за короткий відрізок;

між новими речами - «вікно гри» (пограти, відчути);

UI підказки – короткі і контекстні (табл. 3.39).



Рис. 3.22. Мікро-цикл навчання

Prompt -> Safe Try -> Feedback -> Repeat -> Variation -> Mastery

Таблиця 3.39

Дозування інформації

Етап	Дати гравцю	Не давати
0-60 сек	1 ціль + 1 дія	5 кнопок + лор-стіну
1-5 хв	1-2 механіки+1 ризик	«меню налаштувань на 30 пунктів»
5-15 хв	варіації+перший «іспит»	постійні туторіал-попапи

3.6.4 Hint systems: правила хороших підказок або «як не дратувати»

Підказка повинна бути

- контекстна (саме тоді, коли треба),
- коротка (1 рядок або 1 піктограма),
- відключувана/затихаюча (не спамить),
- перевіряє, чи гравець уже вміє (табл. 3.40).

Таблиця 3.40

Типи підказок

Тип	Приклад	Коли
Static hint	«Press E to interact»	перший раз біля об'єкта
Dynamic hint	«Try jumping earlier»	3 фейли поспіль
Soft signposting	світло/стрілка/контраст	навігація без тексту
Reminder	показ кнопки на HUD	якщо не натискав 30-60 сек

3.6.5 Input reminders і навчання керування: «пам’ять м’язів»

Гравець запам’ятовує, що відбувається в грі через повтор

спершу – підказка + дія,
потім – дія без підказки,
потім – дія під тиском (таймер/ворог) (рис. 3.23).



Рис. 3.23. «Від підказки до автоматизму»

Hint ON -> Player succeeds -> Hint fades -> Repeat -> No hint needed

3.6.6 Difficulty design: крива складності та «difficulty knobs»

Складність – це керовані ручки (knobs), а не абстракція

шкода/HP/regen,
таймінги (парирування, вікна),
AI агресія/частота атак,
економіка ресурсів,
кількість ворогів/спавн,
швидкість проєктайл/камера,
«assist» опції (aim assist, slowdown) (табл. 3.41).

Таблиця 3.41

Приклади difficulty knobs

Knob	Змінює	Ризик
Enemy damage	виживання	«нудний sponge»
Enemy count	інтенсивність	хаос/перевантаження
Timing window	майстерність	несправедливість
Checkpoint freq	фрустрація	втрата напруги

3.6.7 Assist & Accessibility як частина difficulty (не «чит-коди»)

Assist – це не соромно, це – професійно

Це дає наступні можливості: розширення аудиторії; контроль фрустрації т «підтримка» гравця без руйнування ядра.

Приклади

• aim assist, auto-aim mild,
• slowdown при складних секціях,
• підвищення контрасту об’єктів,
• підказки, що можна вимкнути (табл. 3.42).

Таблиця 3.42

Assist design rules

Правило	Навіщо
Опція, а не примус	повага до гравця
Пояснення ефекту	прозорість («що зміниться»)
Збереження core loop	не ламати гру повністю

3.6.8 UX Telemetry: що логувати і чому (мінімальний набір)

Телеметрія потрібна не «для галочки», а щоб відповісти на питання

Де гравці «застрягають»?
Які підказки спрацьовують?
Скільки часу до першого успіху?
Де виходять з гри? (табл. 3.43)

Таблиця 3.43

Мінімальні події для FTUE/онбордингу

Event	Параметри	Навіщо
session_start	build, level, time	контекст
tutorial_step_shown	step_id	показ кроку
tutorial_step_completed	step_id, time_to_complete	успішність
hint_shown	hint_id, reason	спам/користь
fail	type, location	де застряг
success	action_id	чи навчився
options_changed	key, value	користь ассистів
session_end	duration, progress	відтік

3.6.9 Аналіз telemetry: простий «funnel» та інтерпретація

Навіть без складних систем можна робити аналіз

CSV лог -> таблиця -> граф/підрахунок,

«funnel»: Step1 shown -> Step1 completed -> Step2 …,

• time-to-first-success,

• fail count before success (рис. 3.24).

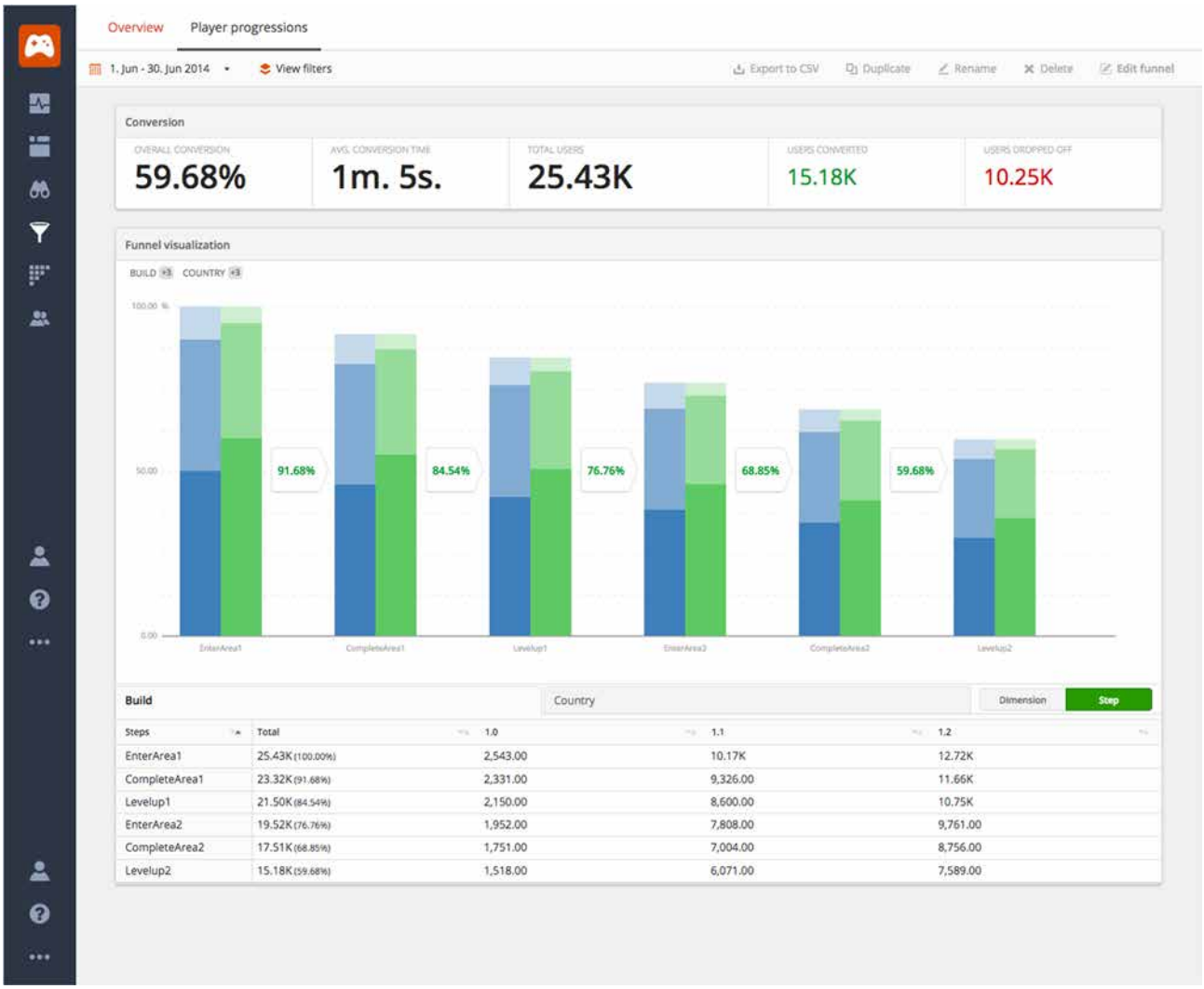


Рис. 3.24. Funnel приклад

Step 1 shown -> Step 1 completed -> Step 2 shown -> Step 2 completed -> Exit (табл. 3.44)

Таблиця 3.44

Пастки інтерпретації

Пастка	**Чому погано**	**Як уникнути**
Логувати все	шум	логувати лише питання, не все
Висновок з 5 людей	слабко	мін. 10-15 тестів
«Застряг» ≠ «погано»	може бути challenge	дивитись контекст+ фідбек

3.6.10 Production-ready FTUE: чек-лист готовності + документація

Онбординг навчає 1-2 ключові механіки через гру;
є progressive disclosure (без перевантаження);
hints не спамлять, є правила показу/затухання;
є knobs/assist (мінімум 2) та пояснення;
телеметрія збирається (мін. 8 подій) і зберігається у CSV/лог;
виконано 1 цикл: telemetry -> висновки -> ітерація;
оформлено FTUE + telemetry design doc (1-2 стор.).

Питання для самоперевірки

1. Чим onboarding відрізняється від tutorial?
2. Опишіть «teach through play» у 3 кроки.
3. Що таке progressive disclosure і чому воно критичне?
4. Назвіть 4 ознаки «хорошої підказки».
5. Як зробити hints, що не дратують?
6. Які етапи формування «пам’яті м’язів» для input?
7. Що таке difficulty knobs? Наведіть 5 прикладів.
8. Чому assist - це частина дизайну, а не «чит»?
9. Які події телеметрії мінімально потрібні для FTUE?
10. Що таке funnel і як його застосувати в онбордингу?
11. Назвіть 3 типові помилки збору/інтерпретації telemetry.
12. Який чек-лист production-ready FTUE?

Практичний кейс (час виконання: 120-180 хв)

«Micro-FTUE + Dynamic Hint + Telemetry»

Мета: створити мікро-онбординг на 3-5 хв з 1 механікою + 1 варіацією, додати динамічну підказку і мінімальну телеметрію, а потім зробити 1 ітерацію за результатами тесту.

Сценарії (A/B/C)

• A (Movement mastery): jump + coyote/buffer, потім застосування під тиском (таймер).

• B (Interaction): interact + prompt + «не можна» (fail state), потім комбо 2 дій.

• C (Combat feel): базовий удар + hit confirm, потім ворог/мішень з варіацією.

Кроки виконання завдання

• Визначте 1 навчальну ціль: що гравець має вміти через 3 хв.

Побудуйте FTUE flow з 3 кроків: show -> try -> apply.

• Додайте 1 dynamic hint (умова: 3 фейли або 20-30 сек без прогресу).

• Додайте input reminder (показ кнопки/іконки, що зникає після успіху).

• Налаштуйте 2 difficulty knobs або assist (наприклад: більший таймінг, lower damage, auto-aim mild).

• Інструментуйте telemetry (мінімум 8 events з табл. 11.6).

• Проведіть 2-3 міні-тести (одногрупники) або self-test з ролями «гравець/спостерігач».

• Зберіть лог (CSV/текст), порахуйте: time-to-complete, fail count, hints shown.

• Зробіть 1 ітерацію: змініть 1 елемент (hint timing, knob, підказка) і повторіть короткий тест.

Що необхідно здати

• Проєкт Unity/UE5: FTUE сцена + hints + knobs/assist + telemetry logging.

• Відео 60-90 сек: показ FTUE і приклад спрацювання hint.

• FTUE + Telemetry Doc (1-2 стор.):

• flow (3-6 кроків),

• правила підказок (умови),

• knobs/assist (що змінюють),

• список events + параметри,

• короткі висновки 1 ітерації.

• CSV/лог файл (або вставлений у документ таблицею).

Критерії оцінювання завдання

• ясність онбордингу (ціль зрозуміла без текстових «лекцій»);

• progressive disclosure (немає перевантаження);

• hints корисні, не спамлять;

• knobs/assist пояснені і логічні;

• telemetry валідна (події мають сенс, є контекст);

• проведено 1 ітерацію і описано зміни.

Рекомендовані програми та сервіси

1. Unity: ScriptableObjects (для FTUE steps), UI Toolkit/UGUI, PlayerPrefs/JSON для логів, Analytics (опц.).
2. UE5: Data Assets/Structs, UMG, SaveGame/JSON, Gameplay Tags (для станів), (опц.) Analytics plugin.
3. Тестування/аналіз: Google Sheets/Excel (CSV), Notion для FTUE doc, OBS для запису плейтесту.

Рекомендовані джерела та матеріали

1. Принципи tutorialization: Portal (teach-through-play) як референс-логіка.
2. UX/FTUE підходи з доповідей GDC (onboarding, hints, difficulty).
3. Accessibility guidance: Game Accessibility Guidelines (для assist/опцій).
4. Практики telemetry/instrumentation (мінімальні події, funnel).

Тема 3.7. Імерсивний дизайн, XR та тактильний зворотний зв’язок

Мета: Сформувати базове розуміння проєктування VR/AR/XR-досвіду, де простір, рух тіла, комфорт, spatial UI, просторовий звук і haptics є частиною ігрової механіки, а не додатковим декоративним шаром.

3.7.1 Імерсивний застосунок як просторово-тілесна система

У традиційній грі користувач спостерігає віртуальний світ через екран і взаємодіє з ним опосередковано. У VR голова, руки та положення тіла стають частиною input system, а масштаб, відстань і швидкість сприймаються фізично. В AR цифровий контент повинен узгоджуватися з реальним середовищем, освітленням, поверхнями, оклюзією та увагою користувача.

Presence виникає тоді, коли сенсорні сигнали не суперечать один одному: рух голови миттєво змінює зображення, звук локалізується у просторі, об’єкт реагує на дотик, а масштаб відповідає очікуванню. Навіть невелика затримка, нестабільний frame time, неправильна висота камери або відсутність feedback руйнують відчуття присутності сильніше, ніж низька деталізація текстур.

У XR необхідно проєктувати не тільки інтерфейс, а й зону безпечної взаємодії. Розробник враховує reach envelope, фізичні межі, сидячий або стоячий режим, домінантну руку, можливість повороту на 180 градусів, втомлюваність і доступність для користувачів із різними моторними можливостями (рис. 3.25).



Рис. 3.25. Компоненти імерсивної взаємодії: простір, тіло, об’єкт і feedback

3.7.2 Комфорт, locomotion і кіберхвороба

Кіберхвороба часто виникає через конфлікт між візуально сприйнятим рухом і сигналами вестибулярної системи. Тому locomotion необхідно

розглядати як comfort-critical механіку. Для навчального прототипу бажано передбачити щонайменше два режими: teleport і плавний рух із параметрами швидкості, snap/smooth turn, vignette та висоти.

Прискорення, різкі оберти, рух камери без дії користувача, head bob, низька частота кадрів і нестабільний latency збільшують ризик дискомфорту. Камеру в VR не слід «трясти» традиційним способом; impact краще передавати через haptics, звук, короткий world-space VFX або мінімальний контрольований pulse.

Комфортні налаштування повинні бути доступні до початку основної сесії. Вибір режиму locomotion, snap angle, поворот сидячи/стоячи, домінантна рука, висота та субтитри належать до базового options menu, а не до прихованого «розширеного» розділу (табл. 3.45) (рис. 3.26).

Таблиця 3.45

Порівняння режимів locomotion у VR

Рішення	Перевага	Ризик	Рекомендація
Teleport	високий комфорт	менше відчуття безперервного руху	дати зрозумілий landing marker і перевірку поверхні
Smooth locomotion	природний для жанру рух	кіберхвороба	контроль швидкості, vignette, стабільний frame time
Snap turn	менший сенсорний конфлікт	дискретність руху	налаштовуваний кут 15-45 градусів
Smooth turn	кінематографічність	високий ризик дискомфорту	регульована швидкість і попередження
Room-scale	тілесність і точність	фізичні межі простору	guardian/chaperone та безпечні reach zones
Seated mode	доступність і триваліші сесії	обмежена амплітуда	переналаштувати висоту та дальність взаємодії

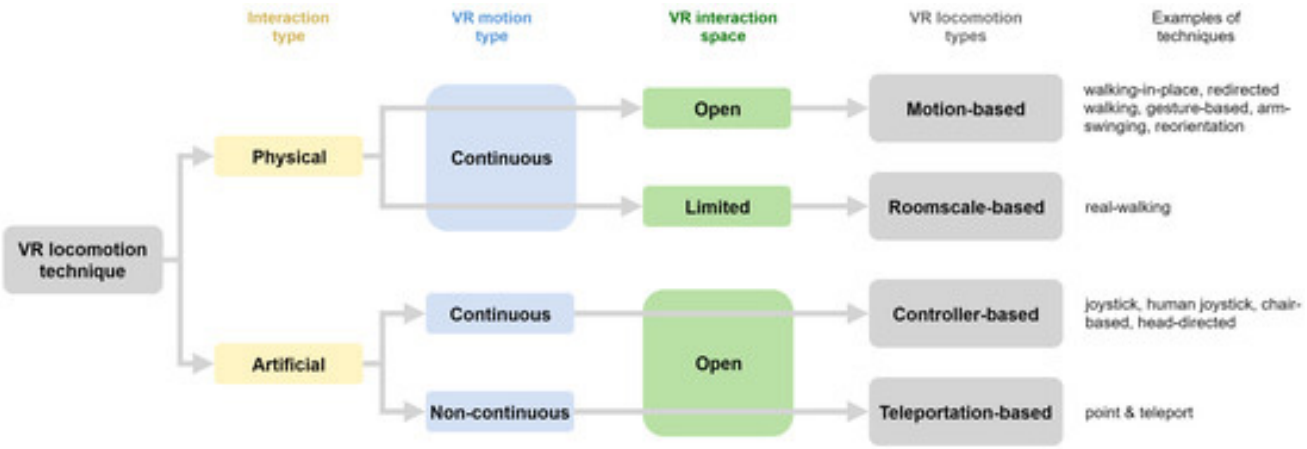


Рис. 3.26. Матриця комфортних параметрів VR locomotion

3.7.3 Просторова взаємодія, affordance і direct manipulation

Об’єкт у VR повинен комунікувати, як із ним взаємодіяти. Форма, положення, підсвітка, рукоятка, анімація й звук створюють affordance. Якщо кожен об’єкт потребує текстової інструкції, система не має сталої мови взаємодії.

Direct manipulation передбачає захоплення, переміщення, обертання, натискання або кидок. Ray interaction доречний для віддалених цілей, меню та точного вибору. Хороша система дозволяє комбінувати обидва способи, але зберігає послідовні правила фокусу, прив’язки, release, двох рук і пріоритетів.

Взаємодія має моделювати намір, а не вимагати лабораторної точності. Grab pose, snap points, magnetism, збільшена collision zone і velocity smoothing можуть підвищити надійність, якщо не руйнують очікування. У системі необхідні стани hover, select, active, invalid, release і failure з окремим feedback.

Для AR важливими є plane detection, anchors, occlusion і масштаб. Контент повинен залишатися стабільним відносно поверхні, не проходити крізь реальні об’єкти й не перекривати критичний простір. Перед розміщенням варто показати preview та пояснити, чому поверхня не прийнята (табл. 3.46; рис. 3.27).

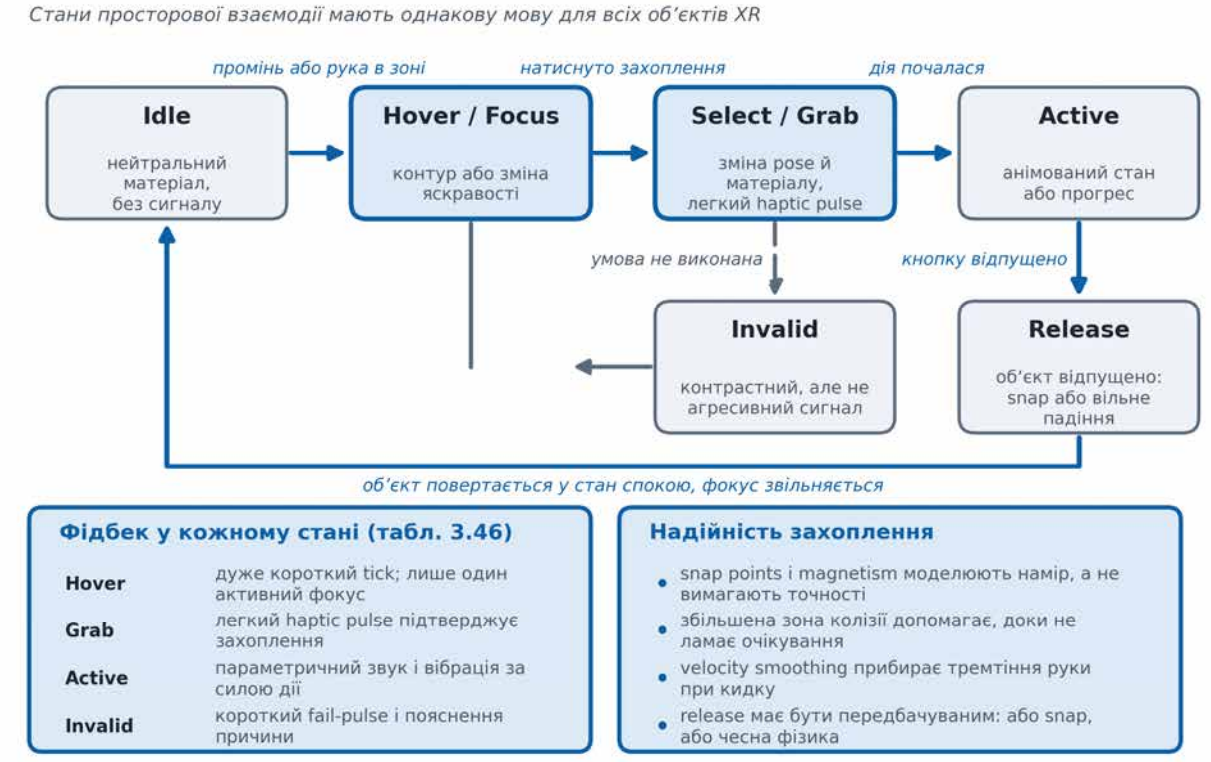


Рис. 3.27. Схема hover - select - active - release для XR-об’єкта

Таблиця 3.46

Стани просторової взаємодії

Стан об’єкта	Візуальний feedback	Аудіо/haptics	Правило
Idle	нейтральний матеріал	без сигналу	не привертати зайву увагу
Hover/Focus	контур або зміна яскравості	дуже короткий tick	один активний фокус
Grab/Select	зміна pose/матеріалу	легкий haptic pulse	підтвердити захоплення

Стан об’єкта	Візуальний feedback	Аудіо/haptics	Правило
Active	анімований стан або прогрес	параметричний звук/вібрація	feedback відповідає силі/часу
Invalid	контрастний, але не агресивний сигнал	короткий fail pulse	пояснити причину
Release/Complete	snap/settle/ефект завершення	confirm sound + haptic	закріпити результат

3.7.4 Spatial UI, звук і haptics

UI у просторі поділяють на world-locked, body-locked і head-locked. World-locked елементи добре пов’язані з об’єктом або місцем. Body-locked HUD рухається разом із користувачем, але не перекриває центр поля зору. Head-locked елементи слід застосовувати обережно: вони швидко викликають втому та відчуття «наклеєного» інтерфейсу.

Текст має бути достатнього кутового розміру, мати високий контраст і комфортну відстань. Плаваючі панелі варто розміщувати у зоні природного погляду, не змушувати користувача довго тримати голову повернутою та уникати дрібних цілей. У меню бажано підтримувати ray і direct touch, якщо це відповідає пристрою.

Просторовий звук допомагає знаходити події поза полем зору, підтверджує матеріал і відстань, формує межі простору. Haptics доповнює контакт, текстуру, силу та завершення дії. Вібрація повинна бути короткою, параметризованою й семантично послідовною: однаковий патерн не варто використовувати для успіху й помилки (рис. 3.28).

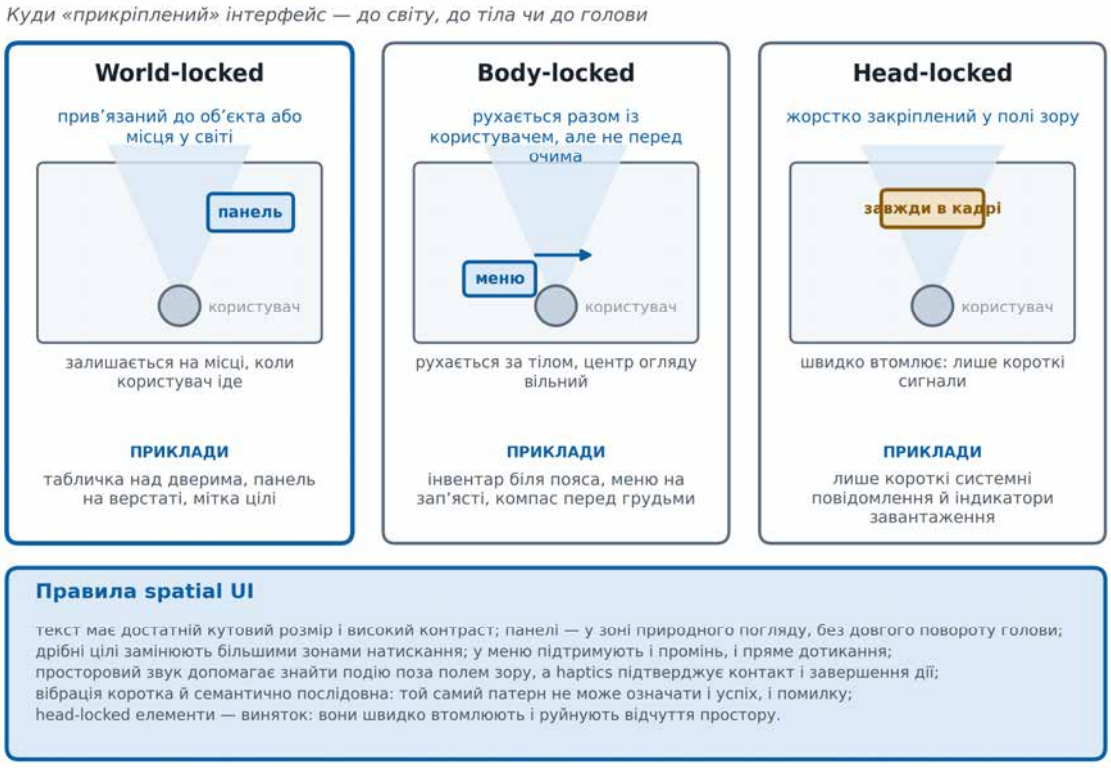


Рис. 3.28. Типи spatial UI: world-locked, body-locked і head-locked

4.1.1 Продуктивність і тестування XR

XR-проєкт має жорсткі вимоги до стабільності кадру, оскільки пропуски кадрів безпосередньо впливають на комфорт. Профілювання проводять на цільовому пристрої, а не лише в редакторі. Бюджети встановлюють для draw calls, полігонів, прозорості, освітлення, тіней, post-FX, частинок і пам'яті.

Тестування включає не тільки функціональність, а й фізичний комфорт: час до втоми, доступність об'єктів, читабельність тексту, точність grab, частоту помилкових активацій, стабільність anchors, поведінку при втраті tracking, сидячий режим і ліву/праву руку.

Для vertical slice достатньо однієї цілісної interaction loop: виявити об'єкт, сфокусуватися, захопити, виконати дію, отримати multimodal feedback, завершити й побачити результат. Якість цієї петлі важливіша за кількість різнорідних XR-функцій.

Практичний кейс «XR Interaction Slice»:

Створіть невелику сцену з трьома типами взаємодій: direct grab, ray selection та world-space UI. Додайте два режими locomotion, просторовий звук, haptic feedback, ліво-/праворукий режим і comfort settings. Проведіть тест із трьома сценаріями та зафіксуйте frame time, помилки взаємодії, дискомфорт і час виконання (табл. 3.47).

Питання для самоперевірки до розділу 3

1. Що входить до поняття game feel?
2. Чим first feedback відрізняється від завершення дії?
3. З яких етапів складається input latency?
4. Як frame time впливає на responsiveness?
5. Навіщо потрібні coyote time та input buffering?
6. Коли smoothing допомагає, а коли робить керування «в'язким»?
7. Які стани повинен мати character controller?
8. Як camera framing, lag і collision впливають на читабельність?
9. У чому перевага action-based input і context maps?
10. Як побудувати повторно використовуваний interactable?
11. Чому feedback потрібно запускати від підтвердженої події?
12. Як anticipation, impact і recovery проявляються в різних каналах?
13. Навіщо анімаційні notifies, curves та hit windows?
14. Коли застосовувати root motion, а коли in-place?
15. Як interrupt priority впливає на чесність і контроль?
16. Чим one-shot, loop, tail і stinger відрізняються?
17. Навіщо потрібні attenuation, priority і voice limits?
18. Що таке RTPC і які gameplay-параметри ним керують?
19. Як progressive disclosure зменшує когнітивне навантаження?
20. Які події telemetry допомагають оцінити onboarding?
21. Які причини motion sickness у VR?
22. Як дублювати feedback для accessibility без інформаційного шуму?

Практичні завдання до розділу 3

Завдання 1. Responsiveness Test Bench

Створіть три дії з debug HUD: стрибок/ривок, атака й interaction. Виміряйте first feedback, додайте buffering/coyote time, порівняйте два набори tunables і оформіть A/B-звіт.

Завдання 2. Interaction Audio & Animation Pack

Реалізуйте interaction із anticipation-impact-recovery, animation notify, VFX, layered SFX, spatial attenuation, одним RTPC і fail feedback. Додайте breakdown та відео 30-60 секунд.

Завдання 3. Onboarding і XR/Accessibility audit

Спроєктуйте перші п’ять хвилин досвіду, progressive hints, reminders і settings. Для XR або звичайної гри проведіть comfort/readability audit та запропонуйте мінімум п’ять покращень.

Висновки до розділу 3

1. Відчуття контролю виникає тоді, коли система швидко приймає ввід, передбачувано керує станами й синхронно відповідає кількома каналами. Надмірна кількість ефектів не компенсує невизначені правила або затримку.
2. Мультимодальний feedback потрібно планувати як систему з пріоритетами, бюджетами й альтернативними каналами. Animation, audio, VFX, UI, camera і haptics мають пояснювати одну подію, а не конкурувати між собою.

Графічні матеріали до розділу 3

Таблиця 3.47

Перелік рекомендованих графічних матеріалів до розділу 3

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
3.1	Принцип «first feedback < 100 ms» як стандарт якості прототипу	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.2	Спрощена шкала затримки: від input event до display	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.3	Ритм дії у фазах anticipation/impact/recovery (схема)	Подати серію з 3-5 кадрів anticipation – impact – recovery, додати часову шкалу та короткі примітки щодо читабельності й бюджету.
3.4	Action-based ввід: розділення «дія» та «прив’язка» (схема)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
3.5	Узагальнена схема grounding і переходів станів	Використати UML-подібну діаграму станів із переходами, умовами та пріоритетами; окремо позначити стани помилки та переривання.
3.6	Параметричний стрибок	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.7	Таймлайн coyote/buffer	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.8	Стек камерного ригу	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.9	Mini-pipeline test	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
3.10	Конвеєр взаємодії «input -> state -> feedback» (узагальнена схема)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.11	Приклад пріоритету цілі (score-based selection)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.12	Подієва інтеграція взаємодії та фідбеку	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.13	Мінімальна FSM взаємодії (приклад)	Використати UML-подібну діаграму станів із переходами, умовами та пріоритетами; окремо позначити стани помилки та переривання.
3.14	Схема «подія -> параметри -> мікс»	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.15	Формула шарування	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.16	Переходи ambience між зонами	Використати схему signal flow або багатошарову часову діаграму; позначити події, параметри, шини міксу та просторові зони.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
3.17	Приклад RTPC мапінгу	Використати схему signal flow або багатошарову часову діаграму; позначити події, параметри, шини міксу та просторові зони.
3.18	Схема locomotion (спрощено)	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.19	Логіка шарів	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.20	Таймлайн атаки	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.21	Цикл оптимізації анімації	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
3.22	Мікро-цикл навчання	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
3.23	«Від підказки до автоматизму»	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.24	Funnel приклад	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.25	Компоненти імерсивної взаємодії: простір, тіло, об’єкт і feedback	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.26	Матриця комфортних параметрів VR locomotion	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.27	Схема hover – select – active – release для XR-об’єкта	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
3.28	Типи spatial UI: world-locked, body-locked і head-locked	Додати анотований скріншот або wireframe з нумерованими зонами, ієрархією інформації, safe area та прикладами станів елементів.

РОЗДІЛ 4. ПРОГРАМНІ СИСТЕМИ, РУШІЇ ТА ПРОДУКТИВНІСТЬ

Мета розділу: розділ висвітлює програмну основу гри: модульну архітектуру, patterns, data-driven design, ECS, роботу в Unity й Unreal Engine, AI, контроль версій, build pipeline, debugging і профілювання.

Архітектура оцінюється не кількістю патернів, а здатністю безпечно змінювати механіки, тестувати системи, працювати командою та збирати стабільні build. Рішення повинні бути пропорційними scope і ризикам проєкту.

Оптимізація виконується на основі вимірювань. Спочатку визначають target і baseline, потім локалізують bottleneck, змінюють одну причину та повторюють capture в однаковому сценарії.

Після вивчення розділу здобувач набуває компетентностей:

• пояснювати компонентну архітектуру, SOLID, event-driven design та patterns;

• розрізняти GameObject/Actor, component, prefab/blueprint, scene/level та data asset;

• застосовувати state machines, pooling, factories, observers і command pattern;

• розуміти ECS/data-oriented підхід і межі його доцільності;

• планувати AI через FSM, Behavior Trees, GOAP, utility та navigation;

• організовувати Git/Perforce workflow, branching, LFS, naming і code review;

• налагоджувати build pipeline, logging, crash reproduction і release configuration;

• профілювати CPU, GPU, memory, rendering, physics, AI, audio та loading.

Ключові термсіни: software architecture, component, pattern, ECS, data-driven design, Unity, Unreal Engine, AI, VCS, build pipeline, profiling, optimization.

Тема 4.1. Ігрові програмні системи: архітектура, патерни, ECS та інструменти

Мета теми: сформувати у студентів системне розуміння архітектури ігрових програмних систем, принципів побудови модульного коду та типових патернів геймдеву; ознайомити з підходами component-based та ECS (Entity-Component-System), а також сформувати практичні навички профілювання і оптимізації (CPU/GPU/пам’ять) з використанням сучасних інструментів розробки

Очікувані результати навчання

• пояснювати різницю між архітектурою ігрового рушія, архітектурою ігрового застосунку та архітектурою геймплейних систем;

• розбивати ігровий проєкт на підсистеми (ввід, симуляція, UI, фізика, аудіо, рендеринг) і визначати межі відповідальності;

• застосовувати базові патерни (State, Command, Observer/Event Bus, Strategy, Factory, Object Pool) без зайвої складності;

• порівнювати OOP/component-based підхід з ECS та обирати доречний варіант під задачу і масштаб проєкту;

• планувати performance budgets і проводити профілювання: формувати гіпотези, вимірювати, оптимізувати, перевіряти результат;

• використовувати профайлери і інструменти діагностики (Unity Profiler, Unreal Insights, RenderDoc тощо) для пошуку вузьких місць;

• оформлювати мінімальну технічну документацію: архітектурна схема, список систем, правила коду, рішення (ADR).

Ключові терміни: Game loop, fixed timestep, update order, subsystem, module, dependency, cohesion/coupling, layered architecture, clean architecture, data-driven design, component-based architecture, ECS (entity, component, system), data-oriented design, cache locality, batching, draw calls, object pooling, event bus, state machine, command buffer, profiling, telemetry, frame budget, CPU bound, GPU bound, overdraw, LOD, occlusion culling, garbage collection, memory allocs, multithreading, job system, instrumentation, regression.

4.1.2 Архітектура ігрового застосунку: рівні абстракції та життєвий цикл (game loop)

Архітектура ігрового застосунку описує, як організовані модулі і підсистеми гри, як вони взаємодіють, у якому порядку виконуються, які дані передаються і хто за що відповідає. На практиці важливо розрізняти три близькі поняття: 1 – архітектура рушія (engine) (рендеринг, фізика, аудіо, ресурси, редактор), 2 – архітектура проєкту гри (модулі, сцени, сервісні шари), та 3 – архітектура геймплейних систем (бойова система, інвентар, квести, економіка). У студентських проєктах помилки найчастіше виникають через змішування цих рівнів.

Центральним механізмом виконання є game loop. У типовому варіанті кожен кадр обробляється введення (input), оновлюються стани (simulation), виконується рендеринг і звук, а також обробляються події. Для стабільності фізики та симуляції часто застосовується fixed timestep (фіксований крок), коли логіка симуляції виконується з постійним інтервалом (наприклад, 50 Гц), незалежно від FPS, а рендер може бути частішим (рис. 4.1).

4.1.3 Межі відповідальності: шари, модулі, залежності (cohesion/coupling)

Якісний код у грі – це практичний баланс між швидкістю розробки, читабельністю та можливістю змін. Для цього використовують принципи: висока зв’язність (cohesion) всередині модуля і низька зчепленість (coupling) між модулями. Простий робочий підхід – розділити гру на шари: Presentation (UI/камера), Gameplay (правила), Simulation (стан світу), Services (збереження, аналітика, мережа) і Platform (ввід/файли/SDK). Важливо, щоб високорівневі правила не залежали від деталей реалізації.

Порада: якщо геймплейний код ‘знає’ про конкретний UI-елемент або конкретну кнопку на сцені – ви з високою ймовірністю змішали відповідальності (табл. 4.1).

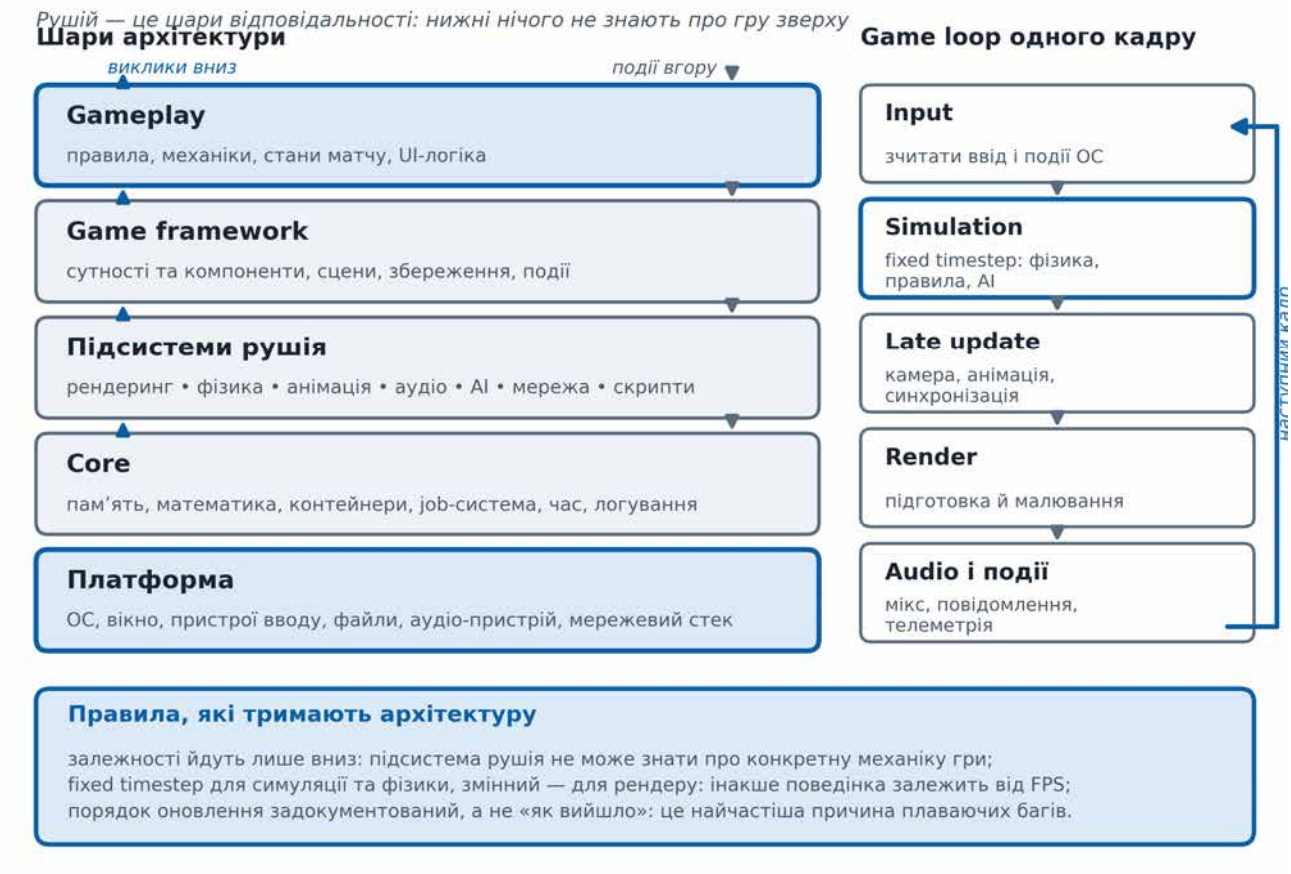


Рис. 4.1. Приклад структури ігрового рушія

Таблиця 4.1

Приклад підсистем гри і типові межі відповідальності

Підсистема	Відповідає за	Не повинна робити	Артефакти/дані
Input	зчитування вводу та перетворення у команди	керувати UI або симуляцією напряму	input map, command buffer
Gameplay rules	правила, стани, результати дій	працювати з кнопками/рендером напряму	state, events, configs
UI/HUD	відображення стану, взаємодія з меню	змінювати правила гри без перевірок	UI screens, bindings
VFX/Audio	фідбек подій, атмосфера	визначати перемогу/поразку	VFX list, audio cues
Save/Load	серіалізація прогресу	містити логіку прогресії	save schema, versioning
Telemetry/Analytics	логування подій, метрики	впливати на геймплей у рантаймі	events, dashboards

4.1.4 Патерни у геймдеві: які використовувати і навіщо

Патерни – це перевірені способи організації коду для типових проблем. У геймдеві важливо застосовувати їх прагматично: патерн має зменшувати складність, а не додавати зайві абстракції.

• State (стани) і finite state machine (FSM):

State корисний, коли об'єкт (персонаж, ворог, меню) має чіткі стани, які по-різному обробляють ввід і події. Наприклад: Idle, Move, Attack, Stunned. FSM дисциплінує переходи і спрощує відладку (рис. 4.2).

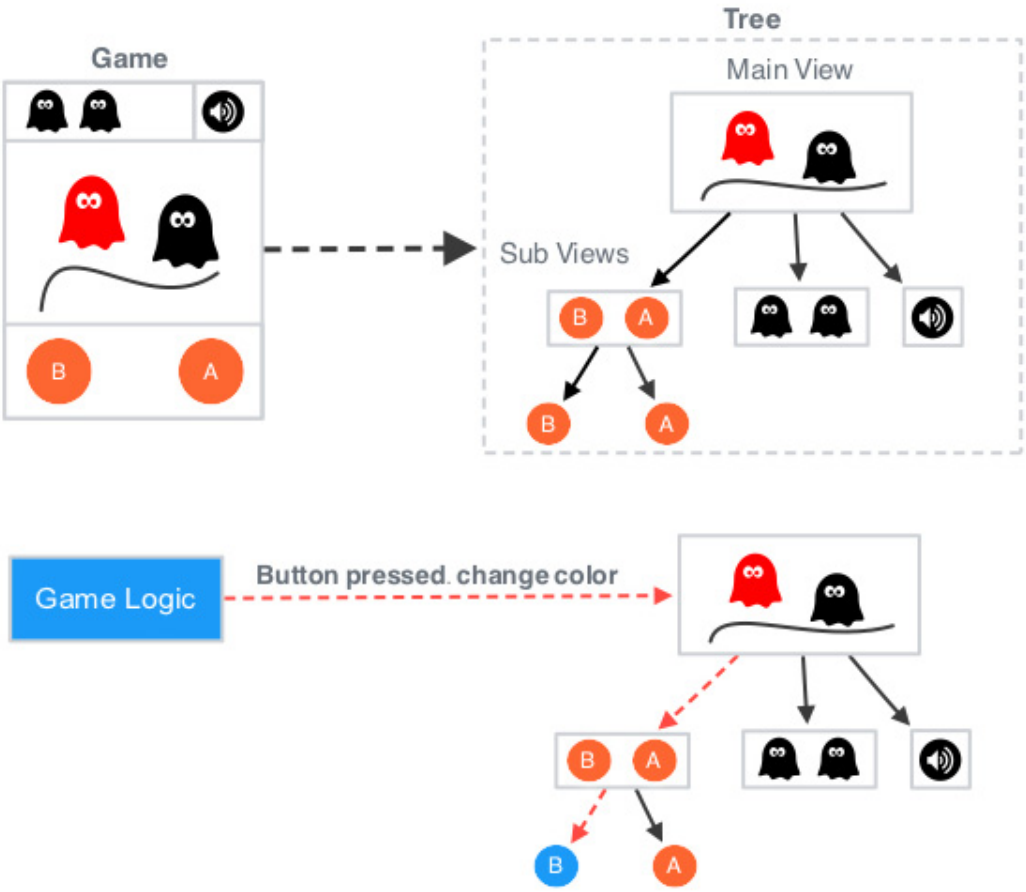


Рис. 4.2. Приклад застосування патернів програмування в ігровій розробці (https://www.haroldserrano.com/)

• Команди (Command) для вводу та реплею:

Command перетворює ввід у команди (MoveLeft, Jump, UseItem). Це дає буферизацію вводу, підтримку різних пристроїв, а також можливість запису/відтворення (replay) і тестування логіки.

