

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ЗАКЛАД
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА
ШЕВЧЕНКА”

Навчально-науковий інститут математики та інформаційних технологій


Кафедра математики та інформатики

Крутько Олександр Олександрович

**ОСОБЛИВОСТІ ЗАСТОСУВАННЯ C# ДЛЯ РОЗРОБКИ
МОБІЛЬНИХ ДОДАТКІВ**

**кваліфікаційна робота
здобувача вищої освіти першого (бакалаврського) рівня
освітньої програми « Комп’ютерні науки та інформаційні
технології » за спеціальністю 122 Комп’ютерні науки**

Особистий підпис  Олександр КРУТЬКО

Науковий керівник  Ольга СМАГІНА
К. пед. н., доцент
кафедри математики
та інформатики

В.о. завідувача
кафедри _____ Ліна БОНДАРЕНКО
К. пед. н., доцент
кафедри математики
та інформатики

Міністерство освіти і науки України
Державний заклад
„Луганський національний університет імені Тараса Шевченка”
Факультет (інститут) Навчально-науковий інститут математики та
інформаційних технологій
Кафедра Кафедра математики та інформатики
Освітній ступень Бакалавр
Напрямок підготовки (спеціальність) 122 Комп’ютерні науки
(код, назва)
Галузь знань 12, Інформаційні технології
(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри МІ

_____ (підпис) (ініціали, прізвище)

“ ” _____ 2024 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ
Крутьку Олександру Олександровичу

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) Особливості застосування C# для розробки мобільних додатків

Керівник кваліфікаційної роботи Смагіна О.О., К. пед. н., доцент кафедри математики та інформатики

(прізвище, ініціали, науковий ступінь, вчене звання)

затверджена наказом по університету від “ ” 202 року №

2. Строк подання студентом проекту (роботи)

3. Вихідні дані до роботи Ігровий додаток “Angry Lambda”, створений для платформи Android.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) визначення та особливості мови програмування C#, використання платформи Android.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання „_____” _____ 202_р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 1 березня	
2	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	До 20 березня	
3	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	До 1 квітня	
4	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	До 15 квітня	
5	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	До 30 квітня	
6	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	До 30 травня	
7	Попередній захист роботи на кафедрі	травень	
8	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державної атестації	
9	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	За 5 днів до державної атестації	

Здобувач вищої освіти

підпис

О.О.Крутько

(ініціали, прізвище)

Керівник проекту (роботи)

О.О. Смагіна

АНОТАЦІЯ

Крутько О.О.

Тема: Особливості застосування C# для розробки мобільних додатків.

Спеціальність: 122 „Комп’ютерні науки”

Установа: ДЗ ЛНУ імені Т.Шевченка, 202_р.

Кваліфікаційна робота містить: 52 с., 1 додат., 3 рис., 23 джерела.

Об’єкт дослідження – ігровий додаток “Angry Lambda”.

Предмет дослідження – розробка ігрового додатку для платформи Android із застосуванням мови програмування C# та рушія Unity.

Мета роботи - розробка ігрового додатку “Angry Lambda” для платформи Android.

Результати роботи. Розроблений ігровий додаток “Angry Lambda” для платформи Android.

Висновки. Обґрунтовано вибір інструментів та технологій, практична реалізація гри з налаштуванням графіки, звуку та оптимізацією продуктивності

Ключові слова. “ANGRY LAMBDA”, 2D, БАГАТОПОТОЧНІСТЬ, КРОСПЛАТФОРМНІСТЬ, РУШІЙ, МЕХАНІКА, СКРИПТ, ОПТИМІЗАЦІЯ.

ABSTRACT

Krutko O.O.

Theme: Features of using C# for mobile application development.

Specialty: 122 "Computer Science"

Institution: DZ of T. Shevchenko LNU, 202

Qualification work contains: 52 pages, 1 appendix, 23 sources.

Object of the research: game application "Angry Lambda".

Subject of research: development of a game application for the Android platform using the C# programming language and the Unity engine.

Purpose of the study: develop the game application "Angry Lambda" for the Android platform.

Results of research. Developed game application "Angry Lambda" for the Android platform.

Conclusions. Reasonable choice of tools and technologies, practical implementation of the game with graphics, sound and performance optimization

Keywords. "ANGRY LAMBDA", 2D, MULTI-THREADING, CROSS-PLATFORM, ENGINE, MECHANICS, SCRIPT, OPTIMIZATION.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІГРОВОГО ДОДАТКУ НА МОВІ C#.....	9
1.1. Огляд мов програмування для розробки мобільних додатків та особливості застосування C#	9
1.1.1. Огляд основних мов програмування для Android (Java, C++, C#)	9
1.1.2. Порівняння нативної розробки та кросплатформних рішень (Xamarin, Unity, .NET MAUI)	13
1.1.3. Введення до розробки на Android	15
1.2. Аналіз та вибір оптимального рушія з підтримкою C#	19
1.2.1. Огляд популярних рушіїв Android (Unity, Unreal Engine, Cocos2d). Приклади ігор на них	19
1.2.2. Переваги та недоліки кожного рушія	20
1.2.3. Вибір оптимального рушія для гри.....	23
1.3. Аналіз існуючих ігрових додатків для Android	24
1.3.1. Огляд популярних ігрових додатків на Android.....	24
1.3.2. Аналіз функціональності та особливостей	26
1.3.3. Аналіз ринку: цільова аудиторія, популярні жанри, бізнес- моделі	28
1.4. Особливості проектування гри	30
1.4.1. Концепція Half-Life 2	30
1.4.2. Концепція Angry Birds	31
1.4.3. Концепція гри проєкту: жанр, сюжет, механіка	32
1.5. Висновки до розділу 1	34
РОЗДІЛ 2. ЕТАПИ РОЗРОБКИ ДОДАТКУ	36
2.1. Встановлення середовища розробки та налаштування проєкту	36

2.1.1. Встановлення необхідних інструментів (Visual Studio, Unity, Adobe Photoshop).....	36
2.1.2. Налаштування проекту Unity на Android (2D гра)	37
2.2. Реалізація головної логіки гри.....	38
2.2.1. Розробка ігрової механіки.....	38
2.2.2. Реалізація основних функцій та взаємодій у грі.....	43
2.2.3. Інтеграція графіки та звуку	46
2.3. Створення інтерфейсу користувача	47
2.3.1. Інтерфейс та його види.....	47
2.3.2. Основи дизайну інтерфейсу.....	48
2.4. Оптимізація та тестування	54
2.5. Висновки до розділу 2	56
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	61
ДОДАТОК.....	63

ВСТУП

У сучасному світі мобільні додатки займають значне місце в повсякденному житті людей. Вони стали невід'ємною частиною багатьох аспектів нашого існування, забезпечуючи зручний доступ до інформації, розваг, комунікацій та послуг. Особливу увагу заслуговує розробка ігрових додатків для мобільних платформ, зокрема для операційної системи Android, яка є однією з найпопулярніших у світі. Розробка ігор для Android вимагає не лише знань у галузі програмування, але й розуміння специфіки мобільних пристроїв, їх обмежень та можливостей. Одним з потужних інструментів для розробки мобільних додатків є мова програмування C#, що дозволяє створювати високоякісні кросплатформні додатки з використанням єдиного коду.

У цій роботі досліджено особливості застосування C# для розробки мобільних додатків, зокрема для платформи Android. Проведено аналіз існуючих ігрових додатків, розглянуто основні технології та інструменти розробки, а також надано практичні рекомендації щодо створення та оптимізації мобільних додатків на базі C#.

Метою даної роботи є дослідження можливостей та переваг використання мови C# для розробки мобільних додатків на платформі Android. Завданнями роботи є аналіз існуючих рішень, вибір оптимальних інструментів та технологій, проектування та реалізація ігрового додатку, а також оптимізація та тестування готового продукту.

Актуальність теми обумовлена зростаючою популярністю мобільних ігрових додатків та потребою у високоякісному програмному забезпеченні, яке може бути розроблене ефективно та швидко. Ми детально опишемо етапи розробки, тестування продукту

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІГРОВОГО ДОДАТКУ НА МОВІ C#

1.1. Огляд мов програмування для розробки мобільних додатків та особливості застосування C#

1.1.1. Огляд основних мов програмування для Android (Java, C++, C#)

Розробка додатків для Android може здійснюватися за допомогою кількох основних мов програмування. Кожна з них має свої особливості та переваги, що дозволяє розробникам вибирати оптимальний інструмент залежно від конкретних потреб проекту. У цьому розділі ми розглянемо Java, C++ та C# як основні мови програмування для Android.

Java є однією з основних мов програмування для розробки додатків на Android. Вона була обрана Google як офіційна мова для Android при запуску платформи у 2008 році. Java є об'єктно-орієнтованою мовою програмування, що забезпечує високу продуктивність та надійність додатків.

Переваги:

- Велика спільнота та підтримка: Завдяки тривалому існуванню та широкому використанню Java має велику спільноту розробників та багатий набір ресурсів, включаючи документацію, бібліотеки та фреймворки.
- Міжплатформеність: Додатки, написані на Java, можуть бути легко портовані на інші платформи.
- Модульність та повторне використання коду: Java дозволяє створювати модульний код, який легко підтримувати та повторно використовувати.

Особливості:

- Гарячі оновлення: Завдяки інструментам, таким як Android Studio, розробники можуть вносити зміни в код і негайно бачити результати.

- Інтеграція з Android SDK: Java тісно інтегрована з Android SDK, що надає доступ до всіх можливостей платформи Android.
- Підтримка багатонитковості: Java має вбудовану підтримку багатонитковості, що дозволяє створювати високопродуктивні додатки з паралельним виконанням завдань [1].

