МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Таврійський державний агротехнологічний університет
імені Дмитра Моторного
Енергетики і комп`ютерних технологій факультет

ЗАТВЕРДЖУЮ

Зав. каф. “Комп’ютерні науки”

доц._____ _____ Сергій ШАРОВ

«16»_червня__2025 р.

Пояснювальна записка

до дипломного проєкту здобувача СВО Бакалавр
(ступінь вищої освіти)

на тему: «Мобільна 2D гра під Android з використанням фізичного рушія на Unity»

52/4КНД.9685385.000000ПЗ

Виконав: здобувач ВО 4 курсу, групи 41КН
спеціальності 122 Комп`ютерні науки
за ОПП Ком`ютерні науки
(шифр і назва спеціальності та ОПП)

_____ Даніїл РАШЕВСЬКИЙ
(підпис)

Керівник доц. _____ Сергій ТЕРЕЩУК
(підпис)

Нормоконтроль, ст.викл. _____ Ольга ЗІНОВ’ЄВА
(підпис)

Консультант, доц._____ Лариса БОЛТЯНСЬКА____
(підпис)

Консультант, доц._____ Михайло ЗОРЯ____
(підпис)

Рецензент, доц._____ Дмитрій ВЕРБІВСЬКИЙ____
(підпис)

Запоріжжя - 2025 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Таврійський державний агротехнологічний університет імені Дмитра Моторного

Факультет: Енергетики і комп'ютерних технологій
Кафедра: Комп'ютерні науки
Ступінь вищої освіти: Магістр
Спеціальність: 122 Комп'ютерні науки
ОПП: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри_КН
к.пед.н., доц. _____Сергій ШАРОВ
"10" _жовтня__2024__ року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

_Рашевський Даніїл Олексійович_____
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Мобільна 2D гра під Android з використанням фізичного рушія на Unity»
керівник проєкту __Терещук С.І д.пед.н.,доцент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом університету від "30" вересня_2024_року №_452-С
2. Строк подання здобувачем вищої освіти роботи __12 червня 2025_____
3. Вихідні дані до роботи: технічне завдання на дипломне проєктування, результати виробничої практики, матеріали обстеження об`єкта дослідження, статистичні дані щодо розвитку ігрової індустрії, науково-технічна література, електронні ресурси, матеріали з відкритих джерел щодо огляду відеоігор та ін.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)____
___1. Опис предметної області та аналіз існуючих рішень._____
___2. Специфікація вимог до гри._____
___3. Вибір та обґрунтування технологій розробки гри.___
___4. Дослідна експлуатація гри._____
___5. Тестування гри._____
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників)
Діаграма використання, діаграми IDEF0, розкадровка інтерфейсу, структурна схема, візуальні компоненти інтерфейсу, алгоритм роботи, інтерфейс програми
Презентація – 11 слайдів
1. Титульний слайд
2. Предмет, об'єкт, мета

3. Завдання дослідження
4. Порівняльна характеристика аналогів програмного продукту
5. Діаграма варіантів використання
6. Інструментальні засоби
7. Інтерфейс і робота гри
8. Інтерфейс і робота гри
9. Тестування
10. Висновок
11. Дякую за увагу

6. Консультанти розділів кваліфікаційної роботи

Розділ/ Підрозділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4.3	Болтянська Л.О.	15.10.2025	15.06.2025
4.4	Зоря М.В.	15.10.2025	15.06.2025

7. Дата видачі завдання___10 жовтня 2024 р._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз предметної області	03.10.2024	виконано
2	Специфікація вимог до гри	17.10.2024	виконано
3	Проектування гри	28.10.2024	виконано
4	Обґрунтування інструментальних засобів	18.11.2024	виконано
5	Розробка гри	16.12.2024	виконано
6	Тестування та налагодження гри	11.04.2025	виконано
7	Підпис керівником роботи	14.06.2025	виконано
8	Підпис завідувачем кафедри	16.06.2025	виконано

Здобувач вищої освіти _____ Данїїл РАШЕВСЬКИЙ _
(підпис) (власне ім’я та прізвище)

Керівник кваліфікаційної роботи _____ _Сергій ТЕРЕЩУК____
(підпис) (власне ім’я та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра: 92 с., 18 рис., 7 табл., 26 посилань, 1 додаток.

Актуальність. Мобільні ігри є однією з найпопулярніших форм розваг у світі. Особливо затребувані 2D-ігри, які поєднують простоту й динамічний геймплей. Використання фізичного рушія Unity дозволяє створювати більш реалістичні та цікаві ігрові механіки, що робить тему розробки актуальною в умовах постійного зростання ринку мобільних ігор.

Об'єкт дослідження: процес розробки мобільної гри для платформи Android.

Предмет дослідження: мобільна 2D гра з фізичним рушієм, розроблена на базі Unity.

Мета роботи: створити мобільну 2D гру під Android з використанням фізичного рушія Unity для покращення якості ігрового процесу та підвищення зацікавленості користувачів.

Задачі дослідження:

1. Проаналізувати існуючі технології та інструменти для розробки мобільних 2D ігор.
2. Вивчити особливості використання фізичного рушія Unity для створення ігрової механіки.
3. Розробити технічне завдання на створення мобільної 2D гри під Android.
4. Реалізувати мобільну гру із застосуванням мови програмування C# та інструментів Unity.
5. Провести тестування гри на мобільних пристроях та оптимізувати її продуктивність.

Практичне значення роботи полягає у створенні готової мобільної 2D гри під Android, яка може бути використана як приклад для подальших розробок у галузі ігрової індустрії. Розроблена гра демонструє можливості використання фізичного рушія для створення реалістичних взаємодій у 2D

середовищі, а також може слугувати основою для комерційних або навчальних проєктів.

Ключові слова: МОБІЛЬНА ГРА, 2D, ANDROID, UNITY, ФІЗИЧНИЙ РУШІЙ, ІГРОВА МЕХАНІКА, ПРОГРАМУВАННЯ.

ABSTRACT

Master's qualification work: 92 pages, 18 figures, 7 tables, 26 references, 1 appendix.

Relevance. Mobile games are one of the most popular forms of entertainment in the world. 2D games, in particular, are in high demand due to their simplicity and dynamic gameplay. The use of the Unity physics engine allows for the creation of more realistic and engaging game mechanics, making the topic of this research relevant in the context of the constantly growing mobile gaming market.

Object of research: the process of developing a mobile game for the Android platform.

Subject of research: a mobile 2D game with a physics engine developed using Unity.

Purpose of the work: to create a mobile 2D game for Android using the Unity physics engine to improve the quality of gameplay and increase user engagement.

Research objectives:

1. To analyze existing technologies and tools for developing mobile 2D games.
2. To study the features of using the Unity physics engine for creating game mechanics.
3. To develop technical specifications for creating a mobile 2D game for Android.
4. To implement the mobile game using the C# programming language and Unity tools.
5. To conduct game testing on mobile devices and optimize its performance.

Practical significance of the work lies in the creation of a ready-to-use mobile 2D game for Android, which can serve as an example for further developments in the gaming industry. The developed game demonstrates the possibilities of using the physics engine to create realistic interactions in a 2D environment and can serve as a foundation for commercial or educational projects.

Keywords: MOBILE GAME, 2D, ANDROID, UNITY, PHYSICS ENGINE, GAME MECHANICS, PROGRAMMING

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку цифрових технологій особливої популярності набуває сфера розробки мобільних застосунків, зокрема мобільних ігор. Мобільні ігри займають провідні позиції серед цифрових продуктів завдяки доступності смартфонів, простоті поширення та великій кількості користувачів. Ігрова індустрія активно використовує сучасні технології та інструменти для створення якісних продуктів, що забезпечують високу продуктивність, привабливу графіку та цікаву ігрову механіку.

Серед доступних засобів розробки ігор особливе місце посідає Unity — потужний кросплатформенний ігровий рушій, що дозволяє ефективно створювати як 2D, так і 3D ігри. Однією з важливих переваг Unity є наявність вбудованого фізичного рушія, за допомогою якого можна реалізувати реалістичні взаємодії між об’єктами в грі. Саме використання фізичних ефектів сприяє підвищенню якості ігрового процесу та залученості користувачів.

Метою даної дипломної роботи є розробка **мобільної 2D гри під Android із використанням фізичного рушія Unity**, яка поєднуватиме динамічний ігровий процес, зрозумілий користувацький інтерфейс та оптимальну продуктивність. Особливу увагу приділено реалізації фізичних взаємодій, які є ключовою складовою ігрової механіки.

Об’єктом дослідження є процес створення конкретного мобільного програмного продукту —2D гри для платформи Android.

Предметом дослідження є методи, засоби та інструменти розробки мобільної гри із використанням фізичного рушія Unity.

Для досягнення поставленої мети необхідно вирішити такі основні завдання:

1. Проаналізувати існуючі технології та інструменти для розробки мобільних 2D ігор.

2. Вивчити особливості використання фізичного рушія Unity для створення ігрової механіки.
3. Розробити технічне завдання на створення мобільної 2D гри під Android.
4. Реалізувати мобільну гру із застосуванням мови програмування C# та інструментів Unity.
5. Провести тестування гри на мобільних пристроях та оптимізувати її продуктивність.

Практичне значення роботи полягає у створенні повноцінного програмного продукту — мобільної 2D гри під Android, що демонструє можливості використання фізичних ефектів для покращення ігрового процесу. Розроблена гра може бути використана як завершений проєкт або стати основою для подальшого розвитку чи комерційного застосування.

Структурно дипломна робота складається зі вступу, чотирьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто теоретичні основи розробки мобільних ігор та аналіз відповідних технологій. У другому розділі наведено технічне завдання, обґрунтовано вибір інструментів і спроєктовано архітектуру гри. Третій розділ присвячено реалізації гри з використанням фізичного рушія Unity. У четвертому розділі наведено результати тестування гри, оцінку продуктивності та перспективи розвитку проєкту.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1 Узагальнена характеристика предметної області

Електронна комерція - це динамічна галузь внутрішньої економіки. Він ґрунтується на передових послугах у торгових угодах, стандартизації, вдосконаленій юридичній рамці та інноваційних інформаційних технологіях. Як результат, спостерігається постійне зростання на ринку електронної комерції як у всьому світі, так і в Україні. Електронна комерція може працювати на трьох рівнях: Перша торгівля на першому рівні. Другий рівень - е-комерція; Третій рівень - це бізнес. Найважливіші учасники електронної комерції включають бізнес чи бізнес (Business – B), споживач (Consumer – C), уряд (Government – G). Існує багато варіацій взаємодії цих учасників на сьогоднішньому електронному ринку [6]. Електронна комерція має багато функцій. Це означає, що комфорт, доброчесність, безпека, координація бізнесу та доступність спеціальних пристроїв та програмного забезпечення, що дозволяє забезпечити безпечний та надійний доступ до Інтернету [6].

Зростаюча популярність електронної комерції також сприяла зростанню ринку реклами в цьому сегменті, але рекламні бюджети оптимізовані. Що стосується формування стратегій розвитку ринку електронної комерції, компанія повинна дотримуватися загальних правил. Післявоєнне відновлення повинно ґрунтуватися на економічному ослабленні, а фінансова підтримка не є альтернативою. Гранти - це також можливість початку, масштабування та розвитку бізнесу, і багато з цих програм є як міжнародними, так і державними донорами, які працюють в Україні. Для підтримки ринкової позиції та залучення зобов'язань з клієнтами підприємці використовують творчі підходи та впроваджують інновації. Конкурентоспроможність, зменшення корупції, доступ до найсучасніших послуг [11].

Одним з найважливіших аспектів розвитку ігрової індустрії є її тісний зв’язок з електронною комерцією. На перший погляд ці області можуть здатися різними, але вони насправді активно взаємодіють і доповнюють один одного. Однією з найбільш вражаючих форм взаємодії є покупки в грі (покупки в грі), коли гравці купують додатковий вміст, такий як фактичні шкури фінансування, зброя, персонажі, рівні чи інші ігрові переваги. Це класичний приклад використання електронної комерції в ігрових умовах. Іншим проявом є розробка цифрових платформ продажів, таких як Steam, Epic Games Store та PlayStation Store. Це працює за принципами інтернет-магазинів. Вони продають цифрові копії гри та додатковий вміст безпосередньо для кінцевих користувачів через Інтернет. Крім того, популярні послуги підписки, такі як Xbox Game Pass, PlayStation Plus та Apple Arcade. Гравці створить оплачувані підписки для доступу до бібліотеки ігор. Це також форма електронної комерції, де фінансові операції відбуваються лише в цифрових умовах. Ще одним важливим компонентом є онлайн -продаж продуктів, пов’язаних із відомими іграми, такими як відомі ігри, одяг , аксесуари, плакати, іграшки та інші продукти через електронні платформи. Новим напрямком було впровадження технології NFT та blockchain в іграх, які дозволяють користувачам купувати, продавати або обмінюватися унікальними цифровими активами. Це також прояв електронної комерції. Варто згадати, що ми розглядаємо роль рекламних платформ у вільних іграх та стратегіях монетизації. Реклама, спеціальні пропозиції та покупки в грі організовані відповідно до класичних принципів електронної комерції, які створюють нові потоки доходу для розробників. Тому електронна комерція стала одним з найважливіших рушій у розробці сучасної ігрової індустрії, пропонуючи не лише нові варіанти монетизації, але й принципово нові бізнес-моделі в галузі розваг.

Ігрова індустрія - одна з найбільш динамічно зростаючих сфер сучасної економіки. Завдяки швидкому розвитку мобільних технологій та зростанню популярності смартфона, мобільні ігри є головним сегментом ринку

відеоігор. Сучасні ігрові додатки не тільки забезпечують розваги, але й активно використовуються в галузі освіти, маркетингу та соціальної взаємодії. Комп'ютерні ігри виграли дуже потужні розробки і нещодавно були розроблені дуже швидко як у графіці, так і в сюжеті. Комп'ютерні ігри, які вийшли в останні роки, дали їм ретельних персонажів. Кожен гравець може створити унікальний персонаж і спробувати весь одяг, щоб його віртуальний аватар ідеально відповідав його стилю [1].

Ігри - найскладніший продукт на медіа-ринку. Це мистецтво, яке поєднує в собі кінотеатри та математику і, безсумнівно, дуже вигідний бізнес, а також малювання та письмо. З використанням різних методів експлуатації в суспільстві завжди робилися спроби споживачів. Близько 2008 року було щось незворотне - був попит на безкоштовні ігри. На перший погляд, все робиться для задоволення покупця. не повинен платити за гру. Все безпосередньо доступно, щоб вийти з гри. Це безкоштовні ігри, які сприяли моделі монетизації як внутрішньої гри в додатковий вміст. Спочатку гравця «годують» винагородою, дають йому багато ресурсів і просто демонструють свою щедрість. Rephrase Насправді це повинно зробити гравців якомога більше, коли їм дуже важко піти. І коли нагорода позбавляє геометричну прогресію, пропозиція починає купувати ігрові ресурси та "зберегти" час. Це практик оперативного бізнесу в найпростішому сенсі [7].

Мобільні ігри - це не остання позиція в ігровій індустрії. З відносно низькою вартістю розробки та обслуговування мобільних ігор, доходів від реклами та використанням варіантів пожертв для високих лояльності в цільовій групі, розробка мобільних ігор правильно вважається галуззю в грайливому галузі з найвищими націнками. Варто також зазначити, що відносна легкість розвитку ігор для мобільних телефонів у присутності навичок навіть може дозволити іншому розробнику без команди. Мережа знайде всю галактику інді-розробників, які успішно створили гру [2].

Слід зазначити, що мобільні програми (програмне забезпечення) можна розділити на розваги (мультимедіа), комунікацію, навігацію, посилання та

програми. Програмне забезпечення для мобільних розваг включає аудіоплеєри та відеофайли, фотографії, електронні книги та ігри. Програма зв'язку відповідає за спілкування користувачів по телефону та SMS [4].

Мобільні інформаційні технології можуть використовуватися в освітніх процесах. Наприклад, необхідно забезпечити використання мобільної системи MLMS для професійного навчання майбутніх викладачів з комп'ютерних технологій:

– проведення навчально-адміністративної роботи: складання навчальних груп, розкладу занять, формування звітів;

– контроль пройденого матеріалу; оцінка навчальних досягнень студентів; роботу в асинхронному режимі з можливістю індивідуального підходу до кожного студента;

– колективну роботу студентів і викладача (вебінар, конференція);

– підтримку електронної пошти, форуму, чату, відеоконференцій, обміну файлами, повідомленнями, спільного використання додатків, віртуальної аудиторії;

– розподіл учасників навчального процесу за ролями: гість, студент, викладач, адміністратор;

– підтримку різних типів навчальних матеріалів – електронних підручників, тестів, симуляцій та лабораторних робіт [4].

У той час як студенти та студенти все частіше використовують мобільні телефони, планшети та інші пристрої, основною метою яких є розваги та ігри в номінованих категоріях населення, можливості користуватися набагато ширшими. З цієї причини типовий вчитель формування середньої та університетської формації - це завдання забезпечити навчальний процес з високоякісною електронною підготовкою. Він також пропонує не тільки комп’ютери, але й інші сучасні пристрої, які можна запропонувати в інших місцях, як у загальних школах, так і в університетських закладах. І сучасні технології, такі як віртуальні, web та хмарні, можуть допомогти змінити

навчальні середовища та зробити освіту більш доступною (вища або більш поширена) [5].