• Події (Observer / Event Bus) без хаосу:

Observer або Event Bus дозволяють підписуватися на події (OnDamage, OnPickup, OnQuestComplete) і зменшують прямі залежності між системами. Важливо контролювати джерела подій і документувати список подій.

• Strategy та Factory для варіативних правил:

Strategy застосовується, коли алгоритм має варіанти (різні AI, різні формули шкоди). Factory потрібна, щоб централізовано створювати об'єкти (ворогів, предмети) і не дублювати логіку створення.

• Object Pool та Flyweight для продуктивності:

Object Pool зменшує витрати на часте створення/знищення об’єктів (кулі, частинки, ефекти), знижує алокації і навантаження на GC. Flyweight корисний, коли багато об’єктів ділять спільні дані (табл. 4.2).

Таблиця 4.2

Коли який патерн застосовувати

Проблема	Патерн	Сигнал, що час застосувати
Багато if-else по станах	State/FSM	кожна дія має різну логіку залежно від режиму
Потрібен буфер вводу або replay	Command	перевага тестованості і відтворення
Системи сильно пов’язані	Observer/Event Bus	UI, VFX, аудіо залежать від геймплею напряму
Алгоритм має варіанти	Strategy	поведінка змінюється за конфігом або типом об’єкта
Часті spawn/destroy	Object Pool	FPS просідає, GC allocs ростуть

4.1.5 Data-driven підхід: конфіги, баланс і швидкі ітерації

Data-driven підхід означає, що параметри (шкода, швидкість, ціни, cooldown) зберігаються у даних, а не зашиті в коді. Це спрощує баланс і командну роботу. У Unity це часто ScriptableObject; в Unreal – Data Assets/Data Tables. Конфіги мають проходити валідацію і мати версійність.

4.1.6 ECS та Data-Oriented Design: коли і навіщо

ECS (Entity-Component-System) – підхід, де entity є ідентифікатором, компоненти містять лише дані, а системи виконують логіку над наборами компонентів. Перевага – продуктивність при великій кількості однотипних сутностей (натовпи, рої, масові NPC) завдяки data-oriented організації і кращій cache locality.

Порівняння: OOP/component-based vs EC

У component-based моделі об’єкт містить компоненти з даними і методами, що зручно для прототипу. ECS переносить логіку у системи і групує дані, що спрощує оптимізацію та паралельність. Практичне правило: десятки об’єктів – component-based; сотні/тисячі однотипних сутностей і performance проблеми – ECS або гібрид (табл. 4.3).

Таблиця 4.3

ECS: плюси/мінуси для навчальних проєктів

Критерій	Переваги	Ризики/обмеження
Продуктивність	краща cache locality, легше паралелити	потрібно вміти профілювати і мислити даними
Масштаб	добре для великої кількості сутностей	для малих проєктів – зайва складність
Архітектура	чітке розділення даних і логіки	складніше дебажити новачкам
Інструменти	Unity DOTS/Entities, Unreal Mass	екосистема може змінюватися, потрібна дисципліна

4.1.7 Оптимізація: методологія ‘виміряй -> зміни -> перевір’ (CPU, GPU, пам’ять)

Оптимізація без вимірювань майже завжди веде до марної роботи. Правильний процес: 1) ціль і KPI, 2) baseline, 3) пошук bottleneck, 4) гіпотеза, 5) зміна, 6) перевірка. Вузьке місце може бути на CPU, на GPU, у пам’яті (алокації/GC) або в I/O (стримінг ресурсів) (рис. 4.3).

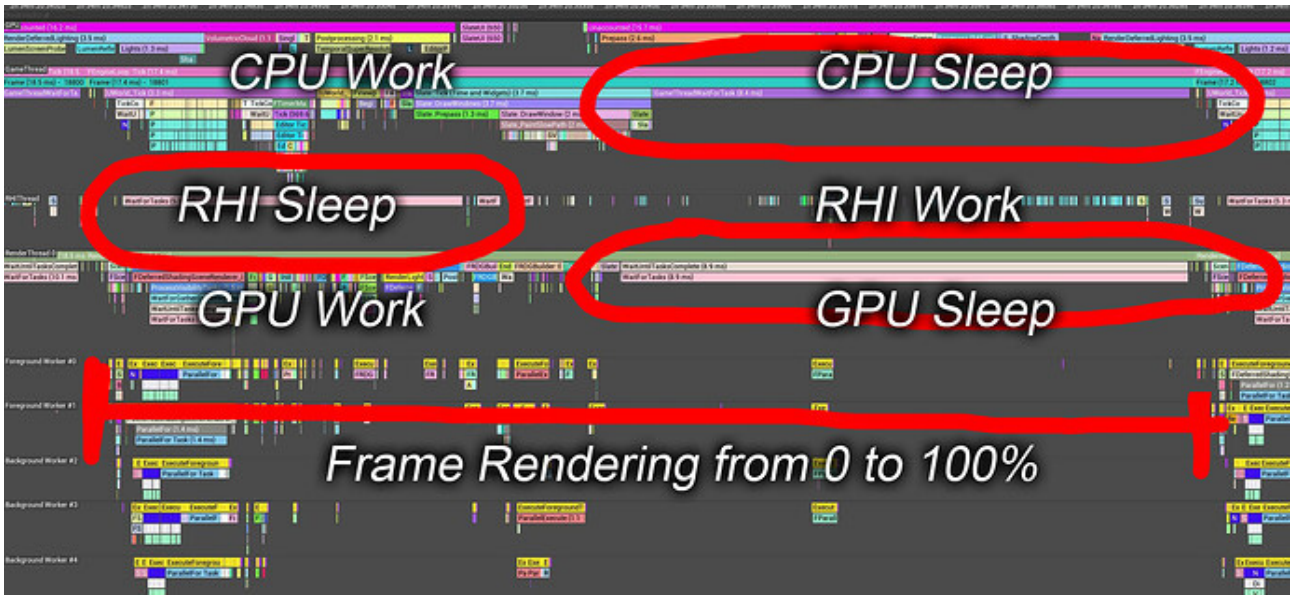


Рис. 4.3. Приклад візуалізації систем оптимізації в ігровій розробці

Frame budget і KPI продуктивності: Зручно мислити в мілі секундах: 60 FPS = 16.67 мс на кадр, 30 FPS = 33.33 мс, 90 FPS (VR) = 11.11 мс. У навчальному проєкті варто мати KPI: середній FPS, піковий frame time, кількість GC allocs/сек.

Типові джерела CPU-повільності

надлишкові оновлення (Update на сотнях об’єктів без потреби);
дорогі пошуки у сцені (Find, GetComponent у циклах), часті алокації;
складна фізика або велика кількість колайдерів;
AI без таймінгів (все обчислюється щокадру);
погане управління ресурсами (завантаження під час гри без кешування).

Типові джерела GPU-повільності

велика кількість draw calls і матеріалів (немає batching/instancing);
overdraw через прозорість (UI, частинки, пост-ефекти);
високі роздільності текстур і важкі шейдери;
динамічні тіні та складні пост процеси без бюджету.

4.1.3 Техніки оптимізації у грі: системні рішення і швидкі перемоги

Оптимізацію поділяють на архітектурні, алгоритмічні, контентні та рушійні налаштування. Найчастіше найбільший ефект дають: object pooling, зменшення прозорості VFX, об’єднання матеріалів, LOD, обмеження частоти оновлення AI, кешування посилань (табл. 4.4).

Таблиця 4.4

Швидкі оптимізації і очікуваний ефект (орієнтовно)

Проблема	Що зробити	Де шукати	Ризик
GC allocs ростуть	Object Pool, уникати алокацій у кадрі	Profiler: GC/Alloc	потрібна дисципліна життєвого циклу
CPU bound (логіка)	тайм-слайс AI, кешувати посилання, прибрати Find()	Profiler: CPU	можна змінити поведінку
GPU bound (рендер)	зменшити draw calls, instancing, спростити шейдери	Frame Debugger / RenderDoc	може змінити вигляд
Overdraw	скоротити прозорі шари, VFX бюджет	GPU профайл / overdraw view	потрібні правки контенту
Стрімінг/підвантаження	preload, кеш, Addressables/Asset Manager	load times профіль	збільшить RAM

4.1.4 Інструменти: профайлери, відладка, телеметрія, збірка

Мінімальний набір: CPU/GPU профайлер, логування, frame capture, базова телеметрія. Підхід ‘спочатку інструменти’ економить час і робить проєкт керованим.

Профайлери та аналіз кадру

Unity Profiler, Profiler Modules: CPU, GPU, Memory, Rendering.

Unity Frame Debugger (draw calls, batching), RenderDoc для capture кадру.

Unreal Insights (CPU/GPU traces), Stat commands (stat unit, stat gpu).

PIX (Windows), Xcode Instruments (iOS), Android GPU Inspector (Android).

Логи, алерти, краш-репорти: Логи потрібні для багів і розуміння поведінки систем. Важливо мати рівні логування і не спамити в релізі. Для крашів корисні Sentry або Firebase Crashlytics (для навчальних проєктів достатньо базової інтеграції).

Регресія продуктивності: Проста практика: мати контрольну сцену і перевіряти базові значення FPS/пам’яті після великих змін. Це оформлюють як performance checklist.

4.1.5 Документація і технічний дизайн: як зробити проєкт підтримуваним

Мінімальний Technical Design включає: архітектурну схему, список систем і відповідальностей, перелік ключових даних і подій, правила коду. Корисна практика – ADR (Architecture Decision Records): коротко фіксувати, чому обране рішення (наприклад, ‘пул для куль зменшив GC’).

Питання для самоперевірки

1. Поясніть різницю між архітектурою рушія та архітектурою ігрового застосунку.

2. Що таке game loop і навіщо використовують fixed timestep?

3. Назвіть 5 підсистем гри і опишіть межі відповідальності для двох з них.

4. Коли доцільно застосувати патерн State? Наведіть приклад зі станами персонажа.

5. Для чого потрібен Command у системі вводу? Яку проблему він вирішує?

6. Чому події (Event Bus) можуть як допомогти, так і нашкодити проєкту?

7. Що означає data-driven підхід і наведіть 3 параметри, які краще винести у конфіг.

8. Що таке ECS: entity, component, system? У яких сценаріях ECS дає найбільшу користь?

9. Опишіть методологію оптимізації: що робити перед тим, як ‘оптимізувати код’?

10. Наведіть 3 типові причини CPU bound і 3 причини GPU bound у грі.

11. Що таке draw call, batching, overdraw? Чому прозорість часто дорога?

12. Які інструменти ви використаєте для профілювання і як зафіксуєте baseline продуктивності?

Практичний кейс (час виконання 120-180 хв): «Мікро-гра з архітектурою та профілюванням»

Мета: зібрати невеликий прототип (top-down shooter або простий платформер), де продемонстровані архітектура систем, 2-3 патерни та базове профілювання/оптимізація.

Умова – оберіть один сценарій

Top-down shooter: рух, стрільба, вороги-спавн, HP, UI, 2 VFX (постріл/хіт).

Платформер: рух, стрибок, пастка, колекційний предмет, UI, 1 анімація станів.

Wave survival: хвилі ворогів, 1 здібність з cooldown, простий лут, UI.

Кроки виконання наступні:

Архітектура (15-20 хв): намалюйте схему модулів (Input, Gameplay, UI, VFX/Audio, Save/Telemetry). Вкажіть напрям залежностей.

Патерни (20-30 хв): реалізуйте мінімум два патерни: State та Command або State та Event Bus.

Data-driven (15-20 хв): винесіть параметри у конфіг (ScriptableObject/DataTable/JSON) і додайте валідацію діапазонів.

Object Pool (15-25 хв): для куль/ефектів або ворогів реалізуйте пул, щоб зменшити spawn/destroy.

Профілювання (20-30 хв): зніміть baseline (FPS, frame time, GC allocs, draw calls). Зафіксуйте 3 найдорожчі точки

Оптимізація (20-30 хв): внесіть 2-3 зміни і повторіть вимірювання (порівняння до/після)

Підсумок (10-15 хв): коротко опишіть, що змінилося і чому. Додайте 1 ADR на 5-8 рядків

Очікувані результати (deliverables)

Архітектурна схема (1 стор.) із межами відповідальності підсистем;
список використаних патернів і місця застосування;
конфіг параметрів (data-driven) і приклад зміни балансу без правки коду;
профайл-репорт: baseline та after (мінімум 3 показники);
1 ADR: опис рішення та обґрунтування.

Критерії оцінювання

Логічність архітектури і відсутність зайвих залежностей;
коректність застосування патернів без надмірної абстракції;
наявність реальних вимірювань і висновків;
читабельність коду і мінімальна техдокументація.

Рекомендовані програми та сервіси

1. Рушії: Unity (Profiler, Frame Debugger), Unreal Engine (Unreal Insights, Stat tools).
2. IDE: Visual Studio / Rider, VS Code.
3. Профілювання та capture: RenderDoc, PIX, Xcode Instruments, Android GPU Inspector.
4. Контроль версій: Git (GitHub/GitLab) та Git LFS; Perforce (за потреби).
5. Трекінг задач/документація: Trello/Jira, Notion/Confluence, Miro.

Корисні посилання (вставити в документ як активні URL)

1. Unity Profiler: https://docs.unity3d.com/Manual/Profiler.html
2. Unity Frame Debugger: https://docs.unity3d.com/Manual/FrameDebugger.html
3. Unity DOTS (Entities): https://docs.unity3d.com/Packages/com.unity.entities@latest
4. Unreal Gameplay Framework: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-framework
5. Unreal Insights: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-insights
6. Unreal Mass: https://dev.epicgames.com/documentation/en-us/unreal-engine/mass-entity
7. RenderDoc: https://renderdoc.org/.

Рекомендована література до теми

1. Nystrom R. Game Programming Patterns.
2. Gregory J. Game Engine Architecture.
3. Martin R. Clean Architecture.
4. Unity/Unreal офіційні гіди з оптимізації та профілювання.

Тема 4.2. Розробка в Unity та Unreal Engine: прототипування, ітерації й інтеграція контенту

Мета теми: сформувати у студентів системне розуміння процесу створення ігрового прототипу в Unity/Unreal Engine, навчити планувати ітерації, організовувати структуру проєкту, інтегрувати контент (2D/3D, матеріали, анімацію, звук, UI, VFX), проводити плейтести та впроваджувати зміни на основі зворотного зв'язку з урахуванням вимог до продуктивності й стабільності збірки

Очікувані результати навчання

• вміти пояснювати відмінності Unity та Unreal Engine, обґрунтовувати вибір рушія під жанр/платформу/команду;

• створювати структуру проєкту, прийнятну для командної роботи (папки, неймінг, сцени/рівні, правила контенту);

• будувати прототипи різних типів (greybox, spike, MVP, vertical slice) і вибирати доцільний тип прототипу;

• організовувати ітерації та плейтест з метриками і протоколом спостережень;

• інтегрувати контент у рушій (імпорт, матеріали, колайдери, LOD, анімація, аудіо, UI, VFX);

• застосовувати базові інструменти профілювання (Unity Profiler/Frame Debugger, Unreal Insights) для виявлення «вузьких місць»;

• формувати мінімальний «production-ready» прототип із чеклистом якості та планом робіт на 2 тижні.

Ключові терміни: Unity, Unreal Engine, Prefab/Blueprint, Scene/Level, Greybox/Blockout, Spike, MVP, Vertical Slice, Iteration, Playtest, Feedback Loop, Content Integration, Import Settings, LOD, Occlusion, Draw Calls, Overdraw, Build Pipeline, Profiling, Unity Profiler, Frame Debugger, Unreal Insights, Cook/Package, UGUI, UI Toolkit, UMG, Niagara, VFX Graph, Addressables, Data Assets, Scriptable Objects, Version Control, CI/CD.

4.1.6 Unity та Unreal Engine: філософія, типові сценарії, критерії вибору рушія

Unity та Unreal Engine – два провідні рушії, які часто використовують у навчальних і комерційних проєктах. Вибір рушія має ґрунтуватися не лише на «популярності», а на відповідності задачам команди, жанру, платформи та обмеженням часу.

Unity традиційно сильний як універсальний рушій для швидкого прототипування, мобільних і 2D/3D проєктів, інді-ігор, освітніх симуляторів, AR/VR. Важливими перевагами є простота входження, велика кількість прикладів, гнучкі підходи до архітектури, а також наявність різних UI-систем (UGUI та UI Toolkit).

Unreal Engine має потужний пайплайн візуалізації, інструменти для побудови великих 3D-проєктів, розвинену екосистему Blueprint для швидкого складання логіки, а також сильні системи рендерингу, освітлення та VFX

(Niagara). Unreal часто обирають для AAA-подібних 3D рішень, кінематографічних сцен, високоякісних демонстрацій та продуктів, де важливі візуальні можливості.

Критерії вибору рушія

Платформа (PC/Console/Mobile/VR) і вимоги до продуктивності;

Жанр (2D платформер, top-down, FPS, RPG, симулятор тощо);

Команда (досвід, сильні сторони, доступність технічного арту);

Термін (скільки часу на MVP/vertical slice);

Контент (чи є готові асети, формат моделювання, анімація);

Інструменти і пайплайн (версійність, складання build, профілювання, налагодження) (табл. 4.5).

Таблиця 4.5

Порівняння Unity та Unreal (для навчальних прототипів)

Критерій	**Unity**	**Unreal Engine**
Поріг входження	Низький/середній (C#, багато туторіалів)	Середній (Blueprint полегшує старт, з C++ старт складніше)
Швидкість прототипування	Дуже висока	Висока (особливо через Blueprint)
Візуальна якість «з коробки»	Добра, залежить від пайплайну	Дуже висока (рендеринг, освітлення, Niagara)
UI	UGUI та UI Toolkit	UMG
VFX	VFX Graph/Particles	Niagara
Мобільні платформи	Сильна сторона	Можливо, але часто складніше оптимізувати
Командний пайплайн	Git та LFS/Plastic/Perforce	Часто Perforce; Git можливий
Профілювання	Unity Profiler, Frame Debugger	Unreal Insights, профайлінг CPU/GPU
Типові курсові проєкти	2D/3D прототипи, мобайл, AR/VR	3D сцени, cinematic, shooter, високий візуал

4.1.7 Організація проєкту в рушії: структура, неймінг, сцени/рівні, командний пайплайн

Найпоширеніша причина «хаосу» у студентських проєктах – відсутність правил організації контенту. В результаті: дублікати асетів, зламані посилання, складні мерджі, проблеми зі збіркою.

Базові принципи організації

• Єдина структура папок для всієї команди

• Один стандарт неймінгу (активи, сцени, префаби/блюпринти)

• Сцени/рівні мають чіткі ролі: MainMenu, Level_01, Sandbox, Test_UI, Test_VFX

• Валідація імпорту: розміри, префікси, налаштування імпорту (scale, compression)

• Версійність: VCS, правила комітів, LFS/Perforce для бінарних файлів
• Виділені тестові сцени (не ламати основну) (табл. 4.6).

Таблиця 4.6

Приклад структури папок (Unity/Unreal-подібна логіка)

Папка	Вміст	Примітка
Project/Docs	короткі специфікації, ADR, лінки	щоб не губити правила
Art/2D	спрайти, UI-графіка	сортувати за підпапками
Art/3D	моделі, матеріали, текстури	LOD/колайдери окремо
Audio	SFX, Music, Mix	неймінг за подіями
Prefabs (Unity) / Blueprints (UE)	ігрові об’єкти	«жива» частина контенту
Scenes (Unity) / Maps (UE)	рівні/сцени	правило: 1 рівень = 1 сцена
Scripts (Unity) / Стата (UE)	код	розбиття за системами
UI	UI Prefabs/Widgets	екрани та компоненти
VFX	системи ефектів	окремий каталог
Settings/Config	конфіги, баланс	data-driven підхід

4.1.8 Прототипування: greybox, spike, MVP і vertical slice у контексті ігрового проєкту

Прототипування – це не «зробити швидко аби було», а метод перевірки гіпотез про геймплей, UX і технічну реалізовність.

Типи прототипів

- Greybox/Greyboxing (Blockout): перевірка простору, керування, темпу, навігації без фінального арту.
- Spike: коротке технічне дослідження ризику (наприклад, «чи потягне мобільний девайс 30 NPC?»).
- MVP (мінімально життєздатний продукт): мінімальний набір механік, що створює ігрову цінність.
- Vertical Slice: «вертикальний зріз» - невелика частина гри з майже «продакшен»-якістю (арттаUIтазвуктаVFXтафлоу) (рис. 4.4).

4.2.5 Ітерації, плейтест та зворотний зв’язок: як керувати змінами без хаосу

Ітерації – це керовані цикли зміни гри. Без правил ітерації перетворюються на «переробляємо все щодня», що вбиває терміни.

Правило 1: кожна ітерація має гіпотезу.

Приклад: «Якщо додати телеграф атаки і звуковий сигнал, гравці перестануть пропускати небезпеку».

Правило 2: фіксуємо спостереження, а не оцінки.

Не «погано/добре», а «гравець 30 секунд шукав кнопку», «не помітив маркер».

Правило 3: одна ітерація – 1-3 зміни, не 20 (табл. 4.7).

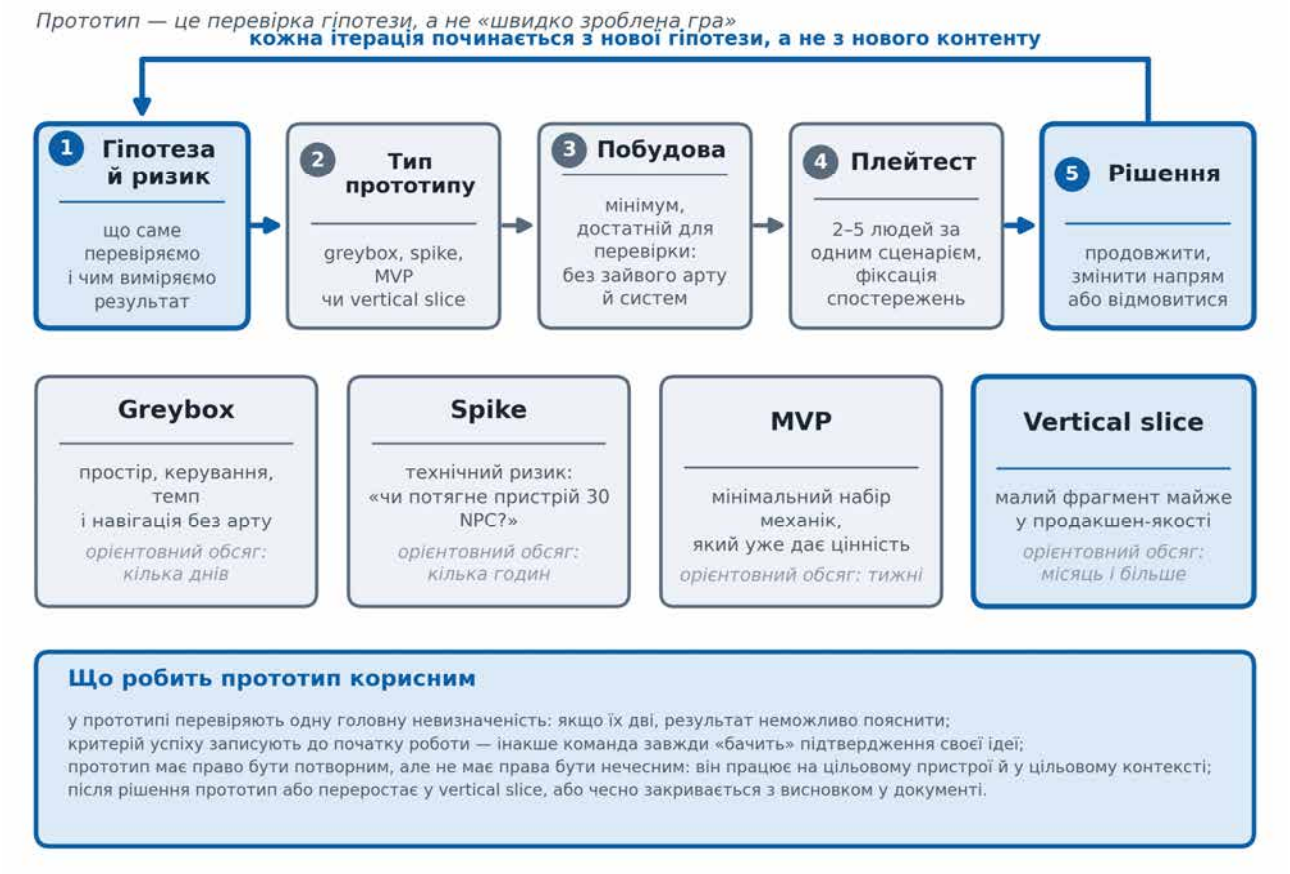


Рис. 4.4. Ітеративний процес створення прототипу

Таблиця 4.7

Шаблон протоколу плейтесту (мінімальний)

Поле	**Зміст**
Версія білду	v0.3.2
Тестер	ім’я/роль (новачок/досвідчений)
Сценарій	що тестуємо (рівень 1, tutorial, UI)
Метрика	час проходження, кількість помилок, FPS
Спостереження	що робив/де застряг/що не помітив
Висновок	причина проблеми (гіпотеза)
Рішення	1-3 правки на наступну ітерацію

4.2.6 Інтеграція контенту: імпорт 2D/3D, матеріали, освітлення, звук, анімація

Інтеграція контенту – критичний етап: навіть хороший асет може «зламати» FPS або виглядати погано через неправильний імпорт (рис. 4.5).

Engine-ready assets checklist (коротко)

• правильний масштаб (1 unit = 1 метр, якщо так домовились);
• pivot/center налаштовані під використання;
• колайдери/mesh-collider визначені (де потрібно);

• текстури мають контроль розміру/формату/компресії;
• матеріали узгоджені (PBR, нормалі, roughness);
• LOD (за потреби) або хоча б план LOD;
• звуки мають неймінг за подією (SFX_UI_Click, SFX_Hit_Light);
• анімації мають чіткі назви і стани.

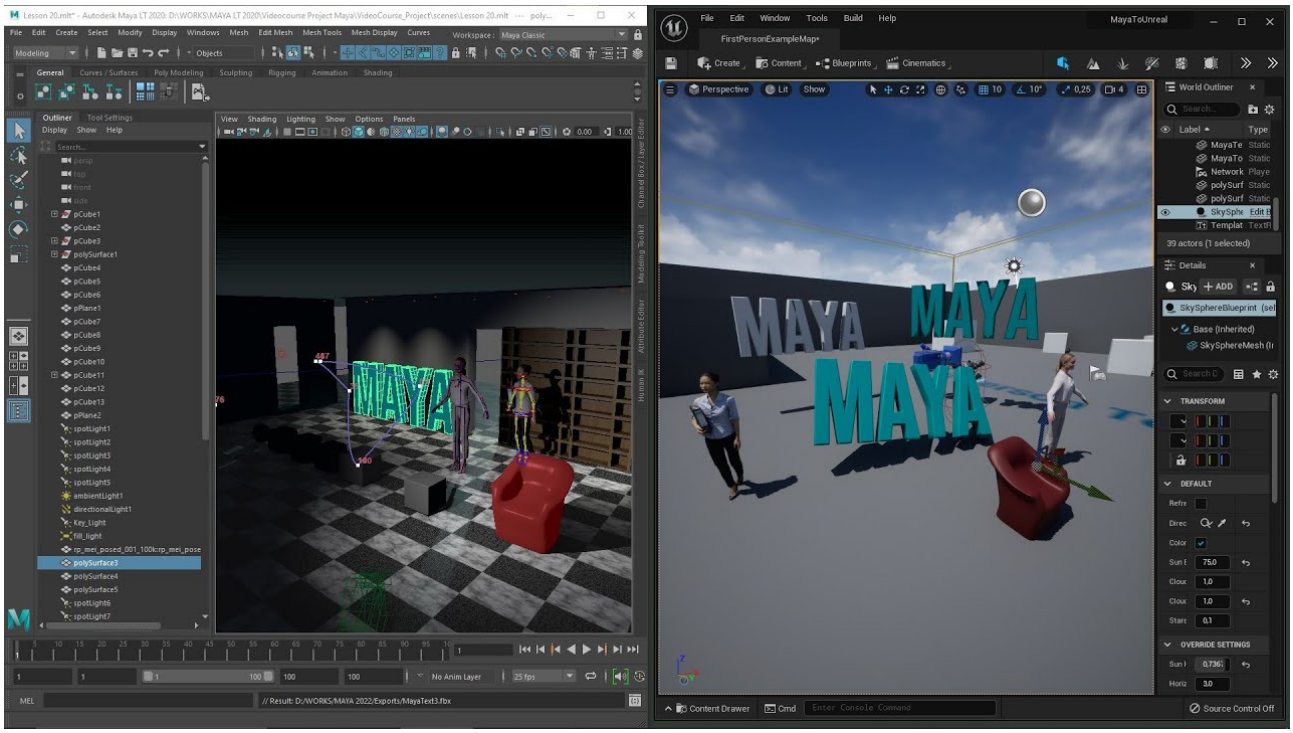


Рис. 4.5. Asset-to-engine - приклад Maya UE5

4.3.1 Прототипування та інтеграція UI: UGUI/UI Toolkit (Unity) і UMG (Unreal)

UI в іграх – це частина геймплею. Прототип UI має з'явитися рано: ще на етапі MVP.

Unity: UGUI – швидко, просто, багато прикладів; UI Toolkit – сучасніший підхід, корисний для складних інструментів/меню.

Unreal: UMG – потужний UI-редактор; добре підтримує виджети, анімацію, прив'язки (табл. 4.8).

Таблиця 4.8

Швидкі правила UI (читабельність і адаптація)

Правило	**Практичний тест**
Контраст	текст читається на фоні з 1 м?
Ієрархія	за 1 секунду видно головне?
Масштабування	перевірити 2-3 роздільності
Мінімалізм	прибрати 20% - чи стало краще?
Доступність	не використовувати лише колір як сигнал

4.3.2 Інтеграція VFX та «відчуття гри» (game feel): події, читабельність, продуктивність

VFX мають бути прив'язані до подій гри (event-driven): удар, підбір предмета, активація здібності, отримання шкоди.

Подієва інтеграція UI/VFX/Audio: Game Event -> (VFX та Audio та UI feedback) -> Player perception -> Decision

Правила

• VFX має бути читабельним (особливо в бою);

• не перевантажувати екран (контроль «шуму»);

• враховувати overdraw, кількість частинок, шейдери.

4.3.3 Сцени/рівні, префаби та інтеграція контенту у геймплей: практичні підходи

Ключовий принцип: не будувати рівень «з нуля кожного разу» - повторно використовувати Prefab/Blueprint.

Практичні підходи

• робити «композиційні» префаби: двері та колайдер та звук та VFX;

• мати тестову сцену для перевірки асетів;

• використовувати data-driven конфіги для параметрів (damage, speed, cooldown).

4.3.4 Збірка, тестування та профілювання: як забезпечити стабільність і продуктивність прототипу

Прототип має бути не лише «працює на моєму ПК», а відтворювано збирається й тестується.

Unity

- Unity Profiler (CPU, GC, rendering),
- Frame Debugger (черга рендеру),
- Build settings та перевірка на цільовому девайсі.

Unreal

- Unreal Insights (CPU/GPU tracing),
- профілювання рендеру і завантаження,
- коректний cook/package.
-

4.3.5 Мінімальний «production-ready» прототип: чеклист готовності і приклад плану ітерацій

«Production-ready прототип» у навчальному курсі – це прототип, який

• відтворювано запускається на цільовій платформі;

має стабільний базовий флоу (меню -> рівень -> результат);

• має базові UI/аудіо/VFX, які підтримують геймплей;

• не падає при стандартному сценарії гри;

• має коротку документацію (як запускати, що тестувати).

Приклад плану на 2 тижні (ітерації)

• Дні 1-3: MVP флоу та greybox рівня

• Дні 4-6: інтеграція контенту (2-3 асети та базовий матеріал та звук)
• Дні 7-8: UI/HUD та фідбек
• Дні 9-10: VFX та анімація станів
• Дні 11-12: плейтест та протокол та правки
• Дні 13-14: профілювання та «полірування» та збірка для показу

Питання для самоперевірки

1. Назвіть 5 критеріїв вибору рушія Unity/Unreal під навчальний проєкт.
2. Чим відрізняються greybox, spike, MVP, vertical slice?
3. Які правила структури папок і неймінгу зменшують хаос у команді?
4. Що обов'язково фіксувати в протоколі плейтесту?
5. Наведіть приклад гіпотези для ітерації.
6. Які 7 пунктів містить «engine-ready assets checklist»?
7. Чим відрізняються UGUI/UI Toolkit і UMG?
8. Що таке event-driven VFX і чому це корисно?
9. Назвіть 3 типові причини падіння FPS у прототипі.
10. Який цикл оптимізації є базовим (measure -> fix -> verify)?
11. Що означає «production-ready» прототип у контексті курсу?
12. Які 5 речей треба перевірити перед демонстрацією білду?

Практичний кейс (час виконання 120-180 хв): «Playable Prototype та інтеграція контенту в Unity або Unreal»

Завдання: створити ігровий прототип з базовим флоу і інтегрованим контентом.

Сценарії – оберіть один із запропонованих

A: Top-down арена (рух, атака, 1 ворог, UI HP, 1 VFX hit, звук)
B: Платформер (рух, стрибок, пастка, колекційний предмет, UI)
C: Простий survival-loop (взаємодія, індикатор ресурсу, UI, звук)

Кроки виконання:

Створіть структуру проєкту та правила неймінгу
Зробіть greybox рівня (1 сцена/карта)
Реалізуйте MVP-механику (1 core loop)
Додайте UI/HUD (мінімум: 2 показники та 1 підказка)
Інтегруйте 2-3 асети (2D/3D) та матеріал/текстури
Додайте 1-2 звуки та 1 VFX на подію
Проведіть плейтест із 2 людьми, заповніть протокол
Зробіть 1 ітерацію за результатами плейтесту
Проведіть коротке профілювання (скрін/нотатка що саме «важке»)
Зберіть build і перевірте запуск на іншому ПК/пристрої

Що необхідно здати

• Посилання на репозиторій/архів проєкту;
• 1 стор. «Readme» (як запустити, управління, цілі тесту);
• протокол плейтесту (табличка);

короткий звіт: 3 проблеми -> 3 рішення -> результат.

Критерії оцінювання

• працездатність флоу (меню/рівень/результат);
• якість інтеграції контенту (читабельність, консистентність);
• наявність плейтесту та ітерації (реальні зміни);
• базове профілювання і усвідомлені оптимізації;
• охайність структури та неймінгу.

Рекомендовані програми та сервіси

1. Unity (LTS), Unity Profiler, Frame Debugger
2. Unreal Engine, Unreal Insights
3. Blender / Figma / Krita / GIMP
4. Audacity (SFX), FMOD (за бажанням)
5. Git та LFS / Perforce (за потреби), GitHub/GitLab
6. Trello/Notion для плану ітерацій

Рекомендовані джерела та матеріали

1. Unity Manual: https://docs.unity3d.com/Manual/
2. Unity Profiler: https://docs.unity3d.com/Manual/Profiler.html
3. Unity Frame Debugger: https://docs.unity3d.com/Manual/FrameDebugger.html
4. Unity Addressables: https://docs.unity3d.com/Packages/com.unity.addressables@latest
5. Unreal Engine Documentation: https://dev.epicgames.com/documentation/en-us/unreal-engine
6. Unreal Insights: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-insights
7. UMG UI Designer: https://dev.epicgames.com/documentation/en-us/unreal-engine/umg-ui-designer
8. Niagara VFX: https://dev.epicgames.com/documentation/en-us/unreal-engine/niagara-visual-effects
9. Blender Manual: https://docs.blender.org/manual/en/latest/
10. Figma Learn: https://help.figma.com/

Тема 4.3. Штучний інтелект і data-driven gameplay systems

Мета теми: сформувати практичне розуміння архітектури ігрового AI: perception, decision making, navigation, action execution і feedback; навчити обирати між FSM, Behavior Tree, Utility AI та GOAP і документувати параметри для ітерацій.

4.3.6 AI як система поведінки, а не «розумний персонаж»

Ігровий AI оцінюють не за академічною складністю алгоритму, а за тим, наскільки поведінка є зрозумілою, послідовною, керованою та цікавою. Ворог може бути технічно простим, але створювати сильний досвід завдяки доброму telegraphing, ролям у групі, просторовим обмеженням, реакціям на гравця та прогнозованим правилам.

Корисно розділяти систему на п’ять шарів: perception збирає факти; memory/context зберігає релевантний стан; decision making обирає намір; navigation/steering планує переміщення; action execution запускає анімацію, здібність, VFX, audio й зміну станів. Така декомпозиція спрощує тестування та заміну окремих алгоритмів.

AI повинен підтримувати pillars і темп гри. У stealth-прототипі важлива читабельна шкала підозри та передбачувані правила огляду. У combat arena потрібні ролі, attack tokens і контроль одночасного тиску. У companion AI пріоритетом є не оптимальність, а відсутність блокування гравця, доречна підтримка й стабільна навігація (рис. 4.6).

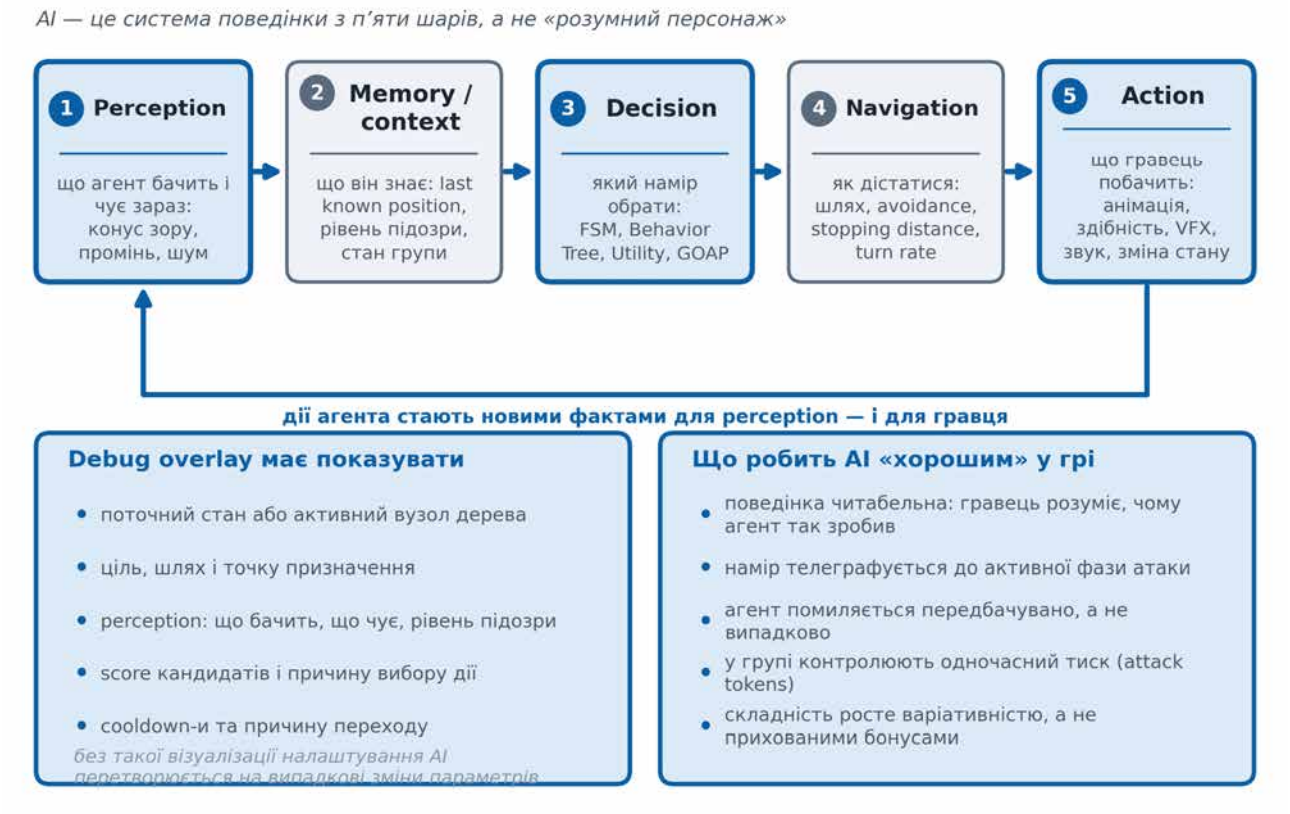


Рис. 4.6. Шари ігрового AI: perception - decision - navigation - action - feedback

4.3.2 FSM, Behavior Tree, Utility AI та GOAP

Finite State Machine добре підходить для невеликої кількості чітких режимів: Idle, Patrol, Alert, Chase, Attack, Stunned, Dead. Проблема виникає, коли кількість переходів зростає квадратично або умови дублюються. Тоді варто застосовувати hierarchical FSM, окремі substates або інший метод прийняття рішень.

Behavior Tree організовує поведінку як дерево selector/sequence/decorator/task. Він зручний для пріоритетів і повторного використання вузлів, добре візуалізується та підтримує live debugging. Водночас великий tree може перетворитися на складний граф без чіткої ownership і правил даних.

Utility AI обчислює оцінку для кількох кандидатних дій на основі параметрів: відстань, здоров’я, наявність укриття, небезпека, роль, cooldown. Метод створює плавну адаптивну поведінку, але потребує нормалізації, кривих і debug-візуалізації, інакше рішення важко пояснити.

GOAP планує послідовність дій від поточного world state до цілі. Він корисний для систем із комбінованими діями та змінним середовищем, але має вищу складність, потребує коректних preconditions/effects і контролю вартості плану. Для навчального vertical slice GOAP слід застосовувати лише тоді, коли планування є предметом перевірки (табл. 4.9; рис. 4.7; 4.8).