C++ використовується для розробки додатків на Android, особливо коли необхідна висока продуктивність та ефективність використання ресурсів. Ця мова дозволяє писати частини коду з низьким рівнем абстракції, що важливо для графічно насичених ігор та обробки відео [2].

Переваги:

- Висока продуктивність: C++ забезпечує високий рівень продуктивності завдяки низькорівневому доступу до пам'яті та процесора.
- Контроль над ресурсами: Розробники мають повний контроль над управлінням пам'яттю та іншими ресурсами, що дозволяє створювати оптимізовані додатки.
- Міжплатформеність: C++ дозволяє створювати кросплатформені додатки, які можуть працювати на різних операційних системах.

Особливості:

- NDK (Native Development Kit): Android NDK дозволяє розробникам писати частини додатків на C++ для забезпечення високої продуктивності.
- Інтеграція з Java: C++ код може бути інтегрований з Java кодом через JNI (Java Native Interface), що дозволяє використовувати переваги обох мов.
- Високий рівень складності: Розробка на C++ вимагає високого рівня знань та досвіду через складність мови та управління пам'яттю [3].

C# є мовою програмування, розробленою компанією Microsoft, яка стає все більш популярною для розробки додатків на Android завдяки використанню

платформи Xamarin та .NET MAUI. C# є об'єктно-орієнтованою мовою з сучасним синтаксисом та великою кількістю бібліотек.

Розробка високопродуктивних додатків вимагає вибору потужних інструментів та мов програмування, що забезпечують ефективність, надійність та продуктивність. Саме тому, для розробки додатку ми використали C#. Основні переваги використання у розробці високопродуктивних додатків були вирішальними при виборі мови програмування [4].

Простота і зручність розробки:

- C# має простий та інтуїтивно зрозумілий синтаксис, що робить його відносно простим в освоєнні для початківців-розробників. Він також має безліч вбудованих функцій і бібліотек, які можуть допомогти розробникам прискорити процес розробки.
- C# є типобезпечною мовою, тобто компілятор перевіряє типи даних під час компіляції програми. Це допомагає уникнути помилок, які можуть призвести до зниження продуктивності.
- C# підтримує об'єктно-орієнтоване програмування, що дає змогу розробникам створювати складні додатки, які легко масштабуються та підтримуються. Це може підвищити продуктивність, особливо для додатків, які обробляють великий обсяг даних.
- C# підтримує багатопоточність, тобто можна виконувати кілька завдань одночасно. Це підвищує продуктивність, особливо для додатків, які обробляють великий обсяг даних.

Інтеграція з різними платформами та технологіями:

- C# підтримує широкий спектр платформ, включно з Windows, macOS, Linux, iOS і Android. Це дозволяє розробникам створювати кросплатформні додатки з мінімальними зусиллями.

- C# може бути інтегрований з різними технологіями, такими як бази даних, веб-сервіси та пристрої. Завдяки інтеграції з хмарними технологіями, такими як Azure і AWS, розробники можуть підвищити продуктивність своїх додатків, переклавши частину навантаження на хмарну інфраструктуру. Інтеграція з Visual Studio є потужною IDE для розробки на C#, яка надає всі необхідні інструменти для створення, тестування та налагодження додатків.
 - C# може бути інтегрований з різними базами даних, такими як SQL Server і MySQL, що забезпечує гнучкість у виборі сховища даних для додатків.
 - C# має багату екосистему та велику спільноту розробників, що забезпечує доступ до великої кількості бібліотек, інструментів та ресурсів. Це сприяє швидкому вирішенню проблем і обміну знаннями серед розробників.
- Ключові особливості C# для створення високопродуктивних додатків
- C# має потужну і сувору систему типів, яка допомагає запобігти безлічі помилок уже на етапі компіляції. Це гарантує створення більш надійних і продуктивних додатків. У системі типів C# також підтримуються поліморфізм, успадкування та інтерфейси, що сприяє гнучкості та розширюваності коду.
 - C# має механізми оптимізації, що дозволяють створювати високоефективний код. Це включає автоматичне керування пам'яттю (збирання сміття) та оптимізацію виконання коду для досягнення найкращої продуктивності.
 - Розробка високопродуктивних додатків часто вимагає ефективної роботи з багатозадачністю та асинхронними операціями. C# надає багаті засоби для роботи з потоками виконання та асинхронними завданнями.
 - Допомагає розпаралелювати виконання завдань і ефективно використовувати багатоядерні процесори. C# надає засоби для створення

та керування потоками, а також синхронізацію доступу до загальних ресурсів.

- Дозволяє виконувати неблокувальні операції, такі як робота з мережею або файлами, без блокування основного потоку виконання. Це сприяє більш чуйному призначеному для користувача інтерфейсу і загальній продуктивності програми.

Також, C# широко використовується в Unity, одному з найпопулярніших рушіїв для розробки ігор. Це надає розробникам додаткову перевагу, оскільки Unity дозволяє створювати як 2D, так і 3D ігри, що працюють на різних платформах [5].

1.1.2 Порівняння нативної розробки та кросплатформних рішень (Xamarin, Unity, .NET MAUI)

Нативна розробка означає створення додатків спеціально для певної платформи з використанням інструментів і мов програмування, що надаються розробниками цієї платформи. Для Android це зазвичай означає використання Android Studio та мов програмування Java або Kotlin [6].

Переваги нативної розробки:

- Висока продуктивність: Додатки мають доступ до всіх функцій апаратного забезпечення та операційної системи, що забезпечує максимальну ефективність.
- Повний доступ до платформи: Використання всіх останніх API та можливостей, що надаються Android SDK.
- Плавний інтерфейс користувача: Можливість створення інтерфейсів, які максимально відповідають рекомендаціям та стандартам платформи.

Недоліки нативної розробки:

- Час та вартість розробки: Розробка окремих додатків для кожної платформи (наприклад, Android та iOS) потребує більше часу та ресурсів.

- **Подвійна підтримка:** Необхідність підтримувати та оновлювати код для кожної платформи окремо.
- **Командні ресурси:** Потреба в спеціалістах для кожної платформи.

Кросплатформені рішення дозволяють розробникам створювати додатки, які можуть працювати на кількох платформах з використанням єдиного кодового базису. Це допомагає скоротити витрати часу та ресурсів на розробку та підтримку додатків для різних операційних систем, таких як Android та iOS.

Переваги Xamarin:

- **Спільний кодовий базис:** Можливість використовувати один код для Android, iOS та інших платформ.
- **Інтеграція з .NET:** Використання екосистеми .NET для створення додатків.
- **Доступ до нативних API:** Можливість використання всіх функцій платформи через Xamarin.Android та Xamarin.iOS.

Недоліки:

- **Великі розміри додатків:** Додатки на Xamarin можуть мати більший розмір порівняно з нативними через інтеграцію з .NET.
- **Продуктивність:** Може бути нижчою у порівнянні з нативними додатками, особливо для графічно насичених або ресурсомістких завдань.

Переваги .NET MAUI:

- **Спільний кодовий базис:** Один код для створення додатків на Android, iOS, Windows та macOS.
- **Сучасний фреймворк:** Наступник Xamarin.Forms, що надає більш зручні та потужні можливості для створення UI.
- **Інтеграція з .NET:** Використання потужної екосистеми .NET для розробки.

Недоліки:

- Відносна новизна: .NET MAUI є відносно новим фреймворком, що може призводити до нестабільності та обмеженої підтримки порівняно з більш зрілими технологіями.
- Продуктивність: Може бути нижчою у порівнянні з нативними додатками, особливо в складних сценаріях.

Переваги Unity:

- Потужний графічний рушій: Ідеальний для створення 2D та 3D ігор з високою якістю графіки.
- Кросплатформеність: Підтримка багатьох платформ, включаючи Android, iOS, Windows, Mac, Linux та ігрові консолі.
- Широкий набір інструментів: Велика кількість інструментів та ресурсів для розробки ігор, включаючи Unity Asset Store.

Недоліки:

- Складність для звичайних додатків: Unity може бути надмірно складним для простих додатків або тих, що не потребують потужних графічних можливостей.
- Розмір додатків: Додатки, створені на Unity, можуть мати великий розмір через інтеграцію графічного рушія та ресурсів.

Отже, нативна розробка та кросплатформенні рішення мають свої переваги та недоліки, і вибір між ними залежить від специфіки проекту. В нашому проєкті були використана мова програмування C#. Саме тому, Unity обрано рушієм для гри. Це надало додаткові переваги для розробки проєкту, дозволяючи створювати високоякісний ігровий продукт для платформи Android.

1.1.3 Введення до розробки на Android

Android є однією з найпопулярніших платформ для розробки мобільних додатків. Операційна система Android, розроблена компанією Google, використовує відкрите програмне забезпечення та надає широкий спектр

інструментів і ресурсів для розробників. У цьому розділі ми розглянемо основні аспекти розробки додатків на Android, включаючи структуру платформи, інструменти для розробки та основні принципи.

Android складається з кількох ключових компонентів:

1. Linux Kernel: Це ядро, що забезпечує низькорівневі системні служби, такі як безпека, управління пам'яттю, управління процесами, мережевий стек та драйвери пристроїв [6].

2. Libraries: Android використовує набір бібліотек, написаних на C/C++, які підтримують різні компоненти системи, включаючи мультимедіа (Libc, SQLite), графіку (OpenGL, SGL), веб-браузер (WebKit) та інші [7].

3. Android Runtime (ART): Android Runtime є середовищем виконання, яке забезпечує виконання додатків. ART був представлений у Android 5.0 (Lollipop) і замінив Dalvik Virtual Machine, забезпечуючи кращу продуктивність та ефективність використання пам'яті.

4. Application Framework: Цей рівень надає високорівневі служби для розробки додатків, такі як управління активностями, вікнами, ресурсами, контентом, повідомленнями та інше [8].