Розробка мобільних ігор безпосередньо пов'язана з вдосконаленням апаратної підтримки мобільних пристроїв та програмного забезпечення для розробки ігор. Створіть якісні ігри, використовуючи спеціальні інструменти або драйвери ігор, які називаються "двигунами". Ігровий двигун є центральним програмним забезпеченням для всіх відеоігор, яке відповідає за весь технічний аспект, і сприяє розробці гри за допомогою спільної систематизації Союзу та його внутрішньої структури. Найвідоміший і вдосконалений двигун - Unreal Engine. Unreal Engine - ігровий рушій, який був розроблений та підтримується Epic Games. [1]. Unity - найпопулярніший рушій для створення 2D та 3D-ігор, які підтримують міжплатформу та інтегровані фізичні драйвери, які забезпечують реалістичні взаємодії предметів у грі.

1.2 Аналіз мобільних 2D ігор під Android з використанням фізичного рушія на Unity

Сьогодні існують численні мобільні ігри, створених на рушії Unity, впроваджуючи фізичні моделі поведінки об’єктів. Найпоширенішими є аркади, головоломки та симулятори, де фізика відіграє важливу роль у взаємодії з ігровим світом.

Більшість сучасних мобільних 2D ігор під Android, створених з використанням фізичного рушія на платформі Unity, орієнтовані на забезпечення інтерактивного ігрового досвіду з використанням реалістичних фізичних моделей. Такі ігри зазвичай реалізують базові принципи фізики: гравітацію, зіткнення, тертя, інерцію та інші взаємодії між об’єктами, що дозволяє гравцеві відчути природність рухів та дій в ігровому середовищі.

Індивідуальні класи таких ігор мають помітну симуляційну або головоломну складову. Це робить фізичні моделі основою геймплею (наприклад, будівництво мостів, вантажний транспорт, збалансовані предмети тощо). У таких проєктах головне завдання полягає не лише в утриманні користувачів, а й у прийнятті стратегічних рішень або вирішенні нестандартних завдань з урахуванням фізичних обмежень.

Unity пропонує достатньо можливостей у галузі створення фізики: від використання вбудованого рушія Box2D до застосування кастомних моделей моделювання. Це дозволяє реалізовувати складні та реалістичні фізичні взаємодії як розробникам-інді, так і великим студіям – навіть за умов обмежених ресурсів, включно з мобільними пристроями.

Більшість цих ігор технічно насичені, незважаючи на простоту візуального дизайну. Вони демонструють хорошу оптимізацію фізики під мобільні платформи, використовують ефективне управління ресурсами та стабільно працюють на різних Android-пристроях. З цієї причини дослідження подібних ігор є надзвичайно важливим при створенні власного проєкту. Це допомагає краще зрозуміти фізичне моделювання, інтерфейсні рішення та підходи до балансування геймплею.

Після аналізу доступних мобільних 2D ігор із фізичними компонентами рекомендуємо створити власну гру на мобільному пристрої на основі рушія Unity.

Poly Bridge – це симулятор будівництва мостів, розроблений новозеландською інді-студією, з музикою канадського композитора Адріана Таленса. Гравці будують мости, переміщуючи частини конструкції [19]. Гра ускладнюється наявністю різних будівельних матеріалів за різними цінами (рис. 1.1.).

Рисунок 1.1 – Логотип Poly Bridge

Мета гри – побудувати міст, щоб транспортні засоби могли його перетнути. Це робиться за допомогою двовимірної моделі мосту шляхом створення креслення та маніпулювання матеріалами, наданими під час гри. Poly Bridge має режим кампанії з численними сценаріями, які вимагають виконання різних завдань та вводять конкретні географічні особливості. Кожен рівень (понад 100) має дві вимоги: витрати на будівництво повинні бути в межах бюджету, а міст має бути достатньо міцним для певної кількості транспортних засобів. Також є режим пісочниці, який дозволяє вільно будувати без обмежень і встановлювати параметри конструкції. Існує різноманітність транспортних засобів – від мотоциклів (швидких і легких) до вантажівок і кранів (повільних і важких). Оскільки кожен транспортний засіб має різний кузов і вагу, довші машини можуть не змогти підняти круті схили, тому конструкція може потребувати адаптації до ситуації. У грі також є перешкоди, такі як трампліни та човни, що дозволяє гравцеві творчо будувати свій міст. Погано побудований міст обвалюється під вагою транспортного засобу, що перетинає його. Гра включає генератор GIF-анімацій, який дозволяє гравцям записувати й ділитися певними моментами під час гри.

Mini Motor Racing – це ізометрична гоночна гра, розроблена The Binary Mill для платформ iOS та Android на рушії Unity [18]. Гра включає

транспортні засоби, гоночну фізику та фізику навколишнього середовища, вдосконалення автомобілів, щоденні завдання, денні та нічні режими, кар'єрний режим, швидкі гонки, багатокористувацьку гру, таблиці лідерів та менеджер ігрового центру (рис. 1.2.).

Рисунок 1.2 – Логотип Mini Motor Racing

Mini Motor Racing використовує iCloud від Apple для синхронізації прогресу на всіх пристроях з iOS 5 і новіших. Замість того, щоб контролювати прискорення та гальмування, користувачі повинні освоїти потужні слайди та своєчасне використання азоту для досягнення успіху. Автомобілі регулярно врізаються в навколишнє середовище і руйнують його, однак самі транспортні засоби не ушкоджуються. Гра використовує модифіковану версію фізичного двигуна Unity для імітації цих зіткнень.

Гравець закінчує гонку на різних позиціях, отримуючи випадкові бонуси. Вони можуть бути використані для придбання нового автомобіля або вдосконалення вже наявного, розблокованого в кар'єрному режимі.

Гра підтримує HD-графіку, що відображається в ізометричному форматі, тоді як автомобілі та траса є повністю 3D-моделями. Фізичне

управління, як у Play Sony Xperia, не підтримується, але деякі елементи управління доступні для користувачів.

У Mini Motor Racing використовується Apple Game Center для соціальних ігор, що забезпечує рейтингові таблиці та досягнення на будь-яких трасах. Багатокористувацький режим підтримується через Wi-Fi і Bluetooth, що дозволяє грати до чотирьох гравців через Wi-Fi та до двох через Bluetooth.

Automatoys – це гра-головоломка, розроблена Idle Friday [12]. У ній гравець веде м’яч через серію складних тривимірних машин. Гра була першим незалежним релізом провідного розробника Стеффана Глінна, колишнього члена State of Play Games. Після випуску Automatoys отримала неоднозначні відгуки: похвалу за дизайн рівнів і ігровий процес, а також критику за коротку тривалість (рис. 1.3.).

Рисунок 1.3 – Логотип Automatoys

Гравець веде м’яч через серію машин під назвою «Автоматої», щоб дістатися до кінця траси. Керування здійснюється одним дотиком: активуються, переміщуються та нахиляються різні механізми, включно з пандусами та катапультами, причому інтенсивність деяких механізмів змінюється натисканням і утриманням. На трасі є кілька перешкод, які змушують м’яч падати, через що гравцю доводиться починати рівень

спочатку. За проходження рівнів гравець отримує рейтинг із трьох зірок залежно від швидкості проходження. У грі 12 рівнів: перші три безкоштовні, а решта дев’ять доступні після придбання гри.

Size Does Matter – це інді-гра в жанрі ритм-індустрії, розроблена DOS Studios та опублікована Channel 4 Television для iOS та Android [21]. Гравці переміщують і змінюють розмір набору блоків, щоб заробляти очки, проходячи крізь рухомі перешкоди (рис.1.4.).

Рисунок 1.4 – Логотип Size Does Matter

Гравець керує набором блоків, які можна переміщувати та змінювати розмір, рухаючи пальцем вгору або вниз по лівому і правому боках сенсорного екрану. Гравець отримує очки за ідеальне проходження проміжків між перешкодами. Роблячи це кілька разів поспіль, він створює комбо і збільшує кількість очок за кожен успішний прохід. Якщо ж гравець натрапляє на перешкоду, він отримує удар, і комбо переривається. Існують бонуси, такі як «Slow-Mo», який уповільнює час на кілька секунд, і «Body Double», що додає ще один стовпець блоків і дозволяє гравцеві отримувати подвійні очки.

Кожен рівень має бонусні послідовності, які можна пройти, якщо зробити менше трьох ударів. Вони базуються на тій самій пісні, але складніші. Оцінка, отримана в кожній частині рівня, об’єднується, щоб сформувати підсумковий рахунок.

Кожен рівень заснований на сегменті пісні в стилі електронної танцювальної музики (EDM). У грі представлені пісні Jonas Eide, Filter, Christian Wojky, Eirika Zurke, Raimdregurag, Savant та Chipzel. Музика, 21 озвучить на головному екрані, – це «Living iPod» гурту Savant.

Спочатку гра під назвою Size Does Matter була створена для студентського конкурсу з геймдизайну Dare to Be Digital 2013, який проводився протягом дев'яти тижнів в Університеті Абертей. Команда складалася з п'яти студентів Норвезької школи інформаційних технологій. Mattis Delerud був керівником команди та художником, Silje Dahl і Lars Evre Andersen – геймдизайнерами, а Nick La Roy і Trond Fasteraune – програмістами. Відео розробки та ранні демоверсії показують, що гра була створена за допомогою ігрового рушія Unity. Під час розробки був створений внутрішній редактор рівнів, протестовано кілька схем введення, а також розроблено візуальні ефекти для підкреслення ритму пісень.

Прототип було представлено на фестивалі Dare ProtoPlay з 8 по 11 серпня в Caird Hall, Данді, де він отримав премію Channel 4 Award. Наступного року гру опублікували в Apple App Store, Google Play Store та Amazon Appstore.

World of Goo – це відеогра, заснована на фізиці, розроблена та опублікована незалежною студією 2D Boy [26]. У грі гравці використовують невеликі кульки липкої речовини для створення конструкцій, подібних до мостів, щоб допомогти іншим кулькам досягти цільової точки та виконати завдання – побудувати цю структуру.

Гра була номінована на численні нагороди – головну премію Сеймаса МакНеллі, нагороду за інновації в дизайні та технічну досконалість – на фестивалі Independent Games Festival , а також виграла кілька інших ігрових нагород, таких як «Видатні досягнення в ігровому дизайні» на 12-й щорічній церемонії вручення премії Interactive Achievement Awards. Гра отримала схвальні відгуки критиків і стала одним із найперших прикладів комерційно успішної інді-гри (рис. 1.5.).

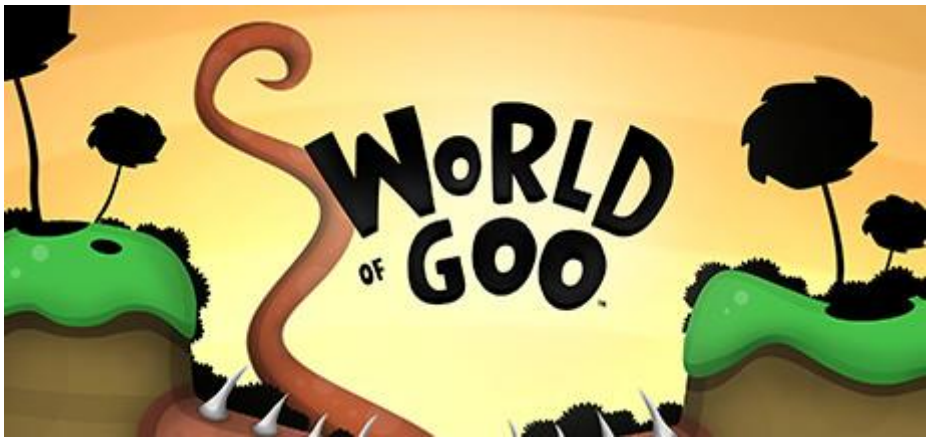

Рисунок 1.5 – Логотип World of Goo

Гра отримала кілька нагород, зокрема головну премію Сеймаса МакНеллі, нагороду за інновації в дизайні та технічну досконалість на фестивалі Independent Games Festival, а також інші відзнаки. Гра поділена на п’ять розділів, кожен із яких містить кілька рівнів. Кожен рівень має унікальні графічні та музичні теми, що створюють неповторну атмосферу. Існує також бонусна мета-гра під назвою World of Goo Corporation. У ній гравці будують найвищу вежу з кульок липкої речовини, які збирають під час проходження гри. Гравці з усього світу можуть конкурувати, оскільки висота вежі та кількість використаних кульок постійно завантажуються на сервер 2D Boy.

Основна мета гри – доставити необхідну кількість кульок липкої речовини до труби, яка символізує вихід. Для цього гравці використовують кульки для будівництва мостів, веж та інших споруд, щоб подолати різні перешкоди, такі як прірви, пагорби, шипи, вітряки та скелі. У грі є кілька типів кульок липкої речовини, кожна з яких має унікальні властивості. Гравці повинні комбінувати їх, щоб пройти всі рівні. Додаткові кульки, зібрані в трубі, додаються до World of Goo Corporation – пісочниці, де гравці змагаються у будівництві найвищої вежі. Також можна спробувати досягти «Прапора нав’язливого завершення» для кожного рівня, виконуючи рівень за більш суворими умовами, наприклад, зібрати більше кульок, пройти рівень за певний час або використати якомога менше ходів.

Версія для WiiWare включає багатокористувацький режим, у якому можуть грати до чотирьох гравців на одній консолі Wii. Ця функція також доступна у порті для Linux, хоча не підтримується офіційно.

Таблиця 1.1. –Порівняння ігор

Назва гри	Характеристика, основні функції та задачі гри
Mini Motor Racing	Аркадна 2D-гонка з виглядом зверху. Гравець керує маленькими автомобілями на різноманітних трасах з фізикою дрифтингу, прискорення, гальмування та зіткнень. Реалістична фізика руху забезпечує цікавий геймплей і моделювання взаємодії машин з трасою.
Poly Bridge	Симулятор побудови мостів з використанням фізики напруги, стискання та гравітації. Гравець проєктує мости, які мають витримати вагу транспортних засобів. Доступна система тестування і навчання з аналізом слабких місць конструкції.
Automatoys	Головоломка з інтерактивними механізмами, у якій гравець проводить кульку крізь складні рівні, активуючи важелі, штовхачі, платформи. Фізичний рушій відповідає за точну динаміку руху об'єктів та реакцію на взаємодію.
World of Goo	Фізична головоломка, де гравець будує структури з кульок «гуду», щоб дістатися до виходу. Гравітація, баланс і деформації відіграють ключову роль у геймплеї. Гра отримала безліч нагород і була високо оцінена критиками.
Size Does Matter	Ритм-гра, в якій гравець керує блоками, змінюючи їх розмір та позицію, щоб пройти через перешкоди в ритмі музики. Гра отримала нагороду BAFTA Ones to Watch 2014.

Отже, можна зробити висновок, що на ринку мобільних 2D ігор під Android, створених на рушії Unity з використанням фізичних моделей, представлено чимало різноманітних проєктів, які відрізняються як за жанром, так і за рівнем складності реалізації фізики. Серед них є як аркадні гонки

(Mini Motor Racing), логічні ігри на побудову (Poly Bridge) та фізичні головоломки (World of Goo, Automatoys), так і нестандартні ритм-ігри (Size Does Matter), де фізика поєднується з музичним ритмом. Більшість цих ігор демонструють потужні можливості Unity у створенні інтерактивного, фізично точного ігрового середовища. Це підтверджує актуальність використання фізичних рушіїв у мобільних 2D-проєктах. Тому ми поставили перед собою завдання – створити власну мобільну 2D-гру на Unity з фізичною моделлю, яка буде поєднувати простоту керування, інтуїтивний геймплей і якісну фізику, з урахуванням досвіду аналізованих ігор.

1.3. Технічне завдання на розробку “Мобільної 2D-гри під Android з використанням фізичного рушія на Unity”

Технічне завдання (ТЗ) – це основоположний документ, який визначає вимоги до створення програмного продукту. У контексті розробки мобільної 2D-гри під Android, технічне завдання слугує детальним описом функціональних і нефункціональних вимог, що дозволяє узгодити бачення замовника та здобувача вищої освіти щодо майбутнього програмного засобу. ТЗ допомагає структурувати процес розробки, уникнути непорозумінь під час виконання робіт, визначити етапи реалізації та критерії приймання проєкту. Крім того, технічне завдання виступає базою для подальшого тестування та оцінки якості програмного забезпечення. Саме на основі ТЗ здійснюється контроль відповідності кінцевого продукту заявленим вимогам.

Зміст

1. Вступ

1.1 Загальні визначення

Повна назва програмного продукту: мобільна 2D гра-платформер для Android з використанням фізичного рушія Unity.

Замовник: особа чи організація, що замовляє розробку гри.

Виконавець: розробник гри, який займається програмуванням, дизайном та тестуванням (у даному випадку – здобувач вищої освіти).

Unity: ігровий рушій, що використовується для розробки гри.

Фізичний рушій Unity Physics 2D: компонент Unity для реалізації фізичних взаємодій у 2D просторі.

1.2 Призначення документу

Даний документ містить повний перелік вимог до розробки, тестування та впровадження мобільної 2D гри-платформера для Android на рушії Unity. У ньому виконавець та замовник погоджують обсяг робіт, технічні характеристики, функціональні можливості та строки реалізації проекту. Усі уточнення та додаткові вимоги після підписання документу узгоджуються окремо.

