МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Розробка мобільного ігрового додатку Libtake з використанням сучасних індустріальних інструментів та бібліотек в середовищі ігрового рушія Unity»**

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Москалик Максим Миколайович

Керівник: *викладач кафедри інформаційних технологій та аналітики даних*
Місай Володимир Віталійович

Рецензент: *ФОП в галузі інформаційних технологій, викладач кафедри менеджменту та маркетингу НаУОА*
Шпак Андрій Григорович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)
Протокол №10 від 28 травня 2025 р.

Острог, 2025

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: *Розробка мобільного ігрового додатку Libtake з використанням сучасних індустріальних інструментів та бібліотек в середовищі ігрового рушія Unity.*

Автор: *Москалик Максим Миколайович*

Науковий керівник: *Місай Володимир Віталійович*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: 56 *(кількість сторінок роботи) с., 5 (кількість рисунків) рис., 0 (кількість таблиць) табл., 0 (кількість додатків) додатків, 7 (кількість джерел) джерел.*

Ключові слова: *МОБІЛЬНА ГРА, UNITY, КОМПОНЕНТНО-ОРІЄНТОВАНА АРХІТЕКТУРА, ZENJECT, UNITASK, DOTWEEN, СТЕЙТ-МАШИНА, ІГРОВА РОЗРОБКА.*

Короткий зміст праці:

Кваліфікаційна робота присвячена розробці мобільного ігрового додатку Libtake для платформи Android з використанням ігрового рушія Unity та сучасних індустріальних бібліотек. У роботі проведено аналіз предметного середовища функціонування комп'ютерних ігор та огляд аналогічних рішень. Спроектовано архітектуру гри на основі компонентно-орієнтованого підходу з використанням стейт-машини для керування життєвим циклом додатку. Розроблено систему збереження даних через JSON-серіалізацію та впроваджено механізми динамічного балансування ігрового процесу. Реалізовано ефективні алгоритми для мобільних пристроїв з використанням асинхронного програмування через UniTask та впровадження залежностей через Zenject. Ігровий додаток моделює діяльність бібліотеки Острозької академії та спрямований на залучення абітурієнтів, демонструючи освітні можливості університету в інтерактивному форматі.

The qualification work is dedicated to the development of a mobile game application Libtake for the Android platform using the Unity game engine and modern

industrial libraries. The paper analyzes the subject environment of computer games functioning and provides an overview of similar solutions. The game architecture was designed based on a component-oriented approach using a state machine to manage the application lifecycle. A data storage system was developed through JSON serialization and mechanisms for dynamic game balance were implemented. Efficient algorithms for mobile devices were implemented using asynchronous programming through UniTask and dependency injection through Zenject. The game application simulates the activities of the Ostroh Academy library and aims to attract prospective students by demonstrating the educational opportunities of the university in an interactive format.

(підпис автора)

ЗМІСТ

ВСТУП

Комп’ютерні ігри загалом є найбільшою частиною розважальної індустрії, яка генерує більше грошей, аніж виробництво фільмів та музики разом взяті, що підтверджується даними журнала Forbes[1].

Мільярди геймерів по всьому світу генерують все більші і більші доходи, і кількість цих людей зростає. За різними оцінками, близько 40% населення планети грає в ігри час від часу. Левова частка цих людей надає перевагу саме мобільним іграм через можливість пограти в транспорті, в черзі, під час прогулянки та завдяки високій адиктивності, котра досягається короткотривалими ігровими сесіями.

До 2027 року прогнозується, що кількість мобільних геймерів зросте до 2,32 млрд осіб, що робить ринок ігор для смартфонів одним з найбільш привабливим для розробників.

Враховуючи залученість людей до гейміфікації різноманітних процесів, ігри можна використовувати не лише для заробітку а й для некомерційних цілей як, наприклад, підняття популярності університету, залучення нових абітурієнтів; дослідження сучасних технологій.

Темою дипломної роботи є розробка мобільного ігрового додатку Libtake для платформи Android з використанням сучасних індустріальних практик та інструментів в середовищі ігрового рушія Unity.

Мета дослідження полягає у знаходженні оптимальних методів для створення ігрових додатків та практичній реалізації мобільної гри з використанням інноваційних технологій та рішень.

Об’єктом дослідження є мобільний ігровий додаток для платформи Android, який розроблено для залучення абітурієнтів до вищого навчального закладу.

Предметом дослідження є створення мобільного ігрового додатку під платформу Android із застосуванням ігрового рушія Unity, мови програмування C# та бібілотек Zenject, UniTask, DoTween, Newtonsoft JSON та інших, а також

дослідження тенденцій, сучасних методологій та інструментів для розробки ігор з їх практичним застосуванням.

Актуальність теми обґрунтована популярністю мобільних ігор на ринку та значною часткою ігрового рушія Unity, яка, за оцінками, становить близько 27,02% серед розробників ігор. Розробка інноваційних ігрових додатків, які об’єднують елементи стратегічного планування та творчого підходу до вирішення завдань, є важливим етапом у розвитку сучасної індустрії мобільних ігор.

Значення теми полягає у необхідності дослідження ефективних практик та інструментів для розробки мобільних ігор; інтеграції освітніх і розважальних компонентів в ігрові додатки; пошуку способів мінімізації витрачених часових ресурсів з метою зменшення витрат для бізнесу та підвищення рентабельності розробки ігор, підвищенню конкурентоспроможності вітчизняних розробників ігор на світовому ринку, пошук нових підходів для більш ефективного використання ресурсів пристроїв гравців; що сприятиме підвищенню кваліфікації молодих розробників; покращенню якості освіти; розвитку освітніх закладів; також покращенню конкурентоспроможності компаній на ринку мобільних додатків, особливо в умовах стрімкого зростання попиту на якісні ігрові продукти; прискоренню розвитку національної економіки, що, в свою чергу призведе до зростання ВВП та рівня життя загалом.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

В останні декади, індустрія розробки комп'ютерних ігор перетворилась з нішового хобі, котре доступне тільки для найбільш досвідчених програмістів, що мусили детально знати будову комп'ютера, математику, радіоелектроніку та фізику; в значний економічний та культурний феномен. Вже в 2015 році кількість людей, що грали в ігри, становила 2 млрд осіб, станом на кінець 2023 року, вона збільшилась до 3.220 млрд осіб[2], що уже становить приблизно 40% населення планети[3], а до кінця 2025 прогнозується зростання до 3.4 млрд осіб або приблизно 41.5% населення планети.

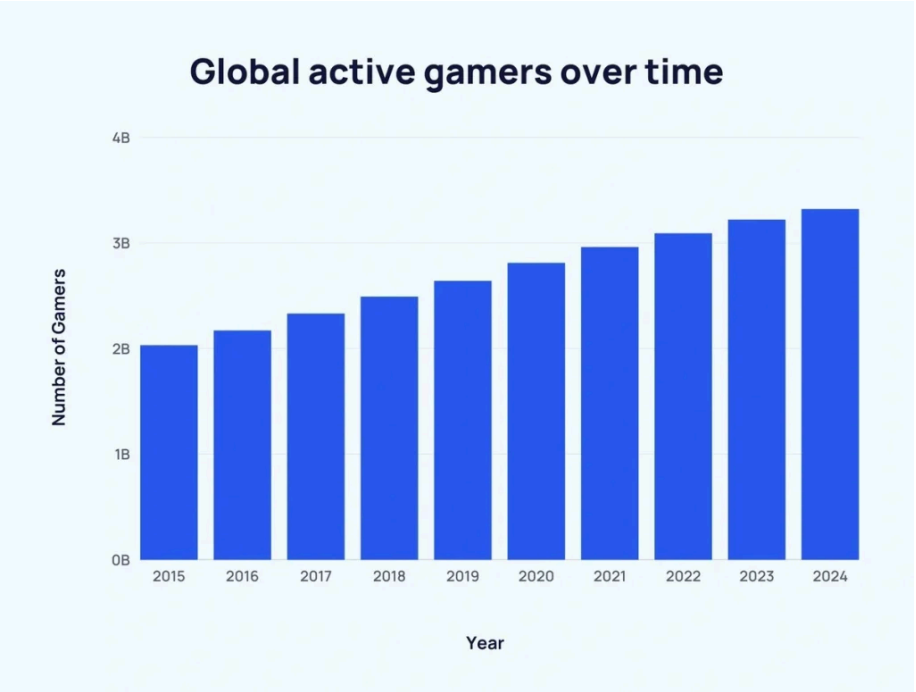

Рис 1.1. Графік зростання кількості геймерів у світі в млрд. осіб за 2015-2024 рр.
Джерело: https://whatsthebigdata.com/number-of-gamers/(дата відвідування: 28.04.2024)

Ця статистика сигналізує те, що відеоігри стали вкрай важливою частиною життя для значної кількості людей. Відповідно до дослідження The Entertainment Software Association[4]:

69% людей грають задля розваги;

36% для саморозвитку;

27% щоб дослідити нові світи та ідеї.

Це все призвело до того, що в останні роки індустрія відеоігор продовжила своє стрімке зростання, перетворившись на один із найбільш прибуткових секторів глобальної економіки. Ринок відеоігор продемонстрував значне фінансове зростання, що перевищує сукупні доходи кіноіндустрії та музичної індустрії, підкреслюючи домінуюче положення ігрової індустрії в сфері розваг[1].

Важливо відзначити, що пандемія COVID-19 стала каталізатором для ще більшого зростання популярності відеоігор. Періоди локдаунів та соціальної ізоляції змусили людей шукати нові форми розваг та соціальної взаємодії в цифровому просторі. Час, проведений за іграми, значно збільшився під час піку пандемії, а багато нових користувачів, які почали грати в цей період, продовжили це робити і після зняття обмежень.

Все більше освітніх закладів впроваджують чисельні елементи гейміфікації в навчальний процес, визнаючи потенціал ігор для розвитку когнітивних здібностей, критичного мислення та навичок вирішення проблем. Тим більше, що це збільшує залученість студентів до навчального процесу, мотивуючи досягати більших результатів, коли навчання або його елементи подані у вигляді гри. Численні дослідження показують, що помірна гра у відеоігри може мати позитивний вплив на розвиток мозку, покращувати координацію рук і очей, просторове сприйняття та здатність до багатозадачності.

Сучасний ринок розробки ігор характеризується двома паралельними тенденціями: з одного боку, концентрація ресурсів у руках великих компаній, які створюють високобюджетні проекти (AAA-ігри) з бюджетами, що часто сягають сотень мільйонів доларів; з іншого боку – розквіт незалежної розробки ігор (інді-сцена), де малі команди або навіть окремі розробники створюють інноваційні проєкти, використовуючи доступні інструменти розробки та цифрові платформи дистрибуції.

Технологічні досягнення останніх років також значно вплинули на індустрію. Розвиток хмарних технологій дозволив створити сервіси потокової передачі ігор,

такі як Google Stadia, Nvidia GeForce Now та Xbox Cloud Gaming, які дають можливість грати у високоякісні ігри без потужного апаратного забезпечення. Крім того, технології віртуальної (VR) та доповненої реальності (AR) відкрили нові горизонти для ігрових розробників, створюючи безпрецедентний рівень занурення гравців.

Феномен інді-ігор заслуговує особливої уваги, оскільки він кардинально змінив ландшафт ігрової індустрії протягом останнього десятиліття. Це ігри, що створюються окремими розробниками або невеликими командами без фінансової підтримки великих видавців. Спочатку це явище було маргінальним сегментом ринку, але з появою цифрових платформ дистрибуції, таких як Steam, Epic Games Store, GOG та консольних магазинів (PlayStation Store, Nintendo eShop, Xbox Store), інді-розробники отримали прямий доступ до масового споживача.

Одним з ключових факторів, що сприяв розвитку інді-сцени, стала демократизація інструментів розробки ігор. Сучасні ігрові рушії, такі як Unity та Unreal Engine, які раніше були доступні лише великим студіям, тепер пропонують безкоштовні або дуже доступні ліцензії для незалежних та малого бізнесу розробників. Крім того, численні онлайн-ресурси, курси, книги та спільноти розробників дозволяють достатньо швидко опанувати необхідні навички без формальної освіти в галузі програмування чи дизайну.