5. Applications: Це кінцевий рівень, на якому розташовані всі додатки, включаючи основні системні додатки, такі як телефон, електронна пошта, браузер, а також сторонні додатки, створені розробниками [9].

Інструменти для розробки

1. Android Studio, є офіційною інтегрованою середою розробки (IDE) для розробки додатків на Android, розробленою компанією Google. Вона базується на IntelliJ IDEA і пропонує розробникам потужний набір інструментів для створення, тестування та налагодження додатків.

Основні можливості:

- Редактор коду: Пропонує зручний та потужний редактор коду з підсвічуванням синтаксису, автодоповненням, інтегрованими підказками та рефакторингом.
- Дизайнер інтерфейсу: Візуальний редактор дозволяє створювати та налаштовувати інтерфейс користувача за допомогою технології "drag-and-drop".
- Емулятор Android: Інтегрований емулятор дозволяє тестувати додатки на віртуальних пристроях з різними конфігураціями та версіями Android.
- Інструменти налагодження та профілювання: Включають інструменти для налагодження коду, аналізу продуктивності додатків та виявлення витоків пам'яті.
- Gradle: Інтегрована система збірки, що дозволяє автоматизувати процеси складання, тестування та розгортання додатків.
- Інтеграція з системами контролю версій: Підтримує популярні системи контролю версій, такі як Git, для ефективного управління кодом [10].

2. Firebase, є хмарною платформою від Google, яка надає набір інструментів та сервісів для розробки, тестування та монетизації мобільних і веб-додатків. Вона дозволяє розробникам швидко створювати потужні та масштабовані додатки.

Основні сервіси:

- Firebase Realtime Database: Хмарна база даних, яка дозволяє зберігати та синхронізувати дані в реальному часі між користувачами.
- Firebase Authentication: Сервіс для автентифікації користувачів з підтримкою соціальних мереж (Google, Facebook, Twitter) та електронної пошти/пароля.
- Firebase Cloud Messaging (FCM): Платформа для надсилання повідомлень та сповіщень на Android, iOS та веб-додатки.

- Firebase Analytics: Потужний інструмент для аналізу поведінки користувачів, що допомагає приймати обґрунтовані рішення для покращення додатку.
- Firebase Crashlytics: Інструмент для моніторингу та фіксації помилок, що допомагає швидко виявляти та виправляти проблеми.
- Firebase Remote Config: Дозволяє змінювати поведінку та зовнішній вигляд додатка без необхідності випускати нову версію [11].

3. Unity, є потужним рушієм для розробки ігор, який дозволяє створювати високоякісні 2D та 3D ігри для різних платформ, включаючи Android. Unity підтримує C# як основну мову програмування і надає розробникам широкий набір інструментів для створення інтерактивного контенту.

Основні можливості:

- Редактор Unity: Інтуїтивно зрозумілий візуальний редактор для створення ігрових сцен, налаштування об'єктів та написання скриптів.
- Підтримка крос-платформеності: Unity дозволяє створювати ігри, які можуть працювати на різних платформах, включаючи Android, iOS, Windows, Mac, Linux, та ігрові консолі.
- Потужний графічний рушій: Підтримка передових графічних можливостей, таких як рендеринг в реальному часі, шейдери, освітлення та тіні.
- Анімація та фізика: Інструменти для створення складних анімацій та фізичних взаємодій в іграх [12].
- Unity Asset Store: Магазин, де розробники можуть купувати або продавати різноманітні ресурси для ігор, такі як моделі, текстури, звуки, скрипти та інше [13].

- Інтеграція з іншими інструментами: Unity легко інтегрується з іншими сервісами та інструментами, такими як Firebase, для додавання функцій автентифікації, аналітики та сповіщень.

Основні розділи розробки

1. Компонентна архітектура: Додатки на Android складаються з компонентів, таких як активності (Activities), служби (Services), приймачі широкомовлень (Broadcast Receivers) та провайдери контенту (Content Providers). Кожен компонент виконує певну роль і може взаємодіяти з іншими компонентами.

2. UI/UX дизайн: Розробка інтуїтивно зрозумілого та привабливого інтерфейсу користувача є ключовим аспектом успіху додатку. Використання матеріального дизайну (Material Design) від Google допомагає створювати сучасні та зручні інтерфейси.

3. Оптимізація продуктивності: Для забезпечення плавної роботи додатку важливо оптимізувати використання ресурсів, таких як пам'ять, процесор та батарея. Це включає управління життєвим циклом компонентів, оптимізацію графіки та зменшення затримок.

4. Тестування: Регулярне тестування додатків на різних пристроях та версіях Android допомагає виявляти та виправляти помилки, забезпечуючи стабільну та надійну роботу додатку.

1.2. Аналіз та вибір оптимального рушія для платформи Android

1.2.1 Огляд популярних рушіїв Android (Unity, Unreal Engine, Cocos2d).

Приклади ігор на них

Unreal Engine, розроблений компанією Epic Games, відомий своєю високою продуктивністю та можливістю створення графічно насичених ігор. Unreal Engine часто використовують для створення ігор на консолях та ПК, але він також підтримує мобільні платформи. Unreal Engine використовує мови програмування C++ та Blueprint (візуальне скриптування) [14].

Приклади ігор:

- PUBG Mobile: Популярна королівська битва, яка використовує Unreal Engine для забезпечення високої якості графіки та продуктивності на мобільних пристроях.
- Fortnite: Відома гра в жанрі "Battle Royale", яка також доступна на мобільних пристроях завдяки Unreal Engine.

Cocos2d є безкоштовним відкритим ігровим рушієм, який спеціалізується на 2D іграх. Він підтримує кілька мов програмування, включаючи C++, Python, Lua та JavaScript. Cocos2d відомий своєю легкістю та високою продуктивністю, що робить його ідеальним для розробки мобільних ігор.

Приклади ігор:

- Angry Birds: Оригінальна версія цієї культової гри була створена на Cocos2d, демонструючи можливості рушія для створення успішних мобільних ігор.
- Clash of Kings: Популярна стратегічна гра, яка використовує Cocos2d для управління своїм графічним та ігровим контентом [15].

Unity є одним з найпопулярніших ігрових рушіїв у світі, особливо серед розробників мобільних ігор. Він підтримує 2D та 3D графіку, надає багатий набір інструментів для створення ігор та інших інтерактивних додатків. Unity використовує мову програмування C#, що робить його доступним і зручним для багатьох розробників.

Приклади ігор:

- Subway Surfers: Ця популярна гра у жанрі "runner" використовує Unity для створення плавного геймплею та яскравої 3D графіки.
- Monument Valley: Візуально приваблива головоломка, яка використовує Unity для реалізації складних візуальних ефектів та механік [17].

1.2.2 Переваги та недоліки кожного рушія

Переваги Unreal Engine:

- **Висока продуктивність:** Оптимізований для створення графічно насичених ігор з високим рівнем деталізації.
- **Потужний графічний рушій:** Підтримка передових графічних технологій, таких як просунуте освітлення та тіні, що робить його ідеальним для AAA-ігор.
- **Blueprint:** Візуальне скриптування, яке спрощує процес розробки та дозволяє швидко прототипувати ігрові механіки.
- **Безкоштовна ліцензія:** Безкоштовний доступ до всіх інструментів, з оплатою роялті лише після досягнення певного рівня доходу.

Недоліки Unreal Engine:

- **Крута крива навчання:** Вимагає більше часу та зусиль для опанування, особливо для новачків.
- **Великі розміри проектів:** Проекти на Unreal Engine можуть займати багато місця та вимагати значних апаратних ресурсів.
- **Менша підтримка 2D:** Порівняно з Unity, Unreal Engine менш орієнтований на розробку 2D-ігор.

Переваги Cocos2d:

- **Легкість:** Рушій оптимізований для 2D ігор, що робить його швидким та ефективним.
- **Відкритий код:** Безкоштовний та з відкритим кодом, що дозволяє розробникам модифікувати рушій відповідно до своїх потреб.
- **Підтримка кількох мов:** Підтримує C++, Python, Lua та JavaScript, що забезпечує гнучкість у виборі мови програмування.
- **Низькі системні вимоги:** Не вимагає потужного апаратного забезпечення, що підходить для розробки на старих чи менш продуктивних пристроях.

Недоліки Cocos2d:

- Обмежені можливості 3D: Основна орієнтація на 2D ігри, що обмежує його можливості для розробки 3D проектів.
- Менша спільнота: Порівняно з Unity та Unreal Engine, Cocos2d має меншу спільноту та менше доступних ресурсів.
- Менше інструментів: Обмежений набір інструментів та функцій у порівнянні з більш універсальними рушіями, такими як Unity.

Переваги Unity:

- Кросплатформеність: Підтримує більше 25 платформ, включаючи Android, iOS, Windows, Mac, консолі та веб-браузери.
- Широкий набір інструментів: Інтегроване середовище розробки (IDE) з потужними інструментами для створення як 2D, так і 3D ігор.
- Велика спільнота та підтримка: Широкий доступ до навчальних матеріалів, документації, форумів та ресурсів на Unity Asset Store.
- Інтеграція з C#: Використання сучасної об'єктно-орієнтованої мови програмування, що полегшує розробку.

Недоліки Unity:

- Вимоги до апаратного забезпечення: Може бути вимогливим до ресурсів комп'ютера, особливо для великих проектів.
- Ціна: Деякі функції та ліцензії можуть бути дорогими для інді-розробників та малих студій.
- Продуктивність: Іноді може поступатися за продуктивністю нативним рішенням або іншим рушіям при роботі з дуже вимогливими графічними завданнями.