2. Призначення розробки

2.1 Призначення і мета розробки програмного засобу

Метою розробки є створення мобільної 2D гри-платформера з використанням фізичного рушія Unity Physics 2D для реалістичної взаємодії ігрових об’єктів. Гра повинна мати зручне сенсорне керування, різноманітні рівні із унікальним дизайном, механіки стрибків, переміщення та взаємодії з перешкодами, а також систему збереження прогресу. Гра розрахована на пристрої з ОС Android.

2.2 Характеристика об’єктів автоматизації

Об’єктом автоматизації є ігровий процес мобільного платформера: рух персонажа, взаємодія з елементами середовища (перешкоди, бонуси), відображення графіки, анімацій і звукових ефектів. Автоматизація дозволяє керувати грою за допомогою сенсорного екрану, забезпечує фізично коректну поведінку об’єктів та зберігає прогрес гравця.

3. Вимоги до програмного засобу

3.1 Вимоги до оформлення інтерфейсу

Інтерфейс повинен бути простим, зручним і адаптованим під сенсорне керування. Кнопки управління (рух, стрибок, взаємодія) мають бути розташовані так, щоб забезпечити комфортне керування на екранах різних розмірів. Меню гри повинні містити елементи вибору рівня, налаштувань звуку та графіки, а також можливість збереження та завантаження гри.

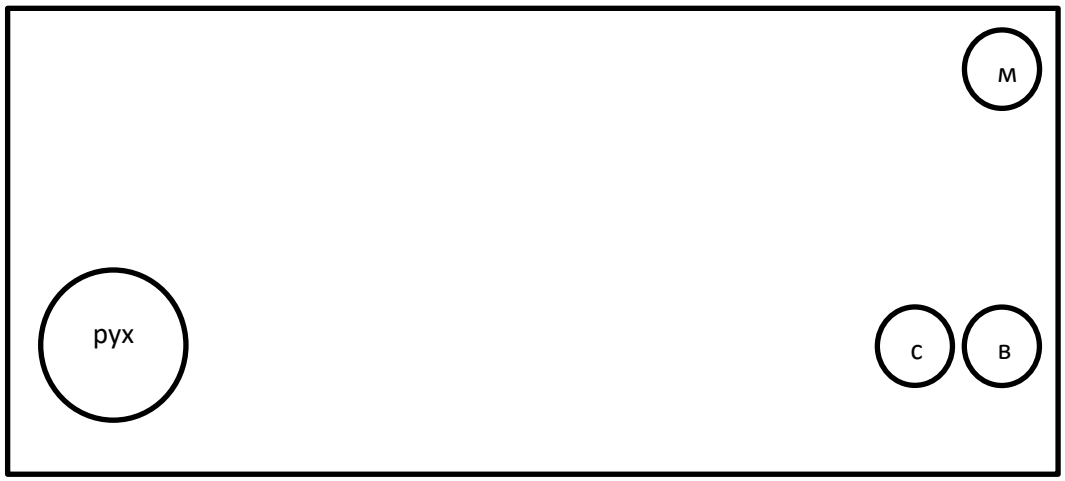

Рисунок 2.1 – Інтерфейс управління гри

3.2 Вимоги до зберігання даних і мов програмування

Дані гри (прогрес, налаштування) зберігатимуться локально на пристрої у форматі JSON або PlayerPrefs (Unity). Розробка здійснюється на рушії Unity з використанням мови C. Використовуються стандартні бібліотеки Unity для роботи з фізикою, анімаціями та звуком.

3.3 Вимоги до розділення доступу

Гра не передбачає складної системи користувачів, проте при необхідності може реалізовуватись зберігання кількох профілів гравців. Авторизація не є обов'язковою.

3.4 Вимоги до програмного та технічного забезпечення

Гра повинна коректно працювати на мобільних пристроях з ОС Android версії 7.0 і вище. Потрібна підтримка роздільної здатності від 720p і вище. Забезпечення плавної роботи з кадровою частотою не менше 30 FPS. Використання фізичного рушія Unity Physics 2D для обробки зіткнень та гравітації. Оптимізація для пристроїв середнього рівня продуктивності.

4. Стадії і етапи розробки програмного засобу

4.1 Склад і зміст робіт із розробки програмного засобу

1. Аналіз концепції гри, механіки та цільової аудиторії.
2. Вибір технологій та інструментального середовища – Unity, C.
3. Проектування ігрових рівнів і механік.
4. Розробка ігрових об'єктів, фізичних моделей та анімацій.
5. Створення інтерфейсу користувача та системи керування.
6. Тестування гри на різних пристроях Android.
7. Виправлення виявлених помилок і оптимізація.
8. Підготовка гри до публікації (пакування, сертифікація).

5. Порядок контролю і приймання

5.1 Перевірка працездатності роботи програми

1. Перевірка коректності роботи сенсорного керування.
2. Тестування фізичної взаємодії об'єктів (зіткнення, гравітація).
3. Перевірка проходження рівнів, збереження та завантаження прогресу.
4. Тестування роботи меню, звукових ефектів та анімацій.

5. Перевірка сумісності з різними пристроями Android.

5.2 Вимоги до введення програмного засобу в дію

1. Підготовка інсталяційного пакету для Android (APK-файл).
2. Завантаження та встановлення гри на тестові пристрої.
3. Проведення демонстрації роботи гри відповідальним особам (замовнику).
4. Надання інструкцій щодо користування та обслуговування гри.

РОЗДІЛ 2. СПЕЦИФІКАЦІЯ ВИМОГ ДО “МОБІЛЬНОЇ 2D ГРИ ПІД ANDROID З ВИКОРИСТАННЯМ ФІЗИЧНОГО РУШІЯ НА UNITY”

2.1 Глосарій

Під час проектування ми будемо використовувати список основних скорочень і термінів, наведений у табл. 2.1.

Таблиця 2.1 – Глосарій

Термін	Опис терміну
1	2
1 Основні поняття та категорії проекту	
Програмне забезпечення	комплекс програм для обробки інформації разом із програмною документацією, яка потрібна для їх використання
Unity	Кросплатформний рушій для розробки ігор і інтерактивного контенту, що підтримує 2D та 3D графіку, використовується для створення мобільних, комп’ютерних і веб-ігор.
Фізичний рушій (Physics Engine)	Програмний модуль, який моделює фізичні взаємодії між об’єктами в грі (зіткнення, гравітація, тертя тощо).
2D-гра	Гра, в якій візуальні елементи та рухи реалізовані у двовимірному просторі.
Android	Операційна система з відкритим кодом, розроблена Google для мобільних пристроїв.
Скрипт	Невелика програма або модуль, що керує поведінкою об’єктів у грі. У Unity скрипти пишуться переважно мовою C#.

Продовження таблиці 2.1

C#	Об'єктно-орієнтована мова програмування, яка використовується для написання скриптів у Unity.
Ігровий об’єкт (GameObject)	Базовий об’єкт у Unity, до якого додаються компоненти (фізика, рендеринг, скрипти тощо).
Компонент (Component)	Окремий функціональний модуль, який додається до GameObject і визначає його поведінку.
Сцена (Scene)	Окремий рівень або частина гри в Unity, яка містить набір об’єктів.
Collider	Компонент у Unity, який визначає область взаємодії об’єкта з іншими в ігровому середовищі.
Rigidbody	Компонент у Unity, який дозволяє об’єкту підпорядковуватися фізичним законам, таким як гравітація та сила.
Інтегроване середовище розробки (IDE)	Програмне середовище, яке забезпечує розробку програм (напр. Visual Studio для C#).
Android SDK	Набір інструментів для створення додатків під Android.
Prefab	Заздалегідь створений шаблон об’єкта, який можна багаторазово використовувати у різних сценах.
UI (User Interface)	Інтерфейс користувача, який забезпечує взаємодію з грою через елементи, як-от кнопки, панелі, тексти.
FPS (Frames Per Second)	Кількість кадрів, що відображаються на екрані щосекунди; показник продуктивності гри.

Продовження таблиці 2.1

Assets	Ресурси в Unity, які використовуються у грі (зображення, звуки, моделі, скрипти).
2 Користувачі системи	
Гравець (User)	Кінцевий користувач, який взаємодіє з грою.
Розробник	Особа, що створює, налаштовує і підтримує гру.
Тестувальник	Особа, що перевіряє працездатність гри, знаходить помилки і повідомляє про них розробникам.
3 Вхідні та вихідні документи	
Технічне завдання	(Product Requirements Document; PRD) – документ, що встановлює основне призначення, показники якості, техніко-економічні та спеціальні вимоги до виробу, обсягу, стадії розроблення та складу конструкторської документації. Є вихідним документом для проектування споруди (архітектурного комплексу), конструювання технічного пристрою (приладу, машини тощо), розробки автоматизованої системи чи інформаційної системи, створення програмного продукту, проведення науково-дослідних робіт (НДР) і дослідно-конструкторських робіт (ДКР). Встановлює основні вимоги до робіт в певній сфері діяльності, обов’язки замовника і виконавця, показники якості, терміни проведення і звітності та економічні показники послуг. Цей документ використовують окремо або у складі договору(контракту) на виконання робіт.

2.2 Специфікація функціональних та нефункціональних вимог

Функціональні вимоги описують основні функції, які повинна реалізовувати мобільна гра. Вони охоплюють ігрову логіку, інтерфейс користувача, роботу з фізичним рушієм та інші компоненти системи.

Таблиця 2.2 - Специфікація функціональних вимог

Ідентифі-катор вимоги	Назва вимоги (варіанта використання)	Атрибути вимог		
		пріоритет	складність	Контакт/виконавець
1	2	3	4	5
1	Головне меню гри	обов’язкове	середнє	Розробник
2	Початок нової гри	обов’язкове	середнє	Розробник
3	Завантаження збереженої гри	рекомендоване	низька	Розробник
4	Управління ігровим персонажем	обов’язкове	висока	Розробник
5	Реалізація фізичних взаємодій (Unity Physics2D)	обов’язкове	висока	Розробник
6	Рівнень гри (дизайн та логіка проходження)	обов’язкове	висока	Розробник
7	Система підрахунку балів / очок	обов’язкове	середнє	Розробник
8	Збереження ігрового прогресу	рекомендоване	низька	Розробник

Продовження табл. 2.2

1	2	3	4	5
9	Екран поразки та перемоги	обов’язкове	середнє	Розробник
10	Музика та звукові ефекти	рекомендоване	середнє	Розробник
11	Налаштування гучності та управління	рекомендоване	низька	Розробник
12	Інтеграція з Google Play для публікації	рекомендоване	низька	Розробник

У таблиці 2.3 вказані нефункціональні вимоги до системи

Таблиця 2.3. Нефункціональні вимоги

№	Назва вимоги	Характеристика
1	Платформа	Гра повинна працювати на Android-пристроях з версією Android 8.0 і вище
2	Зрозумілість інтерфейсу	Інтерфейс гри має бути інтуїтивно зрозумілим та зручним для користувача
3	Продуктивність	Гра повинна забезпечувати стабільну частоту кадрів (FPS) не нижче 30
4	Надійність	Гра не повинна аварійно завершуватись при нормальному використанні
5	Розмір додатку	Розмір APK не повинен перевищувати 200 МБ
6	Безпека	Дані користувача (за наявності збережень) повинні зберігатися локально без стороннього доступу

Продовження табл. 2.3

7	Мова програмування / платформа	Гра розробляється на Unity з використанням C#
8	Вимоги до графіки	Підтримка 2D графіки з анімацією спрайтів
9	Використання фізики	Необхідна реалізація фізичних взаємодій між об’єктами за допомогою Physics2D
10	Тестування	Проведення модульного та інтеграційного тестування

2.3 Концептуальна модель використання мобільної гри

У мові єдиного моделювання (UML) варіанти використання можуть узагальнити деталі користувачів системи (також відомі як актори) та поєднувати взаємодію з системою. Для створення такої схеми ви використовуєте багато спеціальних символів та з'єднань. Ефективні варіанти діаграми допомагають командам обговорити та показувати:

- Сценарії, в яких ваша система або додаток взаємодіє з людьми, організаціями чи зовнішніми системами
- Цілі, яких ваша система або додаток допомагає досягти цим суб’єктам (відомим як актори)
- Межі вашої системи

Діаграма використовуваних варіантів не передає багато деталей. Наприклад, я не думаю, що порядок кроків буде відображатися. Натомість добре побудована схема варіантів використання забезпечує огляд використання відносин з використанням та системами. Експерти рекомендують використовувати діаграми використання як доповнення до більш детального текстового опису сценаріїв використання.

UML - це ряд інструментів моделювання, які дозволяють створювати діаграми. Використовувані параметри відображаються у вигляді овалу з написом. Актори процесу згадуються числами та участю в системі, тобто межами між акторами та варіантами додатків. Прямокутник намальовано навколо використання для визначення системних обмежень. Діаграми варіантів використання в UML ідеально підходить для:

- представлення цілей взаємодії між користувачем і системою;
- визначення та структурування функціональних вимог у систем;
- специфікації контексту та вимог до системи;
- моделювання базового перебігу подій у варіанті використання

Щоб відповісти на запитання: "Що таке діаграма варіантів використання?", спочатку потрібно зрозуміти її основні елементи. Загальні компоненти включають:

– Актори: Користувачі, які взаємодіють із системою. Актором може бути людина, організація або зовнішня система, яка взаємодіє з вашим застосунком чи системою. Вони повинні бути зовнішніми об’єктами, що створюють або споживають дані.

– Система: Це певна послідовність дій і взаємодій між акторами та самою системою. Систему також можуть називати сценарієм.

– Цілі: Кінцевий результат більшості варіантів використання. Успішна діаграма повинна описувати дії та їхні варіації, які ведуть до досягнення цілі [23].

Випадки використання є основним інструментом UML для моделювання функціональних вимог системи. Вони зосереджуються на тому, що повинна робити система (поведінка), а не на тому, як це реалізується. Діаграма показує взаємодію між акторами (користувачами або іншими системами) та випадками використання (функціональні можливості системи). Вона допомагає передати поведінку системи з точки зору кінцевих користувачів і визначити її межі та контекст.

Випадки використання повинні бути простими та загального рівня, без надмірної кількості елементів – якщо на діаграмі більше ніж 20 випадків використання, це може свідчити про її неправильне використання. Такі діаграми допомагають визначити контекст системи, переглянути архітектуру та підтримують створення тестових сценаріїв.

Зв’язки між випадками використання, такі як include, extend та generalization, забезпечують повторне використання та кращу структуру. Include додає спільну поведінку до кількох випадків, extend відображає додаткову або умовну поведінку, а generalization підтримує наслідування між випадками використання.

Моделювання випадків використання виконується на ранніх етапах розробки та зазвичай створюється аналітиками разом із галузевими експертами. Воно виникло ще до появи UML – його вперше запропонував Івар Якобсон у 1980-х роках, і з того часу воно стало широко використовуваним способом фіксації функціональних вимог до системи [25].

Дослідження актуальності. У контексті інформаційного суспільства інформація є одним із головних ресурсів. Зростання обсягів інформації в геометричній прогресії актуалізує потребу в розробці та застосуванні методів її обробки і захисту, а також створенні відповідних програмних рішень. Для задоволення інформаційних потреб користувачів було розроблено прикладні програмні продукти – інформаційні системи (ІС). Їх використання дозволяє оперативно отримувати достовірну інформацію у визначеній предметній області, для якої вони безпосередньо були створені.

Процес проектування програмного забезпечення передбачає створення специфікацій на основі сформованих вимог, які визначаються ще до початку розробки. Як наслідок, об’єкт проектування, в даному випадку – інформаційна система, повинен бути детально описаний у вигляді узгоджених функціональних та інформаційних моделей. Модель у цьому контексті – це опис системи з певної точки зору, який використовується для візуалізації, опису та документування її характеристик. Використання таких

моделей допомагає краще зрозуміти структуру та поведінку майбутньої програмної системи, а також зменшити потенційні ризики на етапах її розробки та подальшої експлуатації.

Однією з сучасних методологій розробки інформаційних систем є технологія CASE (Computer-Aided Software Engineering – інженерія програмного забезпечення за допомогою комп'ютерних засобів). Основною метою CASE-технологій є розмежування процесу проектування програмного продукту та процесу кодування, а також максимальна автоматизація розробки програмного забезпечення. Програмні засоби, які підтримують процеси створення та супроводу інформаційних систем відповідно до інформаційних потреб користувачів, отримали назву CASE-засобів. Їх використання дозволяє зменшити кількість потенційних помилок на різних етапах життєвого циклу розробки програмного забезпечення [3].

Розробка діаграми варіантів використання переслідує мети:

– визначити спільні кордони та контекст модельованої предметної області на початкових етапах проектування системи;

– сформулювати загальні вимоги до функціонального поведінки проектованої системи;

– розробити вихідну концептуальну модель системи для її подальшої деталізації у формі логічних і фізичних моделей;

– підготувати вихідну документацію для взаємодії розробників системи з її замовниками і користувачами.