Інді-ігри часто відрізняються від AAA-проєктів своїм творчим та іноваційним підходом, експериментальним геймплеєм та унікальною художньою стилістикою. Не маючи необхідності задовольняти вимоги масового ринку або окупати багатомільйонні інвестиції, незалежні розробники можуть дозволити собі ризики та інновації, які великі студії вважають занадто небезпечними. Це призвело до появи численних креативних проєктів, які згодом вплинули на розвиток усієї галузі.

Інді-сцена також стала інкубатором для інноваційних бізнес-моделей, таких як краудфандинг через платформи Kickstarter та Indiegogo, що дозволяє розробникам фінансувати свої проєкти за рахунок підтримки майбутніх гравців. Модель раннього доступу, коли гра випускається в незавершеному стані, а потім допрацьовується з

урахуванням відгуків спільноти, також здебільшого популяризована саме інді-розробниками.

1.2 Аналіз аналогічних розробок

Аналогічних розробок розроблених в українських університетах знайдено не було, тому подальший аналіз стосуватиметься інді-ігор, розроблених невеликими командми. Він проводиться з метою щоб визначити певні ідеї, що уже перевірені іншими людьми та підходи, що є стандартом для жанру.

Перш за все, було опрацьовано реалізацію таких ігор як Overcooked[5] та PlateUp[6], розміщених в Steam. Їх візуальна складова є досить лаконічною, при цьому, ефективно використовує технічні ресурси пристрою гравця, не використовуючи надміру складних ефектів, що могли б призвести до нестабільної роботи, підвисань, тощо.

На зображені 1.2 передано як виглядає рівень гри Overcooked. На зображенні можна побачити мультиплікаційну стилізацію текстур, використання низькополігональних моделей, відсутність надмірної деталізації, що створює приємну атмосферу і не перевантажує ані гравця, ані пристрій, що обробляє графіку.

Рис 1.2. Візуальний стиль гри Overcooked
Джерело: сторінка гри в цифровому мазазині Steam[5]

В грі PlateUp застосовано схожий підхід, проте камера ще більш віддалена від персонажів щоб сфокусувати погляд користувача на діях, а не на предметах. Це дозволяє не обтяжувати сприйняття, навіть при достатньо складних механіках гри.

Обидві ці гри мають досить прості правила, що забезпечує низький поріг входу для нових гравців, проте для майстерного проходження, необхідно провести в грі багато часу, розбираючись з їх механіками, що створює ідеальний баланс як і для новачків, так і для любителів аркадних ігор.

Додатковим джерелом напруги є те, що гравцю необхідно виконувати кілька завдань одночасно щоб отримати максимальну нагороду, що робить ігровий процес насиченішим та приносить більше задоволення.

У вищезазначених іграх та аналогічно, як і в грі The Escapists 2[7], керування персонажем гравця виконується шляхом управлінням джойстиком(якщо підключений зовнішній контроллер) або завдяки 4 клавішам на клавіатурі. Взаємодія з предметами в середині гри можлива коли вони знаходяться прямо перед персонажем гравця, шляхом виділення. Це значно спрощує розуміння ігрової ситуації та прискорює прийняття рішень, роблячи світ гри більш зрозумілим для гравця. Геймлпей цієї гри зображений рис. 1.3.

Рис 1.3. Скріншот з гри The Escapists 2
Джерело: сторінка гри в цифровому мазазині Steam[7]

У всіх іграх, що були проаналізовані, використовуються короткі та плавні анімації елементів інтерфейсу щоб зробити гру більш живою та підкреслити зміни у

відображені ігрових показників. Це покращує інформованість гравця про поточний стан світу гри та, в цілому, роблять взаємодії з ним приємнішими.

Також, вищезазначені ігри використовують різноманітні індикатори та шкали заповнення прогресу щоб показати наскільки виконана ігрова дія у відсотках, проте не показують це значення текстом, що, знову ж, дозволяє гравцю залишатись в контексті, але не робить кількість показників надто великою щоб запобігти перетворенню їх на відволікаючий фактор.

На екрані, під час проходження рівнів/виконання задач завжди знаходиться інтерфейс, котрий показує базову інформацію про гравця, його навички, предмети(якщо актуально).

Загальною рисою перелічених ігор, є те, що для максимального утримання, гра розбивається на етапи, що відрізняються своєю динамікою. Кожна з них має напружений геймплей основної частини під час виконання замовлень в PlateUp та Overcooked чи пошуку способів вибратись з в’язниці як в The Escapist, проте обов’язковим фактором спокійний фрагмент як підготовка до робочого дня або час для сну, коли персонаж не має значних обов’язків, гравець не обмежений в часі і може перепочити, подумати про свої наступні дії.

Схожий підхід використовують не лише ігри, а й фільми, музика, книги та інші форми мистецтва. Це дозволяє людям заглиблюватись в них більше та не мати перенасичення емоціями.

Не менш важливу роль грає звукове оформлення гри. Звуки є короткими і не надто голосними, проте чітко підкреслюють виконання дії, можуть сигналізувати про завершення прогресу, до напруження від того, що гравець може не встигнути якщо не поспішить виконати певну дію; дають додаткове приємне відчуття від натискань на плавно анімовані кнопки.

Тим не менше, більшість гравців, що грають на мобільних пристроях, відключають звук та музику в грі через коротктривалість їх сесій. Їм просто нема сенсу підключати мобільну гарнітуру коли весь час на гру обмежений 5-10 хвилинами, що будуть проведені в якийсь черзі чи в громадському транспорті, тому

у випадку мобільних ігор, набагато важливішим є вдале використання графічного оформлення і цікавий геймплей, котрий є основою для будь-якої гри.

Кожна гра має систему локалізації для того, щоб показувати тексти зрозумілою мовою для кожної з цільових регіональних аудиторій. При цьому, для здешевлення проекту, відсутня повноцінна озвучка щоб не створювати необхідність її дороговартісного перекладу.

Кожна з перелічених ігор має внутрішню валюту для нагородження гравця за коректні та вчасні дії. Для всіх вищезазначених ігор такою валютою є монетки, що стало стандартом індустрій. Їх постійний показ можна побачити на рис. 1.4. у верхньому лівому кутку.

Рис 1.4. Скріншот з гри PlateUp

Джерело: сторінка гри в цифровому мазазині Steam[6]

Також, на цьому рис. 1.4 можна побачити, що, так як за правилами гри, постійно приходять нові і нові відвідувачі, котрих має обслуговувати гравець, готуючи їм їжу; вони шикуються в чергу, дозволяючи гравцеві оцінити скількох людей ще потрібно нагодувати для успішного закінчення рівня.

Це дозволяє дати гравцю відчуття прогресу в середині рівня на малому масштабі. Також, на це працює відображення номера дня, скільки вже гравець пропрацював над своїм кафе.

Натомість загальна прогресія гри може бути реалізована шляхом створення певного довгострокового проекту в масштабах гри. Прикладом такого проекту є розбудова власного ресторана в грі PlateUp чи створення складних інструментів з великої кількості проміжних крафтів в грі The Escapists 2.

Коли гравець отримує значиму нагороду за велику кількість коректно виконаних дій, він може більше насолодитись результатом, якщо той допомагає легше грати в гру. В PlateUp таким результатом буде більша автоматизація процесів приготування страв, що дозволяє створювати легше та/або більше порцій їжі, збільшуючи кількість отриманих монет.

1.3 Постановка задачі

Отже, було вирішено розробити гру про Острозьку академію для демонстрації освітніх можливостей університету, залучення нових абітурієнтів, шляхом демонстрації локації навчального закладу та тих навичок і вмінь, котрими володіють її студенти, що можна буде проаналізувати дивлячись на технічний стан готової гри.

Було визначено більш ефективним розробити саме мобільну гру, так як більшість людей завжди мають з собою свій смартфон і зручніше всього буде запустити її саме на ньому, дозволяючи ознайомитись з сетінгом національного університету в будь-якому місці, де б не було б людина, якщо в неї є вільні кілька хвилин. До того ж, розробка мобільної гри є дешевшою як і в плані вартості публікації, так і в плані виробництва самої гри, враховуючи, що гравці мають зовсім різні очікування в плані масштабності мобільних та комп’ютерних/консольних ігор.

Найбільш вдалим вибором жанру було визначено аркадну рольову гру, так як це легкозрозумілий концепт, що дозволить більшості людям ознайомитись з грою без додаткових складнощів; а мультиплікаційна стилістика зробить гру прийнятною для будь-якого віку цільового користувача і підкреслить наскільки приємно навчатись в в цьому закладі.

Так як було прийнято рішення розробити гру про Острозьку академії, її основною локацією необхідно було зробити територію університету. Після розгляду різноманітних фото і відео з університету, самостійного дослідження локацій офлайн

було прийнято рішення розгорнути події гри в приміщенні бібліотеки Острозької академії, яка має вкрай незвичну форму, що легко запам’ятовується після першого відвідування.

Ця кругла будівля із зеленим дахом зображена на рис. 1.5. і в такому ж вигляді, вона повинна бути перенесена у віртуальний світ.

Рис 1.5. Фотографія бібліотеки Острозької академії
Джерело: Вікіпедія

На це рішення значно вплинуло те, що бібілотека асоціюється з книгами та знаннями, що є основою навчального процесу в університеті, який було обрано продемонструвати в грі в гейміфікований формі.

За задумом, гравець повинен взяти на себе роль бібліотекаря, котрий буде інформаційно забезпечувати навчальний та науково-дослідницький процес, а саме видавати книги викладачам та студентам за їх запитом, що постійно відбувається і в реальному навчальному процесі. За ці дії він буде нагороджуватись монетами.

Також, одним з його завдань буде наповнювати бібліотеку все новими книгами, що будуть регулярно підвозитись у віртуально бібліотеку так само, як і постійно це робиться з реальною.

На додаток, щоб продемонструвати постійні процеси цифровізації, що тривають в університеті, гра мусить надавати можливість отримувати додаткові монети за сканування книг, створюючи цифрову бібліотеку.

Щоб зробити гру більш цікавою, головний герой зможе читати книги, отримуючи нові навички з головних наукових дисциплін:

- фізики;
- інженерної справи;
- біології;
- програмування;
- математики;
- хімії.

Далі, отриманні знання можна буде застосувати для виконання глобальних практичних цілей, що продемонструють наскільки отриманні знання є актуальними та корисними. Такими глобальними цілями було визначено зробити:

- запуск ракети в космос;
- виготовлення вакцини для зцілення зомбі і повернення його в людський облік;
- розробки гри LibTake прямо всередині цієї ж гри.

Для отримання такого ж якісного балансу по насиченості та спокою в грі, що вкрай важливо для довготривалого утримання користувача, як і в проаналізованих іграх; важливим є також реалізувати спокійну та напружену фазу геймплея, дозволяючи гравцю відпочивати від виснажливих етапів та не втомлюватись від гри надто швидко. Для цього ігрові доби було вирішено розділити на дві фази:

- ранок;
- робочий день.

Під час ігрового ранку гравець зможе спланувати наступні дії та спокійно підготуватись до робочого дня; а, під час останнього, виконувати багато основних напружених дій.

До того ж, в грі необхідно реалізувати якісні плавні анімації та додати звуковий супровід всіх дій.

Висновки

Було здійснено ґрунтовний аналіз предметного середовища функціонування комп’ютерних ігор. Встановлено, що індустрія відеоігор є однією з найдинамічніших та найприбутковіших галузей сучасної цифрової економіки, яка невпинно продовжує зростати; відеоігри мають не лише розважальний характер, а й можуть мати цінність для освітніх, дослідницьких, соціальних цілях, що дозволяє залучати нові аудиторії й формувати нові способи сприйняття інформації.

Аналіз сучасних інді-ігор таких як Overcooked, PlateUp та The Escapists 2 дозволив виявити ключові особливості сучасного створення ігрового програмного забезпечення, з урахуванням необхідності збереження балансу між доступністю, ігровою глибиною, емоційною напругою та спокоєм гравця та ефективністю використання його пристроя.

На основі вищезазначеного, було сформульовано задачу створення мобільної аркадної гри, події якої будуть розгортатись у віртуальному просторі Острозької академії, гравець виконуватиме роль бібліотекаря, видаватиме книги студентам та викладачам, отримувати за це винагороди. Також, в ігровий процес буде інтегровано освітньо-дослідні елементи - персонаж буде здобувати нові знання та використовувати їх для реалізації практичних проектів, котрі і будуть головною ціллю гри.