Кожен із розглянутих рушіїв має свої переваги та недоліки. Unity є універсальним інструментом з широкою підтримкою платформ та великою спільнотою, але може бути вимогливим до апаратного забезпечення. Unreal Engine пропонує потужні графічні можливості та високу продуктивність, але

вимагає більше зусиль для навчання та роботи. Cocos2d є легким та ефективним рішенням для 2D ігор, але обмежений у можливостях для 3D розробки та має меншу спільноту. Вибір оптимального рушія залежить від специфіки проекту, вимог до графіки, ресурсів команди та цільової платформи.

1.2.3 Вибір оптимального рушія для гри

Наш проект полягає в створенні гри, яка поєднує в собі візуальний стиль і механіки Angry Birds з сеттингом світу Half-Life 2. Для реалізації цього амбітного задуму, ми обрали рушієм Unity. У цьому звіті детально пояснюється, чому Unity є найкращим вибором для нашого проекту, враховуючи його технічні можливості, гнучкість, простоту використання та інші важливі фактори.

Потужність та гнучкість рушія

- Unity підтримує як 2D, так і 3D графіку, що дозволяє створювати різноманітні ігрові проекти. Для нашої гри, яка поєднує візуальний стиль Angry Birds з тривимірним середовищем світу Half-Life 2, це надзвичайно важливо. Ми можемо використовувати 2D фізику для реалізації механік запуску об'єктів та взаємодії з оточенням, а також 3D моделі для створення більш реалістичних та деталізованих об'єктів.
- Unity має вбудовані інструменти для роботи з фізикою, включаючи Box2D для 2D фізики та PhysX для 3D фізики. Це дозволяє нам точно моделювати поведінку об'єктів у грі, забезпечуючи реалістичність взаємодії з оточенням, що є ключовим аспектом нашої гри.
- Unity дозволяє легко експортувати гру на різні платформи, включаючи Android. Це особливо важливо для нашого проекту
- Unity має інтуїтивно зрозумілий інтерфейс і багато навчальних ресурсів, що полегшує процес навчання та розробки навіть для новачків. Це дозволяє нам швидко створювати прототипи та вносити зміни до гри в процесі розробки.

- Велика спільнота розробників Unity забезпечує доступ до безлічі безкоштовних та платних ресурсів, скриптів, туторіалів та форумів. Ми можемо легко знайти допомогу з будь-якого питання та використовувати готові рішення для прискорення розробки.
- Unity Asset Store пропонує широкий вибір плагінів та готових ресурсів, що значно прискорює розробку гри. Ми можемо інтегрувати готові фізичні моделі, звукові ефекти та візуальні елементи, зосереджуючись на унікальних аспектах нашої гри.
- Unity надає потужні інструменти для створення користувацького інтерфейсу, що дозволяє легко розробити меню, інформаційні панелі та інші елементи UI. Це важливо для створення зручного та естетично привабливого інтерфейсу нашої гри.
- Для реалізації механік Angry Birds, таких як запуск об'єктів з рогатки та взаємодія з оточенням, необхідна точна та надійна фізична система. Unity має вбудовану підтримку Box2D для 2D фізики, що ідеально підходить для нашої гри.

Враховуючи всі наведені фактори, Unity є оптимальним вибором для нашого проекту. Його потужність, гнучкість, простота використання, велика спільнота розробників та широкі можливості для роботи з фізикою, анімацією та інтерфейсом дозволять нам швидко та ефективно реалізувати всі аспекти нашого проекту, забезпечуючи високу якість кінцевого продукту.

1.3 Аналіз існуючих ігрових додатків для Android

1.3.1 Огляд популярних ігрових додатків на Android

Мобільні ігри займають значну частину ринку мобільних додатків і є одними з найпопулярніших завантажуваних програм у світі. Успіх мобільних ігор залежить від різних факторів, включаючи цікаву ігрову механіку, привабливу графіку, зручність інтерфейсу та ефективну монетизацію. Давайте розглянемо

кілька найпопулярніших ігрових додатків на Android, які стали зразком для наслідування в цій галузі.

1. Subway Surfers

- Розробник: Kiloo, SYBO Games
- Жанр: Раннер (Endless Runner)
- Короткий опис: Subway Surfers — це одна з найпопулярніших ігор у жанрі ранерів. Гравці керують персонажем, що бігає по залізничних коліях, уникаючи перешкод та збираючи монети і бонуси. Гра приваблює яскравою графікою, динамічним геймплеем та постійними оновленнями.

Особливості:

- Високоякісна та яскрава графіка
- Захоплюючий і нескінченний геймплей
- Різноманітні локації, що постійно змінюються
- Регулярні оновлення з новими персонажами та подіями [17].

2. Angry Birds

- Розробник: Rovio Entertainment
- Жанр: Головоломка (Puzzle)
- Короткий опис: Angry Birds — це класична гра у жанрі головоломок, де гравці використовують різноманітних птахів для знищення конструкцій, в яких ховаються свині. Гра здобула велику популярність завдяки своїй оригінальній ідеї, веселій графіці та затягуючому геймплею [18].

Особливості:

- Унікальна ігрова механіка
- Висока якість графіки та звуку
- Різноманітні рівні та персонажі з унікальними здібностями
- Постійні оновлення та нові епізоди [19].

3. Doodle Jump

- Розробник: Lima Sky
- Жанр: Аркада (Arcade)
- Короткий опис: Doodle Jump — це одна з найвідоміших аркадних ігор, де гравці керують персонажем, який стрибає по платформах, намагаючись досягти якомога більшої висоти. Гра відзначається простим, але захоплюючим геймплеєм та впізнаваною графікою в стилі "карандашного" малюнка.

Особливості:

- Проста, але затягуєча ігрова механіка
- Впізнавана графіка у стилі ручного малюнка
- Різноманітні перешкоди та бонуси
- Постійні оновлення з новими темами та рівнями [20].

Аналізуючи популярні ігрові додатки на Android, такі як Subway Surfers, Angry Birds та Doodle Jump, можна зробити висновок, що успіх таких ігор залежить від якості графіки, інноваційної ігрової механіки, зручності користування та регулярних оновлень. Ідеєю є використання мови програмування C# разом з технологіями, надає розробникам потужні інструменти для створення високоякісних ігор, здатних конкурувати з лідерами ринку.

1.3.2 Аналіз функціональності та особливостей

Розглядаючи успішні ігрові додатки для Android, важливо аналізувати не тільки їхню популярність, але й функціональність та особливості, що забезпечили їм таку популярність. Нижче наведено аналіз функціональності та особливостей трьох популярних ігор: Subway Surfers, Angry Birds та Doodle Jump.

1. Subway Surfers

Функціональність:

- Ігровий процес: Гравець керує персонажем, який безперервно біжить по залізничних коліях, збираючи монети та бонуси і уникаючи перешкод. У разі зіткнення з перешкодами, гру закінчено.
- Бонуси та посилення: Гра містить різноманітні бонуси, такі як магніти для монет, реактивні ранці та супер-кеди, що полегшують проходження гри.
- Мікротранзакції: Гравці можуть купувати віртуальну валюту та посилення за реальні гроші.

Особливості:

- Графіка та анімація: Високоякісна, яскрава графіка з плавною анімацією.
- Різноманітні локації: Кожне оновлення додає нові локації, що змінюються відповідно до тематичних свят чи подій.
- Соціальна інтеграція: Можливість змагатися з друзями через підключення до соціальних мереж.
- Персонажі: Великий вибір персонажів, які можна розблокувати за монети чи реальні гроші.

2. Angry Birds

Функціональність:

- Ігровий процес: Гравці запускають птахів з рогатки, намагаючись зруйнувати структури, в яких ховаються свині. Кожен рівень має певну кількість птахів та цілей для знищення.
- Різноманітність птахів: Кожен птах має унікальні здібності, що додають стратегії до гри.
- Мікротранзакції: Можливість придбання додаткових птахів або рівнів за реальні гроші.

Особливості:

- Фізика гри: Реалістична фізика, яка визначає, як руйнуються структури.

- Графіка та звук: Барвиста графіка з веселими звуковими ефектами та музикою.
- Рівні та епізоди: Велика кількість рівнів та тематичних епізодів, що додаються з оновленнями.
- Досягнення та таблиці лідерів: Можливість змагатися з іншими гравцями за високі бали та досягнення.

3. Doodle Jump

Функціональність:

- Ігровий процес: Гравець керує персонажем, що постійно стрибає вгору, уникаючи перешкод і збираючи бонуси. Гра закінчується, якщо персонаж падає або стикається з ворогом.
- Бонуси: У грі є різноманітні бонуси, такі як реактивні ранці, пружини та захисні щити, що допомагають досягати більшої висоти.
- Мікротранзакції: Гравці можуть купувати спеціальні костюми чи бонуси за реальні гроші.

Особливості:

- Графіка: Впізнавана графіка у стилі "карандашного" малюнка, що надає грі унікального вигляду.
- Проста механіка: Легко зрозуміла механіка гри, що робить її доступною для всіх вікових груп.
- Різнманітні теми: Гра має різні тематичні світи, що змінюються з оновленнями.
- Таблиці лідерів: Можливість змагатися з іншими гравцями за найвищий результат.

1.3.3 Аналіз ринку: цільова аудиторія, популярні жанри, бізнес-моделі

Аналіз ринку мобільних ігор дозволяє краще зрозуміти, які фактори впливають на успіх ігор, та як їх розробники можуть досягти високих показників

завантажень та доходів. У цьому розділі ми розглянемо цільову аудиторію, популярні жанри мобільних ігор та різні бізнес-моделі, що використовуються для монетизації.

1) Мобільні ігри мають широку цільову аудиторію, яка охоплює різні вікові та соціальні групи. Основні сегменти цільової аудиторії включають:

- Діти та підлітки: Ця група є однією з найбільших споживачів мобільних ігор. Ігри для цієї аудиторії зазвичай мають яскраву графіку, простий геймплей та безліч міні-ігор.
- Молодь та дорослі (18-35 років): Цей сегмент часто грає у більш складні та стратегічні ігри. Популярні жанри серед цієї групи включають екшн, стратегії, RPG та головоломки.
- Дорослі (35+ років): Цей сегмент зазвичай віддає перевагу казуальним іграм, головоломкам та картковим іграм. Для них важливими є релаксація та розваги у вільний час.

