Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ІНТЕРАКТИВНИЙ 3D-ЗАСТОСУНОК CRYPTO RUNNER GO ДЛЯ РІЗНИХ ПЛАТФОРМ

Виконав: магістрант групи 2КІ-24м спеціальності 123 — Комп'ютерна Інженерія
(шифр і назва напряму підготовки, спеціальності)
Химич Б.В.
(прізвище та ініціали)
Керівник: к.т.н., доц., доцент кафедри ОТ Колесник І.С.
(прізвище та ініціали)
«12» 12 2025р.

Опонент:

к.т.н., доцент, зав. каф. МБІС Карпінець В.В.
(прізвище та ініціали)
«12» 12 2025 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«17» 12 2025 р.

ВНТУ 2025

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія
Освітньо-професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«25» вересня 2025 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Химичу Богдану Валентиновичу

1 Тема роботи: «Інтерактивний 3D-застосунок CRYPTO RUNNER GO для різних платформ», керівник роботи Колесник І.С. к.т.н., доц., доцент кафедри ОТ затверджені наказом вищого навчального закладу від «24» вересня 2025 року №313.

2 Строк подання студентом роботи 04.12.2025

3 Вихідні дані до роботи: проєкт ігрового застосунку, створений в середовищі Unity, графічні 3D- та 2D-елементи у форматах PNG та FBX, мова C#, у середовищі Visual Studio.

4 Зміст розрахунково-пояснювальної записки: вступ, еволюція ігор у суспільстві, проєктування архітектури та дизайну гри, реалізація гри "CRYPTO RUNNER GO", експериментальні дослідження та тестування функціонування програми, висновки.

5 Перелік графічного матеріалу: схема структури гри, інтерфейс головного меню.

6 Консультанти розділів роботи

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийня
1-4	к.т.н., доц., каф. ОТ, Колесник Ірина Сергіївна		
5	к.т.н., доц. кафедри ЕПВМ Адлер Оксана Олександрівна		
Нормоконтроль	асистент кафедри ОТ Швець Сергій Ілліч		

7 Дата видачі завдання 25.09.2025 р.

8 Календарний план наведено в таблиці 2.

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Срок виконання	Примітка
1.	Постановка задачі	26.09.2025	
2.	Аналіз задачі	08.10.2025	
3.	Огляд перших відеоігор	17.10.2025	
4.	Огляд існуючих мов програмування та платформ для створення ігор	28.10.2025	
5.	Структура додатка	01.11.2025	
6.	Програмна реалізація застосунку	05.11.2025	
7.	Інтеграція фреймворків	10.11.2025	
8.	Оформлення пояснювальної записки та презентації	11.11.2025	
9.	Перевірка якості виконання роботи та усунення недоліків	01.12.2025	

Студент _____ Химич Б.В.

Керівник роботи _____ Колесник І.С.

УДК 004.932.2:004.8

Химич Б.В. 3D гра Сгу
кваліфікаційна робота зі сп
програма комп'ютерна інже
На укр. мові. Бібліогр.:
У магістерській квалі
рушію Unity, написаний м
сучасних вимог, з урахува
одночасно розробляти дода
використовувати час. Прогр
Також були інтегровані зо
публікації. Дана гра опублі

Ключові слова: ігров

АНОТАЦІЯ

УДК 004.932.2:004.8

Химич Б.В. 3D гра Crypto Runner Go для мобільних телефонів. Магістерська кваліфікаційна робота зі спеціальності 123 — комп'ютерна інженерія, освітня програма комп'ютерна інженерія. Вінниця: ВНТУ, 2025. 92с.

На укр. мові. Бібліогр.: 30 назв, рис. 47, табл. 9, ліст.13 .

У магістерській кваліфікаційній роботі розробленню ігрового додатку на рушію Unity, написаний мовою C#. Додаток було реалізовано відповідно до сучасних вимог, з урахуванням інтересів користувачів. Завдяки Unity можна одночасно розробляти додаток для 25 різних платформ, що дозволяє ефективно використовувати час. Програмна частина розроблена у середовищі Visual Studio. Також були інтегровані зовнішні бібліотеки для оптимізації проекту та його публікації. Дана гра опублікована на платформі App Store.

Ключові слова: ігровий додаток, Unity, C#, Visual Studio, App Store.

ABSTRACT

Khymych B.V. 3D Game “Crypto Runner Go” for Mobile Phones. Master’s Qualification Thesis in Specialty 123 – Computer Engineering, Educational Program “Computer Engineering”. Vinnytsia: VNTU, 2025. 92 p.

Language: Ukrainian. References: 30 sources, 47 figures, 9 tables, 13 listings.

In my master's thesis, I developed a game application on the Unity engine, written in C#. The application was implemented in accordance with modern requirements, taking into account the interests of users. Thanks to Unity, it is possible to simultaneously develop applications for 25 different platforms, allowing for efficient use of time. The software part was developed in the Visual Studio environment. External libraries were also integrated to optimize the project and its publication. This game has been published on the App Store platform.

Keywords: gaming application, Unity, C#, Visual Studio, App Store.

ЗМІСТ

ВСТУП

У сучасному світі люди постійно шукають способи відволіктись від щоденних турбот і знайти можливість для відпочинку. Одним із таких способів стали відеоігри, які доступні на мобільних пристроях та персональних комп’ютерах. Мотивація до гри у кожного індивідуальна: хтось прагне нових знайомств, інші шукають розваги, відпочинок або азарт змагання. Ігри різних типів задовольняють різноманітні потреби гравців. Варто зазначити, що культура ігор існувала ще задовго до появи цифрових технологій — прикладами є шахи, доміно чи карткові ігри. З появою комп’ютерів ігрова індустрія отримала новий поштовх до розвитку. Отже, відеоігри стали невід’ємною частиною дозвілля та одним із найпоширеніших засобів розваги для широкої аудиторії.

Актуальність дослідження полягає в тому, що мобільні телефони стали невід'ємною складовою повсякденного життя більшості людей. Їхня широка доступність спричиняє зростання попиту на нові ігрові застосунки, які мають відповідати очікуванням і потребам сучасних користувачів. Завдяки мобільним пристроям користувачі отримують змогу грати в будь-який зручний для них час і в будь-якому місці, що робить мобільні ігри особливо популярними засобами для дозвілля та релаксації.

Мета роботи полягає у створенні мобільного ігрового застосунку в жанрі ранер, з інтуїтивно зрозумілим інтерфейсом та сучасним візуальним оформленням.

Об'єкт дослідження — процес розробки мобільного ігрового застосунку.

Предмет дослідження — мобільна гра в жанрі ранер.

Наукова новизна полягає у вперше застосованому компонуванні технологій Unity, Zenject, DOTween, Cinemachine та UniTask для створення мобільної 3D-гри жанру ранер, що забезпечує високу адаптивність архітектури.

Апробація результатів роботи: матеріали роботи опубліковані.

3D гра CRYPTO RUNNER GO для мобільних телефонів // Химич Б.В., Черняк О.І., Матеріали LIII науково-технічної конференції підрозділів ВНТУ (Вінниця, 2024 р.) [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20062

1 ЕВОЛЮЦІЯ ІГОР У СУСПІЛЬСТВІ

1.1 Історія появи перших відеоігор

Історія виникнення відеоігор тісно пов’язана з еволюцією комп’ютерних технологій і зміною суспільних запитів. Поява програмованих комп’ютерів, спочатку розроблених переважно для військових потреб, створила технічну базу для подальшого використання цих пристроїв у сфері розваг. Згодом розвиток телебачення також сприяв популяризації відеоігор серед широкого кола користувачів [1].

Перші кроки у створенні цифрових ігор відбувалися ще у 1940–1950-х роках. Одним із таких проєктів стала гра Bertie the Brain, створена у 1950 році — це була електронна машина, яка дозволяла грати у хрестики-нулики. Її публічна демонстрація відбулася на Канадській національній виставці, що стало важливою подією у формуванні перших уявлень про взаємодію людини з машиною в розважальному форматі, що представлено на рисунку 1.1.

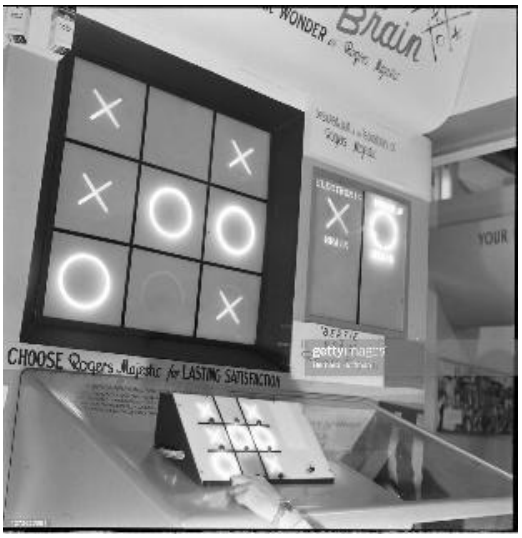

Рисунок 1.1 — Bertie the Brain

Попри те, що Bertie the Brain не мав значного впливу на подальший розвиток ігрової індустрії, він став одним із перших комп’ютерів, створених для розважального використання, і може вважатися одним із перших прикладів цифрової гри.

Наступним етапом стало створення у 1961 році гри Mouse in the Maze, яку

розробили у Массачусетському технологічному інституті. У цій грі користувач міг за допомогою світлового пера створити лабіринт, а потім спостерігати, як віртуальна миша шукає шматочки сиру. Це був ранній приклад взаємодії користувача з комп’ютерною графікою, що представлено на рисунку 1.2.

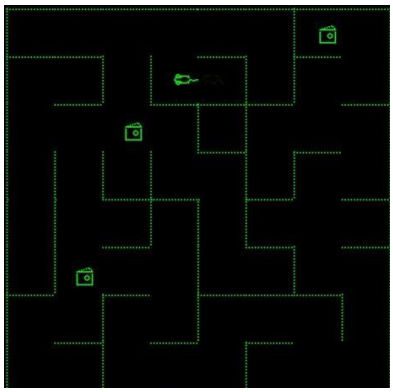

Рисунок 1.2 — Знімок екрану з першої відеогри, «Mouse in the Maze»

З розвитком технологій відеоігри поступово перетворилися на невід’ємну частину сучасної культури та індустрії розваг. Від перших технічних експериментів до широкомасштабного впровадження — шлях становлення був тривалим і динамічним. Поява домашніх ігрових приставок, впровадження тривимірної графіки, а згодом — стрімкий розвиток мобільного геймінгу у 2000-х роках відкрили перед індустрією нові горизонти й надали гравцям ще більше способів взаємодії з віртуальними світами.

1.2 Еволюція ігор

У 1975 році компанія Atari представила одну з перших комерційно успішних домашніх ігрових приставок — Home Pong, яка дозволяла грати у спрощений варіант настільного тенісу на телевізорі. Цей пристрій став важливою віхою в історії відеоігор і започаткував еру домашнього геймінгу, що представлено на рисунку 1.3.

Рисунок 1.3 — Ігрова приставка Home Pong

Цей період розвитку став переломним для індустрії відеоігор, адже саме тоді цифрові розваги почали масово проникати в домівки користувачів. Уперше значна кількість людей отримала змогу грати у відеоігри вдома, що дало потужний поштовх подальшому розвитку галузі. Паралельно з цим активно з’являлися аркадні ігрові автомати, удосконалювалися ігрові консолі та створювалися дедалі доступніші пристрої для запуску ігор.

Вагому роль у формуванні ігрової індустрії відіграв Нолан Бушнелл, який, надихнувшись першими успіхами електронних ігор, заснував компанію Atari Incorporated. Одним із перших гучних успіхів компанії став аркадний автомат із грою Pong, що стрімко здобув популярність — лише на території США в цю гру зіграли мільйони користувачів.

Станом на вересень 1974 року в США налічувалося близько 100 тисяч аркадних автоматів із відеоіграми, які приносили індустрії понад 250 мільйонів доларів прибутку щороку. Ігрові зали швидко перетворилися на популярні осередки дозвілля, особливо серед молоді, та постійно приваблювали нових відвідувачів.

У 1980-х роках на ринку відеоігор з’явилися компанії, що й нині залишаються провідними гравцями галузі, зокрема Nintendo та Sega. Вони розпочали активний випуск власних ігрових консолей і великої кількості ігор до них. Завдяки доступнішій ціні та зручності використання ці пристрої швидко завоювали популярність у всьому світі. Консолі нового покоління значно перевершували попередні за технічними можливостями, дозволяючи створювати складніші ігри з

покращеною, уже кольоровою графікою замість примітивних чорно-білих зображень.

У 1992 році компанія Id Software випустила на персональних комп'ютерах гру Wolfenstein 3D, яка стала важливим кроком у розвитку тривимірних ігор та справила значний вплив на подальший розвиток жанру шутерів від першої особи (зображено на рисунку 1.4).

Рисунок 1.4 — Гра «Wolfenstein 3D»

Ця відеогра стала справжнім технологічним зламом у розвитку тривимірної графіки. Хоча з технічної точки зору вона використовувала псевдо-3D, ігровий процес сприймався користувачами як повноцінний тривимірний простір. Уже в 1993 році на основі вдосконаленого ігрового рушія була створена гра DOOM, яка здобула феноменальний успіх і суттєво вплинула на подальший розвиток індустрії, продемонструвавши, що динамічний геймплей може бути важливішим за складну сюжетну складову.

Від простих аркад із примітивною піксельною графікою до складних інтерактивних світів і концепцій на кшталт мультивсесвіту — відеоігри пройшли значний шлях еволюції з часів перших розробок компанії Atari та аркадних автоматів, що стали відправною точкою становлення ігрової індустрії. З розвитком апаратного забезпечення та програмних технологій ігрові системи дедалі більше адаптувалися до різних завдань, розширюючи можливості взаємодії користувача з віртуальним середовищем.

У 2020 році глобальна індустрія відеоігор була оцінена в неймовірні $159.3 мільярда, переважно завдяки інноваціям, які ми спостерігали у способі, яким ми граємо протягом останніх 2 десятиліть; від ігор на консолях, до широкого поширення мобільних ігор. Однак у Concept вважають, що найкраще ще попереду. Ми передбачаємо, що наступне десятиліття призведе, до виникнення, більш насичених вражень, на наступних платформах нового покоління, консолях та бізнес-моделях.

1.3 Види та жанри ігор

Жанровий поділ відеоігор слугує способом опису їхніх ключових механік та форм взаємодії з користувачем, незалежно від сюжету чи наповнення віртуального світу. На відміну від літератури або кіно, де жанр здебільшого визначається тематикою та оповіддю, у відеоіграх основну роль відіграє саме гравець і характер його дій у грі. Через це не існує єдиної загальноприйнятої системи класифікації: у різних джерелах одна й та сама гра може бути віднесена до кількох жанрів одночасно.

Окремо варто відзначити наявність змішаних або гібридних жанрів, у яких поєднуються елементи різних ігрових напрямів, що ускладнює їх чітке визначення. Попри це, розробники спираються на усталені уявлення про жанри відеоігор, оскільки вони допомагають зрозуміло представити концепцію проєкту, полегшують процес розробки та використовуються в маркетингових стратегіях для інформування потенційних гравців.

Класифікація ігор за жанрами:

— екшн;
— стратегії;
— рольові ігри;
— ігри жахів;
— раннер.

Екшн-ігри зосереджені на інтенсивному та швидкому ігровому процесі, що потребує від гравця миттєвої реакції, точності та спритності [2]. До цього жанру належать бойові ігри, шутери, аркади та інші динамічні формати. Екшн поділяється на низку піджанрів, серед яких шутери, лабіринти, платформери й файтинги, що проілюстровано на рисунку 1.5.

Рисунок 1.5 — Гра «Counter Strike 2»

Стратегії орієнтовані на розвиток логічного та стратегічного мислення, планування дій і ефективне управління ресурсами [3]. У таких іграх гравець може займатися розбудовою міст, командуванням військами або керуванням цілими державами чи цивілізаціями. Приклади ігор цього жанру наведені на рисунку 1.6.

Рисунок 1.6 — Гра «Clash of Clans»

Рольові ігри (RPG) передбачають ототожнення гравця з конкретним персонажем або керівником групи персонажів, кожен з яких має чітко визначену роль і набір умінь [4]. Наприклад, воїн не здатен виконувати функції мага, а вибір ролі безпосередньо впливає на стиль гри та розвиток подій. Основна мета полягає

у виконанні квестів, що сприяють розвитку персонажа або команди, як показано на рисунку 1.7.

Рисунок 1.7 — Гра «The Witcher 3»

Ігри жахів спрямовані на створення напруженої та тривожної атмосфери за допомогою елементів страху, містики та психологічного тиску [5]. Часто гравець змушений обирати між збереженням власного життя та дослідженням небезпечних таємниць ігрового світу. Одним із відомих представників цього жанру є гра Five nights in Freddy’s, за мотивами якої було знято фільм; приклад подано на рисунку 1.8.

Рисунок 1.8 — Гра «Five nights in Freddy’s»

Раннер — це жанр, у якому персонаж постійно рухається вперед, а основним завданням гравця є уникнення перешкод, подолання ворогів і своєчасне реагування на загрози [6]. Ключовою особливістю є безперервний рух, що вимагає концентрації та швидких рішень. Однією з найвідоміших ігор цього жанру є Subway Surfers, зображена на рисунку 1.9.

Рисунок 1.9 — Гра «Subway Surfers»

1.4 Аналіз популярних шаблонів проектування

На даний час є кілька популярних шаблонів проектування [13], а саме:

— MVC;

— MVP;

— MVVM.

Архітектура Model–View–Controller (MVC) є одним із найпоширеніших шаблонів проєктування програмного забезпечення, який широко застосовується під час створення вебзастосунків та інших програмних систем. Основною ідеєю MVC є поділ програмної логіки на три окремі компоненти: Модель (Model), Подання (View) та Контролер (Controller). Такий підхід дозволяє чітко розмежувати обов’язки між складовими, що покращує читабельність коду, його масштабованість і спрощує подальшу підтримку та тестування [7].

