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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільної гри в жанрі аркадного шутера»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр МАРКОВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Олександр МАРКОВСЬКИЙ

Керівник: Олесь ДІБРІВНИЙ
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Марковському Олександру Павловичу

1. Тема кваліфікаційної роботи: «Розробка мобільної гри в жанрі аркадного шутера»
керівник кваліфікаційної роботи доктор філософії (PhD) Олесь ДІБРІВНИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про принципи розробки мобільних ігор, підходи до реалізації механіки об’єднання (merge), особливості жанру аркадного шутера, технічна документація рушія Unity, мови програмування C#, інструментів зберігання даних у форматі JSON, а також засобів розробки й керування проєктом, зокрема JetBrains Rider, Adobe Illustrator та GitHub.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд жанру аркадних шутерів, аналіз існуючих аналогів та визначення вимог до мобільної гри.
2. Огляд та аналіз програмних засобів для розробки мобільних ігор, зокрема рушія Unity, мови C# та інших допоміжних інструментів.

3. Проєктування архітектури гри з використанням механіки об’єднання (merge), реалізація ігрової логіки, інтерфейсу та збереження прогресу.
4. Тестування мобільної гри, оцінка функціональності, продуктивності та юзабіліті застосунку.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів
2. Вимоги до програмного забезпечення
3. Програмні засоби реалізації
4. Діаграма діяльності
5. Діаграми класів
6. Екранні форми
7. Апробація результатів дослідження
8. Висновки

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02 – 04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03 – 13.03.2025	
3	Вибір рушія, мови програмування та інструментів розробки	14.03 – 25.03.2025	
4	Реалізація базової логіки гри (керування гравцем, стрільба, вороги, UI)	26.03 – 31.03.2025	
5	Розробка системи merge, прокачки, збереження	01.04 – 14.04.2025	
6	Реалізація UI/UX, аудіо та оптимізація	15.04 – 21.04.2025	
7	Тестування гри, налагодження архітектури та фінальні правки	22.04 – 28.04.2025	
8	Оформлення роботи: вступ, висновки, реферат	29.04 – 11.05.2025	
9	Розробка демонстраційних матеріалів	12.05 – 18.05.2025	
10	Попередній захист роботи	19.05 – 30.05.2025	

Здобувач вищої освіти _____ Олександр МАРКОВСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 49 стор., 1 табл., 17 рис., 31 джерел.

Мета роботи – вдосконалення жанру аркадного шутера шляхом інтеграції механіки об'єднання об'єктів.

Об'єкт дослідження – процес розробки мобільної гри з елементами аркадного шутера та механікою поєднання.

Предмет дослідження – програмна реалізація merge-механіки в аркадному шутері.

Короткий зміст роботи: В роботі проаналізовано особливості мобільних аркадних ігор із механікою merge та їхній вплив на зацікавленість користувача. розглянуто сучасні підходи до поєднання жанрів в мобільних іграх (гібридизації) та їхній вплив на утримання гравця. проаналізовано ігрові рішення в існуючих застосунках на основі подібної механіки. Розроблено алгоритм логіки merge-системи та програмно реалізовані ключові функціональні можливості, зокрема: стрільба по хвилях супротивників, об'єднання ігрових одиниць (merge), покрокове посилення героя та збереження прогресу. Проведено тестування функціональності гри. В роботі використано ігровий рушій Unity, мову програмування C#, Інтегроване середовище розробки Rider, систему контролю версій GitHub, векторний графічний редактор – illustrator та JSON для збереження данних.

Сферою використання застосунку є розвага орієнтована на казуальних гравців, зацікавлених в аркадному шутері з елементами стратегії.

КЛЮЧОВІ СЛОВА: МОБІЛЬНА ГРА, UNITY, C# , MERGE-МЕХАНІКА, АРКАДНИЙ ШУТЕР.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

UI – User Interface – інтерфейс користувача

UX – User Experience – користувацький досвід

AI – Artificial Intelligence – штучний інтелект

JSON – JavaScript Object Notation – формат обміну даними

AES – Advanced Encryption Standard – алгоритм симетричного шифрування

C# – C-Sharp – мова програмування

URP – Universal Render Pipeline – уніфікований графічний рендер у Unity

DOTween – Demigiant Tween – анімаційна бібліотека для Unity

Zenject – Dependency Injection Framework for Unity – інструмент впровадження залежностей

MVP – Minimum Viable Product – мінімально життєздатний продукт

IDE – Integrated Development Environment – інтегроване середовище розробки

Merge – механіка об’єднання елементів у грі

ВСТУП

Актуальність теми дослідження зумовлена стрімким розвитком ігрової індустрії, яка на сьогодні є однією з найбільш перспективних і прибуткових сфер цифрових технологій. Зростаюча популярність комп'ютерних і мобільних ігор на різних платформах створює сприятливі умови для реалізації інноваційних ідей та професійного розвитку розробників. Розробка власної гри є не лише актуальним викликом для сучасного програміста, а й можливістю вдосконалити практичні навички в галузі розробки програмного забезпечення.

Особистий інтерес автора до створення віртуальних світів став однією з головних причин вибору теми дослідження. Відеоігри сьогодні виходять за межі розваг — вони активно застосовуються в освіті, медицині та професійній підготовці. Зокрема, ігрові симуляції використовують для тренування лікарів, пілотів, рятувальників, а також для реабілітації пацієнтів після інсульту чи травм головного мозку. Це свідчить про високу соціальну та наукову значущість даної сфери.

Розробка мобільної гри у жанрі аркадного шутера на базі рушія *Unity* мовою програмування *C#* дозволяє детально дослідити етапи створення ігрового продукту та вдосконалити ключові модулі гри жанру із врахуванням сучасних вимог до геймплею та технічної реалізації.

Мета роботи – вдосконалення жанру аркадного шутера через інтеграцію механіки об'єднання об'єктів.

Об'єкт дослідження – процес розробки мобільної гри з елементами аркадного шутера та механікою поєднання.

Предмет дослідження – програмна реалізація merge-механіки в аркадному шутері.

Завдання дослідження:

1. Проаналізувати існуючі ігри в жанрі аркадний шутер та визначити їх переваги та недоліки.

2. Ознайомитися з існуючими технічними засобами для розробки ігор та вибрати ті, що найбільше підходять для виконання задачі.
3. Визначити основні параметри гри (жанр, простір, вид графіки, ігрові особливості).
4. Спроєктувати гру, застосовуючи вибрані параметри гри.
5. Розробити гру, використовуючи вибрані засоби розробки.

Методи дослідження: структурний аналіз і проєктування, методи розробки та тестування програмного забезпечення, методи верифікації ігор.

Наукова новизна роботи полягає у поєднанні класичного жанру аркадного шутера з механікою merge та побудовою гнучкої ігрової логіки з урахуванням сучасних вимог до мобільного ігрового досвіду. Така комбінація дозволяє зберегти динаміку та простоту керування, одночасно додаючи стратегічну складову, що підвищує утримання гравців.

Практична значущість роботи полягає у створенні повноцінного, оптимізованого ігрового продукту, який може бути використаний як базова платформа для подальших проєктів у сфері мобільного геймдеву, а також слугувати прикладом для навчання та набуття досвіду в розробці ігор.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Відеоігри

1.1.1 Огляд історії мобільних ігор

Історія мобільних ігор розпочалася ще до появи сучасних смартфонів. У 1994 році на мобільному телефоні Hagenuk MT-2000 було попередньо встановлено гру Tetris, що стало першим прикладом інтеграції гри в мобільний пристрій.

Однак справжній прорив відбувся у 1997 році з появою гри Snake на телефонах Nokia. Ця проста, але захоплююча гра стала культовою і сприяла популяризації мобільних ігор у масах.

У 1999 році японська компанія NTT Docomo запровадила сервіс i-mode, який дозволяв завантажувати ігри на мобільні телефони. Це стало першим кроком до створення екосистеми мобільних ігор з можливістю завантаження контенту.

Справжня революція в мобільному геймінгу відбулася з появою смартфонів. У 2007 році Apple представила iPhone, а в 2008 році запустила App Store, що дозволило розробникам легко розповсюджувати свої ігри. Це призвело до буму мобільних ігор, таких як Angry Birds та Fruit Ninja.

У 2012 році з'явилися ігри, що використовували модель "freemium", де базова гра була безкоштовною, але пропонувалися внутрішньоігрові покупки. Прикладами таких ігор є Candy Crush Saga та Puzzle & Dragons. Ця модель стала домінуючою в мобільному геймінгу.

Сьогодні мобільні ігри охоплюють широкий спектр жанрів та технологій, включаючи доповнену реальність (AR) та віртуальну реальність (VR). Вони стали важливою частиною індустрії розваг, залучаючи мільйони гравців по всьому світу.

Таким чином, мобільні ігри пройшли шлях від простих вбудованих розваг до складних, багатофункціональних продуктів, що формують сучасну культуру геймінгу.

1.1.2 Аналіз сучасного стану ігрової індустрії

Світова ігрова індустрія продовжує демонструвати динамічний розвиток, незважаючи на економічні виклики та структурні зміни. У 2024 році глобальний дохід від відеоігор досяг $184,3 млрд, з яких половину забезпечив мобільний сегмент ($92,5 млрд), що зріс на 2,8% порівняно з попереднім роком.

Мультиплатформеність та мультиплеєрність

Згідно з даними Unity Gaming Report 2024, кількість мультиплатформених ігор зросла на 40% з 2021 року, а серед невеликих студій — на 71%. Мультиплеєрні ігри принесли розробникам $2,3 млрд доходів у 2023 році, що на 10% більше, ніж у попередньому. Мобільні мультиплеєрні ігри мають на 40,2% більше активних гравців на місяць, ніж однокористувацькі.

Розвиток монетизаційних моделей

Розробники все частіше використовують комбіновані моделі монетизації, зокрема внутрішньоігрову рекламу (IAA). У 2023 році доходи від реклами зросли на 26,7%. Найбільшу ефективність продемонстрували ігри, які поєднують винагороджуванні відео та offerwall: утримання користувачів на 7-й день збільшилося на 4%, а на 30-й — на 2%.

Інтеграція штучного інтелекту

Штучний інтелект (AI) стає невід'ємною частиною процесу розробки ігор. 62% студій використовують AI для покращення робочих процесів, зокрема для створення контенту та анімацій. 71% з них відзначили покращення ефективності проєктів, зменшення часу реалізації та зростання якості продуктів.

Зростання ринку мобільних ігор

Мобільні ігри залишаються провідним сегментом індустрії. У 2024 році вони забезпечили половину глобального доходу від відеоігор. Це зростання пов'язане з доступністю смартфонів та розвитком мобільних платформ.

Розвиток інді-ігор та користувацького контенту

Інді-ігри, такі як "Among Us" та "Stardew Valley", демонструють значний успіх завдяки платформам на кшталт Steam та Roblox, які дозволяють легко

розповсюджувати нові ігри. Користувацький контент (UGC) стає важливою частиною ігрового досвіду, сприяючи залученню гравців та розвитку спільнот.

Виклики та перспективи

Незважаючи на позитивні тенденції, індустрія стикається з викликами, такими як затримки у випуску ключових проєктів, що можуть призвести до значних фінансових втрат. Наприклад, затримка релізу Grand Theft Auto VI очікується спричинити втрати в $2,7 млрд у 2025 році.

1.1.3 Роль мобільних ігор в економіці та суспільному житті

Комп'ютерні ігри за останні десятиліття перетворилися на важливий економічний сектор, що характеризується значними обсягами виробництва, збуту та інновацій. Індустрія відеоігор генерує мільярди доларів доходу щорічно, створюючи робочі місця у різних сферах — від розробки програмного забезпечення і графічного дизайну до маркетингу та підтримки користувачів. Завдяки постійному впровадженню нових технологій, таких як штучний інтелект, віртуальна та доповнена реальність, галузь активно стимулює інновації, які поширюються і на інші галузі економіки.

Окрім прямого економічного впливу, комп'ютерні ігри відіграють важливу роль у суспільстві, формуючи нові культурні та соціальні практики. Вони сприяють розвитку творчого потенціалу, логічного мислення та стратегічних навичок, що може позитивно впливати на професійний та особистісний розвиток користувачів. Відеоігри часто виступають платформою для соціальної взаємодії, допомагаючи формувати спільноти за інтересами і розвивати навички командної роботи.

Крім того, комп'ютерні ігри дедалі активніше використовуються в освіті та наукових дослідженнях як ефективний інструмент навчання і мотивації. Вони дозволяють моделювати складні процеси, сприяють глибшому засвоєнню інформації та розвитку критичного мислення.

Важливе місце комп'ютерні ігри займають і в культурному контексті. Вони поєднують у собі елементи мистецтва, музики, літератури та кіно, створюючи унікальні мультимедійні твори. Ігрові сюжети та візуальні рішення часто стають

джерелом натхнення для інших творчих індустрій. Також комп'ютерні ігри сприяють збереженню та популяризації культурної спадщини, передаючи історичні події та традиції у інтерактивній формі.

Отже, значення комп'ютерних ігор виходить далеко за межі розважальної сфери — вони є потужним економічним фактором і вагомим елементом сучасного соціокультурного ландшафту.

1.1.4 Практичне використання мобільних ігор

Мобільні ігри сьогодні займають важливе місце у цифровому середовищі, ставши не лише популярним засобом розваг, а й універсальним інструментом з широким спектром застосувань у різних сферах життя. Розповсюдження мобільних пристроїв, таких як смартфони та планшети, а також зручність доступу до ігор у будь-який час і в будь-якому місці сприяють їхній значній популярності серед користувачів різного віку та соціального статусу.

Розважальна функція

Основним напрямком використання мобільних ігор є розваги та дозвілля. Завдяки великій різноманітності жанрів — від простих головоломок до складних стратегій та рольових ігор — користувачі мають змогу обирати ігри відповідно до своїх інтересів і вподобань. Мобільні ігри забезпечують швидкий доступ до розваг у будь-який час, що особливо важливо у сучасному ритмі життя, де люди шукають можливості відпочити навіть під час коротких перерв.

Освітнє застосування

В останні роки мобільні ігри дедалі частіше використовуються як інноваційний освітній інструмент. Освітні ігри допомагають розвивати логіку, критичне мислення, мовні навички, а також зміцнюють пам'ять і увагу. Інтерактивний формат подачі матеріалу сприяє підвищенню мотивації учнів, робить навчання більш цікавим і доступним. Особливо корисними такі ігри є для дітей молодшого шкільного віку, але їхня ефективність доведена і для дорослих у процесі професійного навчання та підвищення кваліфікації.

Соціальна взаємодія

Мобільні ігри сприяють розвитку соціальної взаємодії, що є важливим аспектом сучасного цифрового життя. Багато ігор мають багатокористувацький режим, що дозволяє гравцям взаємодіяти в режимі реального часу, створювати спільноти за інтересами, брати участь у командних змаганнях та спільних проектах. Такий формат підтримує розвиток комунікативних навичок, вміння працювати в команді та встановлювати міжособистісні контакти.

Маркетинг і гейміфікація

Завдяки своїй масовості мобільні ігри активно використовуються як платформа для маркетингових кампаній і гейміфікації бізнес-процесів. Через інтегровану рекламу, рекламні акції у вигляді внутрішньоігрових подій та спонсорські проєкти бренди отримують можливість залучати широку аудиторію. Гейміфікація, яка передбачає впровадження ігрових механік у навчальні, робочі або комерційні процеси, допомагає підвищити мотивацію та залученість користувачів.

Медичне і реабілітаційне використання

Окрему нішу займає застосування мобільних ігор у медицині. Вони використовуються для когнітивної терапії, зокрема для поліпшення пам’яті, уваги, моторики та координації у пацієнтів з різними неврологічними захворюваннями. Ігрові методики також сприяють зниженню рівня стресу і тривожності, допомагають у психоемоційному відновленні та реабілітації. Медичні ігри часто розробляються з урахуванням специфіки певних захворювань і мають на меті покращити якість життя пацієнтів.

Культурне та творче значення

Мобільні ігри також виступають платформою для розвитку творчості і поширення культурних цінностей. Вони поєднують елементи мистецтва, музики, дизайну і літератури, створюючи унікальні інтерективні твори, які впливають на сучасну культуру. Деякі ігри використовують історичні сюжети і мотиви, що сприяє збереженню культурної спадщини і підвищенню інтересу до історії серед молоді.