Таблиця 4.9

Порівняння методів прийняття рішень у ігровому AI

Метод	Сильні сторони	Обмеження	Доцільний сценарій
FSM	простота, передбачуваність, швидкий debug	state explosion, багато переходів	невеликий ворог, interaction states
Hierarchical FSM	структурує складні стани	потрібні чіткі рівні відповідальності	locomotion + combat substates
Behavior Tree	пріоритети, модульні tasks, візуальний debug	великі дерева стають нечитабельними	бойові NPC, companion behavior
Utility AI	адаптивний вибір, плавні пріоритети	складне tuning і пояснення	тактичний вибір дії
GOAP	динамічні плани, комбінації дій	вартість, складність world state	симуляції та агентні системи
Rule-based	контрольовані реакції та прості правила	конфлікти правил	директор подій, tutorial hints

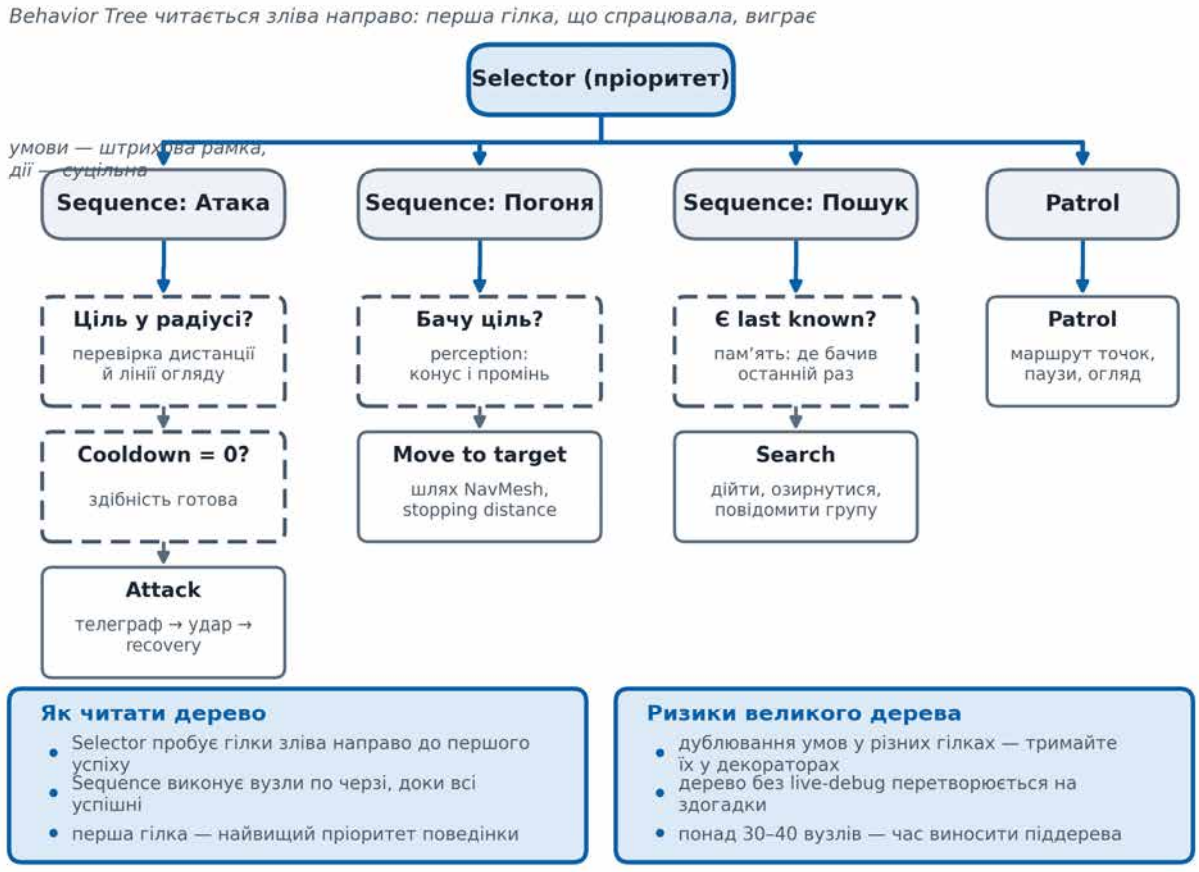


Рис. 4.7. Приклад Behavior Tree для патрулювання, переслідування й атаки

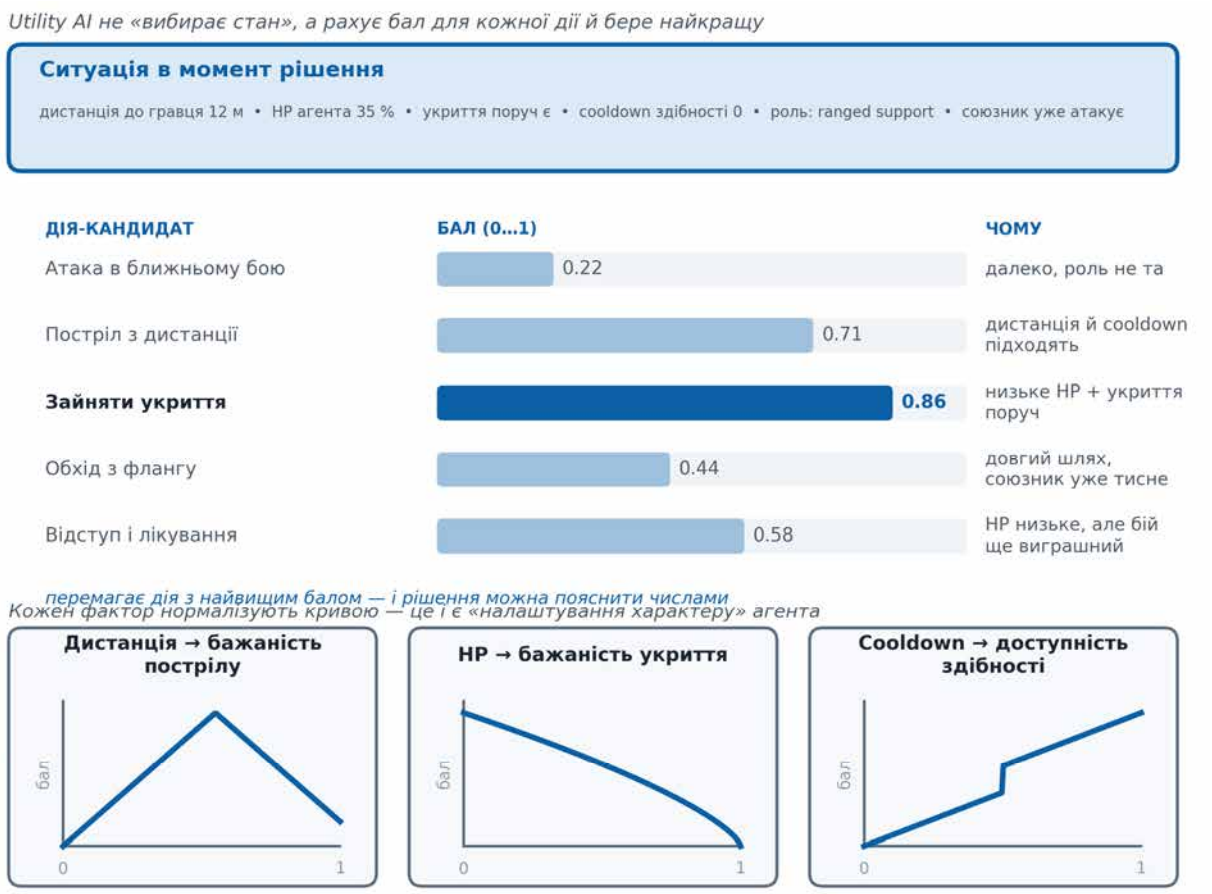


Рис. 4.8. Utility scoring для вибору бойової дії

4.3.3 Perception, пам’ять і телеграфування

Perception не повинна бути «магічною». Vision cone, line of sight, hearing radius, damage stimulus і team communication мають відповідати простору та бути поясненими гравцю. Невидимі правила створюють відчуття несправедливості, навіть якщо алгоритм математично коректний.

Пам’ять AI дозволяє відокремити «бачу зараз» від «знаю останнє місце». Після втрати цілі агент може перейти до Search, перевірити last known position, комунікувати з групою, а потім повернутися до Patrol. Часи забування, рівень підозри та швидкість пошуку повинні бути data-driven.

Telegraphing поєднує анімацію, звук, VFX, UI та просторову поведінку. Ворог повідомляє намір до active frame; гравець отримує час прочитати сигнал і прийняти рішення. Складність краще збільшувати варіативністю, комбінаціями та меншим вікном реакції, а не приховуванням правил (табл. 4.10).

Таблиця 4.10

Модель perception та реакцій AI

Stimulus	Параметри	Стан/реакція	Debug
Vision	кут, дальність, line of sight, освітлення	Alert/Chase	cone + ray + detected target
Hearing	радіус, гучність, occlusion	Investigate	сфера та marker джерела
Damage	джерело, напрямок, величина	Hit/Stagger/Retreat	лог події й вектор

Stimulus	Параметри	Стан/реакція	Debug
Ally message	тип загрози, позиція, час	Group Alert	network/debug channel
Environment	укриття, hazard, objective	Reposition/Use Object	nav annotations

4.3.4 Навігація, steering і групова поведінка

NavMesh або інший граф навігації відповідає лише на питання, де агент може пройти. Якість поведінки також залежить від вибору цілі, avoidance, stopping distance, turn rate, animation sync і взаємодії з динамічними перешкодами. Найкоротший шлях не завжди є найкращим: агент може враховувати небезпеку, укриття, видимість і роль.

У груповому бою доцільно контролювати concurrency. Attack tokens або slots обмежують кількість одночасних атак, formation/spacing зменшує скупчення, а role manager розподіляє pressure, ranged support і flanking. Це створює читабельний ритм і запобігає ситуації, коли всі вороги діють однаково.

Debug overlay має показувати path, target, current state/task, perception, score кандидатів, cooldowns і причину переходу. Без такої візуалізації налаштування AI перетворюється на випадкові зміни параметрів (рис. 4.9).

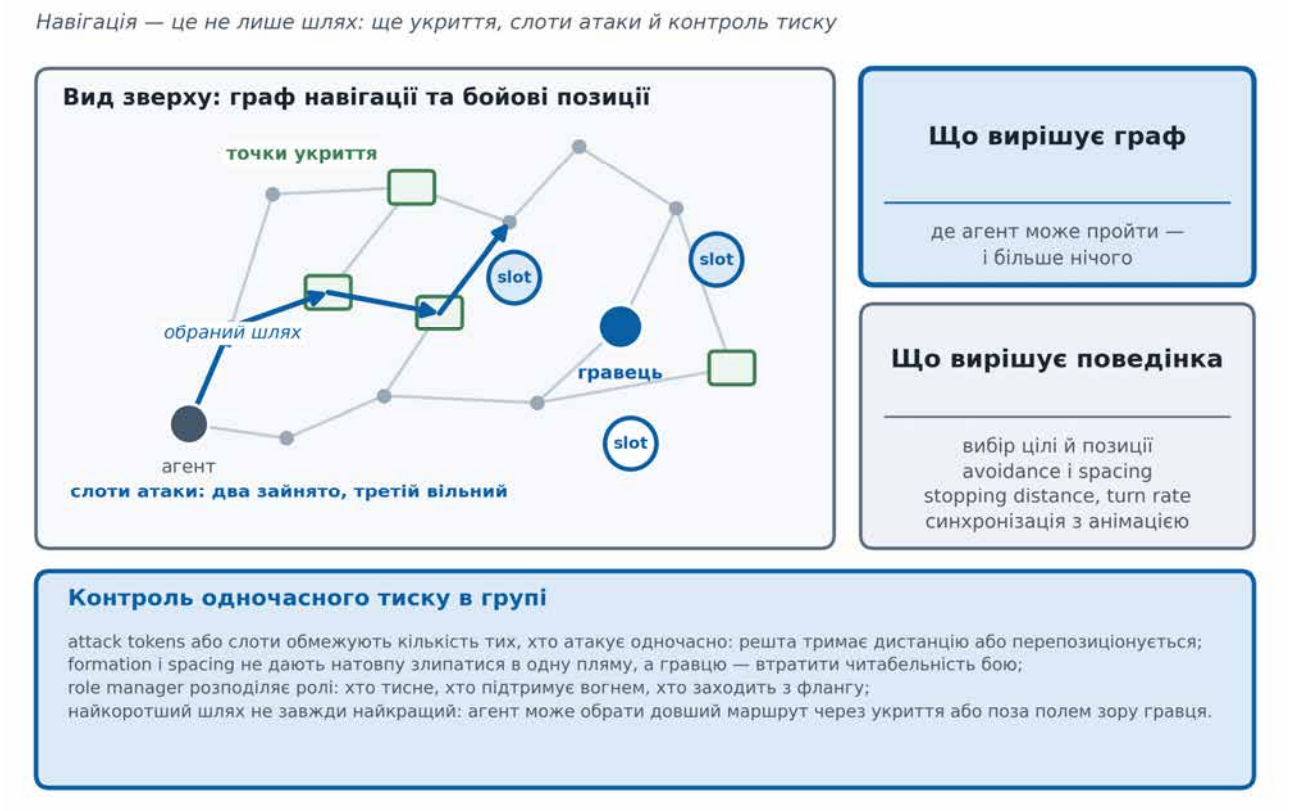


Рис. 4.9. Навігаційний граф, точки укриття та attack slots

4.4.1 Data-driven tuning та тестова арена

Параметри AI необхідно зберігати у конфігураційних assets або таблицях: швидкість, радіуси perception, час reaction, aggression, cooldown, accuracy, health

thresholds, utility curves. Це дозволяє дизайнеру створювати архетипи без зміни коду та проводити A/B тести.

Тестова арена повинна ізолювати поведінку: прямий коридор для detection, кімната з укриттями, поворот і втрата line of sight, вузький прохід, група з кількох ролей, hazard, недоступна ціль. Для кожного сценарію фіксують очікуваний стан і метрику.

Production-ready AI для vertical slice має стабільний state flow, не застрягає, не проходить крізь геометрію, не бачить крізь стіни, телеграфує атаки, коректно реагує на смерть/disable, має fallback при недоступній цілі та дає розробнику повний debug.

Практичний кейс «AI Test Arena»:

Реалізуйте одного агента з perception, FSM або Behavior Tree, трьома бойовими діями та debug overlay. Додайте дві data-driven конфігурації поведінки. Проведіть шість сценаріїв, зафіксуйте state transitions, помилки навігації, час реакції та результат tuning.

Тема 4.4. Контроль версій, налагодження та build pipeline

Мета теми: навчити організовувати безпечну командну роботу з проєктом, великими асетами й збірками, фіксувати дефекти, відтворювати проблеми та формувати повторюваний pipeline від commit до тестового build.

4.4.2 VCS як інфраструктура проєкту

Система контролю версій зберігає історію змін, авторство, можливість повернення та паралельну роботу. Для Unity-проєктів часто достатньо Git із Git LFS, якщо команда правильно налаштувала.gitignore, текстову серіалізацію assets і правила сцен/prefabs. Для великих Unreal-проєктів із численними binary assets доцільним може бути Perforce з file locking.

Найважливішою є не назва інструмента, а дисципліна: короткі змістовні commits, одна логічна зміна, відсутність тимчасових файлів, перевірка build перед merge, явна ownership великих сцен і регулярне резервне копіювання remote repository.

Гілкова стратегія для навчальної команди має бути простою. Main містить демонстраційно стабільний стан; короткі feature branches використовуються для окремих задач; integration/develop потрібна лише тоді, коли команда справді підтримує її. Довгоживучі гілки без регулярного sync створюють дорогі конфлікти (табл. 4.11) (рис. 4.10).

Таблиця 4.11

Правила зберігання файлів у VCS

Об’єкт	Рекомендація	Типова помилка
Код і конфіги	зберігати як текст, форматувати перед commit	великі commits без пояснення
Текстури/моделі/аудіо	Git LFS або Perforce; зберігати source окремо від runtime	binary files у звичайному Git
Сцени/рівні	ownership, additive scenes, короткі блокування	одночасне редагування монолітної сцени
Generated/Library/Intermediate	виключити через ignore	репозиторій містить кеші
Builds	артефакти CI/release storage, не commit	змішування source і build
Secrets	environment/secret storage	ключі та паролі в репозиторії

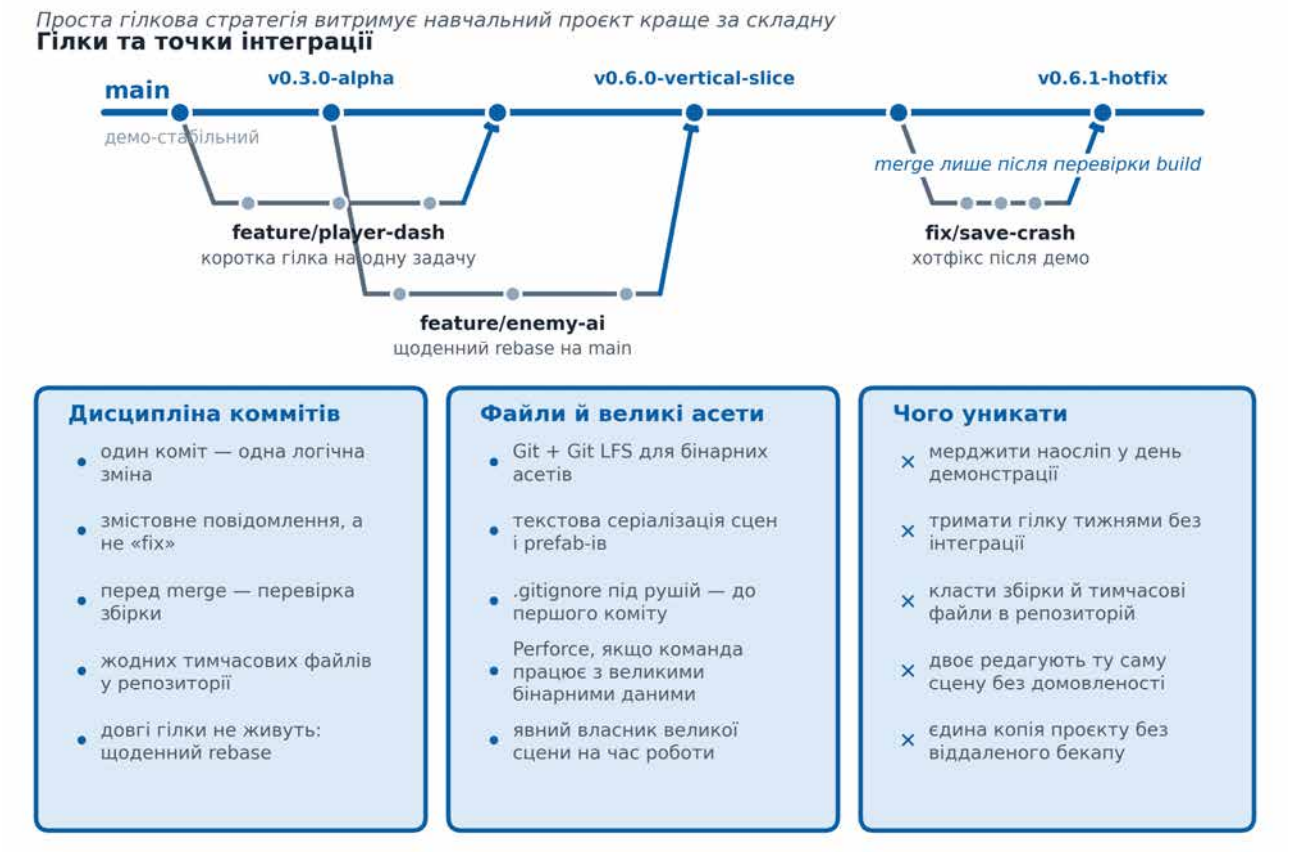


Рис. 4.10. Рекомендований Git/Perforce workflow для студентської команди

4.4.3 Неймінг, структура папок і change control

Структура папок повинна відображати відповідальність і runtime-призначення: /Art, /Audio, /UI, /Gameplay, /Levels, /Data, /Tests, /ThirdParty. У великих проєктах корисно організовувати assets за feature або модулем, щоб зменшити перехресні залежності.

Неймінг має дозволяти знайти асет без відкриття: CH_Player_Rig, ENV_Lab_Wall_A, MAT_Metal_Rough_01, SFX_UI_Confirm_01, BP_Interactable_Door. Важливо не копіювати бездумно чужі префікси, а створити короткий style guide для власної команди.

Зміни параметрів, що впливають на баланс або UX, слід супроводжувати коротким записом: гіпотеза, старе значення, нове значення, build і результат тесту. Це допомагає відрізнити обґрунтовану ітерацію від випадкового tuning (табл. 4.12; 4.13; 4.14; рис. 4.11).

4.4.4 Debugging: від відтворення до перевірки виправлення

Таблиця 4.12

Шаблон звіту про дефект

Поле bug report	**Приклад**
Заголовок	[UI][Gamepad] Confirm не працює після remap
Build/commit	0.6.12 / a1b2c3d
Платформа	Windows 11, Xbox Controller
Передумови	змінити binding Confirm у Settings
Кроки	1) відкрити меню; 2) remap; 3) повернутися; 4) натиснути Confirm
Очікувано	дія виконується з новим binding
Фактично	prompt оновлено, але подія не викликається
Частота/Severity	5/5, Major
Докази	video, input log, player.log

Таблиця 4.13

Шаблон звіту про дефект

Поле bug report	**Приклад**
Заголовок	[UI][Gamepad] Confirm не працює після remap
Build/commit	0.6.12 / a1b2c3d
Платформа	Windows 11, Xbox Controller
Передумови	змінити binding Confirm у Settings
Кроки	1) відкрити меню; 2) remap; 3) повернутися; 4) натиснути Confirm
Очікувано	дія виконується з новим binding
Фактично	prompt оновлено, але подія не викликається
Частота/Severity	5/5, Major
Докази	video, input log, player.log

Таблиця 4.14

Шаблон звіту про дефект

Поле bug report	Приклад
Заголовок	[UI][Gamepad] Confirm не працює після remap
Build/commit	0.6.12 / a1b2c3d
Платформа	Windows 11, Xbox Controller
Передумови	змінити binding Confirm у Settings
Кроки	1) відкрити меню; 2) remap; 3) повернутися; 4) натиснути Confirm
Очікувано	дія виконується з новим binding
Фактично	prompt оновлено, але подія не викликається
Частота/Severity	5/5, Major
Докази	video, input log, player.log



Рис. 4.11. Debugging loop: reproduce - isolate - fix - verify - prevent

4.4.5 Build pipeline і автоматизація

Build повинен бути повторюваним. Команда фіксує версію рушія, packages/plugins, цільову платформу, build configuration, сцену запуску, scripting backend, quality preset і спосіб пакування даних. Ручні «секретні» кроки одного учасника створюють ризик перед демонстрацією.

Мінімальний CI pipeline може виконувати checkout, перевірку структури, compile, unit/edit tests, build, архівацію та публікацію артефакту. Навіть без

повного CI корисно мати build script і checklist, що запускаються однаково на кожному milestone.

Versioning доцільно поєднувати з release notes. Для навчального проєкту достатньо major.minor.patch або milestone-build: 0.3.0-alpha, 0.6.0-vertical-slice, 0.6.1-hotfix. Release notes містять нові функції, зміни, виправлення, відомі проблеми й інструкції запуску (табл. 4.15; рис. 4.12).

Таблиця 4.15

Мінімальний build pipeline

Етап pipeline	Перевірка	Вихід
Checkout/Sync	правильний branch/tag, LFS/assets	чистий workspace
Validate	missing references, naming, licenses, configs	validation report
Compile/Test	errors, unit/edit/play tests	test results
Build	target platform, quality, scenes, version	executable/package
Smoke test	launch, input, save, critical loop	pass/fail
Package	README, credits, licenses, release notes	delivery archive
Publish	artifact storage/itch.io/internal channel	download link + checksum



Рис. 4.12. Pipeline від commit до тестового build

Практичний кейс «One-Click Build»:

Налаштуйте VCS, ignore/LFS, коротку branch policy та скрипт збірки. Створіть build із номером версії, README, release notes і credits. Зареєструйте один дефект, відтворіть, виправте, виконайте regression check і зафіксуйте commit.

Тема 4.5. Performance & Optimization: profiling, budgets, LOD, instancing та latency

Мета теми: надати студентам практичний, production-орієнтований підхід до профілювання та оптимізації ігрових прототипів у Unity або Unreal Engine 5: навчити визначати вузькі місця (bottlenecks) CPU/GPU/Memory, працювати з performance budgets, зменшувати навантаження рендеру (LOD, instancing, occlusion, shadow budgets), контролювати shader variants, VFX бюджети, стрімінг/пам'ять, і забезпечувати стабільність FPS та input responsiveness. Результат – 2 підтверджені оптимізації («measure -> fix -> verify») у власному vertical slice.

Очікувані результати навчання

• розрізняти CPU-bound vs GPU-bound сценарії та визначати bottleneck;
• будувати performance budget під цільову платформу (PC/low-end);
• користуватися профайлерами Unity/UE (CPU, GPU, memory) та читати звіти;
• знаходити «топ-2» джерела лагів/спайків і пріоритезувати їх;
• застосовувати LOD, instancing, batching та обмеження draw calls;
• керувати тінями, світлом, post-FX та прозорістю (overdraw);
• зменшувати shader variants та вартість матеріалів/шейдерів;
• оптимізувати VFX (particle counts, collision, trails, update rate);
• контролювати пам'ять, стрімінг, розмір текстур та імпорт-налаштування;
• документувати performance pass та доводити результат метриками «до/після».

Ключові терміни: Bottleneck, CPU-bound, GPU-bound, frame time, ms/frame, FPS, profiling, spike, GC (garbage collection), draw calls, batching, instancing, LOD, occlusion culling, shadow cascades, overdraw, shader variants, material complexity, particle budget, update rate, tick, memory footprint, streaming, texture compression, mipmaps, latency, input responsiveness.

4.5.1 Performance mindset: «frame time» замість «FPS»

Оптимізацію робимо через frame time (мс/кадр), бо FPS «маскує» спайки (рис. 4.13).

60 FPS ≈ 16.67 ms/frame

30 FPS ≈ 33.33 ms/frame

120 FPS ≈ 8.33 ms/frame (табл. 4.16)

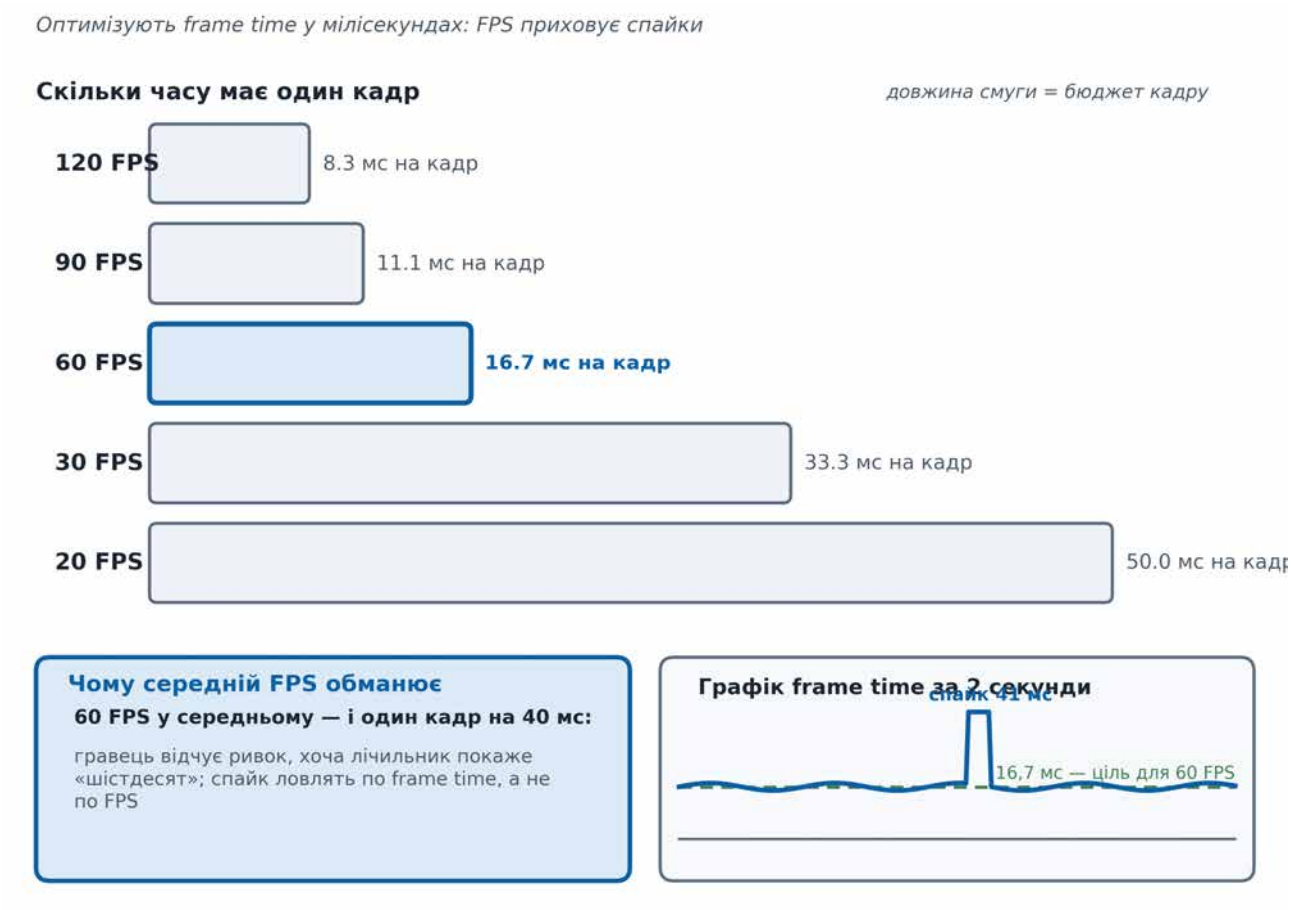


Рис. 4.13. Конверсія

Таблиця 4.16

Що важливіше

Метрика	Чому
Average frame time	загальний комфорт
1% low / spikes	«рваність» і відчуття лагу
Input latency	game feel (з теми 1)

4.5.2 Bottlenecks: CPU vs GPU vs Memory (як швидко визначити)

На першому кроці необхідно зрозуміти, що саме обмежує

CPU-bound: багато логіки, physics, AI, tick/update, GC.

GPU-bound: важкий рендер, тіні, post-FX, overdraw, шейдери.

Memory-bound: стрімінг, своп, великі текстури, підвантаження (табл. 4.17) (рис. 4.14).

Таблиця 4.17

Симптоми bottleneck

Тип	Симптом	Швидкий тест
CPU-bound	FPS падає навіть на low графіці	зменшити resolution -> майже без ефекту
GPU-bound	FPS росте при lower resolution	зменшити resolution -> помітний приріст
Memory/Streaming	підфризи при вході в зону	вимкнути стрімінг/зменшити текстури

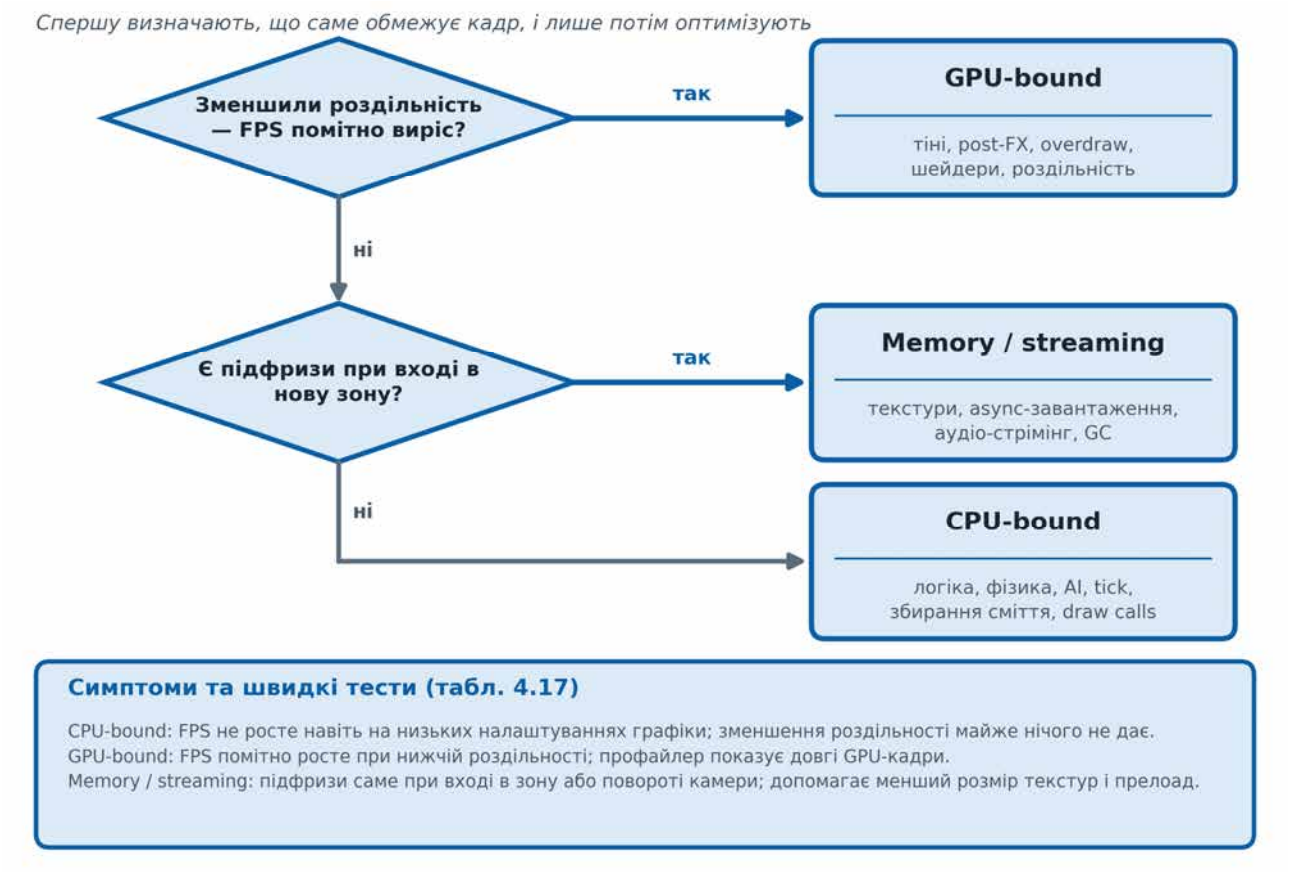


Рис. 4.14. Decision flow

Lower resolution helps a lot? -> GPU-bound
No big change? -> CPU-bound or memory
Spikes on loading? -> streaming/memory

Performance budgets: встановлюємо цілі для оптимізації

Без визначення бюджету оптимізація стає хаотичною. Для навчального vertical slice задаємо

- ціль FPS (наприклад 60 на mid PC),
- бюджет кадру 16.67 ms,
- умовний розподіл: CPU 8 ms, GPU 8 ms (або під ваш кейс) (табл. 4.18).

Таблиця 4.18

Приклад бюджетів (навчальний стандарт)

Підсистема	Бюджет (60 FPS)	Коментар
CPU gameplay/logic	3-5 ms	без зайвого tick
Physics/AI	1-3 ms	обмежити частоту
Rendering CPU	1-3 ms	draw calls, culling
GPU total	8-12 ms	світло/тіні/пост/VFX

Профайлінг у Unity/UE: що міряти і як читати

Unity: Profiler (CPU/GPU/Memory), Frame Debugger, Profile Analyzer.
UE5: Stat Unit, Stat GPU, Unreal Insights, Session Frontend (табл. 4.19).

Таблиця 4.19

Мінімальний набір профайлінгу

Що	Unity	UE5	Навіщо
CPU frame	Profiler CPU	Stat Unit/Insights	знайти update/tick/GC
GPU frame	Profiler GPU	Stat GPU	знайти тіні/post/VFX
Draw calls	Frame Debugger	RenderDoc/Stats	batching/instancing
Memory	Profiler Memory	Insights/Memory	фризи/стрімінг

4.5.5 Rendering fundamentals: draw calls, batching, instancing, occlusion

Оптимізація рендеру часто дає «найшвидший виграш»

- зменшити draw calls (батчінг, instancing),
- LOD (менше полігонів на дистанції),
- occlusion culling (не рендерити невидиме),
- обмежити тіні (shadow distance, cascades) (табл. 4.20).

Таблиця 4.20

Швидкі вираші в рендері

Техніка	Дає	Ризик
Instancing	менше draw calls	матеріали мають бути сумісні
LOD	менше GPU cost	«поп» без fade
Occlusion	менше overdraw	неправильні bounds
Shadow budget	великий FPS boost	«плоска» картинка, якщо перебор

4.5.6 Lighting & Post-FX optimization: тіні, GI, bloom, volumetrics

Повторюємо головне з попередньої теми, але з позиції бюджету, а саме

• тіні – дорогі (особливо високі каскади та дистанція);
• volumetrics – дуже дорогі;
• постефекти - «ножиці» по GPU (табл. 4.21).

Таблиця 4.21

Дорогі елементи та типові фікси

Елемент	Симптом	Фікс
Shadow cascades	GPU spikes	зменшити cascades/distance
Volumetric fog	GPU падає	локалізувати/знизити quality
SSR/Reflections	шум/вартість	probes/captures/нижче якість
Bloom/DOF	мило + cost	clamp intensity/disable in gameplay

4.5.7 Shaders & Materials: instruction count, variants, texture cost

Просунуті шейдери з теми № 10 мають бюджет

• менше варіантів (keywords),
• менше дорогих нодів (noise, loops),
• обмежити прозорість,
• компресія текстур + mipmaps (табл. 4.22).

Таблиця 4.22

Типові проблеми шейдерів

Проблема	Симптом	Рішення
Shader variants	довга збірка/пам’ять	мінімізувати keywords
Важкий noise	GPU cost	LUT/простий noise
Прозорі матеріали	overdraw	cutout/обмежити площу
Великі текстури	memory	compression + mipmaps

4.5.8 VFX budgets: частинки, колізії, trails, update rate

VFX є частим винуватцем «каші» і падіння FPS, а саме

- particle count,
- collision на частинках,
- trails,
- висока частота оновлення (табл. 4.23) (рис. 4.15).

Таблиця 4.23

Бюджет VFX (навчально)

Параметр	Рекомендація	Чому
Particles per effect	50-500	залежить від стилю
Collision	лише де критично	дорого
Overdraw	мінімізувати	GPU bottleneck
Lifetime	коротший краще	менше накопичення



Рис. 4.15. VFX бюджетний принцип

Readability first -> then cost -> then polish

4.5.9 Memory & Streaming: текстури, аудіо, асети, підвантаження

Підфризи часто виникають через

- великих текстур без mipmaps,
- надто важких мешів,
- неадекватного стрімінгу,
- GC (у Unity) через алокації (табл. 4.24).

Таблиця 4.24

«Невидимі» джерела фризів

Джерело	Симптом	Фікс
Texture streaming	hitch при повороті	lower res, mipmaps
Async loading	паузи	preload ключового
GC allocations	spikes кожні N сек	кешування, object pooling
Audio streaming	stutter	preload SFX, stream music

4.5.10 Optimization loop: «measure -> fix -> verify» + документація

Оптимізація без розуміння «до/після» - не оптимізація. Стандарт це

- замір до (скрін профайлера),
- 1 конкретний фікс,
- замір після,
- висновок: що змінилось і чому (рис. 4.16).

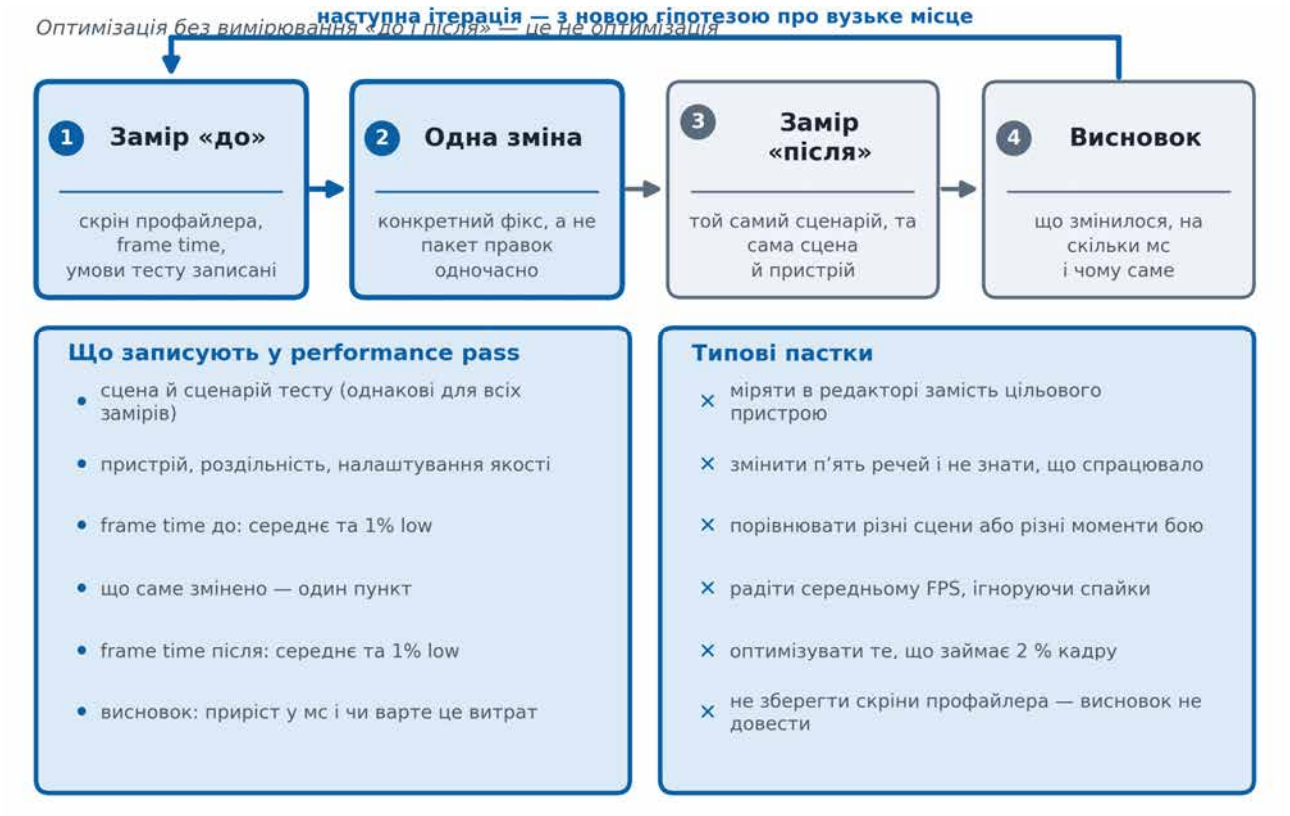


Рис. 4.16. Цикл

Baseline -> Identify top cost -> Fix -> Verify -> Regression check (табл. 4.25)

Таблиця 4.25

Шаблон звіту (коротко)

Пункт	**Вміст**
Baseline	ms/frame, топ-2 cost
Fix 1	що зробили + результат
Fix 2	що зробили + результат
Regression	чи не зламали якість/UX
Next	1-2 ідеї на майбутнє

Питання для самоперевірки

1. Чому frame time важливіший за FPS?
2. Як швидко відрізнити CPU-bound від GPU-bound?
3. Що таке performance budget і навіщо він потрібен?
4. Які 4 основні інструменти профайлінгу у Unity/UE?
5. Що таке draw call і як instancing допомагає?
6. Чому тіні часто «з'їдають» GPU?
7. Як shader variants впливають на build та memory?
8. Що таке overdraw і чому прозорість небезпечна?
9. Назвіть 4 способи оптимізації VFX.
10. Які типові причини «hitch» при вході в зону?
11. Чому GC allocations викликають спайки (Unity) і як це зменшити?
12. Опишіть цикл «measure -> fix -> verify» на прикладі двох фіксів.

Практичний кейс (час виконання: 120-180 хв)

«Performance Pass: Fix Top-2 Spikes»

Мета: виконати повний оптимізаційний прохід для вашого vertical slice/сцени та довести 2 підтверджені покращення метриками.

Сценарії – A/B/C

A (Rendering-heavy): багато світла/тіней + постефекти + VFX -> GPU-bound.

B (Logic/AI-heavy): багато агентів/physics/update -> CPU-bound.

C (Streaming-heavy): підвантаження/великі текстури -> hitch spikes.

Кроки виконання

Baseline профайл (5-10 хв):

• замірте середній ms/frame і 1-2 спайки;

• визначте bottleneck (CPU/GPU/Memory).

• Вибір Top-2 проблем:

• одна довготривала витрата (наприклад, тіні/overdraw),

• один spike (GC/streaming/loading).

• Fix #1 (Render або Logic) – приклади:
• shadow distance/cascades ↓;
• instancing/LOD;
• вимкнути дорогий post-FX в геймплеї;
• зменшити tick/частоту AI;
• pooling замість instantiate/destroy.
• Verify #1: повторний профайл + скрін/лог «до/після».
• Fix #2 (Spike/Memory) – приклади:
• mipmaps + compression;
• preload ключового контенту;
• прибрати алокації;
• зменшити VFX collision/particles.
• Verify #2: повторний профайл, порівняння.
• Regression check: чи не впала читабельність/якість (на 1-2 скрінах).
• Документація: короткий performance report.

Що потрібно здати

Проєкт Unity/UE5 з внесеними оптимізаціями.
Performance report (1-2 стор.): baseline -> fix #1 -> fix #2 -> verify.
2-4 скріни профайлера/статів «до/після».
Відео 30-60 сек (опц.) з демонстрацією плавності.

Критерії оцінювання виконання завдання

• коректно визначений bottleneck;
• два фікси мають метричний ефект (ms/frame, spikes);
• оптимізація не зруйнувала UX/readability;
• документування ясне і відтворюване.

Рекомендовані програми та сервіси

1. Unity: Profiler, Profile Analyzer, Frame Debugger, Memory Profiler, (опц.) Addressables.
2. Таблиці/аналіз: Google Sheets/Excel (для фіксації метрик).
3. Організація: Notion/Trello (performance backlog).

Рекомендовані джерела та матеріали

1. Unity Manual: Profiling, batching/instancing, memory, render pipeline optimization.
2. Unreal Engine Docs: Performance profiling (Insights), GPU optimization, Lumen/shadows best practices.
3. Практики «budgets» і «performance passes» з матеріалів GDC/engine talks.
4. RenderDoc/PIX (advanced) – як інструменти глибокої діагностики.

Тема 4.6. Мережеві ігрові системи: авторитет, реплікація та latency compensation

Мета теми: сформувати базове системне розуміння multiplayer architecture, server authority, replication, prediction, reconciliation, interpolation, bandwidth budgets, security і тестування мережевих умов.

4.6.1 Моделі мережевої архітектури

Мережевий режим змінює дизайн механік, а не лише спосіб передачі даних. Кожна дія повинна мати джерело істини, правила авторитету, допустиму затримку, спосіб корекції та поведінку при втраті зв'язку.

Dedicated authoritative server забезпечує єдине рішення, кращий контроль cheat і масштабованість, але потребує інфраструктури та збільшує операційні витрати. Listen server простіший для невеликих сесій, але host має перевагу. Peer-to-peer зменшує серверну вартість, проте ускладнює NAT, довіру й синхронізацію.

Не всі дані мають однаковий пріоритет. Position важливий часто, але може бути приблизним; inventory або match result передається рідше, проте надійно. Cosmetic events допускають втрату, critical transactions – ні (табл. 4.26) (рис. 4.17).

Таблиця 4.26

Порівняння мережевих моделей

Модель	Переваги	Обмеження	Типові жанри/масштаб
Dedicated server	authority, anti-cheat, persistence	вартість і DevOps	competitive, MMO, service
Listen server	швидкий старт, менша вартість	host advantage, migration	co-op, small matches
Peer-to-peer	низька інфраструктурна вартість	security, NAT, desync	невеликі deterministic games
Relay	спрощення connectivity	додаткова latency/fee	session-based multiplayer
Asynchronous	не потребує одночасної присутності	конфлікти стану	turn-based, social

4.6.2 Tick, snapshots і реплікація стану

Server tick визначає частоту симуляції та надсилання стану. Високий tick зменшує часову похибку, але збільшує CPU і bandwidth. Частота simulation, snapshot і render не обов'язково однакова.

Replication може передавати повний snapshot, delta, події або команди. Стан квантують, пріоритизують за relevancy і distance, агрегують та стискають. Об'єкт, який не впливає на конкретного клієнта, не повинен витрачати bandwidth.

