Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *Каракая Владислава Сергійовича*
(ПІБ)

академічної групи *122-20-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка комп'ютерної гри-платформера Bouncing-Ball на базі Unity Game Engine та мови програмування C#*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Бердник М.Г.*			
розділів:				
спеціальний	*проф. Бердник М.Г.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Я

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *122-20-3* (група) *Каракая Владислава Сергійовича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка комп'ютерної гри-платформера Bouncing-Ball на базі Unity Game Engine та мови програмування C#*

затверджена наказом ректора НТУ «ДП» від 23.05.2024 р. № 469-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видала (підпис) *проф. Бердник М.Г.* (посада, прізвище, ініціали)

Завдання прийняла до виконання (підпис) *Каракай В.С.* (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 80 с., 21 рис., 3 дод., 25 джерел.

Об'єктом дослідження є мобільний ігровий додаток у жанрі платформер Bouncing-Ball, створений на базі Unity Game engine та мови програмування C#

Мета кваліфікаційної роботи: розробка комп'ютерної гри-платформера Bouncing-Ball на базі Unity Game Engine та мови програмування C#, яка б забезпечила захоплюючий ігровий процес та відповідала сучасним вимогам до якості графіки та продуктивності.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовано наявні ігрові рішення, обрано платформу для розробки, виконано проектування та створення гри в жанрі платформера. Детально описано роботу гри, її алгоритми та структуру функціонування, а також процес запуску та завантаження. Визначено вхідні і вихідні дані, а також охарактеризовано технічні параметри необхідного обладнання.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню додатку та розраховано час на його створення.

Практичне значення полягає у створенні комп'ютерної гри, яка забезпечує захоплюючий ігровий досвід, сприяє розвитку навичок програмування та дизайну ігор, а також демонструє можливості використання Unity для створення якісного інтерактивного контенту.

Актуальність розробки комп'ютерної гри в жанрі платформера є значною для сучасної індустрії розваг. Мобільні ігри користуються високим попитом серед користувачів, що сприяє їхньому широкому розповсюдженню та популярності. В умовах стрімкого розвитку цифрових технологій та зростання кількості геймерів у всьому світі, створення якісних ігор стає важливим аспектом економічного та культурного розвитку, відкриваючи нові можливості для бізнесу та креативної індустрії.

Список ключових слів: ANDROID, ІГРОВИЙ РУШІЙ, UNITY, ПЛАТФОРМЕР, ІНТЕРАКТИВНИЙ КОНТЕНТ, SDK, ГЕЙМДИЗАЙН, C#

ABSTRACT

Explanatory Note: 80 pages, 21 figures, 3 appendices, 25 sources.

The research object is a mobile gaming application in the platformer genre, Bouncing-Ball, created using the Unity Game engine and the C# programming language.

The objective of the qualification work is to develop a platformer computer game, Bouncing-Ball, based on the Unity Game Engine and the C# programming language, which would provide an engaging gameplay experience and meet modern requirements for graphics quality and performance.

The introduction covers the analysis and current state of the problem, specifies the goal of the qualification work and its application field, provides justification for the relevance of the topic, and clarifies the task statement.

The first chapter analyzes the subject area, defines the relevance of the task and the purpose of the development, formulates the task statement, and specifies the requirements for the software implementation, technologies, and software tools.

The second chapter analyzes existing gaming solutions, selects the platform for development, and performs the design and creation of the platformer game. It details the game's operation, its algorithms and functional structure, as well as the process of launching and loading. It specifies the input and output data, and characterizes the technical parameters of the necessary equipment.

The economic section determines the labor intensity of the developed information system, calculates the cost of work for creating the application, and estimates the time required for its creation.

The practical significance lies in creating a computer game that provides an engaging gaming experience, fosters the development of programming and game design skills, and demonstrates the potential of using Unity to create high-quality interactive content.

The development of a platformer computer game is highly relevant to the modern entertainment industry. Mobile games are in high demand among users, contributing to their widespread distribution and popularity. In the context of rapid digital technology development and the increasing number of gamers worldwide, creating quality games becomes an important aspect of economic and cultural development, opening new opportunities for business and the creative industry.

List of keywords: ANDROID, GAME ENGINE, UNITY, PLATFORMER, INTERACTIVE CONTENT, SDK, GAME DESIGN, C#

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SRP – Single Responsibility Principle;

SDK – Software Development Kit;

JDK – Java Development Kit;

NDK – Native Development Kit;

RP – Render Pipeline;

ПЗ – програмне забезпечення;

ОС – операційна система;

UI – User interface;

ВСТУП

Мобільні ігри зараз знаходяться на передовій сучасного світу. Розвиток технологій впливає на всі існуючі напрямки і індустрія розробки ігор не є винятком , це призводить до : зменшення розмірів портативних пристроїв, збільшення продуктивності всіх електронних пристроїв . Але, нові технології - це не єдине, що робить комп'ютерні ігри, такими унікальними. Велика перевага перед іншим програмним забезпеченням - це унікальність спектру емоцій , що відчуває кожна людина , адже гра в першу чергу має пробуджувати його різноманітні форми будь то азарт чи бажання перемогти.

Розробка гри в жанрі платформера, такої як Bouncing-Ball, є важливою та актуальною задачею. Такі ігри забезпечують гравців захоплюючим ігровим процесом, де вони повинні вирішувати різноманітні завдання, долати перешкоди та досягати поставлених цілей. Основні особливості платформерів включають:

- створення неповторного досвіду;
- завчасно запрограмовані процеси та розгортання подій ;
- наявність основної механіки;
- естетичне відчуття, графічний вид тощо.

Ціллю кваліфікаційної роботи є створення гри Bouncing-Ball в жанрі платформера, яка ставить перед гравцем кілька серйозних викликів:

- пройти рівні гри, уникаючи перешкод;
- дослідити нові рівні та механіки;
- досягти максимальної кількості очок;
- завершити гру у найкоротший час.

Результатом виконання даної кваліфікаційної роботи є програмне забезпечення, яке створює всі вищезазначені виклики перед користувачем, забезпечуючи захоплюючий ігровий процес та високу якість виконання.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

У загальному контексті, гра є інтерактивним програмним забезпеченням, створеним для розваги, навчання або занурення у віртуальний світ. Гра поєднує в собі різні елементи, такі як геймплей, сюжет, графіку та звук, щоб створити захоплюючий та інтерактивний досвід для користувача.

Основні характеристики гри включають:

1. Ігровий процес (геймплей) – сукупність правил, механік та дій, які визначають, як гравець взаємодіє з грою. Це може включати пересування персонажа, виконання завдань, розгадування головоломок та боротьбу з ворогами.
2. Сюжет – історія або сценарій, що супроводжує ігровий процес. Сюжет може бути простим або складним, але завжди має на меті залучити гравця та надати контекст для його дій у грі.
3. Графіка – візуальне оформлення гри, включаючи дизайн персонажів, оточення та ефектів. Якісна графіка робить гру привабливою та допомагає створити потрібну атмосферу.
4. Звук - аудіо елементи гри, такі як музика, звукові ефекти та діалоги, що доповнюють візуальний досвід та сприяють глибшому зануренню гравця у гру.
5. Інтерактивність – можливість гравця впливати на ігровий світ та змінювати його своїми діями. Інтерактивність є ключовим аспектом, що відрізняє ігри від інших видів розваг, таких як фільми чи книги [2].

Ігри можуть бути як простими, так і складними, пропонуючи різні рівні викликів та завдань. Вони можуть розроблятися для різних платформ, включаючи персональні комп'ютери, консолі, мобільні пристрої та віртуальну реальність, задовольняючи потреби широкої аудиторії гравців.

Як було сказано вище, ігри можуть бути різними та мають неймовірний спектр різноманіття як у жанровому контексті, так і в загальному, оскільки навіть представники одного жанру можуть сильно відрізнятися та використовувати різну ідейність

Аналізуючи існуючі ігрові додатки, що є представниками жанру платформерів, ми можемо зрозуміти, які елементи гри найкраще сприймаються гравцями і є ключовими для успіху таких ігор. Наприклад, Super Mario Bros. від Nintendo відома своїми простими, але захоплюючими рівнями, які постійно викликають інтерес до подальшого прогресу. Головний герой, Маріо, володіє інтуїтивно зрозумілими механіками стрибків та збору предметів, що стимулює гравців до дослідження різноманітних шляхів проходження рівнів.

Sonic the Hedgehog від Sega виділяється швидким темпом ігрового процесу і унікальним дизайном рівнів. Ця гра надає гравцям можливість насолоджуватися динамічними перегонами з чудово розробленими механіками, що дозволяють виконувати вражаючі трюки і обходити перешкоди з ефектом швидкості, що зберігає напруження гри від початку до кінця.

Rayman від Ubisoft привертає увагу гравців своєю барвистою графікою та оригінальним стилем. Гра відзначається яскравими малюнками і великим різноманіттям ігрових локацій, що включають у себе як казкові, так і фантастичні елементи. Цей підхід створює неповторну атмосферу, яка захоплює гравців і залишає незабутні враження.

Ці ігри демонструють, що ключовими компонентами успіху у жанрі платформерів є інтуїтивно зрозумілі механіки, цікаві й різноманітні рівні, які викликають бажання вирішувати головоломки та подолати виклики, а також відмінна графіка і приємний звуковий супровід, що додає глибини імерсії в ігровий процес.

Сучасні ігрові двигуни [8], такі як Unity, значно полегшують процес розробки платформерів, надаючи розробникам потужні інструменти для створення складних ігрових механік та візуальних ефектів. Unity відкриває перед

творцями ігор безліч можливостей, що дозволяють створювати інноваційні проекти з різноманітними ігровими концепціями та арт-стилями.

Платформа підтримує багато платформний розвиток, що означає можливість розробляти ігри для ПК, мобільних пристроїв, консолей та віртуальної реальності. Це значно розширює аудиторію і покриття ігрового продукту, забезпечуючи доступність гри на різних платформах і для різних типів гравців.

Проте, незважаючи на значний прогрес, існують певні виклики, які стикаються розробники. Одним з основних викликів є оптимізація продуктивності гри на різних платформах. Різні пристрої мають різні обмеження щодо ресурсів, і важливо забезпечити, щоб гра працювала ефективно на всіх цих платформах, не втрачаючи якості геймплею і візуальних ефектів.

Іншим викликом є забезпечення стабільності роботи гри. На різних пристроях можуть виникати різні проблеми, такі як затримки чи артефакти, які впливають на загальний досвід гравця. Розробники повинні активно тестувати гру на різних пристроях і вчасно виправляти всі виявлені проблеми, щоб забезпечити якісний продукт.

Окрім того, створення інтуїтивно зрозумілого користувацького інтерфейсу також є важливим аспектом розробки гри на Unity [5]. Чітке та легко відтворюване управління, зрозумілі інструкції та інтерактивні елементи дозволяють гравцям швидко засвоювати механіку гри і насолоджуватися ігровим процесом без зайвих труднощів.

Отже, Unity інтегрує усі необхідні компоненти для створення високоякісних платформерів, проте успіх вимагає від розробників не лише технічної експертності, але й уважності до деталей і бажання досягати найвищих стандартів якості.

1.2. Призначення розробки та область застосування

Цільовою аудиторією моєї комп’ютерної гри є молодше покоління , більшою мірою чоловіки від 10 до 25 років, які активно цікавляться іграми та шукають нові виклики і розваги. Гра пропонує користувачам можливість випробувати свої сили через підвищення складності на кожному рівні, що стимулює їх до активної ігрової діяльності та викликає почуття потоку - максимального зосередження і задоволення від гри.

Гра розроблена для широкої аудиторії, оскільки доступна на мобільних пристроях з операційною системою Android. Це робить її доступною для гравців у будь-який час і в будь-якому місці [11]: вдома, під час подорожі на транспорті, в парку або навіть на прогулянці по вулиці. Гра пропонує геймерам можливість насолоджуватися ігровим процесом в будь-якому зручному середовищі, де вони можуть знайти час для розваг і відпочинку.

Головною особливістю є те, що гра дозволяє користувачам зануритися у захоплюючий світ ігрового процесу без обмежень місця і часу, що робить її ідеальною для сучасного геймера, що веде активний спосіб життя і цінує можливість грати в будь-яких умовах.

1.3. Підстава для розробки

Підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма спеціальності 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- затверджена наказом ректора НТУ «ДП» від 23.05.2024 № 469-с;
- завдання на кваліфікаційну роботу на тему: «Розробка комп'ютерної гри Bouncing-Ball в жанрі платформера на базі Unity Game engine та мови програмування C#»;

1.4. Постановка задачі

У даній кваліфікаційній роботі необхідно розробити комп’ютерну гру в жанрі платформера, за допомогою мови програмування C#. Програмне забезпечення повинно відповідати сучасним вимогам та дозволяти розробникам легко додавати новий функціонал та механіки. Код гри має відповідати принципам чистого коду, бути добре структурованим, з корисними назвами змінних та функцій, що забезпечить легкість у його розумінні та подальшому обслуговуванні.

Основні механіки гри:

1. М'яч з властивістю надпригучості.
2. Механіка руху по платформі.
3. Система зчитування дій (свайпи).
4. Логіка ігрового рівня.
5. Набір рівнів:
   - розробка унікальної структури;
   - моделювання складності.
6. Ігрова валюта.
7. Збереження даних.
8. Магазин.
9. Налаштування:
   - зміна звукового супроводу;
   - чутливості екрану.
10. Додаткова ігрова логіка:
    - пауза;
    - перезавантаження рівня;
    - повернення на початковий екран.
11. Візуальна частина:
    - моделі для ігрових рівнів;

- між-рівневі переходи;
- екрани завантаження.

13. Звукове супроводження.

12. Реклама.

Програмне забезпечення повинно бути реалізоване як мобільний додаток у форматі APK для платформи Android, з використанням мови програмування C# та інструменту Unity.

1.5. Вимоги до програми або програмного виробу

Вимоги до програми включають: стабільність роботи на платформі Android, ефективність використання ресурсів пристрою, інтуїтивно зрозумілий користувацький інтерфейс, плавний ігровий процес, високу якість графіки та звукового супроводу, можливість розширення функціоналу та регулярне оновлення контенту.

1.5.1. Вимоги до функціональних характеристик

Взаємодія з грою має відбуватися за рахунок стандартних інструментів введення даних, тобто за допомогою сенсорного екрану. Процес введення здійснюватиметься через розроблений програмний інтерфейс, де основними елементами будуть кнопки, що дозволяють гравцеві впливати на ігровий процес або взаємодіяти з додатковою ігровою функціональністю.

1.5.2. Вимоги до інформаційної безпеки

Забезпечення інформаційної безпеки є критично важливим аспектом у розробці комп’ютерних ігор, особливо в контексті мобільних додатків для Android.

Вимоги до інформаційної безпеки включають:

1. Захист особистих даних користувачів – гра повинна забезпечувати конфіденційність особистих даних користувачів, таких як ім’я, адреса електронної пошти, вік тощо. Всі такі дані повинні бути шифровані відповідно до сучасних стандартів безпеки.
2. Захист від несанкціонованого доступу – розробка повинна включати механізми для захисту від несанкціонованого доступу до ігрових ресурсів та приватних даних користувачів. Це може включати використання автентифікації користувача, паролів, а також мережевих інтерфейсів з шифруванням даних.
3. Виявлення і усунення вразливостей – періодична перевірка безпеки коду гри для виявлення потенційних вразливостей і вразливих місць. Всі виявлені уразливості повинні бути негайно виправлені через оновлення програмного забезпечення.
4. Відповідність вимогам платформи Android – гра повинна відповідати вимогам безпеки, встановленим платформою Android, таким як політика обмеження дозволів, захист від шкідливих програм і безпечні механізми зберігання даних.
5. Аналіз і обробка помилок – розробка повинна включати систему для збору, аналізу і обробки помилок, що дозволить вчасно виявляти та виправляти проблеми, що стосуються безпеки.
6. Резервне копіювання і відновлення даних – механізми для регулярного резервного копіювання даних гравців і можливість швидкого відновлення ігрового прогресу у випадку втрати даних або інциденту з безпекою.