Таким чином, сфера використання мобільних ігор є багатогранною і продовжує активно розширюватися. Вони вже давно перестали бути виключно засобом розваг, ставши важливим інструментом у сфері освіти, соціальної взаємодії, маркетингу, медицини та культури. Це підтверджує їхній високий потенціал та значення у сучасному цифровому суспільстві.

1.2 Жанри ігор

У сучасній індустрії мобільних ігор виділяють низку основних жанрів, кожен із яких має специфічні ігрові механіки, аудиторні особливості та моделі монетизації. Нижче наведено класифікацію жанрів із коротким описом їх ключових характеристик та сучасних трендів.

Казуальні ігри

Казуальні (casual) ігри характеризуються простою механікою, короткими ігровими сесіями (1–3 хвилини) та мінімальними вимогами до навичок користувача. Завдяки інтуїтивно зрозумілим таповим і свайповим діям, вони призначені для широкої аудиторії, включаючи непрофесійних гравців і користувачів старших вікових категорій. Моделі монетизації ґрунтуються на безкоштовному завантаженні з вбудованою рекламою, внутрішньо-ігрових покупках косметичних предметів або додаткових «життів». У 2024 р. помітно зросла популярність hyper-casual ігор, які пропонують ще простішу графіку та механіку «коротких досягнень», сприяючи швидкому утриманню користувача.

Головоломки

Puzzle-ігри спрямовані на розвиток логічного та просторового мислення через вирішення завдань або маніпуляцію об’єктами в межах умовної системи правил. Такі ігри стимулюють концентрацію уваги і когнітивну гнучкість, що доведено низкою психологічних досліджень. Монетизація зазвичай реалізується через продаж «підказок» або нових рівнів, а також демонстрацію рекламних роликів між спробами. Серед поточних трендів — інтеграція доповненої реальності (AR) для формування головоломок у реальному середовищі гравця.

Рольові ігри (RPG)

RPG на мобільних платформах вирізняються глибинною кастомізацією персонажів, системами розвитку навичок і ресурсів, а також розгалуженими сюжетними лініями. Соціальні функції — кланові союзи, PvP-арени та спільні рейди — сприяють високому рівню залученості й утриманню аудиторії. Основні моделі монетизації включають «battle pass», випадкові «loot box» та сезонні оновлення із спеціальними подіями. У 2025 р. провідні проєкти активно практикують Live Ops-підхід із щотижневими оновленнями контенту.

Стратегії

Жанр охоплює декілька підтипів: tower defense, масові онлайн-стратегії (MMORTS) та city builder. Гравці виконують планування ресурсів і тактичне розміщення об’єктів, що стимулює довготривалу взаємодію з проєктом. Монетизація здійснюється через пришвидшення будівництва, покупку преміальних ресурсів і VIP-статусів із прискоренням очікування. Сучасна тенденція — впровадження елементів блокчейну для обміну внутрішньо-ігровими активами.

Екшн та шутери

Екшн-ігри та шутери на мобільних пристроях поєднують швидкі рефлекси, тактичні перестрілки та командні режими. Розвиток хмарних ігрових сервісів покращив якість графіки і знизив затримку управління, що особливо важливо для жанру battle royale з участю сотень гравців одночасно. Основна монетизація — косметичні набори для персонажів і зброї, преміальні проходи сезонів та внутрішньо-ігрові бафи.

Симулятори

Симуляторні ігри відтворюють конкретні види діяльності: фермерство, керування транспортом або тренінг для медичних та навчальних цілей. Вони вирізняються деталізованими економічними та фізичними моделями, які використовуються не лише в розважальних, а й у професійних тренажерах. Популярні моделі монетизації — підписка на щоденні бонуси й покупка рідкісних

ресурсів. Новий тренд — поява AR/VR-додатків для глибокого занурення у професійні симуляції.

Аркади

Аркадні ігри — один із найстаріших жанрів, що зберігає актуальність завдяки простій механіці нескінченних раундів і системам лідербордів. Вони стимулюють «ще один раунд» через короткі динамічні виклики та рекордні таблиці. Монетизація передбачає показ реклами за бонус і одноразові покупки для розблокування нових персонажів. Серед сучасних трендів — повернення до 8- і 16-бітної естетики та колаборації з відомими брендами ретро-франшиз.

Idle/Clicker ігри

Idle- або clicker-ігри базуються на автоматизованому накопиченні ресурсів, навіть коли користувач неактивний. Відчуття прогресу створюється за рахунок регулярних апгрейдів і «лотерейних» елементів. Монетизація здійснюється через прискорення процесів і щоденні бонуси за вхід у гру. Сучасні гібриди жанру поєднують механіки RPG із комплектами героїв і кампаніями, що додає глибини геймплею.

1.3 Процес створення мобільних ігор

Розробка мобільних ігор складається з кількох послідовних фаз, кожна з яких має чітко визначені завдання, артефакти та критерії завершення. Нижче подано опис основних етапів процесу з посиланнями на авторитетні джерела.

Ідея та концепція

На першому кроці формулюється загальна ігрова ідея та визначаються стратегічні параметри проєкту: аналіз ринку, цільова аудиторія, унікальна торгова пропозиція, вибір платформи (iOS, Android) та орієнтовний бюджет. Результатом цього етапу є Game Concept Document, який містить опис жанру, ключових механік, сюжетну лінію, приблизні технічні й арт-вимоги, а також стратегію монетизації.

Пре-продакшн

Підготовча фаза включає детальне опрацювання концепції:

- Геймдизайн: побудова докладного плану рівнів, системи прогресу та балансу;
- Технічний вибір: визначення рушія (Unity, Unreal Engine або власні рішення) та набору інструментів;
- Прототипування: створення мінімального життєздатного прототипу (Minimum Viable Product, MVP) для перевірки основних ігрових механік;
- Планування: розробка дорожньої карти проєкту (roadmap) із розподілом відповідальностей і тимчасових рамок.

Продакшн (розробка)

Центральна фаза, де реалізуються всі компоненти гри:

- Програмування: написання коду ігрової логіки, інтеграція мережевих сервісів, платіжних API та аналітики;
- Арт-ресурси: створення 2D/3D-моделей, анімацій, UI/UX-дизайну;
- Аудіо: запис та інтеграція музичних треків, звукових ефектів і голосових доріжок;
- Ітераційне тестування: внутрішні перевірки (smoke tests, regression tests) та виправлення критичних багів із використанням системи контролю версій (Git, Perforce).

Альфа- та бета-тестування

- Alpha: стадія «feature complete» — усі основні механіки реалізовані, гра функціонує, але потребує оптимізації та наповнення
- Beta: фінальна перевірка стабільності та юзабіліті, зазвичай із залученням зовнішніх тестувальників (closed/open beta) для збору фідбеку і виявлення залишкових помилок.

Реліз та маркетинг

Підготовка до виходу гри включає:

- App Store/Google Play: налаштування метаданих (скріншоти, опис, ключові слова), локалізація під різні регіони;

- ASO-оптимізація: аналіз конкурентів і підбір релевантних ключових слів;
- Маркетингові кампанії: digital-реклама, колаборації з інфлюенсерами, тизери й трейлери в соціальних мережах;
- Моніторинг метрик релізного дня: кількість завантажень, конверсія інсталяцій, ретеншн на день 1 і 7.

Підтримка та Live Ops

Після публікації гра переходить у режим постійної підтримки:

- Live Ops: регулярні оновлення контенту (нові події, рівні, косметика), виправлення помилок і забезпечення стабільності;
- Аналітика: відстеження ключових показників (DAU/MAU, ARPU, churn rate) та A/B-тестування для оптимізації ігрового процесу;
- Сезонні івенти: обмежені за часом заходи для підтримки інтересу й стимулювання внутрішньоігрових покупок.

1.4 Рушії для розробки ігор

У сучасній індустрії розробки ігор рушій (game engine) виступає ключовим програмним середовищем, яке надає розробникам готові компоненти для обробки графіки, фізики, звуку, сценарної логіки та мережевої взаємодії. Нижче розглянуто найпоширеніші рушії, їхні технічні особливості, моделі ліцензування та поточні ринкові тренди.

Unity

Unity є одним із найпопулярніших ігрових рушіїв у світі, особливо у сфері мобільної розробки, інді-проєктів, 2D- та 3D-ігор. Його було створено компанією Unity Technologies у 2005 році, і з того часу він зазнав значної еволюції. Unity підтримує понад 25 платформ, включаючи Android, iOS, Windows, macOS, WebGL, PlayStation, Xbox, Switch тощо. Головною перевагою Unity є його компонентна

архітектура: основною сутністю є GameObject, до якого можна прикріплювати будь-які компоненти, що задають поведінку, фізику, візуалізацію тощо.

Unity дозволяє розробникам писати код на мові C#, що є зручною та сучасною об’єктно-орієнтованою мовою з великою кількістю навчальних матеріалів. Також рушій має потужну екосистему: Asset Store, Unity Services (аналітика, реклама, монетизація), Addressable Assets, URP/HDRP для гнучкої роботи з графікою, підтримку XR для розробки під AR/VR, а також DOTS (Data-Oriented Technology Stack) для високопродуктивної розробки. Unity активно використовується як у навчальних, так і в комерційних проєктах, і залишається вибором №1 для розробки мобільних ігор.

- Мови програмування: C# (Mono/IL2CPP).
- Платформи: мобільні (iOS, Android), десктоп, консолі, WebGL, VR/AR.
- Особливості: широкий набір готових пакетів (Asset Store), велика спільнота та об’ємна документація.
- Модель ліцензування: безкоштовна Personal для доходу < \$100 000/рік; платні Plus/Pro з додатковою підтримкою та можливістю видалення Unity-брендингу.
- Ринкові позиції: за даними Video Game Insights, у 2024 році 51 % ігор на платформі Steam було випущено з використанням Unity. Проте з 2021 р. зафіксовано незначне зниження частки Unity через посилення конкуренції з боку Unreal Engine та відкритих рушіїв.

Unreal Engine 5

Unreal Engine 5, розроблений компанією Epic Games, є потужним рушієм, який найчастіше використовується у створенні високоякісних AAA-проєктів, проте останніми роками все частіше застосовується й у мобільній розробці та інді-іграх. Основною особливістю UE5 є надзвичайно реалістична графіка, яку забезпечують технології Nanite (віртуалізована геометрія) та Lumen (глобальне освітлення в реальному часі). Ці інновації дозволяють отримати кінематографічну якість графіки навіть на середньому залізі.

UE5 підтримує C++ як основну мову програмування, однак пропонує й систему Blueprints — візуальне програмування, яке дозволяє створювати логіку без написання коду. Ця гібридна система є дуже привабливою для дизайнерів та новачків. Рушій має відкрите ядро, розширену документацію, потужну спільноту та доступ до Marketplace. Його застосовують у сфері не лише ігор, а й архітектурної візуалізації, кіно, симуляторів та віртуальних продакшенів. UE5 краще підходить для високобюджетних проєктів або ігор з акцентом на графіку.

- Мови програмування: C++ та Blueprints (візуальне скриптування).
- Платформи: ті самі, що й Unity, з особливим наголосом на високопродуктивні консолі та ПК.
- Особливості: потужний рендеринг (Nanite, Lumen), розвинені інструменти для віртуального виробництва, інтегровані сервіси Unreal Marketplace.
- Модель ліцензування: 5 % роялті з доходів після перших \$1 000 000 прибутку від кожного продукту.
- Ринкові позиції: Unreal Engine стабільно нарощує частку ринку: станом на 2024 р. вона становить 28 % усіх ігор на Steam, що демонструє поступове перетягування частки від власних рушіїв великих студій.

Godot

Godot Engine — це рушій з відкритим кодом, який стрімко набирає популярності завдяки своїй гнучкості, легкості у вивченні та підтримці як 2D, так і 3D-графіки. Основною особливістю Godot є наявність вбудованого редактора сцен, власної мови програмування GDScript (що нагадує Python), а також підтримка C#, C++ та VisualScript.

Godot має модульну структуру, що дозволяє розширювати рушій під потреби проєкту. Його 2D-рушій працює незалежно від 3D-частини, що робить його надзвичайно ефективним для створення 2D-ігор. Завдяки відкритому коду, розробники мають повний контроль над рушієм, що важливо для академічних або експериментальних проєктів. Godot є повністю безкоштовним, без ліцензійних

зборів або обмежень, що робить його привабливим для інді-розробників та стартапів.

- Мови програмування: GDScript (схожий на Python), C#, C++ (модулі).
- Платформи: мобільні, десктоп, Web, консолі (через додаткові порти).
- Особливості: легковагова архітектура, гнучкий редактор сцен, вбудована система UI та анімацій, відсутність роялті чи ліцензійних обмежень.
- Ринкові позиції: частка Godot на ринку рушіїв зросла до близько 8 % у 2024 р., що відповідає 140 % зростанню з 2022 р. завдяки оновленням на Vulkan та підтримці C#.

Construct

Construct — це рушій, орієнтований на створення 2D-ігор без необхідності програмування. Він дозволяє реалізовувати геймплей за допомогою системи подій (Event Sheets), що схожа на логічні блоки. Construct ідеально підходить для новачків, викладачів, геймдизайнерів і тих, хто хоче швидко створити прототип.

- Мови програмування: не потребує кодування, логіка створюється через Event Sheets.
- Платформи: HTML5, Android, iOS, Windows тощо.
- Особливості: інтуїтивний інтерфейс, швидке створення прототипів.
- Модель ліцензування: підписка (безкоштовна версія обмежена).
- Ринкові позиції: використовується переважно в освітніх цілях або інді-проєктах для вебу.

GameMaker Studio

GameMaker — ще один рушій, орієнтований на створення 2D-ігор. Він підтримує мову GML (GameMaker Language), яка проста у вивченні, але дозволяє створювати досить складну логіку. GameMaker має зручний редактор сцен, систему спрайтів, анімацій, частинок та аудіо, що робить його чудовим вибором для створення платформерів, піксельних RPG, аркадних ігор.

- Мови програмування: GML (GameMaker Language).
- Платформи: Windows, macOS, Android, iOS, HTML5, консолі.
- Особливості: простота та швидкість створення 2D-ігор, зручний редактор.
- Модель ліцензування: підписка або разова покупка ліцензії.
- Ринкові позиції: частка на Steam — близько 4 %, серед відомих проєктів — Undertale, Katana ZERO.

Кожен із розглянутих рушіїв має свої сильні та слабкі сторони, і вибір залежить від цілей, досвіду команди, типу гри, вимог до графіки та платформи. Unity — універсальний ігровий рушій із широкою підтримкою платформ і багатою екосистемою. Unreal Engine 5 — це рушій для високоякісної графіки та великих проєктів. Godot — гнучкий, відкритий рушій, ідеальний для інді-геймдеву та навчання. Construct та GameMaker — чудові інструменти для швидкого створення 2D-ігор без глибоких знань програмування. Знання цих інструментів дозволяє обрати оптимальне середовище для реалізації власної гри залежно від поставлених завдань.

2 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ МОБІЛЬНОЇ ГРИ

Для реалізації мобільної гри було використано низку сучасних інструментів, що охоплюють всі етапи розробки – від програмування логіки гри до дизайну графіки та керування версіями проєкту. У цьому розділі розглянуто ключове програмне забезпечення, що використовувалося під час створення гри, його особливості, переваги та роль у загальному процесі розробки.

2.1 Unity

Unity — це сучасний багатоплатформенний ігровий рушій, який забезпечує засоби для створення 2D та 3D інтерактивних застосунків, зокрема ігор, візуалізацій, симуляцій та доповненої реальності. Розроблений компанією Unity Technologies, цей рушій є одним із найпопулярніших інструментів для створення мобільних ігор, завдяки чому активно використовується як інді-розробниками, так і великими ігровими студіями.

Основою архітектури Unity є компонентна модель. Усі об’єкти сцени — GameObjects — є носіями компонентів, які визначають їхню поведінку, властивості, зовнішній вигляд або взаємодію з іншими об’єктами. Такий підхід забезпечує модульність та гнучкість розробки, дозволяючи швидко створювати складні ігрові системи.