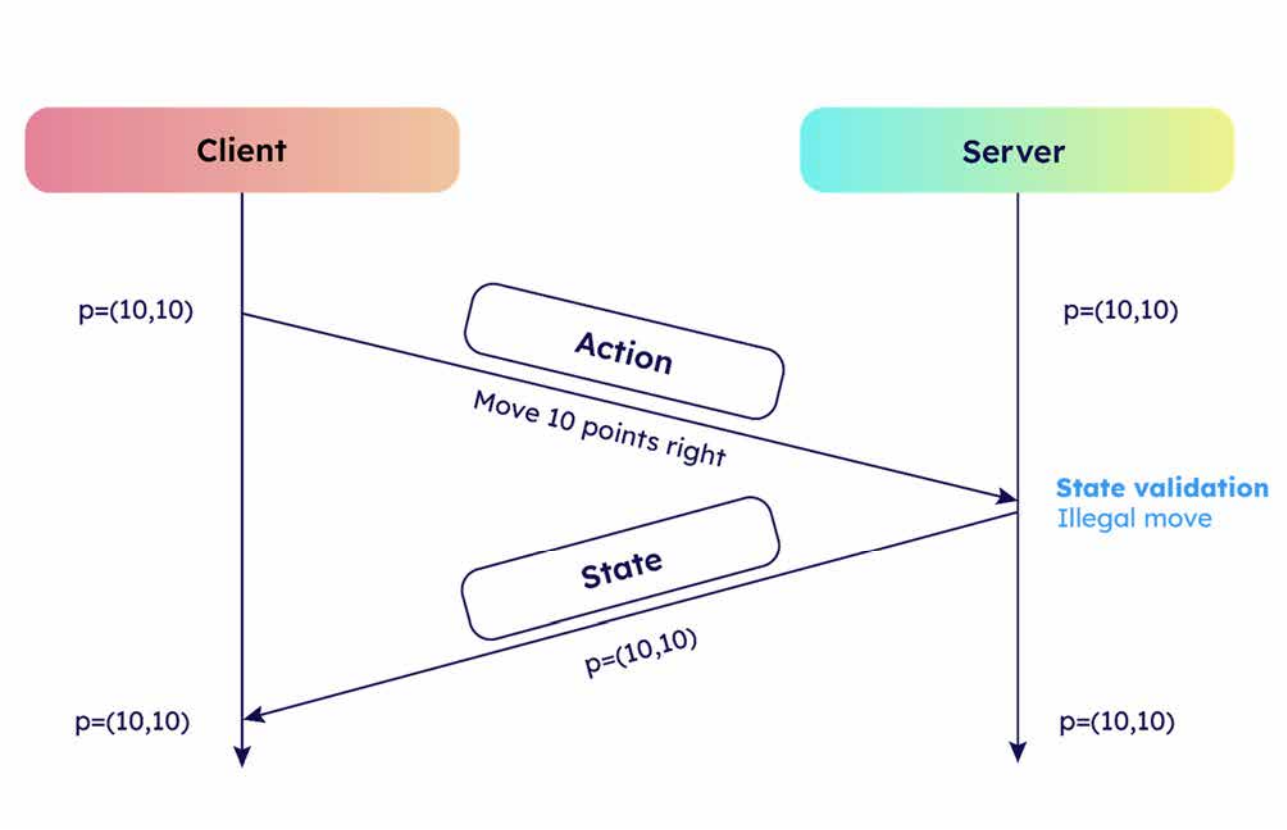


Рис. 4.17. Client-server architecture з каналами авторитету

Надійність обирають за семантикою. Reliable ordered потрібний для критичної послідовності, але head-of-line blocking може затримати наступні повідомлення. Unreliable підходить для частих snapshots, які швидко застарівають (табл. 4.27; рис. 4.18).

Таблиця 4.27

Стратегія реплікації різних типів даних

Дані	Частота	Надійність	Оптимізація
Player transform	10-30 Hz	unreliable snapshots	delta, quantization, relevancy
Input commands	кожен tick / batch	sequenced	bit packing
Inventory transaction	за подією	reliable	server validation
Animation state	за зміною + params	mixed	semantic state, not raw bones
VFX/SFX event	за подією	часто unreliable	local reconstruction
Match result	одноразово	reliable/persistent	idempotent transaction

4.6.3 Prediction, reconciliation та interpolation

Без prediction гравець відчуває round-trip delay перед локальним рухом. Client-side prediction одразу застосовує власний input, а сервер пізніше підтверджує authoritative state. Якщо стани відрізняються, client виконує reconciliation: повертається до підтвердженого стану й повторює невизнані inputs.

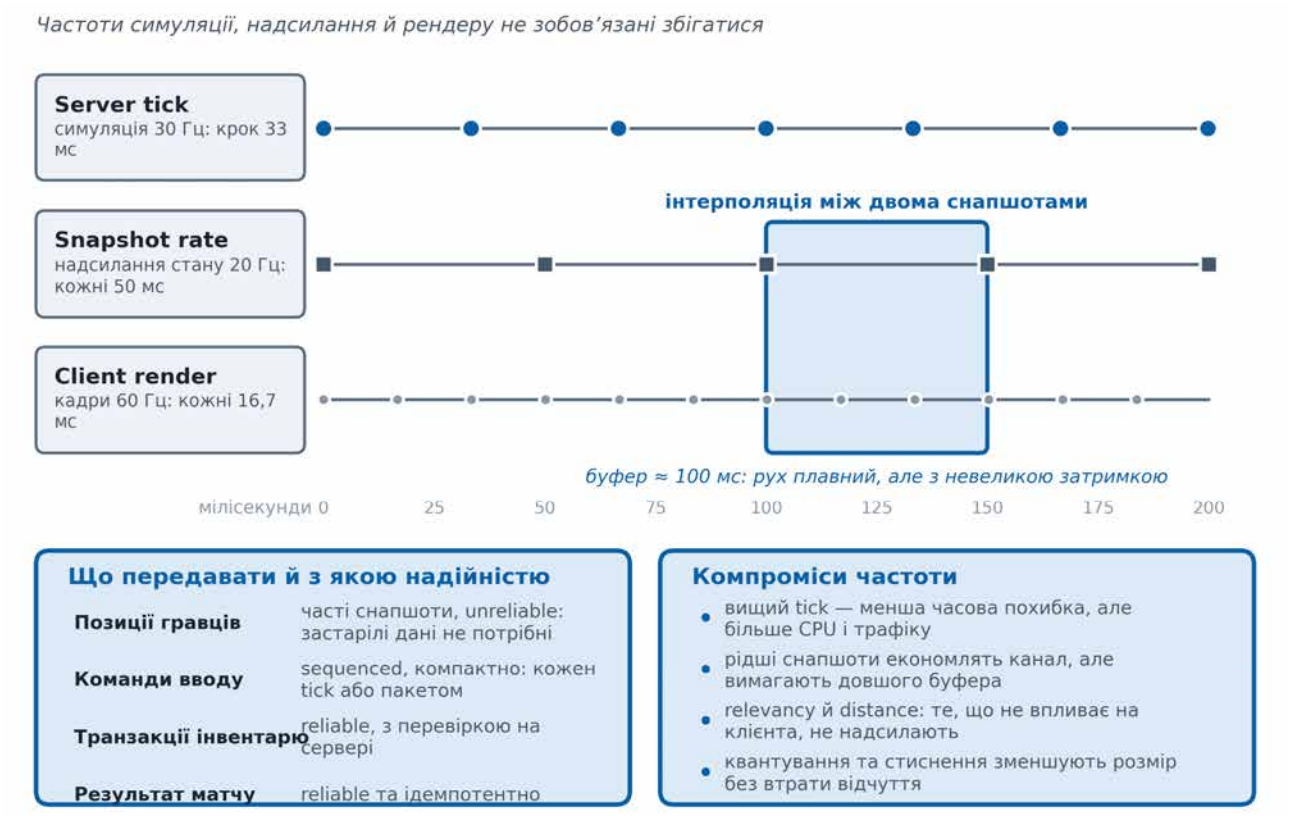


Рис. 4.18. Server tick, snapshot rate та render interpolation timeline

Remote players зазвичай відображаються через interpolation між buffered snapshots. Extrapolation прогнозує майбутній стан, але помилка зростає при різких змінах. Buffer додає візуальну затримку, зате зменшує jitter.

Correction не повинна створювати різкі teleport або порушувати collision. Невелику похибку можна згладити в presentation layer, тоді як simulation layer одразу приймає authoritative state (табл. 4.28; рис. 4.19).

Таблиця 4.28

Компенсація мережевої затримки

Техніка	Для кого	Перевага	Ризик
Client prediction	локальний player	responsive control	mis-prediction
Server reconciliation	локальний player	authoritative consistency	visible correction
Interpolation	remote entities	smooth motion	added delay
Extrapolation	remote entities	менша delay	error on turns
Lag compensation	hit validation	fairness for delayed shooter	peeker/rewind edge cases
Rollback	deterministic simulation	responsive fighting/action	strict determinism and cost

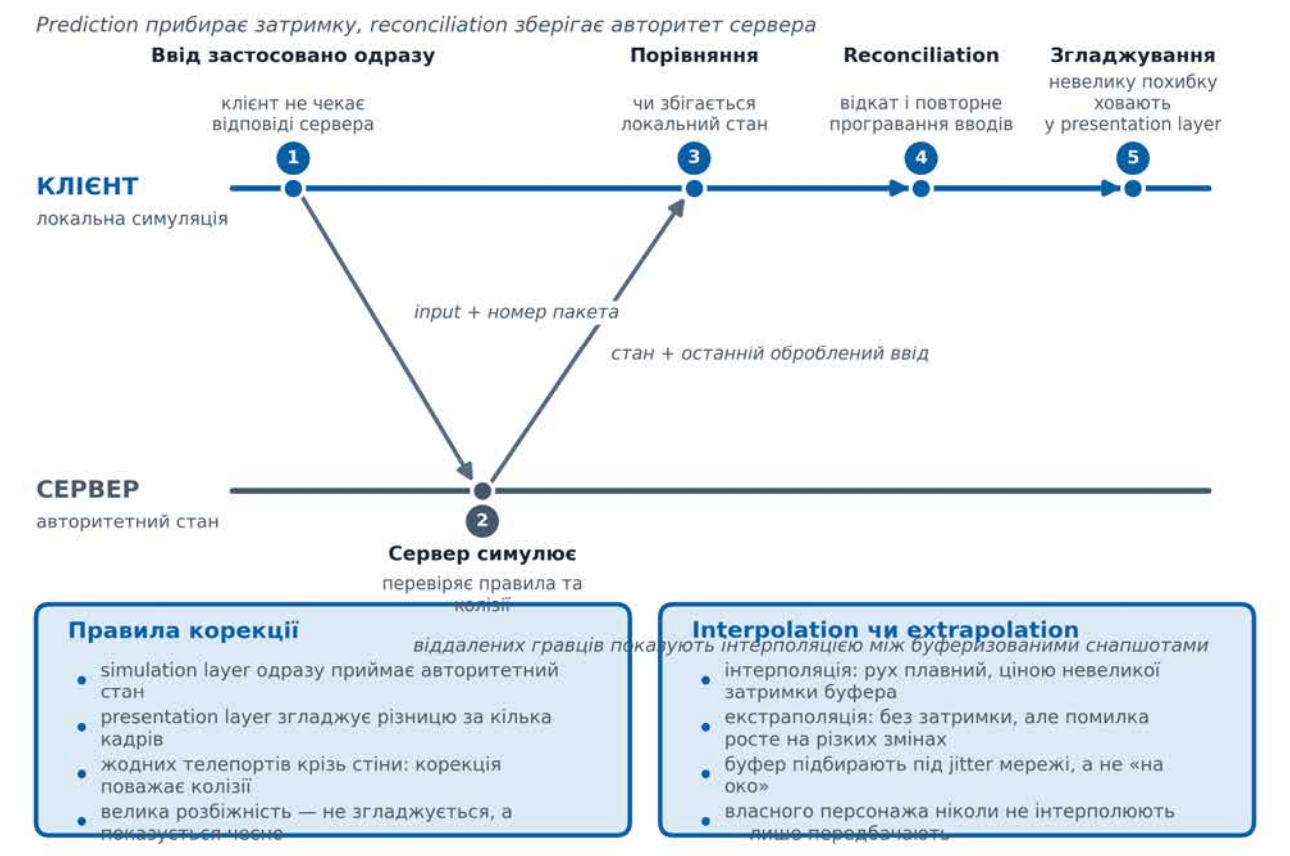


Рис. 4.19. Client prediction і server reconciliation крок за кроком

4.7.1 Gameplay authority, anti-cheat і безпечні транзакції

Клієнт не повинен повідомляти «я завдав 100 damage» як факт. Він надсилає input або request, а сервер перевіряє state, range, cooldown, line of sight, resource cost і визначає результат. Чим вища конкурентна цінність дії, тим суворіша server validation.

Навіть authoritative server не усуває всі атаки. Потрібні rate limits, sanity checks, secure authentication, encrypted transport за потреби, server logs і anomaly detection. Важливо також захистити економіку від duplicate transactions і replay.

Idempotency означає, що повторна доставка тієї самої транзакції не створює дубль. Для rewards, purchases і progression використовують unique request ID, atomic update і audit trail (табл. 4.29).

Таблиця 4.29

Приклади authoritative validation

Ризик	Server-side контроль	Додатковий evidence
Speed hack	max acceleration/speed, movement validation	movement anomaly log
Fire-rate hack	server cooldown	timestamp sequence
Wall hit	server ray/rewind validation	position history
Inventory duplication	atomic transaction, unique ID	audit log
Spam RPC	rate limiting	per-client counters
Fake score	score computed server-side	match event log

4.7.1 Мережеве тестування та UX

Multiplayer потрібно тестувати не лише на localhost. Створюють matrix із latency, jitter, packet loss, reorder, disconnect, reconnect і різною продуктивністю клієнтів. Сценарій має бути повторюваним.

UI повинен пояснювати connecting, matchmaking, timeout, migration, reconnect, desync, maintenance і queue. Мовчання системи перетворює мережеву проблему на помилку користувача.

Метрики включають RTT, packet loss, bandwidth, correction count, snapshot age, input delay, disconnect rate і match completion. Логи клієнта й сервера повинні корелюватися через session/player/request IDs (табл. 4.30).

Таблиця 4.30

Матриця мережевих умов

Профіль	Latency	Jitter	Loss	Очікуваний критерій
Local baseline	0-20 ms	< 5 ms	0%	немає corrections
Typical regional	40-80 ms	5-15 ms	< 1%	responsive gameplay
High latency	120-180 ms	10-30 ms	1-2%	зрозумілі corrections
Unstable	80-200 ms	30-60 ms	3-5%	graceful degradation
Disconnect	N/A	N/A	100%	reconnect або clean failure

Практичний кейс: replicated interaction

Реалізуйте мережеву дію з local prediction, server validation, replicated result і fail feedback. Додайте debug overlay з RTT/corrections, протестуйте п’ять network profiles і оформіть diagram authority + таблицю bandwidth/events.

Тема 4.7. Процедурна генерація контенту та інструменти для команди

Мета теми: пояснити procedural content generation як керовану систему правил, constraints, seeds, validation і авторського контролю; навчити проєктувати editor tools, які прискорюють production без втрати якості.

4.7.1 PCG: від випадковості до керованого простору варіантів

Процедурна генерація не означає «випадково розмістити об’єкти». Якісний генератор визначає grammar, параметри, constraints, послідовність етапів, seed, validation і критерії відбракування. Випадковість лише обирає варіант у дозволеному просторі.

Runtime PCG створює варіативність під час гри, але потребує швидкої генерації, детермінізму, save/replay strategy й перевірки крайніх випадків. Offline PCG працює як авторський інструмент: генерує варіанти, які дизайнер редагує і зберігає.

Найкращі системи поєднують automation і authorship. Генератор виконує повторювану роботу, а людина формує high-level intent, locks важливі ділянки, коригує винятки й приймає результат (табл. 4.31; рис. 4.20).

Таблиця 4.31

Методи процедурної генерації

Підхід	Що генерує	Перевага	Ризик
Noise / fields	terrain, density, masks	органічна варіативність	слабка gameplay structure
Grammar / rules	будівлі, rooms, quests	керована структура	складна authoring
Constraint solving	puzzles, layouts	гарантовані умови	дорогий search
Wave Function Collapse	tiles/patterns	локальна узгодженість	global goals не гарантовані
Search / evolutionary	levels, parameter sets	оптимізація під metrics	непередбачувана quality
Hybrid authored-PCG	варіанти навколо anchors	контроль і масштаб	tool complexity

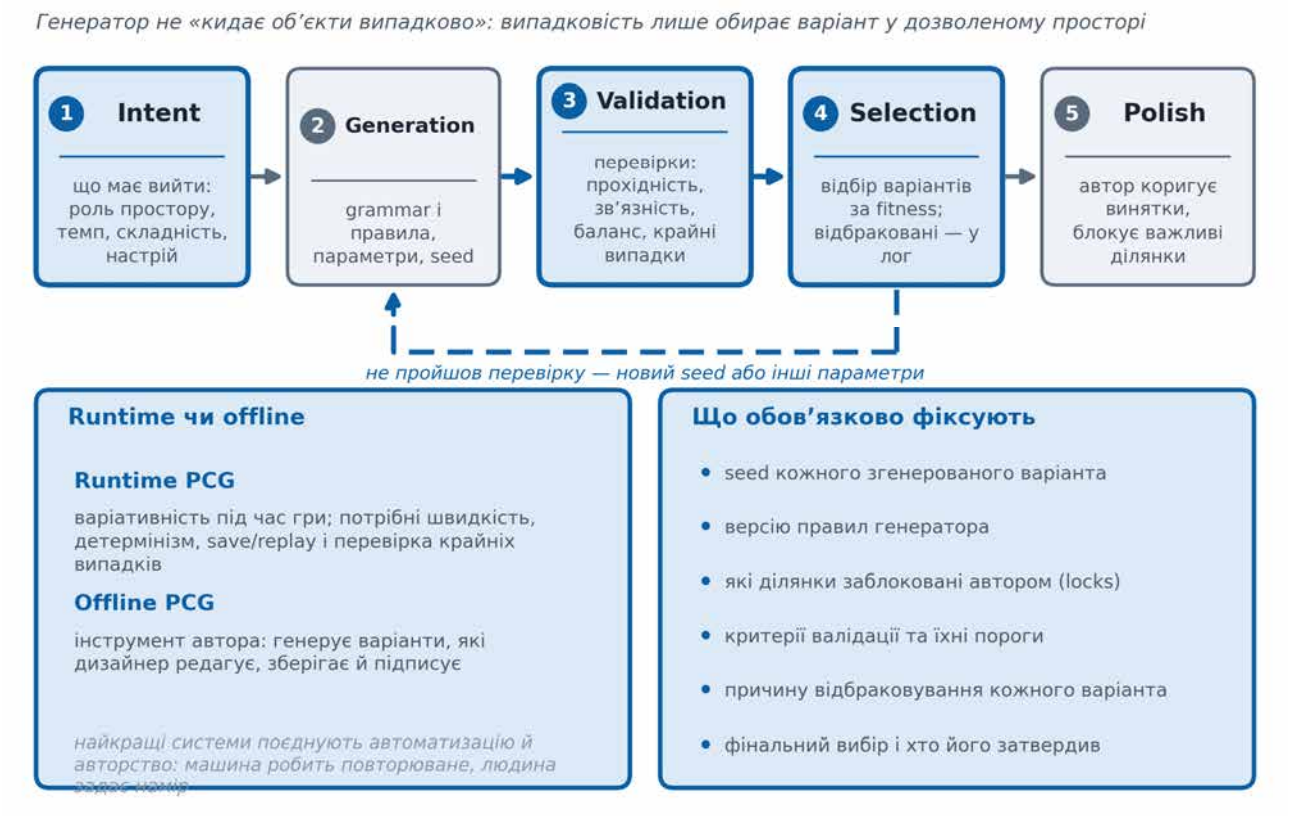


Рис. 4.20. PCG pipeline: intent - generation - validation - selection - polish

4.7.2 Seeds, determinism і reproducibility

Seed дозволяє відтворити конкретний результат. Усі random sources мають бути контрольованими, а порядок викликів – стабільним. Зміна версії алгоритму може змінити результат навіть з тим самим seed, тому build/version зберігається разом із save або test case.

Determinism важливий для debugging, multiplayer, replay і QA. Якщо генерація залежить від frame order, floating-point differences або незафіксованих external data, дефект стає майже невідтворюваним.

Корисний debug UI показує seed, stages, rejected attempts, constraints і cost. Дизайнер повинен мати змогу regenerate окрему зону, lock result і порівняти варіанти (табл. 4.32–4.35).

Таблиця 4.32

Метадані процедурного результату

Metadata	**Навіщо зберігати**
Seed	точне відтворення варіанта
Generator version	сумісність після змін алгоритму
Parameter set	контекст і difficulty
Content database version	доступні tiles/encounters/rewards
Validation result	пояснення, чому варіант прийнято
Manual overrides	збереження авторських правок

4.7.3 Validation і fitness metrics

Таблиця 4.33

Приклади PCG validation

Constraint / metric	**Тип**	**Приклад критерію**
Reachability	hard	start -> goal існує для всіх required abilities
Critical path length	soft range	8-12 хвилин
Resource sufficiency	hard/soft	мінімум 2 способи відновлення
Encounter difficulty	soft	цільовий threat budget
Sightline	soft	landmark видно у ключових точках
Backtracking	soft max	< 20% маршруту без нової події
Performance	hard	object/triangle/VFX budget
Diversity	soft	не більше двох однакових rooms поспіль

Таблиця 4.34

Приклади PCG validation

Constraint / metric	**Тип**	**Приклад критерію**
Reachability	hard	start -> goal існує для всіх required abilities
Critical path length	soft range	8-12 хвилин
Resource sufficiency	hard/soft	мінімум 2 способи відновлення
Encounter difficulty	soft	цільовий threat budget
Sightline	soft	landmark видно у ключових точках
Backtracking	soft max	< 20% маршруту без нової події

Constraint / metric	Тип	Приклад критерію
Performance	hard	object/triangle/VFX budget
Diversity	soft	не більше двох однакових rooms поспіль

Таблиця 4.35

Приклади PCG validation

Constraint / metric	Тип	Приклад критерію
Reachability	hard	start -> goal існує для всіх required abilities
Critical path length	soft range	8-12 хвилин
Resource sufficiency	hard/soft	мінімум 2 способи відновлення
Encounter difficulty	soft	цільовий threat budget
Sightline	soft	landmark видно у ключових точках
Backtracking	soft max	< 20% маршруту без нової події
Performance	hard	object/triangle/VFX budget
Diversity	soft	не більше двох однакових rooms поспіль

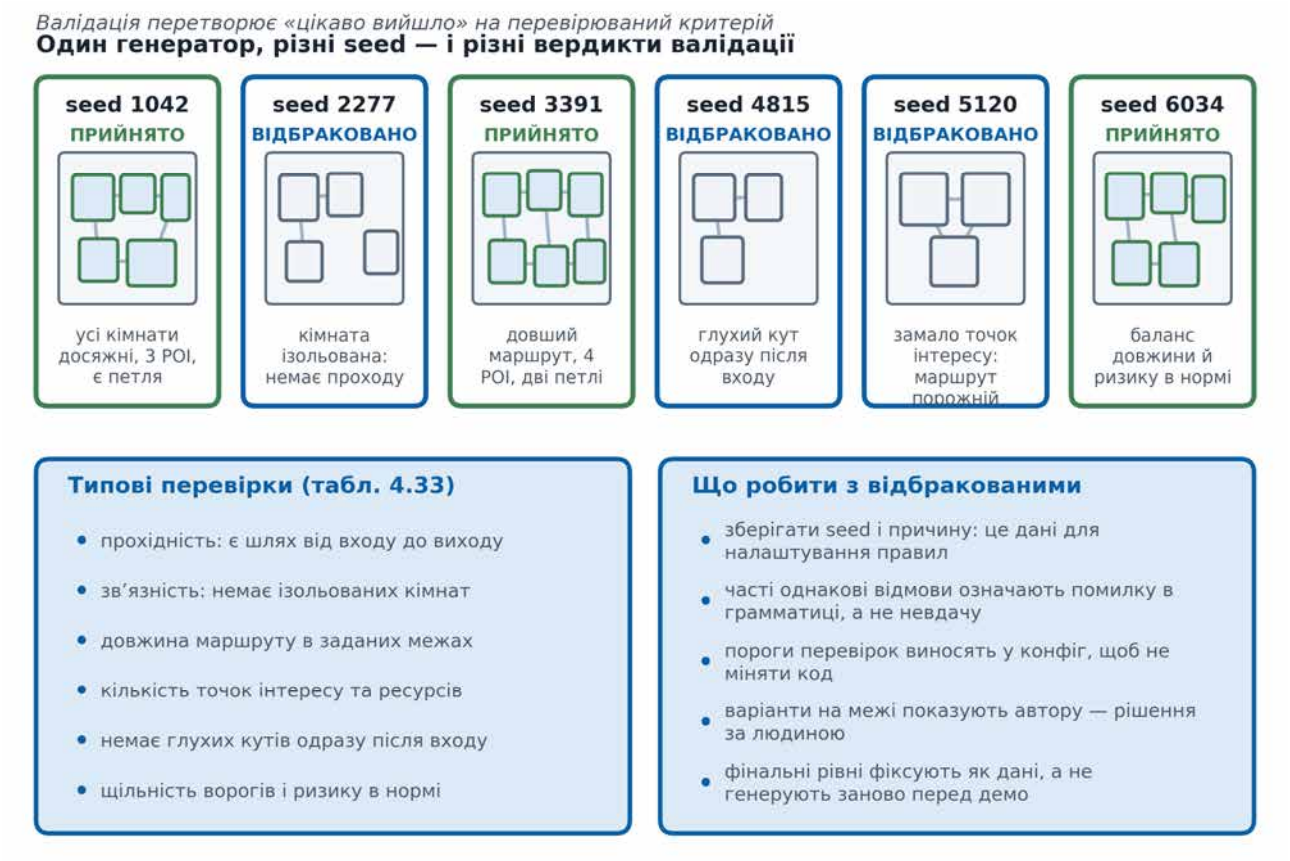


Рис. 4.21. Візуалізація accepted/rejected procedural layouts

5.1.1 Editor tools і UX для розробника

Внутрішній tool також є інтерфейсом. Він повинен мати зрозумілу ментальну модель, preview, undo/redo, validation, presets, safe defaults, batch

operations і повідомлення про помилки. Інструмент, який потребує постійної допомоги програміста, не масштабує production.

Tool ROI оцінюють через частоту задачі, час ручної роботи, кількість користувачів, помилки й вартість підтримки. Не кожну одноразову операцію треба автоматизувати; проте повторюваний import, placement, tagging, localization або validation часто дає великий виграш.

Зміни інструменту версіонуються. Data migration, backward compatibility і documentation є частиною DoD. Приклад сцени та коротке відео допомагають команді швидко прийняти новий workflow (табл. 4.36).

Таблиця 4.36

Мінімальні вимоги до внутрішнього інструменту

Функція tool	**Очікувана поведінка**
Preview	результат до застосування
Undo/redo	безпечний експеримент
Preset	повторювані стандарти
Validation	помилка + причина + шлях виправлення
Batch	обробка набору асетів
Logging	відтворення операцій і дефектів
Documentation	короткий workflow і examples
Metrics	економія часу та зменшення помилок

Практичний кейс: PCG або production tool

Створіть генератор room layout/loot/vegetation або editor tool для import/placement/validation. Додайте seed, presets, constraints, preview, undo, debug view, test set із 100 генерацій і звіт про прийняті/відхилені результати (табл. 4.37).

Питання для самоперевірки до розділу 4

1. Які ознаки має модульна архітектура гри?
2. Коли композиція краща за спадкування?
3. Як event bus зменшує зв’язність і які створює ризики?
4. Для чого використовують state, command, observer, factory та object pool?
5. Що означає data-driven design?
6. Чим ECS відрізняється від класичної об’єктної моделі?
7. Коли ECS є надмірним для проєкту?
8. Які відповідники базових сутностей існують у Unity та Unreal Engine?
9. Як відокремити gameplay logic від presentation?
10. Як організувати tunables і конфігурації без hard-coded значень?
11. Чим FSM, Behavior Tree, GOAP і utility AI відрізняються?
12. Які дані потрібні AI для сприйняття й прийняття рішення?

13. Як Git LFS або Perforce допомагають працювати з бінарними асетами?
14. Яка branching strategy доцільна для невеликої студентської команди?
15. Що повинно входити до README і release notes?
16. Як відтворити crash або nondeterministic bug?
17. Чому optimization without profiling є ризикованою?
18. Як визначити, чи гра CPU-bound або GPU-bound?
19. Які бюджети слід встановити до art/content production?
20. Як перевірити, що оптимізація не погіршила якість і gameplay?

Практичні завдання до розділу 4

Завдання 1. Модульна gameplay-система

Реалізуйте механіку через компоненти, data assets і events. Додайте state diagram, unit/functional tests, debug UI та коротке пояснення обраних patterns.

Завдання 2. AI encounter prototype

Створіть AI з perception, decision і action layers. Порівняйте FSM та Behavior Tree/utility для одного сценарію, зберіть telemetry і поясніть вибір.

Завдання 3. Performance lab

Зафіксуйте baseline на цільовому hardware, знайдіть один CPU і один GPU bottleneck, зробіть контрольовані зміни, повторіть capture та оформіть before/after звіт.

Висновки до розділу 4

1. Добра архітектура підтримує ітерацію: дозволяє змінити параметр або замінити реалізацію без каскаду випадкових наслідків. Вона має бути зрозумілою команді, документованою через схеми й підтвердженою робочими тестами.
2. Продуктивність є частиною дизайну. Frame time, memory, loading, latency і resource budgets визначають, які механіки, ефекти та масштаби світу реально доступні на цільовій платформі.

Графічні матеріали до розділу 4

Таблиця 4.37

Перелік рекомендованих графічних матеріалів до розділу 4

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
4.1	Приклад структури ігрового рушія	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.2	Приклад застосування патернів програмування в ігровій розробці	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
4.3	Приклад візуалізації систем оптимізації в ігровій розробці	Рекомендовано вставити знімок профайлера або дашборда з виділеним bottleneck, baseline та результатом повторного вимірювання.
4.4	Ітеративний процес створення прототипу	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
4.5	Asset-to-engine – приклад Maya UE5	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.6	Шари ігрового AI: perception – decision – navigation – action – feedback	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.7	Приклад Behavior Tree для патрулювання, переслідування й атаки	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.8	Utility scoring для вибору бойової дії	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.9	Навігаційний граф, точки укриття та attack slots	Рекомендовано поєднати план зверху та перспективний greybox-кадр, позначити critical path, опційні маршрути, точки інтересу і зони ризику.
4.10	Рекомендований Git/Perforce workflow для студентської команди	Подати як swimlane-діаграму відповідальностей або карту артефактів із зазначенням власника, входів, виходів і критеріїв готовності.
4.11	Debugging loop: reproduce – isolate – fix – verify – prevent	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
4.12	Pipeline від commit до тестового build	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
4.13	Конверсія	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.14	Decision flow	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.15	VFX бюджетний принцип	Подати серію з 3-5 кадрів anticipation – impact – recovery, додати часову шкалу та короткі примітки щодо читабельності й бюджету.
4.16	Цикл	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
4.17	Client-server architecture з каналами авторитету	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.18	Server tick, snapshot rate та render interpolation timeline	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.19	Client prediction і server reconciliation крок за кроком	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
4.20	PCG pipeline: intent – generation – validation – selection – polish	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
4.21	Візуалізація accepted/rejected procedural layouts	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.

РОЗДІЛ 5. ТЕСТУВАННЯ, ВИРОБНИЦТВО, РЕЛІЗ І ПОРТФОЛІО

Мета розділу: фінальний розділ об’єднує QA, плейтест, аналітику, командний production, презентацію vertical slice, підготовку build, ліцензування та професійне представлення результатів.

Реліз — не одноразова подія, а керований процес: критерії готовності, test plan, bug triage, performance verification, credits, store requirements, backup і план підтримки. Навчальний проєкт повинен пройти спрощену, але реалістичну версію цього циклу. Особлива увага приділяється артефактам, які перетворюють роботу на портфоліо-доказ: коротка демонстрація, breakdown, метрики, postmortem, чіткий опис власної ролі та відповідальне використання стороннього контенту.

Після вивчення розділу здобувач здобуває компетентності:

- розрізняти QA, testing, playtest, usability test і analytics;
- створювати test plan, test cases, bug reports, severity/priority та regression suite;
- організовувати командні ролі, backlog, milestones, risk register і peer review;
- планувати vertical slice, capture, trailer, storyboard, shot list і pitch deck;
- оцінювати build readiness, platform requirements, licensing і credits;
- застосовувати гейміфікацію для підтримки навчальної поведінки без маніпуляції;
- оформлювати postmortem і портфоліо-кейс із доказами власного внеску;
- будувати індивідуальну кар’єрну траєкторію.

Ключові поняття: QA, playtest, test case, bug triage, production, milestone, vertical slice, trailer, pitch deck, gamification, licensing, portfolio.

Тема 5.1. Тестування, аналітика та підготовка до релізу: QA, user research, telemetry, build і publishing

Мета теми: сформувати у студентів цілісне уявлення про якість (QA) як безперервний процес у розробці ігрових застосунків, навчити планувати та проводити плейтести і користувацькі дослідження, збирати й аналізувати метрики, виконувати базовий баланс ігор, організовувати баг-трекінг та підготовку збірок (build) до демонстрації й публікації

Очікувані результати навчання

- вміти пояснювати різницю між QA, тестуванням, user research та аналітикою ігрового продукту;
- планувати тестову стратегію для прототипу (що тестуємо, як, ким і коли);
- створювати мінімальний набір артефактів QA: тест-план, чеклисти, протокол плейтесту, баг-репорти;
- налаштовувати просту телеметрію/логування подій та інтерпретувати базові метрики (наприклад, AARRR, retention, completion rate);
- проводити балансування на основі даних і спостережень (таблиці балансу, криві складності, економіка);

- виконувати тріаж помилок, визначати пріоритети та ризики релізу;
- підготувати релізний пакет: build, реліз-ноти, сторінка продукту, мінімальний маркетинговий набір і план підтримки.

Ключові терміни: QA (Quality Assurance), тест-план, тест-кейс, чеклист, регресійне тестування, smoke test, playtest, usability test, telemetry, event tracking, funnel, retention, session length, churn, AARRR, KPI, баланс (difficulty curve), економіка гри, triage, severity/priority, bug report, release candidate (RC), build pipeline, versioning, patch notes, storefront (Steam/itch.io/Google Play).

5.1.2 QA у геймдеві: навіщо, коли і як працює цикл якості

У навчальних проєктах QA часто сприймається як «перевірити в кінці». У реальному виробництві це призводить до накопичення технічного боргу та непередбачуваних зривів. Правильний підхід: QA – це процес, що супроводжує розробку від першого прототипу до релізу (рис. 5.1).

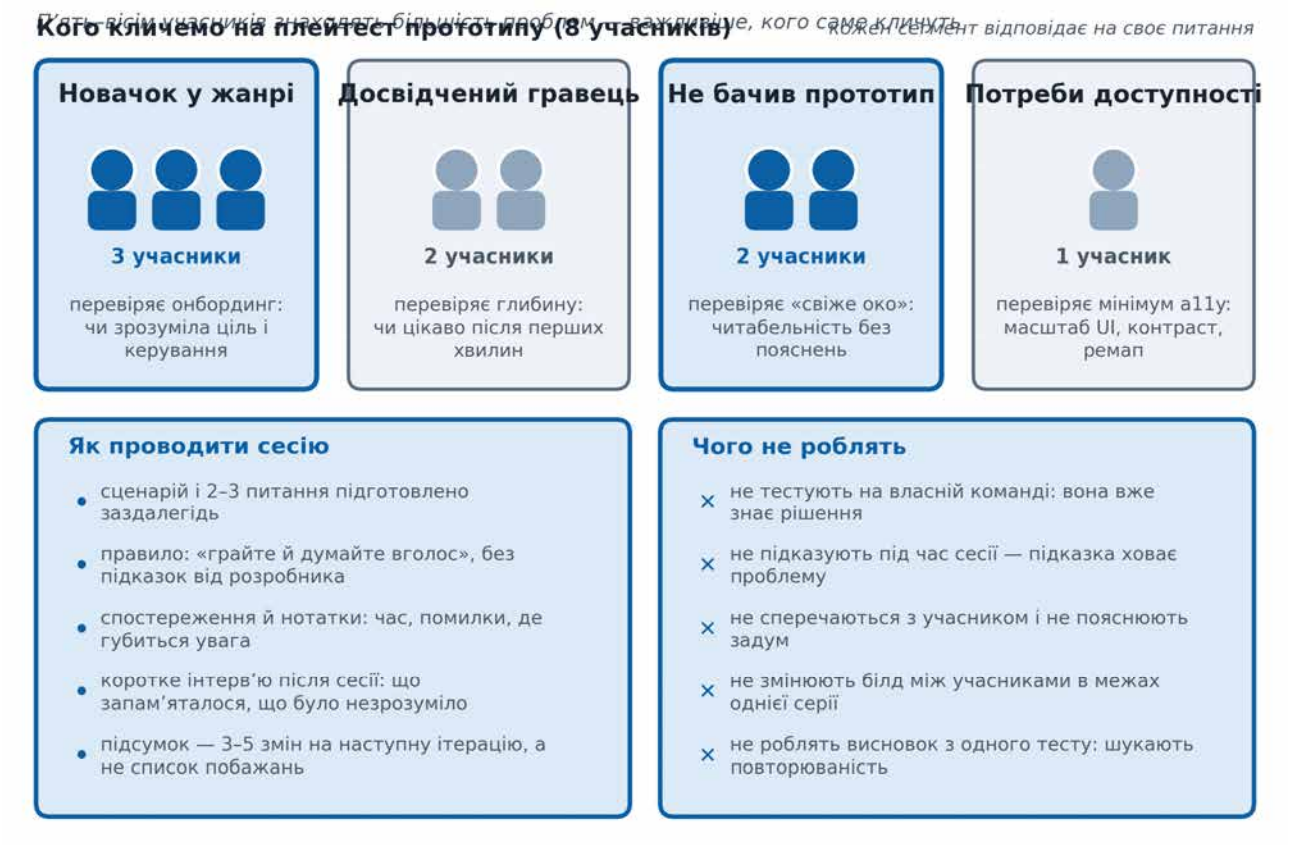


Рис. 5.1. Демонстрація таргет-групи для тестування гри

Рекомендовані практики для студентських команд

- Smoketest перед кожною демонстрацією (3-5 хв): запуск, меню, основний цикл, завершення, вихід.
- Чеклисти на рівень/механіку/інтерфейс замість «сотень тест-кейсів».
- Короткі регресійні перевірки після правок (чи не зламалося те, що працювало).
- Окрема «Test scene/map» для перевірки асетів, UI, VFX, анімації (табл. 5.1).

Таблиця 5.1

Мінімальні рівні тестування для навчального прототипу

Рівень	Мета	Приклад	Частота
Smoketest	Переконатися, що build запускається і базовий флоу працює	Меню->Геймплей->Фініш	кожен білд/ перед демо
Функціональне	Перевірити конкретні механіки	Стрибок/атака/інвентар	після змін
Usability	Зрозумілість інтерфейсу і навчання	Гравець знаходить кнопку/ціль	кожні 1-2 ітерації
Performance	Стабільний FPS і відсутність лагів/піків	Профайлінг CPU/GPU	перед реліз-кандидатом
Compatibility	Запуск на різних ПК/пристроях	інший ноутбук/ОС	перед публікацією

5.1.3 Баг-репорти, severity/priority і тріаж: як керувати помилками

Управління дефектами – це не лише «знайти баг». Це процес прийняття рішень: що виправляємо зараз, що переносимо, що є блокером релізу, а що – косметика.

Стандартний баг-репорт має бути відтворюваним. Мінімальний шаблон

- Заголовок (коротко: що зламалося і де).
- Середовище: платформа, версія білду, сцена/рівень.
- Кроки відтворення (1-5 кроків).
- Очікуваний результат vs фактичний результат.
- Вкладення: скрін/відео/лог (за можливості).
- Severity (вплив) і Priority (терміновість) (табл. 5.2).

Таблиця 5.2

Severity і Priority: практичний приклад

Важливість	Приорітет	Приклад	Рекомендації
Severity S1 (Blocker)	Гра не запускається / прогрес неможливий	Краш при старті рівня	Фікс негайно
Severity S2 (Major)	Значна поломка механіки	Не працює атака/взаємодія	Фікс у поточній ітерації
Severity S3 (Minor)	Незначна, але помітна помилка	Плаває UI, але можна грати	Фікс за наявності часу
Severity S4 (Cosmetic)	Візуальна/текстова дрібниця	Опечатка, дрібний артефакт	Backlog / після релізу
Priority P0-P2	Терміновість від «зараз» до «пізніше»	P0 для реліз-блокера	Визначає PM/лід

5.1.4 Плейтест і user research: як збирати фідбек, що реально покращує гру

Плейтест – основний інструмент перевірки дизайну. Його задача – побачити, як гравець поводиться, а не що він каже.

Базова структура плейтесту

• Підготовка сценарію (що саме тестуємо) і 2-3 запитань.

• Пояснення правил: «грайте, думайте вголос», без підказок від розробника.

• Спостереження і нотатки (часи, помилки, де губиться увага).

• Коротке інтерв'ю після сесії (що запам'яталося, що було незрозуміло).

• Підсумок: 3-5 змін на наступну ітерацію.

Рекомендація для навчального курсу: плейтест кожні 1-2 ітерації, мінімум 2 тестери на версію. Міні-підхід «Think-Aloud та 5 Why» часто дає більше користі, ніж довгі анкети.

5.1.5 Телеметрія та аналітика: які події збирати і як не потонути в даних

Аналітика в іграх починається не з «великого даталейку», а з правильних питань. Для прототипу достатньо 8-15 ключових подій. Головна мета: підтвердити гіпотези про флоу і складність (рис. 5.2; табл. 5.3).



Рис. 5.2. Мінімальний пайплайн телеметрії: Подія в грі -> Лог/сховище -> Дашборд -> Рішення на ітерацію

Таблиця 5.3

Рекомендовані події та метрики для навчального прототипу

Подія (Event)	**Навіщо**	**Метрика/похідна**	**Типова інтерпретація**
SessionStart/End	Тривалість і частота	Session length, DAU	Занадто коротко – нудно/важко
LevelStart/Complete	Прохідність рівня	Completion rate, time-to-complete	Провал – навігація/складність
Death/Cause	Складність і помилки дизайну	Deaths per minute, hotspots	Багато смертей – поганий телеграф
TutorialStep	Зрозумілість навчання	Drop-off per step	Випадіння – переписати туторіал
UIOpen/Click	Зручність UI	Misclick rate, time-to-action	Багато кліків – неочевидний UI
EconomySpend/Earn	Баланс економіки	Net gain, sink/source	Інфляція – треба sink/обмеження

Практичне правило: збираємо лише те, що впливає на рішення. Якщо метрика не веде до конкретної дії, її краще не збирати (рис. 5.3).

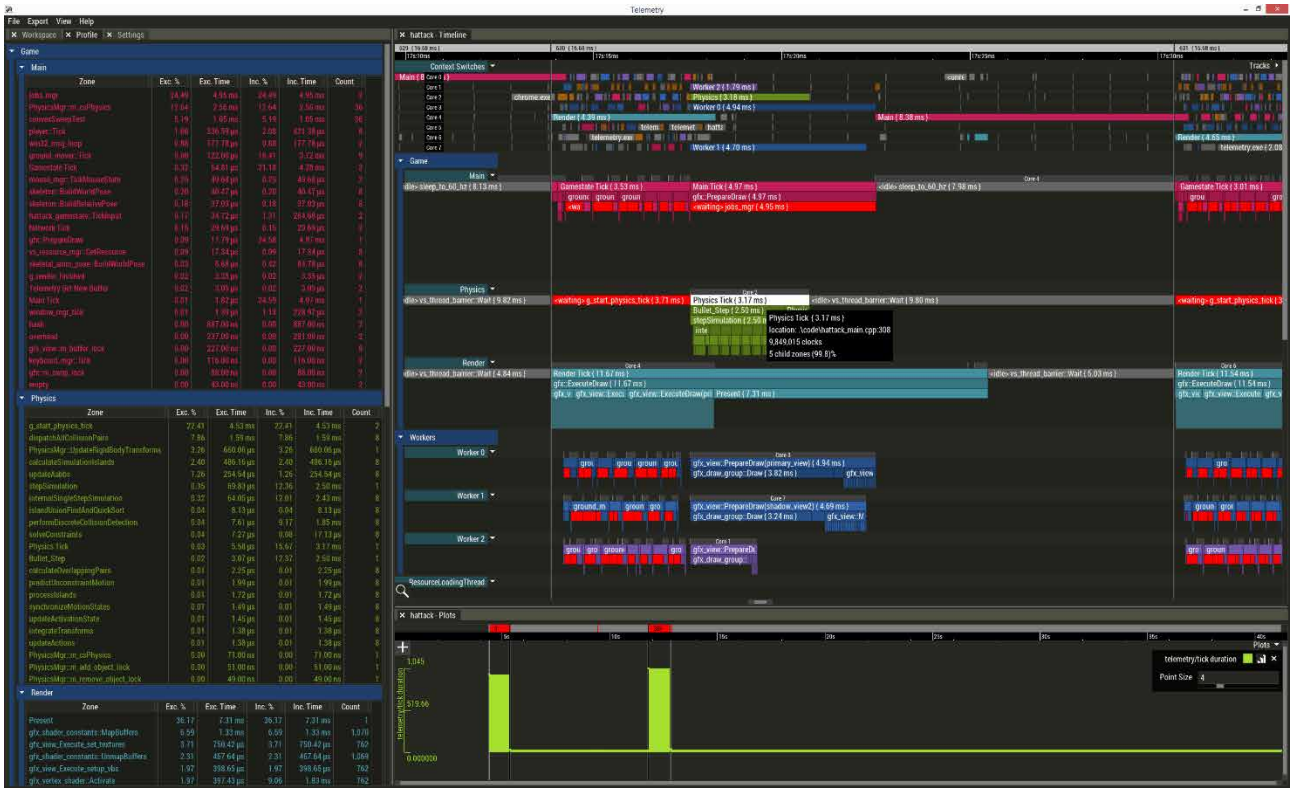


Рис. 5.3. Приклад візуальної телеметрії в ігровому рушії

5.1.10 Балансування: складність, ризик/нагорода і економіка

Баланс – це кероване налаштування систем так, щоб гра була справедливою, зрозумілою і цікавою. У прототипі балансування робиться швидко, через таблиці й тести, а не через «відчуття» розробника.