Модель (Model) відповідає за бізнес-логіку програми та роботу з даними. Вона здійснює збереження, оновлення та отримання інформації з різних джерел, зокрема баз даних або файлів. Модель функціонує автономно й не залежить від представлення чи контролера, забезпечуючи коректну обробку даних.

Представлення (View) призначене для відображення даних користувачеві у зручній візуальній формі. Воно не містить бізнес-логіки, а лише демонструє інформацію, отриману від моделі, та реагує на дії користувача, забезпечуючи взаємодію з системою.

Контролер (Controller) виступає координатором між моделлю та

представленням. Він приймає запити користувача, аналізує їх і визначає, які дії необхідно виконати — оновити модель або змінити подання, керуючи загальним потоком виконання програми.

Застосування MVC сприяє логічному поділу функціональності, що значно полегшує розробку, супровід та розширення програмного продукту. Окрім цього, архітектура підтримує повторне використання компонентів і підвищує гнучкість системи.

Іншим підходом до побудови програм із користувацьким інтерфейсом є архітектура Model–View–Presenter (MVP), яка має схожі риси з MVC, проте відрізняється способом взаємодії компонентів [8]. У межах MVP система також складається з Моделі (Model), Представлення (View) та Презентера (Presenter).

Модель у MVP виконує ті ж функції, що й у MVC — реалізує бізнес-логіку та працює з джерелами даних, не маючи залежностей від інших компонентів. Представлення (View) відповідає виключно за інтерфейс користувача, відображає дані та передає події до презентера, не виконуючи логічних обчислень. Презентер (Presenter) обробляє дії користувача, взаємодіє з моделлю, отримує необхідні дані та визначає, як саме вони мають бути відображені у представленні.

Серед недоліків MVP можна відзначити складність реалізації для початківців, а також збільшення кількості класів, що іноді ускладнює структуру проєкту. Водночас головна відмінність MVP від MVC полягає в активнішій ролі презентера, який повністю керує логікою взаємодії між інтерфейсом і моделлю. Це сприяє кращому контролю над поведінкою програми та спрощує модульне тестування.

Архітектура Model–View–ViewModel (MVVM) є ще одним шаблоном проєктування, орієнтованим на сучасні програмні платформи. Вона була запропонована Джоном Госсманом у 2005 році як розвиток концепції Presentation Model та активно використовується у середовищах Windows Presentation Foundation і Silverlight від Microsoft [9].

Основною метою MVVM є відокремлення логіки обробки даних від реалізації графічного інтерфейсу. ViewModel виконує роль посередника між

моделлю та представленням, перетворюючи дані у формат, зручний для відображення, та керуючи станом інтерфейсу. Завдяки цьому забезпечується чітке розмежування відповідальностей і можливість незалежного розвитку кожного компонента.

Застосування MVVM дозволяє розробникам змінювати бізнес-логіку без впливу на зовнішній вигляд програми, а дизайнерам — працювати над інтерфейсом, не заглиблюючись у програмну реалізацію. Такий підхід значно покращує командну роботу та підвищує ефективність розробки.

MVVM була створена з урахуванням проблем взаємодії між програмістами та дизайнерами, які виникали під час використання технологій на кшталт Swing або MFC. У таких середовищах створення інтерфейсу вимагало глибоких технічних знань, що ускладнювало співпрацю. MVVM усуває ці обмеження, дозволяючи кожному фахівцю зосередитися на своїй галузі, поєднуючи технічну реалізацію та зручний дизайн в єдину ефективну систему.

1.5 Огляд технологій для створення ігор

У ранні періоди розвитку ігрової індустрії створення ігор вимагало програмування на мовах низького рівня, які передбачали безпосередню взаємодію з апаратною частиною комп’ютера. Однак зі стрімким розвитком обчислювальної техніки та інформаційних технологій з’явилися сучасні мови програмування й інструменти, що суттєво спростили цей процес. У наш час розробка ігор стала значно доступнішою, ніж десять–двадцять років тому, завдяки використанню спеціалізованих середовищ та бібліотек, які приховують складні технічні деталі реалізації під конкретні апаратні платформи.

Сучасний розробник відеоігор дедалі більше нагадує митця, який зосереджується на творчій складовій та кінцевому враженні від продукту, а не на низькорівневих аспектах програмування, таких як управління пам’яттю чи оптимізація виконання інструкцій для певного процесора.

На сьогодні існує велика кількість ігрових рушіїв, як безкоштовних, так і комерційних. Найпоширенішими серед них є Unity та Unreal Engine. Для реалізації цього проєкту було обрано рушій Unity, оскільки він є безкоштовним для

навчальних і невеликих проєктів, а також має потужну й активну спільноту, що значно полегшує пошук рішень і навчальних матеріалів.

У процесі розробки в Unity використовується мова програмування C#, яка є об’єктно-орієнтованою та працює на платформі .NET. Вона характеризується суворою системою типів і високим рівнем безпеки. Порівняно з мовами на кшталт C++, C# забезпечує більший рівень абстракції, що дозволяє розробникам зосередитися на логіці гри та архітектурі застосунку, не занурюючись у складні механізми керування пам’яттю. Найбільш поширеними середовищами розробки для C# є Visual Studio від Microsoft та Rider від JetBrains, що зображено на рисунку 1.10.

Рисунок 1.10 — Логотипи Visual Studio IDE та Rider IDE

Visual Studio — це комплексне інтегроване середовище розробки програмного забезпечення, створене компанією Microsoft. Воно надає широкий набір інструментів для створення різноманітних типів програм, включаючи настільні, веб- та мобільні застосунки, а також ігрові проєкти. Visual Studio підтримує багато мов програмування, зокрема C#, C++, Visual Basic, F#, JavaScript та інші [10].

До основних переваг Visual Studio можна віднести широкий функціонал, що включає потужний редактор коду, автодоповнення, засоби налагодження, інтеграцію з системами контролю версій і тестування. Можливість розробки програм для різних операційних систем і платформ, зокрема Windows, Android, iOS, macOS та Linux. Зручну організацію командної роботи завдяки вбудованим засобам спільної розробки; велику спільноту користувачів, яка активно ділиться досвідом і допомагає у вирішенні технічних проблем.

Rider — це багатоплатформенне середовище розробки від компанії JetBrains, орієнтоване на створення програм мовою C# та іншими мовами екосистеми .NET. Воно тісно інтегрується з інструментами JetBrains, зокрема ReSharper, що значно полегшує аналіз і рефакторинг коду та підвищує ефективність роботи розробника [11].

Серед ключових переваг Rider можна виділити високу швидкодію навіть при роботі з масштабними проєктами. Розвинені інструменти статичного аналізу коду, які допомагають знаходити помилки та оптимізувати рішення. Гнучкі налаштування інтерфейсу й робочого процесу та тісну інтеграцію з іншими продуктами JetBrains.

З огляду на універсальність, багатий функціонал і широку підтримку різних платформ, для реалізації даного проєкту було обрано середовище розробки Visual Studio, яке забезпечує зручність роботи та ефективність на всіх етапах створення програмного продукту.

1.6 Основні мови програмування в розробці ігор

Вибір мови програмування для створення ігрового застосунку визначається низкою чинників, серед яких ключову роль відіграють сумісність з обраним ігровим рушієм, вимоги до швидкодії, можливість кросплатформенного запуску, а також досвід і пріоритети команди розробників. У сучасній ігровій індустрії використовується декілька мов програмування, кожна з яких має свої особливості, переваги та обмеження [12].

C++ — це об’єктно-орієнтована мова програмування, створена Б’ярном Страуструпом у 1985 році, яка широко застосовується у професійній розробці ігор. Її головною перевагою є висока продуктивність і можливість безпосереднього контролю над апаратними ресурсами та пам’яттю, що є критично важливим для складних і ресурсомістких ігрових проєктів. Саме тому C++ використовується в таких популярних ігрових рушіях, як Unreal Engine. Водночас мова характеризується складним синтаксисом і необхідністю ручного керування пам’яттю, що підвищує ризик виникнення помилок і ускладнює процес навчання.

C# — об’єктно-орієнтована мова, розроблена компанією Microsoft у 2000 році, яка набула великої популярності завдяки простоті використання та широким можливостям. Вона є основною мовою програмування для рушія Unity, що робить її одним із найпоширеніших інструментів у геймдеві. На відміну від C++, C# використовує автоматичне керування пам’яттю за допомогою механізму збору сміття, що зменшує кількість типових помилок і спрощує розробку. Зрозумілий синтаксис, підтримка багатопотоковості та велика кількість бібліотек дозволяють швидко створювати та оптимізувати ігрові застосунки.

JavaScript спочатку створювався для забезпечення інтерактивності вебсторінок, однак з часом став активно застосовуватися і в розробці браузерних ігор. Завдяки таким фреймворкам, як Phaser.js та Three.js, ця мова дозволяє створювати 2D та 3D ігри, що працюють безпосередньо у веббраузері. Хоча JavaScript поступається нативним мовам за продуктивністю, його головною перевагою є простота освоєння та висока кросплатформеність.

Python — це високорівнева мова програмування, яка вирізняється читабельністю та легкістю у вивченні. У сфері розробки ігор Python найчастіше застосовується для створення прототипів або простих проєктів із використанням бібліотек на кшталт Pygame. Вона дозволяє швидко реалізувати ідею та перевірити ігрові механіки, проте через невисоку швидкодію не підходить для створення масштабних та високопродуктивних ігор.

Java — це об’єктно-орієнтована мова, яка забезпечує кросплатформеність завдяки використанню Java Virtual Machine (JVM). Вона широко застосовується у розробці мобільних ігор для платформи Android, а також підтримується такими ігровими фреймворками, як LibGDX. Java має велику спільноту розробників і розвинену екосистему, проте її використання у високопродуктивних іграх може потребувати додаткової оптимізації.

Swift — сучасна мова програмування, створена компанією Apple для розробки застосунків під iOS та інші продукти екосистеми Apple. Вона поєднує високу швидкодію з безпечними механізмами роботи з пам’яттю та зручним синтаксисом. Swift добре підходить для створення мобільних ігор, однак її

прив’язка до платформ Apple обмежує можливості використання у багатоплатформених проєктах.

Rust — відносно нова мова програмування, яка поступово набирає популярності в ігровій індустрії, особливо для реалізації низькорівневих та системних компонентів. Вона поєднує високу продуктивність із підвищеною безпекою пам’яті завдяки сучасним механізмам контролю доступу до ресурсів. Rust дозволяє уникати багатьох критичних помилок, характерних для традиційних системних мов, однак складність синтаксису та високий поріг входу можуть ускладнювати її освоєння.

Таким чином, кожна мова програмування має власну нішу в ігровій розробці, а остаточний вибір залежить від поставлених цілей, технічних вимог та особливостей конкретного проєкту.

1.7 Опис жанру

Ранери — це різновид відеоігор, орієнтований на максимально широку аудиторію, адже їхні правила та керування є інтуїтивно зрозумілими й не потребують тривалого навчання. У більшості таких ігор ігровий процес є безкінечним: персонаж автоматично рухається вперед, а завдання гравця полягає в ухилянні від перешкод, подоланні небезпек і зборі різноманітних бонусів. Завдяки простому управлінню та коротким ігровим сесіям ранери чудово підходять для гри в дорозі, під час перерв або у будь-який вільний момент. Крім того, виробництво таких ігор зазвичай не потребує значних витрат, а більшість із них розповсюджується безкоштовно, що робить жанр ще доступнішим для масового користувача.

Поява та стрімкий розвиток раннерів на мобільних платформах тісно пов’язані з поширенням смартфонів і планшетів, а також із розвитком цифрових магазинів додатків, зокрема App Store та Google Play. Перші представники жанру мали доволі просту реалізацію, однак з часом ігри почали ускладнюватися, з’явилися нові механіки, візуальні ефекти та різноманітні ігрові елементи.

Одним із ранніх і найбільш відомих прикладів жанру є гра Temple Run, створена студією Imangi Studios (рисунок 1.11).

Рисунок 1.11 — Гра "Temple Run"

Випущена у 2011 році, вона швидко здобула популярність завдяки легкому керуванню, динамічному ігровому процесу та привабливому візуальному стилю. У грі користувач керує персонажем, який тікає крізь стародавні храмові руїни, уникаючи пасток і збираючи монети.

Ще одним знаковим представником жанру став Subway Surfers, розроблений компаніями Kiloo та SYBO Games і випущений у 2012 році (рисунок 1.12).

Рисунок 1.12 — Гра " Subway Surfers "

Це також нескінченний раннер, у якому гравець рухається залізничними коліями, ухиляючись від потягів і перешкод. Завдяки яскравій графіці, регулярним оновленням з новими локаціями та захоплюючій, але простій механіці, гра швидко увійшла до числа найпопулярніших мобільних проєктів.

Для отримання прибутку розробники раннерів застосовують різні моделі монетизації. Однією з найпоширеніших є внутрішньоігрові покупки (In-App Purchases), які дозволяють придбати ігрову валюту, нових персонажів або предмети, що спрощують проходження чи пришвидшують прогрес. Часто гравці, прагнучи швидшого розвитку в грі, погоджуються на такі витрати.

Ще одним важливим джерелом доходу є реклама, яка вбудовується у безкоштовні ігри у вигляді банерів, відеороликів або інтерактивних оголошень, що з’являються під час пауз або між ігровими сесіями. Проте надмірна кількість рекламних вставок може негативно впливати на досвід користувачів і зменшувати аудиторію.

Окремо варто відзначити модель реклами з винагородою, за якої гравець добровільно переглядає рекламний контент в обмін на бонуси, додаткові життя або ігрову валюту. Такий підхід часто сприймається позитивніше, оскільки надає користувачеві вибір.

Серед найвідоміших ігор жанру раннерів можна назвати Temple Run, Subway Surfers, Sonic Dash, Jetpack Joyride та Minion Rush, які й сьогодні залишаються популярними серед гравців у всьому світі.

1.8 Ігровий рушій Unity

Unity — це сучасний ігровий рушій, орієнтований на створення багатоплатформених застосунків, розроблений компанією Unity Technologies. Програмні продукти, створені за його допомогою, можуть працювати як у двовимірному, так і в тривимірному середовищі, а також підтримують технології віртуальної реальності [13].

Основною мовою програмування в Unity є C#. Раніше рушій також підтримував мови Boo та JavaScript, однак з часом їх використання було припинено на користь єдиної мови, що спростило підтримку та розвиток платформи.

Робоче середовище Unity складається з набору функціональних вікон. У стандартному інтерфейсі користувач має доступ до панелі ресурсів проєкту, інспектора для налаштування властивостей об’єктів, вікна сцени з різними

режимами перегляду, а також ієрархії, що відображає структуру поточної сцени, як показано на рисунку 1.13.

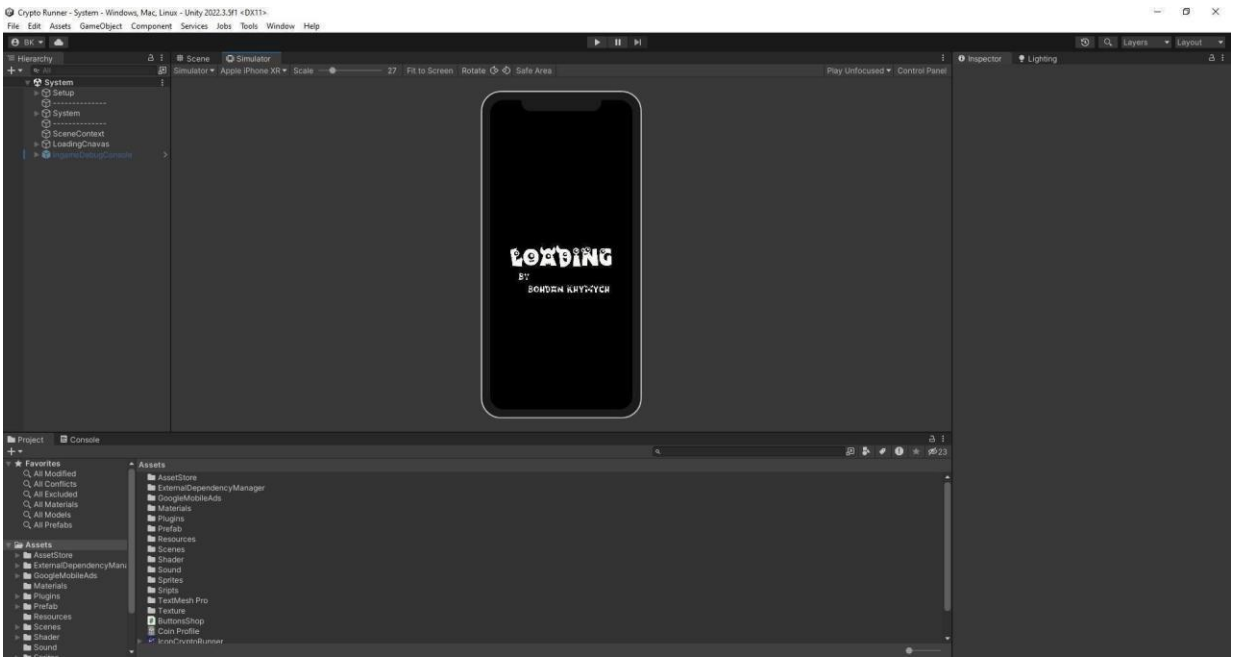

Рисунок 1.13 — Приклад інтерфейсу рушія Unity

Проєкти в Unity організовуються у вигляді сцен — окремих файлів, які містять дані про певний рівень або логічний етап гри. Сцени створюються розробником вручну та можуть включати різноманітні об’єкти: двовимірні спрайти, тривимірні моделі або порожні елементи, до яких додаються скрипти. Такі об’єкти можуть виконувати функції контролерів, тригерів, точок збереження чи переходів між сценами.

Розробник має можливість змінювати положення, масштаб і обертання об’єктів за допомогою спеціальних інструментів або через параметри в інспекторі. Для зручної організації елементів сцени передбачено систему тегів і шарів. Рушій підтримує фізику твердих тіл і тканин, дозволяючи налаштовувати взаємодію об’єктів, гравітацію та інші фізичні параметри як через скрипти, так і через глобальні налаштування проєкту.