Здійснення цих вимог допоможе забезпечити високий рівень захисту інформації та збереже довіру користувачів до гри, зменшуючи ризики порушення безпеки та негативних наслідків.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для забезпечення стабільності мобільних пристроїв рекомендовані такі характеристики:

- Кількість ядер: не менше 6;
- Частота процесору: 1.8 ГГц або вище;
- Оперативна пам'ять: 3 ГБ або більше;
- Оперативна система: Android 7 (Nougat) або новіше;

1.5.4. Вимоги до інформаційної та програмної сумісності

У якості основної мови програмування для розробки гри було обрано C#. Ця мова є основною для інструменту Unity, що значно спрощує процес створення ігор для різних платформ і операційних систем.

Unity надає можливість розробляти програмне забезпечення для широкого спектру платформ, що включає ПК, мобільні пристрої та ігрові консолі. Це робить ігровий процес доступним для широкого кола користувачів і забезпечує гнучкість у виборі платформи для випуску гри [10].

Для зручності розробки було обрано середовище розробки Rider, яке відоме своїм багатим функціоналом та спрощує роботу розробника, забезпечуючи ефективне управління проектами і підтримку ключових аспектів розробки ігор.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Оскільки основою цієї кваліфікаційної роботи було створення гри, то основною функцією є покращення емоційного стану людини, даруючи найяскравіші емоції під час гри [3]. Важливою деталлю цього завдання є доступність гри як технічно (завдяки оптимізації, що дозволяє грати навіть на слабкому пристрої), так і фізично (через підтримку гри на телефонах і можливість грати без підключення до Інтернету). Це дозволяє кожному гравцеві насолоджуватися грою незалежно від місця і часу.

На практичному рівні основний функціонал представляє Game Loop, який включає в себе основну ігрову складову. У випадку платформерів і з урахуванням особливостей нашого проекту мів може бути представлений у вигляді простої схеми : гравець поступово проходить рівень, що покращує його майстерність. Зі збільшенням майстерності рівні починають набувати нових видів та отримують нові доповнення, які вимагають від гравця адаптації та розвитку своїх навичок [25] (рис. 2.1) . Паралельно цей процес супроводжує додаткова мотивація у вигляді монет, що не є обов’язковою, але при цьому надає ігровому процесу більшого емоційного забарвлення.

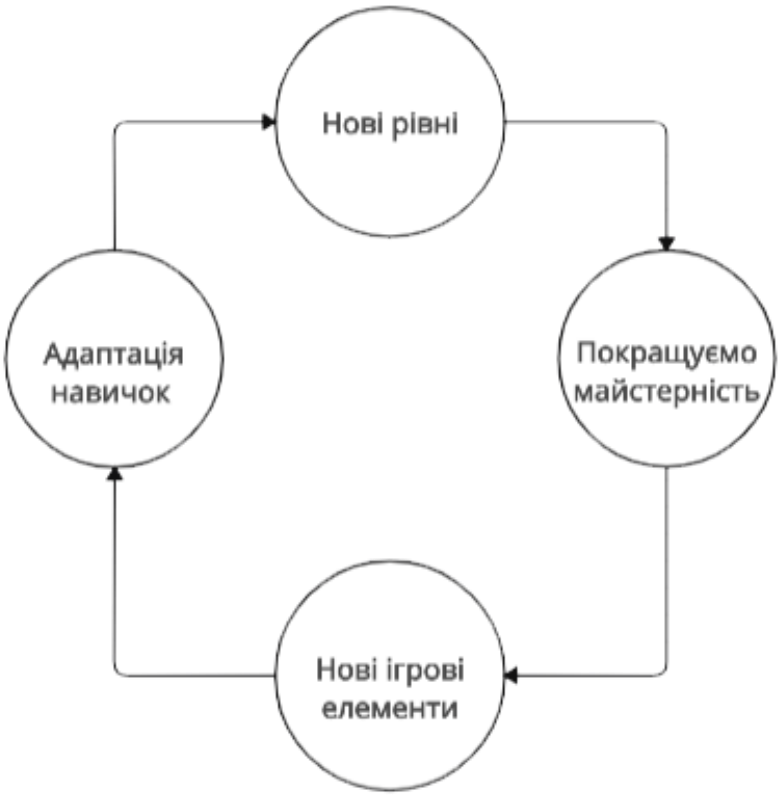

Рис. 2.1. Game Loop

2.2. Опис застосованих математичних методів

У розробці гри жанру платформер було застосовано декілька важливих математичних методів для реалізації фізики об'єктів та забезпечення реалістичності ігрового процесу.

Лінійна інтерполяція (Lerp) : для плавної зміни масштабу та обертання об'єкта використовується лінійна інтерполяція (Lerp). Цей метод дозволяє поступово змінювати значення однієї величини до іншої за певний проміжок часу, забезпечуючи гладкий перехід та уникнення різких змін у вигляді та поведінці об'єктів.

Векторна математика : в моєму випадку застосовується для обчислення напрямку та величини швидкості об'єкта. Використовуючи вектори, можна

точно визначити, в якому напрямку рухається об'єкт і з якою швидкістю, що є ключовим для створення реалістичних фізичних взаємодій та анімацій.

Кватерніони : під час обертання об'єктів у просторі використовуються кватерніони. Вони дозволяють уникнути проблем, пов'язаних із кардинальними сингулярностями (gimbal lock), які можуть виникати при використанні традиційних методів обертання (наприклад, кутів Ейлера). Кватерніони забезпечують плавні та безперервні обертання об'єктів у тривимірному просторі.

Проекція векторів : проекція вектора використовується для визначення складових швидкості об'єкта на різних осях. Це дозволяє оцінити, наскільки сильно об'єкт взаємодіє з поверхнями під час зіткнень, що є важливим для моделювання відскоків та деформацій.

Математичні функції : математичні функції, такі як Mathf.Clamp, застосовуються для обмеження значень параметрів у заданих межах. Це допомагає контролювати максимальні і мінімальні значення змінних, таких як масштаб об'єкта, запобігаючи виходу за допустимі межі.

Кінематичні рівняння : використовуються для моделювання руху об'єктів під впливом сили тяжіння та інших сил. Це включає обчислення нових позицій та швидкостей об'єктів на основі їхніх початкових параметрів і прикладених сил.

У висновку комбінація цих математичних методів дозволяє створити реалістичну фізику об'єктів у грі, забезпечити плавні анімації та точні взаємодії між об'єктами. Це робить ігровий процес більш захоплюючим і привабливим для гравців, створюючи відчуття реалістичності та занурення у віртуальний світ.

2.3. Опис використовуваної архітектури та шаблонів проектування

Оскільки для створення нашого ігрового додатку було використано ігровий двигун Unity, ми будемо говорити про комбінацію архітектури та шаблонів [4], яку запроваджує сам Unity, а також наші особисті рішення як розробника.

На початку розглянемо Unity в контексті його переваг як двигуна та його архітектурну частину.

Для нашої гри, яка буде реалізована як мобільний додаток для платформи Android, Unity є ідеальним вибором. Движок дозволяє легко адаптувати гру для різних мобільних пристроїв, забезпечуючи доступ до широкої аудиторії.

Нашій грі потрібна висока продуктивність для забезпечення плавного ігрового процесу та якісної графіки. Unity оптимізовано для мобільних пристроїв, що дозволяє створювати візуально привабливі та добре оптимізовані ігри.

Мова програмування C# є основною для Unity, що відповідає нашим вимогам до розробки. Знання C# допоможе створити чистий та зрозумілий код [1], що полегшить підтримку та розширення функціональності гри в майбутньому.

Unity надає власний фізичний движок, що значно спрощує реалізацію фізичних взаємодій у грі. Це дозволяє нашій грі мати реалістичну фізику руху, стрибків та зіткнень, що є критично важливим для жанру платформерів. Також Unity має власний рендеринг-модуль [12], який забезпечує високу якість візуальних ефектів. Для нашої гри це означає можливість створення красивої графіки та деталізованих ігрових середовищ, що підвищить загальне враження від гри для користувачів.

Багато з вище зазначених пунктів вказують на наявність всередині Unity важливих архітектурних рішень , які лежать в основі його оптимізації та багатьох вбудованих функціональних можливостей.

1. Ігровий цикл (Game Loop) – поняття гри не може існувати без поняття ігрового циклу , цей фундаментальний архітектурний шаблон забезпечує безперервне оновлення та рендеринг ігрових сцен.
2. Наглядач (Observer) – Unity використовує цей шаблон для реалізації системи подій та підписки. Об'єкти можуть підписуватися на події та реагувати на них, що спрощує управління станом гри та взаємодією між об'єктами.

3. Одинак (Singleton) – Шаблон Singleton використовується в Unity для створення глобальних об'єктів, які мають лише один екземпляр. Це дозволяє забезпечити єдиний доступ до глобальних ресурсів, таких як менеджери ресурсів або інші глобальні налаштування.
4. Пристосуванець Flyweight – В контексті Unity, шаблон Flyweight використовується для оптимізації роботи з ресурсами. Наприклад, система ресурсів (assets) і їх використання в різних частинах гри дозволяє ефективно керувати і спільно використовувати однакові ресурси, зменшуючи використання пам'яті.
5. Entity-Component-System (ECS) – Цей шаблон є фундаментальним для організації об'єктів у грі. Unity використовує ECS для розподілу поведінки об'єктів на компоненти (components) і сутності (entities), що дозволяє ефективно керувати великою кількістю об'єктів та їх взаємодією.

Також говорячи про архітектуру Unity не можливо уникнути поняття GameObject який є основою компонентної моделі та слугує початком (Entry Point) будь-якої гри (рис. 2.2).

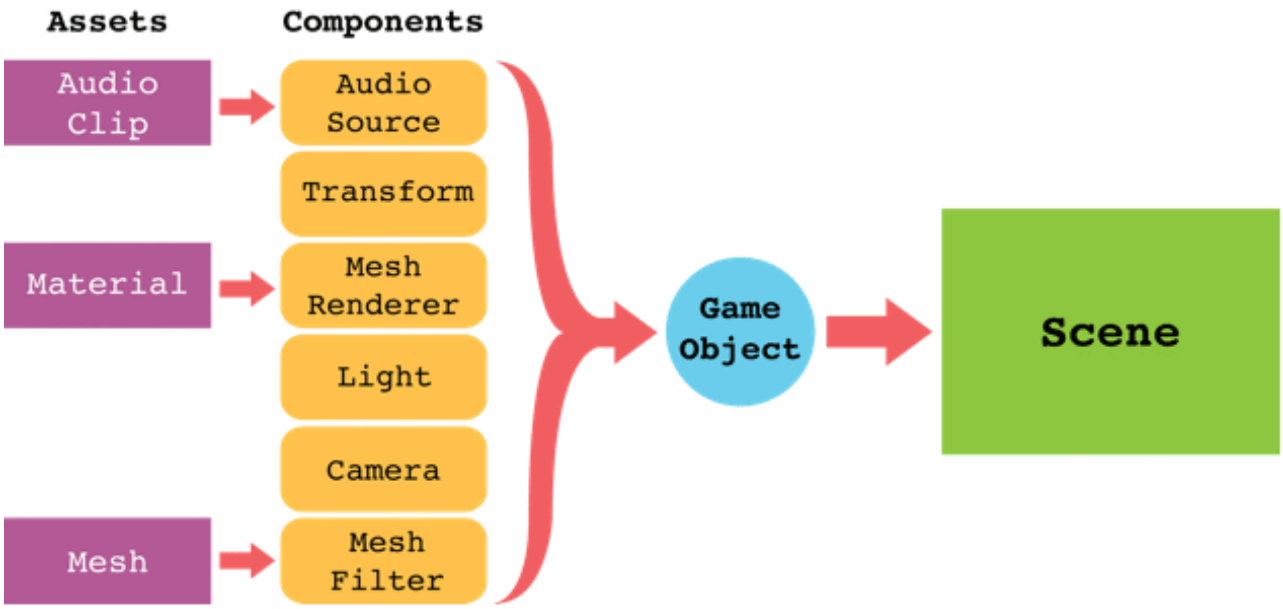

Рис. 2.2. Компонентна модель об’єктів на сцені

Повертаючись до вищезазначених шаблонів, маємо звернути увагу, що шаблон Game Loop [22] цікавить нас не тільки в контексті внутрішньої роботи ігрового двигуна. Ми також хочемо мати доступ до різної поведінки цього циклу та впроваджувати туди необхідну логіку. В основі вирішення цієї потреби стоїть клас MonoBehaviour , саме він є умовним провайдером цього шаблону (рис. 2.3).

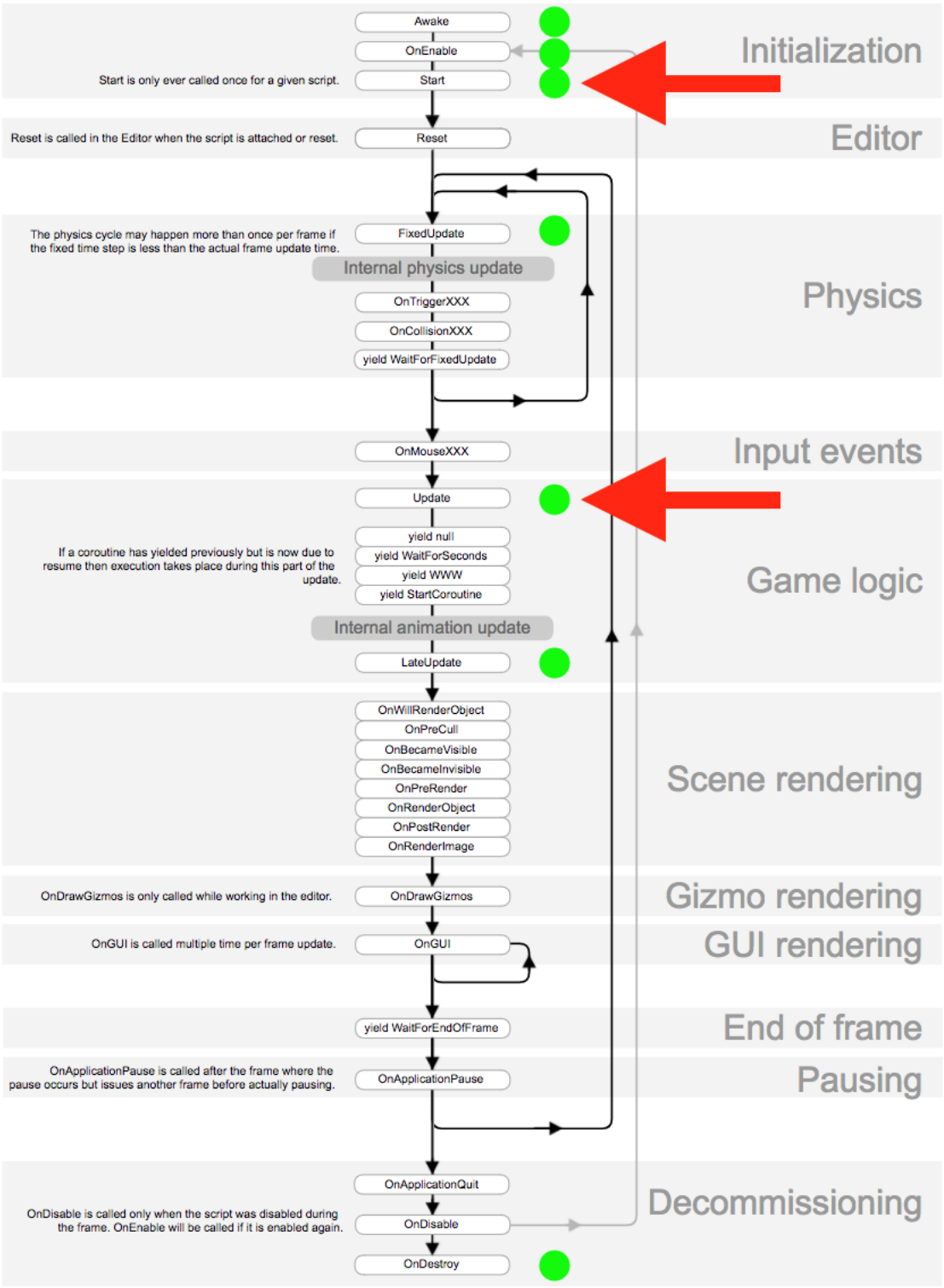

Рис. 2.3. Ігровий цикл MonoBehavior компоненту

Основі функції нашого циклу :

1. Awake – викликається один раз, коли об'єкт створюється. Використовується для ініціалізації стану об'єкта і знаходження посилань на інші компоненти.
2. Start – викликається перед першим кадром оновлення. Використовується для налаштування і початкового стану компонента.
3. Update – викликається кожен кадр для виконання логіки, що потребує частого оновлення, наприклад, переміщення об'єктів або обробка введення.
4. FixedUpdate – викликається на кожному фіксованому кроці фізичного двигуна Unity. Використовується для всіх операцій фізики, таких як керування фізичними об'єктами.
5. LateUpdate – Викликається після оновлення всіх інших об'єктів в поточному кадрі. Використовується для виконання логіки після оновлення стану інших об'єктів, наприклад, слідкування за об'єктами або камерою.
6. OnEnable – Викликається, коли компонент стає активним. Використовується для налаштування підписок на події або інших динамічних налаштувань.
7. OnDisable – Викликається, коли компонент вимикається. Використовується для відписки від подій або звільнення ресурсів, щоб запобігти витоку пам'яті.