Типові об'єкти балансу

• Крива складності (difficulty curve) і темп прогресії.

• Дамаг/HP/швидкість/кулдауни для ворогів і гравця.

• Економіка: джерела/витрати ресурсів, ціни, нагороди.

• Risk/Reward: чи виправданий ризик і чи зрозуміла винагорода.

Рекомендація: використовувати data-driven підхід ScriptableObjects/DataAssets/JSON), щоб змінювати параметри без перекомпіляції білду.

5.1.11 Performance QA: швидкі перевірки продуктивності у Unity/Unreal

Навіть найкращий дизайн втрачає цінність, якщо гра працює нестабільно. Для курсових робіт достатньо базового підходу: виміряти, знайти «вузьке місце», внести правку, перевірити. Підхід «Performance budget» для прототипу: визначити ціль FPS і 2-3 типові сцени навантаження (найгірші випадки).

Unity: Profiler, Frame Debugger, Memory Profiler (за потреби). Unreal: Unreal Insights, Stat Unit/Stat GPU, профілювання рендеру.

5.1.12 Build management: версії, збірки, реліз-кандидати і нотатки

Якість демонстрації залежить від дисципліни збірок. Найкраща практика: кожен білд має версію, короткий changelog і мінімальний smoketest.

• Versioning (наприклад, v0.1.0, v0.1.1): патчі для багфіксів, мінор для нових фіч, мажор для великих змін.

• Release notes: 5-10 рядків про зміни, відомі проблеми, інструкції запуску.

• Release Candidate (RC): версія, що «може бути релізом», якщо не знайдено блокерів.

5.1.13 Підготовка до публікації: сторінка продукту, маркетингові матеріали, етика

Навіть навчальна гра повинна бути презентована професійно. Мінімальний пакет публікації містить

• Короткий опис (1-2 абзаци) та жанр та платформа.

• Скріншоти (3-6) і коротке відео/гіф (15-30 сек).

• Інструкція керування, системні вимоги, контакт автора.

• Політика використання контенту: ліцензії на асети, авторство, джерела.

• Етичні аспекти: попередження про контент, робота з персональними даними (якщо є).

5.1.14 Підтримка після релізу: фідбек, патчі, roadmap

Реліз – це не кінець, а початок циклу підтримки. Навіть для навчального проєкту варто показати розуміння post-release процесу:

• Збір фідбеку: коротка форма (Google Forms) та 3 питання.

• План патчів: P0 блокери, P1 основні, P2 поліпшення.

• Roadmap на 2-4 тижні: 3-5 найважливіших пунктів, що дають максимум ефекту.

5.1.15 Production-ready прототип: чеклист готовності та план дій

Нижче наведено мінімальний чеклист, який допомагає довести прототип до стану демонстрації/публікації (табл. 5.4; рис. 5.4).

Таблиця 5.4

Чеклист готовності прототипу до релізу/демо

Елемент	**Опис**
Stability	Немає крашів у базовому флоу; оброблені помилки
Gameplay	Core loop працює; прогрес можливий; немає блокерів
UI/UX	Керування зрозуміле; читабельність; підказки/туторіал
Content	Асети коректно імпортовані; немає «missing refs»
Performance	Ціль FPS тримається у worst-case сцені
Build	Версія, реліз-ноти, інструкція запуску, smoketest пройдений
Legal/Ethics	Ліцензії/авторство на асети; контент-попередження
Presentation	Скріншоти та коротке відео/гіф; опис проєкту





Рис. 5.4. Приклади готових ігор

Питання для самоперевірки

1. Чому QA є процесом, а не одноразовою перевіркою наприкінці?
2. Які елементи має містити мінімальний тест-план для прототипу?
3. Чим відрізняються severity і priority? Наведіть приклад.
4. Які 5 пунктів обов’язкові в баг-репорті?
5. Чому в плейтесті важливі спостереження, а не «оцінки» гравця?
6. Які події (events) є мінімальними для аналітики прототипу?
7. Що таке funnel і як він допомагає знайти проблемний крок туторіалу?

8. Назвіть два підходи до балансування, які можна застосувати без складних інструментів.
9. Які типові причини просідання FPS у студентських проєктах?
10. Який цикл оптимізації вважається базовим і чому він ефективний?
11. Що таке release candidate (RC) і як він використовується?
12. Які елементи входять до мінімального пакету публікації/демо?

Практичний кейс (час виконання 120-180 хв): «QA та аналітика та підготовка реліз-кандидата»

Мета кейсу: перетворити ваш поточний прототип на реліз-кандидата (RC) через серію керованих перевірок: smoketest, плейтест, баг-тріаж, базову телеметрію та релізний чеклист.

Вихідні дані: ваш прототип з Тем 5-6 або наданий демонстраційний проєкт (Unity/Unreal).

Сценарії – оберіть один

• A) Прототип арени (рух, атака, 1 ворог, HUD).

• B) Платформер (рух, стрибок, пастки, колекції).

• C) Невеликий квест/взаємодія (інтеракції, інвентар або задачі).

Кроки виконання завдання

Підготуйте smoketest-чеклист (5-10 пунктів) і виконайте його для поточного білду.

Проведіть плейтест з 2 тестерами: запишіть протокол (Таблиця 6.3 або власний).

Зареєструйте мінімум 10 дефектів/проблем (включно з UX проблемами) у баг-трекері (Trello/Jira/Notion/Google Sheet).

Виконайте тріаж: визначте severity і priority, позначте 3 реліз-блокери (P0).

Додайте 8-12 телеметричних подій (або логування) і сформуйте простий funnel (туторіал або рівень).

Зробіть 1 ітерацію: виправте P0 та 2 P1 проблеми, оновіть білд і виконайте регресійний smoketest.

Проведіть коротке профілювання (Unity Profiler/Unreal Insights): зафіксуйте 1-2 вузькі місця і зробіть одну оптимізацію.

Сформуйте реліз-ноти (5-10 пунктів) і заповніть чеклист готовності (Таблиця 7.4).

Зберіть RC-білд і протестуйте запуск на іншому ПК/пристрої (мінімум один запуск поза вашим середовищем).

Що необхідно здати

Посилання на білд/репозиторій або архів проєкту з версією RC.

Smoketest-чеклист та позначки виконання.

Протокол плейтесту (2 тестери) і список висновків/правок.

Таблиця багів з severity/priority та статусами після ітерації.

Список подій телеметрії та 2-3 висновки з даних/логів.

Release notes (5-10 рядків) і заповнений чеклист (Таблиця 7.4).

Критерії оцінювання

Дисципліна QA (наявність чеклистів, протоколів, регресії).
Якість тріажу і правильність визначення пріоритетів.
Наявність даних (телеметрія/логи) і коректні висновки.
Покращення прототипу після ітерації (видимий ефект).
Готовність RC до демонстрації (стабільність, інструкції, реліз-ноти).

Рекомендовані програми та сервіси

1. Unity Profiler, Frame Debugger, (за потреби) Memory Profiler.
2. Unreal Insights, Stat Unit/Stat GPU, Niagara Debug Tools.
3. Bug tracking: Jira / Trello / Notion / GitHub Issues.
4. Аналітика/телеметрія (для навчальних прототипів): просте логування у файл/JSON, Google Sheets; (опційно) GameAnalytics, Unity Analytics.
5. Запис відео: OBS Studio; скріншоти/гіфи: ShareX, ScreenToGif.
6. Збірка і дистрибуція: itch.io (найпростіше для навчальних релізів), Steam (за наявності доступу).

Рекомендовані джерела та матеріали

1. Офіційні документи і практичні гіди (посилання вставляти як активні URL):
2. Unity Manual (Profiling): https://docs.unity3d.com/Manual/Profiler.html
3. Unity Manual (Optimization): https://docs.unity3d.com/Manual/BestPracticeUnderstandingPerformanceInUnity.html
4. Unreal Engine Documentation: https://dev.epicgames.com/documentation/en-us/unreal-engine
5. Unreal Insights: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-insights
6. itch.io Docs (publishing): https://itch.io/docs
7. Book: Robert Nystrom – Game Programming Patterns (patterns): https://gameprogrammingpatterns.com/

Тема 5.2. Командний проєкт: механіка, ігровий цикл і прототип рівня

Мета теми: організувати командну роботу над основами гри: обрати жанр, сформулювати одну ключову механіку, побудувати core loop, описати сценарій першого рівня, порівняти референси, створити псевдокод і діаграму станів, перевірити кілька blockout-варіантів.

5.2.1 Обсяг і логіка завдання

Командний навчальний проєкт повинен бути достатньо малим, щоб за короткий цикл дати перевірюваний результат. Команда з 1-4 осіб обирає одну основну механіку, один core loop і один сценарій рівня. Додаткові системи допускаються лише тоді, коли вони підсилюють ядро, а не маскують його відсутність.

Проєкт виконується за логікою design thinking: емпатія до гравця та контексту; визначення бажаного досвіду; генерація кількох механік; вибір за цінністю й feasibility; швидкий прототип; плейтест; ітерація. Кожне рішення повинно бути представлене і текстом, і візуальним артефактом.

Результатом є не повна гра, а доказ того, що команда розуміє системний зв'язок між жанром, механікою, правилами, ігровим циклом, простором, feedback і ролями учасників (рис. 5.5).

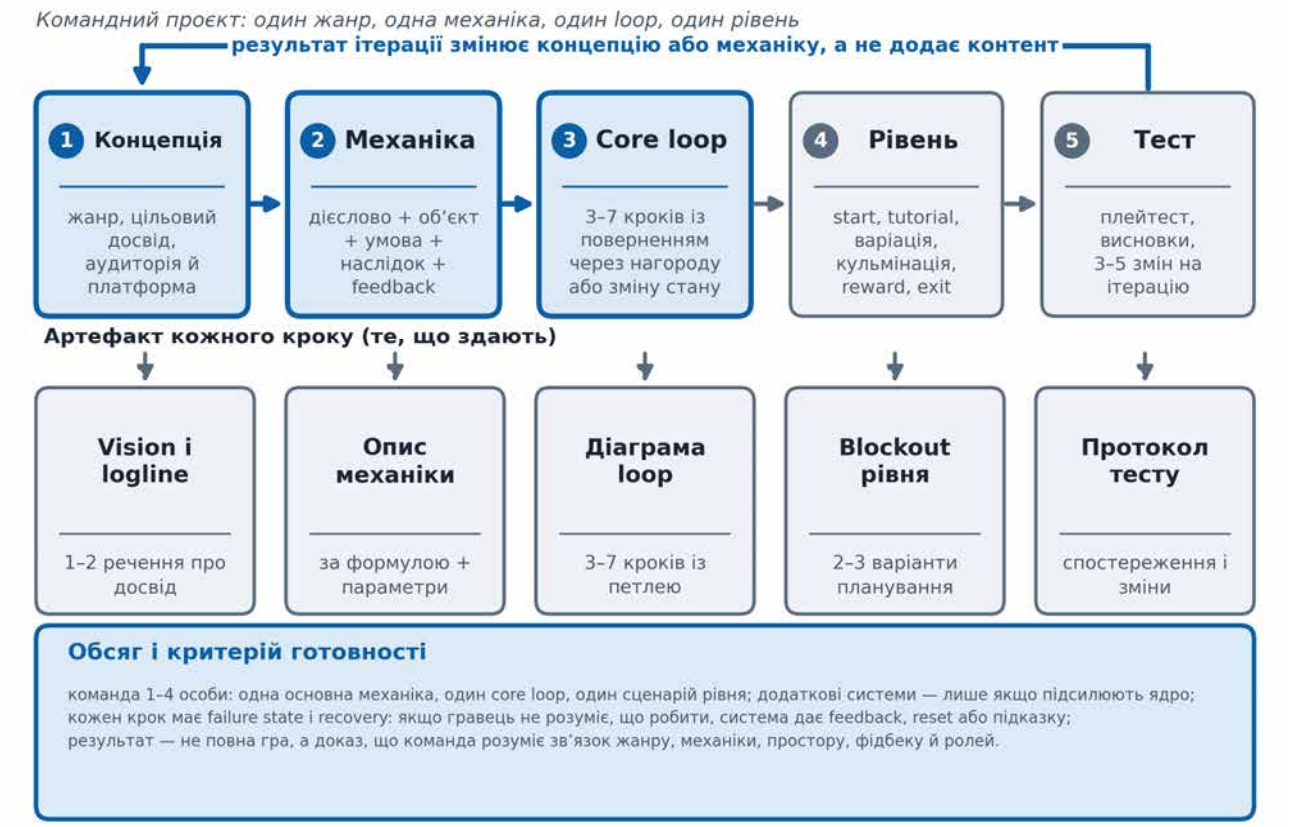


Рис. 5.5. Структура командного проєкту: концепція - механіка - loop - рівень - тест

5.2.2 Крок 1. Жанр, аудиторія та дизайн-виклик

Оберіть жанр або гібрид жанрів, але опишіть його через очікувані дії гравця. Формулювання «RPG» занадто широке; корисніше: «коротка тактична RPG, де гравець збирає команду з трьох персонажів і змінює позиції між атаками».

Сформулюйте persona, платформу, контролер і тривалість сесії. Далі визначте design challenge: яку потребу або емоцію задовольняє гра. Наприклад: «дати гравцю відчуття винахідливості через комбінування простих фізичних інструментів у 10-хвилинних рівнях».

Проведіть короткий ideation sprint. Кожен учасник пропонує окрему mini-game idea, після чого команда використовує dot voting та feasibility matrix. Обрана концепція має бути реалізована у greybox без складного контенту (табл. 5.5).

Таблиця 5.5

Матриця вибору концепції

Критерій вибору ідеї	Питання	Оцінка 1-5
Цінність для гравця	Чи зрозуміло, що саме є цікавим?	
Відповідність аудиторії	Чи відповідає контексту та платформі?	
Перевірюваність	Чи можна перевірити ядро за 2-4 години?	
Технічний ризик	Чи має команда необхідні навички?	
Контентний ризик	Чи не вимагає ідея забагато асетів?	
Унікальність	Чим рішення відрізняється від референсів?	
Масштабування	Чи можна додати глибину після прототипу?	

5.2.3 Крок 2. Одна механіка, core loop і сценарій

Ключову механіку опишіть формулою «дієслово + об’єкт + умова + наслідок + feedback». Наприклад: «Гравець перенаправляє промінь дзеркалом, якщо має line of sight; правильний напрям відкриває двері, супроводжується зміною кольору, звуком і рухом механізму».

Core loop має містити 3-7 кроків і повертатися до початку через нагороду або зміну стану. Сценарій першого рівня пояснює, як loop розкривається у просторі: start, безпечний tutorial, перша варіація, поєднання, кульмінація, reward, exit.

Для кожного кроку визначте failure state та recovery. Якщо гравець не розуміє, що робити, система повинна мати feedback, reset або hint. Відсутність відновлення робить плейтест непоказовим: учасник застрягає не через складність ядра, а через технічний dead end (табл. 5.6; рис. 5.6).

Таблиця 5.6

Шаблон опису механіки та loop

Елемент	Опис
Механіка	дія, об’єкт, умова, результат, feedback
Core loop	повторювана послідовність 3-7 кроків
Meta loop	прогрес між рівнями/сесіями
Ціль рівня	що має досягти гравець
Failure	як і чому виникає невдача
Recovery	checkpoint, reset, retry, alternate path
Нагорода	ресурс, відкриття, mastery, історія

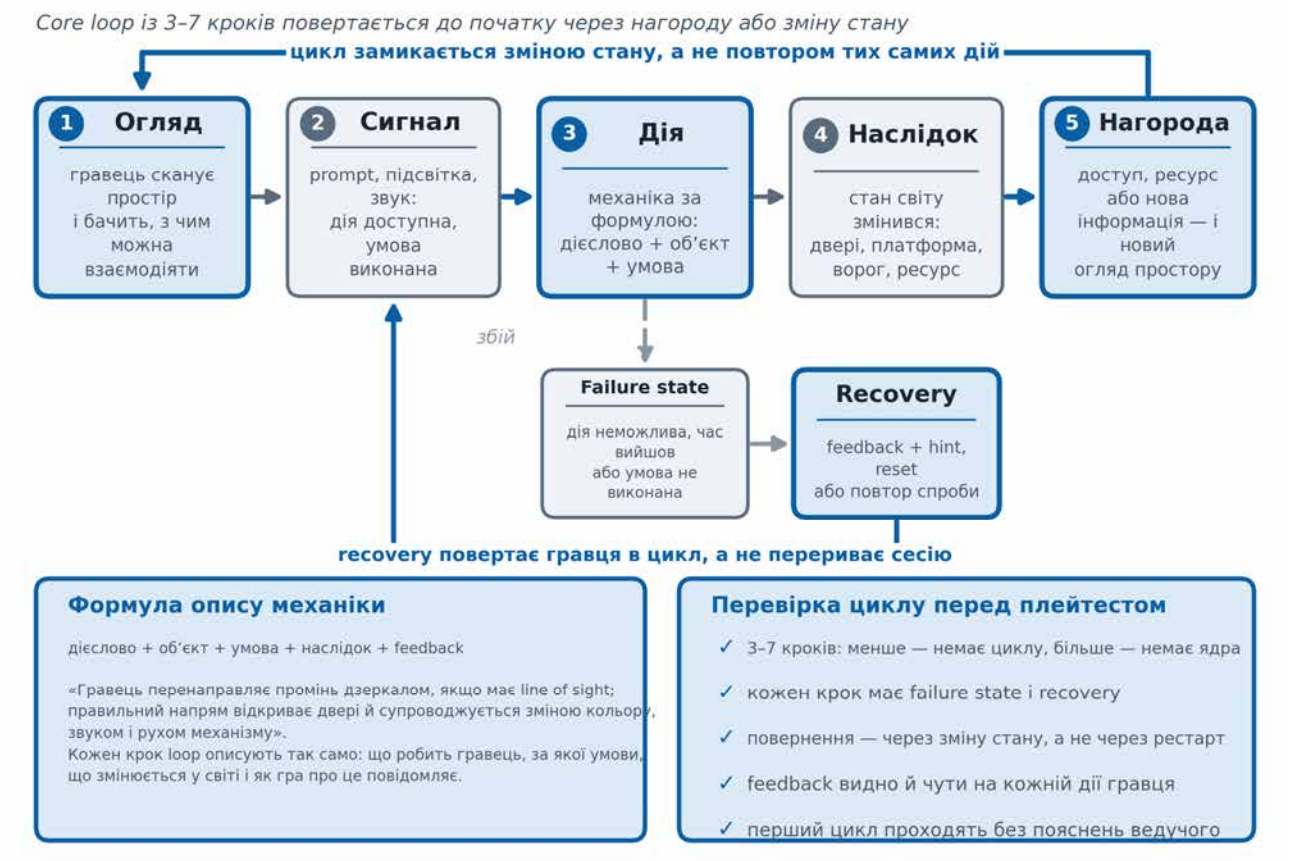


Рис. 5.6. Блок-схема core loop командного проєкту

5.2.4 Крок 3. Порівняння трьох референсів

Знайдіть три ігри зі схожою механікою або цільовим досвідом. Порівнюйте не загальну «схожість», а конкретні рішення: input, camera, timing, level structure, progression, accessibility, feedback, platform constraints. Для кожного референсу зафіксуйте одну практику, яку варто адаптувати, й один ризик, якого слід уникати (табл. 5.7).

Таблиця 5.7

Порівняльна таблиця референсів

Критерій	Власна ідея	Гра 1	Гра 2	Гра 3
Основна механіка				
Core loop				
Дизайн рівнів				
Складність				
Платформа/контролер				
Feedback				
Accessibility				
Висновок				

5.2.5 Крок 4. Псевдокод і діаграма станів

Псевдокод показує порядок дій, а діаграма станів – допустимі режими та переходи. Для ключової механіки обов'язково зазначте input accepted, blocked/invalid, active, success, fail, cooldown/recovery. Додайте події, що запускають animation, VFX, audio та UI (рис. 5.7).

```
state = Explore
while (sessionActive) {
 readInput();
 updateState();
 if (interactionAvailable) showPrompt();
 if (actionConfirmed) { executeMechanic(); giveFeedback(); }
 if (goalReached) completeLevel();
}
```

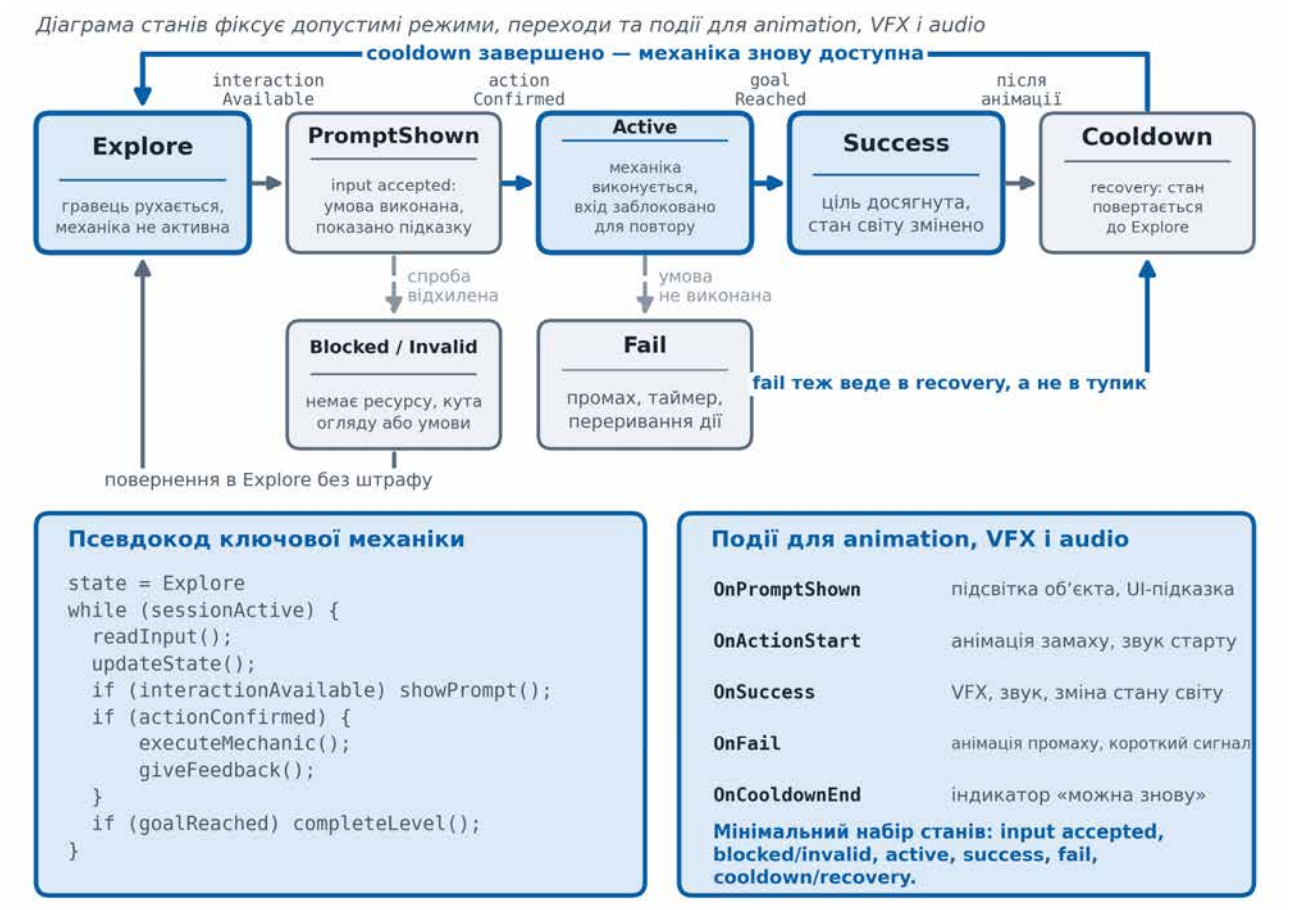


Рис. 5.7. Діаграма станів ключової механіки командного проєкту

5.2.6 Крок 5. Три blockout-варіанти першого рівня

Створіть 2-3 карти зверху або greybox-варіанти. Варіанти мають відрізнятися не косметикою, а структурою: лінійний шлях, петля, хаб, розгалуження, вертикальність або опційний ризик. Для кожного позначте start, goal, critical path, optional path, tutorial zone, challenge, checkpoint і reward.

Проведіть однаковий плейтест для кожного варіанта. Вимірюйте час проходження, кількість зупинок, помилки навігації, використання механіки, failure/retry та суб'єктивну ясність. Оберіть варіант на основі даних, а не особистої прихильності автора (рис. 5.8; табл. 5.8).

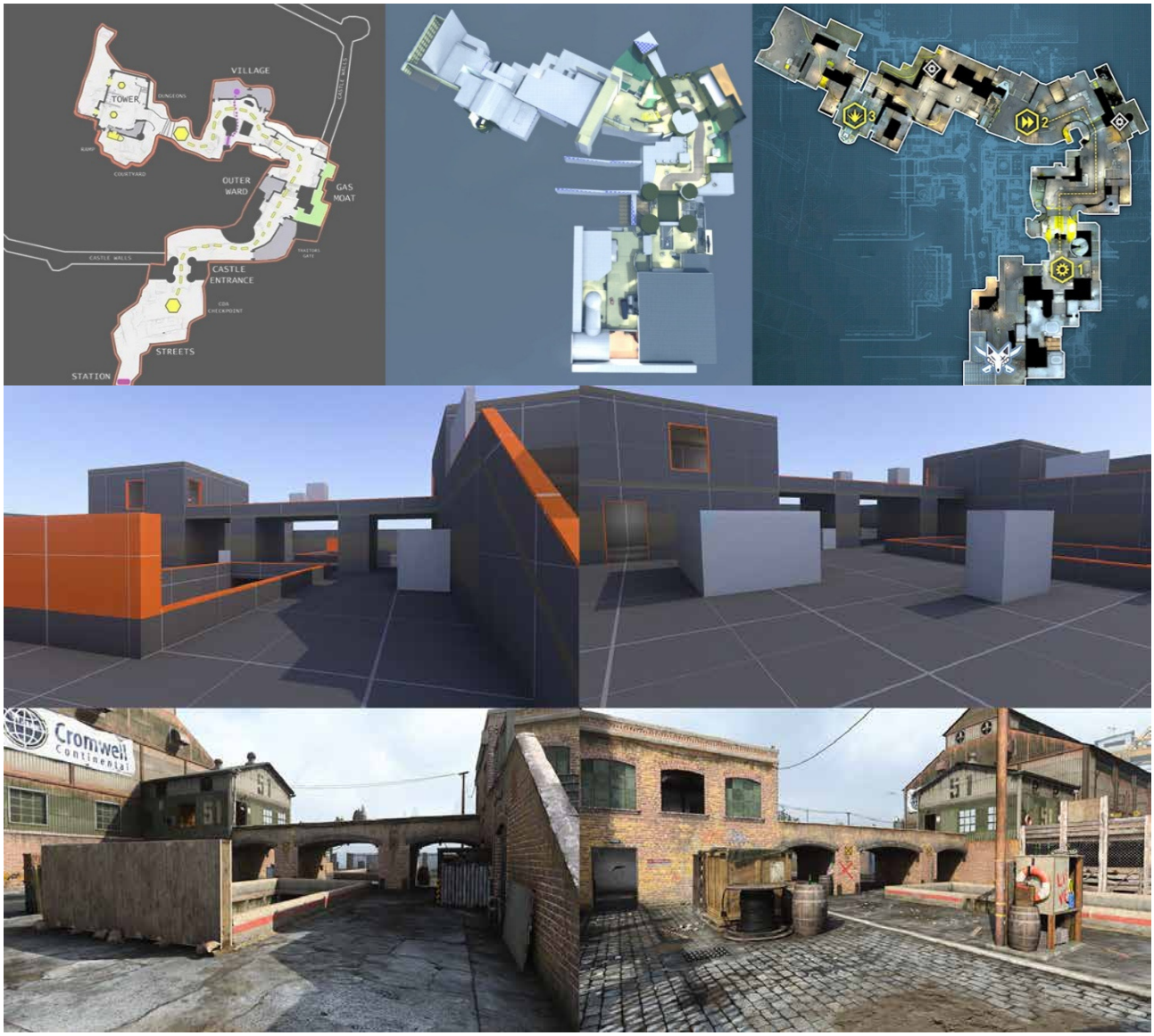


Рис. 5.8. Порівняння трьох top-down blockout-варіантів рівня

5.8.1 Крок 6. Ролі, процес і презентація

Таблиця 5.8

Ролі у командному проєкті

Роль	Основна відповідальність	Артефакт
Mechanics Architect	правила, параметри, прототип механіки	mechanic sheet + tuning
Loop Designer	core/meta loop, economy, progression	loop diagram
Level Visionary	blockout, navigation, pacing	2-3 maps + selected LDD
Comparative Analyst	референси, тестові критерії	comparison matrix
Integrator/Producer	build, VCS, scope, презентація	backlog + demo build

Презентація триває 5-7 хвилин і повинна показати: жанр та persona; одну механіку; core loop; порівняльну таблицю; псевдокод і state diagram; найкращий blockout; результати плейтесту; ролі й наступну ітерацію.

Усна доповідь має спиратися на візуальні матеріали. Не слід дублювати абзаци зі слайда. Кожен артефакт повинен відповідати на запитання: яке рішення прийнято, на яких даних воно ґрунтується і який ризик зменшує (табл. 5.9).

Таблиця 5.9

Критерії оцінювання командного проєкту

Критерій	**Частка**
Чіткість концепції та аудиторії	15%
Якість механіки й core loop	25%
Сценарій і blockout рівня	20%
Технічна формалізація: псевдокод/state diagram	15%
Порівняльний аналіз і аргументація	10%
Командний процес, build і презентація	15%

Тема 5.3. Мудборд, storyboard і візуальна комунікація ігрового бачення

Мета теми: навчити передавати атмосферу, стиль, ритм і сценарій гри до початку дорогого виробництва контенту за допомогою moodboard, reference board, color script, storyboard і flow diagram.

Мудборд як система обмежень, а не колекція красивих зображень

Мудборд повинен відповідати на конкретні питання: який тон і емоція; яка палітра; як виглядають матеріали, освітлення, силуети й камера; який рівень деталізації; як UI співвідноситься зі світом. Випадковий набір зображень без підписів не формує art direction.

Кожен референс слід анотувати: «потрібна холодна палітра з одним теплим акцентом», «використовуємо великі прості форми», «контрове світло відокремлює персонажа», «камера низька й підсилює масштаб». Додатково вказується джерело і ліцензійний статус, якщо матеріал планується використати не лише як натхнення.

Для навчального проєкту достатньо 1-2 сторінок: environment, character/silhouette, color, lighting, materials, UI/typography, camera. Після moodboard команда створює короткий style frame – власну композицію, що показує, як референси перетворюються на цілісне рішення (табл. 5.10) (рис. 5.9).

Таблиця 5.10

Структура moodboard ігрового бачення

Блок moodboard	**Що включити**	**Що підписати**
Середовище	архітектура, масштаб, форми, природні елементи	просторова мова, навігаційні акценти
Персонажі	силуети, пропорції, матеріали	читабельність ролі й стану
Палітра	5-8 базових кольорів і акценти	функція кольору
Освітлення	key/fill/rim, fog, time of day	mood і readability
UI/типографіка	HUD, меню, іконки, шрифти	ієрархія та доступність
Камера	FOV, композиція, відстань, рух	як кадр підтримує механіку



Рис. 5.9. Приклад анотованого moodboard ігрового проєкту

Storyboard і сценарій взаємодії

Storyboard показує послідовність дій, реакцій і змін стану. Для геймплею важливо фіксувати не лише кадр, а й input, мету, feedback, camera, звук, очікуваний стан гравця та перехід до наступного beat. Чотири-шість кадрів достатні для базової interaction loop; 8-16 – для трейлера або складнішої сцени.

Розкадровка може бути грубою. Її якість визначається ясністю, а не художнім рівнем. Стрілки руху, номери кадрів, тривалість, тип плану, підписи SFX/VFX та критичні UI-події роблять storyboard виробничим документом.

Flow diagram є альтернативою, коли потрібно показати меню, прогрес, розгалуження або нелінійний сценарій. Гібридний формат поєднує блок-схему з ключовими кадрами: рішення, наслідок, зміна стану, повернення до loop (табл. 5.11; рис. 5.10; 5.11).

Таблиця 5.11

Приклад storyboard breakdown

Кадр	Дія/стан	Камера	UI/VFX/SFX	Тривалість/ перехід
1	вхід у зону, ціль ще не відома	wide establishing	ambience, objective appears	3 с, cut
2	гравець помічає інтерактивний об’єкт	medium, focus	outline + prompt + tick	2 с
3	активація механіки	close/action	anticipation, charge SFX	1.5 с
4	impact і зміна світу	camera impulse	VFX impact, confirm, haptic	0.5 с
5	відкрито шлях/нагороду	reveal	music stinger, objective update	3 с, continue loop

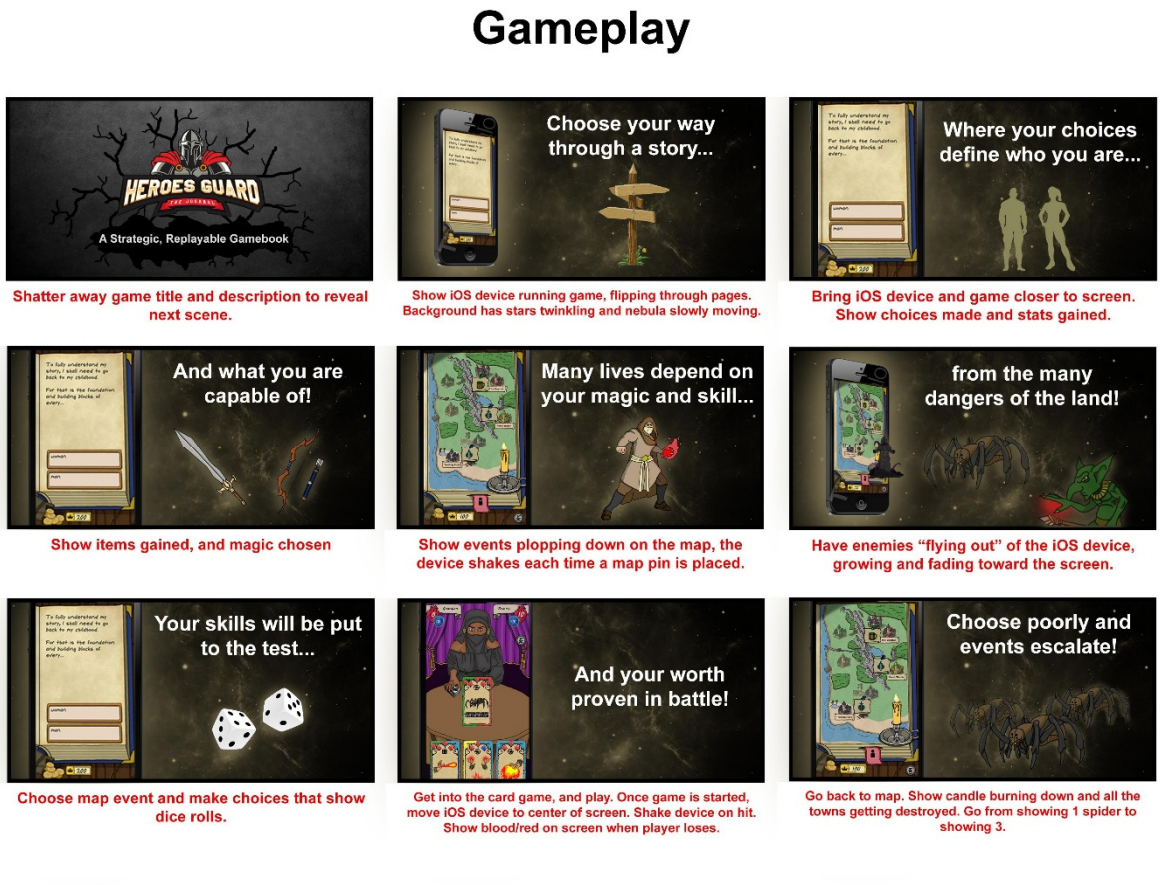


Рис. 5.10. Storyboard із дев’яти кадрів для ключової interaction loop

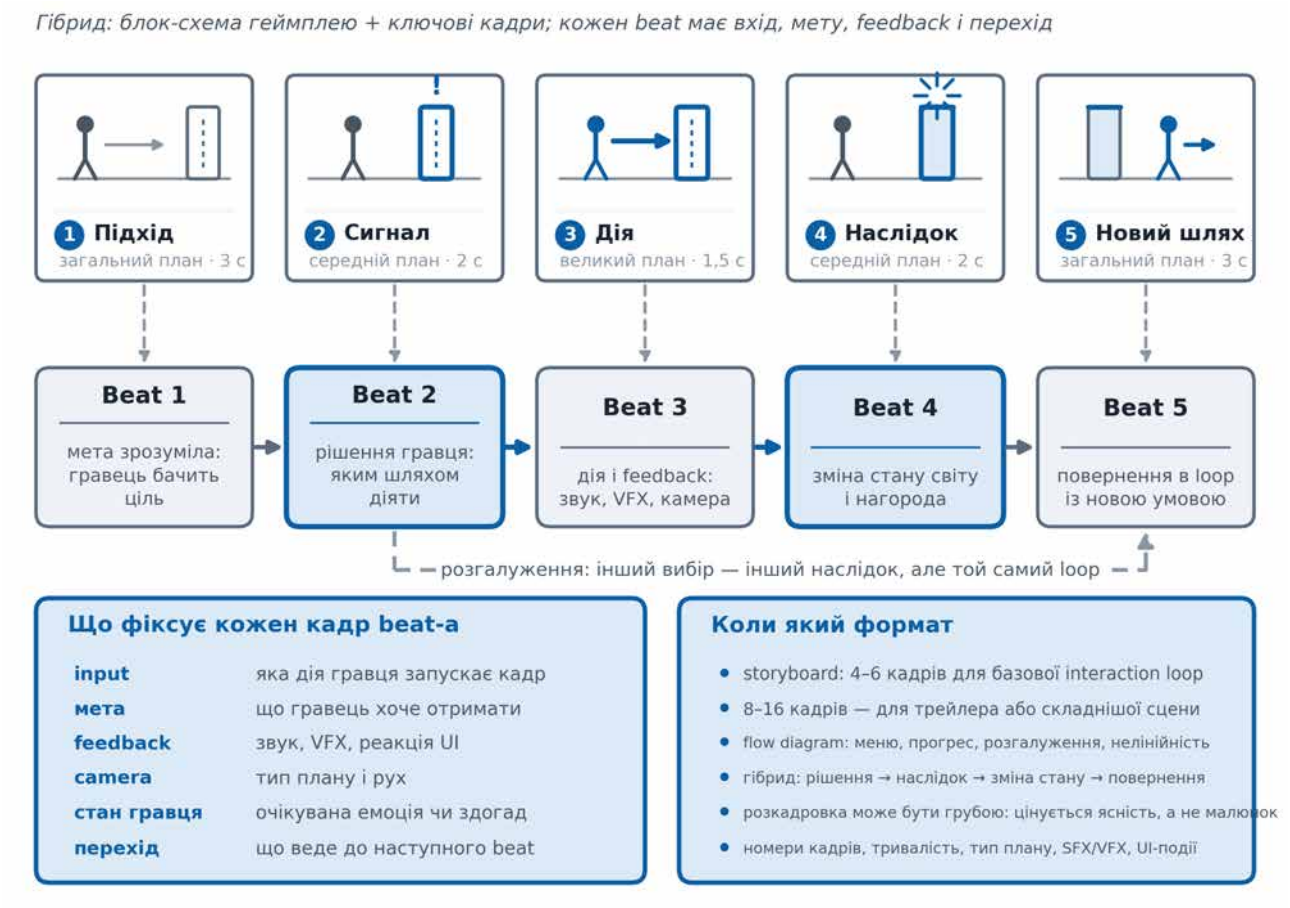


Рис. 5.11. Гібридна схема gameplay flow та сюжетних beats

Практичне завдання «Ігрове бачення»

1. Створіть moodboard на 1-2 сторінки або одну цифрову дошку. Додайте 5-10+ референсів середовища, персонажів, палітри, освітлення, UI та камери. Кожне зображення підпишіть.
2. Створіть storyboard на 4-6 кадрів або flow diagram, що показує рух гравця, core loop чи сюжетний фрагмент. Позначте input, feedback і результат.
3. Напишіть резюме 150-200 слів: робоча назва, жанр, основна дія, core loop, емоція, платформа та чому обраний візуальний напрям підтримує механіку.
4. Оформіть джерела референсів і зазначте, що є натхненням, а що планується використати як асет (табл. 5.12).

Таблиця 5.12

Критерії оцінювання moodboard та storyboard

Критерій	**Частка**
Креативність та оригінальність	30%
Візуальна злагодженість, тон і атмосфера	25%
Чіткість gameplay/story flow	25%
Оформлення, читабельність і джерела	20%

Тема 5.4. Фінальний vertical slice, трейлер і pitch deck

Мета теми: об’єднати результати курсу у цілісний vertical slice або демонстраційну сцену, підготувати стабільний build, коротке відео й презентацію, що пояснюють ідею, аудиторію, core loop, візуальну мову, технічні рішення, результати тестування та наступні кроки.

5.4.1 Що є vertical slice

Vertical slice – невеликий, але цілісний фрагмент гри, який показує очікувану якість ключових систем. Це не просто рівень із великою кількістю контенту і не technical prototype. У slice повинні узгоджуватися механіка, керування, камера, UI, анімація, VFX, звук, lighting, performance і основний production pipeline.

Обсяг визначається через одну демонстраційну обіцянку. Наприклад: «30-90 секунд керованого бою з однією ability, читабельним ворогом і завершеним feedback stack» або «коротка головоломка з двома правилами та зрозумілим onboarding». Все, що не підтримує цю обіцянку, переноситься до next steps.

Перед інтеграцією команда створює Definition of Done: запуск без критичних помилок, цільовий frame time, робочий input, зрозуміла ціль, завершена interaction loop, базовий options/accessibility, відсутність missing assets, credits/licenses, інструкція запуску й стабільний запис відео (рис. 5.12) (табл. 5.13).