2) Мобільні ігри охоплюють широкий спектр жанрів, серед яких деякі з найпопулярніших включають:

- Казуальні ігри (Casual Games): Це ігри з простими правилами та короткими ігровими сесіями. Прикладом є Candy Crush Saga.
- Раннери (Endless Runners): Ігри, де персонаж постійно рухається вперед, а гравець повинен уникати перешкод. Прикладом є Subway Surfers.
- Головоломки (Puzzle Games): Ігри, що вимагають від гравців розв'язання різноманітних завдань та головоломок. Прикладом є Angry Birds.
- Стратегії (Strategy Games): Ігри, що вимагають стратегічного планування та управління ресурсами. Прикладом є Clash of Clans.
- Рольові ігри (RPG): Ігри, де гравці керують персонажами, виконують квести та борються з ворогами. Прикладом є Genshin Impact.

3) Для монетизації мобільних ігор використовуються різні бізнес-моделі, серед яких найпоширенішими є:

- Freemium: Основна модель, за якою гра безкоштовна для завантаження, але містить платні елементи (мікротранзакції) для придбання додаткових функцій, посилень або косметичних предметів. Прикладом є Clash of Clans.
- Реклама (Ad-supported): Ігри, що показують рекламу під час гри або між рівнями. Гравці можуть мати можливість відключити рекламу шляхом внутрішньоігрових покупок. Прикладом є Doodle Jump.
- Преміум (Paid Games): Ігри, що продаються за фіксовану ціну. Ця модель менш популярна на мобільних платформах через високий рівень конкуренції та велику кількість безкоштовних альтернатив.
- Підписка (Subscription): Ігри або сервіси, що пропонують доступ до преміум-контенту за щомісячну або річну підписку. Ця модель стає все більш популярною завдяки сервісам на зразок Apple Arcade та Google Play Pass.

1.4 Особливості проектування гри

1.4.1 Концепція Half-Life 2

"Half-Life 2" - це відома відеогра, розроблена компанією Valve Corporation і випущена у 2004 році. Гра є продовженням культової "Half-Life" і здобула широкий визнання за свій інноваційний підхід до геймплею та захоплюючий сюжет.

Сюжет: Гравець у "Half-Life 2" відіграє роль Гордона Фрімена, фізика, який стає героєм боротьби проти влади інопланетних загарбників. Дія гри розгортається у місті City 17, яке знаходиться під контролем загарбників. Гравець прокладає шлях через різноманітні локації, вступаючи в боротьбу з ворожими силами, спілкуючись з іншими персонажами та вирішуючи загадки для досягнення своїх цілей.

Геймплей: Гра пропонує різноманітні геймплейні механіки, включаючи елементи шутера від першої особи, пазли, фізичні головоломки та елементи прихованих об'єктів. Гравець володіє різноманітними знаряддями, такими як гравітаційна пушка та різні види зброї, які допомагають йому в боротьбі з ворогами та вирішенні завдань. "Half-Life 2" відома своєю реалістичною фізикою та інноваційним використанням середовища для створення цікавих і високодинамічних боїв та сценаріїв [21].

1.4.2 Концепція Angry Birds

"Angry Birds" - це популярна мобільна гра, розроблена компанією Rovio Entertainment. Гра вийшла у 2009 році і стала однією з найуспішніших ігор у світі мобільних додатків.

Сюжет: у грі "Angry Birds" немає складного сюжету. Гравець керує групою пташок, які вирішили помститися злобним свиням, які вкрали їхні яйця. Мета гравця - використовуючи рогатку, запускати пташок у будівлі, де перебувають свині, знищуючи їх і відновлюючи свої яйця.

Геймплей: у "Angry Birds" гравець керує напрямком та силою вистрілу пташкою з рогатки, щоб знищити споруди, за якими приховані свині. Гравець може використовувати різні типи пташок, кожна з яких має унікальні властивості, наприклад, пташки можуть літати, розбивати, вибухати тощо.

Рогатка в грі використовується для запуску пташок у напрямку свиней та їхніх будівель. Гравець може регулювати напрямок та силу вистрілу, щоб досягти максимального ушкодження.

Головними ворогами в грі є злобні свині, які приховуються у різних будівлях. Гравець повинен знищити будівлі та свиней, використовуючи пташок. Для цього гравець має враховувати різні фізичні закони та використовувати унікальні властивості кожної пташки для максимально ефективного ураження цілей.

1.4.3 Концепція гри проєкту: жанр, сюжет, механіка

Концепція гри проєкту

Жанр: аркадна головоломка

Сюжет: Події гри відбуваються у постапокаліптичному світі гри Half-Life 2, однак гра представлена у візуальному стилі та механіках, схожих на Angry Birds. Після повстання проти Альянсу, люди продовжують боротьбу з залишками загарбників та їхніми пристосуваннями. Гравець бере на себе роль одного з повстанців, які використовують імпровізовану зброю та різні пристрої для знищення баз ворога.

- Основна механіка. Гравець використовує катапульту для запуску різних снарядів (імпровізованої зброї) з метою зруйнування ворожих укріплень та знищення ворогів. Снаряди(сам гравець) можуть мати різні властивості: вибухові, розсічні, з ефектом розсіювання об'єктів тощо.
- Рівні. Кожен рівень представляє собою невелике укріплення ворога з різними структурами та перешкодами. Завдання гравця — першочергово знищити ворогів та руйнувати укріплення, використовуючи обмежену кількість снарядів.
- Бали. За кожен зруйнований об'єкт та знищеного ворога гравець отримує бали. Мета — набрати якомога більше балів, використовуючи якнайменше снарядів.
- Спеціальні снаряди. Гравець може отримувати спеціальні снаряди, які мають унікальні властивості (наприклад, електромагнітні імпульси для знищення електронних пристроїв Альянсу або вибухові снаряди для знищення великої кількості об'єктів).
- Вороги. Вороги можуть бути різними: солдати Альянсу, турелі, страйдери та інші ворожі одиниці з Half-Life 2. Кожен ворог має свої особливості та способи взаємодії з ними.

- Візуальний стиль гри буде поєднувати елементи з Half-Life 2 та Angry Birds. Гра матиме мультяшний вигляд, але з елементами постапокаліптичного світу, властивими Half-Life 2: зруйновані будівлі, механізми Альянсу, повстанці у саморобному спорядженні тощо.

Основні елементи:

- Мультиплікаційна графіка з постапокаліптичними елементами. Звуковий супровід включатиме як музику, так і звукові ефекти, властиві світу Half-Life 2 (звуки при запуску снарядів та розриви, характерні для вибухів, вороги, взаємодія із оточенням).
- Простий і інтуїтивно зрозумілий інтерфейс, схожий на Angry Birds, з додатковими елементами, що пояснюють ситуацію у світі Half-Life 2. Основними елементами є головне меню, екран вибору рівнів, цільова зона, інформаційна панель, меню паузи, екран завершення рівня, підказки щодо управління.
- Локації представлятимуть різні зони з світу Half-Life 2, такі як Сіті 17, лігва створінь Зен, лісові табори повстанців та інші.

Гра поєднає у собі динамічний та захоплюючий геймплей Angry Birds з атмосферою та елементами світу Half-Life 2, надаючи гравцям унікальний і захоплюючий досвід.

Висновки до розділу 1

У цьому розділі було розглянуто теоретичні аспекти розробки мобільних ігрових додатків на платформі Android з використанням мови програмування C# та різних ігрових рушіїв.

Аналіз існуючих ігрових додатків для Android показав, що популярні ігри, такі як Subway Surfers, Angry Birds і Doodle Jump, мають спільні риси: захоплюючий геймплей, просте управління та яскравий візуальний стиль. Ці ігри використовують інноваційні механіки, що робить їх цікавими та привабливими для широкої аудиторії. Основні функції цих ігор, такі як різноманітні рівні, системи прогресії та інтеграція соціальних елементів, забезпечують довготривалу залученість гравців. Аналіз ринку показав, що основними цільовими аудиторіями є підлітки та молоді дорослі, а популярними жанрами залишаються аркади, головоломки та екшн-ігри. Сучасні бізнес-моделі, такі як freemium та рекламо-орієнтовані моделі, дозволяють розробникам досягати успіху та стабільного доходу.

Огляд технологій розробки додатків на рушії Android виявив, що Android Studio, Firebase і Unity є основними інструментами для розробки. Android Studio пропонує потужний IDE з інструментами для нативної розробки додатків, Firebase надає функціонал для бекенд-інфраструктури та аналітики, а Unity дозволяє створювати як 2D, так і 3D ігри з використанням C#. Серед мов програмування, Java, C++ та C# є основними для розробки на Android, кожна з яких має свої переваги: Java - простота та широке використання, C++ - висока продуктивність, а C# - зручність та потужні можливості в контексті Unity.

Порівняння нативної розробки та кросплатформних рішень показало, що нативна розробка забезпечує максимальну продуктивність і доступ до всіх можливостей платформи, але кросплатформні рішення дозволяють створювати додатки для кількох платформ одночасно, зменшуючи час та ресурси, необхідні для розробки.

Проектування гри включало вибір інструментів та технологій для проектування, таких як Unity та C# для основної розробки, а також Adobe Photoshop для створення візуального контенту. Планування ігрового процесу та основних функцій допомогло визначити ключові елементи гри, включаючи жанр, сценарій, механіку, систему управління, прогресію та досягнення, що забезпечують захоплюючий та цілісний геймплей.