Архітектура проекту у контексті розробки програмного забезпечення визначає структуру [14], організацію та взаємозв'язки між компонентами програми. Це план або дизайн, який визначає, як система буде розбита на частини, як вони будуть взаємодіяти між собою, і як будуть вирішуватися основні архітектурні питання, такі як розширюваність, підтримка, ефективність та зручність для розробки.

Запроваджена архітектура проекту в своїй основі спирається на провайдери даних [24], які фактично повинні представляти собою сервісну ідею розподілу функціональних можливостей окремих елементів програми. Це дає змогу

керуватися принципами SRP та інкапсуляції, що, в свою чергу, дозволяють зберігати "чисту" архітектуру.

Для реалізації даної архітектури дуже важливо забезпечити правильну ініціалізацію всіх даних [19] та мати можливість безпечно отримувати та маніпулювати ними (рис. 2.4). Для цього було створено окремий компонент який зберігає в собі список всіх систем даних та проводить їх ініціалізацію в потрібному порядку .

Для забезпечення максимально коректної логіки Entry Point було створено окрему ігрову сцену , яка на меті мала єдину задачу – створити необхідну точно входу та почати ініціалізації всіх даних (рис. 2.5).

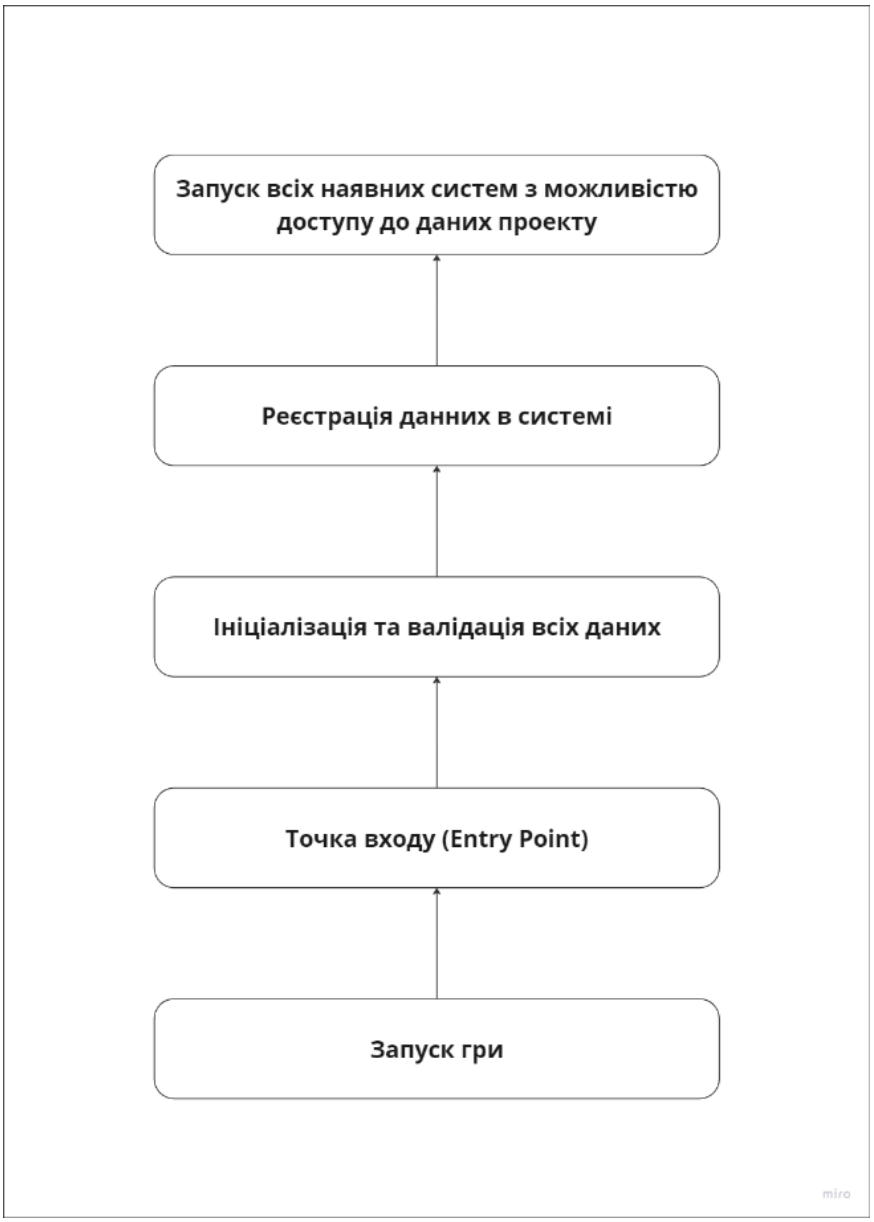

Рис. 2.4. Система ініціалізації даних гри

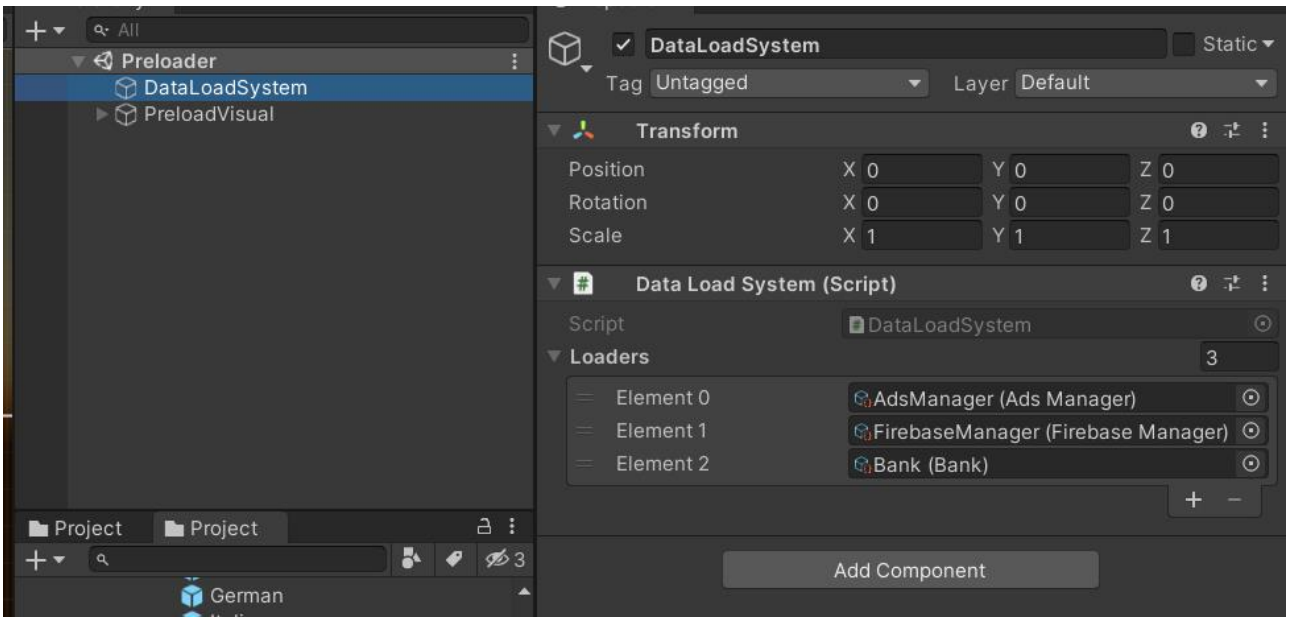

Рис. 2.5. Сцена та компонент ініціалізації даних гри

2.4. Опис використаних технологій та мов програмування

Для реалізації даної кваліфікаційної роботи були використано наступні технології та мови програмування :

- C#;
- Unity;
- Physics Engine;
- Universal Render Pipeline;
- Android Build Support;
- Scriptable Objects;
- StreamingAssets;
- Firebase SDK;
- Unity Ads;
- UI;
- TextMeshPro;
- Timeline;
- Shader Graph;

– LeanTween;
– Particle System.

C# є основною мовою програмування для розробки у Unity. Вона використовується для написання скриптів, компонентів та реалізації логіки гри.

Unity є інтегрованим середовищем розробки, яке дозволяє створювати ігри та інші інтерактивні додатки. Воно забезпечує інструменти для створення, редагування і візуалізації ігрових сцен, об'єктів та інтерфейсів.

Фізичний двигун Unity відповідає за симуляцію фізичних законів у грі. Він забезпечує обробку колізій, реалістичну фізику об'єктів та їх поведінку в середовищі гри.

Universal Render Pipeline (URP) – це легкий і оптимізований для мобільних платформ рендерний шлях Unity. Він надає засоби для створення візуально привабливих ігор з підтримкою різних освітлення, тіней та ефектів.

Android Build Support або підтримка збірки для Android у Unity дозволяє розробникам створювати і тестувати ігри для мобільної платформи Android, використовуючи всі можливості Unity [13]. Це також дає широкі можливості налаштування всіх необхідних умов для стабільної працездатності гри (рис. 2.6).

Scriptable Objects у Unity є спеціальними об'єктами, які дозволяють створювати легкі та повторно використовувані конфігураційні об'єкти для компонентів гри, таких як налаштування, інвентар, статистика тощо.

StreamingAssets: Папка StreamingAssets у Unity використовується для зберігання файлів, які можна читати під час роботи програми. Це дозволяє вбудовувати ресурси, такі як тексти, зображення або інші дані, безпосередньо в додаток.

Firebase SDK у Unity забезпечує зручний доступ до ряду сервісів Firebase, таких як база даних, аутентифікація, зберігання файлів тощо, що дозволяє легко інтегрувати хмарні сервіси в гру.

Unity Ads – це платформа реклами, яка інтегрована безпосередньо у Unity. Вона надає інструменти для впровадження рекламних матеріалів у гру з метою заробітку на рекламі.

Інтерфейс користувача (UI) в Unity використовується для створення і редагування елементів інтерфейсу гравця, таких як кнопки, текстові поля, панелі тощо.

TextMeshPro – це розширення для текстових компонентів у Unity, яке надає більші можливості форматування тексту, шрифтів і анімації тексту.

Timeline у Unity дозволяє створювати і управляти анімацією та подіями в грі через візуальний інтерфейс, що спрощує процес створення скриптів анімації та кінематографії.

Shader Graph – це візуальний інструмент у Unity для створення шейдерів, який дозволяє візуально налаштовувати та з'єднувати вузли, щоб створювати складні візуальні ефекти.

LeanTween – це бібліотека анімації для Unity, яка забезпечує простий і потужний інтерфейс для створення і управління анімаціями об'єктів.

Particle System у Unity використовується для створення та управління візуальними ефектами, такими як вогонь, дим, вибухи, водяні фонтани тощо.

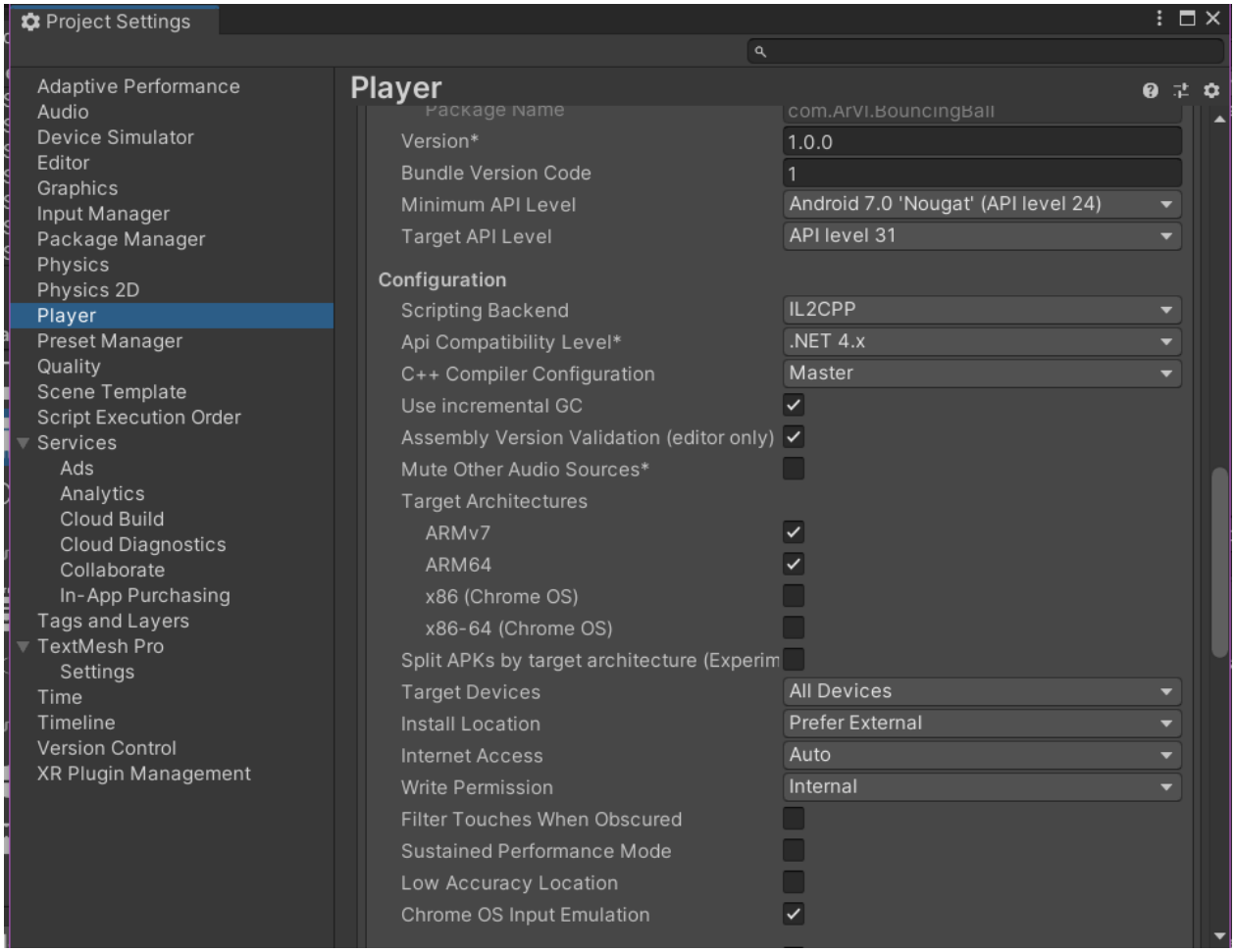

Рис. 2.6 Вікно налаштувань збірки

2.5. Опис структури програми та алгоритми її функціонування

Для отримання повноцінного розуміння структури програми проведемо аналіз всього ігрового циклу – від кліку на іконку нашої гри, що є початком всього ігрового процесу, до виходу з неї.

Як уже було сказано вище, для нашої гри дуже важливою є підготовка та завантаження даних. Тому перший цикл оновлення нашої гри є маркером початку цього процесу (рис. 2.4) і супроводжується анімацією завантаження.