Таким чином, визначені концептуальні засади проекту, обґрунтовано вибір жанру, досліджено як ігрові елементи мають впливати на досвід гравця та сформовано задачу розробки.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної області

Предметною областю кваліфікаційної роботи є мобільний ігровий додаток "Libtake", що моделює функціонування бібліотеки національного університету "Острозька академія". Основний об'єкт дослідження - це віртуальне середовище бібліотеки, в якому відбуваються процеси обслуговування користувачів, управління книжковим фондом та реалізації освітньо-наукових проектів.

Основні об'єкти предметної області:

Персонаж гравця - представляє бібліотекаря з набором навичок та характеристик:

- Вхідні дані: дії користувача (переміщення, взаємодія)
- Вихідні дані: зміна балансу монет, навичок, статусу виконання цілей

Книга - базова одиниця інформаційного ресурсу:

- Вхідні дані: назва, категорія, статус (фізична/цифрова, прочитана/не прочитана)
- Вихідні дані: зміна статусу, нарахування навичок гравцю

Відвідувачі - студенти та викладачі:

- Вхідні дані: тип (студент/викладач), запит на книгу
- Вихідні дані: показники задоволення, нагорода гравцю

Системи навичок - 6 наукових дисциплін:

- Вхідні дані: прочитані книги відповідної категорії
- Вихідні дані: рівень навички, доступність крафтів

Глобальні цілі - проекти для виконання:

- Вхідні дані: виконані етапи крафту, наявні ресурси
- Вихідні дані: прогрес виконання, завершення гри

Ці об’єкти взаємодіють між собою через ігрові механіки, моделюючи послідовність дій у межах одного дня(планування або робота) та на протязі однієї сесії.

В основі функціонування гри лежить інформаційна модель, що визначає потоки даних між об’єктами. Гравець виконує роль роль центрального вузла цієї системи, що приймає вхідні запити від читачів та плану глобальної цілі, реагує на них через маніпуляції з книгами та крафатми.

Вхідні дані для системи:

- Дії користувача через інтерфейс
- Запити відвідувачів на книги
- Таймери для обмеження дій
- Збережені дані гри

Вихідні дані системи:

- Візуальне відображення стану гри
- Оновлені характеристики гравця
- Стан виконання завдань та цілей
- Анімації та звуковий супровід

Діаграма використання наведена на рис. 2.1. Відповідно до неї, перед початком гри, гравець обирає глобальну ціль в головному меню. Під час фази планування він забирає книги з вантажівки та розміщує їх на столі так, щоб було зручно видавати відвідувачам.

У фазі робочого дня, паралельно гравець обслуговує читачів шляхом пошуку і видачею книг та читає книги самостійно для підвищення навиків, що дозволяє йому виконувати крафти для глобального проекту.

У випадку якщо гравець втрачає всі життя, відбувається програш та показується відповідна катсцена.

Якщо глобальна ціль виконана - гра вважається виграною та показується відповідна катсцена.

Якщо глобальна ціль не виконана, наступає новий день, що повторює цей цикл. Вищеописані дії є ключовими для гри та забезпечують ігровий процес різноманіттям і логічним завершенням.

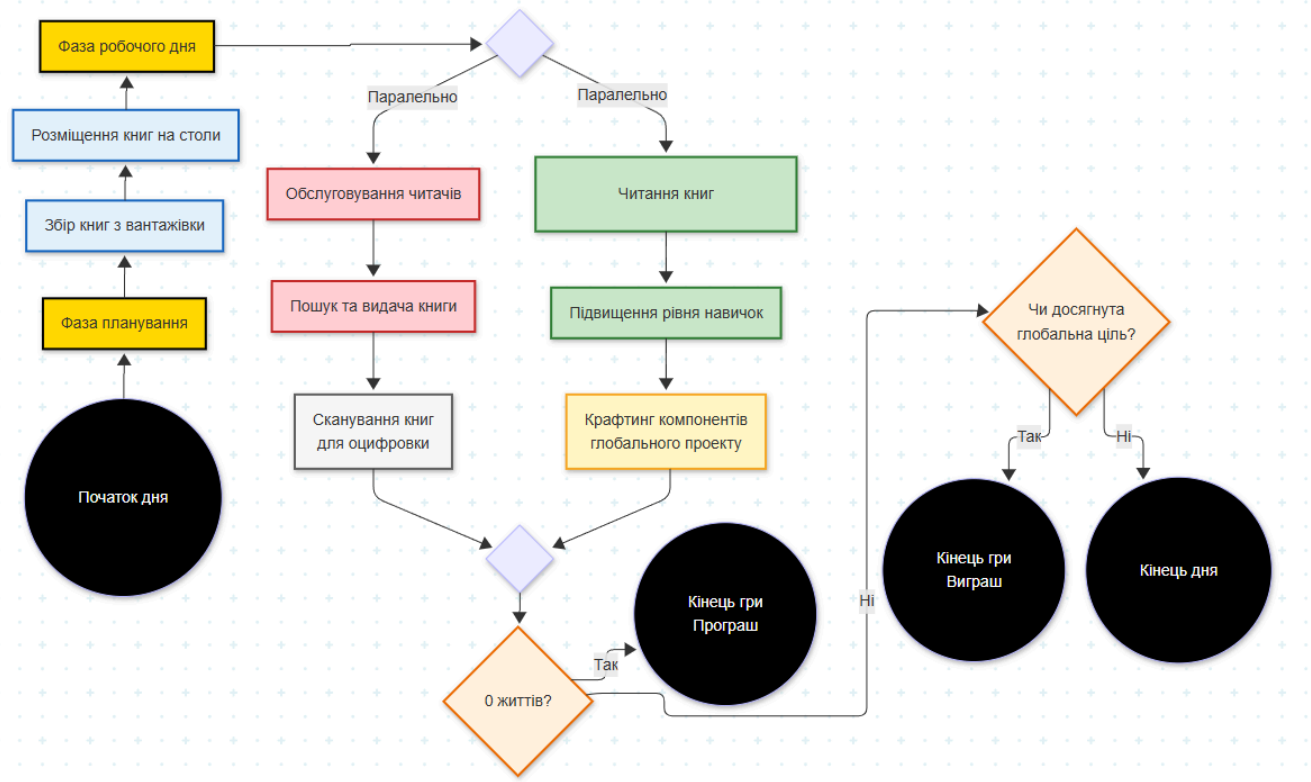

Рис 2.1. Діаграма використання гри Libtake
Джерело: складено автором.

Гравець самостійно організовує простір: розставляє книги, розподіляє час, обирає пріоритети (обслуговування читачів чи розвиток навичок), що додає елементів стратегічного планування. Механіка гри реалізує концепцію "обмежених ресурсів" — часу, життів, простору.

Вхідні та вихідні дані відіграють вкрай вагому роль у геймплейному циклі гри. Нижче узагальнено ключові дані, що обробляються під час геймплею:

Книга

- Вхідні дані: назва, категорія, статус(оцифрована/прочитана);
- Вихідні дані: отримання навички, зміна статусу книги скануванням/читанням.

Читач

- Вхідні дані: тип (студент / викладач), запит на конкретну книгу;
- Вихідні дані: задоволення / незадоволення, нагорода, втрата життя гравцем.

Гравець (бібліотекар)

- Вхідні дані: дії користувача (переміщення, вибір дії, натискання);
- Вихідні дані: зміна балансу монет, зміна рівня навичок, прогрес у глобальній цілі.

Таймер

- Вхідні дані: ініціація дії (обслуговування читача, читання книги, крафт);
- Вихідні дані: результат дії (успішно / провалено), оновлення стану гри.

Навичка

- Вхідні дані: категорія книги, кількість прочитаних книг відповідної тематики;
- Вихідні дані: відкриття можливості для крафту, вплив на швидкість / ефективність дій.

Крафт

- Вхідні дані: перелік ресурсів, рівень навички, вибрана мета;
- Вихідні дані: створення артефакту (компонент ракети, ліки тощо), оновлення цілі.

Глобальна мета

- Вхідні дані: виконані етапи крафту, наявні навички та ресурси;
- Вихідні дані: фінальна катсцена, завершення гри, демонстрація успіху.

Система гри LibTake побудована з урахуванням низки обмежень, які спрямовані на збереження балансу складності, зрозумілості й продуктивності:

- Один читач активний одночасно.
- Гравець не може носити більше однієї книги.
- Час видачі книги обмежений таймером.
- Відвідувачі ніколи не беруть цифрові копії(беруть за лаштунками через інтернет, без участі гравця).

- Книги мають кольорове маркування — тип навички.
- Гравець може читати книги лише на спеціальному столі для читання.
- Усі дії (видача, читання, крафт) потребують натискання однієї кнопки.
- Прогрес зберігається лише локально, після кожного ігрового дня.
- Усі об’єкти гри ведуть себе спрощено, так як в мультиплікаціях, відсутня повноцінна фізична симуляція.

Завдяки цим обмеженням, гра стане більш привабливою для гравця і з точки зору ігрових правил; і швидкості й плавності роботи гри; а також зменшується поріг входу для нових користувачів, що спрощує залученість нової аудиторії для гри.

2.2 Проектування системи

Вибір компонентно-орієнтованого підходу для розробки мобільної гри LibTake базується на фундаментальних принципах сучасної розробки програмного забезпечення та специфічних вимогах ігрового проекту. Компонентно-орієнтований підхід є альтернативою традиційній об'єктно-орієнтованій ієрархії наслідування, що особливо актуально для розробки ігор. Цей підхід базується на принципі "композиція над спадкуванням", який визнаний одним з основних принципів сучасного проектування програмних систем. Відповідно до принципів SOLID, зокрема Принципу Єдиної Відповідальності та Принципу Відкритості/Закритості, компонентний підхід дозволяє створювати більш модульні, розширювані та підтримувані системи.

Традиційне об'єктно-орієнтоване програмування через наслідування має ряд фундаментальних проблем, які особливо проявляються в розробці ігор. "Diamond problem" виникає при множинному наслідуванні, коли клас успадковується від кількох базових класів через проміжні класи, створюючи неоднозначність. Жорсткі ієрархії класів стають складними для модифікації по мірі розвитку проекту, а глибокі ієрархії наслідування важко розуміти і підтримувати. В контексті гри LibTake, де об'єкти мають різноманітні комбінації поведінок (наприклад, книга може бути одночасно прочитаючитаною, оцифрованою та виданою), традиційний підхід призвів би до складних та негнучких ієрархій класів.

Компонентний підхід розв'язує ці проблеми, надаючи гнучкість у комбінуванні поведінки об'єктів. Замість створення ієрархій класів, функціональність розподіляється по незалежних компонентах, які можна динамічно додавати або видаляти. Це забезпечує високий рівень повторного використання коду та спрощує управління складністю системи. У контексті LibTake, кожен ігровий об'єкт складається з набору компонентів, що визначають його поведінку: компонент взаємодії, компонент зберігання книг, компонент анімації тощо.

Unity Engine використовує компонентну архітектуру як основний парадигмальний підхід. Кожен GameObject в Unity є контейнером для компонентів, а не класом з вбудованою функціональністю. Це знайшло відображення в Entity-Component-System архітектурі рушія, де дані відокремлені від логіки обробки. Монобех'євіор компоненти слугують базовим класом для реалізації логіки гри, забезпечуючи стандартизований життєвий цикл об'єктів та можливість незалежної розробки і тестування компонентів.

В контексті реалізації LibTake, компонентний підхід демонструє практичні переваги через модульність характеристик ігрових об'єктів. Наприклад, стіл для читання складається з компонентів BookStorage (зберігання книг), ReadingProgress (прогрес читання) та Interactable (взаємодія з гравцем). Кожен з цих компонентів може бути створений, протестований та оптимізований незалежно. При необхідності додавання нової функціональності, як-от механіка чарівних книг, створюється новий компонент EnchantedBookComponent, який додається до існуючих об'єктів без модифікації вихідного коду.

Розширюваність функціональності особливо важлива для ігрових проектів, де вимоги часто змінюються під час розробки. Компонентний підхід дозволяє додавати нові механіки через створення нових компонентів без зміни існуючого коду. Це відповідає принципу відкритості/закритості та зменшує ризики внесення помилок при модифікації системи. При додаванні нових глобальних цілей або типів книг у LibTake, розробники можуть створювати нові компоненти паралельно з розвитком основної функціональності.