Однією з найважливіших переваг Unity є його кросплатформеність — рушій підтримує збірку проєктів під понад 20 платформ, включно з Android, iOS, Windows, macOS, WebGL, консолями тощо. Це дозволяє створювати універсальні застосунки, не переписуючи код під кожну платформу окремо.

Завдяки відкритості, документації, великій спільноті та можливості розширення за допомогою плагінів і скриптів, Unity є одним із найзручніших та найбільш масштабованих рушіїв для розробки мобільних ігор.

Крім того, Unity має вбудовані системи оптимізації продуктивності мобільних застосунків: використання Universal Render Pipeline (URP) дозволяє зменшити навантаження на GPU, а інструменти профілювання (наприклад, Profiler**,** Frame Debugger) допомагають ідентифікувати вузькі місця у продуктивності. Також рушій підтримує інтеграцію з рекламними мережами**,** аналітику**,** in-app покупки та хмарні сервіси, що є ключовими складовими у створенні комерційно успішних мобільних ігор.

2.2 C#

Одним із ключових інструментів програмної реалізації мобільної гри є мова програмування C# (Сі-Шарп). Вона була розроблена компанією Microsoft як частина платформи .NET і зарекомендувала себе як сучасна, об'єктно-орієнтована, типобезпечна та високоінтегрована мова. Завдяки своїй зручності, строгій структурі та широкій функціональності, C# стала основною мовою розробки в середовищі Unity.

Мова C# підтримує всі принципи об’єктно-орієнтованого програмування (ООП): інкапсуляцію, спадкування, поліморфізм та абстракцію. Це дозволяє будувати модульні, гнучкі, масштабовані архітектури програмного забезпечення. У розробці ігор це критично важливо, оскільки дає змогу створювати чітко структуровану взаємодію між ігровими об’єктами, реалізовувати шаблони проєктування, а також ефективно організовувати кодову базу.

Окрему увагу варто приділити підтримці делегатів, подій та лямбда-виразів, які значно спрощують розробку подієво-орієнтованої логіки. У ігрових проєктах ці можливості дозволяють гнучко обробляти взаємодію між гравцем, супротивниками та об’єктами навколишнього середовища.

C# також підтримує Generics (універсальні типи), які дозволяють створювати гнучкі, типобезпечні структури даних, та LINQ (Language Integrated Query), що спрощує роботу з колекціями, базами даних та масивами. Завдяки цьому розробник отримує інструменти для швидкої обробки внутрішньоігрових даних — наприклад,

сортування інвентарю, пошук об'єктів за певними умовами або управління конфігураційними файлами.

Ще однією надзвичайно важливою особливістю C# є асинхронне програмування за допомогою async/await, що дозволяє безпечно виконувати довгі задачі — наприклад, завантаження ресурсів з диска чи з Інтернету — без зупинки ігрового процесу. У мобільних застосунках це критично для збереження плавного ігрового досвіду.

У контексті Unity, C# використовується для створення скриптів поведінки об’єктів (MonoBehaviour), побудови станів гри, UI-інтеракцій, керування анімацією, опрацювання колізій, створення геймплейної логіки, а також для взаємодії з сервером, базами даних та API.

Завдяки активному розвитку, відкритій документації та величезній кількості навчальних матеріалів, C# є доступним для новачків, але водночас надзвичайно потужним інструментом у руках досвідченого розробника.

2.3 JetBrains Rider

JetBrains Rider — це сучасне інтегроване середовище розробки, створене компанією JetBrains, яке спеціалізується на підтримці мов програмування з родини .NET, зокрема C#, і широко використовується в розробці застосунків на основі Unity. Його основна перевага — глибока інтеграція з рушієм Unity, завдяки чому Rider став популярним вибором серед професійних розробників ігор. На відміну від інших редакторів коду, таких як Visual Studio чи Visual Studio Code, середовище Rider поєднує в собі високу продуктивність, інтелектуальні підказки, ефективний рефакторинг та точний аналіз коду завдяки вбудованому механізму ReSharper.

У роботі над проєктом Rider використовувався як основний редактор для написання C#-коду. Середовище підтримує всі ключові функції, необхідні для розробки гри: автодоповнення для Unity API, підсвічування помилок ще до компіляції, швидку навігацію між класами і методами, а також вбудоване налагодження. Завдяки інтеграції з Unity, розробник може миттєво бачити, які

компоненти MonoBehaviour використовуються, які методи пов’язані з життєвим циклом об’єкта, і навіть які поля будуть серіалізовані у редакторі. Rider дозволяє налагоджувати гру прямо під час її виконання, зупиняючи її на потрібних ділянках коду, змінюючи значення змінних у реальному часі, та діагностуючи помилки логіки або взаємодії між об’єктами.

Інтерфейс Rider гнучко налаштовується, а завдяки підтримці плагінів його функціональність може бути розширена відповідно до потреб конкретного проєкту. Висока швидкість роботи з великими рішеннями, стабільність, регулярні оновлення та підтримка платформ Windows, macOS і Linux роблять JetBrains Rider надійним інструментом для розробки мобільних ігор на Unity.

2.4 GitHub

GitHub — це платформа для хостингу та спільної розробки програмного забезпечення, яка базується на системі контролю версій Git. У контексті створення мобільної гри GitHub виконує надзвичайно важливу роль: він дозволяє організувати надійне зберігання проєкту, вести журнал змін, працювати над різними гілками розробки, а також співпрацювати з іншими учасниками команди або отримувати зворотний зв’язок від спільноти. Сервіс забезпечує відстеження історії всіх змін у коді, що дозволяє розробнику повернутися до попередніх версій, швидко знаходити й усувати помилки, а також експериментувати з новими функціями без ризику пошкодити основну версію гри.

У роботі над проєктом GitHub використовувався для централізованого зберігання вихідних файлів гри, управління версіями та створення резервних копій. Через систему гілок (branches) було реалізовано ізольовану розробку окремих функціональностей, після чого ці зміни об’єднувалися в основну гілку проєкту. Завдяки pull request-ам можна було перевірити код перед об’єднанням, а в разі необхідності — обговорити або відкоригувати запропоновані зміни. GitHub також автоматично відображає відмінності між версіями файлів, що значно полегшує налагодження та командну роботу.

Також GitHub має офіційний графічний клієнт GitHub Desktop. Цей інструмент надає графічний інтерфейс до Git-функціональності, що особливо зручно для швидкого перегляду змін, виконання комітів, створення гілок або злиття змін без потреби у командному рядку. Його простота використання дозволила пришвидшити щоденні рутинні операції, знизити ймовірність помилок і краще візуалізувати структуру репозиторію. Таким чином, GitHub Desktop став надійним доповненням до основного робочого процесу, особливо на етапах активної розробки та тестування функціоналу гри.

2.5 Adobe Illustrator

Adobe Illustrator — це професійний графічний редактор для створення та редагування векторної графіки, розроблений компанією Adobe. У контексті розробки мобільної гри Illustrator використовується для створення якісних і масштабованих графічних елементів: інтерфейсів користувача, кнопок, іконок та інших візуальних об'єктів, які зберігають чіткість незалежно від роздільної здатності екрана.

Однією з головних переваг Illustrator є підтримка векторного формату, що дозволяє зручно редагувати об’єкти, не втрачаючи якості, і експортувати зображення в різних розмірах. Це особливо важливо для мобільної гри, яка повинна добре виглядати на пристроях з різною діагоналлю та роздільною здатністю.

У дипломному проєкті Adobe Illustrator використовувався для створення графічного стилю гри: проєктування ігрових об'єктів у форматі PNG, підготовки UI-елементів та адаптації графіки до потреб рушія Unity.

2.6 JSON

JSON (JavaScript Object Notation) — це легковаговий формат обміну даними, що використовується для зберігання та передачі структурованої інформації між клієнтом і сервером, або всередині програмного застосунку. Хоча спочатку JSON

був розроблений на основі синтаксису JavaScript, він став універсальним і підтримується більшістю сучасних мов програмування, включно з C#.

JSON відзначається своєю простотою, читабельністю та гнучкістю. Формат базується на двох основних структурах: парах ключ-значення, які утворюють об’єкти, та впорядкованих списках значень, які утворюють масиви.

У розробці ігор, зокрема мобільних, JSON часто використовується для збереження прогресу гравця, налаштувань, конфігурацій ігрових рівнів, локалізації текстів та обміну даними з сервером. У середовищі Unity C# має вбудовану підтримку для серіалізації та десеріалізації JSON через клас JsonUtility, а також через сторонні бібліотеки, як-от Newtonsoft.Json (Json.NET), які надають більше можливостей.

Перевагою використання JSON у грі є легкість зберігання та редагування даних. У випадку проєктування мобільної гри JSON застосовувався для зберігання даних про збереження результатів гри, рівнів гравця та іншої внутрішньоігрової інформації.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Геймдизайн гри

Геймдизайн є ключовим етапом у розробці будь-якої гри, адже саме він формує ігровий досвід користувача, визначає структуру ігрового процесу, способи взаємодії, візуальну подачу та рівень залучення. У рамках цього проєкту було розроблено концепцію мобільної гри в жанрі аркадного шутера з додатковою механікою об’єднання елементів (merge). Основна ідея полягала в тому, щоб поєднати простоту й динамічність аркадного геймплею з елементами стратегічного прийняття рішень.

Ідея виникла як відповідь на популярність гібридних ігор, зокрема таких як Archero, Ball Blast, Shoot 2048, які ефективно комбінують жанри для створення нового ігрового досвіду. Під час аналізу цих проєктів було виявлено, що просте управління, швидкий темп, короткі сесії та система прокачки сприяють високій залученості гравців. Це стало основою для власного геймдизайну.

Головне меню гри є основним хабом. Саме з нього починається знайомство з проєктом, воно формує перше враження. Меню побудоване так, щоб забезпечити швидкий доступ до основних функцій, не перевантажуючи гравця зайвою інформацією. Інтерфейс оформлений мінімалістично, щоб не відволікати увагу від головного — початку гри та прокачки. Вся навігація адаптована під управління однією рукою, що є ключовим принципом у мобільному геймдизайні.

У верхній частині екрана розміщується інформація, яка має постійне значення для гравця: це його рекорд і кількість рубінів — внутрішньої валюти, яку можна використовувати для прокачки. Виведення рекорду на головний екран створює постійний виклик і стимул повернутись до гри, щоб перевершити свій попередній результат. Поруч із рекордом показано накопичені рубіни — винагороду за ефективну гру, яку можна витратити у вкладці Deck. Це дозволяє гравцеві бачити свій ресурс у реальному часі, не заходячи в окремі розділи.

Центральну частину меню займає персонаж — візуальне уособлення гравця, з яким він буде проходити бойові сесії. На цьому етапі персонаж має фіксований вигляд, але надалі можлива система кастомізації, яка дозволить змінювати його зовнішність або зброю. Це додає персоналізації, а отже, і глибшого емоційного залучення. Персонаж може бути анімованим або інтерактивним — реагувати на дії гравця, що створює відчуття живого зв’язку з грою.

У нижній частині розташована навігаційна панель із трьома вкладками: Settings, Fight та Deck. Вкладка Settings дозволяє керувати звуковим супроводом — вмикати або вимикати музику та ефекти. Вона максимально проста і зрозуміла, не вимагає від гравця багато часу. Вкладка Fight виконує функцію старту гри. Це основна кнопка інтерфейсу, тому вона розташована в центрі. Остання вкладка — Deck — відповідає за управління здібностями. Тут гравець може переглядати доступні типи снарядів, покращувати їх за рубіни та ознайомлюватись із їхніми ефектами. Саме через цю вкладку реалізується стратегічна складова гри: гравець вирішує, які снаряди розвивати в першу чергу, щоб мати перевагу в бою.

Таблиця 3.1

Характеристики типів снарядів у грі

Тип снаряда	**Базова дія**	**Ефект прокачки**	**Ігрове призначення**
Standard	Прямий урон	Збільшення урону	Основний, простий для старту
Ice	Сповільнює ворога	Збільшення тривалості сповільнення	Контроль швидкості ворогів
Penetration	Пробиває кілька ворогів наскрізь	Пробиває більшу кількість ворогів	Ефективний проти груп цілей
Poison	Завдає урон з часом	Збільшує тривалість та інтенсивність отруєння	Підходить для затяжних боїв
Explosion	Завдає урон по радіусу	Збільшує радіус ураження	Добре діє проти скупчення ворогів

Головне меню виконує одразу кілька важливих функцій. По-перше, воно задає настрій гри. По-друге, воно виступає інформаційним центром: тут відображаються всі ключові дані про стан прогресу гравця. І по-третє — це пункт управління, звідки можна швидко перейти до бою або прокачки. Завдяки такій структурі меню підтримує ігрову петлю, забезпечує зручність користування і стимулює повторні сесії. Успішне меню не нав’язується — воно стає невидимим помічником, який супроводжує гравця в його ігровій подорожі, не відволікаючи, а доповнюючи основний досвід.

Концепція гри базується на нескінченному режимі виживання, де гравець керує героєм, що автоматично стріляє у ворогів, які падають зверху. Завдання гравця — за допомогою свайпів вліво та вправо знищувати ворогів. Кожен знищений супротивник може залишити ресурс, який потрібно підібрати щоб додати до інвентаря, розташованого в нижній частині екрана. Інвентар складається з 15 слотів (3 рядки по 5 стовбців) (див. рисунок 3.1).

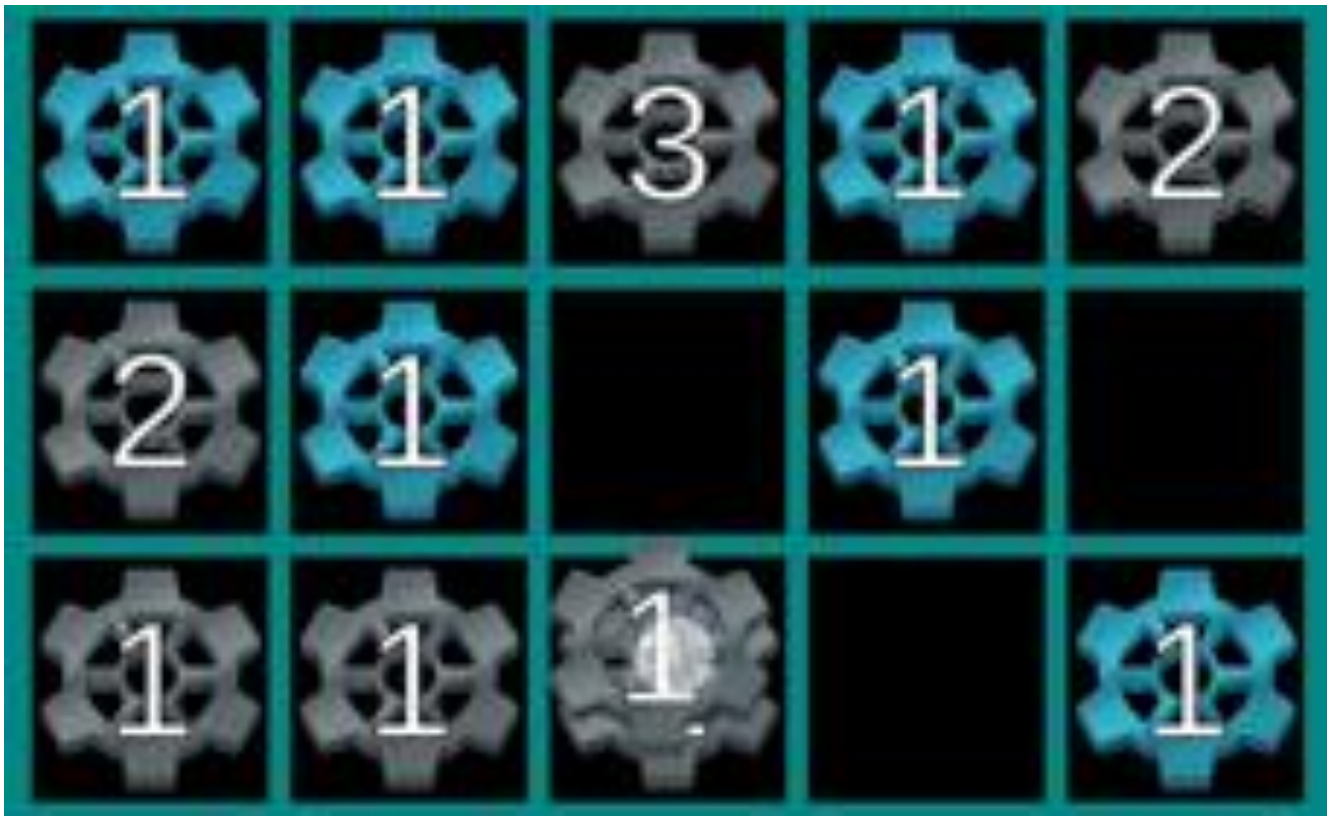


Рис.3.1 Приклад інвентаря ресурсів

Під час гри гравець може вручну об’єднувати однакові ресурси одного рівня. При об’єднанні, рівень ресурса збільшується і кількість снарядів гравця збільшується тим самим він стає сильнішим.