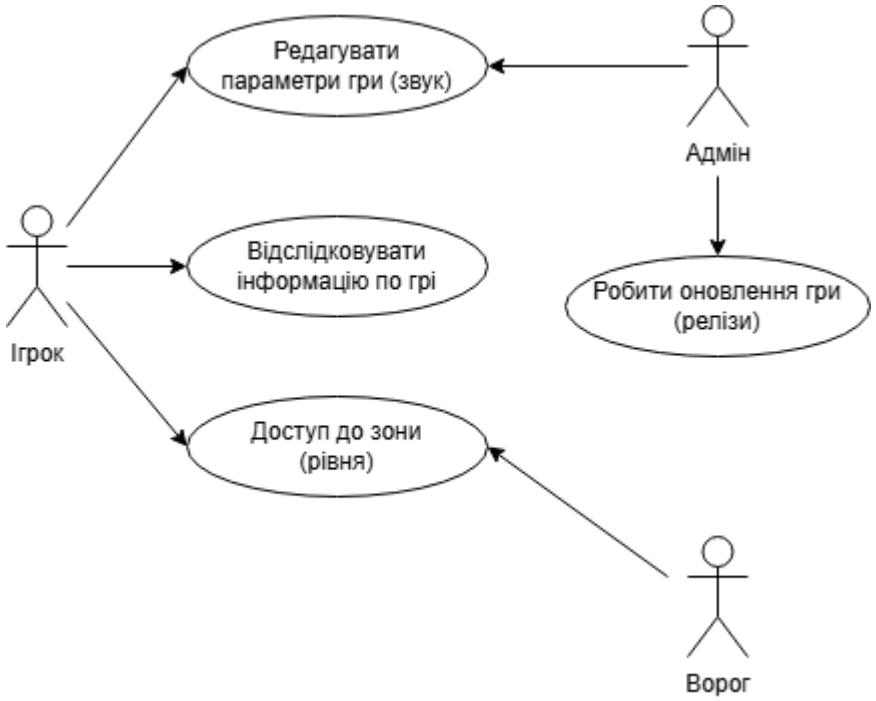

Рисунок 2.1 – Діаграма варіантів використання

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей або акторів, які взаємодіють з системою за допомогою, так званих варіантів використання. При цьому актором (actor) або дійовою особою називається будь-яка сутність, що взаємодіє з системою ззовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка може служити джерелом впливу на модельовану систему так, як визначить сам розробник. У свою чергу, варіант використання (use case) служить для опису сервісів, які система надає акторові. Іншими словами, кожен варіант використання визначає певний набір дій, який чинять системою при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодія акторів із системою.

У мові UML всі уявлення про моделі складних систем фіксуються за допомогою спеціальних графічних конструкцій, які називаються діаграмами. У цьому контексті діаграма означає графічне зображення сукупності елементів моделі у вигляді зв’язного графа, де вершинам і ребрам (або дугам) надається певна семантика.

На сьогодні UML є загальноприйнятим стандартом документування процесу розробки інформаційних систем та програмного забезпечення, який затверджено консорціумом OMG. До програмних засобів, що підтримують роботу з UML, належать IBM Rational Rose, Telelogic TAU G2, Microsoft Visio тощо. Мову UML широко застосовують для проєктування реляційних баз даних, зокрема через використання діаграм класів. У термінології UML діаграма класів – це діаграма, що демонструє набір класів, зв'язки між ними, а також може містити коментарі та обмеження [9].

На сучасному етапі розвитку інформаційних технологій широкого поширення набули інформаційні системи, призначені для задоволення інформаційних потреб кінцевих користувачів. Процес проєктування окремих модулів системи та їх поведінки має передувати безпосередньо розробці інформаційної системи. Одним із засобів моделювання структури та функціональності інформаційної системи є уніфікована мова моделювання UML, зокрема діаграма класів.

Об'єктивні труднощі, пов'язані з формалізованим описом предметної області та побудовою повних, функціонально-несуперечливих інформаційних моделей, стали підґрунтям для появи програмно-технологічних засобів (CASE-засобів), використання яких дозволяє підвищити ефективність процесу розробки програмного забезпечення. CASE-технологія – це сукупність методів проєктування програмного забезпечення та інструментальних засобів, які дозволяють наочно моделювати предметну область і здійснювати аналіз моделі на всіх етапах розробки та супроводу.

Уніфікована мова моделювання UML (Unified Modeling Language) є універсальним інструментом об'єктно-орієнтованого моделювання, розробка якого розпочалася в середині 1990-х років. UML визнано загальноприйнятим стандартом документування процесу створення інформаційних систем і програмного забезпечення. У межах UML модель складної системи представляється у вигляді діаграм – графічних зображень елементів моделі.

У UML клас визначається як іменований опис сукупності об’єктів, що мають однакову структуру, властивості та взаємозв’язки з іншими об’єктами. Атрибут – це іменована властивість класу, яка описує множину можливих значень. Операція – це іменована дія, яку можна виконати над об’єктом класу. Зв’язки між класами можуть бути представлені як залежність, узагальнення або асоціація. Особливою формою асоціації є агрегація, що описує відношення типу «частина–ціле», та композиція, при якій знищення цілого об’єкта тягне за собою знищення його частин.

Таким чином, UML і діаграми класів дозволяють наочно спроєктувати об’єкти предметної області, їх властивості, поведінку та взаємозв’язки, що суттєво спрощує розробку інформаційних систем [10].

2.4 Розробка функціональної моделі мобільної 2D гри під Android з використанням фізичного рушія на Unity

Функціональна модель розробляється за методологією IDEF0 для визначення функціональних вимог мобільної 2D гри для Android з використанням фізичного рушія в Unity. Мета цієї моделі -ідентифікувати основні функції гри, включно з вхідними та вихідними даними на всіх етапах ігрового процесу, а також визначити взаємодії та інформаційні потоки. Функціональна модель чітко окреслює архітектуру гри та демонструє, як різні підсистеми - ігровий рушій, користувацький інтерфейс, фізичні розрахунки - взаємодіють між собою і яку роль виконує кожна з них [22].

Модель IDEF0 відображає ігрову систему як набір взаємопов’язаних функцій, кожна з яких відповідає за певне завдання в грі. Спочатку вся гра подається на контекстній діаграмі (A0), яка демонструє загальний ігровий процес та взаємодію програми із зовнішніми об’єктами - гравцем, налаштуваннями гри та апаратним забезпеченням пристрою [15]. Цей структурований підхід з використанням IDEF0 дає змогу детально зрозуміти,

як дані та управління проходять через систему гри, що дозволяє оптимізувати продуктивність, ефективно розподіляти ресурси та забезпечити плавну взаємодію між компонентами. [22].

До системи надходять команди гравця (жести, дотики), стан гри та параметри фізичного рушія. Вихідними даними є візуальна реакція на екрані, звукові ефекти та оновлений стан гри. Управління включає правила гри, обмеження фізичного рушія та рівень складності. Механізми - це рушій Unity, фізичний рушій, апаратне забезпечення Android-пристрою та інші компоненти, що забезпечують виконання функцій гри. [22, 15].

Контекстна схема містить кілька важливих елементів: введення, вихід, управління та механізми. Вхід – це інформація, яку користувач надає системі для виконання функцій. Це можуть бути дані користувача або параметри для пошуку житла (місто, дати, ціни тощо) чи запити на бронювання. Вихідні – це результати роботи системи: підтвердження бронювання, список доступних об'єктів або інформація про статус бронювання. Управління – це інформація, яка визначає, як і скільки потрібно виконати завдань, наприклад, правила бронювання, перевірка доступності об'єктів на конкретні дати або умови для скасування бронювання. Механізми – це ресурси, які виконують функції: сервери, бази даних, інтерфейс користувача та адміністратори, які забезпечують роботу системи.

Після створення контекстної діаграми кожен із функціональних блоків системи (наприклад, процес бронювання житла) деталізується на підфункції. Кожна підфункція може бути описана окремими блоками для глибшого аналізу. Таким чином можна уточнити, як усі етапи процесу взаємодіють між собою, які дані використовуються та які ресурси необхідні. Наприклад, блок «Оформлення бронювання» можна поділити на такі підфункції: пошук житла та перегляд деталей обраного об'єкта. Кожну з цих підфункцій також можна деталізувати на дрібніші етапи для визначення конкретних вимог та механізмів їх реалізації.

IDEF0 допомагає не лише розробити загальну архітектуру системи, а й деталізувати всі її частини, щоб зрозуміти, які дані та ресурси використовуються в кожному процесі. У результаті можна отримати функціональну модель, яка дає чітке уявлення про функціонування веб-застосунку для бронювання об'єктів, визначає його компоненти та їхню взаємодію (рис. 2.2).

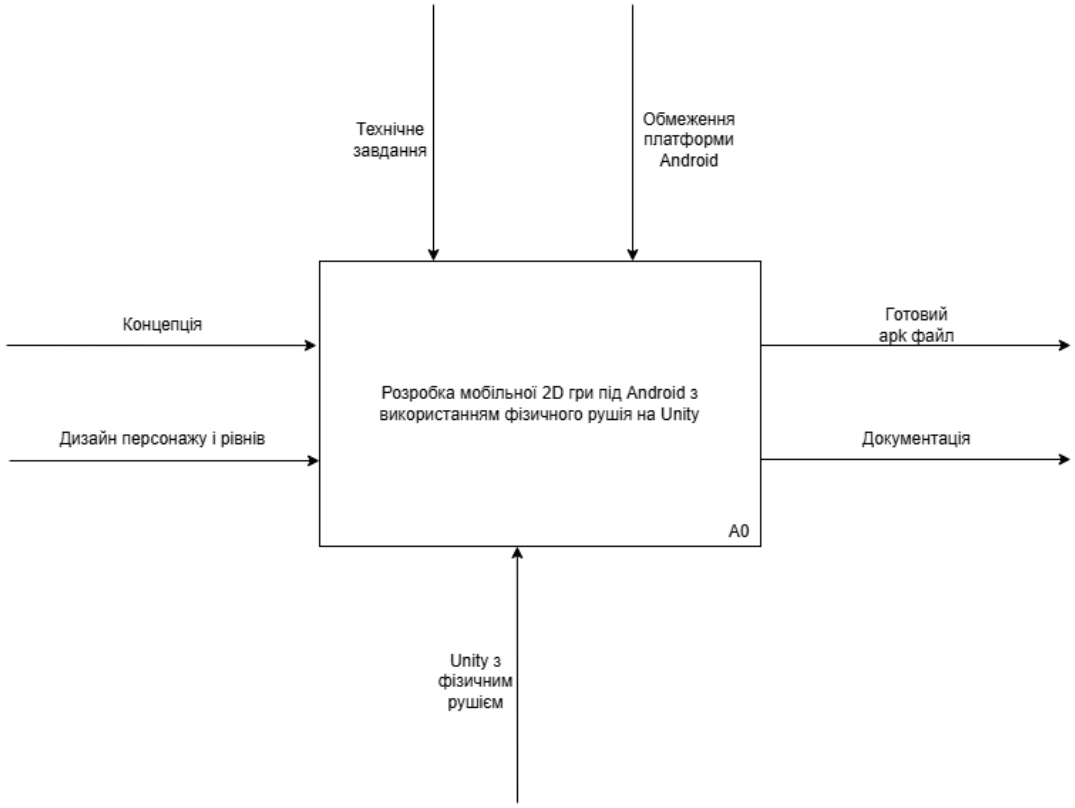

Рисунок 2.2 – Діаграма IDEF0

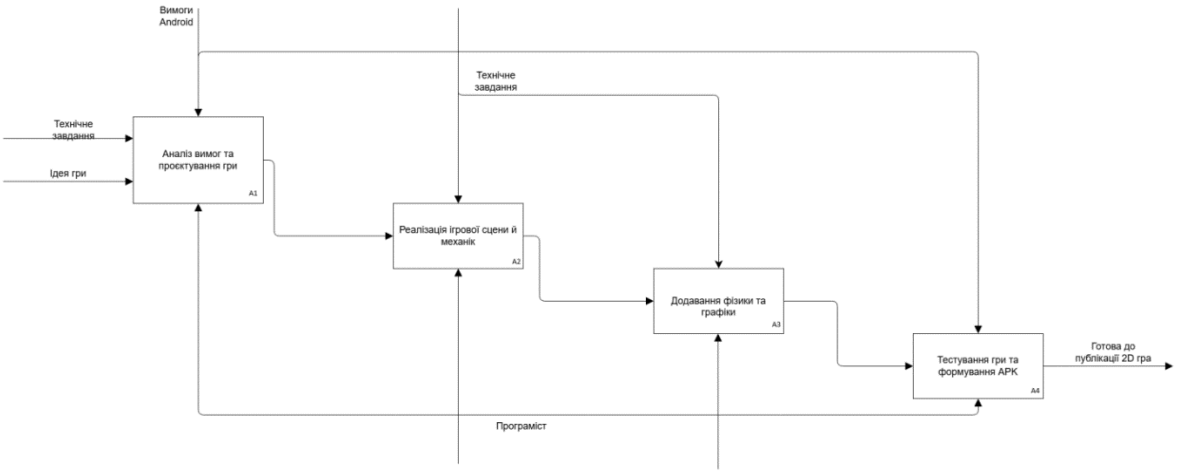

Рисунок 2.3 – Діаграма декомпозиції

РОЗДІЛ 3. ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ ТА ТЕХНОЛОГІЧНИХ РІШЕНЬ

3.1 Розробка об’єктної моделі

Діаграма класів, яка зображена на рисунку 3.3 (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відбивати, зокрема, різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти і підсистеми, а також описує їхню внутрішню структуру і типи відносин.

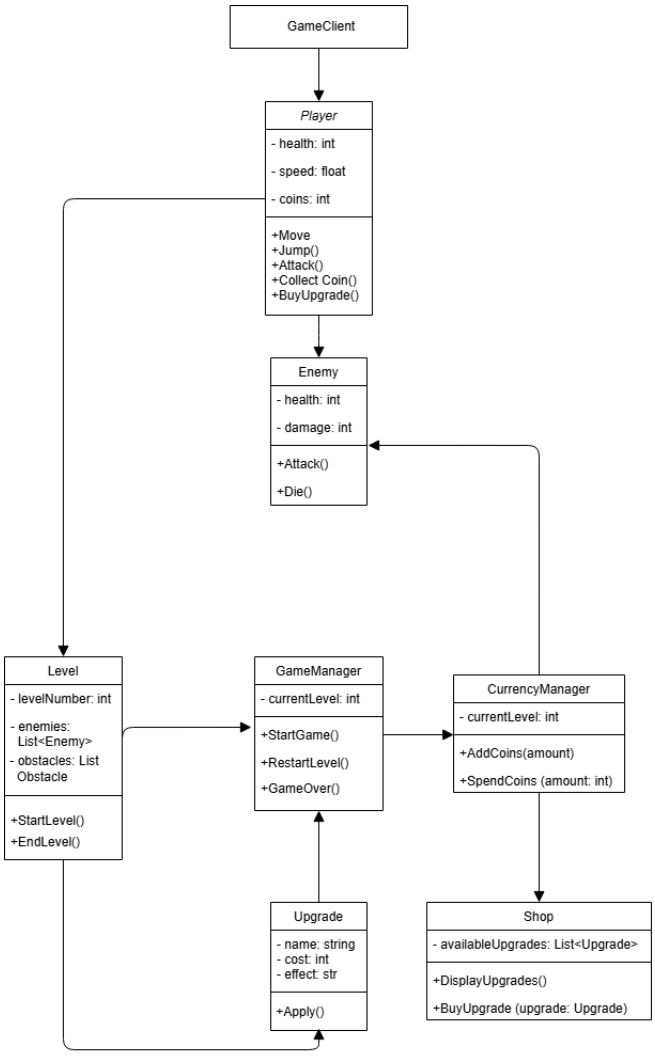

Рисунок 3.3 – Діаграма класів

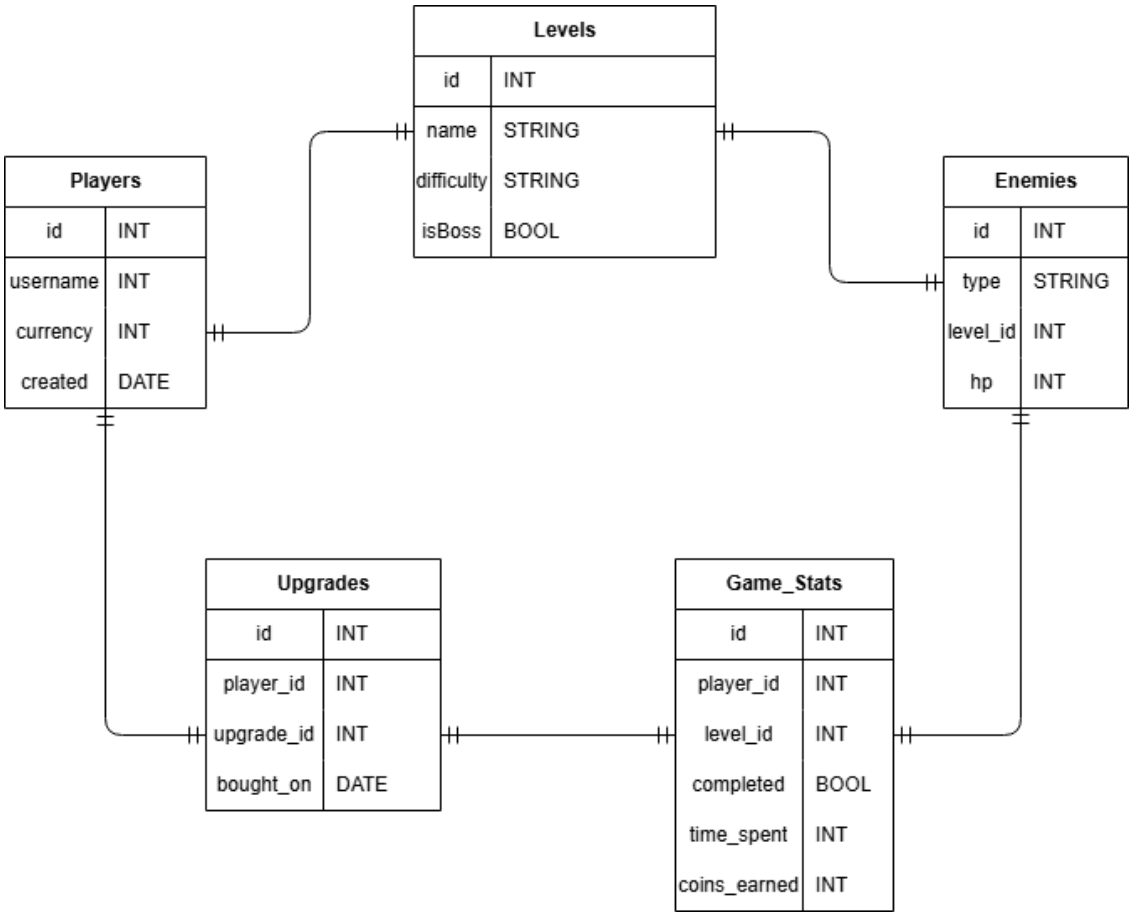

Рисунок 3.3 – ER-діаграма

ER-діаграма відображає структуру бази даних 2D-гри, яка складається з шести основних колекцій (таблиць), що містять ключову інформацію про гравців, рівні, ворогів, покращення та статистику проходження.