Наступним етапом є головне меню [20] (рис. 2.7), яке є центром нашої гри, оскільки воно дає можливість гравцеві перейти безпосередньо до ігрового рівня або отримати доступ до іншого контенту, будь то магазин чи налаштування. На цьому етапі наша програма очікує на введення гравця, щоб зрозуміти його наміри та відреагувати на них (рис. 2.8).

Рис. 2.7 Головне меню

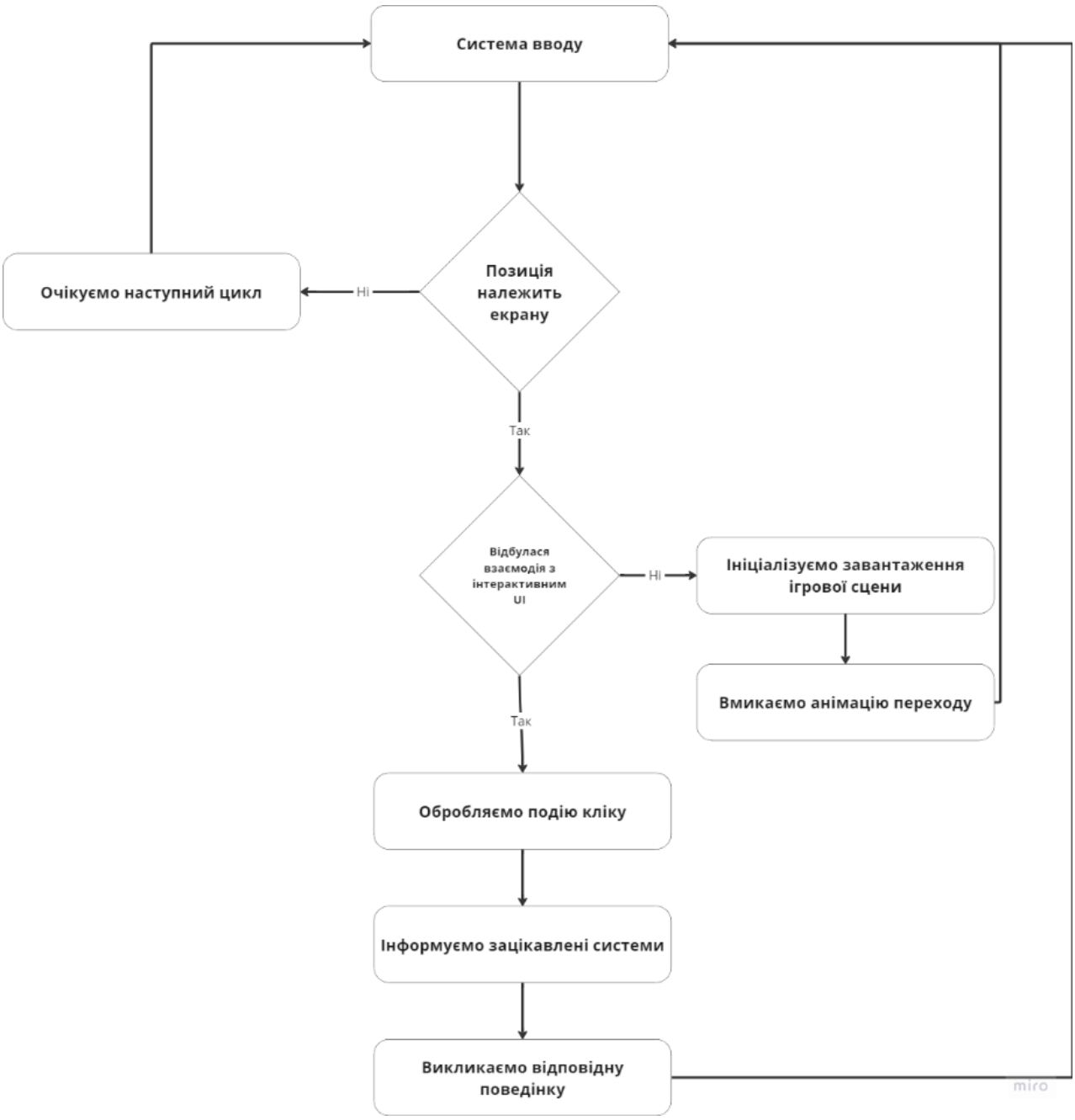

Рис. 2.8 Блок-схема головного меню

Основна структура програми зосереджена саме на ігровому рівні. Після вибору ігрового рівня гравцем, нас відразу зустрічає система завантаження рівнів. Вона використовує дані прогресу гравця, на основі яких визначає, яку саме сцену необхідно завантажити. Ця система має дві додаткові підсистеми. Одна з них є внутрішньою системою Unity для завантаження сцен, а саме SceneManager [7]. Друга відповідає за плавний перехід між сценами, отримуючи

прогрес завантаження від SceneManager і на його основі додаючи відповідне візуальне супроводження.

Після запуску ігрового рівня нас знову цікавить система вводу (Input System) [21], оскільки рух нашого м’яча є відповідною реакцією на свайпи гравця по екрану (рис. 2.9). При перетягуванні на екрані обчислюється вектор переміщення на основі сенсорного введення, чутливості управління та розмірів екрану. Якщо швидкість підйому перевищує заданий ліміт, вертикальна складова вектора коригується. Якщо персонаж не знаходиться на землі, швидкість підйому збільшується [23]. Залежно від стану компоненту унікальної поведінки розтягування , до швидкості руху персонажа додається вектор переміщення, або зберігається вектор переміщення. Крім того, до персонажа додається випадковий крутний момент, щоб додати реалістичності руху. При завершенні роботи компонента відключається обробка події торкання, щоб уникнути помилок.

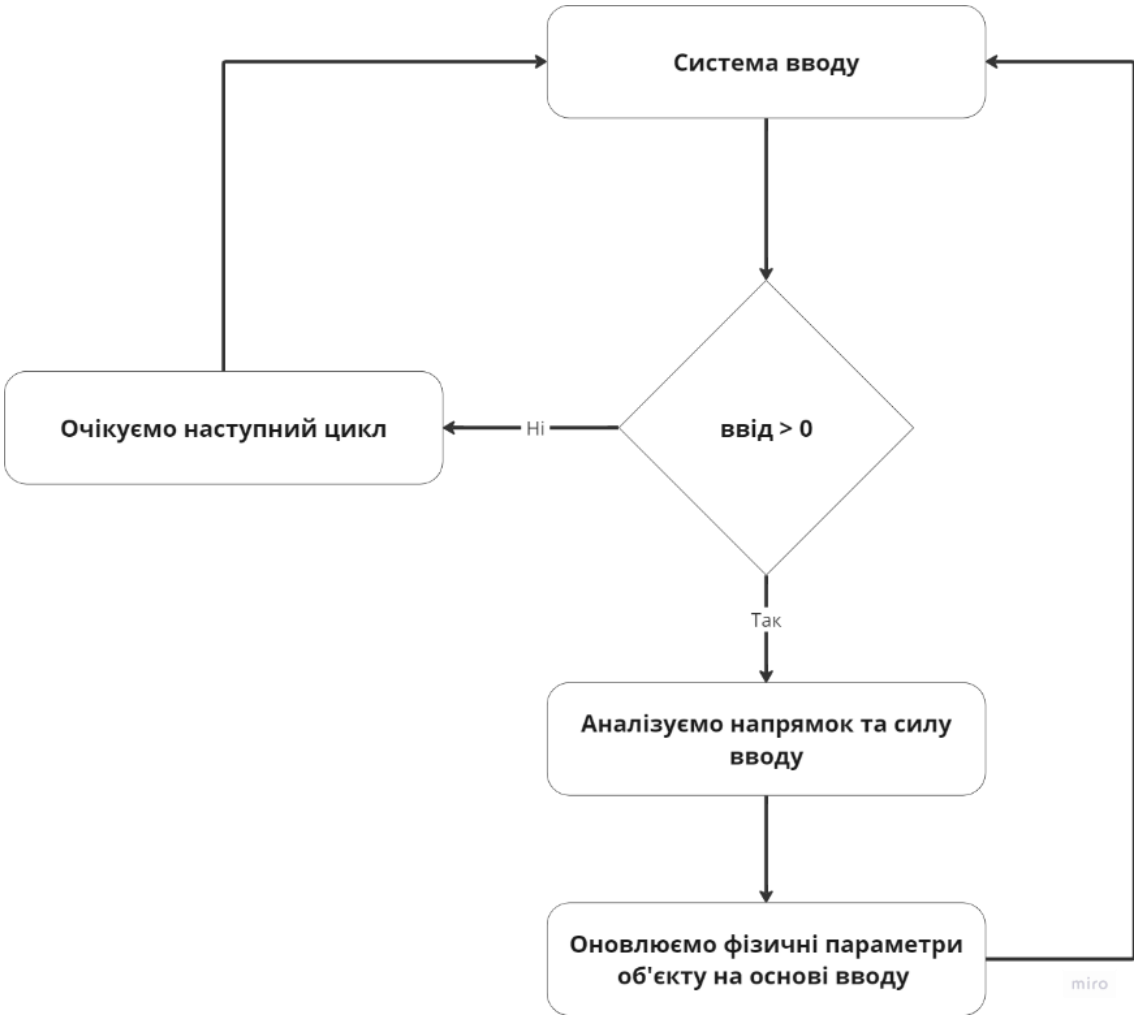

Рис. 2.9 Блок-схема руху

На основі зміни фізичних характеристик відбувається зміна позиції, що, в свою чергу, провокує можливість взаємодії з елементами ігрового оточення [6]. Логіка руху і можливості взаємодії спирається на вбудований в Unity фізичний двигун, який забезпечує коректну симуляцію фізичного середовища, а також прораховує так звані колізії, тобто ситуації, коли тверді та інші тіла зіштовхуються одне з одним. Саме колізія стане основою подальшої логіки, оскільки дозволить нам зрозуміти момент взаємодії з різними типами інтерактивних об’єктів на нашій сцені.

В грі є клас, який представляє гравця (Player) і включає комбіновану логіку. Однією з його частин є обробка подій колізії . Для цього Unity пропонує успадкувати наш клас від MonoBehavior, представити його у вигляді ігрового об’єкта та додати до нього компонент Rigidbody, що є маркером для розділення компонентів на фізичні та не фізичні.

У ситуаціях, коли Unity обробляє будь-яку взаємодію фізичних елементів, вона надсилає відповідні події (Events) нашим компонентам [9]. Ці компоненти можуть отримати доступ до подій, використовуючи вбудовані функції MonoBehavior. Саме тому успадкування цього класу є таким важливим для нас.

Перелік методів до яких може отримати доступ клас обробник :

1. OnCollisionEnter – викликається, коли об'єкт починає торкатися іншого об'єкта з колайдером або Rigidbody. Цей метод використовується для обробки подій зіткнення, таких як відштовхування об'єктів або виклик певних дій при зіткненні.
2. OnTriggerEnter – викликається, коли об'єкт входить в тригер-зону. Метод корисний для активації подій без фізичного зіткнення, наприклад, при вході в область, що запускає анімацію або змінює стан гри
3. OnCollisionExit – викликається, коли об'єкт припиняє торкатися іншого об'єкта. Використовується для скасування ефектів, що застосовуються при зіткненні, або для виконання дій після завершення зіткнення.

4. OnTriggerExit – викликається, коли об'єкт виходить з тригер-зони. Метод застосовується для завершення подій, які були активовані при вході в тригер-зону, таких як вимкнення світла або зупинка звуку.
5. OnCollisionStay – викликається на кожному кадрі, поки об'єкти продовжують торкатися один одного. Це корисно для постійної перевірки стану зіткнення і виконання дій протягом усього часу зіткнення.
6. OnTriggerStay – викликається на кожному кадрі, поки об'єкт залишається в тригер-зоні. Використовується для постійного виконання дій, поки об'єкт знаходиться в тригер-зоні, таких як відновлення здоров'я або збір ресурсів.

Обробка події колізії гравця є основою системи здоров’я. У момент зіткнення ми проводимо перевірку фізичного шару, а саме відповідність шару “Trap”. Додатково в цей момент відбувається перевірка на наявність імунітету до отримання шкоди. Якщо обидві умови виконуються, тобто коли наш гравець зіткнувся з об’єктом, позначеним як пастка, та за умови відсутності імунітету, ми зменшуємо кількість його ігрового здоров’я. Після цього ми додатково перевіряємо, чи достатньо в нашого об’єкта здоров’я для продовження гри. В протилежному випадку ми викликаємо смерть нашого ігрового персонажа та завершення ігрового рівня з незадовільним результатом (рис. 2.10).

Аналогічну логіку має й інша ігрова система, а саме система збору предметів на рівні [15]. У нашому випадку вони виражені у формі монет і є основною ігровою валютою. Всі предмети, які ми можемо зібрати, повинні реалізовувати інтерфейс IPickable, що містить метод OnTake. Окрема система паралельно з усіма іншими обробляє взаємодію м’яча. Під час кожної взаємодії відбувається спроба отримати доступ до компоненту, що реалізує інтерфейс IPickable. Якщо ця спроба є успішною, то, отримавши доступ до екземпляру цього об’єкта, ми викликаємо метод OnTake, який містить логіку збору предмету [16] (рис. 2.11).

Рис. 2.10 Блок-схема системи здоров’я

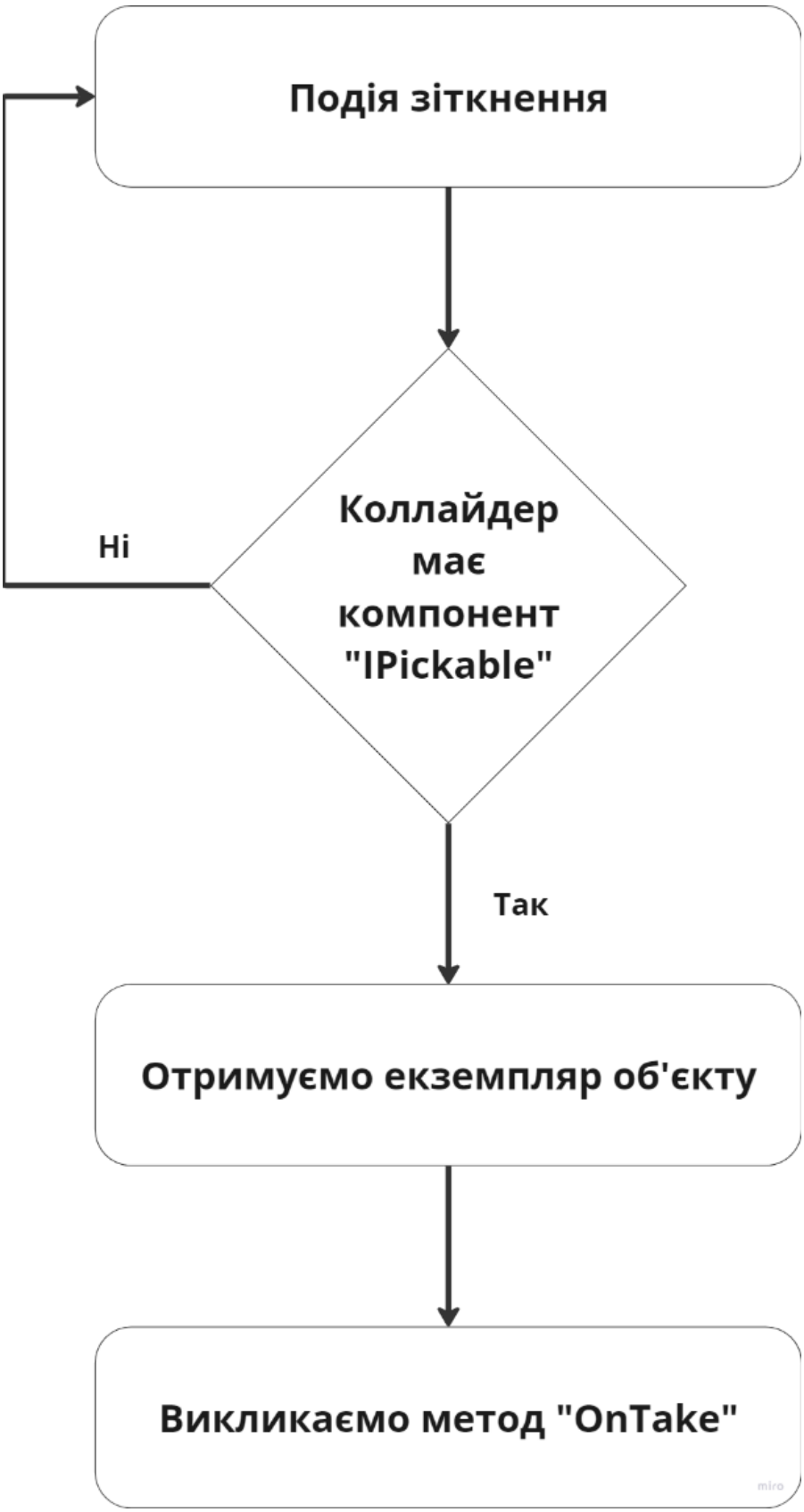

Рис. 2.11 Блок-схема системи підбору предметів

2.6. Опис розробленого програмного забезпечення

2.6.1. Використані технічні засоби

Розробка ПЗ була виконана на персональному ЕОМ з такими характеристиками та периферією:

- процесор: Intel(R) Core(TM) i5-9300H CPU;
- модулі пам’яті: Kingston Fury DDR4-3200 CL22 SODIMM DR;
- накопичувач : Goodram 1tb m2 2280 nvme pcie 4.0;
- підтримка відео;
- можливість підключення до двох моніторів;
- можливості підключення Ethernet;
- зв'язок Wi-Fi;

2.6.2. Використані програмні засоби

Перелік програмних засобів:

- ПЗ було створена завдяки інструменту для розробки ігор Unity 2020.3.17f1;
- IDE було обрано середовище Rider від компанії Jetbrains;
- контроль версій – git, з використанням консолі та візуальної оболонки Fork;
- створення зображень та блок-схем - Miro;
- пошук безкоштовних 3Д моделей використовувався браузер Google Chrome;

2.6.3. Виклик та завантаження програми

Для запуску програми, потрібно встановити .apk файл на мобільний пристрій чи емулятор з ОС Android.