Після програшу з’являється екран поразки з кнопкою повернення в меню. Це дозволяє внести зміни в арсенал та спробувати знову, що підсилює ефект "ще однієї спроби" (one more try effect). Така петля — сесія → покращення → нова спроба — є критично важливою для утримання гравців у мобільних проєктах (див. рисунок 3.2).

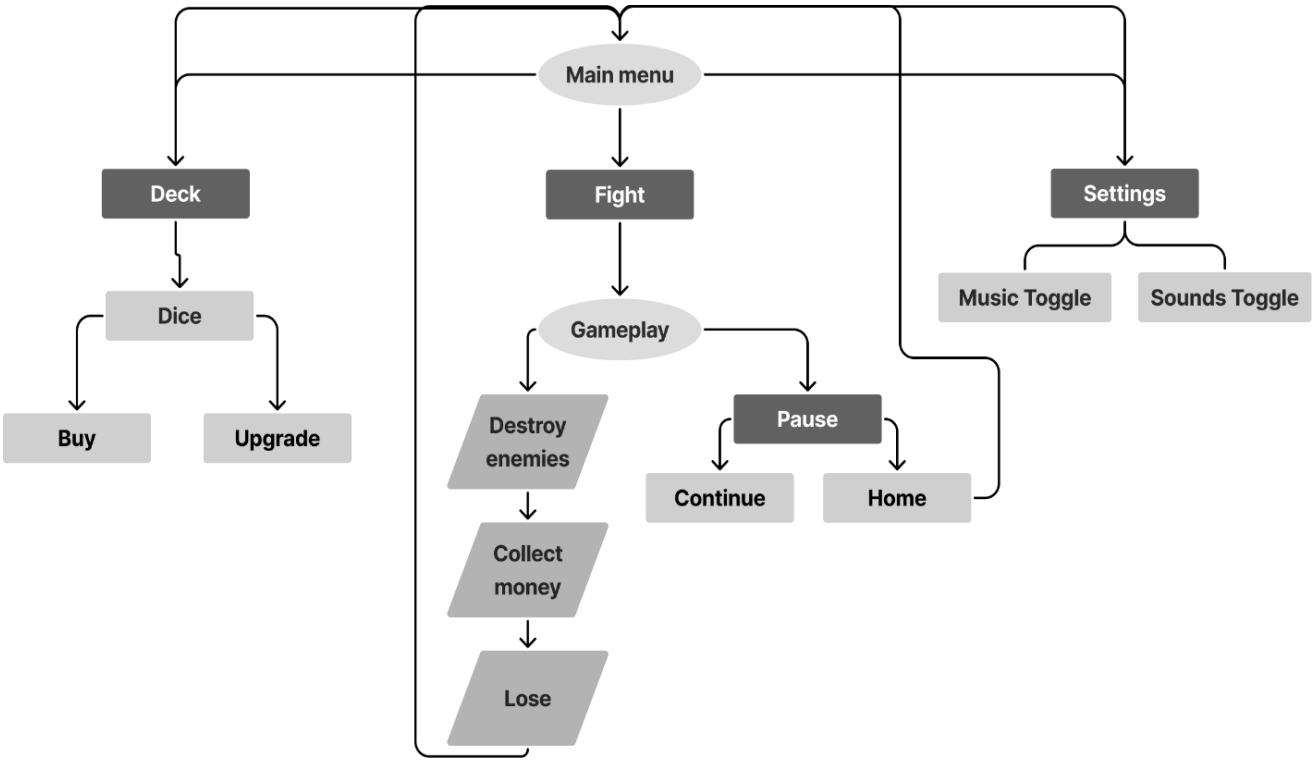


Рис.3.2 Схема діяльності

Таким чином, геймдизайн цієї гри поєднує жанрові особливості аркадного шутера з адаптованими merge-механіками, створюючи баланс між динамікою, стратегією, економікою та інтуїтивним управлінням. Це робить гру водночас доступною для новачків і захопливою для досвідчених гравців, які цінують оптимізацію ресурсів і гнучкість тактичних рішень.

3.2 Архітектура застосунку та структура ігрових модулів

Під час проєктування програмного забезпечення гри було вирішено застосувати модульний підхід із чітким розділенням обов’язків між окремими компонентами системи. Така архітектура дозволяє підвищити масштабованість, спростити відлагодження та полегшити підтримку проєкту у майбутньому. Одним із головних завдань при побудові архітектури було забезпечити логічну ізоляцію різних підсистем — таких як управління сценами, звукове оформлення, інтерфейс користувача, збереження даних, банківська система, тощо.

У якості основи для інверсії залежностей було обрано фреймворк Zenject, який дозволяє впроваджувати залежності між компонентами через інтерфейси, тим самим забезпечуючи слабку зв'язаність модулів і зручність у тестуванні. Очікується, що усі ключові сервіси гри, включно з менеджерами ресурсів, інтерфейсу, таймерів, логіки геймплею та збереження даних, будуть інжектовані через централізовану систему ініціалізації.

Також на етапі планування була передбачена потреба у плавній роботі анімацій та переходів між станами. Для цього до архітектури буде включено популярну бібліотеку DOTween, яка забезпечить ефективну реалізацію анімаційної логіки без надмірної складності.

Кожен модуль системи буде реалізовувати окремий інтерфейс, що дозволить розмежовувати відповідальність і уникнути дублювання логіки. Наприклад:

- менеджер сцен відповідатиме лише за завантаження та перехід між рівнями;
- аудіосистема — за програвання музики та звукових ефектів;
- менеджер інтерфейсу — за відображення екранів і вікон;
- ігрова логіка — за організацію послідовності подій у процесі бою;
- банківська система — за управління внутрішньоігровою валютою.

Інтерфейсна частина гри проєктується з урахуванням системи екранів і спливаючих вікон. Кожен екран — це окрема логічна одиниця, яка має свій життєвий цикл, а для спливаючих вікон буде реалізована централізована система управління. Передбачається можливість розширення інтерфейсу новими елементами без потреби змінювати вже реалізовану базову структуру.

Додатково проєктується використання пулів об'єктів, що дасть змогу уникнути зайвих витрат ресурсів на створення та знищення об'єктів під час гри — особливо важливо для мобільних платформ із обмеженими ресурсами.

Загалом запланована архітектура передбачає логічну структуру, в якій кожен модуль відповідає лише за свою частину функціоналу, а взаємодія між ними відбувається через добре визначені канали зв’язку. Такий підхід відповідає принципам чистої архітектури (Clean Architecture), коли зовнішні залежності не впливають на внутрішню логіку гри, а сама система легко адаптується до змін і доповнень.

4 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Планування розробки ПЗ

На початковому етапі створення будь-якого програмного продукту важливо визначити загальну стратегію розробки та логічну послідовність реалізації основних функціональних частин. Для розробки мобільної гри було складено покроковий план, який охоплює всі етапи — від створення ідеї до фінального тестування.

Насамперед було сформовано загальну концепцію гри. Було обрано жанр аркадного шутера з вертикальним ігровим простором, що дозволяє забезпечити динамічний геймплей із простим та інтуїтивним керуванням. Щоб урізноманітнити гравецький досвід, до гри додано систему об’єднання (merge), яка дозволяє комбінувати елементи для отримання сильніших одиниць, тим самим створюючи додаткову мотивацію до проходження рівнів і прокачки героя.

Для підвищення якості геймплею, а також адаптації складності до дій користувача, реалізовано механізм поступового ускладнення гри. Це дає змогу підтримувати інтерес гравця протягом тривалого часу, поступово підвищуючи рівень виклику.

Загальний план реалізації ігрового проєкту включав такі основні етапи:

- проєктування структури ігрової сцени та її модулів;
- налаштування початкового середовища розробки та створення стартової логіки;
- створення системи збереження прогресу гравця у форматі JSON;
- реалізація логіки руху персонажа та базових бойових механік (стрільба, шкода, захист);
- побудова системи керування ворогами та генерації хвиль;
- інтеграція системи прокачки через об’єднання однакових елементів;

- впровадження внутрішньоігрової економіки, яка включає віртуальну валюту, магазин та нагороди;
- розробка та налаштування інтерфейсу користувача: панелі керування, меню, екрани переходу;
- підключення аудіоефектів, анімацій та візуальних фідбеків;
- оптимізація коду та підвищення стабільності проєкту;
- тестування всіх компонентів та виявлення технічних помилок;
- виправлення недоліків і остаточна підготовка до релізу.

У процесі роботи застосовувалася система контролю версій Git, з розміщенням репозиторію на GitHub (див. рисунок 4.1). Це дозволило ефективно організувати збереження змін, створювати контрольні точки у проєкті та швидко повертатися до попередніх станів за потреби. На ілюстрації нижче представлено фрагмент репозиторію, що відображає прогрес реалізації гри.

Такий структурований підхід до розробки забезпечив послідовне впровадження функціоналу та зменшив ризики технічних помилок на пізніх етапах. Завдяки продуманій архітектурі, створена гра може бути розширена у майбутньому без необхідності повної перебудови системи.

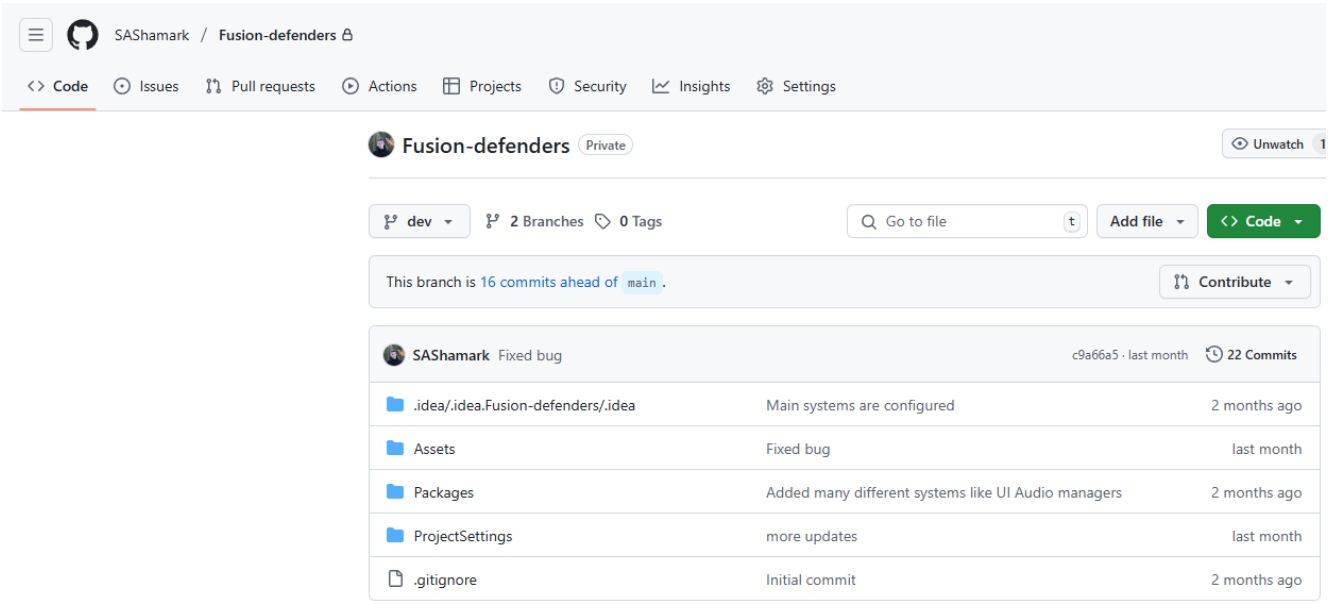


Рис. 4.1 GitHub репозиторій проєкта

4.2 Підготовка проєкту

Підготовка проєкту розпочалася з вибору рушія, на якому буде реалізовано мобільну гру. Вибір зупинився на Unity — одному з найпопулярніших ігрових рушіїв, який забезпечує ефективну розробку як 2D, так і 3D ігор для мобільних пристроїв. Було встановлено версію Unity 2022.3.53f1 LTS, що забезпечує стабільність і сумісність із довготривалими оновленнями, а також підтримку сучасних функцій та оптимізацій (див. рисунок 4.2).

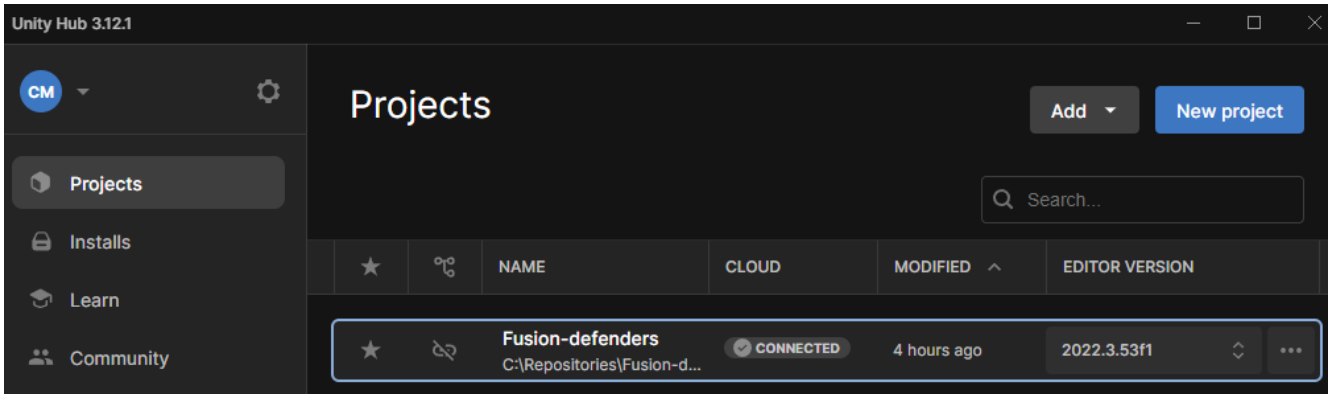


Рис. 4.2 Встановлена версія Unity 2022.3.53f1

Після встановлення рушія було виконано налаштування середовища розробки. Інтерфейс редактора Unity був адаптований під власні потреби — компоненти сцени, інспектор, вікно гри, консоль та менеджер проєктів були розміщені для зручного доступу та швидкої взаємодії (див. рисунок 4.3). Подібна кастомізація середовища дозволила зменшити час на навігацію між компонентами проєкту.