Відображення сцени здійснюється за допомогою віртуальних камер. У межах однієї сцени може використовуватись кілька камер, положення та поведінка яких контролюються розробником. Камери можуть працювати як у перспективному, так

і в ортографічному режимі, забезпечуючи переміщення об’єктів у глибину сцени [14].

Графічна підсистема Unity використовує DirectX на платформі Windows, OpenGL на Windows, macOS та Linux, а також OpenGL ES на мобільних пристроях. Для створення візуальних ефектів застосовується мова ShaderLab, що підтримує шейдери, написані з використанням GLSL і Cg. Рушій дозволяє створювати власні скрипти, які додаються до об’єктів у вигляді компонентів і розширюють їх функціональність.

Unity здобув широку популярність завдяки зручному візуальному редактору та потужній мультиплатформенній підтримці. Це дозволяє розробляти ігрові проєкти для різних операційних систем і пристроїв, використовуючи єдину кодову базу та набір інструментів.

1.9 Платформи GooglePlay та AppStore

Розміщення мобільного застосунку в магазинах Google Play та Apple App Store є завершальним і водночас одним із найважливіших етапів розробки, адже саме він забезпечує доступ користувачів до продукту. Процедура публікації складається з кількох послідовних кроків, кожен із яких спрямований на перевірку якості додатку та його відповідність правилам конкретної платформи.

Першим етапом є створення облікового запису розробника. Для розміщення застосунків у Google Play необхідно зареєструвати обліковий запис розробника, сплативши одноразовий внесок у розмірі 25 доларів США. Після цього надається доступ до сервісу Google Play Console, який використовується для керування додатками. Для публікації в Apple App Store потрібно приєднатися до програми Apple Developer Program, вартість якої становить 99 доларів на рік. Це відкриває доступ до Apple Developer Account та платформи App Store Connect.

Наступний крок — підготовка самого додатку до публікації. Для Android необхідно сформувати файл формату APK або AAB, а для iOS — IPA. Окрім цього, розробник має підготувати візуальні матеріали: іконку застосунку, стартовий екран, скріншоти інтерфейсу та, за потреби, промо-відео. Також важливим є створення змістовного й зрозумілого опису додатку для користувачів.

Для завантаження додатку в Google Play потрібно створити новий проєкт у Google Play Console, додати файл збірки та заповнити інформацію про застосунок: опис, ключові слова, категорію, контактні дані та посилання на політику конфіденційності. Далі завантажуються графічні матеріали, обираються регіони поширення й задається цінова модель. Після цього заповнюється форма вікового рейтингу, і додаток надсилається на модерацію. Після проходження перевірки застосунок стає доступним для користувачів.

Публікація в Apple App Store передбачає використання середовища Xcode. Спочатку необхідно авторизуватися в Apple Developer Account, створити архів застосунку та завантажити його в App Store Connect. Після цього заповнюється сторінка додатку: додається опис, графічні матеріали, обираються країни розповсюдження, встановлюється ціна та вказується посилання на політику конфіденційності. Також проходиться процедура присвоєння вікового рейтингу. Перевірка з боку Apple може тривати кілька днів, після чого, у разі успіху, додаток публікується в магазині.

Після релізу важливо підтримувати застосунок у актуальному стані. Це включає аналіз відгуків користувачів, роботу з коментарями, усунення виявлених помилок і додавання нового функціоналу. Оновлення публікуються за процедурою, схожою на первинне розміщення, із завантаженням нової версії та описом змін у Google Play Console або App Store Connect.

Отже, процес публікації мобільного застосунку в Google Play та Apple App Store охоплює реєстрацію розробника, підготовку технічних і візуальних матеріалів, налаштування сторінки додатку, проходження модерації та подальшу підтримку. Дотримання всіх етапів дозволяє успішно представити продукт широкій аудиторії та забезпечити стабільну роботу застосунку.

1.10 Технології для реалізації застосунка

Для створення гри «3D Crypto Runner Go» застосовано низку спеціалізованих фреймворків та бібліотек, що дозволяють реалізувати комплексну функціональність проєкту та забезпечити високий рівень продуктивності та зручності розробки.

DOTween — це популярний інструмент для створення анімацій в Unity [18], який використовується як альтернатива стандартному аніматору, особливо в проєктах з 2D-графікою. Головна перевага DOTween полягає в простоті використання та високій продуктивності, що дозволяє розробникам створювати плавні та складні анімації без додаткових витрат часу. На відміну від вбудованого аніматора Unity, DOTween не вимагає створення окремих анімаційних кліпів або контролерів. Цей інструмент дозволяє анімувати будь-які параметри об’єктів — позицію, масштаб, обертання, прозорість, колір та інші властивості — безпосередньо через код.

Крім того, DOTween надає можливість створювати послідовності анімацій, повторювані цикли та ефекти затримки, що робить його надзвичайно гнучким і зручним для різних сценаріїв ігрової логіки. Інтеграція з редактором Unity дозволяє розробникам переглядати та редагувати анімації в реальному часі, що прискорює процес розробки та спрощує налагодження.

Zenject (Extenject) — це фреймворк для Unity, який реалізує патерн впровадження залежностей (Dependency Injection, DI) [19]. Основною складовою Zenject є DIContainer, який дозволяє легко налаштовувати залежності між об’єктами, знижуючи зв’язність коду та полегшуючи додавання нового функціоналу.

Завдяки Zenject розробникам легше тестувати окремі компоненти та підтримувати проєкт у модульному стані, що особливо важливо для великих ігрових проєктів з великою кількістю об’єктів та механік [20]. Завантажити фреймворк можна через офіційний репозиторій на GitHub або через Unity Asset Store [23], що забезпечує швидку інтеграцію у проєкт (рисунок 1.14).

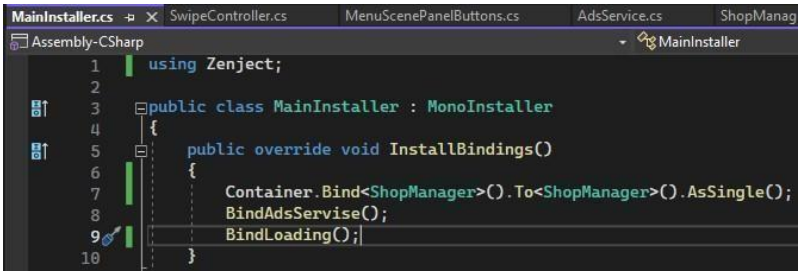

Рисунок 1.14 — Приклад коду з використанням Zenject

Cinemachine — це потужний інструмент для управління камерами в Unity, що дозволяє створювати динамічні та складні рухи камери без необхідності програмувати кожен аспект вручну [21]. Основні можливості включають:

— використання віртуальних камер для створення кількох режимів спостереження та легкого перемикання між ними;

— плавне слідкування за персонажем або об’єктами, забезпечуючи більш природну картину руху;

— автоматичне кадрування для оптимального положення камери та кращого огляду сцени;

— встановлення обмежень руху камери для запобігання виходу за межі сцени;

— додавання ефектів шуму та тряски, що імітують фізичні динамічні явища.

Cinemachine також інтегрується з Unity Timeline та Post-Processing Stack, що дозволяє створювати кінематографічні ефекти та покращує загальну якість візуальної подачі гри. Завдяки цьому інструменту розробники можуть легко організовувати камери навіть у складних проєктах з багатьма локаціями та сценаріями (рисунок 1.15).

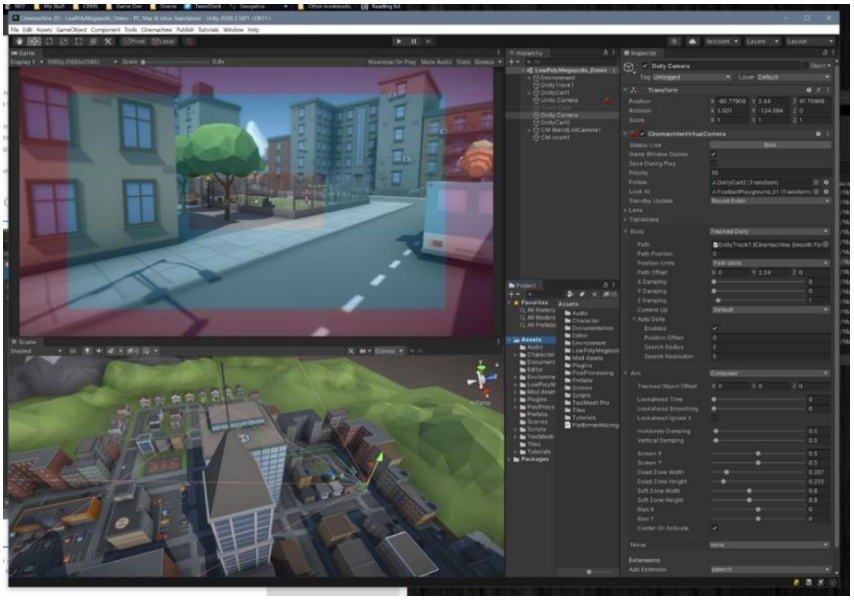

Рисунок 1.15 — Зображення вікна Cinemachine

UniTask — це асинхронна бібліотека, яка замінює стандартні корутини Unity, пропонуючи більш ефективний і зручний спосіб роботи з асинхронним кодом [22]. На відміну від Coroutine, UniTask дозволяє використовувати звичайні методи C# для виконання асинхронних операцій без створення додаткових MonoBehaviour-об’єктів. Переваги UniTask включають:

— більш зручний та читабельний синтаксис для асинхронного програмування;

— підвищену продуктивність завдяки зменшенню витрат системних ресурсів;

— ефективне управління пам’яттю без генерації непотрібних об’єктів;

— підтримку стандартних асинхронних операцій, таких як запити до HTTP, очікування таймерів, робота з файловою системою та інші.

Використання DOTween, Zenject, Cinemachine та UniTask у проєкті «3D Crypto Runner Go» забезпечує високий рівень контролю над ігровими механіками, анімаціями, камерами та асинхронними процесами. Ці інструменти дозволяють створювати плавні та інтерактивні ігрові сцени, оптимізувати ресурси, підвищувати продуктивність і спрощують подальший розвиток та підтримку гри.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ДИЗАЙНУ ГРИ

2.1 Архітектура додатку

Структура додатку представлена на рисунку 2.1.

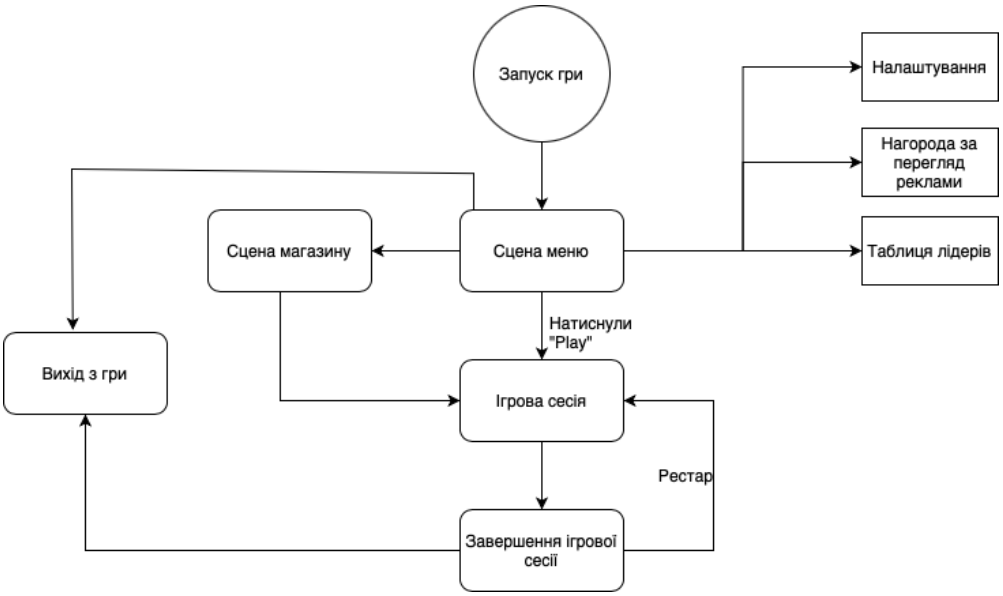

Рисунок 2.1 — Графічне представлення структури гри

Життєвий цикл ігрового додатку демонструє послідовність дій від моменту запуску до завершення сесії та включає проміжні етапи взаємодії користувача з грою. При вході в додаток спочатку завантажується спеціальна сцена завантаження, яка відповідає за ініціалізацію основних систем гри та підготовку ресурсів. Після завершення цього процесу відкривається головна ігрова сцена.

На початковому етапі користувач бачить інтерфейс головного меню, яке містить п’ять основних кнопок для взаємодії з грою. Натискання цих кнопок дозволяє відкривати різні UI-вікна: меню налаштувань, таблицю лідерів, систему отримання бонусів за перегляд реклами та магазин, де можна обрати персонажів із різними зовнішніми образами. П’ята кнопка відповідає за запуск ігрової сесії, під час якої гравець безпосередньо керує персонажем.

У ході ігрової сесії персонаж рухається постійно вперед, а завдання гравця полягає в тому, щоб уникати перешкод та збирати різноманітні бонуси. Серед них монети, що виступають внутрішньою валютою гри, та прискорення, яке тимчасово подвоює швидкість персонажа. Перешкоди та бонуси генеруються динамічно, а

рівень складності підвищується разом із прогресом гравця, а швидкість персонажа поступово зростає. Основною метою гравця є накопичення максимальної кількості монет та встановлення або побиття власного рекорду за очками. У разі зіткнення персонажа з перешкодою сесія завершується, а результати автоматично зберігаються.

Щоб підвищити залучення користувачів, гра пропонує інтуїтивно зрозуміле управління, яскраву графіку та захопливий геймплей. Наявність різноманітних персонажів у магазині дає гравцям змогу обирати вподобані образи, що стимулює інтерес до гри. Крім того, система нагород за перегляд рекламних оголошень підтримує монетизацію додатку та забезпечує користувачам додаткові бонуси, що робить ігровий процес більш привабливим і динамічним.

2.2 Інтерфейс

Інтерфейс у будь-якій грі відіграє важливу роль, оскільки він забезпечує гравця необхідною інформацією для розуміння того, що відбувається на екрані, та допомагає правильно реагувати на події гри. Деякі ігри намагаються відображати якомога більше візуальних елементів, щоб дати користувачеві повний контроль, тоді як інші прагнуть зменшити кількість інформації, залишаючи лише найважливіше, щоб не перевантажувати гравця [17].

У грі «3D Crypto Runner GO» інтерфейс спроектовано так, щоб обмежити гравця лише необхідними елементами. Першим, що бачить користувач, є головне меню. На цьому екрані розташовані панелі, що показують рекорд гравця та кількість монет (ігрової валюти), назва гри, анімований персонаж та п’ять основних кнопок: «Play» — для початку ігрової сесії, «Settings» — для налаштувань, «Ads Reward» — для отримання бонусів за перегляд реклами, «Leaderboard» — для відкриття таблиці лідерів, «Shop» — для переходу в магазин ігрових персонажів.

Більшість кнопок розташовані у нижній частині екрану, що забезпечує зручність і звичність управління, а також легкий доступ до основних функцій гри, як показано на рисунку 2.2. Такий підхід дозволяє користувачеві швидко

орієнтуватися в меню і зосередитися на ігровому процесі.

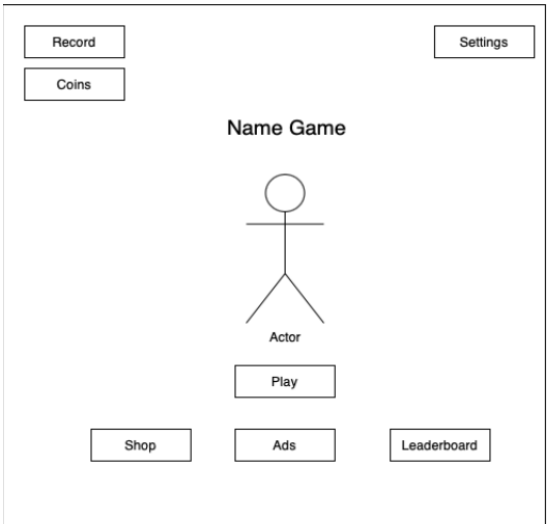

Рисунок 2.2 — Схема головного екрана

Порядок взаємодії користувача з інтерфейсом наступний. Після натискання кнопки «Settings» у головному меню відкривається вікно налаштувань, яке зображене на рисунку 2.3.

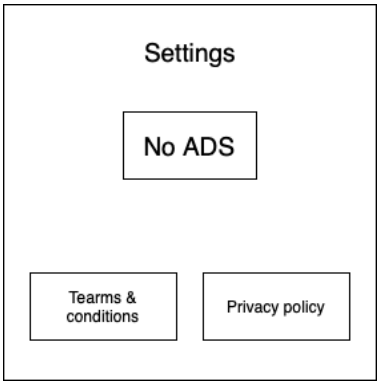

Рисунок 2.3 — Схема налаштування в головному меню

Це вікно включає одну кнопку «No ads» та два посилання: на правила і умови використання, а також на політику конфіденційності. Дані посилання необхідні для того, щоб при публікації додатку на мобільні платформи команда перевірки не відхилила його через відсутність цієї інформації. Натискання кнопки «No ads» призводить до відкриття екрана з підтвердженням відключення реклами за реальні кошти.

Якщо користувач обирає кнопку «Ads Reward» у головному меню для отримання бонусів за перегляд реклами, з’являється вікно підтвердження, яке

запитує, чи бажає він переглянути рекламний ролик. Відсутність цього екрана може призвести до повернення додатку службою перевірки, як показано на рисунку 2.4.