Аналіз та вибір оптимального рушія для платформи Android розглянув популярні рушії, такі як Unity, Unreal Engine та Cocos2d. Unity був обраний як оптимальний рушій завдяки його універсальності, кросплатформеності, підтримці 2D та 3D графіки, легкості у використанні, великій спільноті та інтеграції з іншими інструментами.

Отже, перший розділ забезпечив всебічне розуміння теоретичних основ розробки мобільних ігрових додатків на платформі Android, допоміг визначити ключові аспекти вибору інструментів та технологій, а також обґрунтував вибір C#, як мови програмування, а Unity — як оптимального рушія для створення високоякісних ігрових продуктів. Це створює міцну основу для подальшого практичного етапу розробки, який буде розглянуто в наступних розділах.

РОЗДІЛ 2

ЕТАПИ РОЗРОБКИ ДОДАТКУ

2.1. Встановлення середовища розробки та налаштування проекту

2.1.1. Встановлення необхідних інструментів (Visual Studio, Unity, Adobe Photoshop)

У рамках розробки мобільної гри на C# було проведено встановлення та налаштування наступних програмних інструментів:

1. Visual Studio:

- 1) Завантажено та встановлено безкоштовну пробну версію Visual Studio 2022 з офіційного сайту Microsoft.
- 2) Налаштовано середовище розробки C# згідно з особистими уподобаннями.
- 3) Ознайомилися з основними функціональними можливостями IDE Visual Studio.

2. Unity:

- 1) Завантажено та встановлено безкоштовну особисту версію Unity 2023 з офіційного сайту Unity.
- 2) Налаштовано середовище розробки 2D проєктів згідно з особистими уподобаннями.
- 3) Під час встановлення обрано потрібні модулі, включаючи підтримку платформи Android.
- 4) Ознайомився з основними функціональними можливостями ігрового рушія Unity [22].

3. Adobe Photoshop:

- 1) Завантажено та встановлено безкоштовну пробну версію Adobe Photoshop 2023 з офіційного сайту Adobe.

- 2) Налаштовано інтерфейс користувача Photoshop для зручної роботи з графікою.
- 3) Встановлення та налаштування вищезазначених інструментів створило фундамент для подальшої розробки мобільної гри на C#.

2.1.2 Налаштування проекту Unity на Android (2D гра)

1. Створення нового проекту:

Запустили Unity. Вибрали тип проекту "2D". Надали назву проекту "MobileGame2D". В полі "Build Settings" вибрано платформу "Android".

2. Налаштування Player Settings:

- 1) Перейдено до Edit > Build Settings.
- 2) У розділі Platform підтверджено вибір "Android".
- 3) У розділі Player Settings налаштовано наступні параметри:
- 4) Target SDK Version: Android 13 (найновіша версія SDK, що підтримується).
- 5) Minimum API Level: Android 11 (мінімальна версія Android, на якій буде працювати гра).
- 6) Screen Size and Resolution: HD (1280x720), що відповідає типовому розміру екранів мобільних пристроїв для 2D ігор.
- 7) Scripting Backend: IL2CPP для кращої продуктивності.
- 8) Texture Compression: ETC2 для оптимального балансу між якістю та розміром текстур.
- 9) Other Settings: Іконка програми та ім'я пакета налаштовані відповідно до брендування гри.

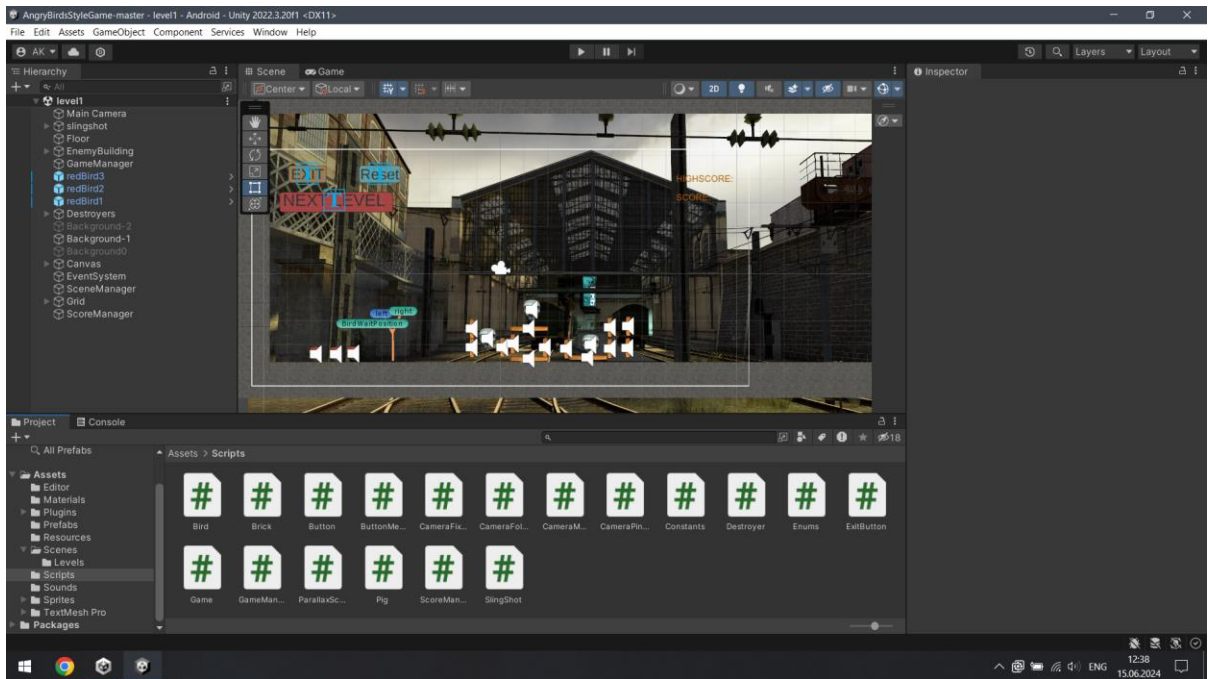


Рис.2.1. Середовище розробки, інтерфейс Unity

3. Налаштування сцен:

Перейшли до Window > Scene. Переконалися, що всі сцени, які повинні бути включені до гри, додані до списку Build Settings. Налаштували порядок сцен у списку Build Settings.

4. Будівництво проекту:

У File > Build Settings натиснули кнопку Build. обрали розташування для збереження APK-файлу гри. Unity збудував APK-файл, який можна встановити на пристрій Android або емулятор.

В результаті проведених налаштувань проект Unity "Angry Lambda" підготовлено до впровадження основних механік [23].

2.2 Реалізація головної логіки гри

2.2.1 Розробка ігрової механіки

Перш за все, визначено основні елементи ігрової механіки, які будуть складати основу гри:

- Правила гри: Встановлення правил, за якими буде відбуватися ігровий процес.
- Цілі та завдання: Окреслення цілей, які гравець повинен досягти для переходу на новий рівень або завершення гри.

Було вирішено, в першу чергу додати механіку рогатки Angry Birds та тестові об'єкти на першу версію сцени.

Скрипт SlingShot відповідає за механіку рогатки в грі, включаючи взаємодію користувача з рогаткою та кидання птаха(персонажа).

Метод Start

```
void Start()
{
    // Встановлення імені шару сортування для лінійних рендерерів
    SlingshotLineRenderer1.sortingLayerName = "Foreground";
    SlingshotLineRenderer2.sortingLayerName = "Foreground";
    TrajectoryLineRenderer.sortingLayerName = "Foreground";

    slingshotState = SlingshotState.Idle;
    SlingshotLineRenderer1.SetPosition(0, LeftSlingshotOrigin.position);
    SlingshotLineRenderer2.SetPosition(0, RightSlingshotOrigin.position);
    // Направлення на середню позицію двох векторів
    SlingshotMiddleVector = new Vector3((LeftSlingshotOrigin.position.x +
    RightSlingshotOrigin.position.x) / 2,
    (LeftSlingshotOrigin.position.y + RightSlingshotOrigin.position.y) / 2, 0);
}
```

- Встановлює початкові значення для лінійних рендерерів.
- Встановлює початковий стан рогатки (Idle).
- Визначає середню позицію між лівою та правою частинами рогатки.

Метод Update

```
void Update()
{
```

```

switch (slingshotState)
{
    case SlingshotState.Idle:
        InitializeBird();
        DisplaySlingshotLineRenderers();

```

- **void Update():** Це стандартний метод Unity, який викликається один раз за кадр. Він використовується для оновлення логіки гри.
- **switch (slingshotState):** Використовується для вибору дії залежно від поточного стану рогатки (slingshotState).
- **case SlingshotState.Idle:** Виконується, коли рогатка перебуває в стані "Idle" (очікування).
- **InitializeBird():** Викликає метод, який встановлює птаха в початкову позицію для готовності до запуску.
- **DisplaySlingshotLineRenderers():** Відображає лінії рогатки, які з'єднують птаха з лівою і правою частинами рогатки.

```

    case SlingshotState.UserPulling:
        DisplaySlingshotLineRenderers();
        if (Input.GetMouseButton(0))
        {
            Vector3 location = Camera.main.ScreenToWorldPoint(Input.mousePosition);
            location.z = 0;

```

- **case SlingshotState.UserPulling:** Цей блок виконується, коли гравець тягне птаха назад перед кидком.
- **DisplaySlingshotLineRenderers():** Викликає метод, який відображає лінії рогатки.
- **if (Input.GetMouseButton(0)):** Перевіряє, чи натиснута ліва кнопка миші.


```

                if (Vector3.Distance(location, SlingshotMiddleVector) > 1.5f)
                {

```

```

        var maxPosition = (location - SlingshotMiddleVector).normalized * 1.5f +
SlingshotMiddleVector;
        BirdToThrow.transform.position = maxPosition;
    }
    else
    {
        BirdToThrow.transform.position = location;
    }