Підтримуваність коду значно покращується через дотримання принципу єдиної відповідальності, коли кожен компонент має чітко визначений функціонал. За рекомендаціями Роберта Мартіна щодо "чистого коду", середній розмір файлу компонента в LibTake становить 38 рядків, що забезпечує легкість розуміння і модифікації коду. Відсутність "god-objects" спрощує навігацію по кодовій базі і знижує когнітивне навантаження на розробників.

Таким чином, компонентно-орієнтований підхід оптимально відповідає вимогам проекту LibTake, забезпечуючи гнучкість архітектури, високу продуктивність, простоту розширення та ефективну підтримку коду. Цей вибір підтверджується як теоретичними основами сучасної розробки ПЗ, так і практичними результатами впровадження в ігровому проекті.

Центральною частиною архітектури мобільної гри LibTake є стейт-машина (**GameStateMachine**), яка реалізує паттерн State для управління життєвим циклом гри та контролю складних переходів між різними ігровими станами. Стейт-машина виступає як координуюча сутність, що забезпечує чіткий контроль над поточним станом гри та керує переходами між станами відповідно до логіки ігрового процесу.

Архітектура стейт-машини базується на інтерфейсному підході з чіткою ієрархією абстракцій. Основою виступає інтерфейс **IExitableState**, який визначає базовий контракт для всіх станів із методом Exit() для завершення роботи стану. На його основі побудовані два спеціалізованих інтерфейси: **IGameState** для простих станів без параметрів (Start() та Exit()) та **IPayloadedState<TPayload>** для станів, що потребують вхідних даних при ініціалізації.

Реалізація стейт-машини включає 11 основних станів, кожен з яких відповідає за певний аспект життєвого циклу гри. **BootstrapGameState** є початковим станом, що ініціалізує всі основні сервіси та налаштовує системи збереження прогресу. Цей стан демонструє принцип інверсії залежностей, отримуючи через конструктор усі необхідні сервіси через Zenject DI framework[8].

WarmupGameState виконує завантаження статичних даних, ініціалізацію системи локалізації та підготовку до переходу в меню. Його асинхронна природа

забезпечується використанням UniTask, що дозволяє ефективно обробляти операції завантаження без блокування основного циклу гри.

MenuGameState представляє мінімалістичний стан для головного меню, демонструючи принцип єдиної відповідальності, коли стан не містить непотрібної логіки. **LoadProgressState** реалізує завантаження збереженого прогресу гри або створення нового, використовуючи параметризований підхід для різних сценаріїв запуску.

LoadLevelState є найбільш складним станом, відповідальним за завантаження рівня гри. Він ініціалізує весь ігровий світ: книжкові полиці, столи для читання, сканери, NPC та елементи інтерфейсу. Стан демонструє координацію різних фабрик і сервісів для створення повноцінного ігрового середовища.

MorningGameState керує ранковою фазою ігрового дня, обмежуючи певні дії гравця та організовуючи доставку книг. Стан зберігає прогрес гри та координує анімацію вантажівки з механікою взяття книг. **DayGameState** управляє активною фазою гри, запускаючи паралельні процеси обслуговування клієнтів та моніторингу життів гравця через асинхронну логіку.

FinishGlobalGoalState координує фінальну кат-сцену при досягненні глобальної цілі, вимикаючи ігрові механіки та запускаючи візуальні ефекти. **GameOverGameState** обробляє програш гри, переходячи до відповідної сцени, а RestartGameState забезпечує повне очищення даних для нової гри.

Кожен стан реалізує чіткий контракт входу та виходу, забезпечуючи коректну очистку ресурсів та підготовку до наступного стану. Стани використовують dependency injection для отримання необхідних сервісів, демонструючи слабку зв'язність компонентів. Асинхронна природа переходів реалізована через UniTask, що забезпечує плавні переходи без втрати відгуку інтерфейсу.

Стейт-машина координує взаємодію між різними підсистемами гри: сервісами збереження, локалізації, завантаження контенту, UI та ігровою логікою. Це забезпечує централізоване управління життєвим циклом додатку та чітке розділення відповідальностей між компонентами системи.

Також, стани забезпечують ізоляцію ігрових частин одна від одної та дозволяють обмежити виконання коду, що від носиться до різних частин гри цими ж частинами. UML-діаграма, що представляє переходи між цими станами зображена на рис. 2.2.

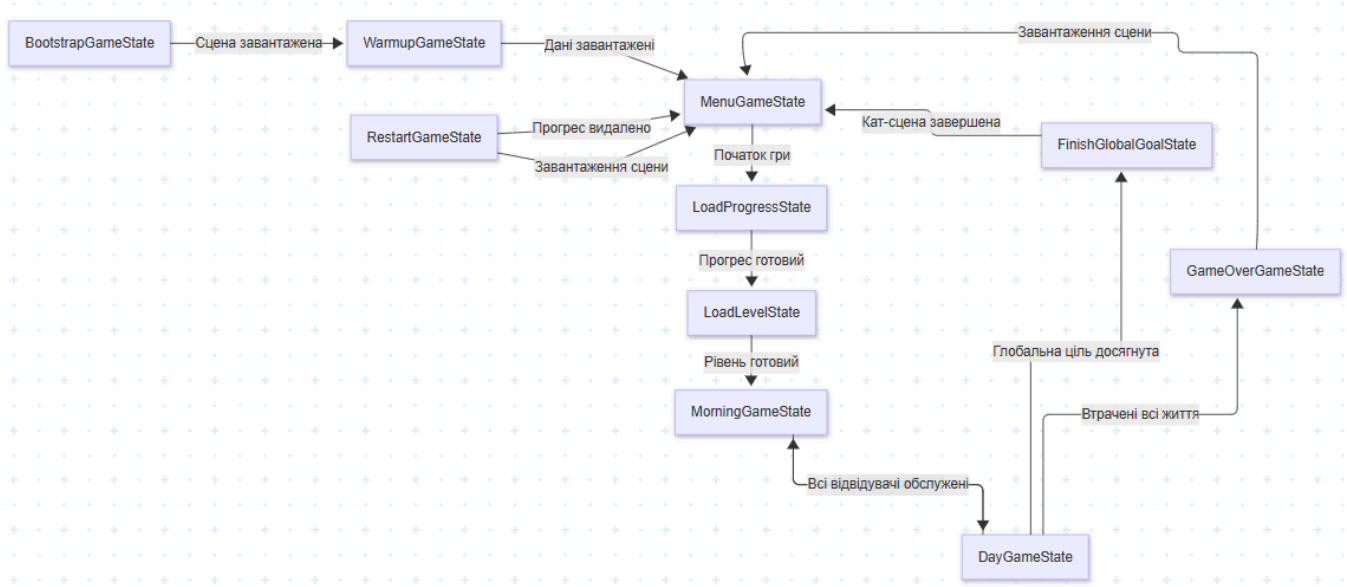

Рис 2.2. UML-діаграма GameStateMachine
Джерело: складено автором.

Фундаментальною частиною архітектури LibTake є використання підходу до управління залежностями через Dependency Injection (DI). Цей підхід дозволяє створювати слабко зв'язані компоненти системи, які легко тестувати, розширювати та підтримувати. Використання DI в архітектурі обумовлене кількома ключовими факторами, що важливі для розробки мобільної гри.

По-перше, DI забезпечує реалізацію принципу інверсії залежностей (Dependency Inversion Principle), одного з принципів SOLID. Це дозволяє високорівневим модулям не залежати від низькорівневих, а обидвом залежати від абстракцій. В контексті LibTake, це означає, що основна ігрова логіка не залежить від конкретної реалізації збереження даних, системи вводу чи механік взаємодії – всі ці залежності ін'єктуються через інтерфейси.

По-друге, використання DI значно спрощує процес тестування компонентів. Кожен клас отримує свої залежності ззовні, що дозволяє легко замінювати реальні

імплементації на моки або заглушки під час модульних тестів. Це особливо важливо для ігрового проекту, де правильна робота механік критична для якості продукту.

По-третє, DI забезпечує центральне управління життєвим циклом об'єктів та їх одиничностей (singletons). Це дозволяє гарантувати, що критичні сервіси гри (як системи збереження даних, управління локалізацією, фізичний движок) існують у єдиному екземплярі протягом усього життєвого циклу додатку.

В архітектурі LibTake DI реалізовано через контейнерний підхід, де всі залежності реєструються на етапі ініціалізації програми та автоматично надаються компонентам, які їх потребують. Це досягається через механізм інжекції через конструктор або спеціальні методи, що особливо важливо для компонентів Unity, які не підтримують конструктори в класичному розумінні.

Концепція сервісів є центральною частиною архітектури LibTake, забезпечуючи єдиний та узгоджений інтерфейс для взаємодії різних частин системи. Сервіси в даному контексті – це одиночні об'єкти (singletons), які надають функціональність для основної ігрової логіки та забезпечують абстракцію над низькорівневими операціями.

Кожен сервіс реалізує універсальний контракт взаємодії через інтерфейси, що дозволяє замінювати імплементації без зміни коду, який ці сервіси використовує. Ця архітектурна модель забезпечує чітке розділення відповідальностей та дотримання принципу єдиної відповідальності (Single Responsibility Principle).

Переваги сервісного підходу наступні:

1. Централізація логіки: Кожен сервіс відповідає за конкретний аспект гри, дозволяючи централізовано управляти складною логікою.
2. Повторне використання: Сервіси можуть бути використані будь-якою частиною гри, що потребує їх функціональність.
3. Тестованість: Сервіси легко тестуються ізольовано завдяки їх інтерфейсній природі.
4. Розширюваність: Нові сервіси можуть бути додані без модифікації існуючого коду.
5. Однозначність: Кожен сервіс має чітко визначену область відповідальності.

Такий підхід забезпечує чистоту архітектури та дотримання принципів SOLID, роблячи систему більш контрольованою та придатною для довгострокової підтримки та розширення.

В архітектурі LibTake, обробка статичної інформації відіграє критичну роль у забезпеченні ефективного управління даними та централізованого доступу до конфігураційних параметрів гри. Система динамічного завантаження статичних даних реалізована через спеціалізований сервіс StaticDataService, який дотримується принципів SOLID та забезпечує структурований доступ до всіх конфігураційних ресурсів.

В концепції роботи зі статичними даними реалізовано принцип єдиної відповідальності, де кожен тип статичної інформації представлений окремим ScriptableObject класом. Цей підхід дозволяє створювати, модифікувати та зберігати конфігурації незалежно від основного коду гри. Методичні рекомендації визначають структуру кваліфікаційної роботи та вимоги до оформлення результатів дослідження, що робить цей підхід зручним для документації та подальшого аналізу.

Система використовує ресурсні шляхи для організації статичних даних, завдяки чому досягається модульність та масштабованість. Кожна категорія даних зберігається у своїй папці "Resources" з відповідною структурою: книги, рівні, інтерактивні об'єкти, глобальні цілі, UI елементи, персонажі та налаштування балансу. Це забезпечує чітку організацію файлової системи проекту та спрощує пошук і оновлення конкретних ресурсів.

Особливістю обробки статичних даних є їх кешування в пам'яті за допомогою словників та списків. Інформаційна модель визначає потоки даних між об'єктами у системі, що дозволяє швидко отримувати доступ до об'єктів без постійних звернень до файлової системи. Це критично для мобільної платформи, де оптимізація роботи з пам'яттю та швидкість доступу до даних є ключовими факторами.

Статична інформація в LibTake представлена різними типами об'єктів: книги з унікальними ідентифікаторами та локалізованими назвами, інтерактивні об'єкти з покроковыми інструкціями взаємодії, глобальні цілі зі структурованими етапами

виконання та налаштуваннями камери. Кожен тип статичних даних має власний формат та методи доступу, визначені через інтерфейси. Це забезпечує типобезпеку та можливість валідації даних на етапі розробки.

Архітектурно важливим аспектом є те, що StaticDataService виступає посередником між низькорівневим завантаженням ресурсів через Resources.Load[9] та високорівневою логікою гри. Використання dependency injection для цього сервісу забезпечує слабку зв'язку компонентів системи та спрощує тестування. Сервіс реалізує принцип "завантажити один раз, використовувати повсюди", що критично для оптимізації пам'яті на мобільних пристроях.