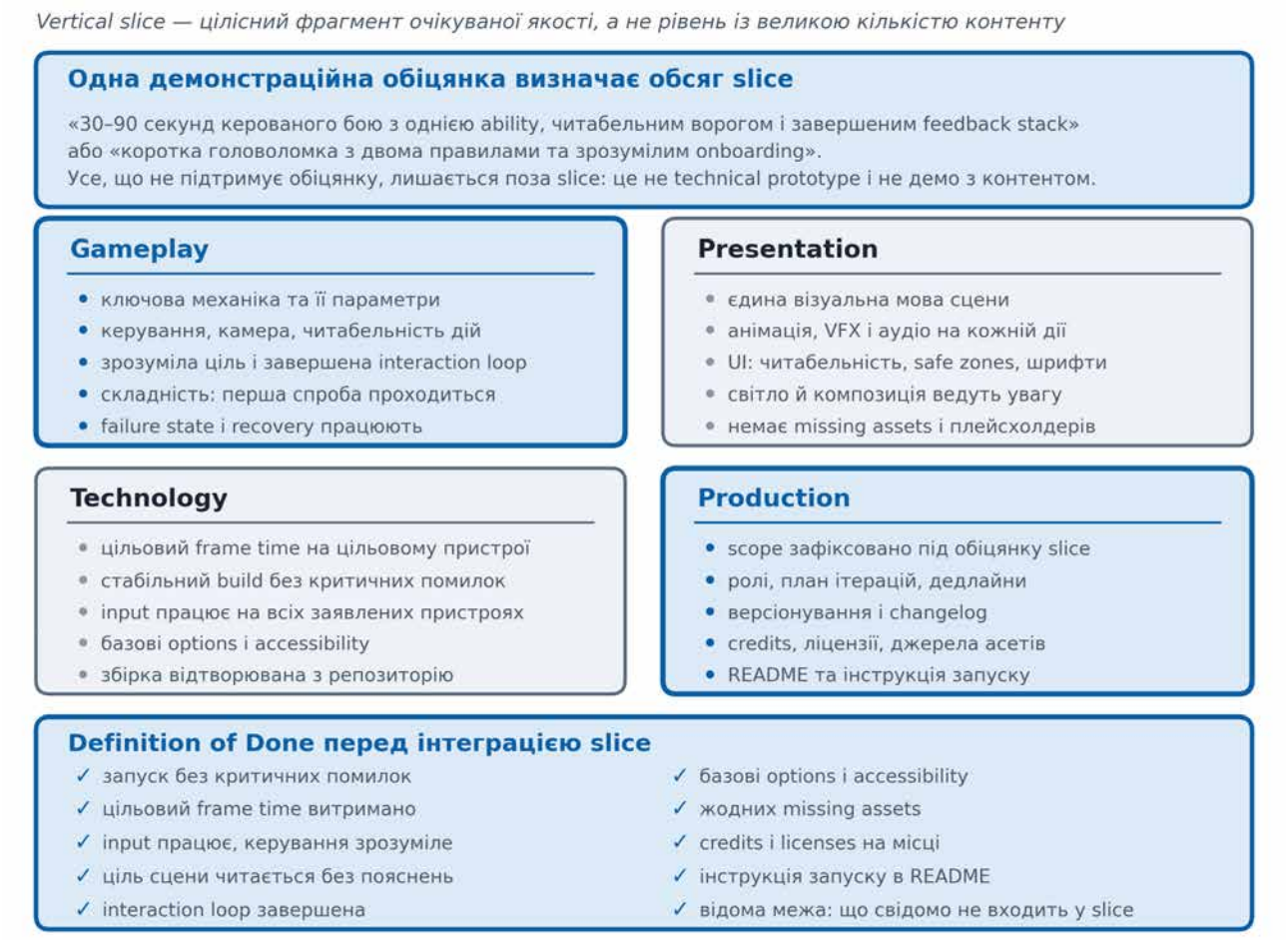


Рис. 5.12. Структура vertical slice: gameplay, presentation, technology і production

5.4.2 Формати демонстраційного відео

Таблиця 5.13

Формати фінальної демонстрації

Формат	Тривалість	Рекомендована структура
Трейлер гри	60-120 с	hook 0-3 с; fantasy 5-15 с; core action 20-60 с; escalation; title/CTA
Проліт сцени	30-90 с	establishing; pan/dolly/orbit; точки інтересу; hero asset; outro
Демонстрація персонажа/елемента	45-90 с	силует; матеріал; анімація/FX; контекст; title card
Тизер	30-60 с	проблема/інтрига; 2-3 функції; перевага; title
Музичне відео	60-120 с	ритмічний монтаж; macro/detail; повторюваний motif; фінальний reveal
Технічний breakdown	45-120 с	до/після; debug overlay; feature; profiler; результат

Відео має показати результат, а не приховувати його монтажем. Основна механіка повинна бути зрозумілою без усного коментаря. Для навчального трейлера допустимі title cards, але вони не повинні замінювати gameplay.

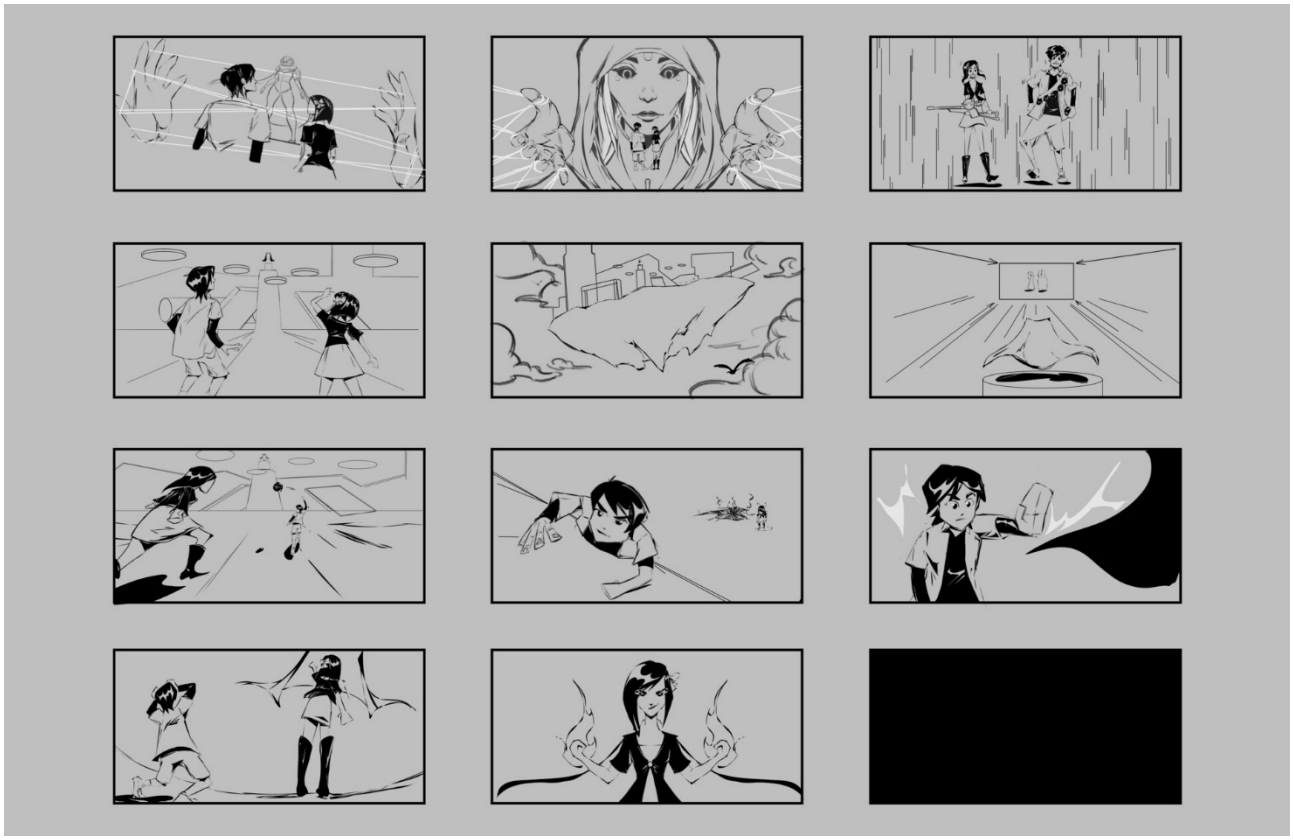
Запис здійснюють із рушія: Unity Cinemachine/Timeline/Recorder, Unreal Sequencer/Movie Render Queue або OBS для live gameplay. Перед записом вимикають debug, перевіряють resolution, frame rate, audio levels, cursor, safe area і стабільність сцени. Рекомендується записати кілька повних дублів, а не лише окремі кадри.

Shot list зменшує хаос: номер сцени, кадр, план/об’єктив, рух, тривалість, дія, потрібні асети, VFX, SFX, UI та статус. Storyboard дозволяє перевірити темп до запису (табл. 5.14; 5.15; рис. 5.13).

Таблиця 5.14

Шаблон shot list

№	Сцена/кадр	Камера	Дія	Тривалість	Асети/SFX	Статус
1	Establishing	wide dolly	показати світ і героя	4 с	ambience, logo off	
2	Mechanic intro	medium follow	перша дія	6 с	UI prompt, SFX	
3	Impact	close + impulse	результат механіки	3 с	VFX, hit, stinger	
4	Escalation	dynamic	комбінація/ризик	10 с	enemy, music rise	
5	Reveal/CTA	hero shot	нагорода/назва	5 с	title, logo	



The Pen - Working | Shot Lists | View Only | StudioBinder Inc.

Shot Lists | 12 shots

Sc. 1 | INT. EMPTY CONFERENCE ROOM - DAY 12 Shots

STATUS	IMAGE	SHOT	DESCRIPTION	CAMERA	SHOT SIZE	SHOT TYPE	MOVEMENT	EQUIPMENT
		1	Intern darts off	CAM A	WS	Eye Level	Static	Hand Held
✓		2	Intern finds a stairwell	CAM B	WS	Eye Level	Static	Hand Held
✓		3	Intern rushes down the stairs	CAM C	WS	Overhead	Static	Hand Held
		4	Jamie takes the elevator	CAM A	MS	Eye Level	Static	Sticks
		5	Intern gets to courtyard, looks for escape	CAM B	EWS	HA	Pan	Sticks
		6	360 on Jamie	CAM D	MS	LA	360 Spin	Gimbal / Circular

Рис. 5.13. Storyboard і shot list фінального трейлера

5.4.3 Pitch deck на 5-7 слайдів

Таблиця 5.15

Структура pitch deck

Слайд	Зміст
Title + Logline	назва, одне речення про fantasy і цінність
Аудиторія та контекст	persona, платформа, тривалість сесії, потреба
Core loop / Feature focus	одна діаграма, ключові механіки й pillars
Look & Feel	moodboard, палітра, lighting, UI, camera
Демонстраційна структура	storyboard, level flow або architecture slice
Results & Learnings	плейтест, telemetry, performance до/після, проблеми
Next Steps / CTA	scope наступної ітерації, ролі, ризики, посилання на build

Pitch deck повинен пояснювати рішення, а не дублювати трейлер. На кожному слайді бажано мати один висновок, один головний візуальний матеріал і мінімум тексту. У доповіді потрібно показати причинний зв’язок: проблема – рішення – доказ.

Слайд Results & Learnings є обов’язковим. Навіть невдалий тест має цінність, якщо команда пояснює гіпотезу, дані, зміну й результат. Слайд Next Steps демонструє розуміння scope та production, а не просто перелік бажаних функцій (рис. 5.14) (табл. 5.16).



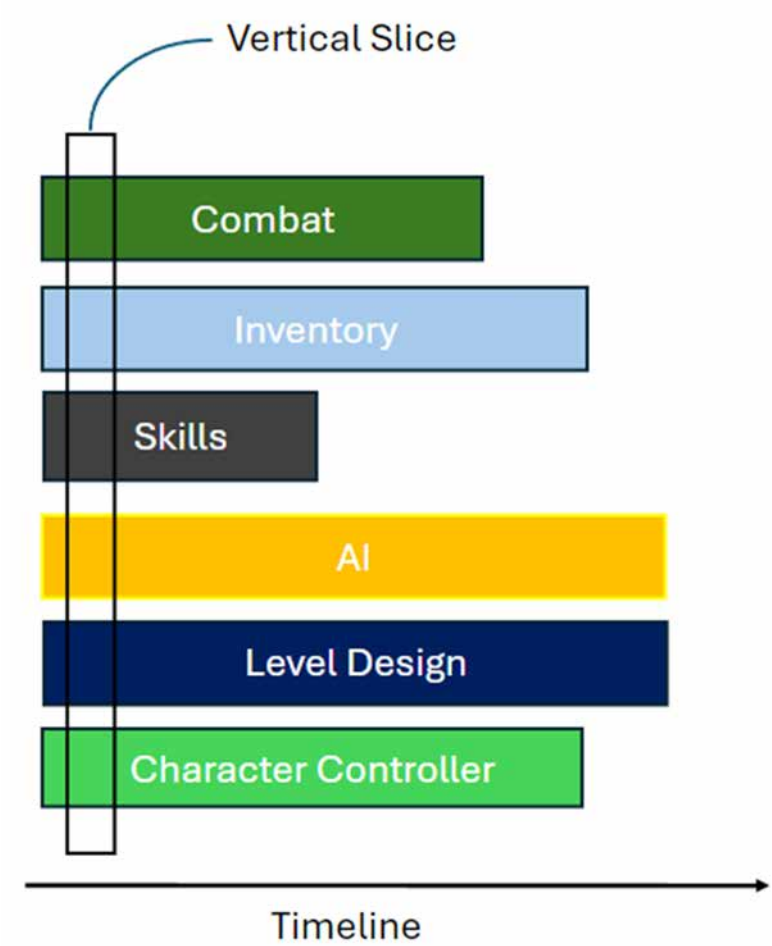


Рис. 5.14. Приклад структури pitch deck для vertical slice

5.4.4 План виконання на чотири тижні

Таблиця 5.16

План фінального проєкту

Тиждень	Основні результати	Gate
1	logline, persona, core loop, scope, moodboard, storyboard, shot list	концепція затверджена; ризики визначені
2	асети, blockout, механіка, camera/UI, lighting/FX/audio baseline	feature complete для запису
3	інтеграція, плейтест, performance pass, rough cut відео	vertical slice stable; rough cut переглянуто
4	polish, regression, final capture, audio mix, titles, pitch deck	release candidate і фінальний захист

5.4.5 Технічні характеристики та якісні обмеження

• Відео: MP4/H.264, 1080p, стабільні 24/25/30 або 60 кадрів за секунду залежно від матеріалу.

• Звук: stereo; діалог/VO читається; піки без кліпінгу; музика не маскує SFX.

• Титри: достатній час читання, високий контраст, однакова типографіка.

• Субтитри: рекомендовані; обов’язкові для важливої мовної інформації.

• Ліцензування: credits і посилання на ліцензії всіх сторонніх assets.

• Photosensitivity: уникати сильних повторюваних спалахів; додати попередження за потреби.

• Build: README, керування, системні вимоги, відомі проблеми, номер версії (табл. 5.17).

5.4.6 Оцінювання фінального проєкту

Таблиця 5.17

Рубрика фінального проєкту

Вид роботи	Що оцінюється	Частка
Концепція та аудиторія	logline, persona, ціннісна пропозиція	15%
Дизайн та наратив	core loop, feature focus, цілісність beats	20%
Візуальний напрям	moodboard, style frame, UI/типографіка	15%
Кінематограф і монтаж	кадр, ритм, переходи, операторська логіка	15%
Аудіо	музика/SFX, синхронізація, мікс	10%
Polish і доступність	контраст, субтитри, settings, відсутність критичних дефектів	10%
Процес і артефакти	board, shot list, VCS, credits, ліцензії	10%
Рефлексія та pitch deck	learnings, next steps, якість захисту	5%

Пакет здачі завдання:

1) build або відео vertical slice; 2) трейлер/демо 30-120 с; 3) pitch deck 5-7 слайдів; 4) README і керування; 5) credits/licenses; 6) short postmortem 250-500 слів; 7) посилання на репозиторій або changelog за вимогою викладача.

Тема 5.5. Гейміфікація навчального процесу та мотивація команди

Мета теми: пояснити, як ігрові механіки можуть підтримувати активне навчання, командний production і рефлексію, не підмінюючи внутрішню мотивацію поверхневими балами та значками

5.5.1 Гейміфікація і гра: принципова різниця

Гейміфікація використовує елементи ігрового дизайну в неігровому контексті: прогрес, цілі, challenge, feedback, статус, колекціонування, choice, collaboration. Вона не перетворює автоматично навчальне завдання на повноцінну гру. Її якість залежить від того, чи підтримують механіки реальну навчальну поведінку.

Бали, значки й leaderboard є лише зовнішнім шаром. Якщо вони винагороджують кількість замість якості, створюють токсичну конкуренцію або дозволяють «фармити» прості дії, система шкодить навчанню. Натомість meaningful gamification пов’язує прогрес із компетентністю, автономією, співпрацею й видимою майстерністю.

Для курсу з game development доречно гейміфікувати не оцінку як таку, а виробничий процес: короткі milestones, badges за конкретні професійні практики, peer review quests, iteration streak, bug-hunt, performance challenge, documentation achievement і showcase (рис. 5.15).

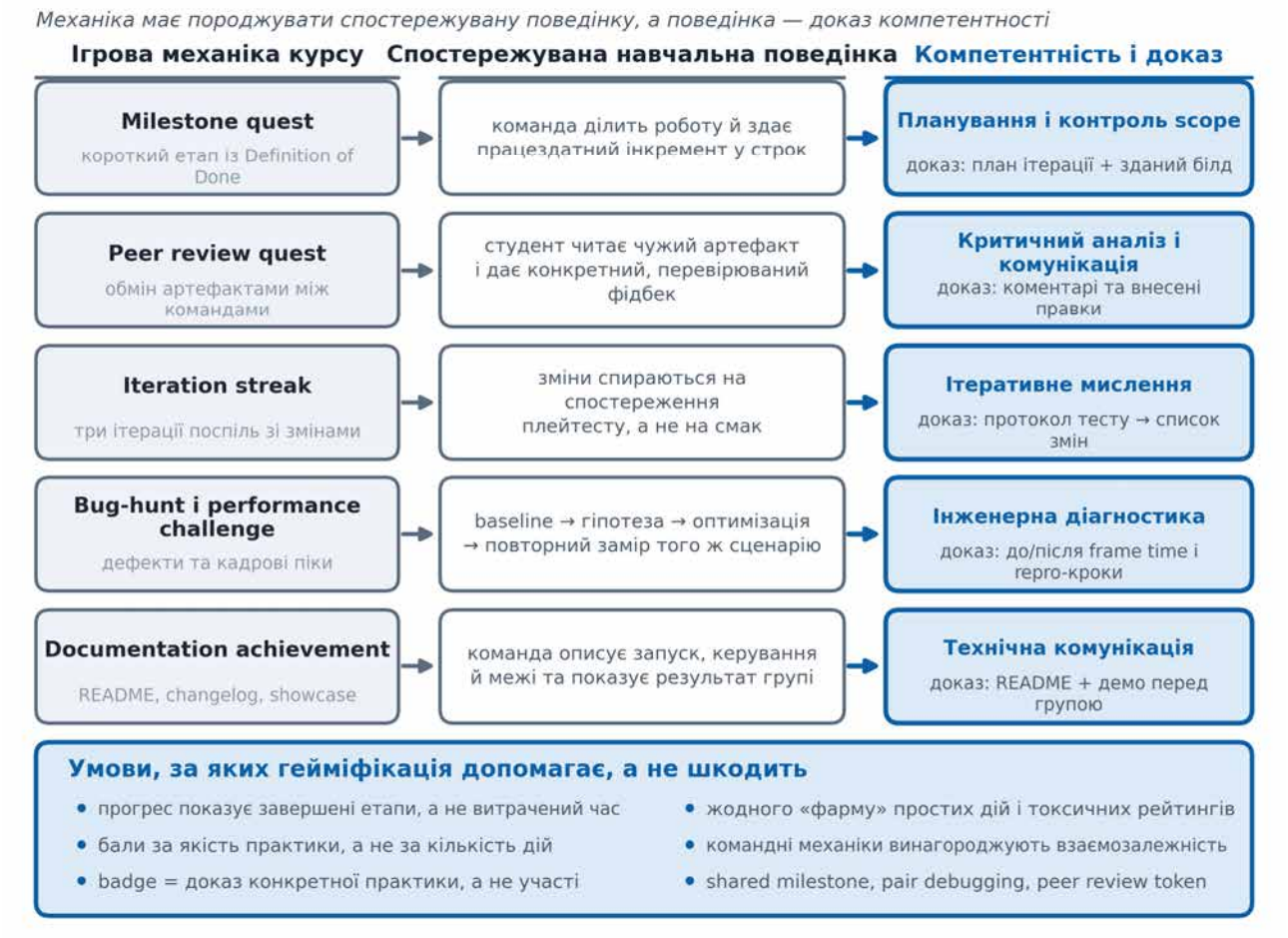


Рис. 5.15. Зв’язок ігрових механік, навчальної поведінки та компетентностей

5.5.2 Модель «ціль – дія – feedback – прогрес – рефлексія»

Кожна гейміфікована активність повинна починатися з навчальної цілі. Далі визначаються спостережувані дії, швидкий feedback, спосіб показати прогрес і коротка рефлексія. Наприклад, ціль «навчитися профілювати» перетворюється на quest: зафіксувати baseline, знайти bottleneck, зробити одну зміну, повторити capture, пояснити результат.

Progress bar доцільний, якщо відображає завершені етапи, а не просто час. Badge має бути доказом конкретної практики: Vision Crafter – чіткий logline і persona; Board Master – актуальний backlog; Sound Designer – аудіо підсилює interaction; Accessibility Advocate – settings і субтитри; Iteration Pro – задокументований цикл hypothesis-change-result.

Командні механіки повинні винагороджувати взаємозалежність. Shared milestone, pair debugging, peer review token або cross-discipline challenge допомагають бачити результат команди, а не лише індивідуальні бали (табл. 5.18; 5.19; рис. 5.16).

Таблиця 5.18

Гейміфікаційні механіки в курсі з розробки ігор

Механіка	Навчальна мета	Приклад	Ризики
Quest	структурувати складну роботу	baseline -> fix -> verify	перетворити на чекбокс без осмислення
Badge	позначити підтверджену компетентність	Iteration Pro, Accessibility Advocate	нагороджувати участь без якості
Progress map	показати етапи проєкту	concept -> prototype -> slice -> release	плутати прогрес і обсяг
Challenge	стимулювати deliberate practice	зменшити frame time на 20%	оптимізувати заради числа
Peer review	розвинути аналіз і комунікацію	2 конкретні спостереження + 1 пропозиція	неконструктивна оцінка
Leaderboard	порівняння в однаковій вправі	bug hunt за обмежений час	токсична конкуренція
Unlock	progressive disclosure	advanced shaders після базового PBR	штучне блокування ресурсів

5.5.3 Badges і критерії доказу

Таблиця 5.19

Приклади компетентнісних badges

Badge	Доказ	Критерій
Vision Crafter	logline, persona, pillars	концепція зрозуміла за 30 секунд
Loop Architect	core/meta loop diagram	дії, feedback, reward і progression пов’язані
Feel Tuner	A/B build + метрики	одна змінна, зафіксований результат
Board Master	backlog і milestone board	актуальні статуси, власники, DoD
Sound Layerer	audio breakdown + demo	anticipation/impact/recovery і мікс
Performance Hunter	profiler before/after	bottleneck підтверджено даними
Accessibility Advocate	settings + test	реальна опція, а не лише опис
Iteration Pro	postmortem	гіпотеза, зміна, дані, висновок



Рис. 5.16. Карта прогресу курсу з badges та milestones

5.6.3 Гейміфікація без маніпуляції

Етична гейміфікація є прозорою: студент розуміє правила, критерії, вагу й спосіб отримання feedback. Вона не повинна використовувати темні патерни, штучний дефіцит, непередбачувані покарання або публічне приниження.

Варто забезпечити альтернативні шляхи до однакової компетентності. Один студент може продемонструвати UI/UX через прототип меню, інший – через accessibility audit. Bonus не повинен компенсувати відсутність базових результатів навчання.

Система має підтримувати reflection. Після milestone команда відповідає: що планували; що зробили; що довели; що змінимо. Саме рефлексія перетворює зовнішній reward на усвідомлену професійну практику.

Міні-дизайн гейміфікації: оберіть одну тему курсу. Визначте навчальну ціль, цільову поведінку, quest, feedback, progress indicator, badge з доказом і коротку reflection. Перевірте систему за питаннями: чи можна «грати» метрику; чи є альтернативний шлях; чи не шкодить конкуренція; чи зрозумілі правила.

Тема 5.6. Академічна доброчесність, ліцензування, портфоліо і кар'єрні траєкторії

Мета теми: сформувати відповідальне ставлення до використання стороннього контенту, генеративних інструментів, даних плейтесту й командної праці; навчити перетворювати навчальні проєкти на доказ компетентності у портфоліо.

5.6.4 Ліцензії та provenance асетів

Кожен сторонній асет має походження, автора, ліцензію та умови використання. Команда повинна вести asset register: назва, джерело, автор, URL, ліцензія, зміни, місце використання. Формулювання «знайдено в інтернеті» не є дозволом.

CC0, CC BY, royalty-free, marketplace license та educational license мають різні умови. Навіть безкоштовний асет може вимагати attribution або забороняти повторне розповсюдження source-файлу. Перед публікацією слід перевірити не лише модель, а й текстури, шрифти, музику, SFX, плагіни та логотипи.

Генеративний контент потрібно позначати відповідно до політики закладу, платформи та проєкту. Важливо зберігати промпти/версії за потреби, перевіряти права, не використовувати персональні дані без згоди й не видавати автоматично створений матеріал за власну ручну роботу (табл. 5.20).

Таблиця 5.20

Шаблон реєстру сторонніх асетів

Поле asset register	**Приклад**
Назва/ID	SFX_Door_Metal_01
Тип	audio / WAV
Джерело/автор	назва бібліотеки, автор
Ліцензія	CC BY 4.0 / marketplace license
Атрибуція	текст для credits
Зміни	trim, pitch, layering
Місце використання	Level_Lab / Door interaction
Статус перевірки	approved / replace before release

5.6.5 Дані плейтесту, приватність і відповідальна аналітика

Плейтестери повинні розуміти, які дані збираються і для чого. Для навчального прототипу достатньо мінімальної telemetry: події, час, помилки, hardware/performance за необхідності. Не слід збирати ім’я, email, голос, відео чи інші персональні дані без явного згоди.

Дані потрібно зберігати обмежений час, захищати доступ і використовувати лише для заявленої мети. У звіті краще подавати агреговані результати та анонімні ідентифікатори. Відео плейтесту не публікують без дозволу учасника.

Аналітика не повинна бути інструментом маніпуляції. Retention, monetization і engagement оцінюються разом із добробутом користувача, прозорістю, віковими обмеженнями та можливістю контролювати сповіщення, витрати й час (табл. 5.21–5.23; рис. 5.17).

5.6.6 Портфоліо як доказ процесу й результату

Таблиця 5.21

Портфоліо за професійними напрямами

Напрям	Що показати	Слабкий варіант
Game Design	loop, mechanics, tuning, playtest, LDD	лише великий GDD
Gameplay Programming	architecture, code, debug, performance	скріншот редактора без пояснення
Technical Art	shader/VFX breakdown, budgets, tools	лише красивий render
UI/UX	flow, states, accessibility, telemetry	статичний макет без interaction
Level Design	goals, blockout iterations, metrics	фінальна сцена без flow
Audio	event map, layers, RTPC, mix	окремий WAV без інтеграції
Production/QA	board, risks, test plan, release process	перелік обов’язків без артефактів

Таблиця 5.22

Портфоліо за професійними напрямами

Напрям	Що показати	Слабкий варіант
Game Design	loop, mechanics, tuning, playtest, LDD	лише великий GDD
Gameplay Programming	architecture, code, debug, performance	скріншот редактора без пояснення
Technical Art	shader/VFX breakdown, budgets, tools	лише красивий render
UI/UX	flow, states, accessibility, telemetry	статичний макет без interaction
Level Design	goals, blockout iterations, metrics	фінальна сцена без flow
Audio	event map, layers, RTPC, mix	окремий WAV без інтеграції
Production/QA	board, risks, test plan, release process	перелік обов’язків без артефактів

Таблиця 5.23

Портфоліо за професійними напрямами

Напрям	Що показати	Слабкий варіант
Game Design	loop, mechanics, tuning, playtest, LDD	лише великий GDD
Gameplay Programming	architecture, code, debug, performance	скріншот редактора без пояснення

Напрям	Що показати	Слабкий варіант
Technical Art	shader/VFX breakdown, budgets, tools	лише красивий render
UI/UX	flow, states, accessibility, telemetry	статичний макет без interaction
Level Design	goals, blockout iterations, metrics	фінальна сцена без flow
Audio	event map, layers, RTPC, mix	окремий WAV без інтеграції
Production/QA	board, risks, test plan, release process	перелік обов’язків без артефактів

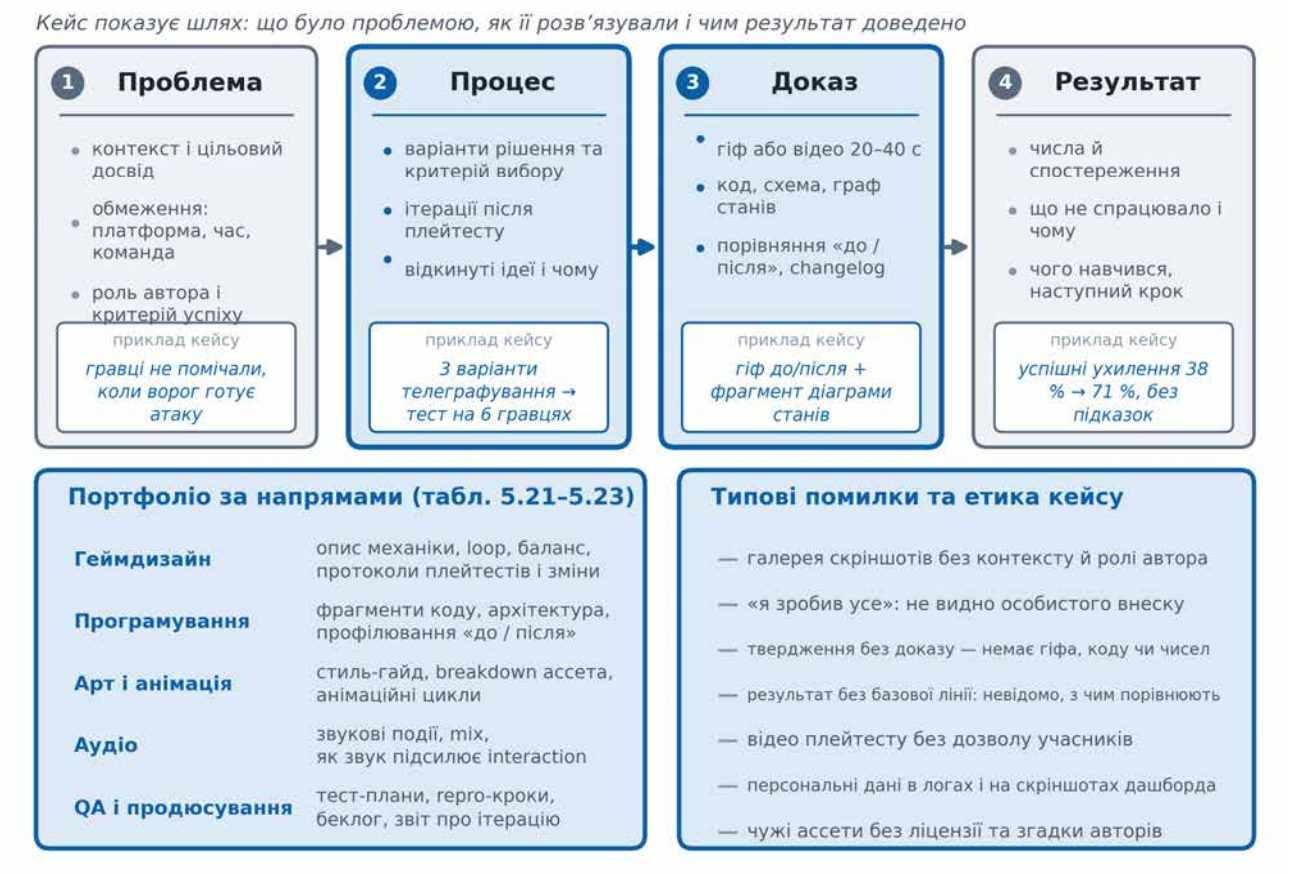


Рис. 5.17. Структура портфоліо-кейсу: проблема - процес - доказ - результат

5.7.1 Кар’єрні траєкторії та план розвитку

Game development містить багато спеціалізацій: game/level/system designer, gameplay/engine/tools programmer, technical artist, UI/UX designer, 2D/3D artist, animator, VFX artist, sound designer, QA, producer, data analyst, AI programmer. На старті важливо обрати primary track і одну суміжну компетентність.

Навчальний план розвитку доцільно будувати через портфоліо-проєкти, а не лише список курсів. Для кожного кварталу визначаються компетентність, невеликий deliverable, критерій якості, peer review і публікація. Один завершений і пояснений vertical slice цінніший за п’ять незакінчених великих ідей.

Після кожного проєкту варто проводити postmortem: що спрацювало; що не спрацювало; які ризики були недооцінені; які інструменти слід поглибити; що стане наступним портфоліо-кейсом. Така рефлексія формує професійне мислення й допомагає обирати роль свідомо.

Підсумковий кар’єрний план:

Оберіть primary track і supporting track. Складіть матрицю «компетентність – доказ – поточний рівень – наступний проєкт – термін». Додайте один кейс із цього курсу, перепакуйте його за структурою Context/Role/Process/Evidence/Result/Learnings і підготуйте 60-секундну демонстрацію.

Тема 5.7. Бізнес-моделі, publishing, локалізація та Live Ops

Мета теми: сформувати уявлення про зв’язок продуктового дизайну, способу монетизації, платформи, операційної підтримки, локалізації, community management та етичних обмежень після релізу.

5.7.2 Бізнес-модель як дизайн-обмеження

Premium, free-to-play, subscription, advertising, DLC, cosmetics і early access формують різні очікування щодо сесій, progression, контенту, retention і підтримки. Монетизація не додається наприкінці без впливу на дизайн.

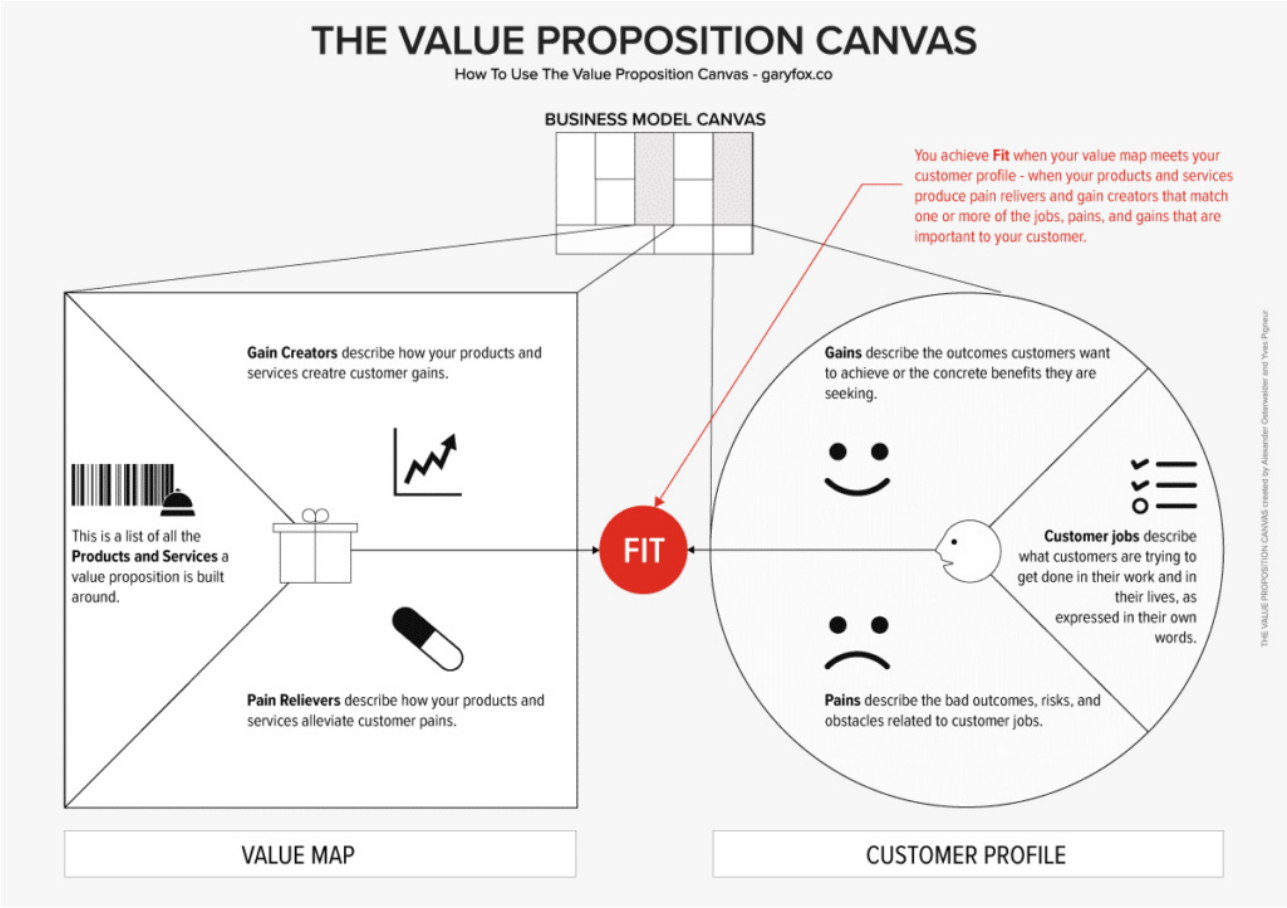
Здорова модель створює обмін цінності: користувач розуміє, за що платить, які обмеження існують і чи можна повноцінно користуватися продуктом без прихованих пасток. Dark patterns, штучне тертя, непередбачувані витрати й exploitation особливо ризиковані для неповнолітніх.

Для навчального проєкту достатньо описати hypothetical business model, але механіки повинні залишатися етичними й узгодженими з цінністю. Наприклад, premium puzzle game не потребує infinite grind лише для штучного retention (табл. 5.24) (рис. 5.18) (табл. 5.25) (табл. 5.26) (табл. 5.27).

Таблиця 5.24

Порівняння бізнес-моделей

Модель	Дизайнерський вплив	Перевага	Етичний/виробничий ризик
Premium	завершений досвід, upfront value	прозора покупка	висока вимога до launch quality
DLC/expansion	модульний post-launch content	продовження цінності	fragmentation
Free-to-play cosmetics	довга progression/social identity	низький barrier	pressure to retain
Subscription	регулярний content cadence	predictable revenue	continuous production load
Ads	session breaks/rewarded ads	доступність	disruption/privacy
Early access	iterative community feedback	рання валідація	unfinished reputation / scope creep



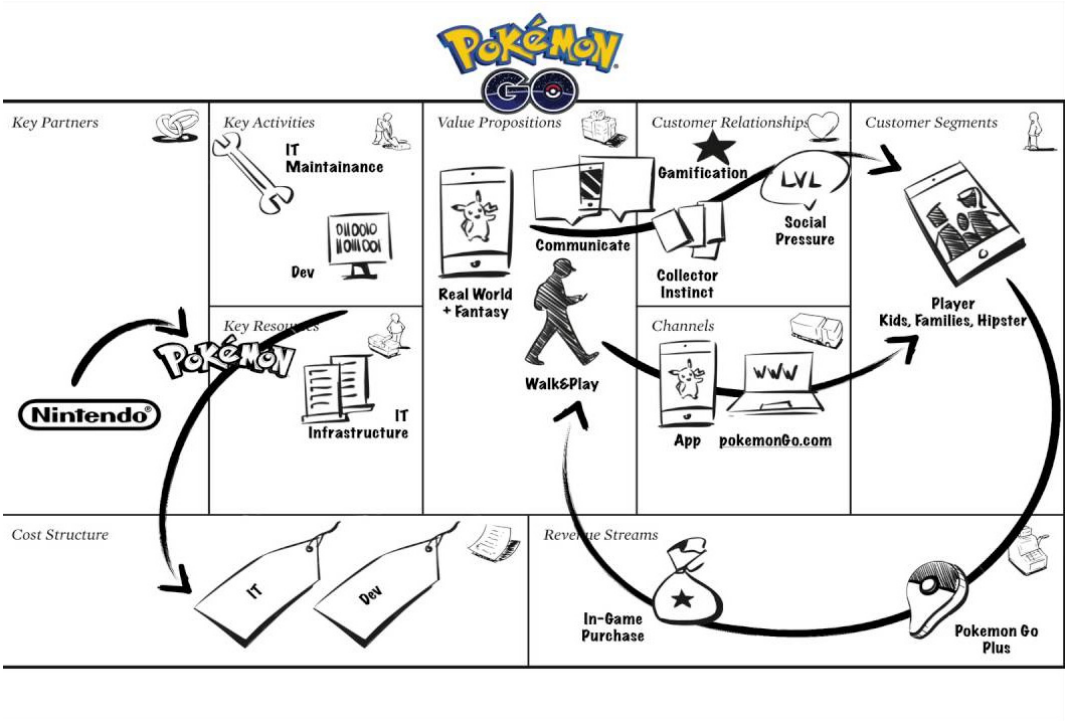


Рис. 5.18. Зв’язок value proposition, core loop і business model

5.7.3 Store readiness, ratings і platform compliance

Таблиця 5.25

Store readiness checklist

Категорія	Артефакт / перевірка
Product	назва, short/long description, features, tags
Media	icon, screenshots, capsule art, trailer
Technical	build, depots, system requirements, controller
Legal	licenses, privacy, EULA, third-party notices
Ratings	вікові анкети, content descriptors
Localization	store text, UI, subtitles, fonts
Operations	support, crash reports, patch process
Accessibility	feature disclosure, captions, remapping, scaling

Таблиця 5.26

Store readiness checklist

Категорія	Артефакт / перевірка
Product	назва, short/long description, features, tags
Media	icon, screenshots, capsule art, trailer
Technical	build, depots, system requirements, controller
Legal	licenses, privacy, EULA, third-party notices
Ratings	вікові анкети, content descriptors
Localization	store text, UI, subtitles, fonts
Operations	support, crash reports, patch process
Accessibility	feature disclosure, captions, remapping, scaling

Таблиця 5.27

Store readiness checklist

Категорія	Артефакт / перевірка
Product	назва, short/long description, features, tags
Media	icon, screenshots, capsule art, trailer
Technical	build, depots, system requirements, controller
Legal	licenses, privacy, EULA, third-party notices
Ratings	вікові анкети, content descriptors
Localization	store text, UI, subtitles, fonts
Operations	support, crash reports, patch process
Accessibility	feature disclosure, captions, remapping, scaling

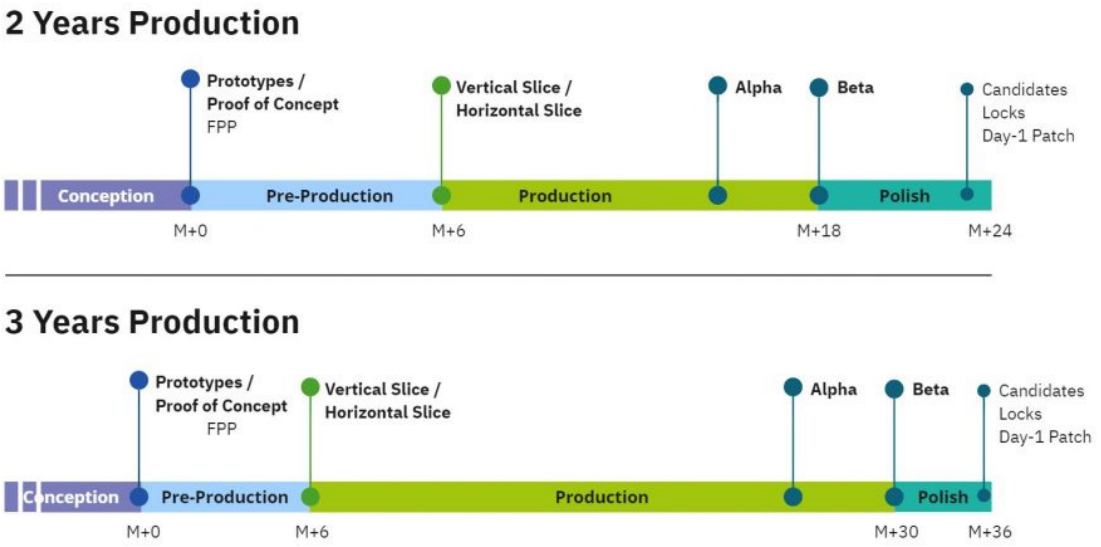


Рис. 5.19. Release pipeline від RC build до store publication

5.7.4 Локалізація та internationalization

Internationalization означає, що система технічно готова до різних мов і культур: string IDs, external tables, plural forms, date/number formats, right-to-left, font fallback, text expansion, input methods і subtitle timing. Localization є перекладом та культурною адаптацією конкретною мовою.

Текст не повинен бути hard-coded у texture або code. Контекстні коментарі, screenshots, speaker, gender/number і character limits допомагають перекладачу. Псевдолокалізація виявляє truncation, missing glyphs і layout assumptions до реального перекладу.

Voice-over і cinematics потребують окремого плану: script locking, timing, lip sync, retake, subtitle split і version tracking. Навіть без VO потрібно перевірити швидкість читання й можливість повторити репліку (табл. 5.28).

Таблиця 5.28

Локалізаційні ризики

Ризики	Профілактика
Text expansion	30-50% запас, responsive layout
Missing glyphs	font fallback і language test
Hard-coded strings	string table / localization keys
Wrong context	comments, screenshots, speaker metadata
Plural/gender errors	ICU-like rules / structured variables
Text in textures	separate localized layer
Subtitle too fast	reading-speed limits і reflow
RTL failure	mirroring strategy і native review

5.7.5 Live Ops, контентний календар і керування змінами

Live Ops – це система регулярної підтримки після релізу: patches, events, balance, content, community, experiments, customer support і incident response. Вона потребує календаря, ownership, telemetry, rollback і комунікації.

Подія має мати ціль, аудиторію, mechanics, rewards, schedule, localization, QA, monitoring і exit plan. Накладання подій може створити economy inflation, когнітивний шум і production overload.

Балансні зміни повинні мати patch notes і пояснення intent. A/B test застосовують лише там, де він етично й методологічно виправданий; групи мають бути порівнюваними, metric – попередньо визначеною, а результат – інтерпретований разом із якісним feedback (табл. 5.29; рис. 5.20).