2.6.4. Опис інтерфейсу користувача

Інтерфейс у нашій грі є простим і інтуїтивно зрозумілим. Він спрощує взаємодію гравця з ігровим світом, забезпечуючи необхідну інформацію та контроль над діями через зрозумілі та легкі у використанні елементи і кнопки [17].

Після відкриття гри нас зустрічає вікно завантаження, яке включає шкалу заповнення та текстовий опис прогресу. На фоні цього вікна постійно програється анімація, яка чітко ілюструє прогрес завантаження, надаючи гравцеві зрозуміле уявлення про актуальний стан завантаження (рис. 2.12).

Рис. 2.12 Екран завантаження

Основою гри є головне меню (рис. 2.7) з обширним ігровим інтерфейсом. У центрі ігрового екрану розміщений показник ігрового прогресу у вигляді тексту з відображенням актуального рівня, а також підказка для початку гри. Додатково інтерфейс містить інші інтерактивні елементи, такі як кнопки налаштувань, магазину та лічильник монет.

Вікно налаштувань (рис. 2.13) представляє з себе “Pop-up” тобто воно відображається поверх існуючого зображення , але не закриває його повністю. До списку доступних опцій в вікні налаштувань входить : зміна гучності , зміна чутливості управління , зміна локалізації , можливість ознайомитися з політикою конфіденційності (рис. 2.14).

Рис. 2.13 Вікно налаштувань

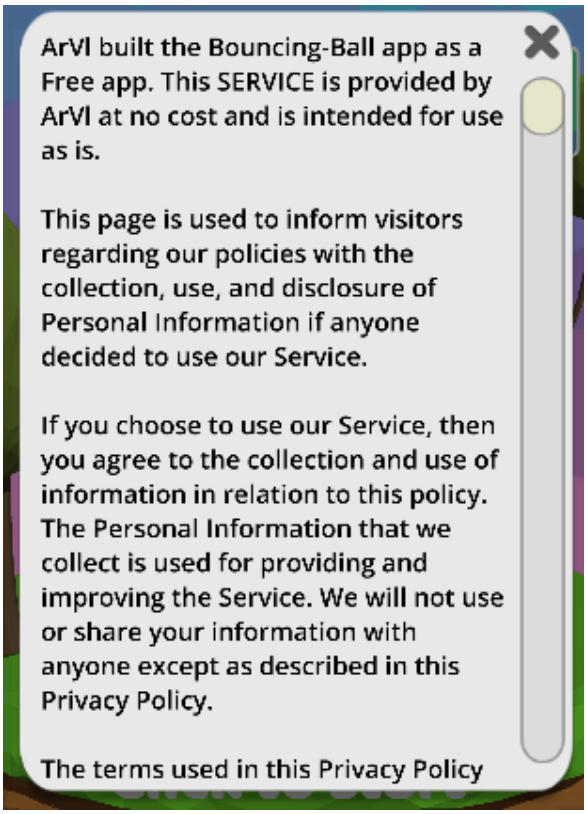

Рис. 2.14 Політика конфіденційності

Ігровий магазин представлений набором доступних візуальних модифікаторів [18] ("skins"). Актуальний візуальний модифікатор, який застосовується до м’яча у центрі, відображається на екрані. Нижня панель інтерфейсу надає можливість купувати, змінювати і продавати ці модифікатори (рис. 2.15). Також доступна функція сортування для зручного організування списку модифікаторів (рис. 2.16).

Рис. 2.15 Ігровий магазин

Рис. 2.16 Сортування товарів

Ігрова сцена (рис. 2.17) включає мінімальний набір необхідних елементів інтерфейсу. На екрані присутній лічильник монет, який відображає поточну кількість монет гравця, кількість доступних життів і кнопка паузи. Натискання на кнопку паузи відкриває вікно паузи (рис. 2.18) , де гравець може призупинити гру або налаштувати параметри.

Рис. 2.17 Інтерфейс ігрового рівня

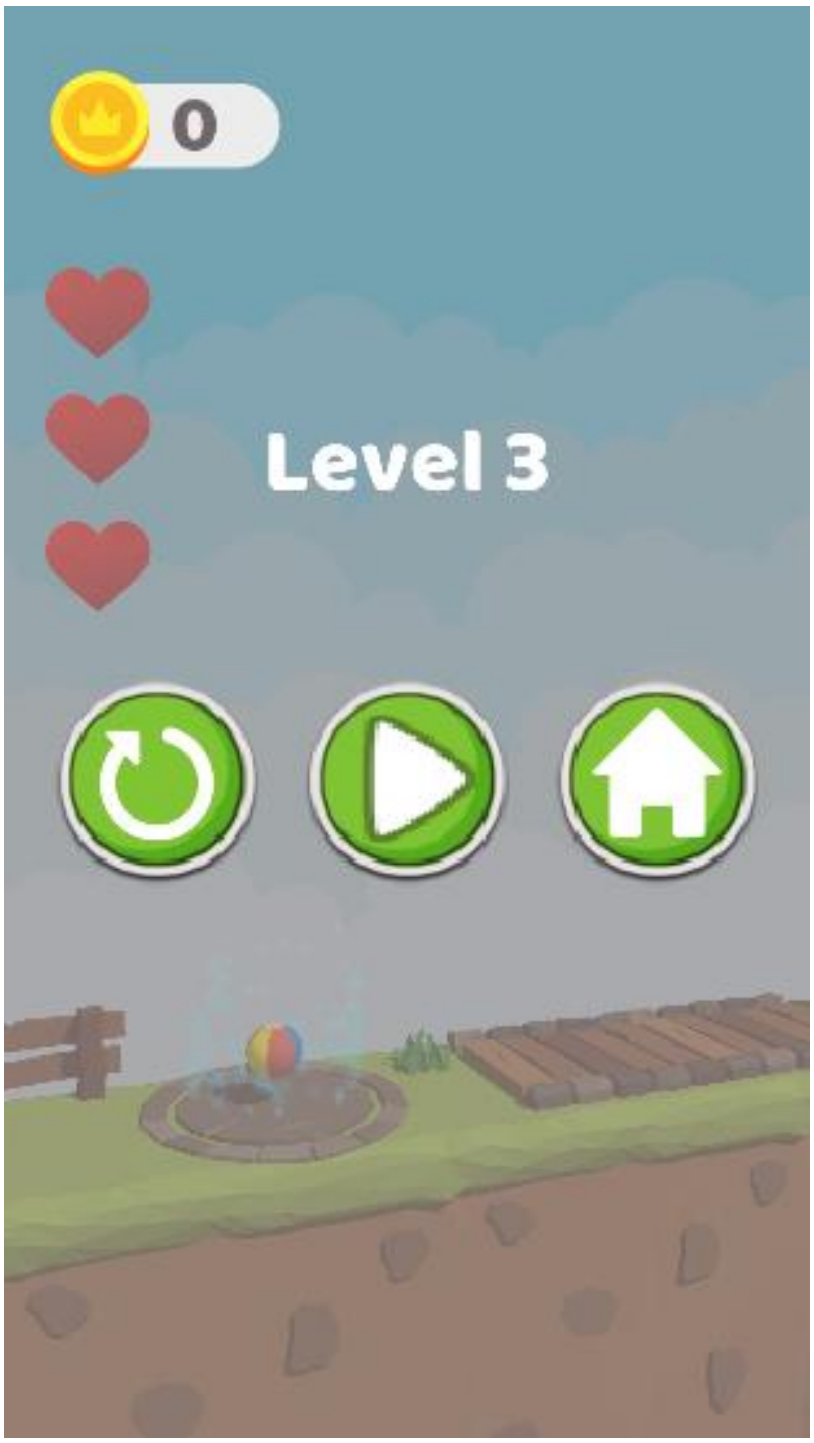

Рис. 2.18 Інтерфейс вікна паузи

Після проходження рівня відкривається вікно (рис. 2.19), що містить інформацію про кількість монет, яку гравець отримав під час проходження рівня. Окрім текстової інформації, вікно включає кілька кнопок: "Множення", "Наступний рівень" і "Додому".

Кнопка "Множення" надає можливість гравцеві подвоїти кількість монет, переглянувши рекламу (рис. 2.20) .

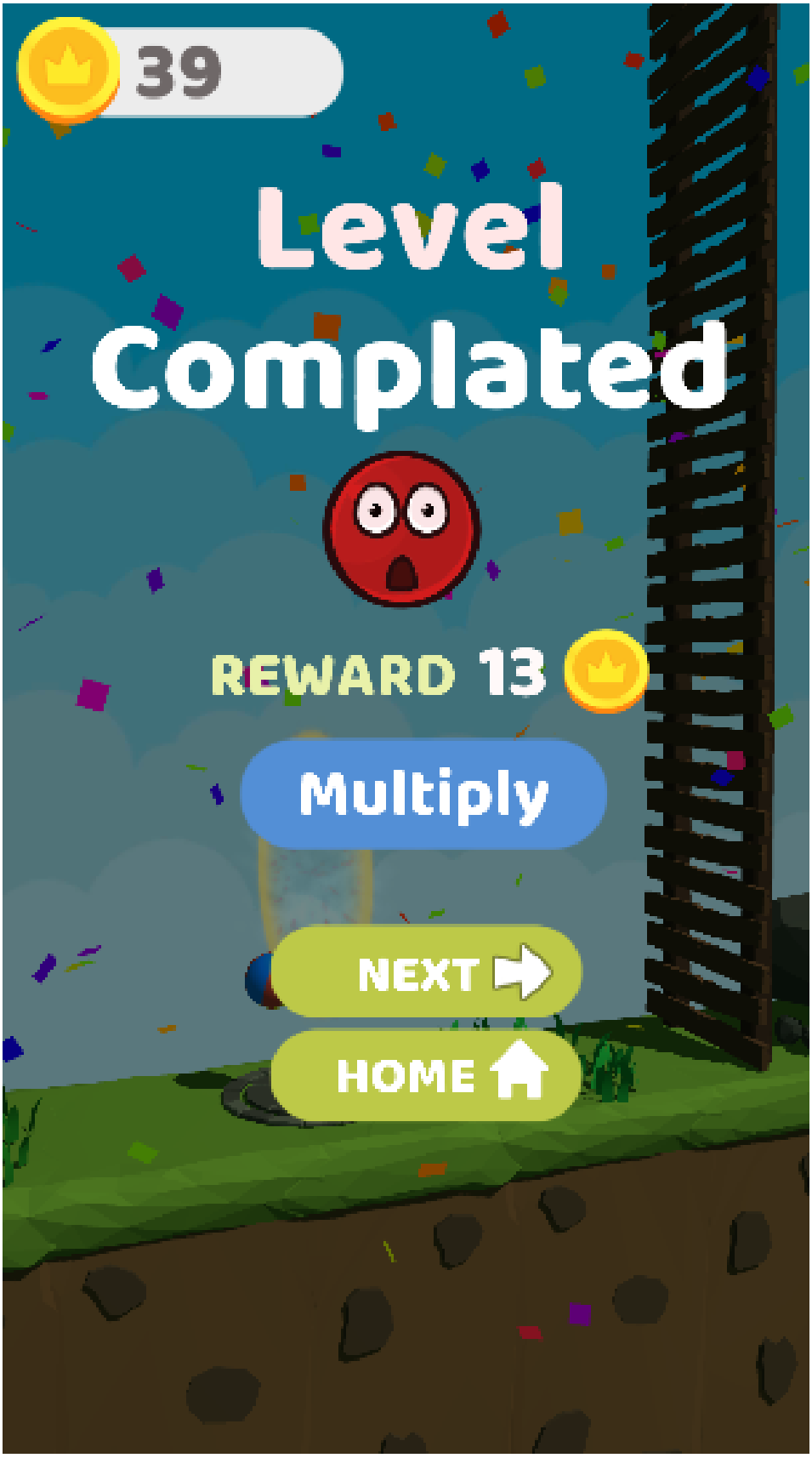

Рис. 2.19 Вікно завершення рівня

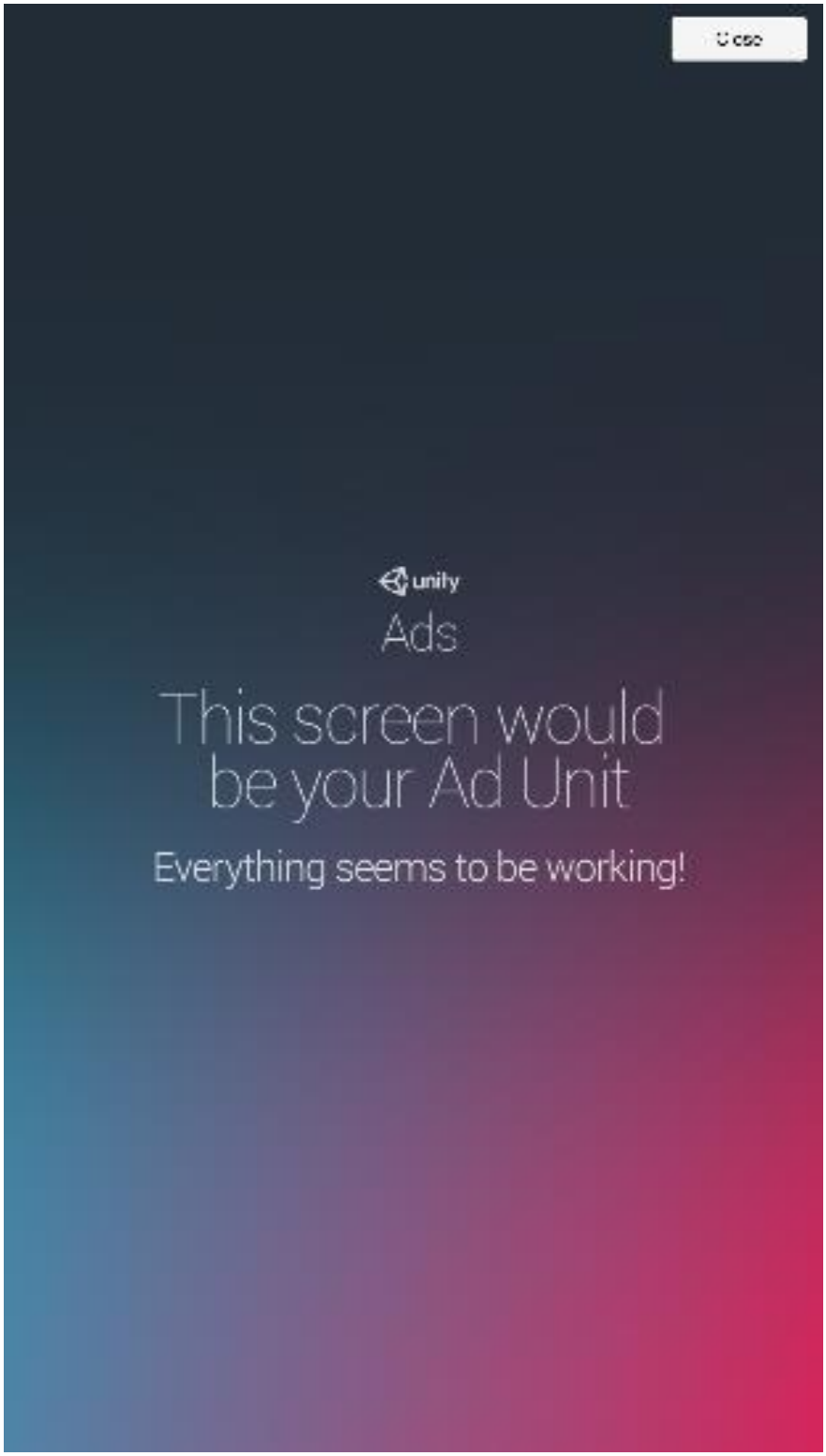

Рис. 2.20 Вікно перегляду реклами

У разі поразки на ігровому рівні викликається окреме вікно (рис. 2.21), яке інформує гравця про поразку, показує кількість зароблених монет і пропонує дві опції: перезапустити рівень або повернутися в головне меню.