Система підтримує розширюваність через добре структурований API, який дозволяє легко додавати нові типи статичних даних без модифікації існуючого коду. Це особливо важливо для подальшого розвитку гри та додавання нових механік. Обробка статичної інформації в LibTake демонструє застосування сучасних принципів проектування для забезпечення надійної, масштабованої та ефективної архітектури мобільної гри.

2.3 Математичне та алгоритмічне забезпечення

Фундаментальним компонентом математичного апарату є система обчислення прогресу виконання дій, реалізована через клас Progress. Вона перевикористовується в численних механіках гри, тому написана універсальною. Ця система представляє собою модель інкрементального наповнення, де приріст прогресу визначається як відношення часового дельта-інтервалу до загального часу виконання операції: Time.deltaTime / _timeToFinish. Такий підхід забезпечує плавну та рівномірну візуалізацію прогресу незалежно від частоти кадрів гри. Оновлення стану прогресу здійснюється асинхронно з використанням технології async/await, представленою бібліотекою UniTask(для повноцінної інтеграції з середовищем ігрового рушія Unity)[10], що дозволяє уникнути блокування основного потоку обчислень. Нормалізація значень прогресу у діапазоні [0, 1] спрощує подальшу обробку та візуалізацію інформації.

Економічна модель гри базується на системі винагород, що використовує нелінійну залежність між продуктивністю гравця та матеріальною винагородою. Математична функція винагороди визначає розмір компенсації на основі залишкового часу виконання завдання, створюючи градієнт стимулів від максимального значення у 4 монети при залишку часу понад 75% до мінімального значення у 1 монету при критично низькому залишку. Така економічна модель стимулює ефективність виконання завдань та створює додаткову напругу в ігровому процесі.

Алгоритм динамічного балансування бібліотечного фонду реалізований через функцію доставки книг, що враховує поточний стан бібліотеки та прогресію гри. Центральним елементом цього алгоритму є формула оптимізації кількості доставки: max(shouldBeInLibrary - inLibrary, 1), що забезпечує адаптивне поповнення колекції відповідно до поточних потреб. Метод визначення цільової кількості книг використовує підхід поступового масштабування складності, враховуючи номер поточного дня гри, проте все ще не дозволяє бібліотеці переповнитись, не залишивши гравцю можливості покласти всі книги на столи, заблокувавши подальше проходження.

Система просторового виявлення інтерактивних об'єктів використовує модифікований алгоритм рейкастингу з множинним скануванням. Замість традиційного односпрямованого променя, система проектує векторний набір рейкастів, що створює конус виявлення навколо позиції гравця. Цей підхід забезпечує більш інтуїтивну взаємодію користувача з ігровим середовищем, особливо за умов тактильного управління на мобільних пристроях. Алгоритм виявлення оптимізовано через застосування спеціалізованих шарів для інтерактивних об'єктів, мінімізуючи кількість обчислювальних операцій.

Математична модель руху гравця базується на комбінації векторної трансформації та принципів фізики інерції. Базовий вектор руху отримується через трансформацію вводу з локальних координат камери в світові координати за допомогою формули _camera.transform.TransformDirection(input), з подальшою нормалізацією горизонтальної складової через обнулення компоненти у.

Горизонтальна швидкість обчислюється як добуток базової швидкості та напрямного вектора, в той час як загальна швидкість є векторною сумою горизонтальної швидкості та гравітації. Алгоритм обробки інерції використовує лінійну інтерполяцію Vector3.Lerp між поточною швидкістю та нульовим вектором зі швидкістю затухання _inertiaDecayRate, що забезпечує природне уповільнення після припинення введення. Ротація персонажа реалізується через сферичну лінійну інтерполяцію Quaternion.Slerp, створюючи плавний поворот до цільової орієнтації.

Система контролю камери використовує комбінацію декількох математичних моделей для забезпечення динамічного та плавного спостереження. Цільова позиція камери визначається як target.position + _offset, де офсет може варіюватись між низькою та високою позицією камери залежно від налаштувань гравця. Плавні переходи реалізовані через DOTween[11] з використанням кривих анімації Ease, що забезпечує природний рух камери. Система спостереження реалізує безперервне оновлення напрямку погляду через transform.LookAt з обмеженням осей, де стартовий перехід анімується для плавного початку спостереження. Важливою особливістю є корутинний механізм оновлення цілі треку, що періодично перераховує кінцеву позицію анімації, дозволяючи камері слідкувати за рухомими об'єктами з динамічною корекцією траєкторії.

Архітектура реалізації глобальних цілей представляє собою систему верифікації стану, що включає каскадний аналіз умов виконання. Алгоритм перевірки можливості виконання етапу оцінює економічний стан гравця (оплата етапу) та його компетенції через порівняння поточного рівня навичок з вимогами етапу. Система компетенцій використовує мультидименсіональний підхід, де кожна навичка представлена як окремий вектор розвитку, а загальна придатність визначається перевіркою всіх компонентів вектора вимог.

Архітектура системи персистентності даних базується на концепції автономної серіалізації, де кожен компонент гри самостійно управляє власним станом через уніфікований інтерфейс ISavedProgress. Структура GameProgress представляє собой ієрархічну модель даних, що агрегує стани всіх важливих компонентів гри. Вибір формату JSON для серіалізації обумовлений його гнучкістю та можливістю ручного

редагування для налагодження та тестування, а також ефективністю парсингу на мобільних платформах. Асинхронне завантаження та синхронізація стану забезпечують швидкий запуск гри та мінімальний час очікування для користувача.

Стохастичні процеси в грі реалізовані через спеціалізований RandomService, що забезпечує псевдовипадковий вибір елементів з дискретних множин. Алгоритм використовується для диверсифікації ігрового досвіду через рандомізацію процесів доставки та видачі книг. Метод GetInRange реалізує рівномірний розподіл ймовірностей у заданому діапазоні.

Загальна оптимізація алгоритмів спрямована на ефективне використання обмежених ресурсів мобільних пристроїв. Застосування асинхронної моделі програмування через UniTask дозволяє розподіляти обчислювальне навантаження між кадрами, уникаючи блокування основного потоку виконання. Це особливо критично для підтримки високої частоти кадрів на пристроях з обмеженою обчислювальною потужністю. Крім того, реалізовані алгоритми використовують ефективні структури даних та мінімізують кількість динамічних виділень пам'яті, що важливо для стабільної роботи на платформі Android.

Також, наведені вище математичні формули розроблені з урахуванням необхідності зробити процес гри більш захопливим для гравця та додати можливість реіграбельності щоб утримувати гравця довше та дозволити продовжувати гру навіть після досягнення всіх глобальних цілей передбачених грою.

Висновки

У другому розділі проведено комплексний аналіз інформаційного та математичного забезпечення мобільної гри LibTake. Встановлено, що предметна область представляє собою складну систему взаємодіючих компонентів: персонажа гравця, книг, відвідувачів, системи навичок і глобальних цілей. Визначено основні потоки даних та обмеження системи, що забезпечують збалансований ігровий процес.

Розроблена архітектура гри базується на компонентно-орієнтованому підході, що дозволяє ефективно управляти складністю проекту через діаграми використання та стейт-машини. Застосування принципу композиції над спадкуванням обумовлене

специфікою ігрової розробки, де об'єкти можуть мати різноманітні комбінації поведінок. Компонентний підхід забезпечує високий рівень модульності, розширюваності та підтримуваності коду.

Впроваджена система управління станами через стейт-машину забезпечує централізоване керування життєвим циклом додатку. Спроектовано 11 основних станів гри, кожний з яких має чітко визначену область відповідальності. Це дозволяє ефективно контролювати переходи між різними фазами гри та оптимізувати використання ресурсів.

Розроблено математичний апарат для ключових ігрових механік. Система обчислення прогресу виконання дій базується на нормалізованому інкрементальному наповненні, що забезпечує плавну візуалізацію незалежно від частоти кадрів. Економічна модель використовує нелінійну залежність винагороди від ефективності, створюючи додатковий стимул для оптимального виконання завдань.

Алгоритми просторового виявлення об'єктів, динамічного балансування книжкового фонду та управління камерою оптимізовані для мобільних пристроїв. Модифікований алгоритм рейкастингу з конусним скануванням забезпечує інтуїтивну взаємодію на сенсорних екранах, а система контролю камери реалізує плавні переходи через ефективні кубічні інтерполяції.

Важливим досягненням є створення системи персистентності даних на основі JSON-серіалізації, що поєднує гнучкість розробки з ефективністю виконання. Кожний компонент гри самостійно управляє власним станом через уніфікований інтерфейс, що спрощує масштабування та підтримку системи.

Таким чином, розроблене інформаційне та математичне забезпечення створює ефективну основу для подальшої реалізації мобільної гри LibTake, забезпечуючи оптимальний баланс між функціональністю та продуктивністю системи.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

При виборі інструментарію для розробки мобільного ігрового додатку LibTake було проведено аналіз наявних ігрових рушіїв на ринку з метою обрання оптимального рішення. Основними кандидатами для порівняння стали Unity[12], Unreal Engine[13] та Godot[14], оскільки вони є найпопулярнішими ігровими рушіями для розробки мобільних додатків.

Unity Engine було обрано як основний засіб розробки через низку переваг, що роблять його оптимальним вибором для створення мобільних ігор. Unity має широку підтримку кросплатформеної розробки, що дозволяє створювати додатки для Android з можливістю легкого портування на iOS та інші платформи. Частка ринку Unity складає приблизно 27,02% серед розробників ігор для мобільних платформ, що підтверджує його актуальність та поширеність в індустрії.

Порівняно з Unreal Engine, Unity пропонує більш інтуїтивний інтерфейс та значно нижчий поріг входу, що особливо важливо при розробці першого проєкту. Мова програмування C#, що використовується в Unity, є більш сучасною та простою для освоєння порівняно з C++ в Unreal Engine. Крім того, Unity має оптимізовані інструменти для роботи з 2D графікою та UI елементами, що є критичним для мобільних додатків.

На відміну від Godot, який має обмежені можливості для оптимізації та меншу спільноту розробників, Unity надає розширені інструменти для оптимізації продуктивності на різних пристроях, що особливо важливо для мобільних платформ з обмеженими ресурсами. Unity Asset Store містить велику кількість готових рішень, компонентів та інструментів, які значно прискорюють процес розробки.

Додатковими засобами розробки, що були обрані для проєкту, стали:

1. **Zenject (Extenject)** - фреймворк для впровадження залежностей, що дозволяє створювати більш модульний та тестований код. Вибір цього інструменту обумовлений його популярністю серед розробників ігор та здатністю спростити архітектуру проєкту[8].

2. **DoTween** - бібліотека для створення анімацій, що дозволяє легко реалізувати плавні переходи та ефекти. Цей інструмент є індустріальним стандартом для анімацій в Unity та зустрічається у більшості вакансій для розробників[11].
3. **UniTask** - бібліотека, що надає можливість використовувати сучасний асинхронний підхід із застосуванням конструкцій async/await в середовищі Unity. Це дозволяє писати більш читабельний та підтримуваний код при реалізації довгих операцій[10].
4. **Unity Input System** - нова система введення, що надає високорівневий API для обробки користувацького введення на різних платформах, забезпечуючи легке налаштування керування під мобільні пристрої[15].

Для написання коду використовувалось середовище розробки **Rider**[16] від JetBrains, яке має глибоку інтеграцію з Unity та надає розширені інструменти для рефакторингу та аналізу коду.

3.2 Вимоги до технічного та програмного забезпечення

Для коректної роботи мобільного ігрового додатку LibTake необхідне відповідне технічне та програмне забезпечення як для розробки, так і для кінцевого користувача. Розглянемо вимоги відповідно до архітектури відкритих систем стандарту OSI.