Таблиця 5.29

Шаблон Live Ops event brief

Live Ops елемент	Питання
Goal	яку проблему або цінність вирішує подія?
Segment	для кого вона релевантна?
Content	що нове, а що повторно використовується?
Economy	sources/sinks, inflation, fairness
Schedule	start/end, timezone, conflicts
Quality	QA, localization, accessibility
Monitoring	health metrics, alerts, support
Exit	що відбувається з rewards/progress після завершення?

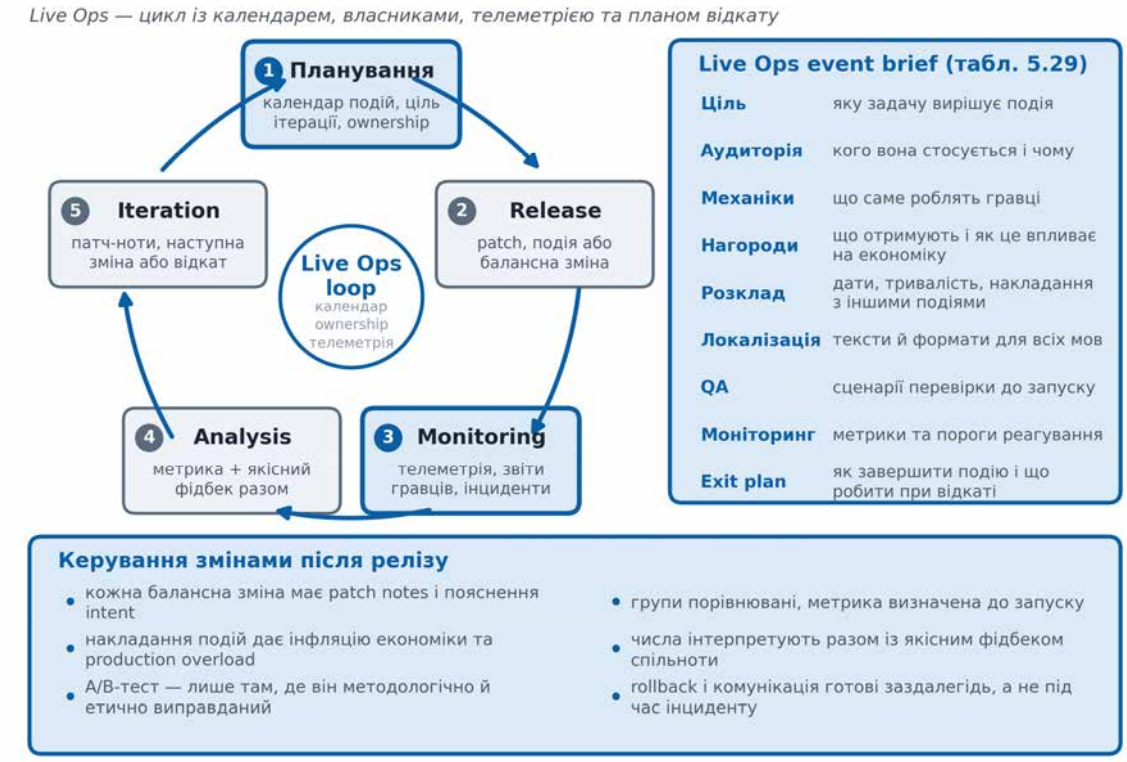


Рис. 5.20. Live Ops loop: планування - release - monitoring - analysis - iteration

5.7.5 Community, support та incident response

Community channels є джерелом qualitative evidence, але не репрезентують усіх гравців. Гучний feedback потрібно співвідносити з telemetry, support tickets, reviews і цільовою аудиторією.

Incident response визначає severity, on-call/owner, communication, mitigation, rollback, root-cause analysis і prevention. Навіть навчальний online prototype повинен мати план, що робити при втраті server, save або authentication.

Публічна комунікація має бути своєчасною, конкретною й чесною. Не слід обіцяти термін без evidence. Після виправлення корисний post-incident review без пошуку винного: умови, системна причина, чому detection не спрацював, які зміни внесено.

Практичний кейс: release & Live Ops plan

Підготуйте store-ready пакет для vertical slice: description, screenshots, trailer brief, ratings/accessibility disclosure, localization matrix, release checklist і 4-тижневий hypothetical Live Ops calendar з event brief, metrics та rollback plan.

Тема 5.8. Серйозні ігри, симуляції та оцінювання навчального результату

Мета теми: показати, як ігрові технології застосовуються в освіті, медицині, безпеці, промисловості та науці; навчити пов’язувати learning/performance objectives з механіками, evidence, feedback і валідним оцінюванням.

5.8.1 Від навчальної мети до інтерактивної моделі

Serious game має подвійну відповідальність: бути достатньо залучальною інтерактивною системою та досягати зовнішньої мети - навчити, тренувати, оцінити, пояснити або підтримати рішення. Fun може бути засобом, але не є єдиним критерієм успіху.

Починати потрібно з performance objective: яку дію користувач повинен виконувати краще після досвіду, у яких умовах і за яким стандартом. Потім визначають knowledge/skill model, типові помилки, сценарії, feedback і evidence.

Механіка повинна репрезентувати суттєві причинно-наслідкові зв’язки в предметній області. Якщо навчання стосується triage, рішення щодо пріоритету та обмежених ресурсів важливіші за декоративні badges. Якщо тренується безпечна процедура, система повинна моделювати послідовність, ризики та наслідки (табл. 5.30; рис. 5.21).

Таблиця 5.30

Каркас serious game design

Компонент	Питання	Артефакт
Performance objective	що користувач робить після навчання?	Observable objective
Domain model	які правила реального процесу критичні?	Expert map
Error model	які типові хибні рішення?	Misconception list
Scenario	у якому контексті проявляється навичка?	Scenario script

Компонент	Питання	Артефакт
Evidence	які дії доводять компетентність?	Evidence model
Feedback	коли й як пояснювати помилку?	Feedback policy
Assessment	як вимірюється transfer/retention?	Test plan

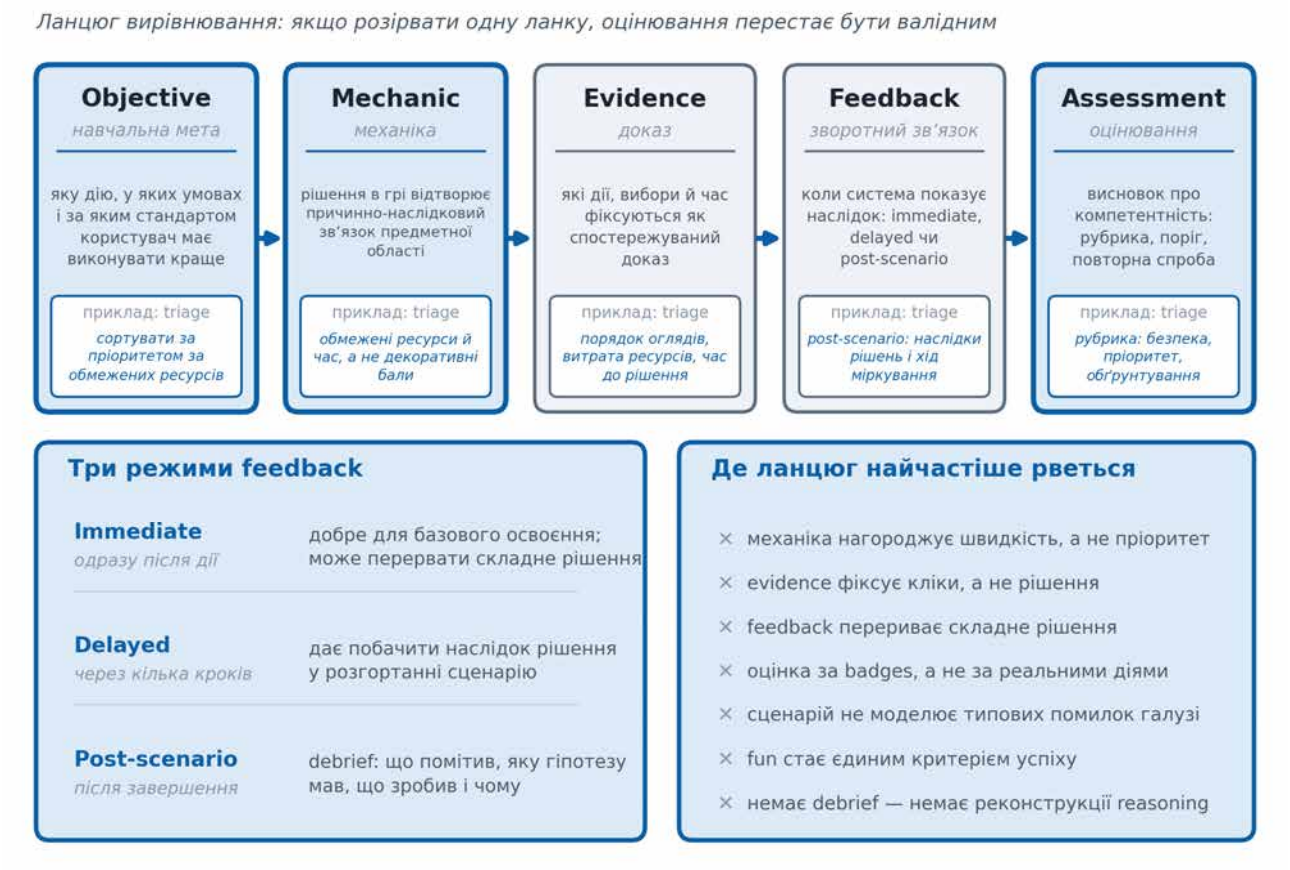


Рис. 5.21. Alignment: learning objective - mechanic - evidence - feedback - assessment

5.8.2 Fidelity, симуляція і межі моделі

Висока візуальна fidelity не гарантує навчальну цінність. Функціональна fidelity відображає ключові дії й причинність; фізична – вигляд і відчуття середовища; психологічна – тиск, відповідальність і контекст рішення. Потрібний рівень залежить від мети.

Кожна симуляція спрощує реальність. Межі моделі треба задокументувати, щоб користувач не сформував помилкову впевненість. Для високоризикових сфер потрібна експертна валідація, контроль версій знань і чітке формулювання, що тренажер не замінює сертифіковану практику.

Сценарії повинні включати нормальні, граничні та помилкові умови. Однак покарання має підтримувати навчання: безпечний failure, пояснення, можливість повтору й поступове зменшення допомоги (табл. 5.31).

Таблиця 5.31

Рівні fidelity у симуляції

Тип fidelity	Що відтворює	Коли критично
Functional	правила, дії, наслідки	майже завжди
Physical	вигляд, scale, controls	equipment/procedure training
Temporal	таймінг і ритм	emergency, coordination
Social	ролі та комунікація	team training
Psychological	тиск і невизначеність	decision-making
Data fidelity	реалістичні діапазони/сигнали	diagnostics, analytics

5.8.3 Feedback, debrief і оцінювання

У serious games feedback може бути immediate, delayed або post-scenario. Immediate підходить для базового освоєння, але може перервати складне рішення. Delayed дозволяє побачити наслідок. Debrief допомагає відновити reasoning: що помітив, яку гіпотезу мав, чому обрав дію, що змінилося.

Score має бути інтерпретованим. Один сумарний бал приховує різні профілі помилок. Краще показати dimensions: accuracy, time, safety, resource use, communication, confidence calibration. Метрика не повинна винагороджувати поведінку, яка суперечить реальній практиці.

Валідність означає, що оцінка справді вимірює заявлену компетентність. Reliability – що результат стабільний у порівнюваних умовах. Для навчального прототипу достатньо явно сформулювати evidence claim і перевірити його з предметним експертом та невеликою групою (табл. 5.32; рис. 5.22).

Таблиця 5.32

Політики навчального feedback

Момент feedback	Перевага	Ризик	Приклад
Immediate	швидке виправлення	залежність від підказок	базова процедура
At checkpoint	не руйнує потік	затримка пояснення	етап складного кейсу
Natural consequence	сильна причинність	може бути неочевидно	симуляція системи
Post-scenario debrief	аналіз reasoning	потребує часу	team/decision training
Adaptive hint	підтримка за потреби	алгоритмічна помилка	progressive mastery

5.8.4 Етика, безпека та робота з даними

Симуляція може впливати на реальні рішення, тому потрібно уникати необґрунтованих claims, прихованого profiling і стигматизації. Дані навчання мають мінімізуватися, зберігатися безпечно й використовуватися лише для заявленої мети.

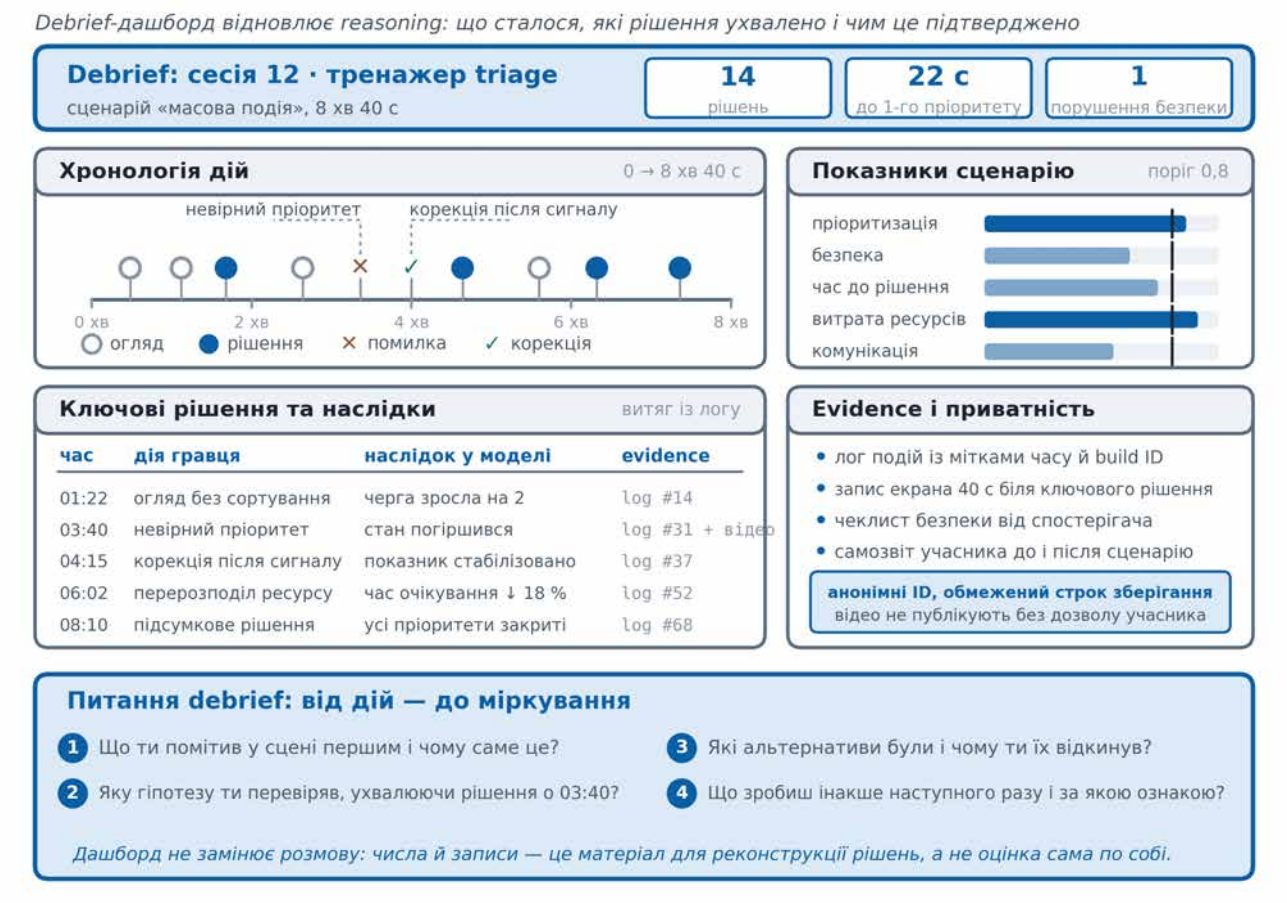


Рис. 5.22. Debrief dashboard із діями, рішеннями та evidence

Для медичних, психологічних або травматичних сценаріїв потрібні content warnings, opt-out, контроль інтенсивності й участь компетентних фахівців. Gamification не повинна тривіалізувати страждання або заохочувати небезпечну поведінку заради score.

Accessibility є частиною валідності: якщо результат залежить від недоречного сенсорного або моторного бар'єра, система вимірює не ту компетентність. Потрібні альтернативні способи вводу, captions, темп, contrast і можливість повтору.

Практичний кейс: serious game concept

Оберіть освітню або професійну задачу. Сформуйте performance objective, domain/error/evidence model, core mechanic, сценарій 10 хвилин, feedback policy, assessment rubric, accessibility та ethics checklist. Перевірте концепцію з предметним експертом (табл. 5.33).

Питання для самоперевірки до розділу 5

1. Чим QA відрізняється від фактичного виконання тестів?
2. Яка різниця між functional test, usability test і playtest?
3. Що має містити якісний bug report?
4. Як severity відрізняється від priority?

5. Що таке regression testing?
6. Як планувати тестове покриття за ризиками?
7. Які метрики плейтесту є корисними без надмірного збору даних?
8. Як поєднати якісні спостереження й telemetry?
9. Що входить до release candidate checklist?
10. Як визначити blocker, critical, major і minor defect?
11. Які ролі та правила потрібні команді з 1-4 осіб?
12. Як backlog, milestone і Definition of Done пов’язані?
13. Що має довести vertical slice?
14. Як побудувати storyboard і shot list для трейлера?
15. Які сім слайдів достатні для короткого pitch deck?
16. Як оформити credits та asset register?
17. Коли gamification допомагає навчанню, а коли шкодить?
18. Які докази компетентності потрібні для різних професійних ролей?
19. Як описати власний внесок у командний проєкт?
20. Які питання повинен містити postmortem?
21. Як побудувати primary і supporting кар’єрні траєкторії?
22. Які критерії визначають готовність навчального проєкту до публікації?

Практичні завдання до розділу 5

Завдання 1. Командний game concept sprint

Команда 1-4 особи обирає жанр, core mechanic, loop і перший рівень; створює конкурентну матрицю, псевдокод, state diagram, три blockout-варіанти, ролі та 5-7-хвилинну презентацію.

Завдання 2. QA та release candidate

Підготуйте test plan, не менше 20 test cases, risk-based regression suite, bug triage, performance smoke test, credits і release checklist. Випустіть RC build і короткі release notes.

Завдання 3. Фінальна демонстрація

Створіть video 60-120 секунд і pitch deck 5-7 слайдів для vertical slice. Додайте storyboard, shot list, asset register, performance evidence, postmortem і портфоліо-опис власної ролі.

Висновки до розділу 5

1. Виробнича зрілість проявляється у здатності не лише створити функцію, а й перевірити, задокументувати, інтегрувати, відтворити, випустити та пояснити її. QA, production і presentation є частинами одного циклу якості.
2. Фінальний vertical slice повинен бути компактним доказом бачення й компетентності. Його сила визначається не масштабом, а узгодженістю механіки, feedback, візуальної системи, performance, тестування та чесно представленого внеску команди.

Графічні матеріали до розділу 5

Таблиця 5.33

Перелік рекомендованих графічних матеріалів до розділу 5

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
5.1	Демонстрація таргет-групи для тестування гри	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
5.2	Мінімальний пайплайн телеметрії: Подія в грі -> Лог/сховище -> Дашборд -> Рішення на ітерацію	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
5.3	Приклад візуальної телеметрії в ігровому рушії	Рекомендовано вставити знімок профайлера або дашборда з виділеним bottleneck, baseline та результатом повторного вимірювання.
5.4	Приклади готових ігор	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
5.5	Структура командного проєкту: концепція - механіка - loop - рівень - тест	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
5.6	Блок-схема core loop командного проєкту	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
5.7	Діаграма станів ключової механіки командного проєкту	Використати UML-подібну діаграму станів із переходами, умовами та пріоритетами; окремо позначити стани помилки та переривання.
5.8	Порівняння трьох top-down blockout-варіантів рівня	Рекомендовано поєднати план зверху та перспективний greybox-кадр, позначити critical path, опційні маршрути, точки інтересу і зони ризику.
5.9	Приклад анотованого moodboard ігрового проєкту	Додати порівняння до/після для readability pass і mood pass; вказати ключове, заповнювальне та контрове світло, експозицію і пост-ефекти.
5.10	Storyboard із шести кадрів для ключової interaction loop	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
5.11	Гібридна схема gameplay flow та сюжетних beats	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.

№ рисунка	Назва	Коротка рекомендація до підготовки зображення
5.12	Структура vertical slice: gameplay, presentation, technology і production	Подати як swimlane-діаграму відповідальностей або карту артефактів із зазначенням власника, входів, виходів і критеріїв готовності.
5.13	Storyboard і shot list фінального трейлера	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
5.14	Приклад структури pitch deck для vertical slice	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
5.15	Зв’язок ігрових механік, навчальної поведінки та компетентностей	Побудувати схему «механіки - динаміки - досвід» із конкретним прикладом та окремими позначками для мотивації, винагороди і зворотного зв’язку.
5.16	Карта прогресу курсу з badges та milestones	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
5.17	Структура портфоліо-кейсу: проблема - процес - доказ - результат	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
5.18	Зв’язок value proposition, core loop і business model	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
5.19	Release pipeline від RC build до store publication	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
5.20	Live Ops loop: планування - release - monitoring - analysis - iteration	Подати як лінійну або циклічну блок-схему з 5-8 етапами, стрілками та короткими підписами; ключові переходи виділити кольором.
5.21	Alignment: learning objective - mechanic - evidence - feedback - assessment	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.
5.22	Debrief dashboard із діями, рішеннями та evidence	Додати високоякісну схему, інфографіку або анотований скріншот (не менше 1600 px по ширині), з короткими підписами та єдиним стилем позначень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Друковані видання та наукові праці

1. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. Boca Raton : CRC Press, 2019. 610 p.
2. Rogers S. Level Up! The Guide to Great Video Game Design. 2nd ed. Chichester : Wiley, 2014. 552 p.
3. Fullerton T. Game Design Workshop: A Playcentric Approach to Creating Innovative Games. 4th ed. Boca Raton : CRC Press, 2018. 522 p.
4. Salen K., Zimmerman E. Rules of Play: Game Design Fundamentals. Cambridge : MIT Press, 2003. 688 p.
5. Adams E., Dormans J. Game Mechanics: Advanced Game Design. Berkeley : New Riders, 2012. 352 p.
6. Swink S. Game Feel: A Game Designer’s Guide to Virtual Sensation. Burlington : Morgan Kaufmann, 2008. 376 p.
7. Totten C. W. An Architectural Approach to Level Design. 2nd ed. Boca Raton : CRC Press, 2019. 474 p.
8. Kremers R. Level Design: Concept, Theory, and Practice. Natick : A K Peters, 2009. 408 p.
9. Nystrom R. Game Programming Patterns. Genever Benning, 2014. 354 p.
10. Gregory J. Game Engine Architecture. 3rd ed. Boca Raton : CRC Press, 2018. 1240 p.
11. Millington I., Funge J. Artificial Intelligence for Games. 3rd ed. Boca Raton : CRC Press, 2016. 898 p.
12. Rabin S. Game AI Pro: Collected Wisdom of Game AI Professionals. Boca Raton : CRC Press, 2014. 598 p.
13. Akenine-Möller T., Haines E., Hoffman N. Real-Time Rendering. 4th ed. Boca Raton : CRC Press, 2018. 1198 p.
14. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. 4th ed. Cambridge : MIT Press, 2023. 1324 p.
15. Hughes J. F. et al. Computer Graphics: Principles and Practice. 3rd ed. Boston : Addison-Wesley, 2013. 1264 p.
16. Williams R. The Animator’s Survival Kit. Expanded ed. London : Faber & Faber, 2009. 392 p.
17. Thomas F., Johnston O. The Illusion of Life: Disney Animation. New York : Disney Editions, 1995. 576 p.
18. Stevens R., Raybould D. The Game Audio Strategy Guide: A Practical Course. 2nd ed. New York : Routledge, 2020. 430 p.
19. Collins K. Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design. Cambridge : MIT Press, 2008. 216 p.
20. El-Nasr M. S., Drachen A., Canossa A. Game Analytics: Maximizing the Value of Player Data. London : Springer, 2013. 792 p.
21. Isbister K. How Games Move Us: Emotion by Design. Cambridge : MIT Press, 2016. 192 p.

22. Norman D. The Design of Everyday Things. Revised and expanded ed. New York : Basic Books, 2013. 368 p.
23. Tidwell J., Brewer C., Valencia A. Designing Interfaces: Patterns for Effective Interaction Design. 3rd ed. Sebastopol : O'Reilly Media, 2020. 600 p.
24. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. Berkeley : New Riders, 2014. 216 p.
25. Lidwell W., Holden K., Butler J. Universal Principles of Design. 2nd ed. Beverly : Rockport Publishers, 2010. 272 p.
26. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
27. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
28. McConnell S. Code Complete. 2nd ed. Redmond : Microsoft Press, 2004. 960 p.
29. Deterding S., Dixon D., Khaled R., Nacke L. From Game Design Elements to Gamefulness: Defining Gamification // Proceedings of MindTrek 2011. Tampere, 2011. P. 9–15.
30. Werbach K., Hunter D. For the Win: How Game Thinking Can Revolutionize Your Business. Philadelphia : Wharton Digital Press, 2012. 148 p.
31. Chou Y.-K. Actionable Gamification: Beyond Points, Badges, and Leaderboards. Milpitas : Octalysis Media, 2015. 514 p.
32. Hunicke R., LeBlanc M., Zubek R. MDA: A Formal Approach to Game Design and Game Research // Proceedings of the AAAI Workshop on Challenges in Game AI. San Jose, 2004. P. 1–5.
33. Schell J. The Art of Game Design: A Deck of Lenses. 3rd ed. Pittsburgh : Schell Games, 2019.
34. Hodent C. The Gamer's Brain: How Neuroscience and UX Can Impact Video Game Design. Boca Raton : CRC Press, 2017. 266 p.
35. Callele D., Neufeld E., Schneider K. Requirements Engineering and the Creative Process in the Video Game Industry // Proceedings of the 13th IEEE International Conference on Requirements Engineering. 2005. P. 240–252.
36. Cooper A., Reimann R., Cronin D., Noessel C. About Face: The Essentials of Interaction Design. 4th ed. Indianapolis : Wiley, 2014. 720 p.
37 Jerald J. The VR Book: Human-Centered Design for Virtual Reality. New York : ACM Books, 2015. 523 p.
38. LaViola J. J. Jr. et al. 3D User Interfaces: Theory and Practice. 2nd ed. Boston : Addison-Wesley, 2017. 624 p.
39. Koster R. A Theory of Fun for Game Design. 2nd ed. Sebastopol : O'Reilly Media, 2013. 300 p.
40. Björk S., Holopainen J. Patterns in Game Design. Hingham : Charles River Media, 2004. 400 p.

Електронні ресурси та офіційна документація

41. Unity Manual. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 11.07.2026).

42. Unity Learn. URL: https://learn.unity.com/ (дата звернення: 11.07.2026).
43. Unity Input System package documentation. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@latest (дата звернення: 11.07.2026).
44. Unity UI Toolkit documentation. URL: https://docs.unity3d.com/Manual/UIElements.html (дата звернення: 11.07.2026).
45. Unity Profiler documentation. URL: https://docs.unity3d.com/Manual/Profiler.html (дата звернення: 11.07.2026).
46. Unity Frame Debugger documentation. URL: https://docs.unity3d.com/Manual/FrameDebugger.html (дата звернення: 11.07.2026).
47. Unity Addressables documentation. URL: https://docs.unity3d.com/Packages/com.unity.addressables@latest (дата звернення: 11.07.2026).
48. Unity Entities documentation. URL: https://docs.unity3d.com/Packages/com.unity.entities@latest (дата звернення: 11.07.2026).
49. Unity Shader Graph documentation. URL: https://docs.unity3d.com/Packages/com.unity.shadergraph@latest (дата звернення: 11.07.2026).
50. Unity Visual Effect Graph documentation. URL: https://docs.unity3d.com/Packages/com.unity.visualeffectgraph@latest (дата звернення: 11.07.2026).
51. Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine (дата звернення: 11.07.2026).
52. Unreal Engine Gameplay Framework. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-framework-in-unreal-engine (дата звернення: 11.07.2026).
53. Unreal Engine Enhanced Input. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/enhanced-input-in-unreal-engine (дата звернення: 11.07.2026).
54. Unreal Motion Graphics UI Designer. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/umg-ui-designer-for-unreal-engine (дата звернення: 11.07.2026).
55. Unreal Engine Niagara. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/niagara-visual-effects-in-unreal-engine (дата звернення: 11.07.2026).
56. Unreal Engine Materials. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-materials (дата звернення: 11.07.2026).
57. Unreal Engine Insights. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-insights-in-unreal-engine (дата звернення: 11.07.2026).
58. Unreal Engine MetaSounds. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/metasounds-in-unreal-engine (дата звернення: 11.07.2026).

59. Unreal Engine Mass Entity. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/mass-entity-in-unreal-engine (дата звернення: 11.07.2026).
60. Game Programming Patterns. URL: https://gameprogrammingpatterns.com/ (дата звернення: 11.07.2026).
61. The Level Design Book. URL: https://book.leveldesignbook.com/ (дата звернення: 11.07.2026).
62. GDC Vault. URL: https://www.gdcvault.com/ (дата звернення: 11.07.2026).
63. Game Accessibility Guidelines. URL: https://gameaccessibilityguidelines.com/ (дата звернення: 11.07.2026).
64. Web Content Accessibility Guidelines Quick Reference. URL: https://www.w3.org/WAI/WCAG22/quickref/ (дата звернення: 11.07.2026).
65. Blender Manual. URL: https://docs.blender.org/manual/en/latest/ (дата звернення: 11.07.2026).
66. Figma Learn. URL: https://help.figma.com/hc/en-us/categories/360002051613-Get-started (дата звернення: 11.07.2026).
67. FMOD Studio Documentation. URL: https://www.fmod.com/docs/2.03/studio/welcome-to-fmod-studio.html (дата звернення: 11.07.2026).
68. Audiokinetic Wwise Documentation. URL: https://www.audiokinetic.com/en/library/ (дата звернення: 11.07.2026).
69. RenderDoc Documentation. URL: https://renderdoc.org/docs/ (дата звернення: 11.07.2026).
70. Steamworks Documentation. URL: https://partner.steamgames.com/doc/home (дата звернення: 11.07.2026).
71. itch.io Creator Documentation. URL: https://itch.io/docs/creators/ (дата звернення: 11.07.2026).
72. Git Documentation. URL: https://git-scm.com/doc (дата звернення: 11.07.2026).
73. Git Large File Storage. URL: https://git-lfs.com/ (дата звернення: 11.07.2026).
74. Perforce Helix Core Documentation. URL: https://help.perforce.com/helix-core/ (дата звернення: 11.07.2026).
75. OpenXR Specification. URL: https://registry.khronos.org/OpenXR/ (дата звернення: 11.07.2026).
76. NVIDIA Nsight Graphics Documentation. URL: https://docs.nvidia.com/nsight-graphics/ (дата звернення: 11.07.2026).
77. Microsoft Game Development Kit documentation. URL: https://learn.microsoft.com/en-us/gaming/gdk/ (дата звернення: 11.07.2026).
78. Google Play Console Help. URL: https://support.google.com/googleplay/android-developer/ (дата звернення: 11.07.2026).
79. Apple Human Interface Guidelines: Games. URL: https://developer.apple.com/design/human-interface-guidelines/playing-games (дата звернення: 11.07.2026).
80. IDEO Design Thinking. URL: https://designthinking.ideo.com/ (дата звернення: 11.07.2026).

ДОДАТОК А.
ВХІДНА ДІАГНОСТИКА ТА ІНДИВІДУАЛЬНА ТРАЄКТОРІЯ

Вхідна діагностика допомагає викладачеві оцінити попередній досвід групи, а студенту - визначити стартові компетентності та обрати пріоритетну траєкторію. Результати не замінюють оцінювання, а використовуються для адаптації складності, формату команд і переліку додаткових матеріалів (табл. А.1) (рис. А.1).

Таблиця А.1

Анкета вхідної діагностики

Блок	**Запитання / поле для відповіді**
Ігрові рушії	Які рушії цікавлять: Unity, Unreal Engine, Godot, інше? З якими вже працювали?
Мови програмування	C#, C++, Python, HLSL/GLSL, інше. Самооцінка C# та C++ за шкалою 1-10.
Gameplay technologies	Gameplay framework, state machines, behavior trees, GOAP, ECS/DOTS, networking, AI.
Rendering / Tech Art	PBR, shaders, render pipeline, VFX, lighting, profiling GPU.
DCC	Blender, Houdini, Substance 3D, Quixel, 3ds Max, Maya, Krita/GIMP.
VCS	Git, GitHub/GitLab, Perforce, Git LFS. Поясніть, що таке VCS.
Досвід	Коротко опишіть проєкт або модель, яку створювали, свою роль і результат.
Debugging	Опишіть послідовність пошуку та усунення помилки у коді або сцені.
Оптимізація	Що означає оптимізація гри? Які метрики та інструменти ви використали б?
Базові поняття	Поясніть game loop, clamp deltaTime, game asset, PBR та три базові шари дизайну гри.
Ігровий досвід	Улюблені ігри; жанри, які не подобаються; пояснення через механіки, loop, UX і стиль.
Траєкторія	A - програмування; B - графіка/Tech Art; C - game/level design та UX; D - моделювання/анімація; E - AI/gameplay.

ДОДАТОК Б.
ШАБЛОН MINI-GDD ТА КОНКУРЕНТНОГО АНАЛІЗУ

Mini-GDD має залишатися коротким, актуальним і придатним для прийняття рішень. Його завдання - узгодити бачення, правила, scope і критерії якості, а не описати кожну можливу ідею (табл. Б.1) (табл. Б.2) (рис. Б.1) (табл. В.1).

Таблиця Б.1

Шаблон Mini-GDD

Поле Mini-GDD	**Зміст**
Робоча назва та logline	1-2 речення: що це, для кого і чому цікаво.
Жанр, платформа, контролер	Цільовий пристрій, тривалість сесії, обмеження.
Persona / аудиторія	Мотивації, бар'єри, досвід, доступність.
Vision та pillars	3-5 принципів, які не можна порушувати.
Core fantasy	Якою роллю або фантазією живе гравець.
Core loop	3-7 повторюваних кроків: дія - feedback - reward - reinvestment.
Meta loop	Довгострокова прогресія, unlocks, розвиток.
Механіки	Дієслово + об'єкт + умова + feedback + параметри.
Правила та стани	Умови дозволу, помилки, перемоги/програшу.
Рівень / content slice	Що входить у прототип і vertical slice.
UI/HUD та accessibility	Екрани, підказки, settings, альтернативні канали.
Art / audio direction	Moodboard, палітра, camera, SFX/ambience, tone.
Технічний план	Рушій, архітектура, ризики, target hardware, budgets.
Критерії успіху	Метрики fun/clarity/performance/stability.
Scope / out of scope	Що свідомо не робимо у цій версії.



Рис. Б.1.1 Односторінковий Mini-GDD

Project “Lantern” 1-Page GDD

Dark fantasy goes low-poly survival



Genre: 3rd person action RPG, single player, follow camera
Target Audience: 13+ (T for Teen), fans of old school adventure games
Controls: Mouse and keyboard
Thematic Setting: Medieval Eastern Europe / Fantasy / Dark
Tech Stack: Unity 2018.3, Audacity, MagicaVoxel, Blender
Platform(s): PC
MVP Game Moment: 2 minutes of linear level flow with basic combat and **one** survival aspect.

Game Summary: Lantern is a dark 3rd person Witcher-style rpg with survival horror elements. The game is set in a sparsely populated wooded region littered with ruins of old and dark mythological beasts. With natural resources and your wit, you must set up your base, plan expeditions into the wild unknown and make sure the darkness doesn't swallow you. *Keep that lantern lit!*

Core Player Experience: Skill-based, high difficulty.
Central Story Theme: Survival
Design Pillar: Meaningful light and darkness, low-poly art style
Remarkability: Dark and mysterious sense of exploration

Anticipated Demo Launch date: End of 2019

Feature Development Priorities:

- Real-time combat
- Inventory system
- Simple base building

Reference products: Cubeworld, Witcher 3, Over The Garden Wall, Symbaroum

Рис. Б.1.2 Односторінковий Mini-GDD

.

Таблиця Б.2

Матриця конкурентного аналізу

Критерій	**Власна концепція**	**Референс 1**	**Референс 2**	**Референс 3**
Аудиторія / платформа				
Core mechanic				
Core loop				
Level structure				
Difficulty / progression				
UI / accessibility				
Art / audio identity				
Production scope				
Відмінність / value proposition				

ДОДАТОК В.
ШАБЛОН ДОКУМЕНТА ДИЗАЙНУ РІВНЯ

Таблиця В.1

Шаблон Level Design Document

Розділ LDD	**Питання для заповнення**
Мета рівня	Якої механіки, навички або сюжетного стану навчає рівень?
Вхідний стан гравця	Що вже знає, які ресурси та здібності має?
Вихідний стан	Що повинен освоїти, отримати або зрозуміти?
Critical path	Основний маршрут, контрольні точки, орієнтовний час.
Optional paths	Ризик/нагорода, секрети, shortcuts, exploration.
Beat map	Послідовність тиску, паузи, кульмінації, recovery.
Encounter / puzzle sheets	Вороги, ресурси, правила, telegraphing, fail states.
Metrics	Ширина проходів, висоти, jump distance, cover spacing, camera clearance.
Signposting	Landmarks, lighting, composition, audio cues, UI prompts.
Accessibility	Кольори, навігація, складність, альтернативні підказки.
Art / lighting pass	Стиль, матеріали, композиція, readability budget.
Performance budget	Triangles, draw calls, lights, VFX, memory, streaming.
Playtest plan	Завдання, метрики, сценарії, критерії прийняття.
Iteration log	Гіпотеза - зміна - результат - рішення.



Рис. В.1. Top-down карта рівня з beat map

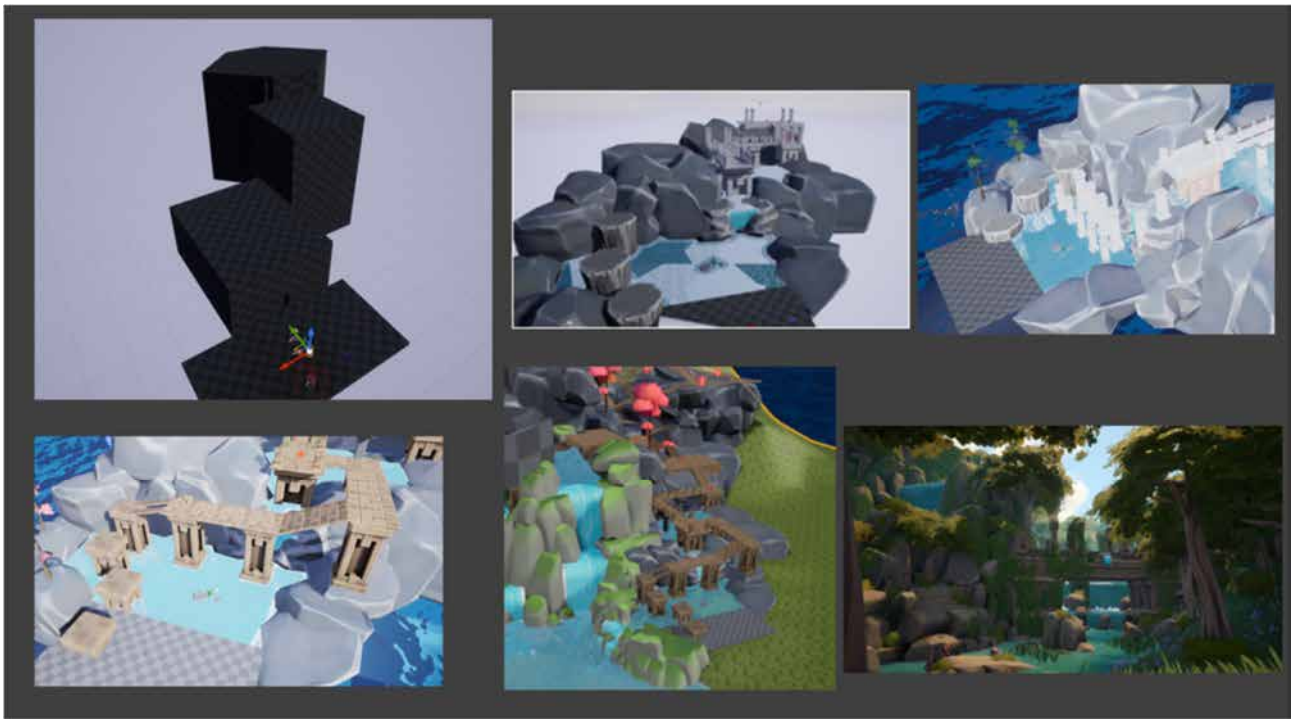


Рис. В.2. Порівняння шістьох blockout-ітерацій

ДОДАТОК Г.
ШАБЛОНИ INTERACTION, VFX, AUDIO ТА ANIMATION BREAKDOWN

Таблиця Г.1

Interaction breakdown

Поле interaction breakdown	Зміст
Назва взаємодії	Дієслово, об’єкт і контекст.
Вхід	Action, device, context, rebinding, dead zone.
Умови дозволу	Range, state, inventory, cooldown, line of sight.
State flow	Idle -> anticipation -> active -> recovery / fail / interrupt.
Feedback stack	Animation, VFX, SFX, UI, camera, haptics.
Параметри	Duration, speed, damage, radius, intensity, curves.
Failure states	Що сталося, як система пояснює відмову.
Accessibility	Alternatives, intensity, remapping, captions.
Test cases	Normal, edge, rapid repeat, interruption, low FPS.

Таблиця Г.2

Мультимодальний feedback breakdown

Фаза	VFX	Audio	Animation	Camera/UI	Таймінг
Anticipation					
Impact / active					
Recovery / tail					
Fail / cancel					

Таблиця Г.3

Audio event map

Audio event	Trigger	Layers	Spatial	RTPC / parameter	Bus / priority
Input confirm					
Main action					
Impact					
Fail					
Ambience zone					

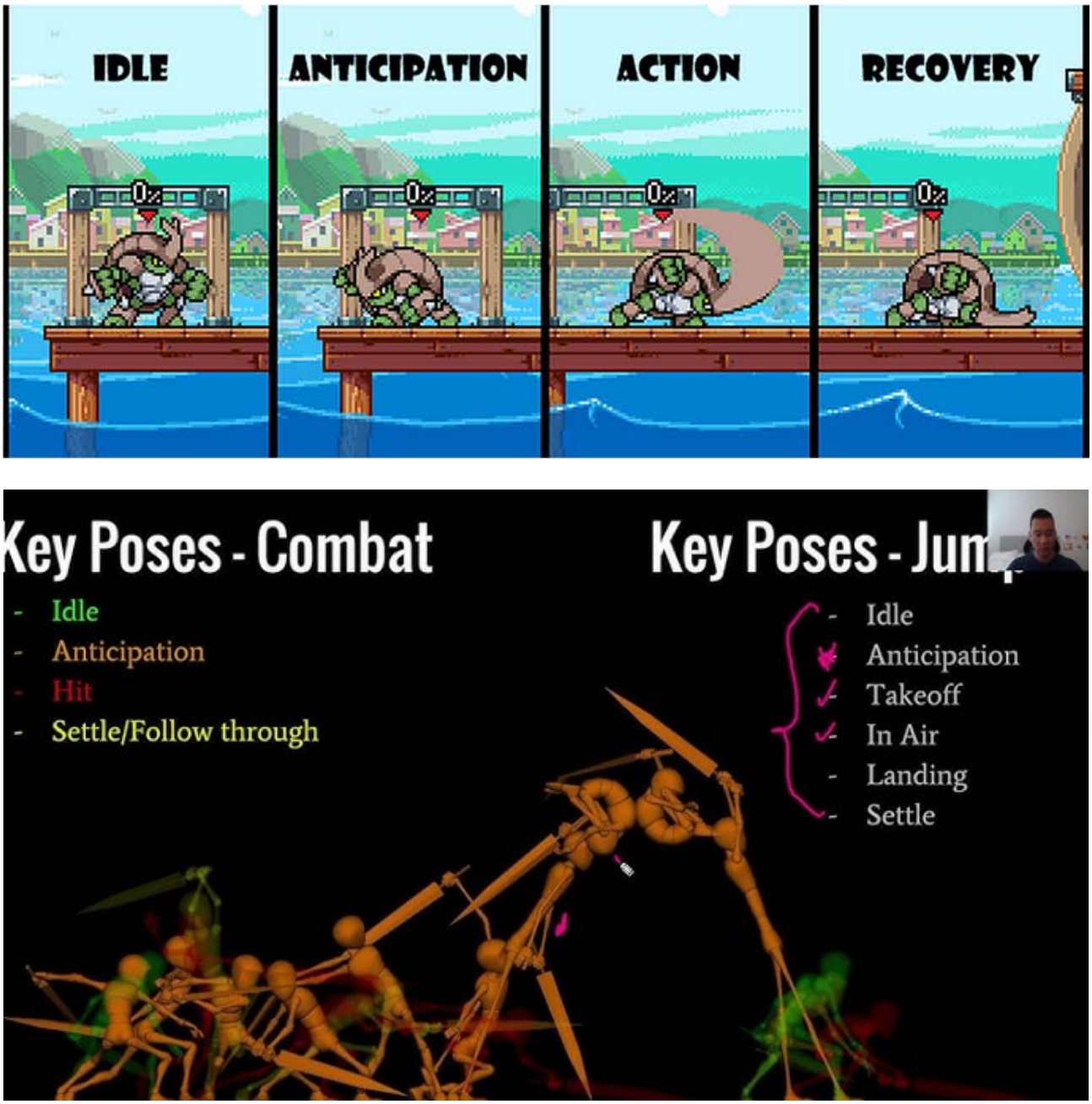


Рис. Г.1. Таймлайн anticipation - impact - recovery

ДОДАТОК Д.
ШАБЛОН ЗВІТУ З ПРОФІЛЮВАННЯ ТА ОПТИМІЗАЦІЇ

Таблиця Д.1

Performance report: baseline - change - result

Поле	**Що зафіксувати**
Target	Платформа, hardware, resolution, quality preset, target FPS/frame time.
Build / scene	Версія, commit, рівень, сценарій, тривалість capture.
Baseline	Average, 1% low, CPU/GPU frame, memory, spikes.
Bottleneck	Симптом, evidence, marker/call stack/pass.
Hypothesis	Чому саме це створює витрати.
Change	Одна або контрольована група змін.
Result	Повторний capture за тим самим сценарієм.
Visual / gameplay impact	Чи змінилася якість або поведінка.
Decision	Keep, revise, revert; наступний крок.