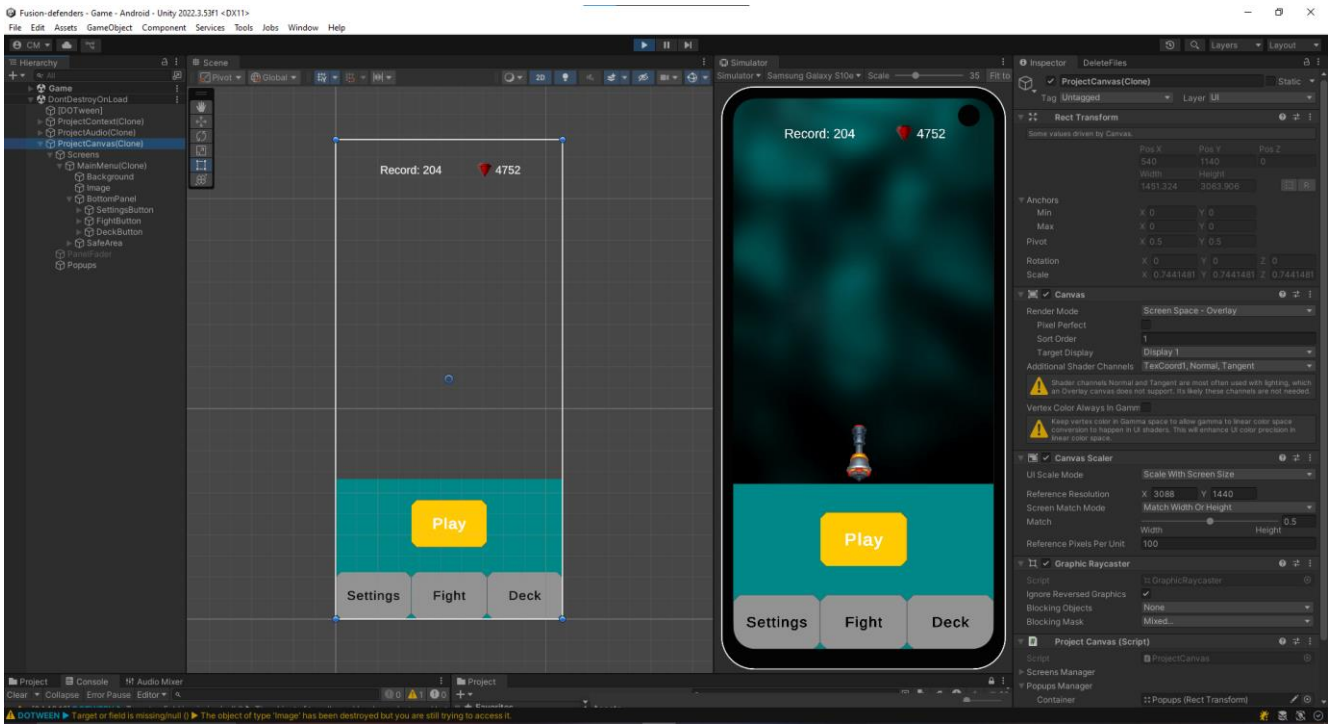


Рис. 4.3 Інтерфейс Unity після налаштування під потреби розробки

Також було підготовлено всі звукові ефекти та музичний супровід. Для цього використовувалися безкоштовні джерела з відкритим доступом, що дозволяють комерційне використання без необхідності ліцензійних відрахувань. Це забезпечило юридичну безпеку проєкту та економію часу на створення аудіоконтенту з нуля. У проєкті використовуються звуки пострілів, вибухів, натискань кнопок, звуки підсилень, а також атмосферна музика для ігрового процесу та меню.

Усі файли були організовані за категоріями: UI, снаряди, вороги, фони, підсилення, анімації, звуки, музика. Це дозволило зберегти порядок у проєкті та швидко знаходити необхідні ресурси на етапі реалізації ігрових механік (див. рисунок 4.4).

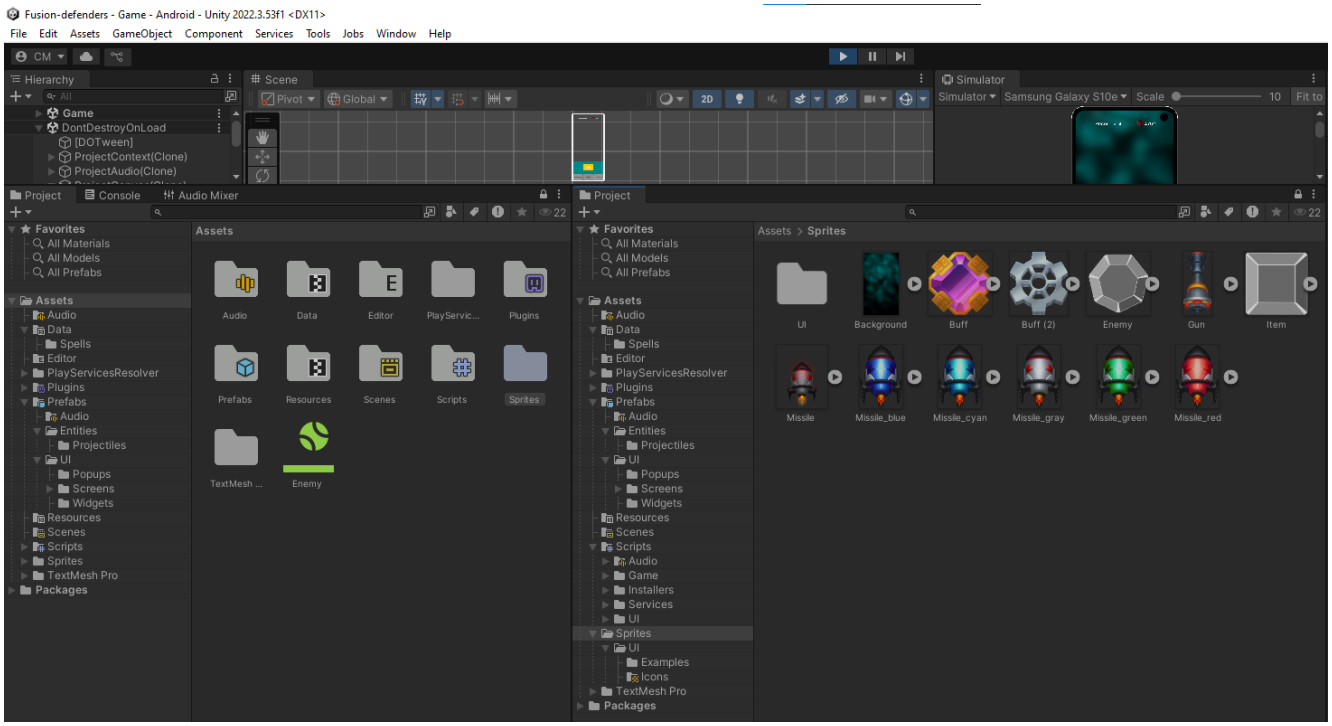


Рис. 4.4 Структура проєкту Assets у Unity

Загальна вага проєкту склала 2,33 ГБ.

Це свідчить про велику кількість унікального контенту, розробленого спеціально під потреби гри. Кожен елемент був оптимізований для мобільних пристроїв.

Протягом підготовки проєкту відбувався постійний контроль якості та сумісності графічних елементів із механіками гри, зокрема, під час створення кнопок, снарядів чи ефектів вибуху.

Таким чином, етап підготовки проєкту охоплював не лише технічне налаштування рушія, а й розробку значного обсягу унікального візуального контенту, що заклав основу для подальшої реалізації гри з єдиним стильовим рішенням та оптимізованим ігровим середовищем.

4.3 Розробка проєкту

4.3.1 Архітектура гри та загальна структура проєкту

опис шаблону побудови коду, папкової структури, принципів модульності.

1. Ініціалізація застосунку

Усі залежності створюються через ApplicationStart та CommonInstaller. Основний запуск сцени здійснюється через ApplicationStart, який через Zenject-інсталяцію ініціалізує сервіси: аудіо, зберігання, сцени, інтерфейс, таймери, банківську систему, тощо.

2. Сервіси ядра

Кожен з сервісів — IAudioManager, ITimerService, ICurrencyService, IBank, ISceneLoader, IStorageService — реалізує свій інтерфейс і відповідає за конкретну відповідальність:

- Звукова система керується AudioManager, яка використовує конфігурації ProjectAudio, AudioConfig, поділяє звуки на групи (AudioGroup, AudioGroupType, AudioMixerGroup) та контролює ефекти (EffectSoundConfig).
- Таймери реалізовані в TimerService, який використовує інтерфейс ITimerService та може бути знищений після використання (Disposable).
- Банківська система (CurrencyService, CurrencyBank) оперує валютами (CurrencyType) і дозволяє зберігати, витрачати та оновлювати грошові одиниці гравця.
- Сервіс зберігання забезпечує роботу з JSON через StorageConstants та DataProtectionUtils.

3. Керування сценами та UI

Клас SceneLoader (через ISceneLoader) відповідає за завантаження сцен та перехід між ними. Типи сцен визначаються в SceneType.

За взаємодію з інтерфейсом відповідають IUIManger, IUIScreenManager, ScreenManager та набір класів, які наслідують BaseScreen: MainMenuScreen, BattleScreen, SettingsPopup, DeciScreen, PausePopup тощо.

Всі UI-елементи типу "вікно" реалізуються через систему PopupManager, BasePopup, PopupData та їх спадкоємців (ResultPopup, PausePopup, LoadingPopup).

Окремо реалізовано логіку drag & drop для інвентарю: Slot, GridPanel, UpgradePanel, Spell, які успадковують інтерфейси для обробки подій перетягування (IDropHandler, IBeginDragHandler, IDragHandler).

4. Ігровий процес

GameplaySequence реалізує логіку послідовного запуску бою, хвиль ворогів, UI переходів.

Кожна сцена має свій контекст, а логіка переходів, запуску ворогів, нарахування винагород реалізована ізольовано від UI.

5. Об'єктні пули

Пули об'єктів (MonoObjectPool, ObjectPool) забезпечують повторне використання prefab-ів ворогів, снарядів та візуальних ефектів, зменшуючи навантаження на систему та уникнення зайвих інстанціювань.

4.3.2 Реалізація основної ігрової логіки

Фундаментальною частиною будь-якого аркадного шутера є інтерактивна взаємодія гравця з ігровим середовищем. Саме тому особливу увагу було приділено реалізації основної механіки — руху гравця, стрільби, появі ворогів хвилями, а також логіці поразки.

Управління гравцем реалізовано на основі обробки позиції пальця на екрані. Рух здійснюється не стрибками, а через плавне слідування до цільової координати, що забезпечує комфортне керування навіть на мобільних пристроях (див. рисунок 4.5).

```
1 reference
public void SliderMovementLogic(float inputXPosition)
{
    float targetX = Mathf.Lerp(-2, 2, inputXPosition);

    _moveTween?.Kill();
    _moveTween = _transform.DOMoveX(targetX, _moveDuration).SetEase(_easeType);
}

1 reference
public void MovementLogic(Vector2 inputPosition)
{
    Vector3 touchPosition = _mainCamera.ScreenToWorldPoint(new Vector3(inputPosition.x, inputPosition.y,
        _mainCamera.nearClipPlane));
    float targetX = Mathf.Clamp(touchPosition.x, -_screenHalfWidth, _screenHalfWidth);
    Vector3 position = _transform.position;
    float newX = Mathf.Lerp(position.x, targetX, _speed * Time.deltaTime);
    position = new Vector3(newX, position.y, position.z);
    _transform.position = position;
}
```

Рис. 4.5 Візуалізація логіки руху гравця

Стрільба реалізована у вигляді автоматичної генерації снарядів у відповідній позиції над гравцем (див. рисунок 4.6).

```
1 reference
public void StartShooting()
{
    _shootingCoroutine = _coroutineServices.StartRoutine(ShootingRoutine());
}

1 reference
private IEnumerator ShootingRoutine()
{
    while (true)
    {
        if (_slots is { Count: > 0 })
        {
            if (_diceShootIndex >= _slots.Count)
            {
                _diceShootIndex = 0;
            }

            var dice = _slots[_diceShootIndex].CurrentDice;
            if (dice != null)
            {
                int level = dice.Level;
                float startX = -(level - 1) * 0.5f * _projectileOffset;

                for (int i = 0; i < level; i++)
                {
                    Vector3 offset = new Vector3(startX + i * _projectileOffset, 0, 0);
                    Vector3 spawnPosition = _firePoint.position + _firePoint.right * offset.x;

                    var data = new BaseSpawnData(dice.Type.ToString(), spawnPosition);
                    BaseProjectile baseProjectile = _projectilesSpawner.Spawn(data);
                    var spellLevel = _spellsDataService.GetSpellDataByType(dice.Type).Level;
                    baseProjectile.Initialize(dice.Type, _firePoint.up, spawnPosition, _projectileSpeed,
                        spellLevel);
                    _audioManager.Play(AudioGroupType.EffectSounds, "Fire");
                }
            }

            _diceShootIndex++;
        }

        yield return new WaitForSeconds(_fireRate);
    }
}
```

Рис. 4.6 Логіка стрільби

Вороги з’являються хвилями, які збільшуються за кількістю та складністю. Для цього реалізовано менеджер хвиль, який керує генерацією ворогів із заданими параметрами — здоров’я, швидкість, тип (див. рисунок 4.7).

```
1 reference
private IEnumerator WaveSpawner(int waveIndex)
{
    var totalHp = Mathf.RoundToInt(_gameplayConfig.Health.Evaluate(waveIndex));

    var minEnemies = Mathf.RoundToInt(_gameplayConfig.EnemiesCountByWave.x);
    var maxEnemies = Mathf.RoundToInt(_gameplayConfig.EnemiesCountByWave.y);
    var cappedMax = waveIndex == 0 ? Mathf.Min(5, maxEnemies) : Mathf.Min(minEnemies + waveIndex, maxEnemies);
    var enemyCount = Random.Range(minEnemies, cappedMax + 1);

    var hpDistribution = DistributeHp(totalHp, enemyCount);

    int maxHp = hpDistribution.Max();
    int minHp = hpDistribution.Min();

    foreach (var hp in hpDistribution)
    {
        var drag = Random.Range(_gameplayConfig.EnemiesDragRange.x, _gameplayConfig.EnemiesDragRange.y);
        SpawnEnemy(hp, drag);

        float time = Mathf.InverseLerp(maxHp, minHp, hp);
        float delay = Mathf.Lerp(0.1f, 1.5f, time);
        yield return new WaitForSeconds(delay);
    }
}
```

```
1 reference
private void SpawnEnemy(int hp, float drag)
{
    var spawnPosition = _screenBoundsProvider.GetRandomTopSpawnPoint();
    spawnPosition = spawnPosition.x > 0
        ? new Vector2(spawnPosition.x - 0.2f, spawnPosition.y + _yOffset)
        : new Vector2(spawnPosition.x + 0.2f, spawnPosition.y + _yOffset);

    var data = new EnemySpawnData("Standard", spawnPosition, hp, drag);
    var enemy = _enemiesSpawner.Spawn(data);
    enemy.Health.OnDeath += () => EnemyDeath(enemy);
    _enemies.Add(enemy);
}
```

Рис. 4.7 Візуалізація хвиль ворогів

Поразка у грі настає у двох випадках: якщо хоча б один ворог досягає нижньої межі екрану, або якщо гравець втрачає усі життя. У такому випадку активується метод Die, а ігровий процес зупиняється (див. рисунок 4.8).

```
5 references
public void Damage(int value)
{
    _currentHealth -= value;
    OnHealthChanged?.Invoke(_currentHealth);

    if (_currentHealth <= 0)
    {
        Die();
    }
}

1 reference
private void Die()
{
    OnDeath?.Invoke();
}
```

Рис. 4.8 Логіка поразки

4.3.3 Система merge

Однією з ключових особливостей гри є система об’єднання ресурсів (merge), яка дозволяє гравцю стратегічно взаємодіяти з інвентарем, посилюючи свою бойову ефективність. Цей підхід поєднує класичну шутерну динаміку з механікою стратегічного зростання.