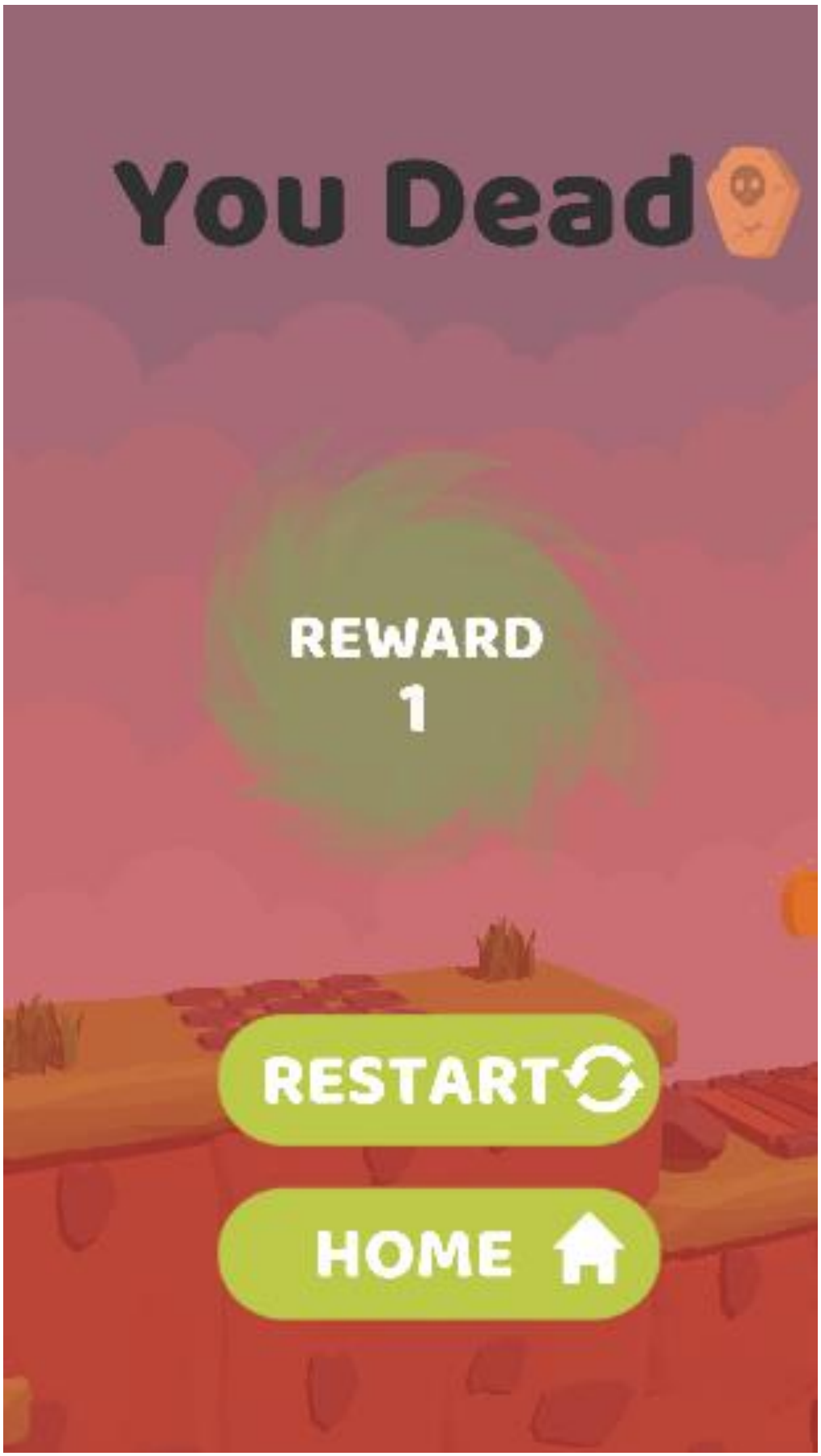

Рис. 2.21 Вікно перегляду реклами

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Визначення трудомісткості та вартості розробки програмного продукту

Початкові дані:

1.Передбачуване число операторів програми – 2240;

2.Коефіцієнт складності програми – 1,25;

3.Коефіцієнт корекції програми в ході її розробки – 0,1;

4.Годинна заробітна плата програміста – 504,84 грн/год;

Виходячи з статистичних даних, що були взяті з сайту jooble, https://ua.jooble.org/salary/unity-developer#hourly, станом на 29 травня 2024 середня заробітна плата позиції Unity Developer становить 1996,91$, що при курсі валют долара до гривні (40,45 грн), згідно Національного банку України прирівнюється до 80775 грн. Тоді, середня заробітна плата програміста в даній галузі становитиме 504,84 грн/год.

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;

7. вартість машино-години ЕОМ – 1,59 грн/год.

Розрахунок вартості витрат проведена з урахуванням часу затраченого на написання програми, що тривала впродовж 9 місяців. Ціна за електроенергію в Україні становить 2,64 грн кВт/год з урахуванням ПДВ. Виходячи зі споживання електроенергії пристроєм, на якому велася розробка, а саме 125Вт на годину, витрати під час роботи на електроенергію складають 0,125*2,64=0,33. Додатково, підключення до інтернет – провайдеру «Київстар» проводиться за умовами тарифного плану «Ваш Оптимум», щомісячна оплата якого складає 250 грн. Таким чином, вартість використання погодинно становить 0,35 грн.

Обслуговування персонального комп'ютера вартістю 40000 грн, гарантована тривалість роботи якого, складає 5 років. Тобто, погодинне значення використання складатиме 0,91 грн. Отже, загальна вартість ЕОМ дорівнює 1,59 грн.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \qquad (3.2)$$

де q - передбачуване число операторів (2240);

C - коефіцієнт складності програми (1,25);

p - коефіцієнт корекції програми в ході її розробки (0,1).

Звідси умовне число операторів в програмі:

$$Q = 1{,}25 \cdot 2240 \cdot (1 + 0{,}1) = 3080$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,} \qquad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1.2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (3080 \cdot 1{,}2) / (75 \cdot 1{,}2) = 41 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25) \cdot k}, \text{ людино-годин,} \qquad (3.4)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), отримаємо:

$$t_a = 3080 / (20 \cdot 1{,}2) = 128{,}25 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}, \text{ людино-годин,} \qquad (3.5)$$

t_n = 3080 / (25 · 1,2) = 102 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}, \text{ людино-годин,} \qquad (3.5)$$

$t_{отл}$ = 3080 / (5 · 1,2) = 513 людино-годин.

- за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл}, \text{ людино-годин,} \qquad (3.6)$$

$t^k_{отл}$ = 1,5 · 513 = 770 людино-годин.

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \qquad (3.7)$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20)\cdot k}, \text{ людино-годин,} \qquad (3.8)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др}\text{, людино-годин,} \qquad (3.9)$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = 3080 \ / (18 \cdot 1{,}2) = 142{,}6 \text{ людино-годин.}$$

$$t_{до} = 0{,}75 \cdot 142{,}6 = 106{,}95 \text{ людино-годин.}$$

$$t_{д} = 142{,}6 + 106{,}95 = 249{,}5 \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 41 + 128{,}25 + 102 + 770 + 249{,}5 = 1\,340{,}75 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}\text{, грн,} \qquad (3.10)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн}, \quad (3.11)$$

де: t - загальна трудомісткість, людино-годин;

$C_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 504,84 грн / год, отримуємо:

$$З_{ЗП} = 1\ 340{,}75 \cdot 504{,}84 = 676\ 864 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн}, \quad (3.12)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (0,68).

Підставивши в формулу (3.12) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 1\ 340{,}75 \cdot 1{,}59 = 2131{,}8 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 2131{,}8 + 676\ 864 = 678\ 995{,}8 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.,} \qquad (3.12)$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин);

t – загальна трудомісткість, людино годин()

Звідси витрати на створення програмного продукту:

$$T = 1\ 340{,}75 / 1 \cdot 176 \approx 7{,}61 \text{ міс.}$$

Висновок: для розробки ігрового застосунку з використанням рушія Unity загальна трудомісткість становить 1 340 людино-годин, тому очікуваний період створення програмного додатку становитиме, приблизно, 7,6 місяців, а сумарні витрати на створення - 678 995 грн.

ВИСНОВКИ

Метою кваліфікаційної роботи є розробка програмного забезпечення для мобільного ігрового додатку в жанрі платформер , призначену для покращення емоційного стану людини .

Актуальність нашої розробки полягає у створенні додатку для платформи Android, який використовує передові інструменти програмування та гейм дизайну, такі як C# та Unity.

В процесі виконання кваліфікаційної роботи було розроблене програмне забезпечення, яке можна виконати як виконавчий файл на мобільних пристроях з операційною системою Android та була спроектована система з використанням патерів проектування, розроблена основна механіка руху кулі з введенням даних через свайпи , створено набір ігрових локацій , що представляють з себе окремі рівні , набір різного роду інтерактивних об’єктів та інтерфейс користувача. Ця гра не лише забезпечує розвагу та відпочинок для користувачів, але й демонструє можливості реалізації складних ігрових механік, таких як фізика об'єктів та інтерактивний геймплей.

Розробка була виконана з використанням ігрового двигуна Unity. Більшість необхідних для роботи компонентів було забезпечено внутрішніми інструментами, а саме Unity Physics для реалізації фізичних взаємодій об'єктів, Unity Ads для інтеграції рекламних оголошень, Animation System для керування анімаціями, UI Toolkit для створення інтерфейсу користувача та Audio System для додавання звукових ефектів і музики. Ці інструменти дозволили створити складну, але ефективну ігрову систему, забезпечивши високу якість та продуктивність нашого проекту.

Увесь код створений за допомогою мови програмування C#

У світі, де мобільні ігри набувають все більшої популярності, наша гра пропонує унікальний ігровий досвід завдяки високій якості графіки, продуманій

фізиці та інтерактивному геймплею. Це сприяє не лише задоволенню користувачів, але й розширює можливості розробників для створення складних та захоплюючих проектів.

В кваліфікаційній роботі було розраховано трудомісткість розробки програмного забезпечення 1340 год, вартість розробки гри 678995 грн , та розрахований період розробки додатку буде складати близько 8 місяців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# 2013
2. Гейм дизайн. Яз зробити гру в яку будуть грати усі. Джесі Шел. 2021.
3. “Game Design Workshop: A Playcentric Approach to Creating Innovative Games” Tracy Fullerton. 2004
4. Паттерни проектування: https://refactoring.guru/uk/design-patterns
5. Голдстоун, В. (2016). Unity 5.x Cookbook: Понад 100 рецептів для початку розробки додатків і ігор з Unity 5
6. Хокінг, Дж. (2015). Unity в дії: Розробка багатоплатформових ігор на C# з Unity 5
7. Бреш, Дж. (2018). Unity 2018 By Example: Вивчайте основи Unity та C# шляхом створення 4 ігор, 2-ге видання. Packt Publishing
8. Грегорі, Дж. (2017). Архітектура ігрових двигунів, третє видання. CRC Press
9. Design Patterns. Elements of Reusable. Object-Oriented Software. Erich Gamma. 2001
10. Ковальчук, М. (2019). Створення ігор на Unity: Від ідеї до реалізації
11. Бойко В.В. Эк підприємства України. Основной курс: Підручник для вузів. - Д.: Пороги, 1997. - 312 с
12. Петров, А. (2018). Розробка ігор на Unity для професіоналів
13. Beginning Android. Games Mario Zechner. 2012
14. The C# Player's Guide (4th Edition). RB Whitaker. 2019.
15. Theory of Fun for Game Design. Raph Koster. 2013
16. Game Feel: A Game Designer's Guide to Virtual Sensation (Morgan Kaufmann Game Design Books). Steve Swink. 2008
17. "Rules of Play: Game Design Fundamentals" Katie Salen and Eric Zimmerman. 2003.
18. "Level Up! The Guide to Great Video Game Design" Scott Rogers. 2010.

19. "Game Programming Patterns" Robert Nystrom. 2014.

20. Unity 2021 Cookbook: Over 140 recipes to take your Unity game development skills to the next level" Matt Smith and Shaun Ferns. 2021.

21. "Learning C# by Developing Games with Unity 2020" Harrison Ferrone. 2020.

22. "Pro Unity Game Development with C#" Alan Thorn. 2014.

23. "Game AI Pro: Collected Wisdom of Game AI Professionals" Edited by Steve Rabin. 2013.

24. "Game Engine Architecture" Jason Gregory. 2017.

25. "Introduction to Game Design, Prototyping, and Development" Jeremy Gibson Bond. 2014.