Системне програмне забезпечення для розробки:

- Операційна система: Windows 10/11, macOS 10.15 або новіше, Ubuntu 20.04 LTS або новіше
- Unity Hub 3.0 або новіше для керування версіями Unity
- Unity Editor 2021.3 LTS або новіше (використовується версія 2022.3 LTS для стабільності проєкту)
- Android SDK 30 або новіше (API Level 30+)
- JDK 11 або новіше для компіляції під платформу Android
- Git для системи контролю версій

Інструментальне програмне забезпечення для розробки:

- Середовище розробки Rider 2022.3 або новіше для написання коду C# (включає інтегровані інструменти для роботи з Git)

- Інструменти командного рядка для роботи з Unity (CLI)

Функціональне (прикладне) програмне забезпечення:

- Zenject 9.2.0 або новіше для впровадження залежностей
- DoTween 1.2.7 або новіше для анімацій
- UniTask 2.3.0 або новіше для асинхронного програмування
- Unity Input System 1.4.0 або новіше для керування
- TextMeshPro 3.0.6 або новіше для відображення тексту
- Newtonsoft JSON для серіалізації даних

Вимоги до пристрою кінцевого користувача:

- Операційна система: Android 8.0 (API Level 26) або новіше
- Процесор: ARM64 або x86_64 з частотою не менше 1.8 ГГц
- Оперативна пам'ять (RAM): мінімум 2 ГБ
- Вільне місце в пам'яті пристрою: мінімум 100 МБ
- Графічний процесор з підтримкою OpenGL ES 3.0 або вище
- Сенсорний екран з мінімальною роздільною здатністю 720x1280 пікселів
- Інтернет-з'єднання: не обов'язково (тільки для початкового завантаження з Google Play)

Для забезпечення оптимальної продуктивності та зручності використання рекомендується використовувати пристрої з більш потужними характеристиками: 4 ГБ оперативної пам'яті, восьмиядерний процесор та середній або високий клас графічного прискорювача.

Розробка гри виконана з урахуванням можливостей різних пристроїв та оптимізована для забезпечення плавного геймплею навіть на бюджетних моделях смартфонів. Це досягається завдяки використанню Universal Render Pipeline (URP) та інших технік оптимізації в Unity, що дозволяють знизити навантаження на апаратні ресурси при збереженні візуальної якості гри.

3.3 Опис програмної реалізації

Програмна реалізація мобільної гри LibTake побудована за принципами компонентно-орієнтованої архітектури та дотримується сучасних підходів до розробки програмного забезпечення. Цей підхід вибрано на противагу класичній

об'єктно-орієнтованій парадигмі з ієрархією наслідування, що дозволяє уникнути проблем "діамантового наслідування" та надмірного зв'язування компонентів. У контексті ігрової розробки компонентний підхід реалізується через створення невеликих, незалежних функціональних блоків (компонентів), які можна динамічно комбінувати для створення об'єктів з різними наборами поведінки. Це особливо важливо для гри LibTake, де гравець взаємодіє з різноманітними об'єктами (книгами, столами, відвідувачами), кожен з яких має унікальний набір характеристик і поведінок.

Компонентно-орієнтована архітектура гри базується на механізмі MonoBehaviour в Unity, але з розширеннями для забезпечення кращої структурованості коду. Замість створення глибоких ієрархій класів, кожна функціональність (читання книг, сканування, видача книг відвідувачам) виділена в окремий компонент, що відповідає за одну конкретну задачу. Наприклад, стіл для читання складається з компонентів BookStorage (зберігання книг), ReadingProgress (відстеження прогресу читання) та Interactable (взаємодія з гравцем), кожен з яких працює незалежно та може бути протестований і модифікований окремо.

У корені проекту знаходиться папка Assets, що містить усі ресурси гри, включаючи код, графіку, звуки, префаби, катсцени та сцени. Основний програмний код розділений на три логічні блоки: Editor, PlayModeTests та Runtime. Це забезпечує чітке розмежування між кодом, що використовується під час розробки, тестування та безпосереднього виконання гри, виключаючи потенційні проблеми із неправильно вказаними залежностями, що може унеможливити збірки фінального застосунку у результуючий файл.

Папка Runtime містить ключові компоненти, що забезпечують функціонування гри. Вона поділена на наступні підкаталоги:

- Data: зберігає структури даних та моделі для збереження прогресу гри;
- Infrastructure: містить базову інфраструктуру проекту, включаючи стейт-машину для керування станами гри;
- Logic: містить основну бізнес-логіку гри, включаючи реалізацію ігрових механік;

- Services: реалізує сервісний шар для доступу до функціоналу через інтерфейси;
- StaticData: містить конфігураційні дані, налаштування балансу та статичні ресурси;
- UI: містить компоненти користувацького інтерфейсу.

Вищеописана структура каталогів відображена на рис. 3.1, а файли, що знаходяться в них, мають простори імен, що відповідають цій структурі, враховуючи кожну папку в шляху до, власне, кожного конкретного файлу.

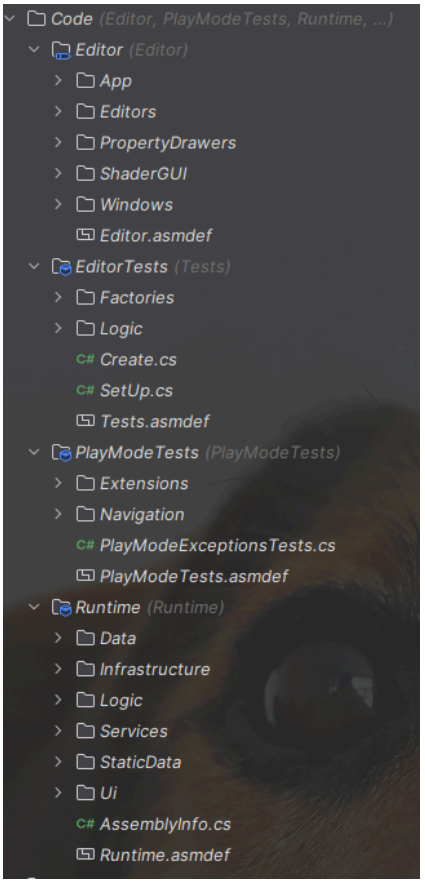

Рис 3.1. Скріншот структури проекту, відображеної в IDE Jetbrains Rider
Джерело: розроблено автором

Архітектурно проект побудований на основі трьох ключових концепцій: стейт-машина для управління життєвим циклом, сервісний підхід для організації бізнес-логіки та dependency injection для забезпечення слабкого зв'язування компонентів.

Стейт-машина (GameStateMachine) є центральним елементом архітектури, відповідальним за керування переходами між різними станами гри. Вона реалізована через систему інтерфейсів IExitableState, IGameState та IPayloadedState<TPayload>, що забезпечують єдиний контракт для всіх станів.

В лістингу 3.1 наведено ключові інтерфейси, що реалізуються станами стейт-машини.

Лістинг 3.1. Ключові інтерфейси основних станів гри.

```
public interface IExitableState
{
   public void Exit();
}
public interface IGameState : IExitableState
{
   public void Start();
}
public interface IPayloadedState<in TPayload> : IExitableState
{
   public void Start(TPayload payload);
}
```

Подібна організація абстракцій, дозволяє поліморфно працювати зі станами гри та реалізовувати їм лише необхідні для них методи. Це відповідає принципу Сегрегації інтерфейсів з принципів проектування S.O.L.I.D..

Контракт IGameState підходить для тих станів гри, котрі не мають додаткових параметрів для переходу в них, якими і є більшість можливих станів. Контракт IPayloadedState<in TPayload> створений для станів, котрим потрібно отримати параметр для коректної роботи. Прикладами таких станів є LoadLevelState, що потребує назви рівня для його коректного завантаження, що передається йому типом string, і LoadProgressState, котрий приймає тип даних LoadProgressOption, в залежності від вмісту якого відбувається завантаження збереженої гри або починається нова гра.

Можливість працювати з різними типами параметрів реалізована за допомогою шаблонів інтерфейсів та шаблонів методів, дозволяючи зменшити кількість boilerplate-коду до мінімуму, зробивши систему максимально гнучкою.

Також, максимальна гнучкість станів забезпечується можливістю легкого отримання будь-якої необхідної залежності одразу через конструктор класу, що реалізує стан.

Після розгляду інтерфейсів станів, важливо зрозуміти повну архітектуру взаємодії компонентів системи, що забезпечує життєвий цикл гри. Стейт-машина є центральним елементом, але її ефективна робота залежить від ряду супутніх компонентів та їхньої правильної інтеграції.

Стейт-машина в LibTake функціонує як координатор високорівневих станів гри, забезпечуючи їх правильне завантаження та виконання. Реалізація стейт-машини базується на шаблоні проектування State, який дозволяє об'єкту змінювати свою поведінку при зміні внутрішнього стану.

Ключовий клас GameStateMachine надає два методи для зміни станів, реалізацію котрих можна побачити на лістингу 3.2, де упущені деякі деталі, але передано основний підхід, включаючи шаблони методів, обмеження, розбиття коду на дрібні функції.

Лістинг 3.2. Переключення станів гри.

```
public void EnterState<TState>()
   where TState : class, IGameState
{
   TState state = ChangeState<TState>();
   state.Start();
}

public void EnterState<TState, TPayload>(TPayload payload)
   where TState : class, IPayloadedState<TPayload>
{
   TState state = ChangeState<TState>();
   state.Start(payload);
```

```
}
```

Ці методи демонструють гнучкість архітектури, дозволяючи переходити як до простих станів, так і до станів, що потребують додаткових параметрів. Наприклад, стан LoadLevelState приймає строкове значення з назвою рівня, а LoadProgressState — параметр, що визначає тип завантаження прогресу.

Детальну реалізацію стейт-машини та супутніх класів продемонстровано на рис. 3.2., де можна побачити як і саму стейт-машину, так і один зі станів для прикладу, а також фабрику та di-інсталлер, що разом забезпечують її роботу.

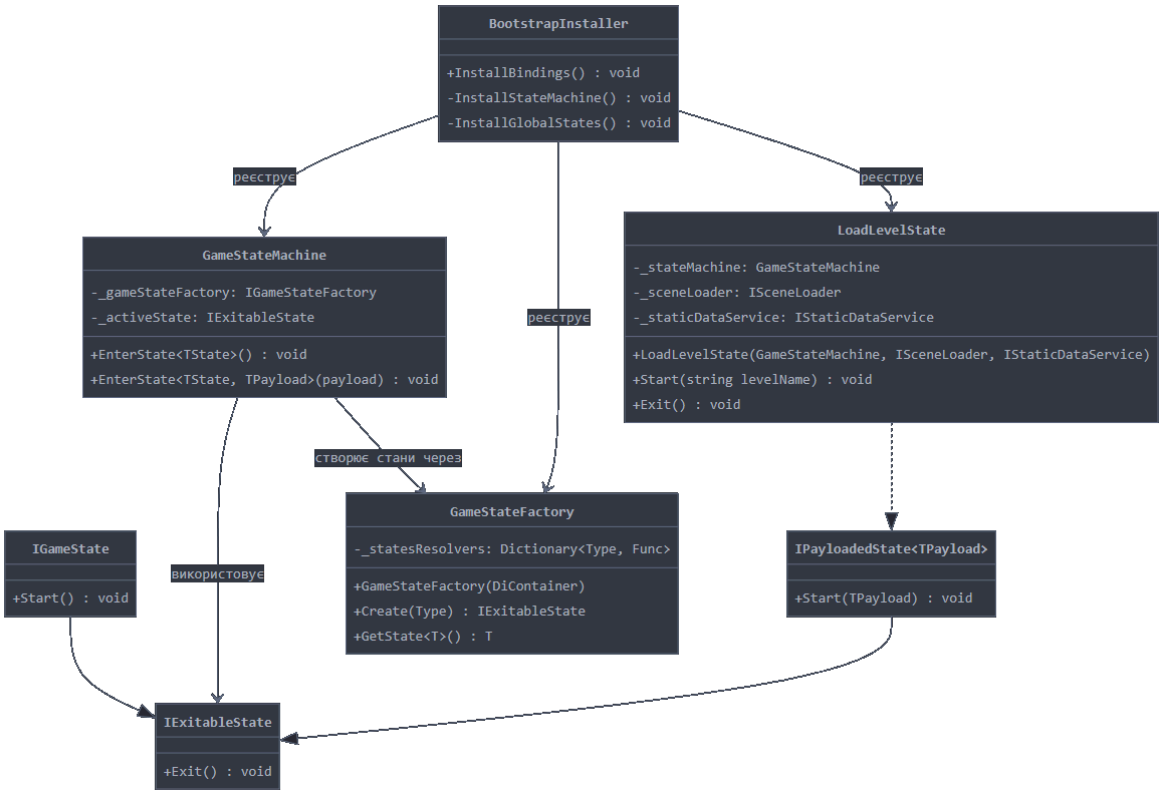

Рис 3.2. Діаграма реалізації центральної стейт машини та супутніх класів
Джерело: розроблено автором

Для створення конкретних екземплярів станів використовується патерн фабрика, реалізований через клас GameStateFactory. Фабрика отримує контейнер залежностей як параметр конструктора і використовує його для створення екземплярів станів, що дозволяє автоматично вирішувати всі залежності кожного

стану. Таким чином, якщо стан LoadLevelState потребує сервіси ISceneLoader та IStaticDataService, вони будуть автоматично надані при створенні цього стану.