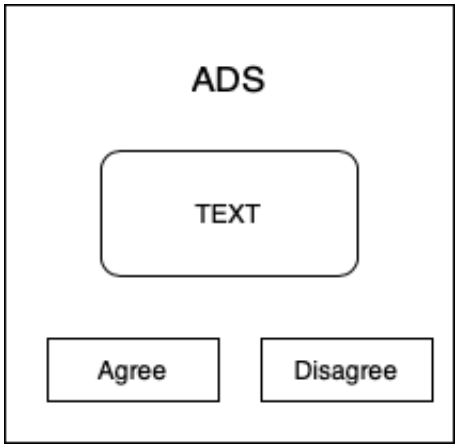

Рисунок 2.4 — Схема показу попереджувального вікна перед переглядом рекламного ролика

Якщо користувач відмовляється від пропозиції перегляду реклами, відповідне вікно закривається. Проте при кожній наступній спробі перегляду рекламного ролика, воно знову з’являється, доки користувач не погодиться з правилами. Після підтвердження правил це вікно більше не відображається, і користувача одразу перенаправляють до перегляду реклами.

Натискання кнопки «Shop» у головному меню переносить користувача на екран магазину, як показано на рисунку 2.5.

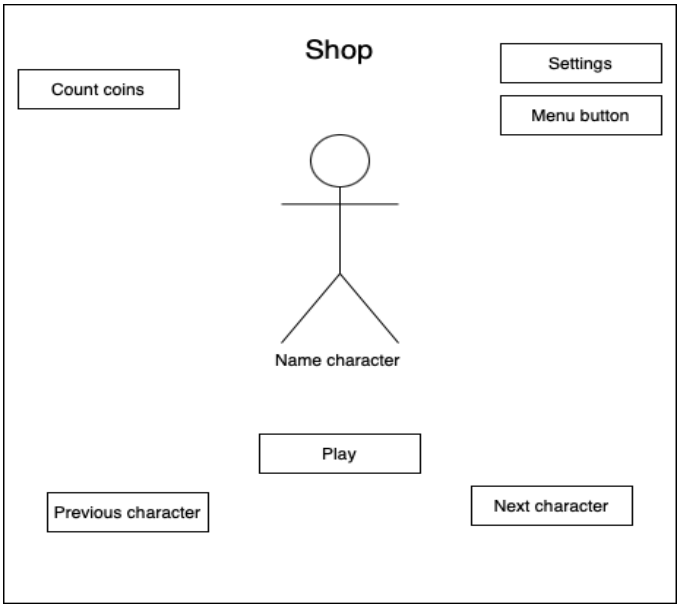

Рисунок 2.5 — Схема демонстрації інтерфейсу екрана магазину

На цьому екрані по центру розташований ігровий персонаж, а за допомогою навігації можна переглядати доступних персонажів та купувати їх. Для покупки можна використати ігрові монети або придбати персонажа за реальні гроші.

Якщо у магазині натиснути кнопку «Settings», відкривається вікно з кнопкою «Restore Purchases» та двома посиланнями, що зображено на рисунку 2.6.

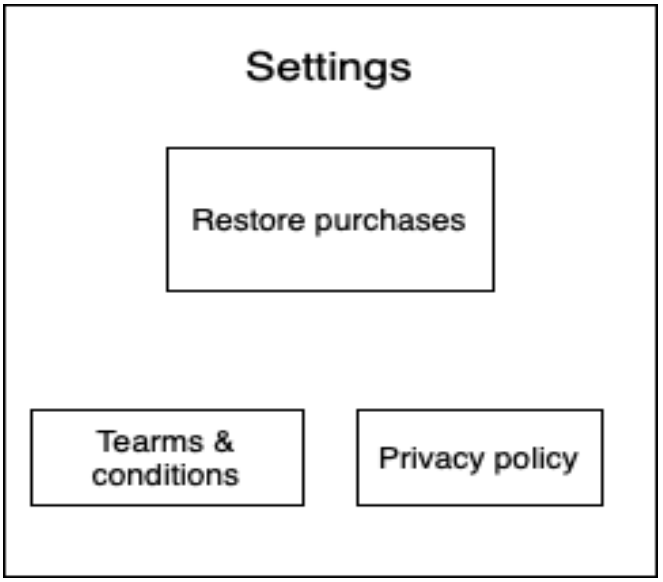

Рисунок 2.6 — Схема демонстрації інтерфейсу налаштувань у магазині

Кнопка «Restore Purchases» дозволяє користувачу відновити покупки, здійснені за реальні гроші, у випадку повторного встановлення додатку, повертаючи всі раніше придбані предмети.

Інші елементи інтерфейсу магазину включають відображення кількості монет, щоб користувач завжди бачив свій баланс, який автоматично оновлюється після будь-якої покупки. Крім того, присутні кнопки для повернення до головного меню та для запуску ігрової сесії.

На рисунку 2.7 показаний екран самої ігрової сесії. Тут відображаються панелі з інформацією про набрані очки, кількість зібраних монет, а також кнопка «Pause», яка дозволяє призупинити гру. Користувач керує персонажем за допомогою жестів пальцем по екрану: вправо, вліво, вгору та вниз.

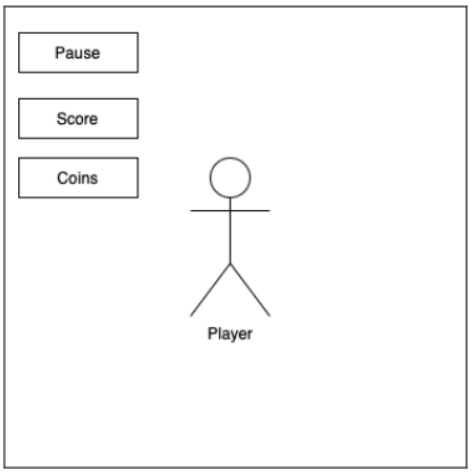

Рисунок 2.7 — Схема відображення інтерфейсу під час ігрової сесії

При натисканні кнопки «Pause» з’являється екран паузи (рисунок 2.8).

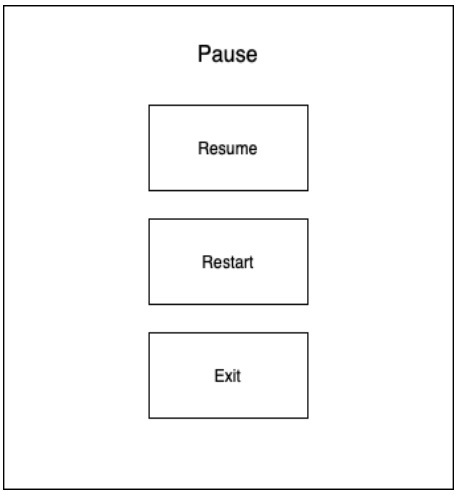

Рисунок 2.8 — Схема відображення екрану паузи

Тут показано три кнопки: “Resume”, “Restart” та “Exit”. Кнопка «Resume» закриває вікно паузи і продовжує поточну ігрову сесію. «Restart» починає нову сесію, при цьому зібрані монети за попередній сеанс зберігаються, а очки не зберігаються навіть якщо вони перевищують попередній рекорд. Кнопка «Exit» повертає користувача до головного меню.

Після завершення ігрової сесії, коли персонаж зіткнеться з перешкодою, відображається екран кінця гри. На ньому користувачу повідомляють про закінчення гри та пропонують дві кнопки: «Restart» для негайного початку нової сесії або «Exit» для повернення до головного меню.

3 РЕАЛІЗАЦІЯ ГРИ “CRYPTO RUNNER GO”

3.1 Використання Zenject

У процесі розробки гри Zenject застосовується здебільшого для організації зв’язків між об’єктами через DIContainer. Іншими словами, Zenject використовується для управління залежностями між класами, що дозволяє легко замінювати конкретні реалізації без потреби змінювати код у всіх скриптах, які залежать від цих класів.

Зазвичай зв’язування здійснюється через інтерфейси, що надає можливість підміняти реалізацію класу без шкоди для решти системи. Найпоширеніший спосіб визначення залежностей — це через конструктор класу. Проте існують ситуації, коли залежності можна передавати як поля класу з атрибутом [Inject] або як параметри методів, теж позначені [Inject]. Ці методи найчастіше застосовуються для класів, що наслідуються від MonoBehaviour, оскільки Unity не дозволяє безпосередньо змінювати чи викликати конструктор таких класів.

Наприклад, для прив’язки сервісу реклами до інтерфейсу можна використати такий підхід: клас AdsService зв’язується з інтерфейсом IAdsService. Це дозволяє Zenject створювати єдиний екземпляр сервісу, який буде використовуватись у всьому проєкті, забезпечуючи централізоване управління та контроль залежностей.

Наступний код дозволяє реалізувати прив’язку класу AdsService до інтерфейсу IAdsService:

Лістинг 3.1 — реалізація класу AdsService

```
private void AdsService ()
{
    Container.Bind<IPurchaseService>().To<PurchaseService>().AsSingle();
}
```

У наведеному прикладі клас AdsService прив’язується до інтерфейсу IAdsService як єдиний екземпляр за допомогою методу AsSingle(). Це забезпечує створення одного спільного об’єкта сервісу для всього проєкту, що дозволяє уникнути дублювання та централізовано керувати рекламою.

Водночас слід зазначити, що це лише один із можливих способів організації прив’язки класів у DIContainer, і Zenject дозволяє реалізовувати й інші схеми, наприклад Transient для створення нового екземпляру при кожному запиті або Cached для тимчасового кешування.

У проєкті створено клас EntryPoint, який відповідає за ініціалізацію основних сервісів гри та використовує Zenject для управління залежностями. Розглянемо більш детально функції класу:

Лістинг 3.2 — реалізація класу EntryPoint

```
public class EntryPoint : MonoBehaviour
{
   [SerializeField] private AsyncLoadingSystem _asyncLoadingSystem;
   private IRewardService _rewardService;
   private IShopService _shopService;

   [Inject]
   public void Construct(IRewardService rewardService, IShopService shopService)
   {
      _rewardService = rewardService;
      _shopService = shopService;
   }
}
```

Це конструктор класу EntryPoint, назва якого вибрана за домовленістю розробників, щоб було зрозуміло, що даний клас є нащадком MonoBehaviour і слугує точкою входу для ініціалізації ключових компонентів гри.

Лістинг 3.3 — Конструктор класу EntryPoint

```
private async void Awake()
{
```

Продовження лістингу 3.3

```
await InitializeSystems();
}
```

Це перевизначена функція класу MonoBehaviour, яка запускається на початку гри.

Лістинг 3.4 — Перевизначена класу MonoBehaviour

```
private async UniTask InitializeModules()
{
   await _iAPService.SetupPurchasesAsync();
   _adsService.InitializeAds();
   _loadingSystemAsync.StartLoading();
}
```

Цей метод відповідає за збереження посилань на сервіси внутрішніх покупок, реклами та екран завантаження. У класі використовується атрибут [Inject], який дозволяє Zenject автоматично підставити необхідні залежності IAdsService та IIAPService у метод Construct(). Після цього у методі Start() відбувається ініціалізація всіх необхідних сервісів для коректної роботи ігрового додатку.

3.2 Використання інструменту DOTween для Popup панелей

Під час розробки універсального механізму анімації для більшості панелей у грі було використано бібліотеку DOTween. Зазвичай анімаційні ефекти прописуються окремо для кожної панелі, що створює зайву дублюваність коду. У нашому випадку створено універсальний клас, який можна застосовувати до більшості панелей у грі.

У проекті реалізовано клас CanvasGroupFadingAnimator, який відповідає за анімацію зміни прозорості групи елементів інтерфейсу.

Клас включає наступні методи:

— AnimateShowing(Action onComplete) — відповідає за плавне появлення панелі;

— AnimateHiding(Action onComplete) — забезпечує плавне зникнення

панелі.

Крім того, клас містить такі властивості:

— CanvasGroup canvasGroup — зберігає посилання на CanvasGroup, що використовується для керування прозорістю UI-елементів;

— float startOpaqueValue — початкове значення прозорості при анімації появи;

— float endOpaqueValue — кінцеве значення прозорості для анімації показу;

— float appearTime — час, за який прозорість досягає кінцевого значення під час появи;

— loat fadeTime — час, за який панель повністю зникає при анімації приховування.

Розглянемо більш детально функції класу, де реалізована анімація альфа-каналу для UI об’єктів:

Лістинг 3.5 — Реалізація анімації альфа-каналу для UI об’єктів

```
public class CanvasOpacityAnimator : BaseViewAnimator
{
   [SerializeField] private CanvasGroup uiGroup;
   [SerializeField, Range(0f, 1f)]
   private float initialAlpha = 0f;
   [SerializeField, Range(0f, 1f)]
   private float targetAlpha = 0.6f;
   [SerializeField]
   private float showDuration = 0.5f;
   [SerializeField]
   private float hideDuration = 0.3f;
   public override void AnimateShowing(Action onAnimationFinished)
   {
```

Продовження лістингу 3.5

```
uiGroup.alpha = initialAlpha;
        uiGroup
            .DOFade(targetAlpha, showDuration)
            .SetEase(Ease.Linear)
            .SetId(transform)
            .OnComplete(() =>
            {
                onAnimationFinished?.Invoke();
            });
    }
}
```

Це метод анімації показу панелі. Встановлюється прозорість UI елемента, який потрібно анімувати.

Лістинг 3.6 — Реалізація методу анімації показу панелі

```
public override void AnimateHiding(Action onFinished)
    {
        uiCanvasGroup
            .DOFade(hiddenAlpha, hideDuration)
            .SetEase(Ease.Linear)
            .SetId(transform)
            .OnComplete(() =>
            {
                onFinished?.Invoke();
            });
    }
}
```

Щоб завершити налаштування анімації об’єктів, необхідно додати відповідний скрипт на об’єкт у Unity, який підлягає анімації. Після цього слід заповнити порожні поля у скрипті, перетягнувши туди потрібні компоненти, а також налаштувати інші параметри відповідно до потреб проєкту. Приклад такої конфігурації наведено на рисунку 3.1.

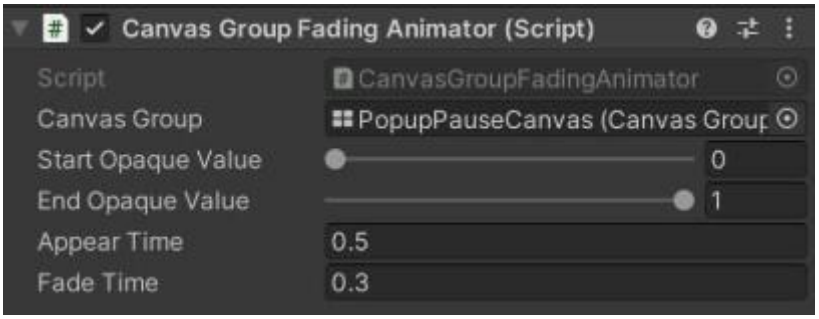

Рисунок 3.1 — Базовий клас для анімацій панелей

3.3 Реалізація керуванням персонажа

У програмі створено клас SwipeController, який відповідає за визначення рухів за допомогою свайпів на екрані. Цей клас містить метод Update, який викликається кожен кадр гри для оновлення стану об'єкту та властивостей bool tap, swipeLeft, swipeRight, swipeUp, swipeDown, які вказують на різні жести користувача: натискання або свайп вниз, вгору, вліво або вправо відповідно, а також bool isDraging - відслідковуємо, чи відбулась зміна напрямку персонажа, та startTouch — вектор, що зберігає позицію початкового торкання пальця або курсора на екрані, swipeDelta — зберігає різницю між поточною позицією пальця або курсора і початковою позицією під час торкання. Використовується для визначення величини свайпу та його напрямку. В даному класі метод “Update”, тому більш детальніше розгялнемо код:

Лістинг 3.7 — Реалізація класу SwipeController

```
public class SwipeController : MonoBehaviour
{
   public static bool isTap, moveLeft, moveRight, moveUp, moveDown;
   private bool isTracking = false;
```

Продовження лістингу 3.7

```
private Vector2 touchStartPosition;
    private Vector2 touchOffset;
    private void Update()
    {
        isTap = moveLeft = moveRight = moveUp = moveDown = false;
        if (Input.touches.Length > 0)
        {
            Touch currentTouch = Input.touches[0];
            if (currentTouch.phase == TouchPhase.Began)
            {
                isTap = true;
                isTracking = true;
                touchStartPosition = currentTouch.position;
            }
            else if (currentTouch.phase == TouchPhase.Ended || currentTouch.phase == 
TouchPhase.Canceled)
            {
                isTracking = false;
                ClearSwipeData();
            }
        }
    }
```

Даний оператор допомагає перевіряти, чи були дотики на екрані.

Лістинг 3.8 — Реалізація оператора перевірки дотика на екран

```
gestureOffset = Vector2.zero;
if (isTouching)
{
    if (Input.touches.Length > 0)
```

Продовження лістингу 3.8

```
        gestureOffset = Input.touches[0].position - touchStartPosition;
    }
    else if (Input.GetMouseButton(0))
    {
        gestureOffset = (Vector2)Input.mousePosition - touchStartPosition;
    }
}
```

Даний оператор допомагає визначити зміщення свайпу.

Лістинг 3.9 — Реалізація оператора визначення зміщення свайпу

```
if (gestureVector.magnitude > minSwipeDistance)
{ float horizontal = gestureVector.x;
    float vertical = gestureVector.y;
    if (Mathf.Abs(horizontal) > Mathf.Abs(vertical))
    {
        if (horizontal < 0)
            moveLeft = true;
        else
            moveRight = true;
    }
    else
    {
        if (vertical < 0)
            moveDown = true;
        else
            moveUp = true;
```

Цей оператор відповідає за визначення напрямку свайпу, якщо його довжина перевищує заданий мінімальний поріг.

Такий підхід забезпечує інтуїтивне та зручне управління для гравців, підвищує комфорт взаємодії з грою та дозволяє реалізовувати різноманітні інтерактивні елементи й функції.

3.4 Реалізація ініціалізації платформ

Під час розробки системи генерації платформ була обрана механіка, за якою персонаж залишається на місці, а платформи рухаються до нього. Такий підхід обрано через складності точного управління глибиною об'єктів у тривимірному просторі, особливо при роботі з типом даних float.