ДОДАТОК А

КОД ПРОГРАМИ

```
using System;
using System.Collections;
using UnityEngine;

[RequireComponent(typeof(Rigidbody), typeof(AudioSource))]
public class Player : MonoBehaviour
{
    public Action Touch;

    [HideInInspector] public SquashAndStretch SqAndStr;
    [HideInInspector] public Rigidbody RB;

    public bool IsGrounded => _isGrounded;
    private bool _isGrounded;

    [Header("Damage Settings")]
    [SerializeField] private UIDamageSystem _uiDamageSystem;
    [SerializeField] private MeshRenderer _immunitySphere;
    private AudioSource _audioSource;

    private float _currentImmunitySphereAmount;

    [Header("Parameters")]
    [SerializeField] private int _maxHealth;
    private int _currentHealth;

    [SerializeField] private float _immunityTime;
    private float _currentImmunityTime;

    [SerializeField] private float _maxBallSpeed;
    [SerializeField] private float _maxBallHorizontalSpeed;

    [SerializeField] private bool _isPlayer;

    private void Awake()
    {
        _currentHealth = _maxHealth;
        _audioSource = GetComponent<AudioSource>();

        RB = GetComponent<Rigidbody>();

        TryGetComponent(out SqAndStr);

        _uiDamageSystem?.ShowHearts(_currentHealth);

        _currentImmunityTime = _immunityTime;
    }

    private void Update()
    {
        if (RB.velocity.magnitude > _maxBallSpeed)
            RB.velocity = Vector3.ClampMagnitude(RB.velocity, _maxBallSpeed);

        if (RB.velocity.x > _maxBallHorizontalSpeed || RB.velocity.x < -_maxBallHorizontalSpeed)
            RB.velocity = new Vector3(Mathf.Clamp(RB.velocity.x, -_maxBallHorizontalSpeed,
_maxBallHorizontalSpeed), RB.velocity.y, RB.velocity.z);
    }

    private void OnCollisionEnter(Collision collision)
```

```
    {
        if (collision.gameObject.layer == Constants.Layers.Trap && _currentImmunityTime >= _immunityTime)
        {
            GetDamaged();
            Handheld.Vibrate();
        }

        _isGrounded = true;

        Touch?.Invoke();
    }

    private void OnCollisionExit(Collision collision)
    {
        _isGrounded = false;
    }

    private void GetDamaged()
    {
        _currentHealth--;

        if (_currentHealth < 0)
        {
            _uiDamageSystem.Death();
        }
        else
        {
            _audioSource.Play();

            _uiDamageSystem.GetDamaged();

            _currentImmunityTime = 0;

            _currentImmunitySphereAmount = 0;

            _immunitySphere.gameObject.SetActive(true);
            _immunitySphere.material.SetFloat(Constants.ImmunitySphereAmount, _currentImmunitySphereAmount);

            StartCoroutine(ImmunityTimer());
        }

        _uiDamageSystem.ShowHearts(_currentHealth);
    }

    private IEnumerator ImmunityTimer()
    {
        while (_currentImmunityTime < _immunityTime)
        {
            _currentImmunityTime += Time.deltaTime;

            _currentImmunitySphereAmount = Mathf.Lerp(_currentImmunitySphereAmount, 1, Time.deltaTime /
_immunityTime);
            _immunitySphere.material.SetFloat(Constants.ImmunitySphereAmount, _currentImmunitySphereAmount);

            yield return null;
        }

        if (_currentImmunityTime > _immunityTime)
        {
            _currentImmunitySphereAmount = 1;
            _immunitySphere.material.SetFloat(Constants.ImmunitySphereAmount, _currentImmunitySphereAmount);
            _immunitySphere.gameObject.SetActive(false);
```

```
        }
    }
}

public abstract class BaseDataLoader : ScriptableObject
{
    public DataLoaders Key;

    public abstract IEnumerator Init();
}

public class DataLoadSystem : MonoBehaviour
{
    [SerializeField] private List<BaseDataLoader> _loaders;

    private static readonly Dictionary<DataLoaders, BaseDataLoader> StaticDictionary = new Dictionary<DataLoaders,
BaseDataLoader>();

    public static T GetLoader<T>(DataLoaders loader) where T : BaseDataLoader
    {
        return StaticDictionary[loader] as T;
    }

    private void Awake()
    {
        Initialize();
    }

    private void Initialize()
    {
        for (int i = 0; i < _loaders.Count; i++)
        {
            var loader = _loaders[i];
            StartCoroutine(_loaders[i].Init());
            StaticDictionary.Add(loader.Key, loader);
        }
    }
}

using UnityEngine;
using UnityEngine.EventSystems;

public class BallControl : MonoBehaviour, IDragHandler
{
    [Range(0.5f, 1.5f)]
    [SerializeField] private float _controlSensivity;
    [SerializeField] private float _limitVerticalBoostSpeed;
    [SerializeField] private float _torque;

    [SerializeField] private Player _player;

    private float _verticalBoostSpeed;

    private SquashAndStretch _sqAndStr;
    private Rigidbody _rb;

    private void Start()
    {
        _controlSensivity = PlayerPrefs.GetFloat(Constants.PPname.ControlSensivity);
        _rb = _player.RB;
        _player.Touch += OnTouch;
```

```
        if (_player.SqAndStr != null)
            _sqAndStr = _player.SqAndStr;

        FirebaseManager firebaseManager =
DataLoadSystem.GetLoader<FirebaseManager>(DataLoaders.FirebaseManager);
        firebaseManager.StartLevel();
    }

    private void OnTouch()
    {
        _verticalBoostSpeed = 0;
    }

    private void AddRandomTorque()
    {
        _rb.AddTorque(RandomFloatNumber(_torque), RandomFloatNumber(_torque), RandomFloatNumber(_torque));
    }

    private float RandomFloatNumber(float num)
    {
        return Random.Range(-num, num);
    }

    public void OnDrag(PointerEventData eventData)
    {
        float screenDifferenceMultiplier = 2160 / (float)Screen.height;

        Vector3 vector = (Vector3)eventData.delta * Time.deltaTime * _controlSensivity * screenDifferenceMultiplier;

        if (_verticalBoostSpeed > _limitVerticalBoostSpeed)
        {
            if (vector.y > 0)
                vector.y = 0;
            else if (vector.y < 0)
                vector.y /= 2;
        }

        if (_player.IsGrounded == false)
            _verticalBoostSpeed += Mathf.Abs(vector.y);

        if (_sqAndStr?.IsGrounded == false)
            _rb.velocity += vector;
        else if (_sqAndStr != null)
            _sqAndStr.AddSaveVelocity(vector);

        AddRandomTorque();
    }

    private void OnDestroy()
    {
        _player.Touch -= OnTouch;
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class UIDamageSystem : MonoBehaviour
{
    [SerializeField] private Player _player;
    [SerializeField] private CanvasGroup _redPanel;
```

```
    [SerializeField] private Canvas _deadScreen;
    [SerializeField] private Canvas _maineScreen;
    [SerializeField] private GameObject[] _hearts;

    [SerializeField] private float _rednessDuration = 0.3f;
    [Range(0, 1)]
    [SerializeField] private float _rednessMaxAlpha = 0.7f;

    public void GetDamaged()
    {
        LeanTween.alphaCanvas(_redPanel, _rednessMaxAlpha, _rednessDuration);
        LeanTween.alphaCanvas(_redPanel, 0, _rednessDuration).setDelay(_rednessDuration);
    }

    public void ShowHearts(int currentHealth)
    {
        if (_hearts == null || _hearts.Length == 0)
            return;

        for (int i = 0; i < 3; i++)
        {
            if (currentHealth > 0)
                _hearts[i].SetActive(true);
            else
                _hearts[i].SetActive(false);

            currentHealth--;
        }
    }

    public void Death()
    {
        _deadScreen.gameObject.SetActive(true);
        _player.gameObject.SetActive(false);
        _maineScreen.gameObject.SetActive(false);
    }
}

using System.Collections;
using UnityEngine;

[CreateAssetMenu(menuName = "Manager/Bank", fileName = "Bank")]
public class Bank : BaseDataLoader
{
    private string PPCoinName = Constants.PPname.CoinNumb;
    private int _coinNumb;

    public Bank()
    {
        Key = DataLoaders.Bank;
    }

    public override IEnumerator Init()
    {
        yield return new WaitForSeconds(0f);
        DataLoad();
    }

    public int GetCoin()
    {
        return _coinNumb;
    }
```

```
    public bool IsEnough(int number)
    {
        return _coinNumb >= number;
    }

    public void PluralIncreaseCoinNumb(int coin)
    {
        if (coin >= 0)
        {
            _coinNumb += coin;
            SaveData();
            GameEvent.ChangeCoinNumb?.Invoke();
        }
        else
        {
            Debug.LogError("You can't plural increase the number of coins by a negative number");
        }
    }

    public void ReduceCoinNumb(int coin)
    {
        if (coin >= 0)
        {
            if (IsEnough(coin))
            {
                _coinNumb -= coin;
                SaveData();
                GameEvent.ChangeCoinNumb?.Invoke();
            }
            else
            {
                Debug.LogError("You can't reduce the number of coins, not enough coins");
            }
        }
        else
        {
            Debug.LogError("You can't reduce the number of coins by a negative number");
        }
    }

    private void DataLoad()
    {
        if (PlayerPrefs.HasKey(PPCoinName))
        {
            _coinNumb = PlayerPrefs.GetInt(PPCoinName);
        }
        else
        {
            PlayerPrefs.SetInt(PPCoinName, 0);
            _coinNumb = PlayerPrefs.GetInt(PPCoinName);
        }
    }

    private void SaveData()
    {
        PlayerPrefs.SetInt(PPCoinName, _coinNumb);
    }
}

using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using UnityEngine.Advertisements;
using UnityEngine.Events;

[CreateAssetMenu(menuName = "Manager/AdsManager", fileName = "AdsManager")]
public class AdsManager : BaseDataLoader, IUnityAdsInitializationListener, IUnityAdsLoadListener
{
    public static bool RewardedAdIsLoad { get; private set; }
    public static UnityEvent<string> AdLoadChange = new UnityEvent<string>();

    [Header("AdsInit")]
    [SerializeField] private string _androidGameId;
    [SerializeField] private bool _testMode;

    [Header("AdsSettings")]
    [SerializeField] private static List<string> _adsUsed = new List<string>{ "Rewarded_Android" };

    public AdsManager()
    {
        Key = DataLoaders.AdsManager;
    }

    public void AdIncluded(string placementId)
    {
        if (placementId == _adsUsed[0])
            RewardedAdIsLoad = false;

        LoadAd(placementId);
        AdLoadChange.Invoke(placementId);
    }

    public override IEnumerator Init()
    {
        Advertisement.Initialize(_androidGameId, _testMode);
        yield return new WaitForSeconds(1f);
        LoadAds();
    }

    private void LoadAds()
    {
        foreach (string ad in _adsUsed)
        {
            LoadAd(ad);
        }
    }

    private void LoadAd(string adName)
    {
        Debug.Log("Loading Ad: " + adName);
        Advertisement.Load(adName, this);
    }

    public void OnInitializationComplete()
    {
        Debug.Log("Unity Ads initialization complete.");
    }

    public void OnInitializationFailed(UnityAdsInitializationError error, string message)
    {
        Debug.Log($"Unity Ads Initialization Failed: {error.ToString()} - {message}");
    }
```

```
    public void OnUnityAdsAdLoaded(string placementId)
    {
        if (placementId == "Rewarded_Android")
            RewardedAdIsLoad = true;

        AdLoadChange.Invoke(placementId);
        Debug.Log("Ad Loaded: " + placementId);
    }

    public void OnUnityAdsFailedToLoad(string placementId, UnityAdsLoadError error, string message)
    {
        Debug.Log($"Error loading Ad Unit {placementId}: {error.ToString()} - {message}");
    }

    public void OnUnityAdsShowClick(string placementId) {}
}

using Firebase.Analytics;
using System.Collections;
using UnityEngine;

[CreateAssetMenu(menuName = "Manager/FirebaseManager", fileName = "FirebaseManager")]
public class FirebaseManager : BaseDataLoader
{
    private Firebase.FirebaseApp _app;

    public FirebaseManager()
    {
        Key = DataLoaders.FirebaseManager;
    }

    public override IEnumerator Init()
    {
        Firebase.FirebaseApp.CheckAndFixDependenciesAsync().ContinueWith(task => {
            var dependencyStatus = task.Result;
            if (dependencyStatus == Firebase.DependencyStatus.Available)
            {
                // Create and hold a reference to your FirebaseApp,
                // where app is a Firebase.FirebaseApp property of your application class.
                _app = Firebase.FirebaseApp.DefaultInstance;

                // Set a flag here to indicate whether Firebase is ready to use by your app.
            }
            else
            {
                UnityEngine.Debug.LogError(System.String.Format(
                  "Could not resolve all Firebase dependencies: {0}", dependencyStatus));
                // Firebase Unity SDK is not safe to use here.
            }
        });

        yield return null;
    }

    private void GetLevelNumber()
    {
        Debug.Log(PlayerPrefs.GetInt(Constants.PPname.NumbActiveLevel));
    }

    public void StartLevel()
    {
        FirebaseAnalytics.LogEvent(FirebaseAnalytics.EventLevelStart,
```

```
            new Parameter(FirebaseAnalytics.ParameterLevelName,
PlayerPrefs.GetInt(Constants.PPname.NumbActiveLevel)));
    }

    public void EndLevel(float timeCompleteLevel)
    {
        FirebaseAnalytics.LogEvent(FirebaseAnalytics.EventLevelEnd,
            new Parameter(FirebaseAnalytics.ParameterLevelName,
PlayerPrefs.GetInt(Constants.PPname.NumbActiveLevel)),
            new Parameter("timeCompleteLevel", timeCompleteLevel),
            new Parameter("coins received", CoinManager.Instance.CoinNumber));
    }
}

using Lean.Localization;
using System;
using UnityEngine;
using UnityEngine.UI;

public class Fader : MonoBehaviour
{
    [SerializeField] private Animator _animator;
    [SerializeField] private LeanLocalToken _loadingPercent;
    [SerializeField] private Image _loadingProgressBar;
    private static Fader _instance;
    private const string FADER_PATH = "Fader";
    private const string FADER_ANIM_NAME = "faded";

    public bool isFading { get; private set;}

    public static Fader instance
    {
        get
        {
            if(_instance == null)
            {
                var prefab = Resources.Load<Fader>(FADER_PATH);
                _instance = Instantiate(prefab);
                DontDestroyOnLoad(_instance.gameObject);
            }
            return _instance;
        }
    }

    private Action _faderStartCallBack;
    private Action _faderEndCallBack;

    public void FadeStart(Action faderStartCallBack)
    {
        if(isFading)
        {
            return;
        }
        SpecialSceneLoader.instace.ChangeProgress += UpdateLoadingPercent;
        SpecialSceneLoader.instace.ChangeProgress += UpdateLoadingProgressBar;
        isFading = true;
        _faderStartCallBack = faderStartCallBack;
        _animator.SetBool(FADER_ANIM_NAME, true);
    }

    public void FadeEnd(Action faderEndCallBack)
    {
```

```
        if (isFading)
        {
            return;
        }
        isFading = true;
        _faderEndCallBack = faderEndCallBack;
        _animator.SetBool(FADER_ANIM_NAME, false);
        SpecialSceneLoader.instace.ChangeProgress -= UpdateLoadingPercent;
        SpecialSceneLoader.instace.ChangeProgress -= UpdateLoadingProgressBar;
    }

    private void UpdateLoadingPercent(float p)
    {
        _loadingPercent.SetValue((int)((p / 0.9) * 100));
    }

    private void UpdateLoadingProgressBar(float p)
    {
        _loadingProgressBar.fillAmount = (p / 0.9f);
    }

    private void FadeStartAnimationOver()
    {
        _faderStartCallBack?.Invoke();
        _faderStartCallBack = null;
        isFading = false;
    }

    private void FadeEndAnimationOver()
    {
        _faderEndCallBack?.Invoke();
        _faderEndCallBack = null;
        isFading = false;
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using UnityEngine.Serialization;

public class Preloader : MonoBehaviour
{
    private AsyncOperation operation;
    [SerializeField] private Animator _anim;
    [SerializeField] private Text _loadingPercent;
    [SerializeField] private Image _loadingProgressBar;
    private const string ANIM_TRIGGER_NAME = "FadeEnd";
    private const string LOADING = "Loading";

    private void LevelLoad()
    {
        StartCoroutine(Loader());
    }

    private IEnumerator Loader()
    {
        operation = SceneManager.LoadSceneAsync(MainLevelList.Menu.ToString());
        operation.allowSceneActivation = false;
        while (operation.progress < 0.89)
```

```
        {
            LoaderUI(operation.progress);
            yield return null;
        }
        _anim.SetTrigger(ANIM_TRIGGER_NAME);
        LoaderUI(operation.progress);
    }

    private void LoaderUI(float progress)
    {
        _loadingPercent.text = LOADING + (int)((progress / 0.9) * 100) + "%";