Перша колекція – Players (Гравці) – це колекція, документами якої є дані про кожного зареєстрованого гравця. У сутності “Гравець” наявні такі поля:

id (INT) – унікальний ідентифікатор гравця;

username (STRING) – ім’я користувача;

currency (INT) – поточна кількість внутрішньої валюти;

created (DATE) – дата створення акаунта.

Друга колекція – Levels (Рівні) – це набір рівнів, які проходить гравець під час гри. У сутності “Рівень” наявні наступні поля:

id (INT) – унікальний ідентифікатор рівня;

name (STRING) – назва рівня;

difficulty (STRING) – складність рівня;

isBoss (BOOL) – чи присутній бос на рівні.

Третя колекція – Enemies (Вороги) – містить інформацію про ворогів, присутніх на різних рівнях. У сутності “Ворог” наявні наступні поля:

id (INT) – унікальний ідентифікатор ворога;

type (STRING) – тип ворога (наприклад, ближній/дальній бій);

level_id (INT) – зв’язок із рівнем, на якому він розміщується;

hp (INT) – запас здоров’я ворога.

Четверта колекція – Upgrades (Покращення) – містить перелік усіх доступних покращень, які гравець може придбати за внутрішню валюту. У сутності “Покращення” наявні такі поля:

id (INT) – унікальний ідентифікатор покращення;

player_id (INT) – ідентифікатор гравця, якому належить покращення;

upgrade_id (INT) – ідентифікатор типу покращення;

bought_on (DATE) – дата придбання.

П’ята колекція – Game_Stats (Статистика проходження) – відображає результати проходження рівнів окремими гравцями. У сутності “Статистика” наявні такі поля:

id (INT) – унікальний ідентифікатор запису;

player_id (INT) – гравець, який пройшов рівень;

level_id (INT) – ID рівня;

completed (BOOL) – чи завершено рівень;

time_spent (INT) – час проходження (у секундах);

coins_earned (INT) – кількість валюти, зароблена на рівні.

Ця ER-діаграма забезпечує логічну організацію ігрових даних, що дозволяє ефективно зберігати та обробляти інформацію про прогрес гравця, структуру рівнів, дії ворогів та ігрову економіку.

3.2 Обґрунтування вибору мови програмування

Ігровий двигун – це центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну сторону, дозволяє полегшити розробку гри за рахунок уніфікації та систематизації її внутрішньої структури. Важливим значенням рушія є можливість створення багатоплатформових ігор. Різні студії використовують різні двигуни. На даний час одним із найбільш відомих та прогресивних рушіїв є Unreal Engine, який розробляється компанією Epic Games.

Unreal Engine був спочатку розроблений для створення шутерів від першої особи, але його функціональність з часом значно розширилась. Він підтримує більшість сучасних платформ, включаючи Windows, Linux, Mac OS, а також мобільні операційні системи. Починаючи з Unreal Engine 3, його почали адаптовувати й для мобільних пристроїв, зокрема iOS, webOS тощо.

Окрему увагу варто приділити четвертому поколінню рушія - Unreal Engine. Він відзначається інтуїтивно зрозумілим інтерфейсом, а також зручним візуальним програмуванням завдяки системі Blueprint, яка дозволяє будувати логіку без глибоких знань мов програмування. Це дає змогу команді розробників фокусуватись на дизайні та геймплеї. Особливістю рушія є потужна система освітлення, що дозволяє створювати реалістичні сцени, а також вдосконалений редактор частинок, який генерує велику кількість об’єктів (наприклад, пил), кожен з яких має індивідуальні характеристики.

Таким чином, рушій постійно оновлюється, адаптується під нові реалії ринку та є гнучким інструментом як для AAA-студій, так і для незалежних розробників. Його відкритий код, розширена документація та спільнота роблять Unreal Engine сильним конкурентом на ринку інструментальних засобів для розробки ігор [1].

Unity надає унікальні можливості, яких немає в інших платформах, однак варто звернути увагу і на його недоліки, особливо для новачків. Наприклад, на відміну від Unreal Engine, Unity не надає шаблонів проєктів -

все доведеться будувати з нуля. Безкоштовна версія доступна, однак платна підписка за $15 на місяць значно розширює інструментарій розробника.

Одним з поширених недоліків є використання пам’яті. Unity вимагає більше ресурсів під час розробки та запуску, що може створити труднощі для розробників на слабших комп’ютерах, особливо коли проєкт ускладнюється. Крім того, у Unity немає зовнішніх бібліотек коду, тому частіше доводиться писати власний функціонал. Платформа досі використовує бекенд Mono, який хоча і потужний, але має недоліки, зокрема повільне оновлення та підтримку [20].

Unity - це кросплатформовий рушій, який підтримує розробку 2D і 3D ігор. Його інтерфейс зручний для новачків, а величезний магазин активів дозволяє легко розширювати функціонал або додавати графіку. Це робить його популярним як серед початківців, так і серед досвідчених команд. Втім, платформа має і певні недоліки. Зокрема, можливі проблеми з продуктивністю в складних проєктах. Також графічні можливості Unity поступаються Unreal Engine, особливо у створенні фотореалістичної 3D-графіки [24].

Одним з викликів було забезпечити високу якість графіки на різних пристроях. Використання Universal Render Pipeline (URP) дало змогу оптимізувати та масштабувати візуальну частину. Для обробки складної поведінки, наприклад, руху зграй риб, команда застосувала систему C# Job System і компілятор Burst, що дозволило зберегти продуктивність [13].

InnerSloth використали Unity Gaming Services для підтримки та масштабування гри Among Us - однієї з найпопулярніших мобільних ігор. Після стрімкого успіху гри, Unity надала стабільну серверну інфраструктуру через Multiplay, що дозволило забезпечити безперебійний онлайн-досвід мільйонам гравців. Крім того, за допомогою Unity Analytics команда отримувала аналітичні дані для подальшого вдосконалення гри. Цей кейс демонструє потужність Unity не лише в графіці, а й у забезпеченні мережевої частини мобільних ігор [16].

Final Fantasy XV: Pocket Edition - це адаптація повноцінної AAA-гри для мобільних платформ. Гру повністю переробили з використанням Unity, аби зробити її доступною на смартфонах, зберігаючи сюжет і основні механіки. Метою було створити гру для ширшої аудиторії, спростивши інтерфейс та оптимізувавши візуальні ефекти. Це доводить, що Unity може бути ефективним навіть у великих проєктах, забезпечуючи при цьому чудову графіку та ігрову логіку на мобільних пристроях [14].

Ludo King- - індійська безкоштовна мобільна гра, розроблена на Unity. Вона доступна для Android, iOS, Windows Phone та ПК, і стала однією з найзавантажуваніших ігор у світі. Це перша індійська гра, яка перетнула межу в 1 мільярд завантажень. Успіх Ludo King доводить, що навіть прості 2D-ігри, створені на Unity, можуть досягти масової популярності завдяки стабільності, низьким вимогам до ресурсів і можливостям для оновлень та кастомізації [17].

Отже, аналіз інструментальних засобів для створення мобільних ігор свідчить про значну конкуренцію між рушіями, серед яких Unreal Engine та Unity посідають провідні позиції. Unreal Engine вирізняється високою якістю графіки, потужними засобами візуального програмування та глибокими можливостями налаштування. Він є оптимальним вибором для створення фотореалістичних 3D-проєктів, однак вимагає більше ресурсів та вищого рівня підготовки. У свою чергу, Unity забезпечує широку підтримку платформ, простий інтерфейс, активну спільноту та наявність готових інструментів і активів, що особливо корисно при створенні мобільних 2D ігор.

Попри певні технічні обмеження, таких як менші графічні можливості порівняно з Unreal Engine або відсутність вбудованих шаблонів, Unity довів свою ефективність у великій кількості успішних проєктів - від простих інді-ігор до адаптацій AAA-проєктів та хітів із масовим онлайном .Обрана мова програмування - C#, яка використовується у середовищі Unity, є сучасною, об'єктно-орієнтованою та добре підходить для розробки як невеликих, так і

масштабних мобільних ігор. Її перевагами є читабельність коду, висока швидкість компіляції та підтримка асинхронного програмування. Також C# дозволяє ефективно використовувати можливості рушія Unity, включаючи систему Job System і Burst Compiler, які критично важливі для оптимізації продуктивності на мобільних пристроях.Таким чином, вибір рушія Unity у поєднанні з мовою програмування C# є обґрунтованим рішенням для реалізації мобільної 2D гри, з огляду на технічні можливості, доступність, гнучкість та підтверджену ефективність у реальних кейсах.

РОЗДІЛ 4. ДОСЛІДНА ЕКСПЛУАТАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1. Функціональні можливості та інструкція користувача інформаційної системи

Для початку гри необхідно завантажити виконуваний файл гри (APK), після чого на екрані мобільного пристрою з’являється головне вікно гри (рис. 3.2).

Рисунок 3.2 – Головне меню гри

У головному меню користувач може отримати доступ до всіх функціональних можливостей гри, а саме:

– кнопка «Start game» – дозволяє розпочати нову гру, завантаживши перший рівень;

– кнопка «Options» – відкриває вікно налаштувань гри, де можна змінити рівень гучності звукових ефектів, музики;

– кнопка «Exit» – вихід з гри.

Для того, щоб керувати персонажем у грі, необхідно виконати наступні дії:

Активізувати кнопку «Start game», після чого відкривається перший ігровий рівень (рис. 3.3).

Рисунок 3.3 – Початок рівня

На екрані з’являються елементи управління:

– кнопки «←» та «→» - для переміщення персонажа відповідно вліво або вправо;

– кнопка «↑» - для виконання стрибка;

– кнопка «Атака» - для атаки ворогів;

– кнопка «Магазин» - відкриває меню магазину з можливістю купувати покращення.

У верхній частині екрана відображається інформаційна панель, яка містить такі елементи:

– назва поточного рівня (наприклад, «Level 1») - у лівому верхньому куті;

– кількість життів гравця (наприклад, «Lives: 3») - посередині вгорі;

– кількість зібраних монет (наприклад, «Cherry: 0») - у правому верхньому куті;

– кнопка магазину – у правому верхньому куті під «Cherry»

Для взаємодії з ігровими елементами необхідно:

1. Збирати cherry які розташовані на рівні. Cherry використовуються як внутрішньоігрова валюта для придбання покращень.
2. Уникати або знищувати ворогів. При зіткненні з ворогами персонаж втрачає одне життя. Для знищення ворогів необхідно застосовувати атаку.
3. Досягти кінцевої точки рівня, уникаючи перешкод, ворогів і пасток.

Для відкриття магазину необхідно натиснути кнопку «Магазин» у правому верхньому куті екрана. Після цього гра автоматично переходить у режим паузи, і на екрані з’являється вікно магазину (рис. 3.4).

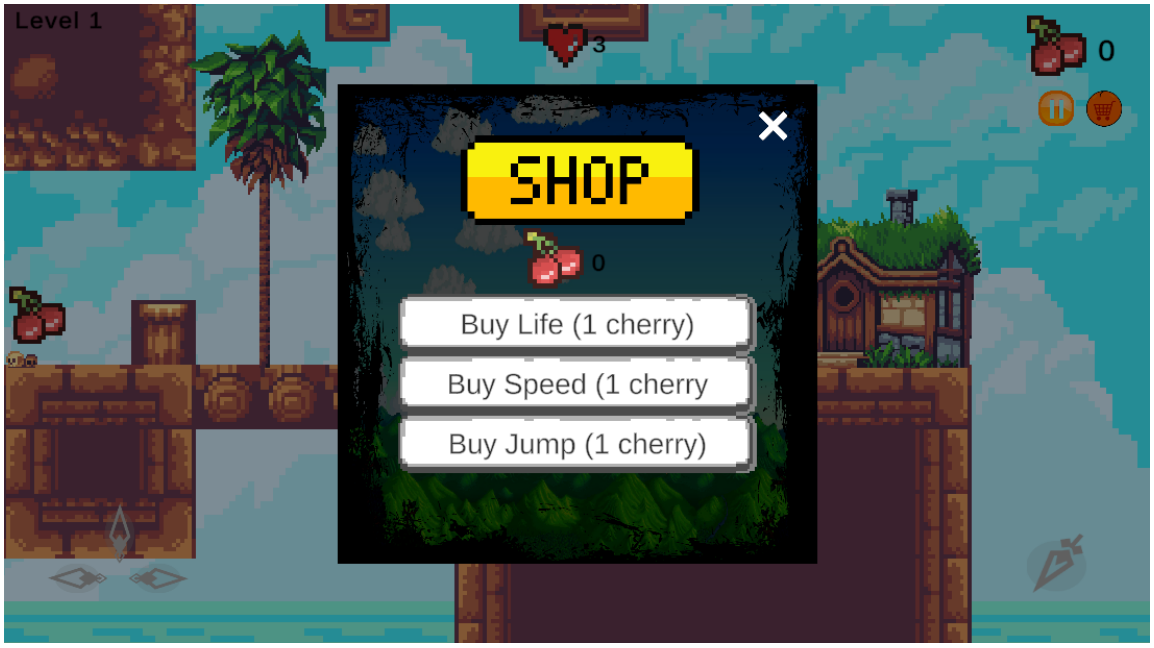

Рисунок 3.4 – Вікно магазину у грі

У вікні магазину доступні наступні можливості:

–придбати додаткові життя;

–придбати покращення стрибка (збільшує висоту стрибка персонажа);

–придбати збільшення швидкості бігу персонажа.

Покупки здійснюються за зібрані cherry. Після покупки, відповідне покращення активується автоматично.

У грі реалізован ворог, який патрулює певну ділянку рівня (рис. 3.5).

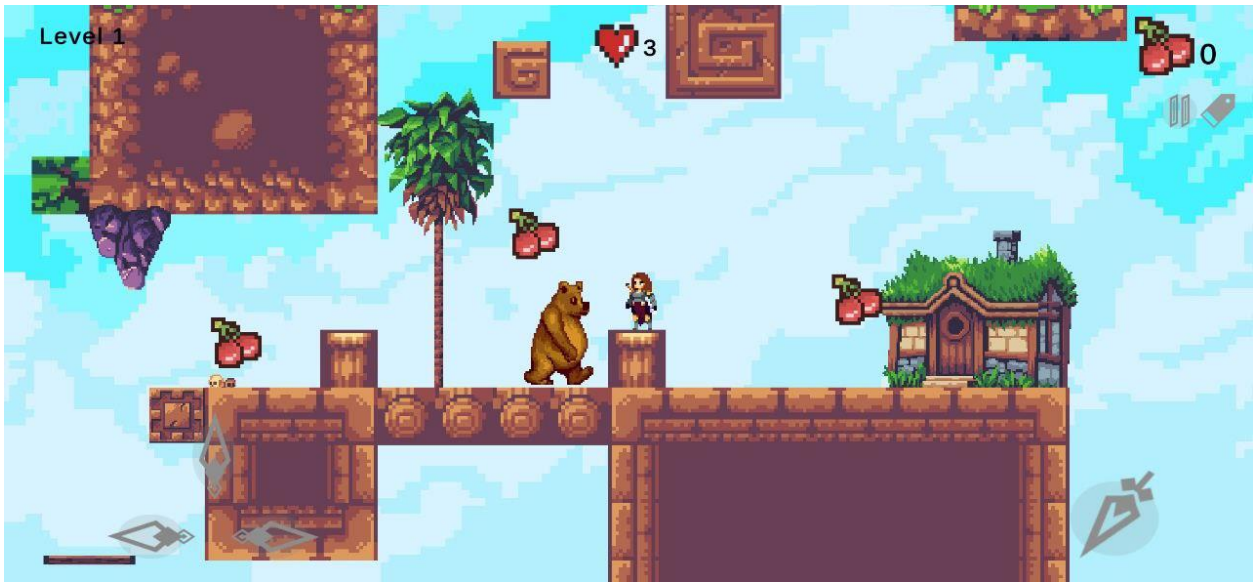

Рисунок 3.5 – Ворог у грі

Основні можливості взаємодії з ворогом:

– при зіткненні персонажа з ворогом зменшується кількість життів на 1

– ворог можуть бути знищені атакою персонажа;

– уникання ворога також можливе завдяки маневрам та стрибкам.