У деяких випадках контроль координати Z може викликати неточності, тому зручніше фіксувати положення гравця, а рух здійснювати за допомогою об’єктів сцени. Це спрощує логіку руху та взаємодії з перешкодами, а також дозволяє уникнути проблем із точністю позиціювання об’єктів.

Система руху платформ реалізована через клас TileGenerator, який відповідає за створення нових плиток та видалення старих. Клас містить методи SpawnTile() та AnimateHiding(Action onComplete), а також властивості:

— List<GameObject> _activeTiles — список активних платформ на сцені;

— int _currentTileIndex — індекс поточної платформи;

— GameObject[] _tilePrefab — масив префабів для генерації;

— float _tileLength — довжина кожної плитки;

— методи SpawnTile() та DeleteTile() для керування платформами.

Ця реалізація дозволяє динамічно створювати нові платформи та видаляти непотрібні, забезпечуючи безперервний ігровий процес та плавний рух світу до персонажа:

Лістинг 3.10 — Реалізація оператора перевірки дотика на екран

```
private void SpawnPlatform()
    int prefabIndex = _currentPlatformIndex % _platformPrefabs.Length;
    GameObject newPlatform = Instantiate(
```

Продовження лістингу 3.10

```
        _platformPrefabs[prefabIndex],
        new Vector3(0, 0, _platformLength * _currentPlatformIndex),
        Quaternion.identity
    );
    if (_activePlatforms.Count > 0)
    {
        Vector3 lastPlatformPos = _activePlatforms[_activePlatforms.Count -
1].transform.position;
        newPlatform.transform.position = new Vector3(0, 0, lastPlatformPos.z +
_platformLength);
    }
    _activePlatforms.Add(newPlatform);
    _currentPlatformIndex++;
  }
}\
```

Цей метод реалізує генерацію нової плитки на сцені. Він визначає індекс потрібного префаба з масиву _tilePrefabs, створює наступну плитку, встановлює її позицію на сцені та додає до списку активних платформ _activeTiles.

Лістинг 3.11 — Реалізація метоу генерації нової плитки на сцені

```
    private void RemovePlatform()
    {
        if (_activePlatforms.Count > 0)
        {
            Destroy(_activePlatforms[0]);
            _activePlatforms.RemoveAt(0);
        }
```

Цей метод видаляє найстарішу платформу зі списку активних плиток, якщо

така існує. Це дозволяє підтримувати стабільну кількість платформ на сцені та запобігає їх надмірному накопиченню.

Завдяки цим методам можна ефективно управляти платформами на сцені: додавати нові плитки у міру необхідності та видаляти ті, що вже не використовуються, що сприяє оптимізації використання ресурсів гри.

3.5 Інтегрування реклами

Для реалізації реклами, яка показується після завершення ігрової сесії або за перегляд за винагороду, у проєкті була використана бібліотека Google Mobile Ads [25]. Було створено клас AdManager, який відповідає за завантаження та відображення реклами.

У цьому класі реалізовано методи:

— LoadInterstitialAd — для підвантаження міжсторінкової реклами;

— ShowInterstitialAd — для її показу гравцю.

Клас також містить властивості та регіони препроцесора #if, #elif, #else, #endif, завдяки яким код виконується лише на тих платформах, які вказані після відповідного регіону. Для кожного регіону передбачено власний унікальний ідентифікатор під назвою adUnitId та об’єкт InterstitialAd _interstitialAd, який представляє міжсторінкову рекламу.

Таким чином, AdManager дозволяє ефективно інтегрувати рекламу в гру, контролювати її показ на різних платформах та забезпечувати монетизацію ігрового процесу.

Лістинг 3.12 — Реалізація методу генерації нової плитки на сцені

```
#if UNITY_ANDROID

private string
_interstitialAdUnitId = "ca-
app-pub-
3940256099942544/10331737

12";
```

Продовження лістингу 3.12

```
#elif UNITY_IPHONE

private string
_interstitialAdUnitId = "ca-
app-pub-
3940256099942544/44114689
10";

#else

private string
_interstitialAdUnitId =
"unused";

#endif
```

Визначення рекламного ідентифікатора в залежності від платформи.

Лістинг 3.13 — Реалізація рекламного ідентифікатора

```
public void LoadInterstitialAd()
{
    if (_currentInterstitialAd != null)
    {
        _currentInterstitialAd.Destroy();
        _currentInterstitialAd = null;
    Debug.Log("Loading the interstitial ad.");
    var adRequest = new AdRequest();
    InterstitialAd.Load(_interstitialAdUnitId, adRequest, (InterstitialAd ad,
LoadAdError error) =>
    {
        if (error != null || ad == null)
Debug.LogError("Interstitial ad failed to load with error: " + error);
```

Продовження лістингу 3.13

```
            return;
        }Debug.Log("Interstitial ad loaded successfully: " + ad.GetResponseInfo());
        _currentInterstitialAd = ad;
    });
```

Цей метод призначений для підвантаження міжсторінкової реклами. Він формує запит на отримання реклами, виконує її завантаження за визначеним ідентифікатором adUnitId та контролює результат завантаження. У разі успішного завантаження реклама зберігається в об’єкті InterstitialAd, щоб її можна було відобразити гравцю в потрібний момент під час ігрового процесу.

Такий підхід дозволяє гарантувати, що реклама буде готова до показу без затримок, забезпечуючи плавний і неперервний ігровий досвід. Крім того, метод передбачає обробку можливих помилок завантаження, що дозволяє уникнути збоїв під час відображення реклами.

Лістинг 3.14 — Реалізація класу ShowInterstitialAd

```
public void ShowInterstitialAd()
{
   if (_currentInterstitialAd != null && _currentInterstitialAd.CanShowAd())
        Debug.Log("Showing interstitial ad.");
      _currentInterstitialAd.Show();
   else
      Debug.LogError("Interstitial ad is not ready to be shown.");
   }
}
```

Цей метод відповідає за відображення міжсторінкової реклами. Спершу він перевіряє, чи завантажена реклама і готова до показу. У разі успішної перевірки реклама демонструється користувачеві. Якщо ж реклама не готова до показу, метод виводить повідомлення про помилку, щоб уникнути непередбачуваної поведінки

додатку.

Використання міжсторінкової реклами є важливою складовою монетизації мобільних додатків, оскільки дозволяє отримувати додатковий дохід без значного втручання у геймплей. Наведений код ілюструє, як налаштувати та управляти міжсторінковою рекламою на різних платформах, зокрема Android та iOS. Завдяки правильній реалізації цього методу можна забезпечити надійну роботу реклами, водночас підтримуючи комфорт та зручність для користувачів.

3.6 Поетапний процес публікації гри на платформі App Store

Процес публікації додатку в App Store включає кілька ключових кроків: спочатку створюється сертифікат у Keychain Access, який потім імпортується в обліковий запис Apple Developer; далі створюється унікальний ідентифікатор додатку, після чого програма завантажується через Xcode у App Store Connect та відправляється на перевірку для публікації в магазині додатків App Store.

Для успішної публікації додатку необхідно мати пристрій від Apple (наприклад, MacBook), встановлений Xcode та активний акаунт Apple Developer [26,27].

На початковому етапі створюється сертифікат на пристрої розробника, який згодом завантажується на сайт Apple Developer. Після вибору опції "Request a Certificate From a Certificate Authority..." з’являється вікно, як показано на рисунку 3.2, де можна виконати налаштування сертифіката для подальшої роботи.

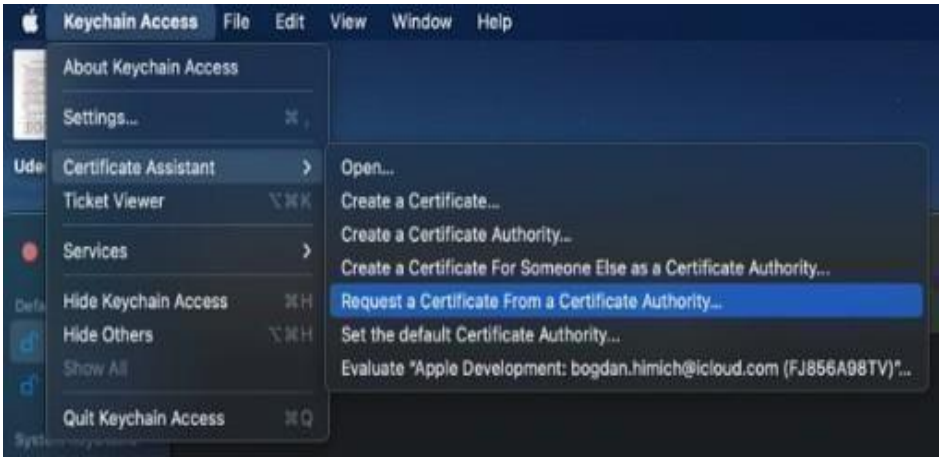

Рисунок 3.2 — Keychain Access

Тут необхідно вказати актуальну електронну пошту та зберегти отриманий файл локально, як представлено на рисунку 3.3.

Рисунок 3.3 — Локальне збереження сертифіката

Після збереження, цей файл автоматично додасться до Keychain Access [28].

Наступним етапом є робота на веб-сайті Apple Developer, де необхідно створити власний обліковий запис. Якщо у вас вже є доступ до організації через існуючий акаунт, власник облікового запису може додати вас до команди розробників. Після успішного входу або приєднання до організації слід прокрутити сторінку до самого низу та обрати розділ "Certificates", як показано на рисунку 3.4.

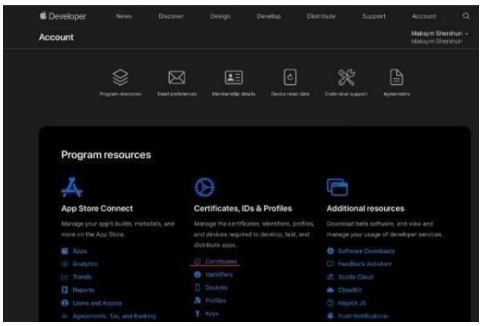

Рисунок 3.4 — Головне меню Apple Developer

Сайт переадресовує на створення сертифіката, де потрібно натиснути на кнопку у вигляді синього "+", як показано на рисунку 3.5

Рисунок 3.5 — Приклад сертифіката

Після натискання на кнопку додати новий сертифікат, оберіть "Apple Distribution", як показано на рисунку 3.6.

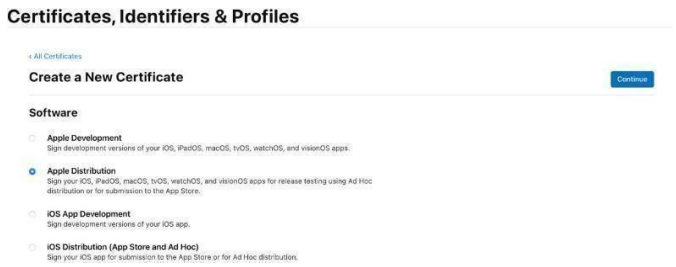

Рисунок 3.6 — Меню вибору сертифіката

Цей тип сертифіката необхідний для публікації додатку в App Store. Натисніть кнопку "Продовжити". Далі слід завантажити файл сертифіката, який був попередньо створений у Keychain Access, у відповідну панель завантаження, як продемонстровано на рисунку 3.7.

Рисунок 3.7 — Встановлення сертифікату

Після виконання даних кроків, потрібно повернутися до вкладки "Certification" і обрати поле "Identifiers", як показано на рисунку 3.8.

Рисунок 3.8 — Генерація “ID” додатку

На цій панелі слід обрати розділ "App IDs", оскільки саме тут створюється Bundle ID для додатку, який слугує унікальним ідентифікатором проєкту. Після виконання всіх необхідних кроків у розділі "Identifiers" з’явиться створений Bundle ID, як показано на рисунку 3.9.

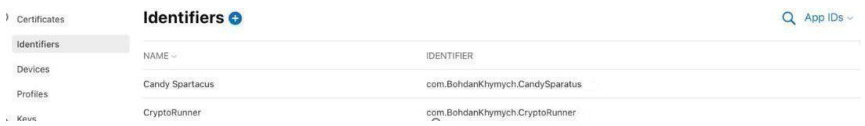

Рисунок 3.9 — Генерація “Bundle Id”

Після створення Bundle ID слід повернутися до Unity, відкрити Project Settings та додати зображення додатку для App Store, а також оформлення зовнішнього вигляду. Далі необхідно вказати отриманий Bundle ID з Apple Developer та виконати збірку проекту. Після завершення компіляції отримані файли потрібно відкрити у Xcode, як показано на рисунку 3.10.

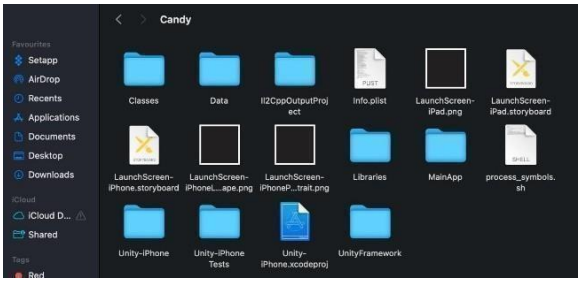

Рисунок 3.10 — Директорія з файлами проекту для Xcode

Слід відкрити синій файл із назвою "Unity — iPhone.xcodeproj" (якщо існує білий файл із такою ж назвою, краще все ж використовувати синій). Після відкриття цього файлу з’явиться панель, представлену на рисунку 3.11.

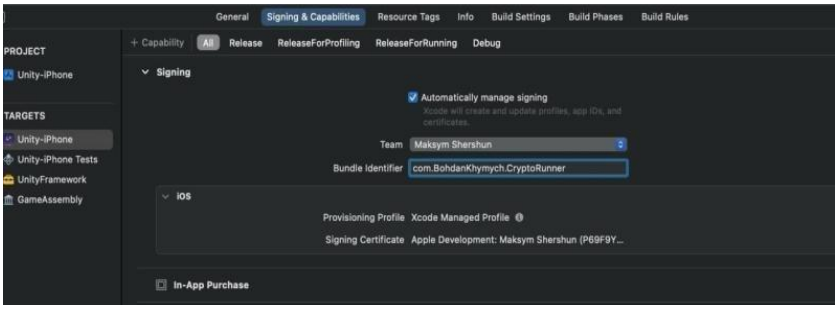

Рисунок 3.11 — Панель вибору організації

На цьому етапі потрібно вказати організацію, після чого залишиться лише архівувати проект і надіслати його на TestFlight через Apple Connect. Після створення архіву відкриється панель, показана на рисунку 3.12.

Рисунок 3.12 —Архів файлу

Натисканням кнопки "Distribute App" (якщо помилок немає) проект автоматично буде відправлено на TestFlight [29].

Для подальшої публікації на сайті Apple Connect у верхньому лівому куті слід натиснути кнопку з плюсом ("+") поруч із версією, як показано на рисунку 3.13.

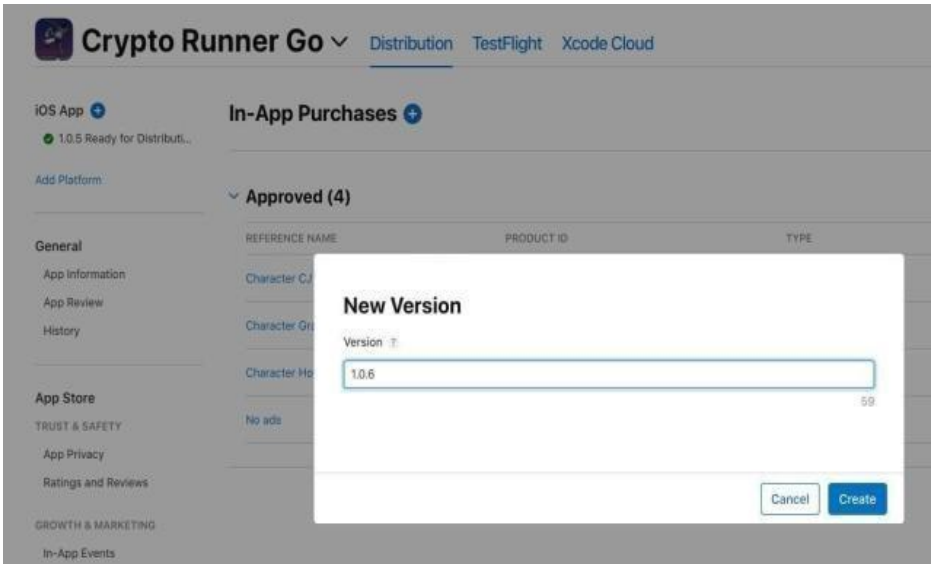

Рисунок 3.13 — Обираємо версію додатку в AppStore

Далі потрібно створити нову версію додатку (кожного разу версію слід збільшувати на 1). Відкриється панель, де слід заповнити всі необхідні дані: персональні відомості, теги, опис та інші поля.

Якщо при натисканні кнопки "відправити на перевірку" залишилися незаповнені обов’язкові поля, система видасть помилки, які потрібно виправити для успішного надсилання проекту на перевірку. Приклад таких помилок наведено на рисунку 3.14.

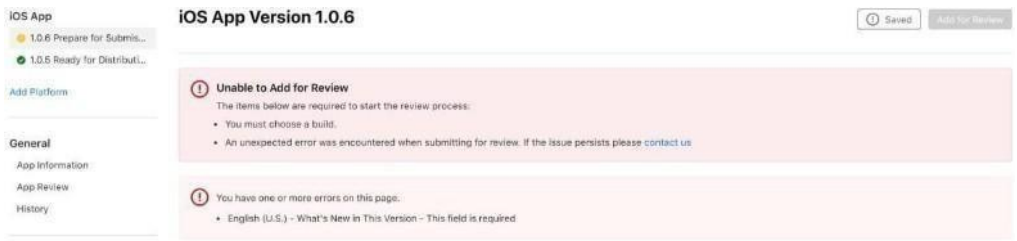

Рисунок 3.14 — Перевірка додатка

Після успішного надсилання додатку на перевірку слід зачекати приблизно один день. Після цього часу проект може бути або схвалений, або відхилений.