Для ініціалізації та реєстрації всіх компонентів у контейнері залежностей використовується клас BootstrapInstaller. Він є своєрідним конфігуратором системи, що визначає, які сервіси та в якому режимі (singleton, transient) будуть доступні в грі:

Ключовими станами є BootstrapGameState (початкова ініціалізація), MenuGameState (головне меню), LoadLevelState (завантаження рівня), MorningGameState (фаза планування), DayGameState (робоча фаза) та інші. Детально описано функціонал кожного стану в підрозділі 2.2.

Організація збереження даних гри здійснюється через поділ на дані гравця та ігрового світу, що реалізовано за допомогою контейнерів PlayerData та WorldData. Ці контейнери містять атрибут серіалізації, який забезпечує можливість їх збереження та завантаження у текстовому форматі.

Для реалізації процесів серіалізації та десеріалізації даних було використано бібліотеку Newtonsoft JSON, яка вважається однією з найпопулярніших та найуніверсальніших у сфері розробки ігор. Формат JSON був обраний через його універсальність, простоту імплементації та можливість редагування даних безпосередньо у файлі збереження за допомогою стандартних текстових редакторів, що значно спрощує процес тестування та налагодження гри. Детальніша структура даних представлена в лістингу 3.3.

Лістинг 3.3. Переключення станів гри.

```
[Serializable]
public sealed class PlayerData
{
    public PlayerInventoryData Inventory = new();
    public Dictionary<BookType, int> Skills = new();
    public List<string> BooksRead = new();
    public List<string> BooksScanned = new();
    public int Lives;
    public GlobalGoalSavedData GlobalGoal = new();
}
```

У представленій структурі PlayerData використано власні типи даних для забезпечення максимальної зрозумілості коду, а також стандартні типи колекцій C# - List та Dictionary. Такий підхід дозволяє зберігати різноманітну інформацію про гравця, включаючи вміст інвентаря, набуті навички, прочитані та відскановані книги, кількість життів та дані про глобальну мету.

Аналогічно до структури PlayerData, WorldData містить інформацію про стан ігрового світу, використовуючи власні структури даних для оптимального представлення різних аспектів ігрового середовища. Це включає стан книжкових тримачів, прогрес різних процесів, стан столів для крафтингу, дані про доставку книг та часові показники. Це відображено у лістингу 3.4.

Лістинг 3.4. Демонстрація структури організації даних для збереження світу гри.

```
[Serializable]
public sealed class WorldData
{
    public BookHoldersState BookHoldersState = new();
    public ProgressesStates ProgressesStates = new();
    public CraftingTableStates CraftingTableStates = new();
    public BooksDeliveringData BooksDeliveringData = new();
    public TimeData TimeData = new();
}
```

Механізм збереження та завантаження гри реалізований за децентралізованим принципом, де кожен конкретний об'єкт відповідає лише за збереження та завантаження даних, що безпосередньо його стосуються. Цей підхід застосовує принципи виділення абстракцій та сегрегації інтерфейсів, що реалізуються через контракти ISavedProgressReader та ISavedProgress.

Інтерфейс ISavedProgressReader надає можливість ігровому об'єкту завантажити дані, що його стосуються. Методу LoadProgress передається повна інформація про стан гри, а об'єкт, що реалізує цей інтерфейс, самостійно визначає та використовує необхідні йому дані для відновлення власного стану.

Інтерфейс ISavedProgressReader приведений у лістингу 3.5, котрий реалізує всього один метод, не зобов’язуючи об’єкти, котрим необхідно виключно зчитувати прогрес гравця з довгострокової пам’яті, до реалізації надлишкового функціоналу.

Лістинг 3.5. Інтерфейс ISavedProgressReader, що визначає контракт завантаження даних гри для конкретного ігрового об'єкту.

```
public interface ISavedProgressReader
{
    public void LoadProgress(GameProgress progress);
}
```

Інтерфейс ISavedProgress приведений у лістингу 3.6, котрий також реалізує лише один метод, але в той же час логічно наслідує ISavedProgressReader щоб мати можливість зчитувати дані, окрім запису.

Лістинг 3.6. Інтерфейс ISavedProgress, що визначає контракт завантаження даних гри для конкретного ігрового об'єкту.

```
public interface ISavedProgress : ISavedProgressReader
{
    public void UpdateProgress(GameProgress progress);
}
```

За аналогією до процесу завантаження прогресу, процес збереження також реалізується через передачу об'єкту інформації про стан гри, де ігровий об'єкт самостійно визначає, які дані про себе він має оновити.

Таким чином, ISavedProgressReader та ISavedProgress формують децентралізовану систему збереження та завантаження даних гри, де кожна сутність відповідає виключно за роботу з власними даними. Це дозволяє уникнути створення громіздких об'єктів та підтримувати проект у довгостроковій перспективі, зберігаючи ефективність розробки навіть за умови зростання складності кодової бази.

Для ефективного управління об'єктами, що потребують збереження даних, використовується реєстр SaveLoadRegistry, який відстежує всі компоненти з можливістю збереження та завантаження прогресу. Цей реєстр дозволяє централізовано керувати процесами збереження та відновлення стану гри.

Центральним компонентом системи є сервіс SaveLoadService, який координує процеси збереження та завантаження даних. Цей сервіс відповідає за взаємодію з PlayerPrefs - вбудованим механізмом Unity для збереження невеликого обсягу даних. Окрім збереження прогресу гри, даний сервіс також зберігає налаштування аудіо, камери та вибір персонажа.

3.3.3. Налаштування балансу гри

У процесі розробки мобільної гри значна увага приділялася створенню та налаштуванню балансу геймплею, що безпосередньо впливає на якість ігрового досвіду користувача. Для ефективного управління численними параметрами гри було застосовано технологію ScriptableObject, запропоновану рушієм Unity, що дозволяє виокремити конфігураційні дані від програмної логіки та забезпечує можливість їх модифікації без перекомпіляції проекту.

Архітектурне рішення щодо використання ScriptableObject обґрунтоване необхідністю швидкого та зручного налаштування параметрів гри у процесі розробки та тестування. Як зазначено у лістингу 2.6, ці об'єкти надають можливість створення автоматичних редакторів інтерфейсу для модифікації значень безпосередньо в середовищі Unity Editor. Подібний підхід відповідає принципам сучасної розробки програмного забезпечення, зокрема принципу розділення відповідальності, що дозволяє проектувати більш підтримувану та масштабовану структуру проекту.

У розробленій грі впроваджено багаторівневу систему конфігурації, яка охоплює всі ключові аспекти геймплею. Ця система включає налаштування характеристик гравця (життя, швидкість переміщення, тривалість виконання дій), параметри ігрової економіки (винагорода за завдання, вартість покращень), часові параметри (тривалість ігрового дня, фаз планування та активної гри) та параметри складності (інтенсивність появи відвідувачів, вимоги до крафтів).

Для централізованого управління конфігураційними даними розроблено сервіс StaticDataService, який відповідає за завантаження всіх статичних даних при ініціалізації гри та надає уніфікований інтерфейс доступу до них іншим компонентам системи. Така архітектура відповідає принципам SOLID, зокрема принципу інверсії залежностей, що підвищує гнучкість системи та спрощує її тестування.

Особливу увагу в розробці приділено системі прогресії складності, яка забезпечує поступове зростання викликів для гравця відповідно до його прогресу. Для цього впроваджено конфігураційні об'єкти з використанням AnimationCurve, що дозволяють гнучко налаштовувати зміну параметрів геймплею в часі через графічний інтерфейс Unity Editor. Цей механізм застосовано для налаштування таких аспектів як частота появи відвідувачів, їхнє терпіння та множники винагород.

Важливим елементом балансування є система глобальних цілей, кожна з яких має власні вимоги до навичок гравця та необхідних ресурсів. Для управління цими параметрами створено спеціалізовані конфігураційні об'єкти, що містять детальні специфікації кожного компонента, необхідного для досягнення цілі. Ці параметри піддавалися ретельному калібруванню для забезпечення оптимального темпу прогресії гравця та підтримки його мотивації через відчуття поступового досягнення.

Для полегшення входження нових користувачів у гру розроблено систему навчальних елементів, яка надає контекстні підказки на початкових етапах гри. Інтенсивність та частота цих підказок також налаштовується через конфігураційні об'єкти, що дозволяє знайти баланс між допомогою користувачу та заохоченням його до самостійного освоєння ігрових механік.

Для ефективного тестування та аналізу балансу гри розроблено спеціалізовані інструменти, які включають режим прискореного геймплею для швидкої перевірки довгострокових ефектів зміни параметрів, системи відладки для моніторингу ключових показників під час гри та інструменти статистики для збору та аналізу даних про проходження гри різними користувачами.

Застосування ScriptableObject у поєднанні з розробленою системою сервісів дозволило створити гнучку та ефективну інфраструктуру для налаштування балансу гри, що відповідає сучасним стандартам розробки ігрових додатків. Цей підхід забезпечив можливість ітеративного вдосконалення ігрового досвіду на основі результатів тестування та зворотного зв'язку від користувачів, що є критично важливим для створення якісного ігрового продукту.

Сервісний підхід забезпечує чітке розділення відповідальності між компонентами системи. Кожен сервіс відповідає за конкретний аспект гри (наприклад, PlayerProviderService, BookSlotInteractionService, StaticDataService) та доступний через відповідний інтерфейс. Такий підхід дозволяє легко змінювати реалізацію без впливу на інші частини системи.

Для впровадження залежностей використовується фреймворк Extenject (Zenject), що дозволяє автоматично вирішувати залежності між компонентами. Це особливо важливо для MonoBehaviour-компонентів, які не підтримують конструктори в класичному розумінні. Реєстрація сервісів відбувається в інсталерах (наприклад, BootstrapInstaller), а їх використання через ін'єкцію в конструкторах або спеціальних методах, позначених атрибутом [Inject].

Асинхронні операції реалізовані за допомогою бібліотеки UniTask, що надає можливість використовувати сучасний синтаксис async/await замість традиційних корутин Unity. Це дозволяє писати більш зрозумілий код для складних послідовних операцій, таких як завантаження рівня, анімації та мережеві запити.

Анімації елементів інтерфейсу та ігрових об'єктів реалізовані через бібліотеку DoTween, що забезпечує плавні переходи та ефекти. Ця бібліотека дозволяє створювати складні послідовності анімацій з використанням різних кривих інтерполяції, що значно покращує користувацький досвід.

Для забезпечення мультимовності гри використовується офіційний пакет Unity Localization, який дозволяє легко додавати підтримку нових мов без зміни основного коду.

Програмний код організований відповідно до принципів чистого коду, з дотриманням єдиної відповідальності, інверсії залежностей та інших принципів

SOLID. Середній розмір файлу становить 38 рядків, що свідчить про високий рівень декомпозиції та відсутність God-Objects, які ускладнюють розуміння та підтримку коду.

Для забезпечення продуктивності на мобільних пристроях використовуються техніки оптимізації, такі як об'єднання статичних об'єктів, ефективне використання пам'яті через пулінг об'єктів та мінімізація виділень пам'яті під час виконання. Візуальна складова оптимізована через Universal Render Pipeline, що забезпечує високу якість графіки при збереженні продуктивності на пристроях з обмеженими ресурсами.