Рівень вважається пройденим, якщо гравець досягає кінцевої точки, не втративши всі життя. У разі втрати всіх життів гра завершується, і гравцю пропонується розпочати рівень спочатку або повернутися до головного меню.

Отже, функціональні можливості мобільної гри дозволяють забезпечити захоплюючий геймплей із використанням елементів фізики, системи збору ресурсів, прокачування персонажа та взаємодії з противниками. Гра адаптована під мобільні пристрої та може бути використана як для розваги, так і для розвитку навичок реакції та стратегічного мислення.

4.2. Тестування мобільної гри

Після завершення основного етапу розробки гри було проведено її тестування з метою перевірки працездатності, коректності відображення

графічних елементів, адаптивності інтерфейсу та стабільності роботи на мобільному пристрої.

Для тестування було застосовано метод Compatibility Testing - перевірка сумісності гри з різними розширеннями екрана мобільних пристроїв. Це тестування було виконано безпосередньо в середовищі Unity. У вікні Game було обрано різні типи розширень екрана, зокрема популярні дозволи екранів мобільних телефонів. У результаті перевірки було встановлено, що всі елементи інтерфейсу гри відображаються коректно, елементи керування та інформаційна панель не перекриваються незалежно від розширення екрана (рис. 4.1).

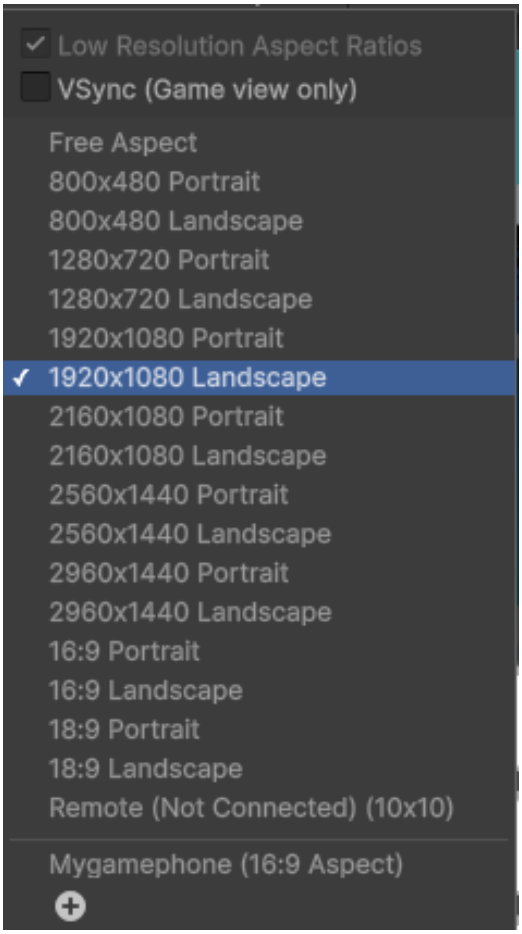

Рисунок 4.1 – Перевірка адаптивності інтерфейсу у Unity

Для проведення фінального тестування гра була встановлена на мобільний пристрій під керуванням Android у вигляді APK-файлу. Після запуску гри було перевірено стабільність її роботи - протягом гри не було

зафіксовано жодних збоїв, зависань чи аварійного завершення процесу (рис. 4.2).

Рисунок 4.2 – Тестування стабільності гри на мобільному пристрої

Наступним етапом стало тестування коректності відображення графічних елементів на екрані мобільного пристрою. Всі спрайти персонажа, ворогів, предметів та фону відображаються належним чином, відсутні графічні артефакти чи розриви (рис. 4.3).

Рисунок 4.3 – Відображення графічних елементів гри

Окремо було протестовано ігрову динаміку: переміщення персонажа, виконання стрибків, атака ворогів, взаємодія з об'єктами на рівні. Особливу увагу приділено тестуванню елементів керування - сенсорні кнопки вірно реагують на натискання, затримка при управлінні відсутня. Була перевірена коректність роботи магазину, можливість купівлі покращень, а також правильне відображення інформаційної панелі (рис. 4.4).

Рисунок 4.4 – Тестування ігрової динаміки та взаємодії з магазином

На завершальному етапі було проведено тестування ігрової механіки: збирання cherry як внутрішньо-ігрової валюти, використання цих ресурсів для придбання покращень, взаємодія з ворогами. Всі ігрові сценарії виконувались відповідно до задуму. Оскільки механізм збереження прогресу у грі не реалізовано, після виходу з гри накопичені ресурси скидаються. Це відповідає специфікації даної версії гри.

У результаті тестування встановлено, що розроблена мобільна гра працює стабільно, коректно адаптована під різні розміри екранів, графічні елементи відображаються якісно, а основні функції реалізовано згідно з поставленими завданнями.

4.3 Оформлення техніко-економічних показників мобільної гри

Для об'єктивної оцінки ефективності впровадження інформаційної системи необхідно розрахувати основні техніко-економічні показники, які характеризують ефективність впровадження продукту.

4.3.1 Витарати, повязані з розробкою продукту.

До даних витрат належать витрати на оплату праці розробників з нарахуваннями, амортизаційні відрахування, витрати на матеріали, електроенергію та інші супутні витрати.

Витрати на оплату праці з нарахуваннями – складаються з витрат на основну заробітну плату, додаткову заробітну плату та нарахувань згідно чинного законодавства (22%).

Витрати на основну заробітну плату – визначаються виходячи з місячного посадового окладу або вартості 1 години роботи розробника та витрат часу, понесених на розробку даного продукту.

Розрахунок основної оплати праці здійснюється за формулою:

$$ЗПосн = \frac{ЗПміс}{ФРЧ} \cdot Тр; \quad (4.1)$$

де ЗПосн – основна заробітна плата , грн;

ЗПміс – місячний посадовий оклад розробника, на підприємстві становить 23021 грн/міс;

ФРЧ – місячний фонд робочого часу, становить 176 (22 дні по 8 год) год./міс;

Тр – трудомісткість (затрати часу) виготовлення програмного продукту, год.

Трудомісткість (затрати часу) розробника на створення продукту визначається виходячи з кількості витрачених на розробку робочих днів та тривалості робочого дня:

$$ФРЧ = Крдм \cdot Трз; \quad (4.2)$$

де Крд – кількість робочих днів на створення продукту, приймаємо 22 дн.

Трз – тривалість робочого дня (зміни), год.

Отже,

$$Тр = 22 \cdot 8 = 172год; \quad (4.3)$$

Таким чином:

$$ЗПосн = \frac{23021}{176} \cdot 104 = 13603{,}3 \text{ грн.};$$

У випадку, якщо при розробці продукту задіяні декілька працівників, розмір основної заробітної плати розраховується за кожним.

Додаткова заробітна плата – передбачає різноманітні види доплат за рівень кваліфікації, стаж роботи тощо. Розмір нарахувань здійснюється відповідно до норм прийнятих на підприємстві-замовника. Відсоток нарахувань додаткової заробітної плати коливається в межах 10-25%. На даному підприємстві додаткова оплата праці становить 21%.

Сума додаткової заробітної плати розраховується від розміру основної:

$$ЗПдод = ЗПосн \cdot \frac{\%дод.}{100}. \quad (4.3)$$

де %дод – відсоток нарахувань додаткової заробітної плати, %

$$ЗПдод = 13603{,}3 \cdot \frac{21}{100} = 2856{,}7грн.$$

Нарахування на заробітну плату здійснюють від суми основної та додаткової зарплати. Розмір нарахувань становить 22%.

$$Нзп = (ЗПосн + ЗПдод) \cdot \frac{\%нар}{100} \quad (4.4)$$

де %нар – відсоток нарахувань заробітної плати, %

$$Нзп = (13603{,}3 + 2856{,}7) \cdot \frac{22}{100} = 3621{,}2грн.$$

Розрахунок загальної суми витрат на оплату праці з нарахуваннями розробника наведемо у вигляді таблиці 2.

Таблиця 2 – Розрахунок витрат на оплату праці з нарахуваннями розробників програмного продукту

Найменування посади	Основна заробітна плата, грн.	Додаткова заробітна плата, год.	Розмір нарахувань на оплату праці, грн.	Всього витрат на оплату праці з нарахуваннями, грн.
Інженер-програміст	13603,3	2856,7	3621,2	20081,2

Розрахунок суми амортизаційних відрахувань обладнання, яке використовується при розробці продукту здійснюється найбільш поширеним прямолінійним методом. При створенні продукту використовувався ноутбук та принтер. Відповідно до податкового кодексу України електронно-обчислювальні машини (ПК) відносять до 4 групи, з терміном корисного використання 2 роки. Відповідно річна норма амортизації становить 50%. Балансова вартість ПК (або ноутбуку) на момент придбання (розрахунку) становить 16000 грн, принтера - початкова вартість 11000 грн, приймаємо річну норму амортизації в розмірі 20% (термін використання 5 років).

Розрахунок річної суми амортизаційних відрахувань:

$$\text{Аріч} = \text{БВобл} \cdot \frac{\%\text{аморт.}}{100}; \quad (4.5)$$

де БВобл. - ціна придбання, або балансова вартість комп'ютера на момент придбання, грн.

%аморт – річний відсоток нарахувань додаткової заробітної плати, %

$$Аріч.комп. = 16000 \cdot \frac{50}{100} = 8000грн.$$

$$Аріч.принт. = 11000 \cdot \frac{20}{100} = 2200грн.$$

Сума амортизаційних відрахувань, що увійде до собівартості програмного продукту розраховується с врахуванням часу використання ноутбуку та принтера при розробці продукту.

$$А = Аріч \cdot \frac{Твик(днів)}{365} \quad (4.6)$$

де Твик. - термін використання персонального комп'ютера та принтера при розробці програмного продукту, міс. або дні.

Таким чином, загальна сума амортизаційних відрахувань, становитиме:

$$A = (8000 + 2200) \cdot \frac{13днів}{365} = 363{,}3грн.$$

Витрати на матеріали – визначаються відповідно до кількості витрачених матеріалів на розробку продукту та ціни на даний матеріал, на момент розрахунку. Витрати на матеріали, що були використані на розробку продукту наведені в таблиці 3.

Таблиця 3 – Розрахунок матеріальних витрат на створення продукту

Найменування матеріалу	Ціна за одиницю матеріалу, грн./од	Витрачено, од	Вартість витрачених матеріалів, грн.
Чорнила, банка	320	5	1600
Папір, уп.	200	3	600
Разом витрат			2200

Витрати на електроенергію – з потужності обладнання, що використовується при розробці продукту (П=0,42), коефіцієнту використання

потужності (Квп=0,95), фактичного часу (трудомісткості) на розробку та ціни електроенергії, що склалась на момент розрахунку (Ц1кВт=4,32 грн.):

$$\text{Вел.} = \text{П} \cdot \text{Квп} \cdot \text{Тр} \cdot \text{Ц1кВт} = 0{,}42 \cdot 0{,}95 \cdot 176 \cdot 4{,}32 = 303{,}4 \text{ грн.}; \quad (4.7)$$

Розрахунок суми витрат на розробку відображені даними таблиці 4.

Таблиця 4 – Розрахунок витрат на створення продукту

Найменування витрат	Вартість, грн.
Витарти на оплату праці з нарахуваннями, грн.	20081,2
Амортизаційні відрахування, грн	363,3
Витрати на матеріали, грн	2200
Витрати на електроенергію, грн	303,4
Разом витрат	17214,5

4.3.2. Витрати на розміщення продукту.

Для публічного розміщення мобільної 2D гри також необхідно врахувати вартість хостингу та домену. Річна вартість хостингового обслуговування становить орієнтовно 250 грн/ міс, або 3000 грн/рік), а річна вартість доменного імені — приблизно 500 грн.

Таким чином, загальні витрати на розміщення інформаційної системи наведено 3750 грн.

4.3.3. Розрахунок собівартості одиниці копії програмного продукту

Сукупні витрати на створення та розміщення програмного продукту складаються з загальної суми витрат:

$$Соб.прод = Вроз. + Врозм. + Врек + Вінш. \quad (4.8)$$

де Врозр. – витрати на розробку програмного продукту, грн;

Врозм.с. - витрати, пов’язані з розміщенням інформації на сервері та вартість хостингу сайту, грн;

Врек – витрати на рекламу, приймаємо на рівні - 3000 грн.

Вінш – інші витрати, які не включені до вищеперелічених витрат, коливаються в межах 10-30% , приймаємо на рівні 22%.

$$Вінш. = (17214{,}5 + 3750 + 3000) \cdot \frac{22}{100} = 5272{,}2 грн.$$

Таким чином, витрати на створення продукту складатиме:

$$Вствор. = 17214{,}5 + 3750 + 3000 + 5272{,}2 = 29236{,}7 \text{грн}.$$

Оскільки, програмний продукт приймається та впроваджується за завданням замовника, а також може реалізовуватись іншим покупцям відповідно доцільно визначити ціну продажу. Ціна реалізації залежить від кількості реалізованого продукту. В нашому випадку плануємо реалізацію 5 од продукту. Таким чином, витрати на створення продукту розподіляються на кількість запланованих продажів та становлять собівартість одиниці продукту для продажу. Таким чином, собівартість одиниці продукту становить:

$$Спрод. = \frac{29236{,}7}{5} = 5847{,}3 \frac{грн}{од}.$$

4.3.4.Розрахунок ціни реалізації проектного рішення

Ціна реалізації визначається виходячи з рівня запланованої прибутковості (рентабельності) задля задоволення цільової аудиторії та прибутковості замовника. Нами було проведено аналіз ринкової вартості данного продукту, відповідно до результатів, рівень рентабельності за згодою замовника може коливається в межах 40-80%. Приймаємо рівень прибутковості 50%, рівень податку на додану вартість – згідно чинного законодавства становить 20%.

Таким чином, ціна реалізації продукту розраховується за формулою:

$$Цр = Спрод \cdot (1 + \frac{Р}{100}) \cdot (1 + \frac{ПДВ}{100}) \quad (4.9)$$

де Р– прийнятий норматив рентабельності, 60%.

ПДВ – ставка податку на додану вартість, приймається на рівні 20%.

$$\text{Цр} = 5847{,}3 \cdot \left(1 + \frac{50}{100}\right) \cdot \left(1 + \frac{20}{100}\right) = 10525{,}2$$

4.3.5.Розрахунок прибутку від створення та продажу програмного продукту.

Розрахунок чистого прибутку від реалізації запланованого обсягу програмного продукту здійснюється з врахуванням річного прибутку від реалізації та податку на прибуток, що існує на момент розрахунку. При розрахунку прибутку ціна реалізації приймається без ПДВ та становить ціну реалізації власника:

$$\text{Цр власн.} = 10525{,}2 \cdot \left(1 + \frac{50}{100}\right) = 15787{,}8\text{грн.}$$

Прибуток від продажу розраховується за формулою:

$$\text{П} = (\text{Цр власн.} - \text{Спрод}) \cdot \text{Опрод} \quad (4.10)$$

де Опрод. – кількість одиниць продажу програмного продукту, од;

Таким чином загальна сума прибутку від продажу становить:

$$\text{П} = (15787{,}8 - 5847{,}3) \cdot 5 = 49702{,}5\text{грн.}$$

Відповідно чистий прибуток за відрахуванням податку на прибуток становитиме:

$$\text{ЧП} = \text{П} - \left(\text{П} \cdot \frac{\%\text{ПП}}{100}\right) \quad (4.11)$$

де %ПП – відсоток податку на прибуток, приймається на рівні 18%.

$$ЧП = 49702{,}5 - \left(49702{,}5 \cdot \frac{18}{100}\right) = 40756{,}1 грн.$$

4.3.6. Термін окупності.

Термін окупності характеризує період часу протягом якого окупляться витрати на розробку програмного продукту замовником.

Розрахунок періоду окупності здійснюється за формулою:

$$Ток = \frac{Вствор.}{ЧП}; \quad (4.12)$$

$$Ток = \frac{29236{,}7}{40756{,}1} = 0{,}72\ року.$$

Таким чином, створення та подальший продаж розробленого програмного продукту є достатньо ефективним для замовника. Витрати на створення продукту становлять майже 29,236 тис. грн. При плануванні обсягу подальшого продажу на рівні 5 од., розмір отриманого прибутку для замовника становить 40,756 тис. грн., а період окупності при даних показниках – 0,72 року. Слід зауважити, що показники ефективності розраховувались за рівнем прибутковості нижчим ринкового показника, також при розрахунках використовували мінімальну кількість проданого продукту, відповідно на перспективу продаж даного продукту є достатньо ефективним.

4.4 Розрахунок освітлення в приміщенні

Розробка мобільної 2D гри під Android із застосуванням фізичного рушія Unity передбачає тривалу роботу за комп’ютером. Тому для забезпечення комфортних умов праці необхідно правильно організувати освітлення робочого простору.