```

- `if (Vector3.Distance(location, SlingshotMiddleVector) > 1.5f)`: Перевіряє, чи відстань між курсором миші та серединою рогатки більше 1.5.
- `var maxPosition = (location - SlingshotMiddleVector).normalized * 1.5f + SlingshotMiddleVector`: Обчислює максимальну позицію птаха для тягнення назад.
- `BirdToThrow.transform.position = maxPosition`: Встановлює птаху позицію для тягнення назад до максимальної відстані.

```

        float distance = Vector3.Distance(SlingshotMiddleVector,
BirdToThrow.transform.position);
        DisplayTrajectoryLineRenderer2(distance);

```

- `float distance = Vector3.Distance(SlingshotMiddleVector, BirdToThrow.transform.position)`: Обчислює відстань між серединою рогатки та птахом.
- `DisplayTrajectoryLineRenderer2(distance)`: Викликає метод, який відображає прогнозовану траєкторію польоту птаха.

```

    }
    else
    {
        SetTrajectoryLineRenderesActive(false);
        TimeSinceThrown = Time.time;
        float distance = Vector3.Distance(SlingshotMiddleVector,
BirdToThrow.transform.position);

```

```

if (distance > 1)
{
    SetSlingshotLineRenderersActive(false);
    slingshotState = SlingshotState.BirdFlying;
    ThrowBird(distance);
}
else
{
    BirdToThrow.transform.position = BirdWaitPosition.position;
}
}
break;

```

- **TimeSinceThrown = Time.time:** Запам'ятовує час, коли птах був кинутий. важлива частина коду, оскільки використовується для визначення часу, який пройшов з моменту кидка птаха. Використання цього параметра дозволяє контролювати різні аспекти гри після кидка птаха, такі як зміна стану рогатки, взаємодія з іншими об'єктами або відлік часу до нового кидка.
- **ThrowBird(distance):** Кидає птаха з визначеною силою та відстанню.

Цей блок коду відповідає за обробку стану рогатки, коли гравець тягне птаха назад перед кидком. Він перевіряє, чи натиснута кнопка миші, обчислює позицію птаха в залежності від положення миші, відображає прогнозовану траєкторію політів птаха та виконує додаткові дії, якщо користувач відпускає кнопку миші.

Наступна частина скрипту відповідає за лінійні рендери рогатки та відображення траєкторії польоту птаха.

Після розробки скрипту, він був налаштований на сцені. Були додані префаби структур та птаха, додані їм фізичні форми. Додана камера що відповідає за стеження за польотом птаха.

2.2.2 Реалізація основних функцій та взаємодій у грі

Наступним етапом було проробити об'єкти Bird та Pig, написати до них однойменні скрипти. Перша є персонажем, що летить з рогатки. Другий – ворог, якого треба усунути

Скрипт Bird

[RequireComponent(typeof(Rigidbody2D))]

Ця атрибутна мітка вказує, що цей клас потребує наявності компонента Rigidbody2D. Якщо компонент відсутній, Unity автоматично додаст його до об'єкту птаха.

У методі Start() метод: Цей метод викликається при старті гри. Він виконано налаштування для птаха:

- Вимикання TrailRenderer, щоб слід птаха не був видимим до кидка.
- Встановлення шару сортування для TrailRenderer.
- Вимкнення гравітації для птаха
- Збільшення радіусу колайдера птаха для полегшення торкання.

FixedUpdate() метод:

```
void FixedUpdate()
{
    if (State == BirdState.Thrown &&
        GetComponent<Rigidbody2D>().velocity.sqrMagnitude <= Constants.MinVelocity)
    {
        StartCoroutine(DestroyAfter(2));
    }
}
```

Цей метод викликається з фіксованою частотою. Він перевіряє, чи птах досить далеко від рогатки після кидка. Також, якщо птах приземлився, він знищується.

Також, було додано: метод `OnThrow()`: Цей метод викликається при кидку птаха з рогатки. Він відтворює звук кидка птаха, активує `TrailRenderer`, дозволяє гравітацію для птаха і змінює розмір його колайдера. Також встановлює стан птаха на "Thrown"; Метод `DestroyAfter()` (у вигляді `IEnumerator`): Цей метод використовується для автоматичного знищення птаха після певного часу після його кидка; Властивість `State`: Ця властивість дозволяє доступ до поточного стану птаха, такого як "BeforeThrown" або "Thrown". Вона дозволяє іншим частинам коду знаходити стан птаха і взаємодіяти з ним відповідно до його поточного стану.

Скрипт Pig

Були додані нові елементи: `Health` - Визначає здоров'я свині. За замовчуванням, воно рівне 150; `SpriteShownWhenHurt`: Спрайт, який відображається при пораненні свині; `ChangeSpriteHealth`: Здоров'я, при якому спрайт свині змінюється; `points`: Кількість очок, які гравець отримує за знищення свині та об'єкт класу `ScoreManager`, який відповідає за ведення рахунку (докладніше описано в пунктах інтерфейсу)

Метод `Start()`:

```
void Start()
{
    ChangeSpriteHealth = Health - 30f;
    audioSource = GetComponent<AudioSource>();
    scoreManager = FindObjectOfType<ScoreManager>();
}
```

- Ініціалізує поле `ChangeSpriteHealth`, віднімаючи 30 одиниць від загального здоров'я `Health`.

- Отримує посилання на аудіо джерело, приєднане до цього об'єкта.
- Знаходить на сцені об'єкт класу ScoreManager і зберігає посилання на нього.

Метод OnCollisionEnter2D(Collision2D col):

```
void OnCollisionEnter2D(Collision2D col)
{
    if (col.gameObject.GetComponent<Rigidbody2D>() == null) return;
    if (col.gameObject.tag == "Bird")
    {
        //GetComponent<AudioSource>().Play();
        StartCoroutine(DestroyPigWithDelay());
    }
    else
    {
        float damage = col.gameObject.GetComponent<Rigidbody2D>().velocity.magnitude
* 10;

        Health -= damage;
        if (Health < ChangeSpriteHealth)
        {
            GetComponent<SpriteRenderer>().sprite = SpriteShownWhenHurt;
        }
        if (Health <= 0)
        {
            StartCoroutine(DestroyPigWithDelay());
        }
    }
}
```

- Це обробник зіткнень, який викликається при кожному зіткненні з цим об'єктом.
- Перевіряє, чи зіткнення має об'єкт з Rigidbody2D. Якщо немає, обробка припиняється.

- У протилежному випадку розраховується пошкодження від зіткнення, зменшується здоров'я свині, змінюється спрайт свині при досягненні певного рівня здоров'я, а також запускається корутина на видалення свині при смерті.

2.2.3. Інтеграція графіки та звуку

Уся графіка в грі виконана в 2Д стилі з використанням спрайтів для надання об'єктам відповідного вигляду. Наприклад, будівлі на ігрових рівнях представлені з дерев'яною основою, що руйнується під час гри. Стіни об'єктів мають текстуру фундаменту, що надає відчуття непробивності. Ці елементи були додані з використанням інструменту Tilemap.

Об'єкти Pig в їхньому звичайному стані відображають текстуру метрокопів з гри Half-Life. Однак, коли вони отримують пошкодження, їх спрайт змінюється на пошкоджений вигляд. Щодо об'єкта Bird, він має перероблений вигляд, відповідний всесвіту гри, з якої він походить. Обидва ці об'єкти використовують спрайти та точні налаштування колайдерів для досягнення необхідної поведінки в грі.

У об'єкті 'Bird' звук кидка птаха надається за допомогою методу 'OnThrow()'. При виклику цього методу відтворюється звук кидка, використовуючи 'GetComponent().Play()'. Окрім цього, також активується 'TrailRenderer', який створює слід за рухом птаха після кидка, що надає більш реалістичний ефект.

У об'єкті 'Pig' звук при знищенні свині надається за допомогою методу 'DestroyPigWithDelay()'. Цей метод є корутином, який спочатку відтворює звук за допомогою 'GetComponent().Play()', а потім затримує виконання програми на 2 секунди, використовуючи 'yield return new WaitForSeconds(2)'. Після цього він додає гравцеві очки за знищення свині і видаляє саму свиню з використанням 'Destroy(gameObject)'.

2.3. Створення інтерфейсу користувача

2.3.1. Інтерфейс та його види

- Дієгетичний інтерфейс вписаний у світ гри і пояснений наративом, тобто не гравець, а самі персонажі можуть взаємодіяти з ним. Такий інтерфейс посилює занурення, але дієгетичні елементи важко зробити зручними. Це вимагає додаткових зусиль дизайнерів.
- Недієгетичні елементи інтерфейсу жодним чином не пояснюються з погляду історії та не присутні в ігровому просторі. Це означає, що їх бачить тільки гравець. Такі інтерфейси використовуються, щоб відстежувати шкоду, очки, час, ресурси, обирати зброю, стежити за міні-картою тощо. Вважається, що вони заважають зануренню, проте саме такі інтерфейси трапляються в більшості ігор.
- Елементи просторового інтерфейсу знаходяться в ігровому світі, але персонажі їх не бачать. Найчастіше такі елементи застосовуються для того, щоб вказати гравцеві на певний предмет або дати якесь пояснення. Це можуть бути стрілки з напрямками в перегонах, світна аура навколо інтерактивних предметів, текстові підписи об'єктів тощо.
- Мета-інтерфейс існує у світі гри, але його не видно в ігровому просторі. Головний герой може знати і не знати про існування цих елементів. Найчастіше такі інтерфейси використовуються для того, щоб показати стан аватара або його відчуття. Це може бути червоний фільтр, який з'являється, коли герой має сильні пошкодження, бруд, краплі крові або дощу, які потрапляють йому на обличчя.

Інтерфейс не можна перевантажувати, він має доповнювати ігровий процес, не перетягуючи на себе всю увагу гравця. Також стоїть завдання зробити інтерфейс зрозумілим, тобто інформативним і таким, що легко сприймається.