У разі відхилення на електронну пошту надсилається лист із детальними вказівками щодо необхідних змін. Якщо такого повідомлення немає, можна перевірити статус додатку безпосередньо в App Store, як показано на рисунку 3.15.

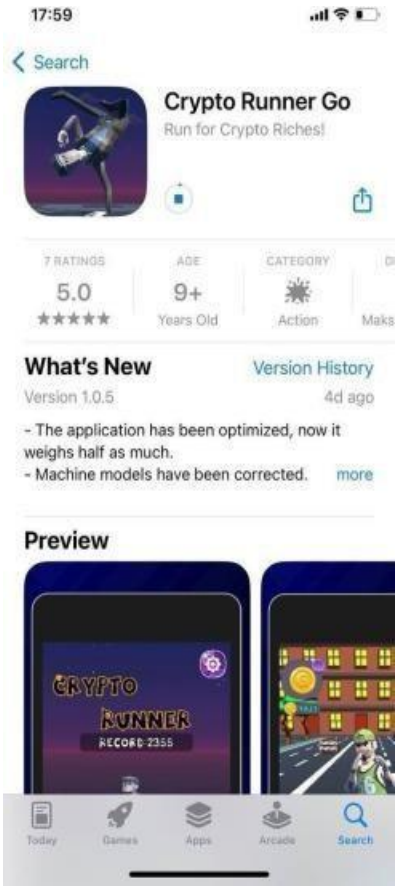

Рисунок 3.15 — Меню додатку в AppStore

Додаток вже доступний на AppStore, тож можна його завантажити та пограти [30]!

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ ФУНКЦІОНУВАННЯ ПРОГРАМИ

4.1 Мета та методика експериментальних досліджень

Метою проведених експериментальних досліджень було комплексне тестування функціональності, стабільності та продуктивності розробленого 3D-застосунку Crypto Runner Go, створеного на базі рушія Unity із використанням мови C#. Основна увага приділялась виявленню можливих недоліків, оптимізації ігрового процесу, перевірці правильності взаємодії різних модулів та оцінці поведінки додатку в умовах реальної експлуатації на мобільних пристроях.

Особливу увагу під час тестування приділяли зручності користувацького інтерфейсу (UI/UX), швидкості реакції ігрових елементів, стабільності роботи програми під час тривалих сесій, коректності обробки внутрішньоігрових подій та відтворення анімацій. Окремо перевірялися механізми взаємодії із зовнішніми сервісами, такими як Unity Ads, а також внутрішньоігрові покупки, реалізовані відповідно до вимог App Store.

Крім того, оцінювалась продуктивність гри на пристроях з різними характеристиками. Було проведено тестування на iPhone SE (2020) та iPhone 13 Pro, що дало змогу оцінити масштабованість додатку, адаптивність інтерфейсу та ефективність використання системних ресурсів.

4.2 Демонстрація роботи застосунку

В рамках експериментального етапу було проведено демонстрацію основних функцій гри Crypto Runner Go. Основний акцент робився не лише на графічній складовій, але й на перевірці внутрішньої логіки гри та реакції системи на дії користувача.

Під час запуску застосунку користувач потрапляє на головний екран, на якому відображено анімованого персонажа, логотип гри та інтерфейс із набором інтерактивних елементів, серед яких кнопки Play, Settings, Shop, Leaderboard та Ads Reward. Інтерфейс розроблено так, щоб він був максимально зрозумілим та зручним: основні елементи керування розташовані в нижній частині екрану, що забезпечує легкий доступ до них під час взаємодії, а у верхній частині

відображаються рекорд гравця та кількість зібраних монет, що дозволяє користувачу відслідковувати прогрес у реальному часі. Головне меню продемонстровано на рисунку 4.1.

Рисунок 4.1 — Головне меню застосунку Crypto Runner Go

Після початку ігрової сесії система формує динамічне ігрове середовище, в якому персонаж рухається вперед, уникаючи різних перешкод. На екрані у верхній частині відображаються поточні очки та зібрані монети, а також кнопка Pause, яка дозволяє призупинити гру у будь-який момент, як показано на рисунку 4.2. Така організація інтерфейсу та відображення інформації забезпечує комфортне управління ігровим процесом та дозволяє користувачу легко орієнтуватися у подіях гри.

Рисунок 4.2 — Активна фаза ігрової сесі

Користувач має можливість призупинити ігровий процес у будь-який момент, після чого на екрані з’являється вікно паузи. Це вікно містить три кнопки: Resume, Restart та Exit. Кнопка Resume дозволяє продовжити поточну сесію гри, Restart перезапускає гру з початку, а Exit повертає користувача на головний екран меню, як продемонстровано на рисунку 4.3.

Рисунок 4.3 — Екран паузи під час гри

У разі завершення ігрової сесії, наприклад, коли персонаж стикається з перешкодою, відображається спеціальне вікно результатів. Воно показує досягнуті очки, кількість зібраних монет, а також надає користувачу можливість одразу розпочати нову гру або повернутися до головного меню за допомогою кнопок Restart та Exit, як показано на рисунку 4.4. Така організація дозволяє гравцю швидко орієнтуватися після закінчення сесії та забезпечує безперервність ігрового процесу.

Рисунок 4.4 — Екран завершення гри

Ще одним ключовим функціональним елементом гри є магазин персонажів, де гравець може переглядати доступних героїв та обирати їх для використання у грі. Користувач має змогу відкривати нових персонажів як за внутрішньоігрову валюту, так і за реальні гроші. Всі зміни відображаються миттєво після вибору, що створює ефект інтерактивності та живого, динамічного інтерфейсу, як показано на рисунку 4.5.

Рисунок 4.5 — Магазин персонажів із можливістю покупки

У процесі тестування також перевірялось відтворення реклами за винагороду. Перед показом рекламного ролика гравцю обов’язково демонструється запит на підтвердження перегляду. Це гарантує, що користувач контролює свої дії, а реалізація відповідає етичним стандартам використання реклами, як показано на рисунку 4.6.

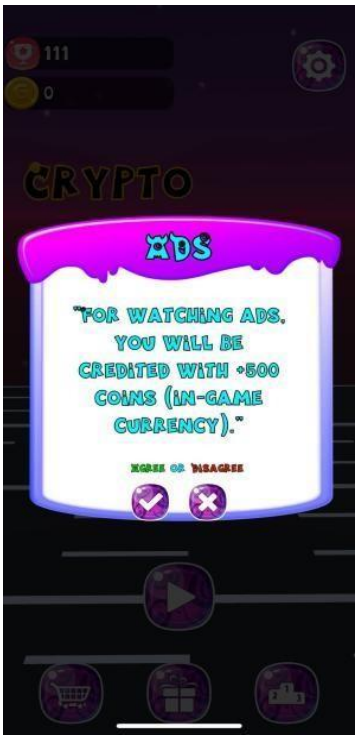

Рисунок 4.6 — Вікно підтвердження перегляду реклами за винагороду

На екрані налаштувань гравець може керувати основними параметрами гри, ознайомитися з політикою конфіденційності та при бажанні вимкнути рекламу через внутрішньоігрову покупку. Всі необхідні функції доступні за кілька натискань, що забезпечує зручну та швидку взаємодію з додатком, як продемонстровано на рисунку 4.7.

Рисунок 4.7 — Екран налаштувань із кнопкою «No Ads»

4.3 Оцінка продуктивності та стабільності

Під час тестування було підтверджено високу стабільність ігрового процесу, навіть у тривалих сесіях. Застосунок працював без збоїв, зависань або втрат кадрів. На сучасних пристроях гра забезпечувала стабільну частоту кадрів (60 FPS), плавність анімацій, швидкий відгук на керування та повну відповідність очікуваній логіці.

Анімаційні ефекти реалізовані за допомогою бібліотеки DOTween, що дозволяє досягти м’яких і природних переходів без зайвого навантаження на систему. Управління через swipe-жести показало точність і відсутність затримок, а використання Zenject забезпечило чітке розмежування залежностей між модулями, що значно спрощує тестування та подальшу підтримку проєкту.

Навіть на менш потужних пристроях, таких як iPhone SE, гра залишалася плавною, хоч і знижувала частоту кадрів до приблизно 30 FPS, проте без втрати функціональності чи стабільності. Це свідчить про ефективну оптимізацію та раціональне використання ресурсів.

Система внутрішньоігрової реклами та покупок функціонувала без збоїв: відеореклама запускалася без затримок, після перегляду користувач одразу отримував відповідну винагороду, а покупки оброблялися коректно із миттєвим відображенням змін у інтерфейсі.

За підсумками експериментальних досліджень встановлено, що Crypto Runner Go демонструє високий рівень стабільності, інтерактивності та продуктивності на різних мобільних пристроях. Усі основні модулі працюють коректно, ігровий процес є динамічним та інтуїтивно зрозумілим, а елементи управління — чутливими й швидкими.

Використання сучасних технологій та бібліотек (Unity, DOTween, Zenject, UniTask) дозволило досягти високої якості програмної реалізації при збереженні ефективного використання ресурсів. Це дає підстави стверджувати, що розроблений застосунок готовий до широкого використання, включно з публікацією в App Store та можливим подальшим масштабуванням.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ.

Актуальність комерційного та технологічного аудиту для розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO випливає з високої конкуренції ринку і швидкої еволюції технологій. Гібридні ігри/застосунки з елементами криптоекономіки потребують перевірки життєздатності бізнес-моделі (монетизація, токеноміка, розподіл доходу), відповідності регуляторним вимогам і відповідності запитам цільової аудиторії на кожній платформі (мобільні ОС, десктоп, web, консолі). Оскільки 3D-інтерактивність і криптофункції додають технічної та юридичної складності, аудит зменшує ризики фінансових витрат та іміджевих втрат під час масштабування продукту.

Проведення комерційного аудиту має включати аналіз ринку і конкурентів, оцінку цільової аудиторії, валідацію монетизаційних сценаріїв (продажі в додатку, підписки, NFT/токени, реклама), прогноз прибутковості та сценарії B-/C-/D-ризиків. Важливо перевірити економіку життєвого циклу користувача (LTV, CAC), юридичну відповідність криптоопераций у цільових юрисдикціях, а також партнерські можливості (маркет-плейси, блокчейн-інфраструктура). Результат — набір рекомендацій щодо пріоритетів розробки, часу виходу на ринок і бізнес-метрик для подальшого моніторингу.

Технологічний аудит повинен охоплювати архітектуру застосунку, вибір рушія (Unity/Unreal/власний), крос-платформену сумісність, оптимізацію продуктивності (фрейм-рейт, пам’ять, мережеві затримки), безпеку (шахрайство, збереження приватних ключів, безпечні API), CI/CD та тестування (автоматизовані тести, профайлінг, нагрузкові сценарії). Окремо — оцінка інтеграції з блокчейном (підтримувані ланцюги, газ-витрати, орієнтовні витрати), локалізації і доступності, а також підготовка до сертифікацій і публікації в магазинах (App Store, Google Play, Steam).

Кінцевий звіт має містити технічний борг, roadmap із ризиками й оцінками ресурсів для кожної платформи.

Для проведення комерційного та технологічного аудиту залучаємо 3-х незалежних експертів, якими є провідні викладачі випускової або спорідненої кафедри.

Оцінювання науково-технічного рівня інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ та її комерційного потенціалу здійснюємо із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, а результати зводимо до таблиці 1.

Таблиця 1 — Результати оцінювання науково-технічного рівня і комерційного потенціалу інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ

Критерії	Експерти		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами		
Технічна здійсненність концепції	3	3	3
Ринкові переваги (наявність аналогів)	2	2	1
Ринкові переваги (ціна продукту)	3	2	2
Ринкові переваги (технічні властивості)	3	3	3
Ринкові переваги (експлуатаційні витрати)	2	3	3
Ринкові перспективи (розмір ринку)	2	2	2
Ринкові перспективи (конкуренція)	2	1	2
Практична здійсненність (наявність фахівців)	3	3	3
Практична здійсненність (наявність фінансів)	2	2	1
Практична здійсненність (необхідність нових матеріалів)	3	3	3

Продовження таблиці 1

Практична здійсненність (термін реалізації)	2	2	2
Практична здійсненність (розробка документів)	3	3	3
Сума балів	30	29	28
Середньоарифметична сума балів, СБ	29	Середньоарифметична сума балів, СБ	29

За результатами розрахунків, наведених в таблиці 1 робимо висновок про те, що науково-технічний рівень та комерційний потенціал інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ - середній.

5.2 Розрахунок витрат на здійснення розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ.

Витрати на оплату праці. Належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці, також будь-які види грошових і матеріальних доплат, які належать до елемента «Витрати на оплату праці».

Основна заробітна плата дослідників. Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{к} \frac{М_{ni} \cdot t_i}{Т_р},$$

де к — кількість посад дослідників, залучених до процесу дослідження;

$М_{пі}$ — місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_р$ — число робочих днів в місяці; приблизно $Т_р$ = (21…23) дні, приймаємо 22 дні;

$t_і$ — число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 2.

Таблиця 2 — Витрати на заробітну плату дослідників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	65 000	2955	30	88636
Розробник	55 000	2500	45	112500
Консультанти	40 000	1818	20	36364
Всього:	237500			

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_і \cdot t_і,$$

де $С_і$ — погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

$t_і$ — час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду С і можна визначити за формулою:

$$С_і = \frac{М_м \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}},$$

де $М_м$ — розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), у 2025 році $М_м$=8000 грн;

$К_і$ — коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ — мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати, складає 1,1;

$Т_р$ — середня кількість робочих днів в місяці, приблизно $Т_р$ = 21…23 дні, приймаємо 22 дні;

$t_{зм}$ — тривалість зміни, год., приймаємо 8 год.

Таблиця 3 — Витрати на заробітну плату робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Створення 3D-моделей та анімацій персонажів	40	3	59	1,18	2360
UI/UX-дизайн мобільного застосунку	32	3	59	1,18	1888
Тестування ігрових механік та оптимізація	60	2	54,5	1,09	3270
Публікація в App Store / Google Play + налаштування SDK	20	3	59	1,18	1180
Всього					8698

Додаткова заробітна плата. Додаткова заробітна плата $З_д$ всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$З_д = 0{,}1 \cdot (З_о + З_р) = 0{,}1 \cdot (237500 + 8698) = 24620 \text{ грн.}$$

Відрахування на соціальні заходи. Нарахування на заробітну плату $Н_{зп}$ розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Н_{зп} = \beta \cdot (З_о + З_р + З_д) =$$
$$= 0{,}22 \cdot (237500 + 8698 + 24620) = 59580 \text{ грн.}$$

де $З_о$ — основна заробітна плата розробників, грн.;

$З_р$ — основна заробітна плата робітників, грн.;

$З_д$ — додаткова заробітна плата всіх розробників та робітників, грн.;

β — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

Розрахунок витрат на матеріали. Витрати на матеріали М, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_i,$$

де $Н_i$ — кількість матеріалів і-го виду, шт.;

$Ц_i$ — ціна матеріалів і-го виду, грн.;

$К_i$ — коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

n — кількість видів матеріалів.

Таблиця 4 — Матеріали, що використані на розробку

Найменування матеріалів	Ціна за одиницю, грн.	Витрачено	Вартість витрачених матеріалів, грн.
3D-моделі персонажів (Asset Store)	1200	3	3600
Пакет текстур та матеріалів для сцени	900	2	1800

Продовження таблиці 4

Музичний трек (ліцензія)	1500	1	1500
Всього, з врахуванням коефіцієнта транспортних витрат			7590

Розрахунок витрат на комплектуючі. Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i,$$

де $Н_i$ — кількість комплектуючих і-го виду, шт.;

$Ц_i$ — ціна комплектуючих і-го виду, грн.;

$К_i$ — коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); n – кількість видів комплектуючих.

Таблиця 5 — Комплектуючі, що використані на розробку

Найменування комплектуючих	Ціна за одиницю, грн.	Витрачено	Вартість витрачених комплектуючих, грн.
Геймпад для тестування	1800	1	1800
Мобільний пристрій Android (тестовий)	5500	1	5500
Мобільний пристрій iOS (тестовий)	9500	1	9500
Bluetooth-клавіатура для дебагу	700	1	700
Всього, з врахуванням коефіцієнта транспортних витрат			19425

Спецустаткування для наукових (експериментальних) робіт. Вартість спецустаткування визначається за прейскурантом гуртових цін або за даними базових підприємств за відпускними і договірними цінами.

$$В_{спец} = \sum_{1}^{к} Ц_i \cdot С_{пр.i} \cdot К_i,$$

де $Ц_i$ — ціна придбання спецустаткування і-го виду, грн.;

$С_{пр.i}$ — кількість одиниць спецустаткування відповідного виду, шт.;

$К_i$ — коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); n – кількість видів спецустаткування.

Таблиця 6 — Витрати на придбання спецустаткування

Найменування спецустаткування	Ціна за одиницю, грн.	Витрачено	Вартість спецустаткування, грн.
Mac Mini для збірки iOS-версії	36000	1	36000
VR-гарнітура для тестування камер (опціонально)	12000	1	12000
Оновлення комплекту периферії (миша +клавіатура)	2200	1	2200
Всього, з врахуванням коефіцієнта транспортних витрат			55220

Програмне забезпечення. До балансової вартості програмного забезпечення входять витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі 10…12% від вартості програмного забезпечення. Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{1}^{к} Ц_{iпрг} \cdot С_{прг.i} \cdot К_i,$$

де $Ц_{iпрг}$ — ціна придбання програмного забезпечення і-го виду, грн.;

$С_{прг.i}$ — кількість одиниць програмного забезпечення відповідного виду, шт.;

$К_i$ — коефіцієнт, що враховує інсталяцію, налагодження програмного забезпечення, $К_i$ = (1,1…1,12);

к — кількість видів програмного забезпечення.