Коротка характеристика приміщення і виконуваних робіт: у приміщенні операторів є два комп'ютери. Комп'ютери об'єднані в локальну мережу. До складу основного обладнання входять персональні ПК та друкуючий пристрій типу Canon S300. У приміщенні є один вхід. Обладнання розміщується таким чином, щоб був забезпечений вільний прохід до всіх робочих місцях. Робота в основному пов'язана з ПК.

Планування і розміщення робочих місць: відповідно до загальних ергономічних вимог за ГОСТ 12.2.049-80 виробниче обладнання повинно відповідати антропометричним, фізіологічним, психофізіологічним, властивостям людини та обумовленим цими властивостями гігієнічними вимогами з метою збереження здоров'я людини і досягнення збільшення ефективності праці, зниження стомлюваності.

Столи робочих місць, на якому виробляються діагностика, має висоту 800 мм, довжину 1400 мм, ширину 800 мм, висота сидіння 450 мм, висота простору для ніг – 650 мм, що забезпечує зручність робочого місця. Висота і конструкція робочого столу вибрані так, щоб було легко переходити з робочого положення сидячи в положення стоячи.

Розрахунок освітлення: Приміщення має розміри 3 м на 4 м. Таким чином, площа складе S = 12 м^2. При висоті H = 3 м обсяг приміщення складе V = 36 м^3. Природне освітлення – одностороннє. Приміщення має одне вікно висотою 2 м, і шириною 3 м, розташованих на одній стіні. Згідно БНІП П-4-79, коефіцієнт природної освітленості (КПО) у цьому випадку при бічному висвітленні повинен складати $e_н$=1,5%.

Примітка: *Коефіцієнт природної освітленості (КПО)* - це процентне відношення природної освітленості у будь-якій точці в середині приміщення до одночасно виміряної на томуж рівні освітленості зовнішньої горизонтальної площини рівномірно розсіяним (дифузійним) світлом усього небосхилу

Враховуючи коефіцієнт світлового клімату m = 0,8 і коефіцієнт сонячності клімату при тому, що вікно приміщення виходе на східний бік, С = 0,7, визначаємо нормоване значення КПО для даного приміщення:

$e = e_н \cdot m \cdot C = 1{,}5 \cdot 0{,}5 \cdot 0{,}7 = 0{,}84\%$.

При бічному освітленні КПО можна оцінити за формулою:

$$e = \frac{\tau_0 \cdot r_1 \cdot S_0}{K_з \cdot \eta_0 \cdot K_{зд} \cdot S_п} \cdot 100\% \; ; \qquad (4.1)$$

де $S_п$ – площа підлоги приміщення, м^2;

S_0 – площа світлових прорізів, м^2;

$K_з$ – коефіцієнт запасу (приймається в межах від 1,2 до 2,0 залежно від можливого забруднення світлових прорізів кіптявою);

η_0 – світлова характеристика вікон;

$K_{зд}$ – коефіцієнт, що враховує підвищення коефіцієнта завдяки світлу, відбитому від поверхні приміщення та підстилаючих шару, який прилягає до будівлі;

τ_0 – загальний коефіцієнт світлопропускання, що визначається за формулою:

$$\tau_0 = \tau_1 + \tau_2 + \tau_3 + \tau_4 ; \qquad (4.2)$$

де τ_1 – коефіцієнт світлопропускання матеріалу (для різних типів скла приймається в межах від 0,65 до 0,9);

τ_2 – коефіцієнт, що враховує втрати світла в межах вікна (в залежності від виду палітурки приймається в межах від 0,5 до 0,9);

τ_3 – коефіцієнт, що враховує втрати світла в несучих конструкціях (від 0,8 до 0,9);

τ_4 – коефіцієнт, що враховує втрати світла в сонцезахисних пристроях (при їх відсутності $\tau_4 = 1$, при їх наявності тоді приймається від 0,6 до 0,9).

Площа світлових прорізів становить S0=6м^2. Площа підлоги приміщення становить Sп=12 м^2. Для складальних цехів коефіцієнт запасу приймається Кз=1.2.

Виходячи зі співвідношення розмірів приміщення і вікон, світлова характеристика вікон $\eta_0 = 8,5$. Коефіцієнт, що враховує КПО за рахунок відбиття від поверхонь приміщення і підстилаючого шару, для даного приміщення становить r1 = 1,3.

Для вікон з подвійними рамами коефіцієнт світлопропускання $\tau_1 = 0,85$. Оскільки палітурка вікна подвійна металопластикова, то $\tau_2 = 0,7$. При бічному освітленні $\tau_3 = 0,9$. Як сонцезахисних пристроїв використовуються регульовані внутрішні штори, тому $\tau_4 = 0,8$.

Таким чином, отримуємо:

$$\tau 1 = 0,85 \cdot 0,7 \cdot 0,9 \cdot 0,8 = 0,4284 \text{ ;}$$

$$e = \frac{0,4284 \cdot 1,3 \cdot 6}{1,2 \cdot 8,5 \cdot 1,3 \cdot 12} \cdot 100\% = 2,1\% \text{ .}$$

Тож коефіцієнт природного освітлення е = 2,1% задовольняє нормі. Природне освітлення істотно залежить від погодних умов, отже, необхідно передбачити штучне освітлення в похмуру погоду. Штучне освітлення необхідно ще й тому, що в приміщенні ведуться роботи не тільки у світлий час доби, але і в темний.

ВИСНОВКИ

Мобільні ігри є однією з найпопулярніших форм розваг у світі. Особливою популярністю користуються 2D-ігри, які поєднують простоту, динамічний геймплей і легкий доступ для користувачів. Використання фізичного рушія Unity дозволяє створювати більш реалістичні та цікаві ігрові механіки, що робить тему розробки актуальною в умовах стрімкого розвитку ігрової індустрії.

У процесі дослідження було проаналізовано сучасні технології та інструменти для створення мобільних 2D ігор. Unity підтвердив свою ефективність завдяки потужному інструментарію, підтримці фізики, кросплатформеності та широкій базі готових рішень. Це дозволило визначити оптимальний набір технологій для реалізації поставленої мети.

Окремо було досліджено особливості застосування фізичного рушія Unity. Завдяки використанню фізичних компонентів, таких як Rigidbody2D і Collider2D, вдалося створити механіку зіткнень та взаємодій, що забезпечує реалістичність рухів і поведінки об’єктів у грі. Це суттєво підвищило якість геймплею й загальну привабливість проєкту для кінцевого користувача.

Для якісної реалізації гри було підготовлено технічне завдання, яке чітко визначило функціональні можливості гри, структуру проєкту, дизайн і ключові вимоги до стабільності роботи. Це стало основою для організованої та послідовної розробки програмного продукту.

У результаті було створено повноцінну мобільну 2D гру під Android із використанням мови програмування C# та середовища Unity. Реалізація охоплювала програмування ігрової логіки, розробку графічних елементів, налаштування інтерфейсу та інтеграцію фізичних властивостей об’єктів.

На завершальному етапі було проведено тестування гри на мобільних пристроях. У процесі тестування здійснено оптимізацію продуктивності, зокрема через корекцію графічних налаштувань, оптимізацію використання

пам’яті та обробки фізики. Це дозволило досягти стабільної роботи застосунку навіть на пристроях із середніми технічними характеристиками.

Отже, у ході виконання дипломної роботи було повністю досягнуто поставленої мети — створено мобільну 2D гру для Android з використанням фізичного рушія Unity, яка демонструє практичне застосування технологій і може слугувати основою для подальших навчальних або комерційних проєктів у сфері ігрової розробки.

СПИСОК ЛІТЕРАТУРИ

1. Бородіна О.О. Майбутнє ігрової індустрії. Unreal Engine / О.О. Бородіна, А.М. Гафіяк, О.Р. Білобров, С.Д. Просвєтов // *Математичне та імітаційне моделювання систем. МОДС 2019 : тези доповідей Чотирнадцятої міжнар. наук.-практ. конф*. (Чернігів, 24–26 черв. 2019 р.). Чернігів : ЧНТУ, 2019. – С. 254–256.
2. Дорошенко Є. М. Дослідження ринку сучасної ігрової індустрії. *Вісник Кам'янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки*. 2022. Випуск 15. С. 40–47.
3. Зінов'єва О. Г. Використання CASE-засобів для проектування інформаційних систем. Українські студії в європейському контексті. 2023. № 7. С. 220–227.
4. Ліпська О., Свіженко О. Мобільні інформаційні технології. *Інформаційні технології у науці, освіті, виробництві: збірник тез І Всеукраїнської науково-практичної Інтернет-конференції здобувачів вищої освіти і молодих учених* (26 квітня 2018 р., м. Маріуполь). Маріуполь: МДУ, 2018. С. 78–80.
5. Медведєва М.О., Жмурко О.І. Мобільні технології в освітньому процесі : навчальний посібник. Умань : Візаві, 2019. 105 с.
6. Садчикова І., Тарасенко А. Дубина М. Теоретичне обґрунтування сутності поняття «Електронна комерція». *Економіка та суспільство*, 2023. № 53. С. 1–9.
7. Сушко Д. В., Герасимов Т. Ю. Ігрова індустрія як спосіб маніпулювання людьми. *Матеріали XLVIII науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2019)* : збірник доповідей. Вінниця : ВНТУ, 2019. С. 11–12.
8. Хмарні та мобільні технології в освіті: навч.-метод. посіб. / уклад. М. О. Медведєва. Умань : Візаві, 2021. 122 с.

9. Шаров С. В., Скрипка С. О. Використання мови UML для інфологічного моделювання реляційної бази даних. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення*. 2014. № 7. С. 5–9
10. Шаров С. В., Шамардак О. А. Концептуальне моделювання інформаційної системи за допомогою мови UML. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення*. 2014. № 7. С. 10–14.
11. Шведа Н. М., Краузе О. І. Електронна комерція: сучасний стан та стратегії розвитку. *Міжнародний науковий журнал «Інтернаука». Серія: Економічні науки*. 2024. Випуск 2 (82). С. 35–42.
12. Automatoys // Wikipedia. URL: https://en.wikipedia.org/wiki/Automatoys.
13. Dave the Diver. Unity. URL: https://unity.com/case-study/dave-diver
14. Final Fantasy XV: Pocket Edition. Wikipedia. URL: https://en.wikipedia.org/wiki/Final_Fantasy_XV%3A_Pocket_Edition
15. IDEF0. Wikipedia. URL: https://en.wikipedia.org/wiki/IDEF0
16. Innersloth – Among Us. Unity. URL: https://unity.com/case-study/innersloth-among-us
17. Ludo King. Wikipedia. URL: https://en.wikipedia.org/wiki/Ludo_King
18. Mini Motor Racing // Wikipedia. URL: https://en.wikipedia.org/wiki/Mini_Motor_Racing.
19. Poly Bridge // Wikipedia. URL: https://en.wikipedia.org/wiki/Poly_Bridge_(video_game).
20. Pros and cons of Unity game engine. Pingle Studio. URL: https://pinglestudio.com/blog/full-cycle-development/pros-and-cons-of-unity-game-engine
21. Size Does Matter // Wikipedia. URL: https://en.wikipedia.org/wiki/Size_Does_Matter.

22. The Complete Guide To Understand IDEF Diagram. EdrawMax. URL: https://www.edrawmax.com/article/the-complete-guide-to-understand-idef-diagram.html
23. UML Use Case Diagram Tutorial. Lucidchart. URL: https://www.lucidchart.com/pages/uml-use-case-diagram
24. Unity pros and cons. Vention. URL: https://ventionteams.com/blog/unity-pros-and-cons
25. What is Use Case Diagram in UML? Visual Paradigm. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
26. World of Goo // Wikipedia. URL: https://en.wikipedia.org/wiki/World_of_Goo).

Додаток А

Програмний код мобільної гри

Скрипт PlayerController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[RequireComponent(typeof(Rigidbody2D))]
public class PlayerController : MonoBehaviour
{
    [SerializeField] private float moveSpeed = 5f;
    [SerializeField] private float jumpPower = 10f;
    [SerializeField] private float attackRange = 1f;
    [SerializeField] private Transform attackPoint;
    [SerializeField] private LayerMask enemyLayers;
    [SerializeField] private LayerMask groundLayer;
    [SerializeField] private Transform groundCheck;
    [SerializeField] private float groundCheckRadius = 0.2f;
    [Header("Jump Settings")]
    [SerializeField] private float coyoteTime = 0.1f;
    private float coyoteTimeCounter;
    private Rigidbody2D rb;
    private Animator animator;
    private bool facingRight = true;
    private bool isAttacking = false;
    private float runSoundTimer = 0f;
    private float runSoundInterval = 0.3f;
    public void IncreaseSpeed(float amount) => moveSpeed += amount;
    public void IncreaseJump(float amount) => jumpPower += amount;
    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
    }
    private void Update()
    {
        if (GameManager.Instance != null && GameManager.Instance.IsGamePaused())
            return;
        UpdateCoyoteTime();
        HandleMovement();
        HandleJump();
        HandleAttack();
        UpdateAnimationStates();
    }
    private void UpdateCoyoteTime()
```

```
    {
        if (IsGrounded())
            coyoteTimeCounter = coyoteTime; // Reset coyote timer when grounded
        else
            coyoteTimeCounter -= Time.deltaTime; // Decrease timer when in air
    }
    private void HandleMovement()
    {
        if (isAttacking && animator.GetCurrentAnimatorStateInfo(0).IsName("Attack"))
            return;
        float moveInput = 0f;
        if (MobileInputManager.Instance != null)
        {
            if (MobileInputManager.Instance.moveLeft) moveInput = -1f;
            if (MobileInputManager.Instance.moveRight) moveInput = 1f;
        }
        else
        {
            moveInput = Input.GetAxisRaw("Horizontal");
        }
        rb.linearVelocity = new Vector2(moveInput * moveSpeed, rb.linearVelocity.y);
        animator.SetFloat("speed", Mathf.Abs(moveInput));
        // Play running SFX
        if (Mathf.Abs(moveInput) > 0.1f && coyoteTimeCounter > 0)
        {
            runSoundTimer -= Time.deltaTime;
            if (runSoundTimer <= 0f)
            {
                AudioManager.Instance?.PlayRun();
                runSoundTimer = runSoundInterval;
            }
        }
        if (moveInput > 0 && !facingRight) Flip();
        else if (moveInput < 0 && facingRight) Flip();
    }
    private void HandleJump()
    {
        bool jumpPressed = false;
        if (MobileInputManager.Instance != null &&
MobileInputManager.Instance.jumpPressed)
        {
            jumpPressed = true;
            MobileInputManager.Instance.jumpPressed = false;
        }
        if (Input.GetKeyDown(KeyCode.Space))
```

```
        {
            jumpPressed = true;
        }

        if (jumpPressed && coyoteTimeCounter > 0) // allow jump during coyote time
window
        {
            if (isAttacking) EndAttack();
            rb.AddForce(Vector2.up * jumpPower, ForceMode2D.Impulse);
            AudioManager.Instance?.PlayJump(); // Play jump SFX
            coyoteTimeCounter = 0; // reset coyote timer to prevent multiple jumps in air
        }
    }
    private void HandleAttack()
    {
        bool attackPressed = false;
        if (MobileInputManager.Instance != null &&
MobileInputManager.Instance.attackPressed)
        {
            attackPressed = true;
            MobileInputManager.Instance.attackPressed = false;
        }
        if (Input.GetKeyDown(KeyCode.J))
        {
            attackPressed = true;
        }
        if (attackPressed && !isAttacking && coyoteTimeCounter > 0) // only allow attack
when grounded or in coyote time
        {
            isAttacking = true;
            animator.SetBool("isAttacking", true);
            animator.SetTrigger("Attack");
            AudioManager.Instance?.PlayAttack(); // Play attack SFX
        }
    }
    // Called by Animation Event at attack hit frame
    public void PerformAttack()
    {
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position,
attackRange, enemyLayers);
        foreach (Collider2D enemy in hitEnemies)
        {
            Destroy(enemy.gameObject);
        }
    }
```

```
    // Called by Animation Event at the end of attack animation
    public void EndAttack()
    {
        isAttacking = false;
        animator.SetBool("isAttacking", false);
    }
    private void UpdateAnimationStates()
    {
        float yVelocity = rb.linearVelocity.y;
        animator.SetBool("isJumping", yVelocity > 0.5f && !IsGrounded());
        animator.SetBool("isFalling", yVelocity < -0.5f && !IsGrounded());
        animator.SetBool("isGrounded", coyoteTimeCounter > 0);
    }
    private void Flip()
    {
        facingRight = !facingRight;
        Vector3 scale = transform.localScale;
        scale.x *= -1;
        transform.localScale = scale;
    }

    private bool IsGrounded()
    {
        return Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius,
groundLayer);
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Coin"))
        {
            GameManager.Instance.AddCoin();
            AudioManager.Instance?.PlayUIClick();
            Destroy(other.gameObject);
        }
        else if (other.CompareTag("Enemy"))
        {
            GameManager.Instance.LoseLife();
            AudioManager.Instance?.PlayDamage();
            Debug.Log("Lost life! Remaining lives: " + GameManager.Instance.lives);
        }
    }
    private void OnDrawGizmosSelected()
    {
        if (attackPoint != null)
```