3.4 Керівництво користувача

У процесі розробки мобільного ігрового додатку LibTake була імплементована комплексна методологія тестування, що включала різноманітні підходи до верифікації функціональності та валідації відповідності вимогам. Застосовані методи тестування:

1. Модульне тестування компонентів системи Ключові архітектурні компоненти, зокрема GameStateMachine, PlayerProviderService та BookSlotInteractionService, були піддані ізольованому тестуванню через фреймворк NUnit із застосуванням технології інверсії залежностей та заміщення залежностей об'єктами-двійниками (mock-об'єктами) за допомогою NSubstitute. Даний підхід дозволив виявити проблемні аспекти функціонування окремих модулів на ранніх етапах розробки.
2. Інтеграційне тестування міжкомпонентної взаємодії Валідація коректності взаємодії між компонентами системи проводилася з особливим фокусом на комунікацію між сервісами та станами стейт-машини. Верифіковано коректність трансферу даних між SaveLoadService та об'єктами ігрового середовища, функціональність системи персистентності та інших критичних аспектів взаємодії.
3. Валідація користувацького інтерфейсу Процес валідації включав перевірку відповідності відображення елементів інтерфейсу специфікації, коректності

реакції на користувацький ввід та плавності анімаційних ефектів на різних апаратних конфігураціях з варіативними параметрами дисплею.

4. Аналіз продуктивності при екстремальних навантаженнях Оцінювання ефективності функціонування системи в умовах максимального навантаження, що включало сценарії з високою кількістю ігрових об'єктів та паралельними процесами, дозволило ідентифікувати потенційні обмеження продуктивності та оптимізувати алгоритмічну складову.
5. Бета-тестування за участю фокус-групи Залучення диверсифікованої групи потенційних користувачів дозволило отримати кваліфікований зворотний зв'язок щодо ергономіки ігрового процесу та виявити нетипові сценарії використання, що не були передбачені на етапі проектування.

Для демонстрації роботи додатку, нижче описано шлях взаємодії користувача з грою.

Інтерфейс ініціалізації представляє користувачеві селекцію глобальної цілі, після чого система транзитує до першої фази планування ігрового дня.

У фазі стратегічного планування користувач має можливість здійснити маніпуляції з об'єктами книг, отримуючи їх із транспортного засобу та розміщуючи на спеціалізованих поверхнях у віртуальному просторі бібліотеки. Інтерфейс забезпечує інформаційну підтримку щодо категоризації літературних об'єктів та їх впливу на прогресію навичок персонажа.

При транзиції до активної фази система ініціює алгоритм генерації відвідувачів та активує часовий обмежувач. Користувач реалізує наступну послідовність дій:

1. Здійснює позиціонування персонажа поблизу відвідувача для отримання інформаційного запиту
2. Ініціює процес документальної фіксації в реєстраційній картці через інтерактивну взаємодію з відповідним об'єктом
3. Локалізує запитаний літературний об'єкт у сховищі
4. Транспортує об'єкт до запитувача

Успішна реалізація запиту винагороджується віртуальною валютою, що може бути алокована на модернізацію устаткування або регенерацію життєвих показників.

За умови наявності часового ресурсу, користувач має можливість:

- Ініціювати процес інтелектуального засвоєння інформації з літературних об'єктів для розвитку професійних компетенцій
- Здійснювати дигіталізацію книг для генерації додаткового фінансового ресурсу
- Синтезувати компоненти для реалізації глобальної цілі

Після вичерпання часового ліміту активної фази, система транзитує до фази планування наступного дня, де цикл повторюється з прогресуючим рівнем складності, що характеризується збільшенням інтенсивності потоку відвідувачів та редукцією часового ресурсу для реалізації запитів.

Механізми обробки аномальних ситуацій

1. Декремент життєвих показників при порушенні часових обмежень У разі невиконання запиту відвідувача в регламентований період система реалізує такий алгоритм обробки:
    - Візуалізується анімаційна послідовність, що відображає негативну реакцію відвідувача
    - Здійснюється декремент лічильника життєвих показників
    - Система генерує інформаційне повідомлення про редукцію життєвих показників
    - Віртуальний об'єкт відвідувача видаляється з черги
2. Функціональність системи зберігається, забезпечуючи можливість обслуговування інших відвідувачів або регенерації втраченого життєвого показника через інтеракцію з об'єктом статуї.
3. Інцидент недоступності функції крафтингу При спробі ініціації синтезу компонента глобальної цілі за відсутності достатнього рівня компетенцій або фінансових ресурсів система:
    - Генерує інформаційне повідомлення про невідповідність ресурсних параметрів

- Візуально акцентує невиконані критерії через хроматичну модифікацію
- Пропонує алгоритмічно сформовані рекомендації щодо літературних об'єктів, асоційованих з необхідними компетенціями

4. Аномалія при персистенції прогресу У випадку виникнення технічних обмежень при збереженні даних (недостатність апаратного ресурсу зберігання):
   - Система візуалізує діалогове вікно з дескриптором помилки
   - Пропонує альтернативні сценарії дій (оптимізація апаратного ресурсу, продовження без персистенції)
   - За критичних умов створює тимчасову резервну копію даних для подальшого відновлення
5. Обробка аномалій завантаження ресурсних компонентів При виникненні технічних обмежень при завантаженні графічних або аудіальних ресурсів:
   - Система застосовує альтернативні ресурси редукованої якості
   - Візуалізується індикатор прогресу завантаження
   - За критичних умов пропонується реініціалізація додатку з діагностичною інформацією
6. Відновлення функціональності після критичного збою У випадку непередбаченого припинення роботи додатку:
   - При наступній ініціалізації візуалізується діалог відновлення
   - Система ініціює процедуру завантаження останньої валідної версії збереженої прогресії
   - Користувачеві пропонується вибір між продовженням з точки аномалії або з початку останнього ігрового дня

Завдяки імплементації комплексної методології тестування та механізмів обробки аномальних ситуацій, мобільний ігровий додаток LibTake забезпечує високий рівень стабільності функціонування за різноманітних умов експлуатації, що суттєво підвищує загальну якість програмного продукту та рівень задоволеності користувацької аудиторії.

Висновки

У третьому розділі кваліфікаційної роботи детально розглянуто програмне та технічне забезпечення мобільного ігрового додатку "LibTake". Проведено аналіз та обґрунтовано вибір інструментарію для розробки, де основним засобом став Unity Engine, завдяки його перевагам у кросплатформеній розробці, інтуїтивному інтерфейсу та оптимізації для мобільних пристроїв. Додатково залучено спеціалізовані бібліотеки, зокрема Zenject для впровадження залежностей, DoTween для створення анімацій, UniTask для асинхронного програмування.

Визначено вимоги до технічного та програмного забезпечення як для розробки, так і для кінцевого користувача відповідно до архітектури відкритих систем стандарту OSI. Детально описано програмну реалізацію додатку, побудовану за принципами компонентно-орієнтованої архітектури. Центральним елементом архітектури стала стейт-машина, яка керує переходами між різними станами гри. Запропоновано ефективну систему збереження даних з використанням контейнерів PlayerData та WorldData, що забезпечують серіалізацію та десеріалізацію ігрового стану.

Особливу увагу приділено налаштуванню балансу гри через технологію ScriptableObject, що дозволяє відокремити конфігураційні дані від програмної логіки. Визначено методи тестування програмного забезпечення, що включають модульне тестування, інтеграційне тестування, валідацію користувацького інтерфейсу та аналіз продуктивності. Наведено керівництво користувача, що докладно описує послідовність взаємодії з грою та механізми обробки аномальних ситуацій.

Розроблений мобільний ігровий додаток "LibTake" відповідає сучасним стандартам розробки програмного забезпечення, забезпечує високу якість користувацького досвіду та демонструє практичне застосування компетентностей, здобутих під час навчання.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено мобільний ігровий додаток "LibTake" для платформи Android з використанням сучасних індустріальних інструментів у середовищі ігрового рушія Unity та основних бібліотек, що є стандартами індустрії. Розробка здійснювалася з метою демонстрації освітніх можливостей Національного університету "Острозька академія", залучення нових абітурієнтів та практичного застосування компетентностей, здобутих під час навчання.

У процесі виконання роботи було проведено детальний аналіз предметного середовища функціонування комп'ютерних ігор, що дозволило встановити їх значимість у сучасній цифровій економіці та культурі. Дослідження аналогічних розробок в сфері інді-ігор (Overcooked, PlateUp, The Escapists 2) дало можливість виявити ключові принципи проектування успішних ігрових продуктів та визначити перевірені успішними резілами ігор кращі практики.

Архітектура додатку побудована на компонентно-орієнтованому підході з використанням глобальної стейт-машини для управління життєвим циклом гри, що забезпечило гнучкість, модульність та масштабованість системи. Розроблено ефективну систему збереження даних з використанням контейнерів PlayerData та WorldData, що серіалізуються та десеріалізуються через JSON, фективно зберігаючі та завантажуючі стан ігрового світу.

Математичне забезпечення додатку включає системи обчислення прогресу виконання дій, економічну модель з нелінійною залежністю винагороди від ефективності, алгоритми динамічного балансування книжкового фонду та просторового виявлення об'єктів, що оптимізовані для мобільних пристроїв.

Програмна реалізація здійснена за допомогою мови C# з використанням ігрового рушія Unity та спеціалізованих бібліотек: Zenject для впровадження залежностей, DoTween для процедурних анімацій з коду, UniTask для інтеграції асинхронного програмування в Unity та Newtonsoft JSON для серіалізації даних.

Структура коду організована відповідно до принципів чистої архітектури та SOLID, що забезпечує високу підтримуваність та розширюваність проекту.

Особливу увагу приділено налаштуванню балансу гри через технологію ScriptableObject, що дозволяє відокремити конфігураційні дані від програмної логіки та забезпечує можливість їх модифікації без перекомпіляції проекту.

Розроблений мобільний ігровий додаток "LibTake" відповідає сучасним стандартам розробки програмного забезпечення, забезпечує високу якість користувацького досвіду та демонструє практичне застосування компетентностей, здобутих під час навчання. Додаток успішно реалізує поставлену мету щодо гейміфікації бібліотечного середовища Острозької академії, створюючи привабливе та інтерактивне представлення навчального закладу для потенційних абітурієнтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Оцінки доходів з ігор, фільмів та музики журналу Forbes URL: https://www.forbes.com/councils/forbesagencycouncil/2023/11/17/the-gaming-industry-a-behemoth-with-unprecedented-global-reach/(дата відвідування: 28.04.2025)
2. How Many Gamers Are There? (2024 Statistics) URL: https://whatsthebigdata.com/number-of-gamers/(дата відвідування: 28.04.2025)
3. Wolrdmeters URL: https://www.worldometers.info/uk/(дата відвідування: 28.04.2025)
4. Global Report Reveals Positive Benefits of Video Gameplay URL: https://www.theesa.com/global-report-reveals-positive-benefits-of-video-gameplay/(дата відвідування: 28.04.2025)
5. Overcooked URL: https://store.steampowered.com/app/448510/Overcooked/(дата відвідування: 29.04.2025)
6. PlateUp URL: https://store.steampowered.com/app/1599600/PlateUp/(дата відвідування: 29.04.2025)
7. The Escapists 2 URL: https://store.steampowered.com/app/641990/The_Escapists_2/(дата відвідування: 29.04.2025)
8. Документація Zenject(Extenject) DI URL: https://github.com/Mathijs-Bakker/Extenject(дата відвідування: 15.05.2025)
9. Unity документація по завантаженню ресурсів URL: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Resources.html(дата відвідування: 15.05.2025)
10. Документація бібліотеки Unitask URL: https://github.com/Cysharp/UniTask(дата відвідування: 15.05.2025)
11. Документація бібілотеки DoTween URL: https://dotween.demigiant.com/(дата відвідування: 15.05.2025)
12. Документація ігрового рушія Unity URL: https://docs.unity.com/(дата відвідування: 15.05.2025)

13. Документація ігрового рушія Unreal Engine URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation(дата відвідування: 15.05.2025)
14. Документація ігрового рушія Godot Engine URL: https://docs.godotengine.org/en/stable/(дата відвідування: 15.05.2025)
15. Документація плагіна Unity Input System URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.14/manual/index.html (дата відвідування: 15.05.2025)
16. Вебсайт Jetbrains Rider IDE URL: https://www.jetbrains.com/(дата відвідування: 15.05.2025)