        _loadingProgressBar.fillAmount = progress / 0.9f;
    }

    private void FadeStartAnimationOver()
    {
        LevelLoad();
    }

    private void FadeEndAnimationOver()
    {
        operation.allowSceneActivation = true;
    }
}

using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SpecialSceneLoader : MonoBehaviour
{
    public static SpecialSceneLoader instace;
    public Action<float> ChangeProgress;

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

    public void LoadScene(string sceneName)
    {
        if (Fader.instance.isFading)
        {
            return;
        }
        StartCoroutine(ILoadScene(sceneName));
    }

    private IEnumerator ILoadScene(string sceneName)
    {
        var waitFading = true;
        Fader.instance.FadeStart(() => waitFading = false);

        while (waitFading)
```

```
        {
            yield return null;
        }

        var asyncOperation = SceneManager.LoadSceneAsync(sceneName);
        asyncOperation.allowSceneActivation = false;

        while(asyncOperation.progress < 0.9f)
        {
            ChangeProgress.Invoke(asyncOperation.progress);
            yield return null;
        }

        asyncOperation.allowSceneActivation = true;
        Fader.instance.FadeEnd(() => waitFading = false);

        while (waitFading)
        {
            yield return null;
        }
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ChangeMus : MonoBehaviour
{
    [SerializeField] private Slider _slider;
    [SerializeField] private Button _activeMusButton;
    [SerializeField] private Button _deactivationMusButton;

    private void Awake()
    {
        _slider.onValueChanged.AddListener(delegate { ChangeMusValue(); });
        _activeMusButton.onClick.AddListener(ChangeMusStatus);
        _deactivationMusButton.onClick.AddListener(ChangeMusStatus);
    }

    private void ChangeMusValue()
    {
        GameEvent.SoundEvents.ChangeSoundValue?.Invoke(_slider.value);
    }

    private void ChangeMusStatus()
    {
        GameEvent.SoundEvents.ChangeSoundOptions?.Invoke();
    }

    private void OnDestroy()
    {
        _slider.onValueChanged.RemoveAllListeners();
        _activeMusButton.onClick.RemoveAllListeners();
        _deactivationMusButton.onClick.RemoveAllListeners();
    }
}
using UnityEngine;
using UnityEngine.UI;

public class SetVisualSetting : MonoBehaviour
```

```
{
    [SerializeField] private GameObject _musOn;
    [SerializeField] private GameObject _musOff;
    [SerializeField] private Slider _musValue;
    [SerializeField] private Slider _sensValue;
    [SerializeField] private SoundOptions _soundOptions;

    private void Start()
    {
        SetActiveMusStatus();
        SetMusValue();
        SetSensValue();
    }

    private void SetActiveMusStatus()
    {
        if (_soundOptions.GetIsMusOn())
        {
            _musOn.SetActive(true);
            _musOff.SetActive(false);
        }
        else
        {
            _musOn.SetActive(false);
            _musOff.SetActive(true);
        }
    }

    private void SetMusValue()
    {
        _musValue.value = _soundOptions.GetMusValue();
    }

    private void SetSensValue()
    {
        _sensValue.value = PlayerPrefs.GetFloat(Constants.PPname.ControlSensivity);
    }
}

using UnityEngine;
using UnityEngine.UI;

public class SetVisualSetting : MonoBehaviour
{
    [SerializeField] private GameObject _musOn;
    [SerializeField] private GameObject _musOff;
    [SerializeField] private Slider _musValue;
    [SerializeField] private Slider _sensValue;
    [SerializeField] private SoundOptions _soundOptions;

    private void Start()
    {
        SetActiveMusStatus();
        SetMusValue();
        SetSensValue();
    }

    private void SetActiveMusStatus()
    {
        if (_soundOptions.GetIsMusOn())
        {
            _musOn.SetActive(true);
```

```
            _musOff.SetActive(false);
        }
        else
        {
            _musOn.SetActive(false);
            _musOff.SetActive(true);
        }
    }

    private void SetMusValue()
    {
        _musValue.value = _soundOptions.GetMusValue();
    }

    private void SetSensValue()
    {
        _sensValue.value = PlayerPrefs.GetFloat(Constants.PPname.ControlSensivity);
    }
}
using Lean.Localization;
using UnityEditor;
using UnityEngine;
using UnityEngine.UI;

public class ItemManager : MonoBehaviour
{
    [SerializeField] private int _price;
    [SerializeField] private int _skinNumb;
    [SerializeField] private Image[] _rareBg;
    [SerializeField] private Text[] _statusText;
    [SerializeField] private LeanLocalToken _priceToken;
    [SerializeField] private CoinTween _coinBuyTween;
    [SerializeField] private CoinTween _coinSellTween;
    [SerializeField] private Color _commonColor;
    [SerializeField] private Color _rareColor;
    [SerializeField] private Color _mythicalColor;
    private int _sellPrice;
    private int _status;
    private bool isPurchased = false;
    private string PPname;


    // Artem
    [SerializeField] private Rarity _rarity;
    public int Rarity => ((int)_rarity);

    private void Awake()
    {
        LoadDate();
    }

    private void Start()
    {
        CheckStatus();
        _sellPrice = _price / 3;
    }

    public void Buy()
    {
        if (DataLoadSystem.GetLoader<Bank>(DataLoaders.Bank).IsEnough(_price) && !isPurchased)
        {
            PlayerPrefs.SetInt(PPname, 1);
```

```
            PlayerPrefs.Save();
            DataLoadSystem.GetLoader<Bank>(DataLoaders.Bank).ReduceCoinNumb(_price);
            UpdateChange();
            GameEvent.SoundEvents.Shop.Buy?.Invoke();
            _coinBuyTween.OnBuy(gameObject.transform);
        }
        else if (isPurchased)
        {
            Select();
        }
    }

    public void TrySell()
    {
        if (isPurchased)
        {
            GameEvent.TrySell?.Invoke(this);
        }
    }

    public void Sell()
    {
        if (isPurchased)
        {
            GameEvent.SoundEvents.Shop.Sell?.Invoke();
            _coinSellTween.OnSell(gameObject.transform);

            if (IsSelectedSkin())
            {
                PlayerPrefs.SetInt(PPname, 0);
                DataLoadSystem.GetLoader<Bank>(DataLoaders.Bank).PluralIncreaseCoinNumb(_sellPrice);
                PlayerPrefs.SetInt("SelectedSkin", 0);
                GameEvent.SkinsUpdate?.Invoke();
                UpdateChange();
                CheckStatus();
            }
            else
            {
                PlayerPrefs.SetInt(PPname, 0);
                UpdateChange();
                _statusText[0].text = _price.ToString();
                DataLoadSystem.GetLoader<Bank>(DataLoaders.Bank).PluralIncreaseCoinNumb(_sellPrice);
            }
        }
    }

    public int GetSellPrice()
    {
        return _sellPrice;
    }

    public bool IsItemBought()
    {
        return isPurchased;
    }

    private void Select()
    {
        PlayerPrefs.SetInt("SelectedSkin", _skinNumb);
        CheckStatus();
        GameEvent.SkinsUpdate?.Invoke();
    }
```

```
private bool IsSelectedSkin()
{
   return PlayerPrefs.GetInt("SelectedSkin") == _skinNumb;
}

private void CheckStatus()
{
   UpdateStatus(_status);
}

private void UpdateStatus(int activeText)
{
   for (int i = 0; i < _statusText.Length; i++)
   {
      _statusText[i].gameObject.SetActive(i == activeText);
   }
}

private void LoadDate()
{
   PPname = "Material" + _skinNumb;
   _status = PlayerPrefs.GetInt(PPname, 0);
}

private void UpdateChange()
{
   LoadDate();
   CheckStatus();
}

private void RarCheck()
{
   switch (_rarity)
   {
      case Rarity.Common:
         UpdateBgColor(_commonColor);
         break;
      case Rarity.Rare:
         UpdateBgColor(_rareColor);
         break;
      case Rarity.Mythical:
         UpdateBgColor(_mythicalColor);
         break;
   }
}

private void UpdateBgColor(Color activeColor)
{
   foreach (Image bg in _rareBg)
   {
      bg.color = activeColor;
   }
}

#if UNITY_EDITOR
[ContextMenu("UpdateUI")]
private void UpdateUI()
{
   foreach (Image bg in _rareBg)
   {
      Undo.RecordObject(bg, "Undo color change");
```

```
            bg.color = _rarity switch
            {
                Rarity.Common => _commonColor,
                Rarity.Rare => _rareColor,
                Rarity.Mythical => _mythicalColor,
                _ => bg.color
            };
        }
        Undo.RecordObject(_statusText[0], "Undo status text change");
        _statusText[0].text = _price.ToString();
    }
    #endif
}
using UnityEngine;
using UnityEngine.UI;

public class CoinManager : MonoBehaviour
{
    [HideInInspector] public static CoinManager Instance;
    [SerializeField] private Text _coinNumbText;
    [SerializeField] private AudioSource _coinSound;
    private int _coinNumber;
    public int CoinNumber => _coinNumber;
    private bool _multiplyAbility = true;

    private void Awake()
    {
        Instance = this;
        _coinSound = GetComponent<AudioSource>();
    }

    private void Start()
    {
        Invoke("UpdateUI", 0.05f);
    }

    public int GetCoin()
    {
        return _coinNumber;
    }

    public void MultiplyCoin(Vector3 pos, int Multiply)
    {
        if (_multiplyAbility)
        {
            int MultiplyNumb = _coinNumber * (Multiply - 1);
            DataLoadSystem.GetLoader<Bank>(DataLoaders.Bank).PluralIncreaseCoinNumb(MultiplyNumb);
            TakePluralCoinTween.Instance.ScreenMove(pos, MultiplyNumb);
            _coinNumber *= Multiply;
            GameEvent.MultiplyCoin?.Invoke();
            _multiplyAbility = false;
        }
    }

    public void IncreaseCoinNumb(int Numb)
    {
        if (Numb >= 0)
        {
            _coinNumber += Numb;
            OnCoinTake();
        }
        else
```

```
        {
            Debug.LogError("Negative number of coins");
        }
    }

    private void OnCoinTake()
    {
        UpdateUI();
        PlayCoinSound();
    }

    private void UpdateUI()
    {
        _coinNumbText.text = _coinNumber.ToString();
    }

    private void PlayCoinSound()
    {
        _coinSound.Play();
    }
}

using UnityEngine;

[ExecuteAlways]
[SelectionBase]
public class Creator : MonoBehaviour
{
    [Min(1)]
    [SerializeField] private int _ount = 1;

    [SerializeField] private Vector3 _offset = Vector3.forward;
    [SerializeField] private Vector3 _rotation;

    private void Start()
    {
        if (Application.isPlaying(this))
        {
            CorrectChildCount();
            Destroy(this);
        }
    }

    private void Update()
    {
        CorrectChildCount();
    }

    private void CorrectChildCount()
    {
        _ount = Mathf.Clamp(_ount, 1, 1000);

        if (transform.childCount < _ount)
        {
            for (int i = transform.childCount; i < _ount; i++)
            {
                Transform instantiate = Instantiate(transform.GetChild(0), transform);
                instantiate.localPosition += (_offset * i);

                if (_rotation != Vector3.zero)
                    instantiate.rotation *= Quaternion.Euler(_rotation * Random.Range(0, 3));
            }
```

```
        }
        else if (_ount < transform.childCount)
        {
            for (int i = transform.childCount - 1; i >= _ount; i--)
            {
                DestroyImmediate(transform.GetChild(i).gameObject);
            }
        }
    }
}
using System;
using UnityEngine;
using UnityEngine.UI;

public class FinishLevel : MonoBehaviour
{
    [SerializeField] private GameObject _controllCanvas;
    [SerializeField] private GameObject _finishCanvas;
    [SerializeField] private Rigidbody _rigidbody;
    [SerializeField] private AudioSource _finishMus;
    [SerializeField] private Button _nextLevelButton;
    [SerializeField] private ParticleSystem _confetti;
    private int _numbLevel;
    private int _finishReward = 10;
    private bool _finished = false;

    private float _timeCompleteLevel;

    private void Awake()
    {
        _nextLevelButton.onClick.AddListener(NextLevel);
        _numbLevel = PlayerPrefs.GetInt(Constants.PPname.NumbActiveLevel);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.TryGetComponent(out SquashAndStretch squashAndStretch))
        {
            OnFinish();
        }
    }

    private void OnFinish()
    {
        if (_finished == false)
        {
            SetData();
            _confetti.gameObject.SetActive(true);
            _controllCanvas.SetActive(false);
            _rigidbody.isKinematic = true;
            _finishCanvas.SetActive(true);
            AddFinishReward(_finishReward);
            _finishMus.Play();

            _timeCompleteLevel = Time.timeSinceLevelLoad;

            _finished = true;
        }
    }

    public void NextLevel()
    {
```

```
        if (Enum.IsDefined(typeof(GameLevelList), _numbLevel + 1))
        {
            SpecialSceneLoader.instace.LoadScene(((GameLevelList)(_numbLevel + 1)).ToString());
        }
    }

    public void RestartButton()
    {
        SpecialSceneLoader.instace.LoadScene(((GameLevelList)_numbLevel).ToString());
    }

    private void SetData()
    {
        PlayerPrefs.SetInt(Constants.PPname.NumbActiveLevel, _numbLevel + 1);
        PlayerPrefs.Save();
    }

    private void AddFinishReward(int reward)
    {
        DataLoadSystem.GetLoader<Bank>(DataLoaders.Bank).PluralIncreaseCoinNumb(reward);
        CoinManager.Instance.IncreaseCoinNumb(reward);
        TakePluralCoinTween.Instance.WorldMove(this.transform.position, reward);
    }

    private void OnDestroy()
    {
        if (_finished == true)
        {
            FirebaseManager firebaseManager =
DataLoadSystem.GetLoader<FirebaseManager>(DataLoaders.FirebaseManager);
            firebaseManager.EndLevel(_timeCompleteLevel);
        }
        _nextLevelButton.onClick.RemoveListener(NextLevel);
    }
}
using UnityEngine;
using UnityEngine.Advertisements;
using UnityEngine.UI;

[RequireComponent(typeof(Button))]
public class RewardedAdButton : MonoBehaviour, IUnityAdsShowListener
{
    [SerializeField] private int _coinMultiplier = 2;
    [SerializeField] private string _placementId = "Rewarded_Android";

    private Button _button;
    private Vector3 _pos;

    private AdsManager _adsManager;

    private void Awake()
    {
        _button = GetComponent<Button>();
        _pos = transform.position;
        _button.interactable = false;
    }

    private void Start()
    {
        _adsManager = DataLoadSystem.GetLoader<AdsManager>(DataLoaders.AdsManager);
        AdsManager.AdLoadChange.AddListener(AdLoaded);
        AdLoaded(_placementId);
```

```
    }

    private void AdLoaded(string placementId)
    {
        if (_placementId.Equals(placementId))
        {
            if (AdsManager.RewardedAdIsLoad)
            {
                _button.interactable = true;
                _button.onClick.AddListener(ShowAd);
            }
        }
    }

    private void ShowAd()
    {
        Advertisement.Show(_placementId, this);
        _adsManager.AdIncluded(_placementId);
        _button.gameObject.SetActive(false);
    }

    public void OnUnityAdsShowFailure(string placementId, UnityAdsShowError error, string message)
    {
        Debug.Log($"Error showing Ad Unit {placementId}: {error.ToString()} - {message}");
    }

    public void OnUnityAdsShowStart(string placementId)
    {
        // Not needed for functionality, but required by interface
    }

    public void OnUnityAdsShowClick(string placementId)
    {
        // Not needed for functionality, but required by interface
    }

    public void OnUnityAdsShowComplete(string placementId, UnityAdsShowCompletionState showCompletionState)
    {
        if (placementId.Equals(_placementId) &&
showCompletionState.Equals(UnityAdsShowCompletionState.COMPLETED))
        {
            CoinManager.Instance.MultiplyCoin(_pos, _coinMultiplier);
            Debug.Log("Unity Ads Rewarded Ad Completed");
        }
    }

    private void OnDestroy()
    {
        AdsManager.AdLoadChange.RemoveListener(AdLoaded);
        _button.onClick.RemoveListener(ShowAd);
    }
}
```

ДОДАТОК Б

ВІДГУК

Керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

«Розробка комп'ютерної гри Bouncing-Ball в жанрі платформера на базі Unity Game engine та мови програмування C#»

Студента групи 122-20-3 Каракая Владислава Сергійовича

Керівник економічного розділу

доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файла	Опис
Пояснювальні документи	
Каракай_122-20-3.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Каракай_122-20-3.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Каракай_122-20-3.zip	Архів. Містить коди програми і откомпільовану програму
Презентація	
Каракай_122-20-3.ppt	Презентація кваліфікаційної роботи