```
        {
            Gizmos.color = Color.red;
            Gizmos.DrawWireSphere(attackPoint.position, attackRange);
        }
        if (groundCheck != null)
        {
            Gizmos.color = Color.green;
            Gizmos.DrawWireSphere(groundCheck.position, groundCheckRadius);
        }
    }
}
```

Скрипт MobileInputManager

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
public class MobileInputManager : MonoBehaviour
{
    public static MobileInputManager Instance;
    [Header("Buttons")]
    public Button jumpButton;
    public Button attackButton;
    [Header("Movement Buttons (Objects with EventTrigger)")]
    public GameObject leftButtonObj;
    public GameObject rightButtonObj;
    [HideInInspector] public bool moveLeft;
    [HideInInspector] public bool moveRight;
    [HideInInspector] public bool jumpPressed;
    [HideInInspector] public bool attackPressed;
    private void Awake()
    {
        if (Instance == null) Instance = this;
        else Destroy(gameObject);
    }
    private void Start()
    {
        jumpButton.onClick.AddListener(PressJump);
        attackButton.onClick.AddListener(PressAttack);
    }
    public void PressJump()
    {
        jumpPressed = true;
    }
    public void PressAttack()
    {
        attackPressed = true;
```

```
    }
    public void StartMoveLeft()
    {
        moveLeft = true;
    }
    public void StopMoveLeft()
    {
        moveLeft = false;
    }
    public void StartMoveRight()
    {
        moveRight = true;
    }
    public void StopMoveRight()
    {
        moveRight = false;
    }
}
```

Скрипт ShopManager

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
public class ShopManager : MonoBehaviour
{
    [Header("UI Elements")]
    [SerializeField] private GameObject shopPanel;
    [SerializeField] private TMP_Text coinsText;
    [SerializeField] private Button buyLifeButton;
    [SerializeField] private Button buySpeedButton;
    [SerializeField] private Button buyJumpButton;
    [Header("Prices")]
    [SerializeField] private int lifePrice = 5;
    [SerializeField] private int speedPrice = 7;
    [SerializeField] private int jumpPrice = 7;
    [Header("Upgrades")]
    [SerializeField] private float speedBoost = 2f;
    [SerializeField] private float jumpBoost = 5f;
    private PlayerController player;
    private bool speedUpgraded = false;
    private bool jumpUpgraded = false;
    private void Start()
    {
        // Find player
        player = FindFirstObjectByType<PlayerController>();
```

```
        buyLifeButton.onClick.AddListener(() =>
        {
            BuyLife();
            AudioManager.Instance?.PlayUIClick(); // Play UI click on button press
        });
        buySpeedButton.onClick.AddListener(() =>
        {
            BuySpeed();
            AudioManager.Instance?.PlayUIClick();
        });
        buyJumpButton.onClick.AddListener(() =>
        {
            BuyJump();
            AudioManager.Instance?.PlayUIClick();
        });
        // Hide shop
        shopPanel.SetActive(false);
        // Coins display
        UpdateCoinUI();
    }
    /// <summary>
    /// Called by UI button to open/close the shop.
    /// </summary>
    public void ToggleShop()
    {
        bool isActive = !shopPanel.activeSelf;
        shopPanel.SetActive(isActive);
        if (isActive)
        {
            Time.timeScale = 0f;
            player.enabled = false;
            UpdateCoinUI();
        }
        else
        {
            Time.timeScale = 1f;
            player.enabled = true;
            Input.ResetInputAxes();
        }
        AudioManager.Instance?.PlayUIClick(); // Play UI click when toggling shop
    }
    /// <summary>
    /// Updates coin display in shop.
    /// </summary>
    private void UpdateCoinUI()
```

```
{
    coinsText.text = GameManager.Instance.coins.ToString();
}
/// <summary>
/// Buy life if the player have enough coins.
/// </summary>
private void BuyLife()
{
    if (GameManager.Instance.TrySpendCoins(lifePrice))
    {
        GameManager.Instance.AddLife();
        UpdateCoinUI();
    }
}
/// <summary>
/// Buy speed boost once per level.
/// </summary>
private void BuySpeed()
{
    if (speedUpgraded) return;
    if (GameManager.Instance.TrySpendCoins(speedPrice))
    {
        player.IncreaseSpeed(speedBoost);
        speedUpgraded = true;
        UpdateCoinUI();
    }
}
/// <summary>
/// Buy jump boost once per level.
/// </summary>
private void BuyJump()
{
    if (jumpUpgraded) return;
    if (GameManager.Instance.TrySpendCoins(jumpPrice))
    {
        player.IncreaseJump(jumpBoost);
        jumpUpgraded = true;
        UpdateCoinUI();
    }
}
/// <summary>
/// Reset upgrades when restarting level or after death.
/// </summary>
public void ResetUpgrades()
{
```

```
        speedUpgraded = false;
        jumpUpgraded = false;
    }
}
```

Скрипт MainMenuManager

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class MainMenuManager : MonoBehaviour
{
    [Header("Panels")]
    [SerializeField] private GameObject mainMenuPanel;
    [SerializeField] private GameObject optionsPanel;
    [Header("Audio Settings")]
    [SerializeField] private Slider musicVolumeSlider;
    [SerializeField] private Slider sfxVolumeSlider;
    private void Start()
    {
        if (optionsPanel != null)
            optionsPanel.SetActive(false);
        // Load saved values or use default
        float savedMusicVol = PlayerPrefs.GetFloat("MusicVolume", 0.1f);
        float savedSFXVol = PlayerPrefs.GetFloat("SFXVolume", 0.3f);
        if (musicVolumeSlider != null)
        {
            musicVolumeSlider.value = savedMusicVol;
musicVolumeSlider.onValueChanged.AddListener(SetMusicVolume);
        }
        if (sfxVolumeSlider != null)
        {
            sfxVolumeSlider.value = savedSFXVol;
sfxVolumeSlider.onValueChanged.AddListener(SetSFXVolume);
        }
        if (AudioManager.Instance != null)
        {
            AudioManager.Instance.SetMusicVolume(savedMusicVol);
            AudioManager.Instance.SetSFXVolume(savedSFXVol);
        }
    }
    public void OpenOptions()
    {
        if (optionsPanel != null)
            optionsPanel.SetActive(true);
    }
```

```
    public void CloseOptions()
    {
        if (optionsPanel != null)
            optionsPanel.SetActive(false);
    }
    public void SetMusicVolume(float value)
    {
        if (AudioManager.Instance != null)
        {
            AudioManager.Instance.SetMusicVolume(value);
            PlayerPrefs.SetFloat("MusicVolume", value);
            PlayerPrefs.Save();
        }
    }
    public void SetSFXVolume(float value)
    {
        if (AudioManager.Instance != null)
        {
            AudioManager.Instance.SetSFXVolume(value);
            PlayerPrefs.SetFloat("SFXVolume", value);
            PlayerPrefs.Save();
        }
    }
    public void StartGame()
    {
        SceneManager.LoadScene("Level1");
    }
    public void ExitGame()
    {
        Application.Quit();
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
#endif
    }
}
```

Скрипт FinishTrigger

```
using UnityEngine;
public class FinishTrigger : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            GameManager.Instance.WinGame();
        }
```

```
    }
}
```

Скрипт DeathZone

```
using UnityEngine;
public class DeathZone : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {

            GameManager.Instance.lives = 0;
            GameManager.Instance.LoseLife();
        }
    }
}
```

Скрипт AudioManager

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class AudioManager : MonoBehaviour
{
    public static AudioManager Instance;
    [Header("Audio Sources")]
    public AudioSource musicSource;
    public AudioSource sfxSource;
    [Header("Music Clips")]
    public AudioClip menuMusic;
    public AudioClip levelMusic;
    [Header("Sound Effects")]
    public AudioClip jumpSound;
    public AudioClip attackSound;
    public AudioClip runSound;
    public AudioClip damageSound;
    public AudioClip deathSound;
    public AudioClip winSound;
    public AudioClip uiClickSound;
    public AudioClip pauseSound;
    public AudioClip coinSound;
    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
```

```
            SceneManager.sceneLoaded += OnSceneLoaded;
            LoadVolumeSettings(); // Load volume settings immediately
        }
        else
        {
            Destroy(gameObject);
        }
    }
    private void Start()
    {
        PlayMusicForScene(SceneManager.GetActiveScene().name);
    }
    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        PlayMusicForScene(scene.name);
        LoadVolumeSettings(); // Ensure volume is updated when changing scenes
    }
    private void PlayMusicForScene(string sceneName)
    {
        if (sceneName == "MainMenu")
        {
            PlayMusic(menuMusic);
        }
        else
        {
            PlayMusic(levelMusic);
        }
    }
    public void PlayMusic(AudioClip clip)
    {
        if (musicSource == null || clip == null) return;

        if (musicSource.clip == clip && musicSource.isPlaying) return;
        musicSource.clip = clip;
        musicSource.loop = true;
        musicSource.Play();
    }
    public void StopMusic()
    {
        if (musicSource != null)
            musicSource.Stop();
    }
    private void PlaySFX(AudioClip clip)
    {
        if (sfxSource == null || clip == null) return;
```

```
        sfxSource.PlayOneShot(clip);
    }
    // SFX Shortcut Methods
    public void PlayJump() => PlaySFX(jumpSound);
    public void PlayAttack() => PlaySFX(attackSound);
    public void PlayRun() => PlaySFX(runSound);
    public void PlayDamage() => PlaySFX(damageSound);
    public void PlayDeath() => PlaySFX(deathSound);
    public void PlayWin() => PlaySFX(winSound);
    public void PlayUIClick() => PlaySFX(uiClickSound);
    public void PlayPauseSound() => PlaySFX(pauseSound);
    public void PlayCoin() => PlaySFX(coinSound);
    public void SetMusicVolume(float volume)
    {
        if (musicSource != null)
            musicSource.volume = volume;

        PlayerPrefs.SetFloat("MusicVolume", volume);
        PlayerPrefs.Save();
    }
    public void SetSFXVolume(float volume)
    {
        if (sfxSource != null)
            sfxSource.volume = volume;

        PlayerPrefs.SetFloat("SFXVolume", volume);
        PlayerPrefs.Save();
    }
    private void LoadVolumeSettings()
    {
        float musicVol = PlayerPrefs.GetFloat("MusicVolume", 0.5f);
        float sfxVol = PlayerPrefs.GetFloat("SFXVolume", 0.5f);

        if (musicSource != null)
            musicSource.volume = musicVol;

        if (sfxSource != null)
            sfxSource.volume = sfxVol;
    }
}
```

Скрипт EnemyPatrols

```
using UnityEngine;
public class EnemyPatrol : MonoBehaviour
{
```

```
    public float speed = 2f;
    public Transform pointA;
    public Transform pointB;
    private Vector3 target;
    void Start()
    {
        target = pointB.position;
    }
    void Update()
    {
        transform.position = Vector3.MoveTowards(transform.position, target, speed *
Time.deltaTime);
        if (Vector3.Distance(transform.position, target) < 0.1f)
        {
            target = target == pointA.position ? pointB.position : pointA.position;
            Flip();
        }
    }
    void Flip()
    {
        Vector3 scale = transform.localScale;
        scale.x *= -1;
        transform.localScale = scale;
    }
}
```

Скрипт PauseMenuManager

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class PauseMenuManager : MonoBehaviour
{
    [Header("UI Elements")]
    [SerializeField] private GameObject pauseMenu;
    private PlayerController player;
    private bool isPaused = false;
    public void ForceClosePauseMenu()
    {
        isPaused = false;
        if (pauseMenu != null) pauseMenu.SetActive(false);
    }
    private void Start()
    {
        // Find PlayerController script
        player = FindFirstObjectByType<PlayerController>();
```

```
    if (pauseMenu != null)
        pauseMenu.SetActive(false);
}
/// <summary>
/// Called by Pause Button in UI
/// </summary>
public void TogglePause()
{
    isPaused = !isPaused;
    if (isPaused)
    {
        Time.timeScale = 0f;
        pauseMenu.SetActive(true);
        // Disable player movement during pause
        if (player != null) player.enabled = false;
        // Reset input axes
        Input.ResetInputAxes();
        // Reset mobile input
        if (MobileInputManager.Instance != null)
        {
            MobileInputManager.Instance.moveLeft = false;
            MobileInputManager.Instance.moveRight = false;
            MobileInputManager.Instance.jumpPressed = false;
            MobileInputManager.Instance.attackPressed = false;
        }
        AudioManager.Instance?.PlayPauseSound(); // Play pause sound
    }
    else
    {
        pauseMenu.SetActive(false);
        Time.timeScale = 1f;
        // Re-enable player movement
        if (player != null) player.enabled = true;
        // Reset input axes
        Input.ResetInputAxes();
    }
}
public void ResumeGame()
{
    isPaused = false;
    pauseMenu.SetActive(false);
    Time.timeScale = 1f;
    if (player != null) player.enabled = true;
    Input.ResetInputAxes();
    if (MobileInputManager.Instance != null)
```

```
        {
            MobileInputManager.Instance.moveLeft = false;
            MobileInputManager.Instance.moveRight = false;
            MobileInputManager.Instance.jumpPressed = false;
            MobileInputManager.Instance.attackPressed = false;
        }
    }
    public void RestartLevel()
    {
        Time.timeScale = 1f;
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
    public void LoadMainMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");  // Scene name MUST match!
    }
}
```

Скрипт CameraFollow

```
using UnityEngine;
public class CameraFollow : MonoBehaviour
{
    public Transform target; // Player
    public Vector3 offset = new Vector3(0f, 0f, -10f);
    [Header("Level Boundaries")]
    public float minX, maxX, minY, maxY; // Camera movement limits
    void LateUpdate()
    {
        if (target != null)
        {
            Vector3 desiredPosition = target.position + offset;
            // Camera pos to level border
            float clampedX = Mathf.Clamp(desiredPosition.x, minX, maxX);
            float clampedY = Mathf.Clamp(desiredPosition.y, minY, maxY);
            transform.position = new Vector3(clampedX, clampedY, desiredPosition.z);
        }
    }
}
```

Скрипт GameManager

```
using UnityEngine;
using TMPro;
using UnityEngine.SceneManagement;
public class GameManager : MonoBehaviour
```

```
{
    public static GameManager Instance;
    public int coins = 0;
    public int lives = 3;
    public int levelNumber = 1;
    public TextMeshProUGUI coinsText;
    public TextMeshProUGUI livesText;
    public TextMeshProUGUI levelText;
    [Header("Menus")]
    public GameObject gameOverMenu;
    public GameObject shopMenu;
    public GameObject winMenu;
    private bool isPaused = false;
    private PlayerController player;
    private PauseMenuManager pauseMenuManager;
    public void WinGame()
    {
        if (player != null)
            player.enabled = false;
        Time.timeScale = 0f;
        isPaused = true;
        if (winMenu != null)
            winMenu.SetActive(true);
        AudioManager.Instance?.PlayWin(); // Play win sound
    }
    private void Awake()
    {
        if (Instance == null)
            Instance = this;
        else
            Destroy(gameObject);
    }
    private void Start()
    {
        player = FindFirstObjectByType<PlayerController>();
        pauseMenuManager =
FindFirstObjectByType<PauseMenuManager>();
        UpdateUI();
```

```
        if (gameOverMenu != null) gameOverMenu.SetActive(false);
        if (shopMenu != null) shopMenu.SetActive(false);
    }
    public void AddCoin()
    {
        coins++;
        UpdateUI();
    }
    public void AddLife()
    {
        lives++;
        UpdateUI();
    }
    public void LoseLife()
    {
        lives--;
        UpdateUI();
        AudioManager.Instance?.PlayDamage(); // Play damage sound on life
lost
        if (lives <= 0)
        {
            GameOver();
        }
    }
    private void GameOver()
    {
        if (pauseMenuManager != null)
            pauseMenuManager.ForceClosePauseMenu();
        if (gameOverMenu != null)
            gameOverMenu.SetActive(true);
        Time.timeScale = 0f;
        isPaused = true;
        AudioManager.Instance?.PlayDeath(); // Play death sound on game
over
        // Disable player movement
        if (player != null)
            player.enabled = false;
        Input.ResetInputAxes();
```

```
        if (MobileInputManager.Instance != null)
        {
            MobileInputManager.Instance.moveLeft = false;
            MobileInputManager.Instance.moveRight = false;
            MobileInputManager.Instance.jumpPressed = false;
            MobileInputManager.Instance.attackPressed = false;
        }
    }
    public void OpenShop()
    {
        PauseGame();
        if (shopMenu != null)
            shopMenu.SetActive(true);
        AudioManager.Instance?.PlayUIClick(); // Play UI click when shop
opens
    }
    public void CloseShop()
    {
        if (shopMenu != null)
            shopMenu.SetActive(false);
        ResumeGame();
        AudioManager.Instance?.PlayUIClick(); // Play UI click when shop
closes
    }
    private void PauseGame()
    {
        Time.timeScale = 0f;
        isPaused = true;
    }
    private void ResumeGame()
    {
        Time.timeScale = 1f;
        isPaused = false;
    }
    public bool IsGamePaused()
    {
        return isPaused;
    }
```

```
    public void RestartLevel()
    {
        ResumeGame();
SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
    public void LoadMainMenu()
    {
        ResumeGame();
        SceneManager.LoadScene("MainMenu");
    }
    public bool TrySpendCoins(int amount)
    {
        if (coins >= amount)
        {
            coins -= amount;
            UpdateUI();
            return true;
        }
        else
        {
            return false;
        }
    }
    private void UpdateUI()
    {
        coinsText.text = coins.ToString();
        livesText.text = lives.ToString();
        levelText.text = "Level " + levelNumber;
    }
}
```