Сітка та слоти (Grid System)

Для реалізації системи об’єднання було створено сітку зі слотів (grid), у які гравець може розміщувати ігрові ресурси. Кожен слот представлений об'єктом Slot, що містить інформацію про заповнення, тип ресурса та рівень прокачки (див. рисунок 4.9).

```
1 reference
public void Enter()
{
    _movementSlider.value = _startSliderValue;

    _gridPanel.Initialize(_itemCollection);
    _gridPanel.OnDiceTableChanged += DiceTableChanged;
    _gridPanel.SpawnDice(ProjectileType.Standard);
}

1 reference
public void Exit()
{
    _movementSlider.value = _startSliderValue;
    _gridPanel.OnDiceTableChanged -= DiceTableChanged;
    _gridPanel.ClearAll();
}
```

Рис. 4.9 Логіка слотів

Коли гравець перетягує ресурс у клітинку, яка вже зайнята іншим ресурсом того самого типу й рівня, відбувається об’єднання: два однакових ресурси зливаються в один із підвищеним рівнем, а друга клітинка звільняється (див. рисунок 4.10). Цей процес дозволяє ефективніше використовувати простір сітки та стимулює гравця до стратегічного розміщення ресурсів.

```
0 references
public void OnBeginDrag(PointerEventData eventData)
{
    var startTransform = _rectTransform.parent;
    startTransform.SetAsLastSibling();

    _startParent = transform.parent;
    _startPosition = _rectTransform.anchoredPosition;
    _canvasGroup.blocksRaycasts = false;
}

0 references
public void OnDrag(PointerEventData eventData)
{
    _rectTransform.anchoredPosition += eventData.delta / _mainCanvas.scaleFactor;
}

0 references
public void OnEndDrag(PointerEventData eventData)
{
    transform.localPosition = Vector3.zero;
    _canvasGroup.blocksRaycasts = true;
}

1 reference
public bool TryMerge(Dice otherDice)
{
    if (otherDice.Level == _level && otherDice.Type == _data.Type)
    {
        _level++;
        Destroy(otherDice.gameObject);
        UpdateAppearance();
        OnMerged?.Invoke(this);
        return true;
    }

    return false;
}
```

Рис. 4.10 Логіка злиття ресурсів

4.3.4 Система збереження даних

У сучасних мобільних іграх збереження даних користувача є важливим аспектом функціональності та безпеки. У розробленому проєкті збереження прогресу реалізовано шляхом серіалізації об’єктів у формат JSON. Для запобігання несанкціонованій зміні збережених даних використано систему шифрування, що

базується на алгоритмі AES з використанням унікального ключа, сформованого на основі DeviceID та ключа гри.

Використано дві основні сервіси: StorageService — для запису/зчитування (див. рисунок 4.11), та DataProtectionManager — для шифрування/дешифрування (див. рисунок 4.12).

```
1 reference
public class StorageService : IStorageService
{
    8 references
    public void SaveData<T>(string key, T data)
    {
        string path = BuildPath(key);
        string jsonWithoutProtection = JsonConvert.SerializeObject(data, Formatting.Indented);
        string json = DataProtectionManager.Encode(jsonWithoutProtection, key);
        File.WriteAllText(path, json);
    }

    10 references
    public T LoadData<T>(string key, T defaultValue)
    {
        string path = BuildPath(key);

        if (File.Exists(path))
        {
            string jsonWithProtection = File.ReadAllText(path);
            string json = DataProtectionManager.Decode(jsonWithProtection, key);
            return JsonConvert.DeserializeObject<T>(json);
        }

        Debug.Log($"Returned default data by key [{key}] by value [{defaultValue}]");
        return defaultValue;
    }

    1 reference
    public void DeleteAllData()
    {
        string directory = Application.persistentDataPath;

        if (Directory.Exists(directory))
        {
            Directory.Delete(directory, true);
            Directory.CreateDirectory(directory);
        }
    }

    2 references
    private string BuildPath(string key)
    {
        return Path.Combine(Application.persistentDataPath, key);
    }
}
```

Рис. 4.11 StorageService.cs

```
1 reference
private static string Encrypt(string data, byte[] keyBytes)
{
    using (Aes aes = Aes.Create())
    {
        aes.Key = keyBytes;
        aes.GenerateIV();
        ICryptoTransform encryptor = aes.CreateEncryptor(aes.Key, aes.IV);

        using (MemoryStream ms = new MemoryStream())
        {
            ms.Write(aes.IV, 0, aes.IV.Length);

            using (CryptoStream cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write))
            using (StreamWriter sw = new StreamWriter(cs))
            {
                sw.Write(data);
            }

            return Convert.ToBase64String(ms.ToArray());
        }
    }
}

1 reference
private static string Decrypt(string data, byte[] keyBytes)
{
    byte[] cipherData = Convert.FromBase64String(data);

    using (Aes aes = Aes.Create())
    {
        byte[] iv = new byte[aes.BlockSize / 8];
        byte[] cipherBytes = new byte[cipherData.Length - iv.Length];

        Array.Copy(cipherData, iv, iv.Length);
        Array.Copy(cipherData, iv.Length, cipherBytes, 0, cipherBytes.Length);

        aes.Key = keyBytes;
        aes.IV = iv;
        ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV);

        using (MemoryStream ms = new MemoryStream(cipherBytes))
        using (CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read))
        using (StreamReader sr = new StreamReader(cs))
        {
            return sr.ReadToEnd();
        }
    }
}
```

Рис. 4.12 DataProtectionManager.cs

Шлях до файлів будується на базі Application.persistentDataPath, що гарантує доступність та захист інформації у файловій системі пристрою. Записані файли мають зашифрований вміст, що знижує ризик модифікації вручну.

На етапі запису об’єкт серіалізується через JsonConvert.SerializeObject(), шифрується та записується у файл. На етапі зчитування файл розшифровується та десеріалізується у відповідний тип.

Цей підхід дозволяє зберігати такі дані як: список прокачаних здібностей, кількість ресурсів, ігрові налаштування тощо. Шифрування активується лише в білдах поза Unity Editor, що дозволяє спростити тестування, не втрачаючи безпеки у релізній версії.

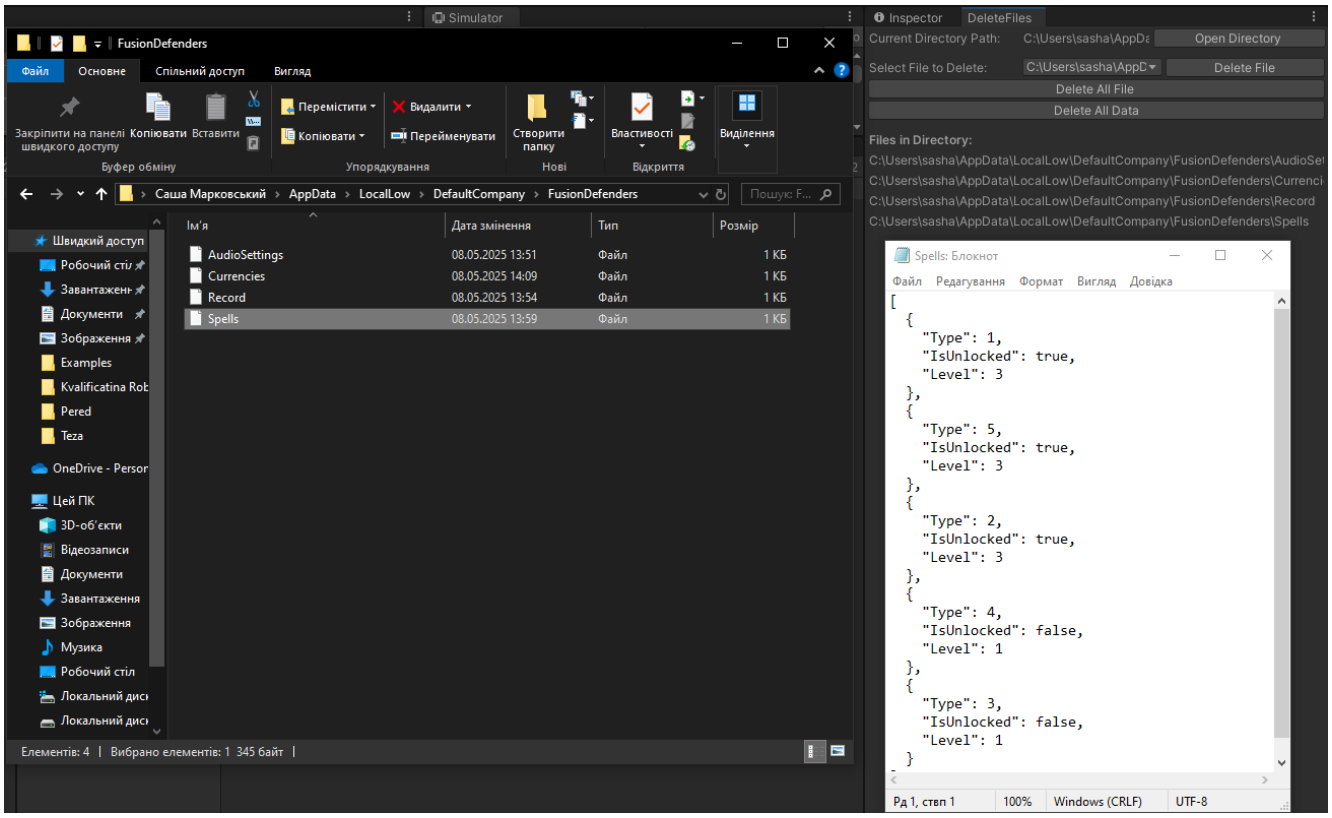


Рис. 4.13 Результат системи збереження

Таким чином, система збереження забезпечує як зручність для користувача, так і захист даних у випадку втручання з боку сторонніх осіб.

4.3.5 Реалізація UI/UX та Аудіо

Графічний інтерфейс користувача реалізовано з дотриманням принципів модульності та повторного використання. Усі вікна (екрани та попапи) базуються на спільних абстракціях BaseScreen та BasePopup, які, у свою чергу, успадковуються від загального класу BaseWindow. Такий підхід дозволяє централізовано керувати відкриттям, закриттям і анімаціями вікон.

Управління елементами інтерфейсу здійснюється через два окремих менеджери: ScreensManager (для повноекранних вікон) та PopupsManager (для діалогових вікон). Обидва менеджери реалізують відповідні інтерфейси (IScreensManager, IPopupsManager) і відповідають за ініціалізацію, черговість і взаємодію між UI-компонентами (див. рисунок 4.14).

Інтерфейс був розроблений з урахуванням принципу mobile-first. Усі інтерактивні елементи мають зручні розміри для взаємодії на сенсорних екранах, збережено чітку ієрархію розташування елементів, забезпечено коректне масштабування інтерфейсу на різних розмірах екрана — від смартфонів до планшетів.

На окремих екранах, зокрема BattleScreen та DeckScreen, реалізовано drag-and-drop функціональність. Це стало можливим завдяки використанню інтерфейсів IDragHandler, IDropHandler та похідних класів Slot, GridPanel, UpgradePanel, що дозволяють гравцю змінювати розташування.

Звукова система гри побудована на основі інтерфейсу IAudioManager, який забезпечує єдину точку доступу до всіх аудіокомпонентів. Центральним керуючим елементом є клас ProjectAudio, що реалізує логіку відтворення звуків, збереження налаштувань, змішування та взаємодії з аудіогрупами .

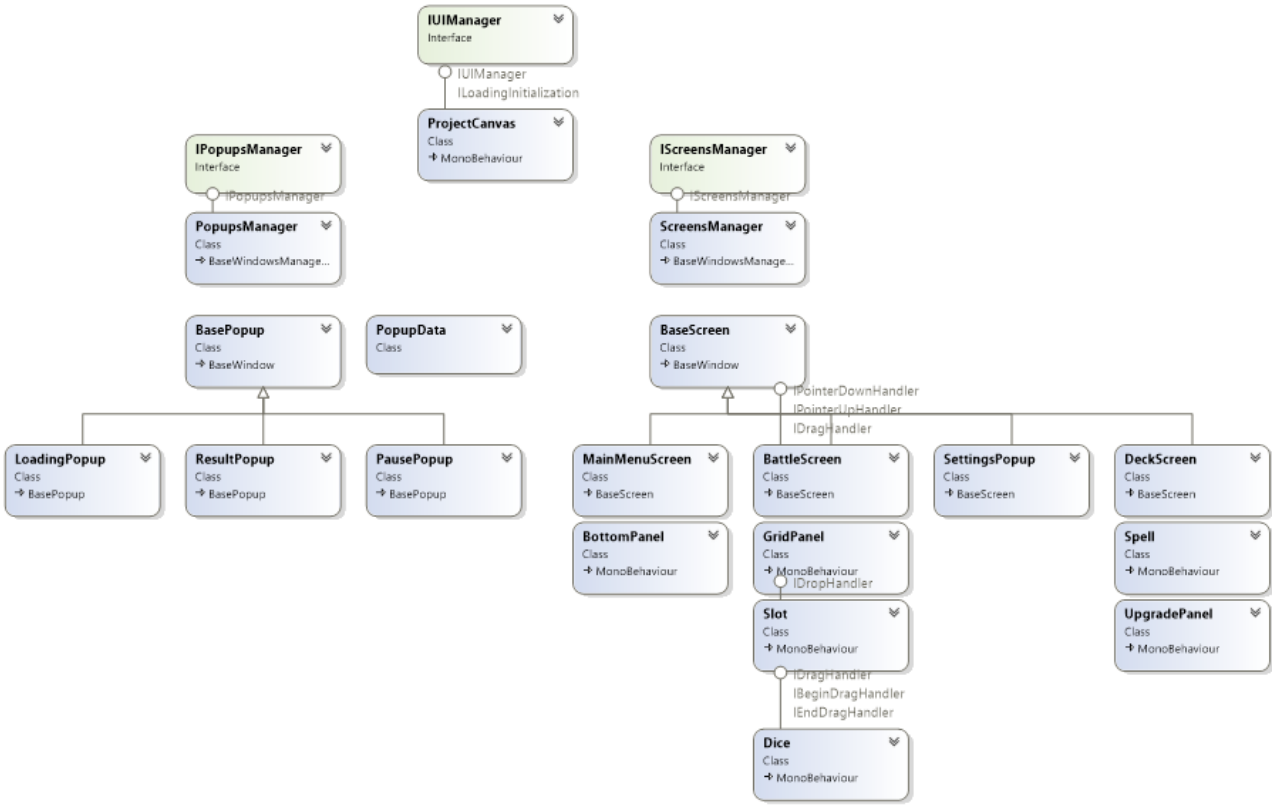


Рис. 4.14 UML-діаграми UI

Конфігурація аудіо здійснюється через об'єкт AudioConfig, який є серіалізованим ScriptableObject і містить посилання на всі необхідні звукові ресурси. Звукові ефекти групуються у AudioGroup, а фактичне зберігання та програвання організовано через AudioContainer (див. рисунок 4.15).

Система побудована з можливістю масштабування: усі типи звуків, групи мікшування та параметри ефектів задаються через перерахування (AudioGroupType, AudioMixerGroupType) і конфігураційні структури (AudioConstants, EffectSoundConfig). Це дозволяє швидко додавати нові ефекти або адаптувати звук під зміни у геймплеї без редагування основного коду.