Таблиця 7 — Витрати на придбання програмного забезпечення

Найменування програмного забезпечення	Ціна за одиницю, грн.	Витрачено	Вартість програмного забезпечення, грн.
Unity Pro / Unity Asset Store контент	3 000	1	3000
Rider (ліцензія)	2 600	1	2600
Figma (прототип UI/UX)	900	1	900
Всього, з врахуванням коефіцієнта інсталяції та налагодження			7215

Амортизація обладнання. Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час (чи для) виконання даного етапу роботи.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$А = \frac{Ц_б}{Т_в} \cdot \frac{t}{12},$$

де $Ц_б$ — загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.;

t — термін використання основного фонду, місяці;

$Т_в$ — термін корисного використання основного фонду, роки.

Таблиця 8 — Амортизаційні відрахування за видами основних фондів

Найменування	Балансова вартість, грн.	Строк корисного використання, років	Термін використання, місяців	Сума амортизації, грн.
Робоча станція (ПК для Unity)	42 000	4	2	1750,0
Монітор 27''	9 000	5	2	300,0
Ноутбук для мобільного тестування	28 000	3	2	1555,6
Всього	3605,6			

Витрати на електроенергію для науково-виробничих цілей. Витрати на

силову електроенергію $В_е$, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

Таблиця 9 — Витрати на електроенергію

Найменування обладнання	Потужність, кВт	Тривалість годин роботи
Робоча станція	0,35	160
Unity-розробника	0,12	120
Ноутбук для тестування	0,06	80

$$Ве = \sum \frac{W_i \cdot t_i \cdot Ц_е \cdot К_{впі}}{ККД}$$

$$= \frac{0,35 \cdot 160 \cdot 4,32 \cdot 0,75}{0,98} + \frac{0,12 \cdot 120 \cdot 4,32 \cdot 0,75}{0,98}$$

$$+ \frac{0,06 \cdot 80 \cdot 4,32 \cdot 0,75}{0,98} = 248,6 \text{ грн.},$$

W_i — встановлена потужність обладнання, кВт;

t_i — тривалість роботи обладнання на етапі дослідження, год.;

$Ц_е$ — вартість 1 кВт електроенергії, 4,32 грн.;

$К_{впі}$ — коефіцієнт використання потужності;

ККД – коефіцієнт корисної дії обладнання.

Інші витрати. До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$Ів = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%} = (237500 + 8698) \cdot \frac{80}{100} = 196958 \text{ грн.},$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

Накладні (загальновиробничі) витрати. До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…200% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Внзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%} = (237500 + 8698) \cdot \frac{100}{100} = 246198 \text{ грн.},$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Витрати на проведення розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ. Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = 237500 + 8698 + 24620 + 59580 + 7590 + 19425 + 55220 + 7215 + 3605 + 248{,}6 + 196958 + 246198 = 866858 \text{ грн.}$$

Загальні витрати. Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи з розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} = \frac{866858}{0{,}5} = 1733717 \text{ грн.},$$

де η — коефіцієнт, що характеризує етап виконання науково-дослідної роботи. Оскільки, якщо науково-технічна розробка знаходиться на стадії розробки дослідного зразка, то η=0,5.

5.3 Розрахунок економічної ефективності науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ за її можливої комерціалізації потенційним інвестором.

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ, є збільшення у потенційного інвестора величини чистого прибутку.

В даному випадку відбувається розробка засобу, тому основу майбутнього економічного ефекту буде формувати: ΔN – збільшення кількості споживачів, яким надається відповідна інформаційна послуга в аналізовані періоди часу;

N — кількість споживачів, яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки;

$Ц_б$ — вартість послуги у році до впровадження інформаційної системи;

$\pm\Delta Ц_o$ — зміна вартості послуги (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу.

Можливе збільшення чистого прибутку у потенційного інвестора ΔПі для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N_i)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}),$$

де ±ΔЦ — зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році

(відносно року до впровадження цієї розробки);

$\pm\Delta Ц_о$ може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни);

N — основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ — основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році;

$Ц_б$ — основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN — зміна основного кількісного показника від впровадження результатів науково- технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ — коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2025 році ставка податку на додану вартість становить 20%, а коефіцієнт λ =0,8333;

ρ — коефіцієнт, який враховує рентабельність інноваційного продукту (послуги).

Рекомендується брати ρ =0,2...0,5; ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році ϑ =18%.

Очікуваний термін життєвого циклу розробки 3 роки, тому:

1-й рік	ΔП1	15727457	грн.
2-й рік	ΔП2	29456963	грн.
3-й рік	ΔП3	43186469	грн.

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} = \frac{15727457}{(1+0{,}1)^1} + \frac{29456963}{(1+0{,}1)^2} + \frac{43186469}{(1+0{,}1)^3} = 71088918 \text{ грн.},$$

де ΔПі — збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн.;

Т — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки (приймаємо Т=3 роки);

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15; t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

Далі розраховують величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ. Для цього можна використати формулу:

$$PV = k_{\text{інв}} \cdot ЗВ = 5 \cdot 1733717 = 8668583 \text{ грн.}$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково- технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо.Зазвичай $k_{інв}$ =1...5, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ становитиме:

$$Е_{абс} = ПП - PV = 71088918 - 8668583 = 62420335 \text{ грн.},$$

де ПП — приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ, грн.;

PV — теперішня вартість початкових інвестицій, грн.

Оскільки $Е_{абс}>0$, то можемо припустити про потенційну зацікавленість у розробці інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність Ев або показник внутрішньої норми дохідності вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ вкладати буде економічно недоцільно.

Внутрішня економічна дохідність інвестицій Ев, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ, розраховується за формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Еабс}{PV}} = \sqrt[3]{1 + \frac{62420335}{8668583}} = 2{,}73,$$

де $Т_ж$ — життєвий цикл розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ, роки.

Далі розраховуємо період окупності інвестицій $Т_о$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних

платформ:

$$T_о = \frac{1}{E_в} = \frac{1}{2{,}73} = 0{,}37 \text{ роки.}$$

Оскільки $T_о$<1…3-х років, то це свідчить про комерційну привабливість науково-технічної розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ і може спонукати потенційного інвестора профінансувати впровадження цієї розробки інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ та виведення її на ринок.

Проведений комерційний і технологічний аудит інтерактивного 3D-застосунку CRYPTO RUNNER GO для різних платформ показав, що проєкт має середній науково-технічний рівень і комерційний потенціал (СБ = 29 балів). Аудит підтвердив доцільність подальшої розробки, але вказав на необхідність удосконалення бізнес-моделі, оптимізації технічної архітектури та зміцнення ресурсної бази.

Загальні витрати на виконання розробки становлять 1 733 717 грн, а з урахуванням інвестиційних потреб потенційного інвестора (k=5) початкові інвестиції мають скласти 8 668 583 грн. Розрахунки економічної ефективності показали значний потенціал зростання прибутку впродовж трирічного циклу комерціалізації, а приведена вартість очікуваних прибутків становить 71 088 918 грн.

Отже, попри значний обсяг початкових витрат, розробка CRYPTO RUNNER GO є економічно привабливою для інвестора та має перспективи успішної комерціалізації за умови коректного управління ризиками, технічної оптимізації та ефективного маркетингового супроводу.

ВИСНОВОК

У межах магістерської кваліфікаційної роботи було здійснено повний цикл розробки інтерактивного 3D-застосунку Crypto Runner Go для мобільних платформ із використанням ігрового рушія Unity. Під час роботи проведено детальний аналіз еволюції відеоігор, особливостей жанру «раннер», сучасних інструментів розробки та патернів проєктування, що дозволило обґрунтовано обрати архітектурні рішення та технології для реалізації проєкту.

У результаті дослідження створено повноцінний програмний продукт із модульною архітектурою, що реалізована за допомогою фреймворку Zenject для управління залежностями. Для асинхронної обробки даних використано UniTask, а інструменти DOTween та Cinemachine забезпечили плавність анімацій і якісну роботу камери. Реалізовано інтерфейс, оптимізований для мобільних пристроїв, ігрову логіку з динамічною генерацією перешкод, систему збору бонусів, магазин персонажів, а також інтеграцію реклами та внутрішніх покупок.

Експериментальні дослідження підтвердили стабільність функціонування гри на різних моделях мобільних пристроїв, високу продуктивність, плавність анімацій та коректну роботу всіх модулів. Під час тестування не виявлено критичних помилок, а час відгуку інтерфейсу та поведінка ігрових об’єктів відповідали сучасним вимогам мобільних застосунків. Окрему увагу приділено процесу публікації гри: розроблений продукт успішно пройшов модерацію та був розміщений у App Store, що підтверджує відповідність застосунку технічним і UX-стандартам платформи.

Отже, поставлені у роботі мета та завдання повністю досягнуті. Розроблений інтерактивний 3D-застосунок Crypto Runner Go демонструє якісну реалізацію механік жанру раннер, забезпечує привабливий і інтуїтивно зрозумілий ігровий досвід і може бути використаний як комерційний продукт у сфері мобільних ігор.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Історія геймдеву xx століття: від Spacewar! До Sonic The Hedgehog — [режим доступу] https://skvot.io/uk/blog/istoriya-geymdevu-hh-stolittya-vid-spacewar-do-sonic-the-hedg ehog

2. Ігри в жанрі “Екшен” — [режим доступу] https://agromolod.org/uk/eksheny/

3. Ігри в жанрі “Стратегія” — режим доступу] https://igropedia.fandom.com/wiki/Стратегічні_відеоігри.

4. Ігри в жанрі “Рольові ігри” — [режим доступу] https://igrodrom.net/ua/rpg/

5. Ігри в жанрі “Жахи” — [режим доступу] https://igrodrom.net/ua/ho rror/ - google_vignette

6. Ігри в жанрі “Безкінечний ранер” — [режим доступу] https://uk.sharpcoderblog.com/blog/runner-game-fundamentals

7. Шаблоний патерн програмування "MVC" — [режим доступу] https://www.tutorialsteacher.com/mvc/mvc-architecture

8. Шаблоний патерн програмування "MVP" — [режим доступу] https://www.c-sharpcorner.com/article/mvp-design-pattern-in-c-sharp/

9. Шаблоний патерн програмування "MVVM" — [режим доступу] https://symphony-solutions.com/insights/implementing-mvvm-pattern

10. Середовище для розробки коду "IDE Visual Studio" — [режим доступу] https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022

11. Середовище для розробки коду "IDE Rider" — [режим доступу] https://www.jetbrains.com/rider/

12. Best Programming Language for Game —[режим доступу] https://hackr.io/blog/best-programming-language-for-games

13. Top 10 Game Development Engines — [режим доступу] https://www.youngwonks.com/blog/Top-10-Game-Development-Engines-Today

14. Unity editor — [режим доступу] https://unity.com/ru

15. AppStore — [режим доступу] https://www.wikiwand.com/uk/App_Store_(iOS)

16. Google Play — [режим доступу] https://play.google/howplayworks/

17. Game Designing. Video Game Development — [режим доступу] https://www.gamedesigning.org/video-game-development/

18. Cysharp. Dotween demigiant — [режим доступу] https://dotween.demigiant.com/

19. Modesttree.Zenject — [режим доступу] https://github.com/modesttree/Zenject.

20. How use Zenject in code — [режим доступу] https://habr.com/ru/articles/781016/

21. Unity Technologies. Cinemachine package — [режим доступу] https://docs.unity3d.com/Packages/com.unity.cinemachine@2.1/manual/index.html

22. Cysharp. UniTask — [режим доступу] https://github.com/Cysharp/U niTask

23. Unity. Asset Store — [режим доступу] https://assetstore.unity.com/

24. Unity. Life Cycle MonoBehaviour — [режим доступу] https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

25. Google Mobile Ads SDK iOS — [режим доступу] https://developers.google.com/admob/ios/download

26. Documentation. Xcode — [режим доступу] https://developer.apple.com/documentation/xcode

27. Apple Developer — [режим доступу] https://developer.apple.com/

28. Apple Keychain access — [режим доступу] https://support.apple.co m/ru-ru/guide/keychain-access/kyca1083/mac

29. Apple Test Flight – [режим доступу] https://vikramios.medium.com/what-is-testflight-how-to-use-it-44dba5d2f263

30. Game in App Store "Crypto Runner Go" — [режим доступу] https://apps.apple.com/ua/app/crypto-runner- go/id6464288913

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ
проф., д.т.н. Азаров О. Д.

«25» вересня 2025р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської кваліфікаційної роботи

«Інтерактивний 3d-застосунок Crypto Runner Go

для різних платформ»

08-54.МКР.014.00.000 ПЗ

Науковий
керівник: доцент
к.т.н. Колесник І.С.
Студент групи:
2КІ-24м Химич Б.В.

ВНТУ 2025

1 Підстава для виконання магістерської кваліфікаційної роботи.

Підставою для виконання магістерської кваліфікаційної роботи є необхідність розробки інтерактивного 3D-застосунку "Crypto Runner Go" для різних платформ, що забезпечить гравцям унікальний досвід гри в умовах високої конкуренції на ринку мобільних ігор. Це вимагає врахування специфічних вимог до розробки, тестування та оптимізації мобільних застосунків з метою підвищення ефективності гри та задоволення користувацьких потреб.

2 Мета і призначення МКР

Метою даної магістерської кваліфікаційної роботи є створення програмної частини гри "Crypto Runner Go" для різних платформ, інтеграція рекламних та покупкових SDK, а також забезпечення публікації додатку на платформі AppStore.

3 Вихідні дані для виконання МКР

3.1 Огляд сучасних мобільних ігор зі схожою механікою.

3.2 Особливості створення програмного забезпечення для мобільних

3.3 платформ на базі C# та Unity.

3.4 Інтеграція SDK для реклами та покупок у застосунок.

3.5 Створення зручного та інтуїтивно зрозумілого користувацького інтерфейсу.

3.6 Тестування гри та оцінка її продуктивності.

3.7 Публікація додатку на платформі AppStore.

4 Вимоги до виконання МКР

Розроблена гра має бути здатна працювати на основних мобільних платформах, зокрема iOS. Використання мови програмування C# та ігрового рушія Unity для реалізації ігрової механіки. Забезпечення інтеграції SDK для реклами та покупок. Розробка зручного користувацького інтерфейсу.

Забезпечення публікації додатку на платформі AppStore з дотриманням всіх вимог і стандартів.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в таблиці А.1.

Таблиця А.1 — Етапи МКР

№ з/п	Назва етапу	Термінвиконання		Очікувані результати
		початок	кінець	
1	Аналіз задачі	26.09.25	30.09.25	Огляд джерел, висновки із взаємодії викладачів та студентів
2	Огляд перших відеоігор	01.10.25	05.10.25	Розділ 1
3	Огляд існуючих мов програмування та платформ для створення ігор	06.10.25	15.10.25	Розділ 1
4	Структура додатка	16.10.25	31.10.25	Розділ 2
5	Програмна реалізація застосунку	01.11.25	17.11.25	Розділ 3
6	Інтеграція фреймворків	18.11.25	20.11.25	Розділ 3
7	Оформлення пояснювальної записки та презентації	21.11.25	01.11.25	ПЗ, графічний матеріал, презентація

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення та порядок виконання МКР

При оформлюванні МКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— документами на які посилаються у вище вказаних.

ДОДАТОК Б

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ
ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інтерактивний 3D-застосунок CRYPTO RUNNER GO для різних платформ
Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)
Підрозділ Кафедра обчислювальної техніки. Факультет інформаційних технологій та комп'ютерної інженерії. Група 2КІ-24м
(кафедра, факультет, навчальна група)
Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 19 %
Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:
Азаров О.Д., завідувач кафедри ОТ _____
(прізвище, ініціали, посада) (підпис)
Мартинюк Т.Б, гарант освітньої програми КІ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С.М
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ к.т.н., доц., доцент кафедри ОТ Колесник І.С.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Химич Б.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Графічне представлення структури гри