Таблиця Д.2

Таблиця технічних бюджетів

Бюджет	**Ціль**	**Фактичне значення**	**Статус / дія**
CPU frame time			
GPU frame time			
Draw calls / batches			
Triangles / visible objects			
Texture memory			
Audio voices			
VFX particles / overdraw			
Physics / AI agents			
Loading / streaming			

МІСЦЕ ДЛЯ РИСУНКА
Профайлер до і після оптимізації

Рис. Д.1. Профайлер до і після оптимізації

Рекомендація до зображення: Вставити два captures з однаковою шкалою; виділити marker або render pass, додати delta у мілісекундах і короткий висновок.

ДОДАТОК Е.
ФІНАЛЬНИЙ VERTICAL SLICE, ТРЕЙЛЕР І PITCH DECK

Таблиця Е.1

Чек-лист фінального vertical slice

Етап	Deliverable	Definition of Done
Concept	logline, persona, value, pillars	зрозуміло за 30 секунд
Core	playable mechanic + loop	feedback, fail, reset, metrics
World & FX	level slice, lighting, VFX, audio	читабельно і стабільно
Interaction & UX	>=3 interactions, onboarding, settings	пройдений UX-test
Performance	profile report + 2 fixes	цільовий frame time
QA	test plan, bug triage, release candidate	0 blocker / critical
Capture	video 60-120 s	1080p, стабільний FPS, чистий звук
Pitch	5-7 slides	ідея, аудиторія, loop, look, result, next steps
Credits	asset register, licenses, team roles	повна атрибуція

Таблиця Е.2

Shot list для демонстраційного відео

№	Сцена / кадр	Кадрування / camera	Дія	Тривалість	Assets / VFX / SFX	Статус
1	Hook			0-3 с		
2	Context			3-10 с		
3	Core mechanic			10-30 с		
4	Escalation			30-55 с		
5	Payoff			55-75 с		
6	Title / CTA			5-10 с		

Таблиця Е.3

Структура pitch deck

Слайд	Зміст
1	Title + logline + ключовий кадр.
2	Аудиторія, проблема, ціннісна пропозиція.
3	Core loop / key feature diagram.
4	Look & feel: moodboard, palette, camera, audio tone.
5	Демонстраційна структура / architecture / level slice.
6	Результати: playtest, performance, learnings.

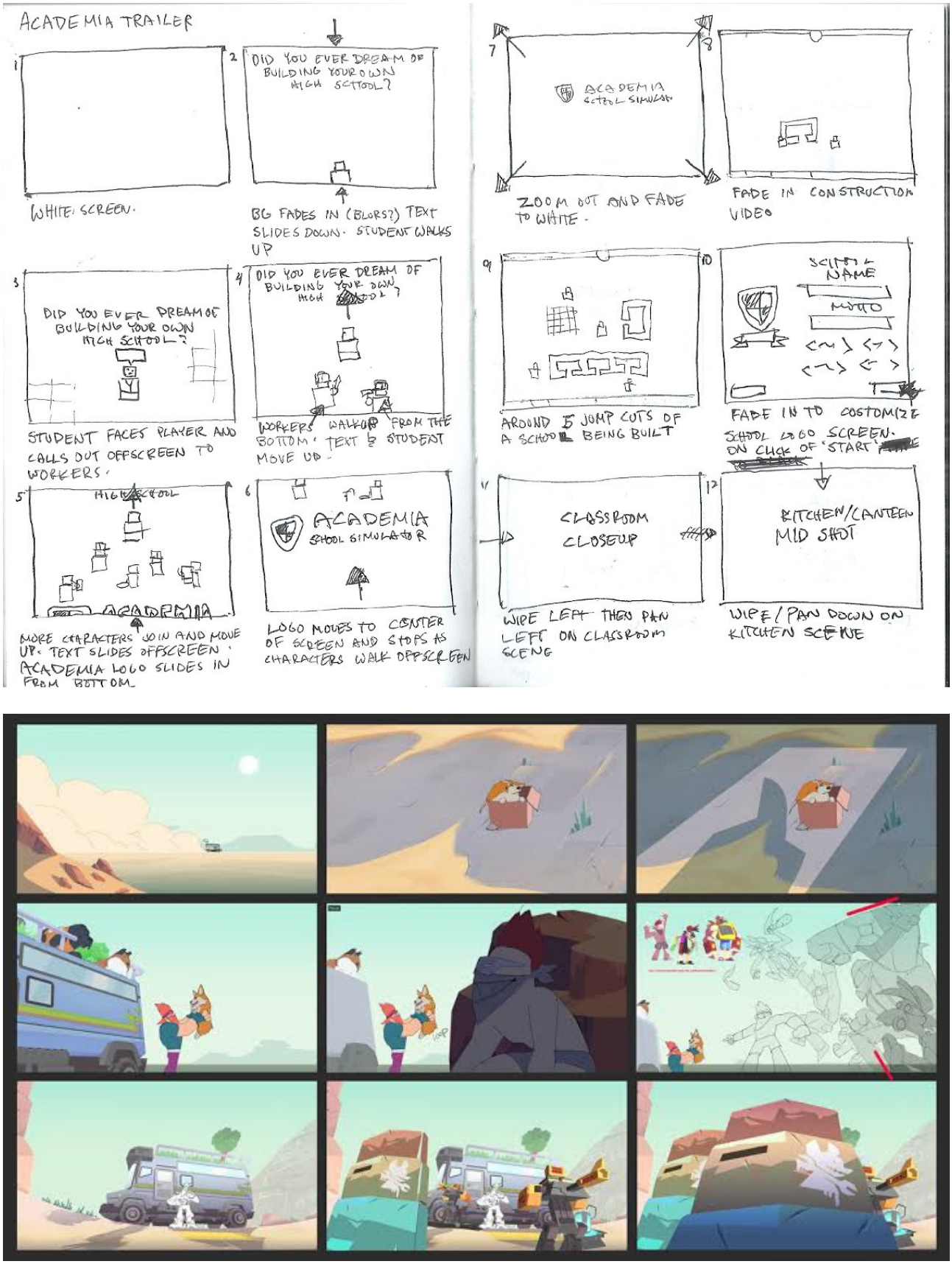


Рис. Е.1. Storyboard трейлера або демонстрації

ДОДАТОК Ж.
ШАБЛОН ПОСТМОРТЕМУ ТА САМООЦІНЮВАННЯ

Таблиця Ж.1

Шаблон postmortem

Питання	**Відповідь / доказ**
Що планували створити?	
Яка головна гіпотеза?	
Що фактично завершено?	
Які дані або плейтести це підтверджують?	
Що спрацювало найкраще і чому?	
Що не спрацювало і чому?	
Який технічний/дизайнерський борг залишився?	
Яке рішення варто повторити?	
Яку одну практику змінити в наступному проєкті?	
Який матеріал додати до портфоліо?	

Таблиця Ж.2

Матриця підсумкового самооцінювання

Критерій	**1 - початковий**	**2 - базовий**	**3 - достатній**	**4 - впевнений**	**5 - професійний доказ**
Game design / systems					
Programming / architecture					
Level / visual design					
Interaction / UX / accessibility					
Audio / animation / feedback					
Performance / QA					
Production / documentation					
Presentation / portfolio					

ДОДАТОК З.
КРИТЕРІЇ ОЦІНЮВАННЯ ТЕМАТИЧНИХ І ПРОЄКТНИХ РОБІТ

Рубрики призначені для прозорого оцінювання результату, процесу й доказів. Викладач може змінювати ваги відповідно до освітньої програми, але критерії мають бути відомі до початку роботи. Високий рівень передбачає не лише наявність функції, а й обґрунтування, перевірку, документацію та відповідність обмеженням (табл. З.1) (табл. З.2) (табл. З.3) (табл. З.4) (табл. З.5).

Таблиця З.1

Рубрика до розділу 1: концепція та ігрові механіки

Критерій	**Вага**	**90-100%: високий рівень**	**75-89%: достатній**	**60-74%: базовий**	**<60%: потребує доопрацювання**
Концепція та аудиторія	15%	logline, persona, value і pillars узгоджені та підтверджені дослідженням	концепція зрозуміла, але evidence обмежене	є загальна ідея без чіткої аудиторії	ідея суперечлива або не сформульована
Mechanics / rules	20%	механіки формалізовані, параметризовані, мають fail/feedback	основна механіка працює, частина edge cases не описана	опис є, але правило або feedback нечіткі	механіка не відтворюється
Core / meta loop	20%	цикл замкнений, reward і reinvestment логічні	core loop зрозумілий, meta loop частковий	цикл описаний без стійкої мотивації	набір дій без циклу
Прототип і тест	25%	playable proof, гіпотези, метрики, 2+ ітерації	прототип працює, тест обмежений	демо є без системного тесту	немає перевірного прототипу
Документація	10%	Mini-GDD актуальний, діаграми й backlog пов’язані	основні артефакти наявні	документи неповні або застарілі	артефакти відсутні
Презентація	10%	чіткий pitch, візуальні докази, рефлексія	результат зрозумілий	надмірний текст або мало evidence	результат неможливо оцінити

Таблиця 3.2

Рубрика до розділу 2: рівень і візуальна система

Критерій	**Вага**	**90-100%: високий рівень**	**75-89%: достатній**	**60-74%: базовий**	**<60%: потребує доопрацювання**
Level flow	20%	critical/optional paths, beats, metrics і signposting перевірені	маршрут читабельний, окремі beats слабкі	blockout функціональний без чіткої прогресії	гравець губиться або рівень не прохідний
Візуальна hierarchy	15%	silhouette, contrast, landmarks і camera узгоджені	більшість інформації читається	помітні конфлікти стилю/контрасту	ключові об’єкти не розпізнаються
UI/HUD	15%	states, focus, platform input і accessibility реалізовані	основний flow працює	статичний UI з обмеженим feedback	UI заважає виконанню задачі
Materials / lighting	15%	цілісна PBR/stylized система та readability pass	стиль переважно узгоджений	окремі коректні матеріали без системи	технічні/візуальні помилки домінують
VFX / animation	15%	timing і feedback синхронні з gameplay	ефекти читаються, але не всі фази завершені	ефект декоративний або нестабільний	ефект маскує стан або відсутній
Pipeline / budget	10%	naming, import, LOD, validation і profiler evidence	основні правила дотримані	частина асетів не стандартизована	pipeline хаотичний
Breakdown	10%	before/after, параметри, рішення й learnings	структура зрозуміла	опис поверхневий	відсутній

Таблиця 3.3

Рубрика до розділу 3: interaction та game feel

Критерій	**Вага**	**90-100%: високий рівень**	**75-89%: достатній**	**60-74%: базовий**	**<60%: потребує доопрацювання**
Responsiveness	20%	first feedback виміряно, latency і tunables обґрунтовані	керування responsive у типових умовах	відчутна затримка або нестабільність	дії не контролюються
State logic	15%	дозволи, fail, interrupts і priorities формалізовані	основні стани працюють	edge cases спричиняють помилки	стани суперечливі
Camera / controller	15%	framing, collision, smoothing і device tests завершені	система придатна для основного сценарію	часті camera/control artifacts	проходження ускладнене
Feedback stack	20%	animation, VFX, audio, UI, camera/haptics узгоджені	основні канали синхронні	feedback неповний або шумний	результат дії не зрозумілий
Onboarding / accessibility	15%	progressive disclosure, settings і test evidence	основні підказки та опції є	підказки перевантажені або неповні	новий користувач не розуміє дій
Telemetry / test	10%	події, сценарії, A/B і висновок	тест проведено	дані не пов’язані з рішенням	перевірка відсутня
Документація	5%	interaction/audio/animation breakdown повний	основні схеми є	неактуальні матеріали	відсутня

Таблиця 3.4

Рубрика до розділу 4: програмні системи та продуктивність

Критерій	**Вага**	**90-100%: високий рівень**	**75-89%: достатній**	**60-74%: базовий**	**<60%: потребує доопрацювання**
Architecture	20%	модулі, interfaces, data flow і ownership чіткі	архітектура підтримує основний scope	висока зв’язність або дублювання	зміна однієї функції ламає систему
Code / Blueprints quality	15%	читабельність, naming, errors, tests і review	працездатно з незначним боргом	hard-coded і слабка обробка помилок	нестабільний або нечитабельний код
Data-driven / tools	15%	tunables, assets, validation і tool UX	параметри винесені частково	налаштування потребують правки логіки	дані не керовані
VCS / build	15%	branches, LFS, commits, CI/build і README дисципліновані	репозиторій робочий	конфлікти/великі файли керуються вручну	немає відтворюваної збірки
Profiling	20%	baseline, bottleneck, change, repeat capture, висновок	є evidence однієї оптимізації	оптимізація без контрольного виміру	профілювання відсутнє
Stability	10%	edge/load/network tests, graceful failures	основний сценарій стабільний	помітні повторювані дефекти	критичні падіння/втрата даних
Technical explanation	5%	рішення пропорційні scope і ясно захищені	пояснення достатнє	перелік технологій без причин	рішення не обґрунтовані

Таблиця 3.5

Підсумкова рубрика vertical slice

Критерій	**Вага**	**90-100%: високий рівень**	**75-89%: достатній**	**60-74%: базовий**	**<60%: потребує доопрацювання**
Цілісність vertical slice	20%	core promise, mechanics, world, feedback і UX узгоджені	демо цілісне з окремими слабкими місцями	функції існують окремо	немає завершеного сценарію
Якість і стабільність	15%	RC build, 0 blocker/critical, контроль edge cases	основний сценарій стабільний	помітні major defects	build не запускається або втрачає прогрес
QA / playtest evidence	15%	test plan, triage, telemetry, iterations	тестування проведено	мало сценаріїв і даних	QA відсутнє
Performance	10%	target досягнуто, 2+ підтверджені fixes	target здебільшого досягнуто	нестабільний frame time	непридатна продуктивність
Production / командна робота	10%	ролі, board, risks, handoff і postmortem	робота організована	відповідальність нечітка	артефакти/внесок не визначені
Трейлер / pitch	10%	hook, gameplay evidence, rhythm, clean audio, 5-7 slides	демонстрація зрозуміла	надмірний montage або текст	не показує продукт
Ліцензії / ethics / accessibility	10%	asset register, credits, privacy та settings повні	базові вимоги виконані	частина джерел/опцій не оформлена	ризик порушення прав або безпеки
Портфоліо / рефлексія	10%	роль, process, evidence, result, learnings чіткі	внесок описано	показано лише фінальний кадр	авторство й результат незрозумілі

МІСЦЕ ДЛЯ РИСУНКА

Матриця оцінювання результату, процесу та evidence

Рис. 3.1. Матриця оцінювання результату, процесу та evidence

Рекомендація до зображення: Подати radar або heatmap за п’ятьма блоками: design, implementation, quality, production, presentation; окремо показати ваги й мінімальні пороги.

ДОДАТОК И.
ОПЕРАЦІЙНІ ЧЕК-ЛИСТИ ДЛЯ ЩОДЕННОЇ РОБОТИ

Чек-листи не замінюють професійне рішення, але зменшують кількість повторюваних помилок. Їх доцільно адаптувати до конкретного engine, платформи та Definition of Done, зберігати у репозиторії й оновлювати після postmortem (табл. И.1) (табл. И.2) (табл. И.3) (табл. И.4) (табл. И.5).

Таблиця И.1

Чек-лист перед commit і code review

Перед commit / merge request	**Так/ні**
Зміна має одну зрозумілу мету, зайві файли не включено	
Проєкт відкривається без missing references/errors	
Naming, folders і formatting відповідають правилам	
Великі бінарні файли використовують LFS/Perforce	
Додано або оновлено test / validation	
Перевірено основний і edge-сценарій	
Не додано secrets, personal data, cache або generated files	
Commit message пояснює що і чому	
Документація/README/diagram актуальні	
Відомі ризики описані у merge request	

Таблиця И.2

Чек-лист проведення плейтесту

Перед плейтестом	**Так/ні**
Сформульована одна головна гіпотеза й 2-4 питання	
Визначені учасники та критерії відбору	
Build, hardware, save/reset і recording перевірені	
Підготовлено consent/privacy notice за потреби	
Фасилітатор не підказує рішення	
Фіксуються дії, час, помилки, цитати й контекст	
Є резервний сценарій при технічній помилці	
Після сесії проводиться коротке інтерв’ю	
Дані анонімізуються й зберігаються обмежений час	
Заплановано synthesis і рішення, а не лише збір вражень	

Таблиця И.3

Чек-лист release candidate

Перед build / release candidate	**Так/ні**
Version, branch, commit і changelog зафіксовані	
Clean build відтворено на окремій машині/agent	
Target settings, quality, resolution і input перевірені	
Save/load, pause, quit, suspend/resume працюють	
Network/offline/reconnect сценарії перевірені	
Performance smoke test пройдено	
0 blocker/critical; major мають рішення або waiver	
Ліцензії, credits, privacy, ratings і accessibility готові	
Crash/logging/support channel активні	
Rollback/backup і відповідальний за випуск визначені	

Таблиця И.4

Чек-лист capture і трейлера

Перед записом демонстрації	**Так/ні**
Storyboard/shot list і цільовий хронометраж затверджені	
Golden path відтворюється без випадкових збоїв	
Debug overlays, курсор, personal data вимкнені	
Resolution, FPS, codec, audio levels протестовані	
Камера, lighting, UI safe area й subtitles перевірені	
Усі потрібні SFX/music мають ліцензії	
Записано кілька takes і clean plates	
Титри читаються 4-5 секунд, contrast достатній	
Монтаж показує gameplay evidence, а не лише cinematic	
Фінальний файл переглянуто на іншому пристрої	

Таблиця И.5

Чек-лист командного handoff

Перед передаванням задачі іншому учаснику	Так/ні
Описано мету, контекст, owner і deadline	
Надані source files, branch, scene і dependencies	
Є Definition of Done і visual/technical references	
Відомі обмеження та edge cases зафіксовані	
Пояснено, що не входить до scope	
Є test steps і очікуваний результат	
Назви параметрів/подій/API узгоджені	
Передбачено review/feedback point	
Зміни інтегруються без ручних прихованих кроків	
Handoff завершено короткою перевіркою розуміння	

Таблиця И.6

Чек-лист портфоліо-кейсу

Перед публікацією портфоліо-кейсу	Так/ні
Контекст, задача, обмеження й власна роль чіткі	
Показано process і ключові рішення, а не лише результат	
Є діаграми, breakdown, metrics або before/after evidence	
Командний внесок інших людей зазначено	
Видалено confidential/personal/internal data	
Перевірено права на screenshots, music, assets і logos	
Відео коротке, стабільне й демонструє функцію	
Repository/build має README і працює	
Опис містить learnings і наступні кроки	
Контакти та посилання актуальні	

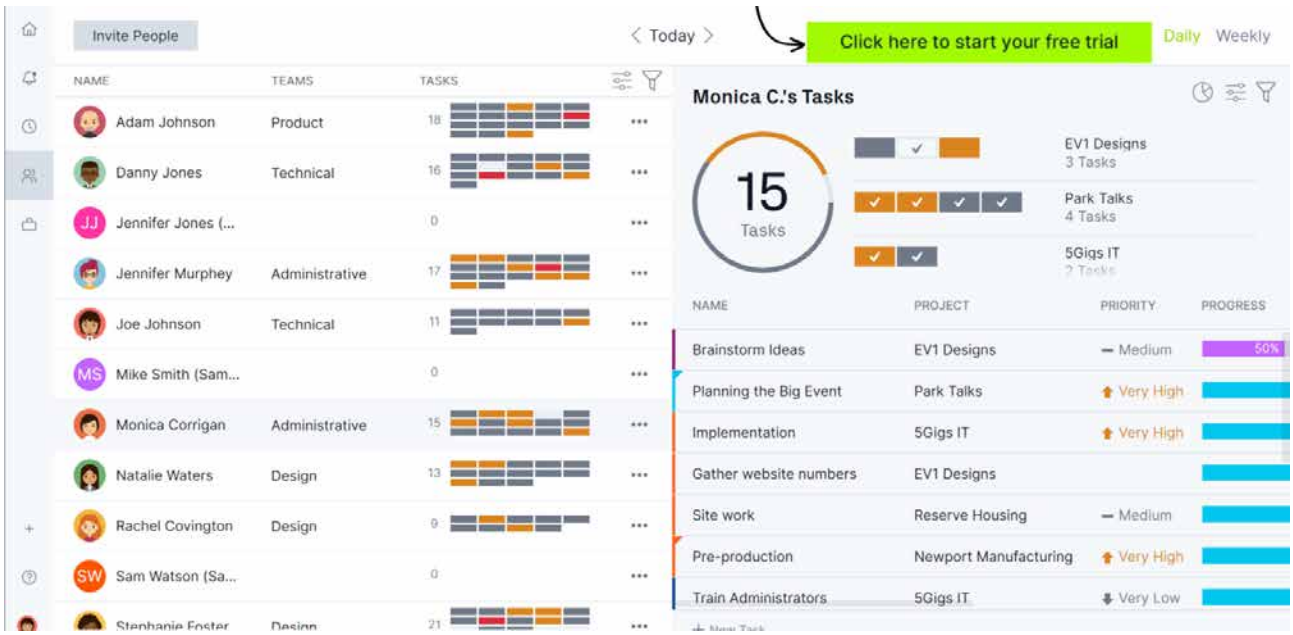


Рис. И.1. Щоденний production checklist у вигляді Kanban

ГЛОСАРІЙ

Глосарій узгоджує українські та англомовні терміни, які використовуються у геймдизайні, програмуванні, Tech Art, UX, QA та production. У межах конкретного проєкту команда може доповнювати цей перелік власним словником і правилами найменування (табл. Гл.1).

Таблиця Гл.1

Основні терміни розробки ігрових додатків

Термін	Визначення
AAA	умовна категорія високобюджетних ігор із великими командами, тривалим виробництвом і значними вимогами до якості.
Ability	дія або здібність персонажа з умовами активації, станами, параметрами, feedback і cooldown.
Acceptance criteria	перевірні умови, за яких задача або функція вважається завершеною.
Accessibility	проєктування гри так, щоб нею могли користуватися люди з різними сенсорними, моторними й когнітивними потребами.
Action mapping	абстракція вводу, що пов’язує ігрову дію з клавішею, кнопкою, жестом або іншим контролером.
Addressables	система адресного керування контентом і асинхронного завантаження в Unity.
Affordance	властивість або сигнал об’єкта, що підказує можливу дію.
AI agent	керований алгоритмом суб’єкт, який сприймає стан середовища та обирає дії.
Alpha	версія гри, у якій основні функції наявні, але контент і якість ще не стабілізовані.
Ambience	фоновий звуковий шар, що формує відчуття простору, часу і настрою.
Animation blend	плавне змішування кількох анімацій за вагою або параметрами.
Animation notify	подія на часовій шкалі анімації для запуску логіки, SFX, VFX або hit window.
Anticipation	підготовча фаза дії, яка робить наступний результат читабельним.
Archetype	узагальнений тип гравця, персонажа або ігрової ролі з характерними мотивами й поведінкою.
Art bible	документ із візуальними принципами, палітрою, формами, матеріалами, масштабами й правилами стилю.
Asset	цифровий ресурс проєкту: модель, текстура, звук, анімація, шрифт, скрипт, prefab, blueprint тощо.
Asset bundle / pak	пакет підготовленого контенту для завантаження або доставки у build.
Backlog	впорядкований список задач, ризиків, дефектів і покращень проєкту.

Продовження табл. Гл.1

Термін	Визначення
Balancing	налаштування числових параметрів і правил для досягнення бажаної складності, темпу та справедливості.
Behavior Tree	ієрархічна модель вибору поведінки AI через вузли умов, послідовностей і пріоритетів.
Beta	версія з повним або майже повним контентом, що проходить стабілізацію, баланс і широке тестування.
Billboard	площина або об’єкт, який орієнтується до камери й використовується для частинок, рослинності чи далеких деталей.
Blockout / greybox	рання геометрична версія рівня без фінального арту, призначена для перевірки масштабу, маршруту й механік.
Bloom	пост-ефект світіння яскравих ділянок зображення.
Blueprint	система візуального програмування Unreal Engine; також загальна назва схеми або шаблону.
Bottleneck	компонент, що найбільше обмежує продуктивність або швидкість виробництва.
Branch	окрема гілка історії змін у системі контролю версій.
Build	зібрана виконувана версія гри для конкретної платформи та конфігурації.
Build pipeline	автоматизована або напівавтоматизована послідовність імпорту, компіляції, пакування, перевірки та публікації build.
Camera rig	набір об’єктів, правил і параметрів, що визначає поведінку ігрової камери.
Cancel window	часовий інтервал, у якому поточну дію дозволено перервати іншою.
Checkpoint	стан або місце, з якого гравець відновлює проходження.
Cinematic	постановочний фрагмент із контрольованою камерою, анімацією, монтажем і звуком.
Clipping	в аудіо - перевищення допустимого рівня сигналу; у графіці - відсікання геометрії або вихід за межі камери.
Collider	геометричне представлення об’єкта для фізичних зіткнень і запитів.
Color grading	корекція кольору, контрасту й тону кадру для єдиної візуальної атмосфери.
Commit	зафіксований набір змін у репозиторії з ідентифікатором і повідомленням.
Component	модуль даних або поведінки, який додається сутності чи об’єкту.
Compression	стиснення даних; в аудіо також обробка динамічного діапазону.

Продовження табл. Гл.1

Термін	Визначення
Concept art	візуальне дослідження ідеї, форми, середовища, персонажа або настрою до виробництва фінального контенту.
Context	умови, у яких відбувається дія: стан гри, місце, пристрій, аудиторія, попередні події.
Core fantasy	роль або уява, яку гра дозволяє систематично переживати гравцеві.
Core loop	короткий повторюваний цикл основних дій, feedback, винагороди та реінвестування.
Coyote time	короткий додатковий час після сходу з краю, протягом якого гра ще дозволяє стрибок.
Critical path	основний маршрут або послідовність задач, необхідна для завершення рівня чи milestone.
Crossfade	плавний перехід між двома анімаціями, аудіошарами, станами або зонами.
Culling	відсікання невидимих або непотрібних для рендерингу об’єктів.
Curve	функція зміни параметра в часі або залежно від іншої величини.
Dead zone	діапазон слабкого сигналу аналогового вводу, який ігнорується для усунення дрейфу.
Debug build	збірка з діагностичними символами, логами, меню й додатковими перевірками.
Debug HUD	екранний інтерфейс для перегляду станів, параметрів, метрик і подій під час розробки.
Definition of Done	узгоджений перелік вимог до завершеності задачі або артефакту.
Delta time	час між двома кадрами або оновленнями, який використовують для незалежного від FPS руху й симуляції.
Design pillar	короткий принцип, що спрямовує рішення й захищає ідентичність гри.
Diegetic UI	інтерфейс, який існує всередині ігрового світу й сприймається персонажами.
Difficulty curve	зміна вимог, ризику й складності впродовж проходження.
Draw call	команда рендерингу на відображення групи геометрії з певним станом матеріалу.
Ducking	тимчасове зменшення гучності одного аудіоканалу, щоб звільнити місце для важливішого.
Dynamic batching	об’єднання невеликих об’єктів для зменшення кількості draw calls.
Dynamics	у MDA - поведінка системи, яка виникає під час взаємодії механік і гравця.

Продовження табл. Гл.1

Термін	Визначення
ECS	архітектура Entity Component System, що розділяє ідентичність, дані та системи обробки.
Emitter	джерело частинок, звуку або іншого потокового ефекту.
Encounter	спроєктована ситуація випробування: бій, головоломка, traversal або соціальна взаємодія.
Event bus	механізм публікації та підписки на події між слабко пов’язаними системами.
Exposure	параметр яскравості й адаптації камери до освітлення.
Fail state	стан, у якому дія або задача не виконана; має містити зрозумілий feedback і шлях відновлення.
Feedback loop	цикл «дія - реакція системи - сприйняття - наступне рішення».
First feedback	перший відчутний сигнал після вводу, який підтверджує прийняття команди.
Fixed timestep	сталий крок симуляції, зазвичай фізики, незалежний від частоти рендерингу.
Flow	стан зосередженого залучення, коли виклик і навички узгоджені.
Fog of war	система приховування невідомих або невидимих ділянок ігрового простору.
Frame budget	максимальний час одного кадру для цільового FPS: наприклад, 16,67 мс для 60 FPS.
Frame pacing	рівномірність часу кадрів, що впливає на плавність сильніше за середній FPS.
FTUE	перший досвід користувача: запуск, пояснення керування, ранні цілі та формування довіри.
Game feel	сукупне відчуття точності, ваги, ритму й задоволення від керування та feedback.
Game loop	основний цикл оновлення гри: ввід, симуляція, логіка, рендеринг, звук.
Game state	цілісний режим або стан гри: меню, gameplay, pause, victory, loading тощо.
GDD	Game Design Document - набір актуальних матеріалів про бачення, механіки, правила, контент і критерії.
Gameplay tag	семантична мітка для станів, характеристик, подій і правил взаємодії.
Gaze interaction	взаємодія, у якій напрям погляду використовується як вказівник або сигнал уваги.
Git LFS	розширення Git для зберігання великих бінарних файлів поза звичайною історією репозиторію.

Продовження табл. Гл.1

Термін	Визначення
GOAP	Goal-Oriented Action Planning - планування AI через цілі, дії, передумови та наслідки.
Golden path	рекомендований сценарій проходження, що демонструє продукт у найкращих умовах.
GPU overdraw	повторне зафарбовування тих самих пікселів, особливо дороге для прозорих ефектів.
Greybox	див. Blockout; функціональна просторова модель рівня з простих форм
Haptics	тактильний feedback через вібрацію, опір, імпульс або інші фізичні сигнали.
Headroom	запас рівня аудіосигналу до кліпінгу або ресурсний запас системи до ліміту.
Hit box	область, яка визначає зону атаки або контакту.
Hit stop	дуже коротка пауза або уповільнення під час удару для підсилення impact.
Hit window	часовий інтервал, у якому атака може завдати шкоди.
HUD	постійний або контекстний екранний інтерфейс із критичною ігровою інформацією.
Input buffer	збереження вводу на короткий час, щоб виконати команду, коли система стане готовою.
Input context	набір активних дій і мапінгів для конкретного режиму або стану.
Input latency	час від фізичної дії до видимого, звукового або тактильного результату.
Instance	окремий екземпляр об’єкта, матеріалу або сутності зі спільним шаблоном.
Iteration	цикл створення, перевірки, аналізу й корекції рішення.
Juice	сукупність додаткових анімацій, звуків, VFX, camera effects і мікровідгуків, що підсилюють дію.
Landmark	помітний об’єкт, який підтримує орієнтацію, композицію та запам’ятовування простору.
Latency	затримка між подією та реакцією системи.
Level streaming	динамічне завантаження й вивантаження частин світу.
Lightmap	заздалегідь обчислена текстура освітлення для статичної геометрії.
Live Ops	підтримка гри після релізу через події, контент, баланс, аналітику й операційні процеси.
LOD	рівень деталізації моделі або ефекту, який змінюється залежно від відстані чи бюджету.
Logline	одне-два речення, що передають ідею, роль гравця й унікальність продукту.

Продовження табл. Гл.1

Термін	Визначення
Look development	пошук і фіксація матеріалів, освітлення, кольору та фінального вигляду сцени.
Loop	повторювана послідовність дій і наслідків.
Material	набір шейдерів, текстур і параметрів, що визначає візуальну реакцію поверхні.
Mechanic	відтворюване правило взаємодії: дія, об’єкт, умова, параметри та feedback.
Memory budget	допустимий обсяг оперативної, графічної або інших видів пам’яті для сцени чи build.
Meta loop	довготривалий цикл прогресії, цілей, unlocks і повернення до core loop.
Metric	вимірювана характеристика поведінки, якості або продуктивності.
Milestone	контрольна точка production з визначеним результатом і критеріями готовності.
Mip map	набір зменшених копій текстури для відображення на різних відстанях.
Mockup	деталізований статичний макет вигляду інтерфейсу або сцени.
Moodboard	колаж референсів, що фіксує тон, стиль, колір, світло, матеріали й емоцію.
Motion sickness	дискомфорт у VR/XR через невідповідність візуального руху й вестибулярних сигналів.
NavMesh	навігаційне представлення доступних зон для руху AI.
Occlusion	перекриття видимості або звуку об’єктами середовища.
Onboarding	послідовність, що навчає базових дій, правил, цілей і ментальної моделі гри.
One-shot	аудіо або анімаційна подія, яка відтворюється один раз.
Open world	структура гри з великим взаємопов’язаним простором і значною свободою маршруту.
Parameter	керована величина, що змінює поведінку системи, матеріалу, анімації або звуку.
Persona	аргументований профіль типового користувача з цілями, мотиваціями, бар’єрами й контекстом.
Pitch	коротка переконлива презентація ідеї, цінності, доказів і наступного кроку.
Playtest	спостереження за реальною грою користувачів для перевірки гіпотез і виявлення проблем.
Pooling	повторне використання створених об’єктів замість частого створення й видалення.

Продовження табл. Гл.1

Термін	Визначення
Post-processing	обробка фінального кадру після основного рендерингу.
Prefab	повторно використовуваний шаблон об’єкта в Unity.
Priority	правило, яке визначає, яка дія, подія, анімація або звук важливіші за інші.
Profiler	інструмент вимірювання часу, пам’яті, викликів і ресурсів під час виконання.
Progression	зміна можливостей, складності, контенту або статусу гравця з часом.
Prototype	швидка версія для перевірки конкретної гіпотези з мінімальним необхідним контентом.
QA	система процесів забезпечення якості: планування, тестування, triage, регресія й контроль ризиків.
Readability	здатність гравця швидко розпізнати важливі об’єкти, стани, небезпеки й результати.
Regression test	повторна перевірка, що раніше працездатна функція не зламалася після змін.
Release candidate	збірка-кандидат на випуск, яка пройшла основні перевірки й не має відомих blocker-дефектів.
Rebinding	можливість змінити прив’язку дії до клавіші, кнопки або жесту.
Recovery	фаза завершення дії, повернення контролю або стабілізації після impact.
Retention	частка користувачів, які повертаються до гри через певний час.
Root motion	рух персонажа, безпосередньо закодований у трансформації анімації.
RTPC	керування аудіопараметрами в реальному часі через значення швидкості, стану, відстані тощо.
Save game	серіалізований стан прогресу, налаштувань або світу.
Scope	узгоджений обсяг функцій, контенту, якості та робіт у конкретній версії.
Scriptable object / data asset	контейнер даних, який відокремлює налаштування від логіки й підтримує data-driven design.
Shader	програма для обчислення вигляду геометрії, поверхні або пікселів.
Signposting	візуальні, звукові й просторові підказки, що спрямовують увагу та рух
Snapshot	збережена конфігурація параметрів міксу, рендерингу або стану для швидкого переходу.
Spatial audio	відтворення звуку з урахуванням положення, відстані, напрямку, occlusion і простору.
Spawn	створення або поява об’єкта в ігровому світі.

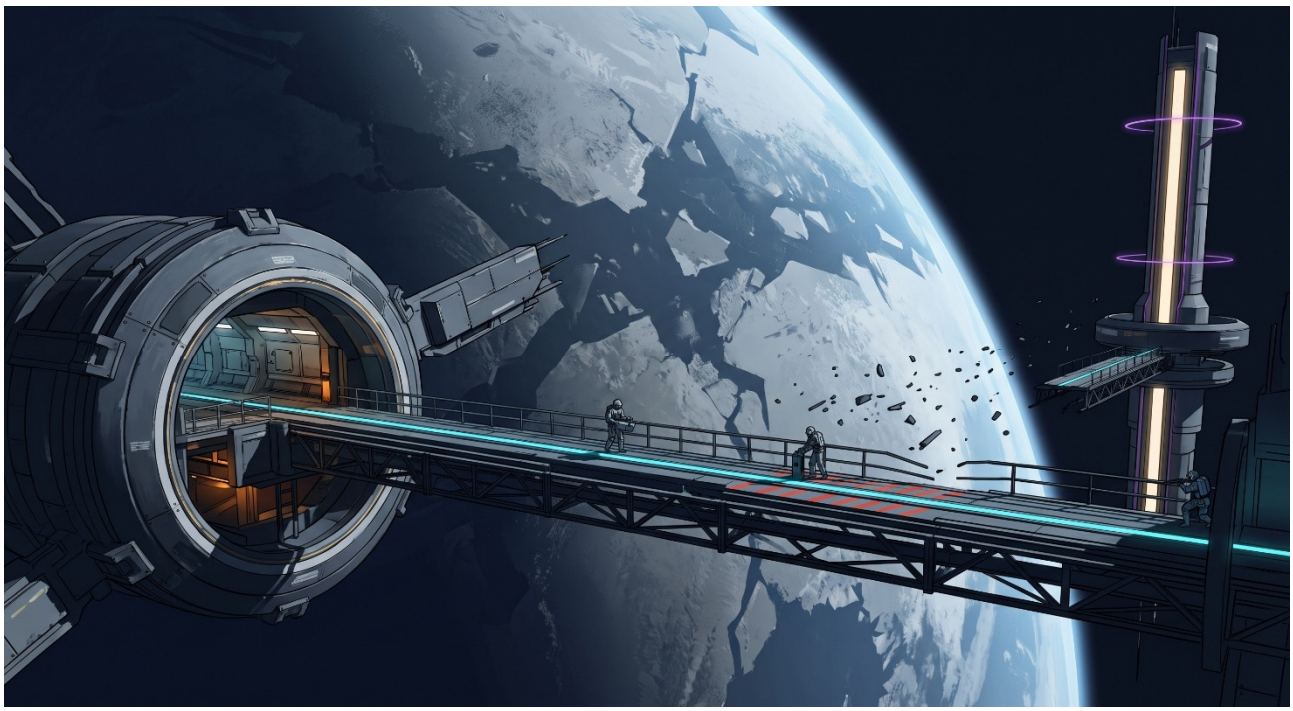
Продовження табл. Гл.1

Термін	Визначення
Sprint	короткий фіксований цикл планування й виконання в ітеративному production.
State machine	модель поведінки через дискретні стани, переходи, умови й події.
Stinger	короткий виразний аудіосигнал, що підкреслює важливу подію.
Storyboard	послідовність кадрів, що описує дію, камеру, темп і ключові моменти.
Telemetry	автоматичний збір подій і метрик використання, поведінки або продуктивності.
Telegraphing	попереднє повідомлення про майбутню дію, небезпеку чи зміну стану.
Test case	формалізований сценарій перевірки з передумовами, кроками, очікуваним і фактичним результатом.
Texture atlas	об’єднання кількох текстур в одну для зменшення перемикань і draw calls.
Time-to-fun	час від початку взаємодії до першого переконливого моменту цінності або задоволення.
Triage	класифікація та пріоритизація дефектів за впливом, ризиком і терміновістю.
Tunable	параметр, який можна швидко змінювати без переписування логіки.
UI flow	схема переходів між екранами, станами й діями інтерфейсу.
User story	короткий опис потреби у форматі «як тип користувача, я хочу дію, щоб отримати цінність».
UV mapping	проєкція координат 2D-текстури на поверхню 3D-моделі.
VCS	система контролю версій, що зберігає історію, гілки, авторство й дозволяє командну роботу.
Vertical slice	невеликий завершений фрагмент гри, який демонструє цільову якість усіх ключових дисциплін.
VFX	візуальні ефекти, що пояснюють події, підсилюють емоцію та формують стиль.
Voice limit	обмеження кількості одночасних аудіоджерел або голосів.
VR comfort	сукупність рішень щодо locomotion, camera, latency, scale й feedback для зменшення дискомфорту.
Wireframe	схематичний макет структури інтерфейсу без фінального візуального стилю.
World space UI	інтерфейс, розміщений у координатах ігрового світу.
XR	узагальнений термін для VR, AR, MR та інших імерсивних технологій.



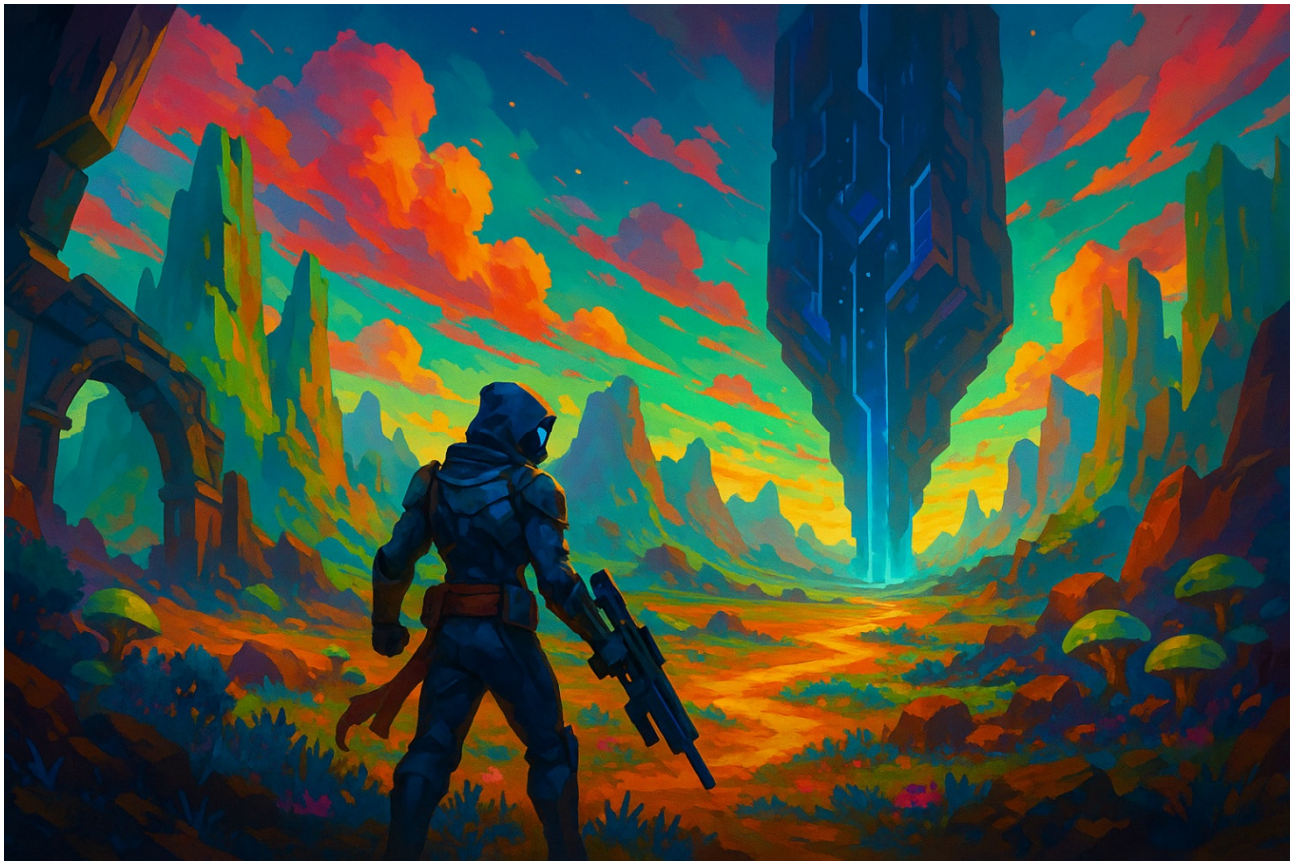















DROIMER
1
2
3
30
210
69




JA'RDEN
DEFEAT THE BOSS
Time Remaining: 25:47
Loot Cache
ELITE
Q
E
36/120



НАВЧАЛЬНЕ ВИДАННЯ

Назаренко Володимир Анатолійович

РОЗРОБКА ІГРОВИХ ДОДАТКІВ

Навчальний посібник

Підписано до друку 20.11.26 Формат 60х84\16
Ум. друк. арк. 18,5 Наклад 100 прим. Зам. № 260645

Видавець і виготовлювач Національний університет біоресурсів
і природокористування України,
вул. Героїв Оборони, 15, м. Київ, 03041.
Свідоцтво суб'єкта видавничої справи
ДК № 4097 від 17.06.2011