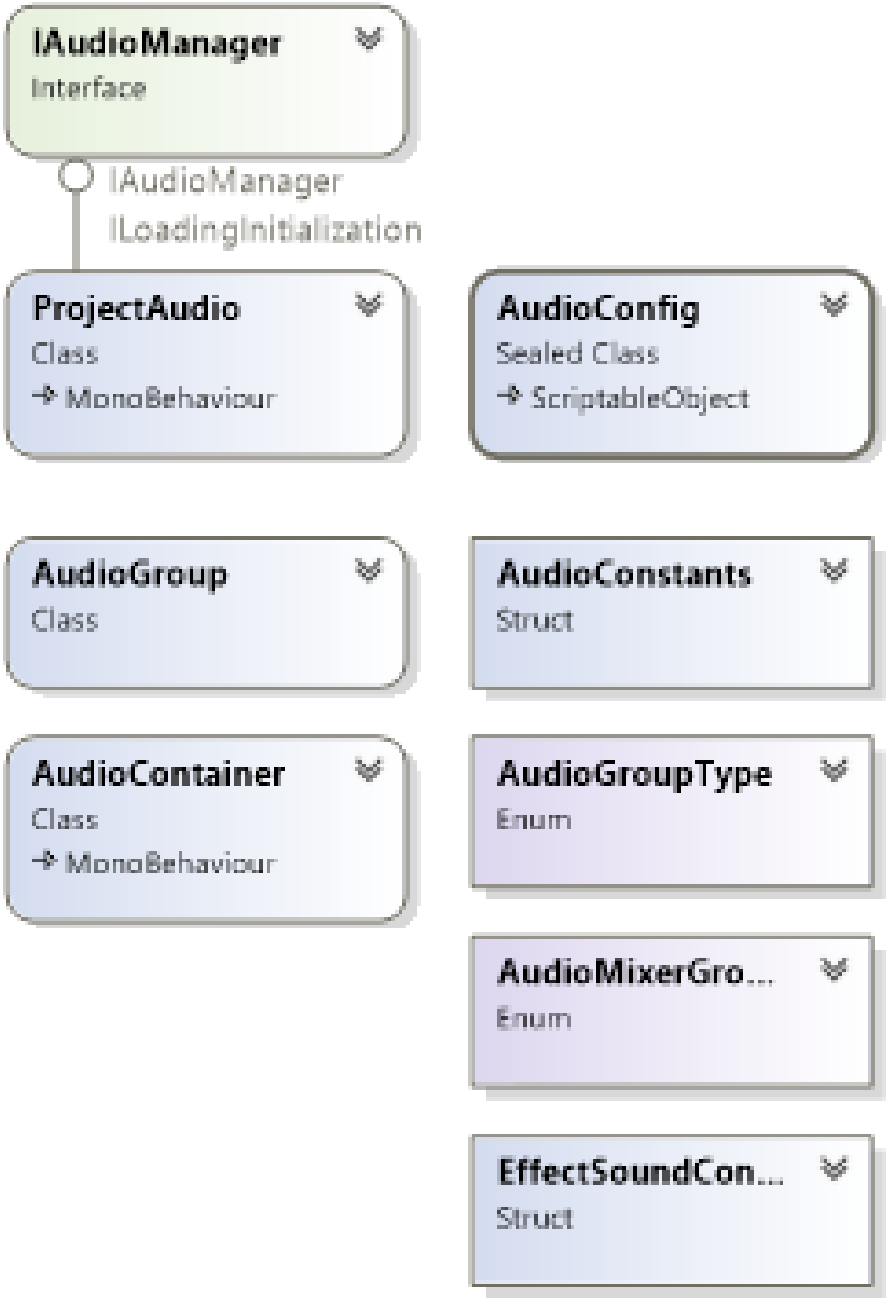


Рис. 4.15UML-діаграми аудіо

4.4 Завершення проєкту

Після реалізації основних механік гри, зокрема керування гравцем, стрільби, хвиль ворогів, системи об'єднання та збереження даних, було виконано завершальне структурування коду, оптимізація логіки та інтеграція візуального й аудіооформлення. Цей етап став ключовим для забезпечення стабільної роботи гри та підготовки її до тестування.

Протягом розробки архітектура проєкту кілька разів змінювалась відповідно до нових потреб і виявлених недоліків. Зокрема, було ухвалено рішення використовувати Zenject як систему залежностей для зручної ініціалізації модулів, що значно полегшило контроль за життєвим циклом об'єктів у грі

Основні труднощі під час завершення роботи були пов’язані з підтриманням гнучкості коду та розширюваності системи, щоб у майбутньому можна було додати нові здібності, нові типи ворогів або розширити функціонал без значного рефакторингу.

Структура проєкту побудована за принципом поділу на функціональні модулі, де кожен компонент виконує окрему відповідальність:

Робота над проектом також передбачала аналіз та оптимізацію продуктивності. Наприклад, для снарядів та ворогів використовувалась система об’єктного пулінгу, що дозволило значно знизити кількість створень та знищень об’єктів під час гри.

Таким чином, завершальний етап розробки включав не лише інтеграцію візуальних та звукових компонентів, але й фіналізацію архітектурних рішень, тестування життєвого циклу ігрових об’єктів, адаптацію до мобільних платформ та оптимізацію під низькоресурсні пристрої. Створена структура дозволяє легко масштабувати проєкт, розширювати функціонал та готувати гру до публічного релізу.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи стало створення мобільної гри у жанрі аркадного шутера з елементами механіки об’єднання (merge), що поєднує динамічний геймплей зі стратегічними елементами розвитку персонажа. Такий підхід дозволив урізноманітнити ігровий досвід, забезпечити гнучку систему прогресії та підвищити залучення гравця.

Під час роботи було досягнуто наступних *результатів:*

1. Проаналізовано предметну область, охоплено особливості жанру аркадного шутера, вивчено поточні ринкові тренди мобільних ігор, а також популярні гейміфікаційні механіки, які підвищують зацікавленість користувача. Проведено огляд популярних ігор-аналогів, визначено їх переваги та недоліки, що дозволило сформувати обґрунтовані вимоги до розроблюваного продукту.
2. Описано та обґрунтовано вибір технологій для реалізації гри: рушія Unity як оптимального середовища для мобільної кросплатформної розробки, мови C# як основного інструменту програмування логіки, JetBrains Rider як інтегрованого середовища розробки, GitHub — для контролю версій, Adobe Illustrator — для створення графіки, JSON — для зберігання ігрових конфігурацій. Цей стек забезпечив ефективну організацію робочого процесу та високу продуктивність розробки.
3. Розроблено архітектуру проєкту, визначено структуру основних ігрових модулів, включно з системою стрільби, генерацією хвиль ворогів, механікою об’єднання, збереженням прогресу гравця та UI/UX.
4. Програмно реалізовано і протестовано базові функціональні модулі гри, зокрема: рух та логіку ворогів, стрільбу, систему merge-покращень, інвентар, обробку ігрових сесій, збереження даних. Здійснено налагодження логіки складності та адаптації ворогів до рівня гравця, що дозволило створити більш збалансований та цікавий геймплей.

5. Проведено функціональне тестування, що підтвердило коректну роботу реалізованих механік, відсутність критичних помилок та готовність продукту до подальшого розширення.
6. Окрема увага приділялася забезпеченню масштабованості та модульності проєкту, що дозволяє в майбутньому легко розширити гру новими рівнями, персонажами, видами зброї, візуальними ефектами та ігровими режимами. Крім того, реалізовано зручну систему збереження прогресу, що дозволяє гравцю повертатися до гри без втрати досягнень.

Робота пройшла апробацію, що підтверджує її відповідність сучасним науково-технічним вимогам. За її результатами було підготовлено наступні тези доповідей:

1. Марковський О.П., Дібрівний О.А. Захист інформації у мобільній грі. XII Всеукраїнській науково-практичній конференції молодих учених «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ – 2025», 15 травня 2025 р., Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУІБГ, 2025. С. 299–300.
2. Марковський О.П., Дібрівний О.А.. Розробка мобільної гри в жанрі аркадного шутера. VI Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології». 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025 С. 267-268.

ПЕРЕЛІК ПОСИЛАНЬ

1. Заблоцький Б. І. *Програмування комп'ютерних ігор: теорія та практика*. – Київ: Наукова думка, 2021. – 312 с.
2. Шрайбер І. *Геймдизайн: як створювати ігри, в які будуть грати всі*. – Київ: Наш Формат, 2022. – 288 с.
3. McGonigal J. *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. – New York: Penguin Press, 2011. – 416 p.
4. Schell J. *The Art of Game Design: A Book of Lenses*. – 3rd ed. – Boca Raton: CRC Press, 2019. – 600 p.
5. Rogers S. *Level Up! The Guide to Great Video Game Design*. – 2nd ed. – Hoboken: Wiley, 2014. – 552 p.
6. Salen K., Zimmerman E. *Rules of Play: Game Design Fundamentals*. – Cambridge: MIT Press, 2003. – 688 p.
7. Fullerton T. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. – 4th ed. – Boca Raton: CRC Press, 2018. – 582 p.
8. Hocking J. *Unity in Action: Multiplatform Game Development in C#*. – 3rd ed. – Shelter Island, NY: Manning Publications, 2022. – 400 p.
9. Sensor Tower. State of Mobile Gaming 2023 [Електронний ресурс] // Sensor Tower. – Режим доступу: https://go.sensortower.com/state-of-mobile-gaming-2023
10. Statista. Global mobile game revenue by genre [Електронний ресурс] // Statista. – Режим доступу: https://www.statista.com/statistics/1221321/global-mobile-game-revenue-by-genre/
11. Newzoo. Global Games Market Report 2023 (Free Version) [Електронний ресурс] // Newzoo. – Режим доступу: https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023-free-version
12. Data.ai. State of Mobile 2024 [Електронний ресурс] // Data.ai. – Режим доступу: https://www.data.ai/en/insights/state-of-mobile-2024/

13. GameAnalytics. Mobile gaming benchmarks for Q1 2024 [Електронний ресурс] // GameAnalytics. – Режим доступу: https://www.gameanalytics.com/reports/mobile-games-benchmarks-q1-2024
14. History of mobile games [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/History_of_mobile_games
15. Mobile game [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/Mobile_game
16. Evolution of Mobile Gaming [Електронний ресурс] / MAGES Institute. – Режим доступу: https://mages.edu.sg/blog/evolution-of-mobile-gaming/
17. The Gaming Industry in 2024 by the Numbers [Електронний ресурс] // App2Top.com. – Режим доступу: https://app2top.com/news/the-gaming-industry-in-2024-by-the-numbers-a-review-by-gamesindustry-276003.html
18. 2024 Unity Gaming Report Highlights Game Studios' Continued Resilience [Електронний ресурс] // StockTitan.net. – Режим доступу: https://www.stocktitan.net/news/U/2024-unity-gaming-report-highlights-game-studios-continued-69kbn4anxpwh.html
19. Melior Games. The Value of Educational Mobile Games [Електронний ресурс] // Melior Games. – Режим доступу: https://meliorgames.com/game-development/the-value-of-educational-mobile-games/
20. AppsFlyer. Mobile game marketing: Essential strategies made simple [Електронний ресурс] // AppsFlyer. – Режим доступу: https://www.appsflyer.com/blog/tips-strategy/mobile-game-marketing/
21. The Impact of Mobile Gaming on Health – A Comprehensive Overview [Електронний ресурс] // themedicon.com. – Режим доступу: https://themedicon.com/pdf/medicalsciences/MCMS-07-245.pdf
22. Video Game Insights: Game Engines on Steam in 2025 [Електронний ресурс] // GameDev Reports Substack. – Режим доступу: https://gamedevreports.substack.com/p/video-game-insights-game-engines-on-steam-in-2025

23. The Big Game Engines Report of 2025 [Електронний ресурс] // Video Game Insights. – Режим доступу: https://vginsights.com/assets/reports/The_Big_Game_Engines_Report_of_2025.pdf
24. Global Game Engine Market to Worth Over US $ 12.84 Billion By 2033 [Електронний ресурс] // GlobeNewswire. – Режим доступу: https://www.globenewswire.com/news-release/2025/04/15/3061731/0/en/Global-Game-Engine-Market-to-Worth-Over-US-12-84-Billion-By-2033-Astute-Analytica.html
25. Unity Manual [Електронний ресурс] // Unity. – Режим доступу: https://docs.unity3d.com/Manual/
26. Unity: Creating Scripts [Електронний ресурс] // Unity. – Режим доступу: https://docs.unity3d.com/Manual/creating-scripts.html
27. C# Documentation [Електронний ресурс] // Microsoft Docs. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/
28. Unity Support in Rider [Електронний ресурс] // JetBrains. – Режим доступу: https://www.jetbrains.com/help/rider/Unity.html
29. GitHub Documentation [Електронний ресурс] // GitHub. – Режим доступу: https://docs.github.com/en
30. Adobe Illustrator User Guide [Електронний ресурс] // Adobe HelpX. – Режим доступу: https://helpx.adobe.com/illustrator/user-guide.html
31. JSON Specification [Електронний ресурс] // JSON.org. – Режим доступу: https://www.json.org/json-en.html

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільної гри в жанрі аркадного шутера

Виконав студент 4 курсу
групи ПД-44
Марковський Олександр Павлович
Керівник роботи
доктор філософії (PhD), доцент кафедри ІПЗ Дібрівний Олесь Андрійович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: вдосконалення жанру аркадного шутера через інтеграцію механіки об'єднання об'єктів.

Об'єкт дослідження: Процес розробки мобільної гри з елементами аркадного шутера та механікою поєднання.

Предмет дослідження: Програмна реалізація merge-механіки в аркадному шутері.

2

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати сучасні тенденції у сфері мобільних ігор, зокрема аркадних шутерів та merge-механік.
2. Обґрунтувати доцільність поєднання жанру аркадного шутера з механікою об'єднання об'єктів.
3. Описати архітектуру мобільного застосунку та вибір технологій для реалізації гри.
4. Розробити логіку базового геймплею із системою об'єднання ігрових об'єктів (merge-механікою).
5. Імплементувати базовий функціонал гри, включаючи бойову систему, ворогів, інтерфейс користувача та прогресію.
6. Провести тестування гри на відповідність функціональним вимогам та стабільності роботи.

3

АНАЛІЗ АНАЛОГІВ

	Ball Blast	**Archero**	**Merge Planes Idle**	**Shoot n Merge**	**Моя гра**
Мердж	-	-	+	+	+
Аркадний шутер	+	+	-	-	+
Унікальні особливості	Розділення однієї кульки на декілька з меншим запасом здоров'я	- Випадкові комбінації вмінь - Поступове зростання складності	Симулятор прокачування літаків	Геймплей подібний до гри 2048 у поєднанні з механікою стрільби з гри Bubble Shooter	Динамічність гри завдяки потребі контролювати об'єднання елементів та відстрілювати ворогів

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Можливість керувати ігровим персонажем для знищення ворогів.
2. Механіка поєднання ігрових об'єктів з метою покращення характеристик.
3. Генерація хвиль ворогів з поступовим ускладненням.
4. Нарахування внутрішньої валюти для відкриття нових здібностей та їх покращення.
5. Система збереження прогресу.

Нефункціональні вимоги:

1. Інтерфейс створено за принципом mobile-first: елементи зручні для сенсорного керування, тексти читабельні, а відображення коректне на різних розмірах екранів.
2. Програма має бути розроблена з урахуванням можливості подальшого масштабування та оновлення контенту.
3. Час запуску гри не повинен перевищувати 5 секунд