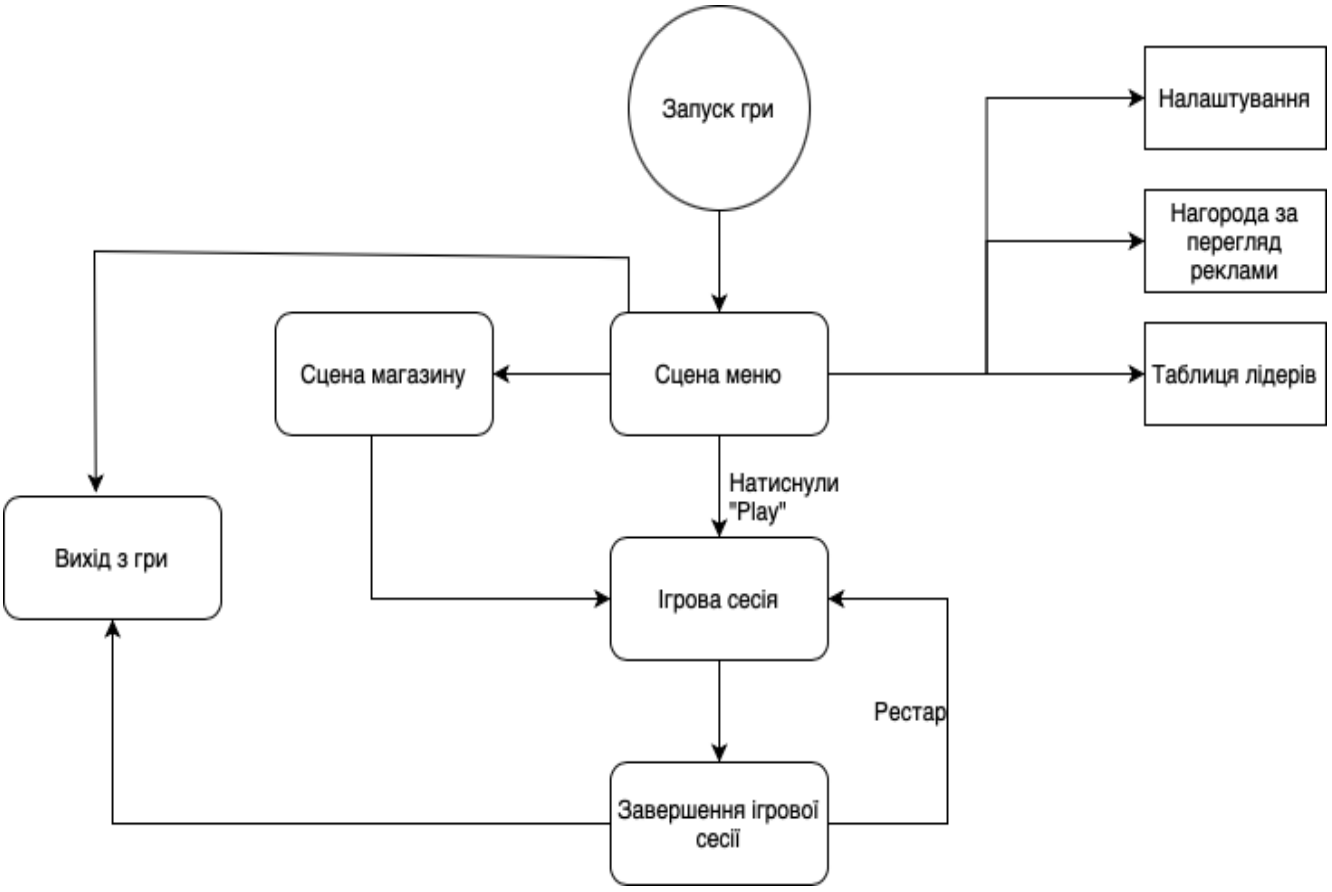

Рисунок В.1 — Графічне представлення структури гри

ДОДАТОК Г

Інтерфейс головного меню

Рисунок Г.1 — Інтерфейс головного меню

ДОДАТОК Д

Код програми

Лістинг Д.1 — Код програми

```
using System.Collections;
using
System.Collections.Ge
neric; using
UnityEngine;

public class TileGenerator : MonoBehaviour
{
   [SerializeField] private GameObject[]
   _tilePrefabs; [SerializeField] private
   Transform _player;
   public float _tileSpeed =
   30.0f;  private float
   _maxSpeed = 75.0f;
   private float
   _destroyDistance = 60.0f;
   private int
   _minTilesOnScene = 6;

   private List<GameObject> _activeTiles = new
   List<GameObject>(); private float _tileLength = 100;
   private int _startTiles
   = 6;  private int
   _currentTileIndex =
   0;

   private void Awake()
   {
      for (int i = 0; i < _startTiles; i++)
      {
         SpawnTile();
      }
   }

   private void Start()
   {
      StartCoroutine(SpeedIncrease());
   }

   private void Update()
   {
      foreach (var tile in _activeTiles)
      {
         tile.transform.Translate(Vector3.back * _tileSpeed * Time.deltaTime);
      }

      if (_activeTiles.Count > 0 && _activeTiles[0].transform.position.z < _player.position.z - _destroyDistance)
      {
         DeleteTile();
      }

      if (_activeTiles.Count < _minTilesOnScene)
      {
         SpawnTile();
      }
```

```
    }

    private void SpawnTile()
    {
        int tileIndex = _currentTileIndex % _tilePrefabs.Length;
        GameObject nextTile = Instantiate(_tilePrefabs[tileIndex], new Vector3(0, 0, _tileLength *
_currentTileIndex), Quaternion.identity);

        if (_activeTiles.Count > 0)
        {
            Vector3 previousTilePosition = _activeTiles[_activeTiles.Count - 1].transform.position;
            nextTile.transform.position = new Vector3(0, 0, previousTilePosition.z + _tileLength);
        }

        _activeTiles.Add(nextTile);
        _currentTileIndex++;
    }

    private void DeleteTile()
    {
        if (_activeTiles.Count > 0)
        {
            Destroy(_activeTiles[0]);
            _activeTiles.RemoveAt(0);
        }
    }

    private IEnumerator SpeedIncrease()
    {
        yield return new
        WaitForSeconds(10); if
        (_tileSpeed < _maxSpeed)
        {
            _tileSpeed += 3;
            StartCoroutine(SpeedIn
            crease());
            Debug.Log($"Speed{_ti
            leSpeed}");
        }
    }
}

using System.Collections;
using
System.Collections.Ge
neric; using
UnityEngine;

public class SwipeController : MonoBehaviour
{
    public static bool tap, swipeLeft, swipeRight, swipeUp,
    swipeDown; private bool isDraging = false;
    private Vector2 startTouch, swipeDelta;

    private void Update()
    {
        tap = swipeDown = swipeUp = swipeLeft = swipeRight =
        false; #region
        if (Input.GetMouseButtonDown(0))
        {
            tap  =
            true;
            isDra
```

```
    ging
    =
    true;
    startTouch = Input.mousePosition;
}
else if (Input.GetMouseButtonUp(0))
{
    isDra
    ging =
    false;
    Reset(
    );
}
#endregion

#region
if (Input.touches.Length > 0)
{
    if (Input.touches[0].phase == TouchPhase.Began)
    {
        tap = true;
        isDraging = true;
        startTouch = Input.touches[0].position;
    }
    else if (Input.touches[0].phase == TouchPhase.Ended || Input.touches[0].phase == TouchPhase.Canceled)
    {
        isDra
        ging =
        false;
        Reset(
        );
    }
}
#endregion

swipeDelta =
Vector2.zero; if
(isDraging)
{
    if (Input.touches.Length < 0)
        swipeDelta = Input.touches[0].position -
    startTouch; else if (Input.GetMouseButton(0))
        swipeDelta = (Vector2)Input.mousePosition - startTouch;
}

if (swipeDelta.magnitude > 100)
{
    float x =
    swipeDelt
    a.x; float y
    =
    swipeDelta
    .y;
    if (Mathf.Abs(x) > Mathf.Abs(y))
    {

        if (x < 0)
            swip
        eLeft =
        true;
        else
            swipeRight = true;
    }
```

```
            else
            {
                if (y < 0)
                    swipe
                Down =
                true; else
                    swipeUp = true;
            }
            Reset();
        }

    }

    private void Reset()
    {
        startTouch = swipeDelta =
        Vector2.zero; isDraging = false;
    }
}

using
UnityEngi
ne; using
UnityEngi
ne.UI;
using
TMPro;
using Zenject;
using System.Collections;


public class PlayerController : MonoBehaviour
{
    [Header("Text")]
    [SerializeField] private TMP_Text _coinsText;

    [SerializeField] private TMP_Text

    _scoreText; [Header("Panels")]
    [SerializeField] private GameObject _losePanel;

    [Header("Scripts")]
    [SerializeField] private Score Score;
    [SerializeField] private SoundManager
    _soundManager; [SerializeField] private
    TileGenerator _tileGenerator;

    private Rigidbody
    _rigidbody;
    private Animator
    _animator; private
    Vector3 _dir;

    private float
    _jumpSpeed = 40;
    private float
    _gravityScale = 10;
    private float
    _fallingGravityScale = 30f;
    private float _flipForce = -
    20.0f;  private float
    _currentGravityScale;
    private float _lineDistanse =
```

```
6.6f; private int
_lineToMove = 1;
private
int
_coins;
private int
_tryCoun
t;  private
bool
_roll;
private bool
_canJump = true;
private bool
_isFlipping = false;
private bool
_isJumping = false;

private IAdsService

_adsService; [Inject]
public void Construct(IAdsService adsService)
{
    _adsService = adsService;
}

private void Start()
{
    Time.timeScale = 1;
    _animator = GetComponentInChildren<Animator>();
    _rigidbody = GetComponentInChildren<Rigidbody>();
    _coins = PlayerPrefs.GetInt("coins");
    _coinsText.text = _coins.ToString();
    _tryCount = PlayerPrefs.GetInt("tryCount");
    _currentGravityScale = _gravityScale;
}

private void Update()
{
    HandlePlayerInputAndMovement();
}

private void HandlePlayerInputAndMovement()
{
    if (SwipeController.swipeRight)
    {
        _soundManager.Sw
        ipeSound(); if
        (_lineToMove < 2)
            _lineToMove++;
    }
    if (SwipeController.swipeLeft)
    {
        _soundManager.Sw
        ipeSound(); if
        (_lineToMove > 0)
            _lineToMove--;
    }

    if (SwipeController.swipeUp)
    {
        if (IsGrounded() && _canJump)
        {
            Jump();
```

```
            _canJump = false;
        }
    }
    else if (SwipeController.swipeDown)
    {
        if (IsGrounded() || !_isJumping)
        {
            PerformFlip();
            _isJumping = true;
        }
    }

    if (IsGrounded() && !_roll)
        _animator.SetBool("isRun
    ning", true); else
        _animator.SetBool("isRunning",  false);

    Vector3 targetPosition = transform.position.z * transform.forward + transform.position.y *
    transform.up; if (_lineToMove == 0)
        targetPosition += Vector3.left *
    _lineDistanse; else if (_lineToMove
    == 2)
        targetPosition += Vector3.right * _lineDistanse;

    if (transform.position ==
        targetPosition) return;
    Vector3 diff = targetPosition - transform.position;
    Vector3 moveDir = diff.normalized * 25 *
    Time.deltaTime; if (moveDir.sqrMagnitude <
    diff.sqrMagnitude)
        _rigidbody.MovePosition(transform.position
    + moveDir); else
        _rigidbody.MovePosition(targetPosition);
}
private void FixedUpdate()
{
    _rigidbody.AddForce(Physics.gravity * (_gravityScale - 1) * _rigidbody.mass);
}

private void Jump()
{
    _soundManager.JumpSound();
    _rigidbody.AddForce(Vector3.up * _jumpSpeed, ForceMode.Impulse);

    if (_rigidbody.velocity.y >= 0)
    {
        _currentGravityScale = _gravityScale;
        _animator.SetTrigger("Jump");
        _animator.SetBool("isRunning",  false);
    }
    else if (_rigidbody.velocity.y < 0)
    {
        _currentGravityScale = _fallingGravityScale;
        _animator.SetTrigger("Jump");
        _animator.SetBool("isRunning",  false);
    }

    _isJumping = true;
}

private void PerformFlip()
{
    if (!_isFlipping)
```

```
    {
        _isFlipping = true;
        _animator.SetTrigger("Flip");
        _rigidbody.velocity = Vector3.up * -1.0f;
        _rigidbody.AddForce(Vector3.up * _flipForce,
        ForceMode.Impulse); StartCoroutine(ResetFlipState());
    }
}

private IEnumerator ResetFlipState()
{
    yield return new WaitForSeconds(0.2f);
    _isFlipping = false;
    _isJumping = false;
}

private bool IsGrounded()
{
    return true;
}

private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "obstacle")
    {
        Time.timeScale = 0;
        _soundManager.GameOver();
        _tryCount++;
        PlayerPrefs.SetInt("tryCount",
        _tryCount); if (_tryCount % 4
        == 0)
            _adsService.TryShowInterstitialAds();
        _losePanel.SetActive(true);
        int lastRunScore =
        int.Parse(Score._scoreText.text.ToString());
        PlayerPrefs.SetInt("lastRunScore", lastRunScore);
    }
    else if (collision.gameObject.tag == "Ground")
    {
        _canJump = true;
    }
}

private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.tag == "Coin")
    {
        _coins++;
        _soundManager.PickupCoins();
        _coinsText.text             =
        _coins.ToString();
        PlayerPrefs.SetInt("coin
        s",                    _coins);
        Destroy(other.gameObje
        ct);
    }

    if (other.gameObject.tag == "BonusStar")
    {
        StartCoroutine(MoveFa
        sterTiles());
        Destroy(other.gameObj
        ect);
```

```
        }
    }

    private IEnumerator MoveFasterTiles()
    {
        _tileGenerator._tileSpeed
        *= 2.0f; yield return new
        WaitForSeconds(7.0f);

        _tileGenerator._tileSpeed /= 2.0f;
    }
}
private IStoreController _storeController;
    private IExtensionProvider
    _storeExtensionProvider; private
    IAppleExtensions _appleExtensions;
    private Dictionary<PurchaseProductType, Product> _products =
    new(); public Dictionary<PurchaseProductType, Product>
    Products => _products; private UniTaskCompletionSource _tcs;

    private bool IsPurchasingInited()
    {
        return _storeController != null && _storeExtensionProvider != null;
    }

    public bool IsInited =>

    IsPurchasingInited(); public async

    UniTask InitializePurchasing()
    {
        if (IsPurchasingInited())
        {
            return;
        }
        _tcs = new UniTaskCompletionSource();

        var builder = ConfigurationBuilder.Instance(StandardPurchasingModule.Instance());

        builder.AddProduct(kGoogleNoAdsProductId,
        ProductType.NonConsumable);
        builder.AddProduct(kCharacterGrandfather,
        ProductType.NonConsumable); builder.AddProduct(kCharacterCJ,
        ProductType.NonConsumable);
        builder.AddProduct(kCharacterHotdog,
        ProductType.NonConsumable);


        UnityPurchasing.Initialize(this,  builder);

        await _tcs.Task;
    }

    public void RestorePurchases(Action<bool> callback)
    {
        _appleExtensions.RestoreTransactions((success, message) =>
        {
            if (success)
            {
                Debug.Log("Restore purchases succeeded!" +
                message); callback?.Invoke(true);
            }
            else
```

```
            {
                Debug.Log("Restore purchases failed!" + message);
                callback?.Invoke(false);
    }
});
        }

        public Product GetProduct(PurchaseProductType product)
        {
            if (!IsPurchasingInited() || !_products.ContainsKey(product))
            {
                return default;
            }
            return _products[product];
        }

        public void BuyProduct(PurchaseProductType productType)
        {
            if (IsPurchasingInited())
            {
                var product = _products[productType];
                Debug.LogFormat("Product id called: {0}",
                product.definition.id);

                if (product != null && product.availableToPurchase)
                {
                    _storeController.InitiatePurchase(product);
                }
                else
                {
                    Debug.LogErrorFormat("Purchase of {0} failed", product.definition.id);
                }
            }
            else
            {
                Debug.LogErrorFormat("Purchase failed, service not initialized!");
            }
        }

        public Dictionary<string, string> GetAppleIntroductoryPriceDictionary()
        {
            if (!IsPurchasingInited())
            {
                return default;
            }
            return _appleExtensions.GetIntroductoryPriceDictionary();
        }

        public void OnInitialized(IStoreController controller, IExtensionProvider extensions)
        {
            _storeController = controller;
            _storeExtensionProvider = extensions;
            _appleExtensions = extensions.GetExtension<IAppleExtensions>();

            foreach (var product in controller.products.all)
            {
                if (!product.availableToPurchase)
                {
                    Debug.LogWarning($"Product not available for purchase {product.definition.id}");
                }
            }

            OnCompleteInitialization(controller.products.all);
```

```
    }

    public void OnInitializeFailed(InitializationFailureReason error)
    {
        _tcs.TrySetResult();
        Debug.LogError("OnInitializeFailed InitializationFailureReason:" + error);
    }

    public void OnInitializeFailed(InitializationFailureReason error, string message)
    {
        _tcs.TrySetResult();
        Debug.LogError("OnInitializeFailed InitializationFailureReason:" + error);
    }

    public void OnPurchaseFailed(Product product, PurchaseFailureReason failureReason)
    {
        ON_PURCHASE_CALLBACK?.Invoke(false, product);
        Debug.LogErrorFormat("Purchase Failed for {0} because of {1}", product,
        failureReason);
    }

    public void OnPurchaseFailed(Product product, PurchaseFailureDescription failureDescription)
    {
        ON_PURCHASE_CALLBACK?.Invoke(false,  product);
        Debug.LogErrorFormat("Purchase Failed for {0} because of {1}", product, failureDescription.message);
    }

    public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)
    {
        var validPurchase = true;
#if !UNITY_EDITOR && RECEIPT_VALIDATION
            var validator = new CrossPlatformValidator(GooglePlayTangle.Data(), AppleTangle.Data(),
Application.identifier);
            try
            {
                var result =
                validator.Validate(args.purchasedProduct.receipt);
                Debug.Log("Receipt valid");
            }
            catch (IAPSecurityException)
            {
                Debug.LogError("Validation failed");
                Debug.Log("Invalid receipt, not unlocking
                content");

                validPurchase = false;
            }
#endif
        if (validPurchase)
        {
            ON_PURCHASE_CALLBACK?.Invoke(true, args.purchasedProduct);
            Debug.LogWarningFormat("Purchase process for {0} is valid", args.purchasedProduct);
        }
        else
        {
            ON_PURCHASE_CALLBACK?.Invoke(false, args.purchasedProduct);
            Debug.LogWarningFormat("Purchase process for {0} is not valid", args.purchasedProduct);
        }

        return PurchaseProcessingResult.Complete;
    }

    private void OnCompleteInitialization(Product[] allProducts)
```

```
    {
        for (int i = 0, j = allProducts.Length; i < j; i++)
        {
            if (!allProducts[i].availableToPurchase)
            {
                Debug.LogWarning($"Product not available for purchase {allProducts[i].definition.id}");
            }

            var product = allProducts[i];
            if (!_products.ContainsValue(product))
            {
                MapProducts(product.definition.id,   product);
            }
        }
        _tcs.TrySetResult();
    }

    private void MapProducts(string id, Product product)
    {
        switch (id)
        {
            case kGoogleNoAdsProductId:
                _products.Add(PurchaseProductType.No_a
                ds, product); break;
            case kCharacterGrandfather:
                _products.Add(PurchaseProductType.Moncl
                er, product); break;

            case kCharacterCJ:
                _products.Add(PurchaseProductType.
                CJ, product); break;
            case kCharacterHotdog:
                _products.Add(PurchaseProductType.Age
                nt, product); break;
            default:
                Debug.LogWarningFormat("Product {0} with id: {1} - was not added to products, because of
target platform", product.definition.id, id);
                break;
        }
    }
}

public delegate void PurchaseCallback(bool success, Product

product); public enum PurchaseProductType
{
    none = 0,
    Moncler = 9,
    CJ = 8,
    Agent = 7,
    No_ads = 10
}

public enum Characters
{
    Mike = 0,
    Chyb = 1,
    FireMan = 2,
    MadaraUchiha = 3,
    RAMPAGE = 4,
    Mira = 5,
    Millionaire = 6,
    Agent = 7,
```

CJ = 8,
Moncler = 9,
No_ads = 10,