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Unity

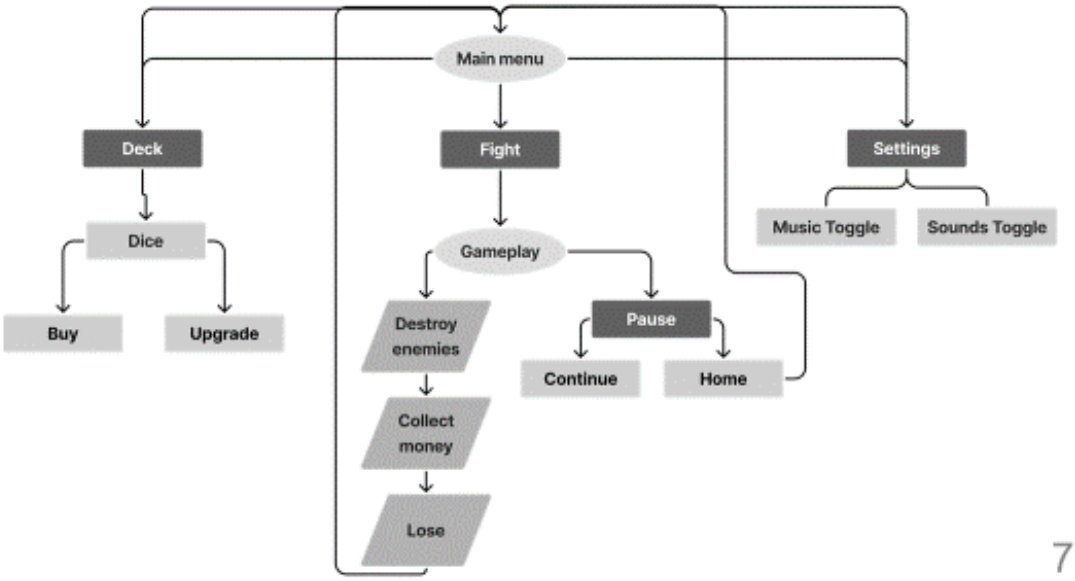
Rider

GitHub

Illustrator

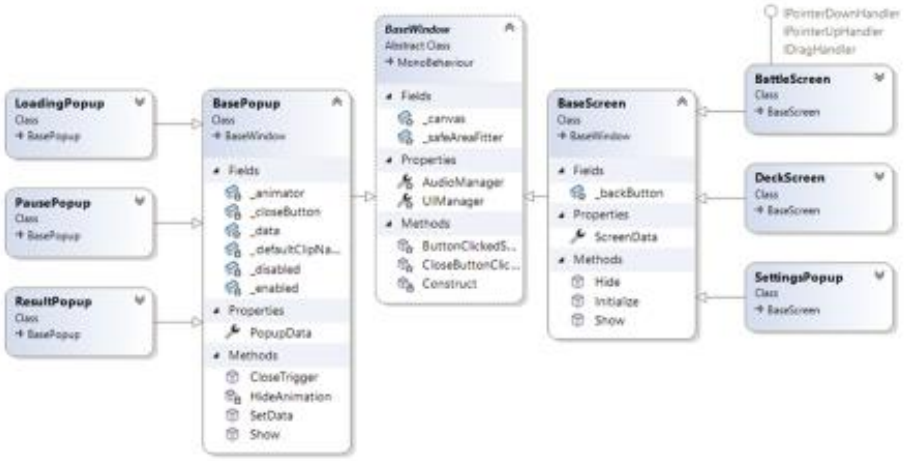
6

СХЕМА ДІЯЛЬНОСТІ



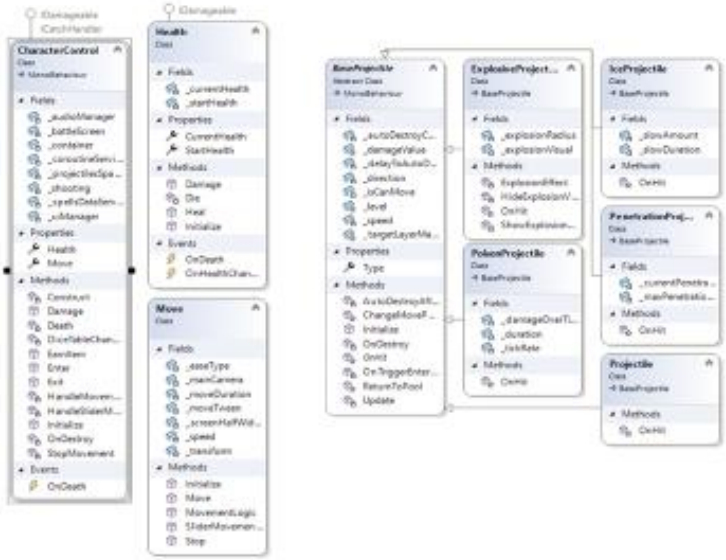
7

ДІАГРАМА UI КЛАСІВ



8

ДІАГРАМА КЛАСІВ ГОЛОВНОГО ПЕРСОНАЖА



9

ДІАГРАМА КЛАСІВ СПАВНА ПРОТИВНИКІВ ТА РЕСУРСІВ

10

ЕКРАННІ ФОРМИ

Налаштування

Головне меню

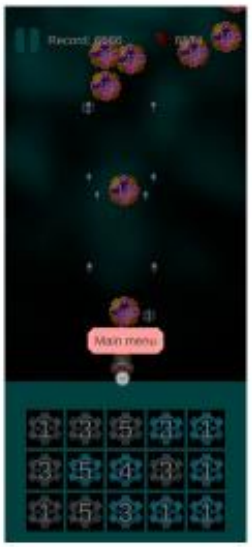
Конфігурація

11

ЕКРАННІ ФОРМИ

Рання стадія битви

Пізня стадія битви



Кінець битви

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Марковський О.П., Дібрівний О.А. Захист інформації у мобільній грі. XII Всеукраїнській науково-практичній конференції молодих учених «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ – 2025», 15 травня 2025 р., Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУІБГ, 2025. С. 299–300.
2. Марковський О.П., Дібрівний О.А.. Розробка мобільної гри в жанрі аркадного шутера. VI Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології». 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025 С. 267-268.

13

ВИСНОВКИ

1. Проаналізовано аркадні шутери та ігри з механікою merge, визначено ключові тенденції, що сприяють зростанню популярності таких гібридних проєктів.
2. Визначено функціональні та нефункціональні вимоги до гри, на основі яких спроєктовано архітектуру із підтримкою масштабування та швидкого запуску.
3. Реалізовано мобільну гру на Unity з використанням C#, у якій поєднано динамічний шутерний геймплей із механікою об'єднання об'єктів для прокачки.
4. Створено адаптивний інтерфейс, систему прогресу, хвиль ворогів, внутрішньоігрову економіку та збереження даних.
5. Проведено тестування ігрових механік, забезпечено стабільну роботу застосунку на середньостатистичних мобільних пристроях.
6. Підготовлено звітну документацію та презентаційні матеріали; захист проєкту дозволив підтвердити практичні навички повного циклу розробки мобільних ігор.

14

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using System.Collections;
using System.Collections.Generic;
using Services;
using Services.Scenes;
using UI.Managers;
using UI.Popups;
using UI.Screens;
using UI.Screens.Variables;
using UnityEngine;
using Zenject;

public class ApplicationStart : MonoBehaviour
{
    [SerializeField]
    private SceneContext _sceneContext;

    [SerializeField]
    private IntroScreen _introScreen;

    private List<ILoadingInitialization> _initList;
    private ISceneLoader _sceneLoader;
    private Coroutine _initializeCoroutine;
    private IUIManager _uiManager;

    private const float INTRO_SCREEN_START_PROGRESS = 0.1f;
    private const float INIT_LIST_PROGRESS_INCREMENT = 0.7f;
    private const float INTRO_SCREEN_DELAY = 0.2f;
    private const float FINAL_PROGRESS = 1.0f;

    [Inject]
    private void Construct(List<ILoadingInitialization> initList, ISceneLoader
sceneLoader, IUIManager uiManager)
    {
        _initList = initList;
        _sceneLoader = sceneLoader;
        _uiManager = uiManager;
    }

    private void Start()
    {
        _initializeCoroutine = StartCoroutine(Initialize());
    }

    private void OnDestroy()
    {
        if (_initializeCoroutine != null)
        {
            StopCoroutine(_initializeCoroutine);
        }
    }

    private IEnumerator Initialize()
    {
        _introScreen.Init();
        yield return null;

        RunSceneContext();
```

```
        yield return InitializeLoadingComponents();
        yield return ShowNextScene();
    }

    private void RunSceneContext()
    {
        _introScreen.UpdateProgress(INTRO_SCREEN_START_PROGRESS / 2);
        _sceneContext.Run();
        _introScreen.UpdateProgress(INTRO_SCREEN_START_PROGRESS / 2);
    }

    private IEnumerator InitializeLoadingComponents()
    {
        for (var index = 0; index < _initList.Count; index++)
        {
            _initList[index].Init();
            float progress = INTRO_SCREEN_START_PROGRESS +
                             INIT_LIST_PROGRESS_INCREMENT * (index + 1) /
_initList.Count;
            _introScreen.UpdateProgress(progress);
            yield return null;
        }
    }

    private IEnumerator ShowNextScene()
    {
        yield return new WaitForSeconds(INTRO_SCREEN_DELAY);
        _introScreen.UpdateProgress(FINAL_PROGRESS);
        _uiManager.PopupsManager.ShowPopup(PopupTypes.Loading);

        _sceneLoader.LoadScene(SceneType.Game, ScreenTypes.MainMenu, false);
    }
}

using UI.Popups;
using UI.Screens;

namespace UI.Managers
{
    public interface IUIManager
    {
        IScreensManager ScreensManager { get; }
        IPopupsManager PopupsManager { get; }
    }
}

using Audio;
using Services;
using Services.Currency;
using UI.Popups;
using UI.Screens;
using UnityEngine;
using Zenject;

namespace UI.Managers
{
    public class ProjectCanvas : MonoBehaviour, IUIManager, ILoadingInitialization
    {
        [SerializeField]
        private ScreensManager _screensManager;

        [SerializeField]
```

```
        private PopupsManager _popupsManager;

        [SerializeField]
        private WindowsConfig _windowsConfig;

        private DiContainer _diContainer;
        private CurrencyService _currencyService;
        private IAudioManager _audioManager;

        public IScreensManager ScreensManager => _screensManager;
        public IPopupsManager PopupsManager => _popupsManager;

        [Inject]
        private void Construct(DiContainer diContainer, CurrencyService
currencyService, IAudioManager audioManager)
        {
            _diContainer = diContainer;
            _currencyService = currencyService;
            _audioManager = audioManager;
        }

        public int Priority => 1;

        public void Init()
        {
            DontDestroyOnLoad(gameObject);
            _screensManager.Initialize(_diContainer);
            _popupsManager.Initialize(_diContainer);
            ConfigLoaded();
        }

        private void ConfigLoaded()
        {
            _screensManager.OnConfigLoaded(_windowsConfig);
            _popupsManager.OnConfigLoaded(_windowsConfig);
        }
    }
}

using System;

namespace UI.Popups
{
    public interface IPopupsManager
    {
        event Action<PopupTypes> OnPopupShowed;
        event Action<PopupTypes> OnPopupHidden;
        void ShowPopup(PopupTypes popupType);

        void HidePopup(PopupTypes popupType);
        BasePopup GetPopup(PopupTypes popupType);

        void HideAllPopups();
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using UI.Managers;
using UnityEngine;
using Object = UnityEngine.Object;
```

```
namespace UI.Popups
{
    [Serializable]
    public class PopupsManager : BaseWindowsManager<BasePopup>, IPopupsManager
    {
        [SerializeField]
        private GameObject _popupFader;

        private Dictionary<PopupTypes, PopupModelData> _popupModelsParsed = new();
        private List<BasePopup> _currentPopups = new();
        public event Action<PopupTypes> OnPopupShowed;
        public event Action<PopupTypes> OnPopupHidden;

        public override void OnConfigLoaded(WindowsConfig windowConfig)
        {
            base.OnConfigLoaded(windowConfig);

            foreach (PopupModelData pmd in WindowConfig.PopupModels)
            {
                if (!_popupModelsParsed.TryAdd(pmd.PopupType, pmd))
                {
                    Debug.LogError("There is already setup up " + pmd.PopupType + "
cant fill it twice!");
                }
            }
        }


        public void ShowPopup(PopupTypes popupType)
        {
            if (!_popupModelsParsed.ContainsKey(popupType))
            {
                Debug.LogError("Not filled page by type: " + popupType);
                return;
            }

            PopupModelData popupModelData = _popupModelsParsed[popupType];
            var basePopup =

DiContainer.InstantiatePrefabForComponent<BasePopup>(popupModelData.Template,
_container);

            if (basePopup == null)
            {
                Debug.LogError(
                    "There is no BasePopup attached to : " +
basePopup.gameObject.name + " of type " + popupType);
                return;
            }

            basePopup.PopupData = popupModelData;

            AddPopup(basePopup);
            OnPopupShowed?.Invoke(popupType);
        }

        public void HidePopup(PopupTypes popupType)
        {
            BasePopup currentPopup = _currentPopups.FirstOrDefault(r =>
r.PopupData.PopupType == popupType);
            if (currentPopup != null)
```

```
            {
                RemovePopup(currentPopup);
                OnPopupHidden?.Invoke(popupType);
            }
            else
            {
                Debug.LogError("There is no popup by type: " + popupType + " active
now!");
            }
        }

        private void AddPopup(BasePopup popup, PopupData data = null)
        {
            _currentPopups.Add(popup);

            if (_currentPopups.Count >= 1)
            {
                bool anyUseTotalFader = _currentPopups.Any(currentPopup =>
currentPopup.PopupData.UseTotalFader);
                if (anyUseTotalFader)
                {
                    _popupFader.SetActive(true);
                }
            }

            popup.SetData(data);
            popup.Show();
        }

        private void RemovePopup(BasePopup popup)
        {
            if (_currentPopups.Contains(popup))
            {
                _currentPopups.Remove(popup);
                bool anyUseTotalFader = _currentPopups.Any(currentPopup =>
currentPopup.PopupData.UseTotalFader);

                if (!anyUseTotalFader || _currentPopups.Count <= 0)
                {
                    _popupFader.SetActive(false);
                }
                else
                {
                    ShowNextPopup();
                }
            }

            Object.Destroy(popup.gameObject);
        }

        private void ShowNextPopup()
        {
            BasePopup newTopPopup = GetTopPopup();
            if (newTopPopup != null)
            {
                newTopPopup.Show();
            }
        }

        public BasePopup GetPopup(PopupTypes type) =>
            _currentPopups.FirstOrDefault(popup => type ==
popup.PopupData.PopupType);
```

```
        private BasePopup GetTopPopup() => _currentPopups[^1];

        public void HideAllPopups()
        {
            if (_currentPopups is { Count: > 0 })
            {
                foreach (BasePopup popup in _currentPopups)
                {
                    RemovePopup(popup);
                }
            }
        }
    }
}

using System;
using UI.Screens.Base;

namespace UI.Screens
{
    public interface IScreensManager
    {
        event Action<ScreenTypes> OnScreenShowed;
        void ShowScreen(ScreenTypes screenTypes, bool isPrevious = false);
        void ShowPreviousScreen();
        void RemoveAllScreens();
        BaseScreen GetScreen(ScreenTypes screenType);
    }
}

using System;
using System.Collections.Generic;
using UI.Managers;
using UI.Screens.Base;
using UnityEngine;

namespace UI.Screens
{
    [Serializable]
    public class ScreensManager : BaseWindowsManager<BaseScreen>, IScreensManager
    {
        private readonly Dictionary<ScreenTypes, ScreenModelData>
_screenModelsParsed = new();
        private readonly Dictionary<ScreenTypes, BaseScreen> _screens = new();
        private readonly Stack<BaseScreen> _previousScreens = new();
        public event Action<ScreenTypes> OnScreenShowed;

        public BaseScreen GetScreen(ScreenTypes screenType)
        {
            BaseScreen baseScreen = null;
            foreach (KeyValuePair<ScreenTypes, BaseScreen> screen in _screens)
            {
                if (screen.Key == screenType)
                {
                    baseScreen = screen.Value;
                }
            }

            return baseScreen;
        }
```

```
        public override void OnConfigLoaded(WindowsConfig windowConfig)
        {
            base.OnConfigLoaded(windowConfig);

            foreach (ScreenModelData screenModelData in WindowConfig.ScreenModels)
            {
                if (!_screenModelsParsed.TryAdd(screenModelData.ScreenType,
screenModelData))
                {
                    Debug.LogError("There is already set up " +
screenModelData.ScreenType + " cant fill it twice!");
                }
            }
        }

        public void ShowScreen(ScreenTypes screenTypes, bool isPrevious = false)
        {
            if (ActiveWindow != null)
            {
                if (ActiveWindow.ScreenData.IsAddToStack)
                {
                    if (!isPrevious)
                    {
                        _previousScreens.Push(ActiveWindow);
                    }

                    ActiveWindow.Hide();
                }
                else
                {
                    RemoveScreen(ActiveWindow);
                    _screens.Remove(ActiveWindow.ScreenData.ScreenType);
                }
            }

            if (_screens.TryGetValue(screenTypes, out BaseScreen screen))
            {
                screen.Show();
                ActiveWindow = screen;
            }
            else
            {
                AddScreen(screenTypes);
            }

            OnScreenShowed?.Invoke(screenTypes);
        }

        private void AddScreen(ScreenTypes screenType)
        {
            if (!_screenModelsParsed.ContainsKey(screenType))
            {
                Debug.LogError("Not filled screen by type: " + screenType);
                return;
            }

            ScreenModelData screenModelData = _screenModelsParsed[screenType];
            var baseScreen =

DiContainer.InstantiatePrefabForComponent<BaseScreen>(screenModelData.Template,
_container);
            if (baseScreen == null)
```

```
            {
                Debug.LogError(
                    "There is no BasePage attached to : " +
baseScreen.gameObject.name + " of type " + screenType);
                return;
            }

            baseScreen.ScreenData = screenModelData;
            _screens.Add(screenType, baseScreen);
            ActiveWindow = baseScreen;
            baseScreen.Initialize();
            baseScreen.Show();
        }

        public void ShowPreviousScreen()
        {
            if (_previousScreens.Count > 0)
            {
                BaseScreen previousScreenType = _previousScreens.Pop();
                ShowScreen(previousScreenType.ScreenData.ScreenType, true);
            }
        }

        public void RemoveAllScreens()
        {
            if (ActiveWindow != null)
            {
                RemoveScreen(ActiveWindow);
            }

            for (int i = 0; i < _previousScreens.Count; i++)
            {
                RemoveScreen(_previousScreens.Pop());
            }
        }
    }
}
```