Після ретельного аналізу різних типів інтерфейсів було зроблено висновок, що недієгетичний інтерфейс є найкращим вибором для даної гри з кількох причин:

- **Практичність:** Недієгетичні елементи інтерфейсу, такі як індикатор здоров'я, карта, меню зброї, зазвичай більш зручні для гравця, адже вони чітко візуалізовані, легко читаються та не захаращують ігровий простір.
- **Збалансування занурення та інформативності:** Незважаючи на те, що дієгетичні інтерфейси можуть посилювати занурення за рахунок їх інтеграції в ігровий світ, вони не завжди практичні. Недієгетичні інтерфейси, хоча й не доповнюють ігровий світ напряму, ефективно надають необхідну гравцеві інформацію, не заважаючи його зануренню в ігровий процес.
- **Естетика:** Недієгетичні інтерфейси мають більше свободи в дизайні, адже вони не обмежені рамками ігрового світу. Це дає можливість створити інтерфейс, який буде одночасно інформативним та естетично приємним.
- **Універсальність:** Недієгетичні інтерфейси широко використовуються в більшості ігор різних жанрів. Це свідчить про їх ефективність та універсальність в забезпеченні зручного та інформативного інтерфейсу для гравців.

На основі проведеного аналізу було зроблено висновок, що недієгетичний інтерфейс є найкращим вибором для даної гри. Він забезпечить практичний, інформативний та естетично приємний інтерфейс, який не завадить гравцеві зануритися в ігровий процес.

2.3.2 Основи дизайну інтерфейсу

Увесь інтерфейс гри реалізовано у вікні об'єкта Canvas. Це вікно можна розділити на три основні частини: кнопки меню, кнопки під час гри та відображення балів.

- Кнопки меню: Ця частина містить кнопки для управління грою поза активними ігровими раундами, такі як кнопка "Почати гру", "Вихід" або "Налаштування". Вони дозволяють гравцеві керувати налаштуваннями гри, розпочати нову гру або вийти з неї.
- Кнопки під час гри: Ця частина включає в себе різноманітні кнопки та елементи керування, які стають доступними під час активного ігрового процесу. Це можуть бути кнопки для керування персонажем, відкриття інвентарю або взаємодія з ігровими об'єктами.
- Бали: Ця частина відображає кількість балів, набраних гравцем під час гри. Вона може бути розташована у верхній частині екрана або в будь-якому іншому зручному місці інтерфейсу. Відображення балів допомагає гравцеві відслідковувати свій прогрес та досягнення в грі.

Кнопки меню:

```
public class ButtonMenu : MonoBehaviour
{
    public void StartButton()
    {
        SceneManager.LoadScene(1);
    }

    public void ExitButton()
    {
        Application.Quit();
    }
}
```

StartButton(): Цей метод викликається при натисканні на кнопку "Почати гру" у меню. Внутрішній код цього методу використовує SceneManager.LoadScene(1) для завантаження сцени з індексом 1. Це означає, що при натисканні на цю кнопку буде завантажено другу сцену гри.

ExitButton(): Цей метод викликається при натисканні на кнопку "Вихід" у меню. Внутрішній код цього методу використовує Application.Quit() для виходу з додатка. Це припиняє виконання програми та закриває її вікно.

Кнопки під час гри:

```
public class Scenes : MonoBehaviour
{
    public GameObject[] pigs; // Масив об'єктів із тегом "pig"
    public Button button; // Посилання на кнопку, яку ми хочемо керувати видимістю

    // Start is called before the first frame update
    public void NextLevel(int _sceneNumber)
    {
        SceneManager.LoadScene(_sceneNumber);
    }

    // Update is called once per frame
    void Update()
    {
        // Перевірка, чи є ще на сцені об'єкти з тегом "pig"
        pigs = GameObject.FindGameObjectsWithTag("Pig");

        // Якщо на сцені немає об'єктів з тегом "pig", зробити кнопку видимою
        if (pigs.Length == 0)
        {
            button.gameObject.SetActive(true);
        }
        else
        {
            // В іншому випадку, зробити кнопку невидимою
            button.gameObject.SetActive(false);
        }
    }
}
```

```
// Метод для рестарту сцени
public void RestartScene()
{
    // Отримуємо індекс поточної сцени
    int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;

    // Перезавантажуємо поточну сцену
    SceneManager.LoadScene(currentSceneIndex);
}
}
```

- Цей код відповідає за керування переходами між рівнями гри та відображенням кнопки "Наступний рівень" в залежності від наявності свиней на поточному рівні: pigs: це масив об'єктів, які мають тег "Pig". Він використовується для знаходження усіх свиней на сцені.

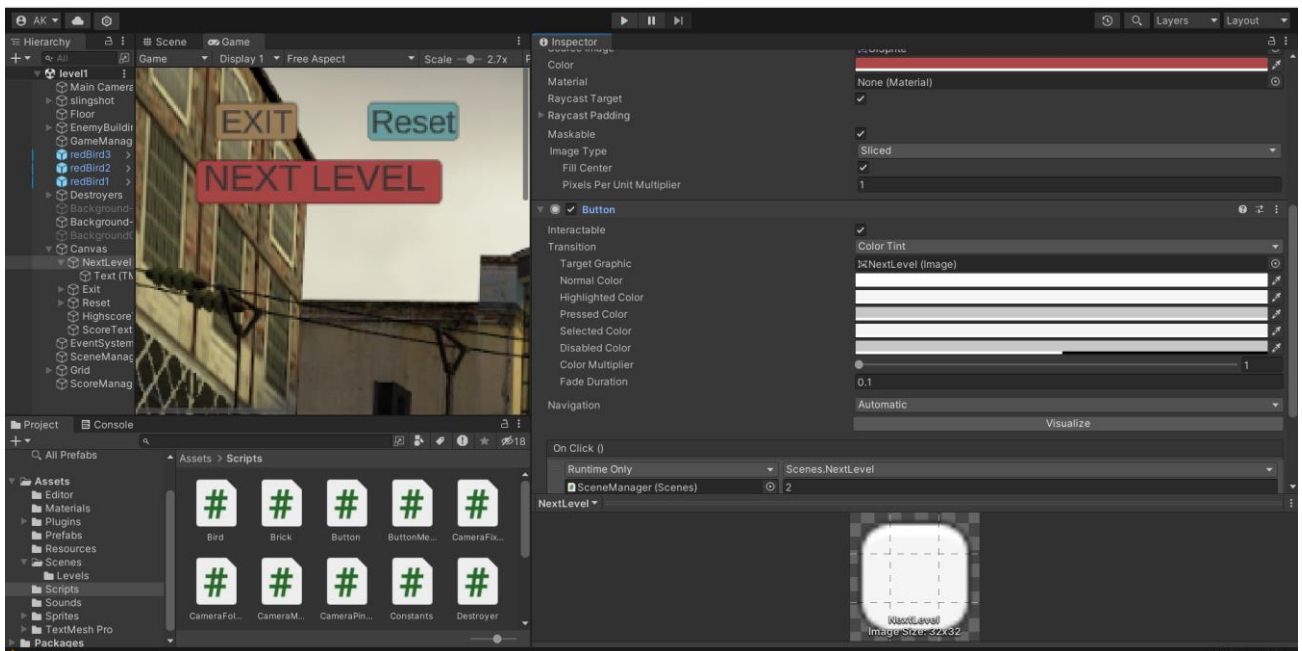


Рис.2.2. Приклад налаштування кнопок інтерфейсу

- button: Це посилання на кнопку, яку потрібно керувати видимістю.

- **NextLevel(int _sceneNumber):** Цей метод викликається для переходу на наступний рівень гри. Він використовує `SceneManager.LoadScene(_sceneNumber)` для завантаження нової сцени з вказаним номером.
- **Update():** Цей метод викликається кожен кадр. Він перевіряє, чи залишилися ще свині на сцені. Якщо кількість свиней на сцені дорівнює нулю, кнопка "Наступний рівень" стає видимою. В іншому випадку, кнопка залишається невидимою.
- **RestartScene():** Цей метод викликається для перезавантаження поточної сцени. Він отримує індекс поточної сцени за допомогою `SceneManager.GetActiveScene().buildIndex`, а потім завантажує сцену з цим самим індексом, щоб відновити гру з початку.

Бали:

```
public class ScoreManager : MonoBehaviour
{
    public Text scoreText;
    public Text highscoreText;
    private int score = 0;
    private int highscore = 0;
    void Start()
    {
        highscore = PlayerPrefs.GetInt("Highscore", 0);
        highscoreText.text = "Highscore: " + highscore;

        scoreText.text = "Score: " + score;
    }
    public void AddScore(int points)
    {
        score += points;
        scoreText.text = "Score: " + score;
    }
}
```

```

        if (score > highscore)
        {
            highscore = score;
            highscoreText.text = "Highscore: " + highscore;
            PlayerPrefs.SetInt("Highscore", highscore);
        }
    }

    public void ResetScore()
    {
        score = 0;
        scoreText.text = "Score: " + score;
    }
}

```

- **scoreText** та **highscoreText**: Посилання на текстові елементи, які відображають поточний рахунок та високий рахунок.
- **score** та **highscore**: Змінні для зберігання поточного та високого рахунків.
- **Start()**: Цей метод викликається при запуску скрипта. Він завантажує високий рахунок збережений у **PlayerPrefs** і встановлює його текстове значення. Також встановлює текстове значення для поточного рахунку.
- **AddScore(int points)**: Цей метод додає очки до поточного рахунку. Очки передаються у параметрі **points**. Він оновлює значення поточного рахунку на екрані та, якщо новий рахунок більший за попередній високий рахунок, оновлює значення високого рахунку та зберігає його в **PlayerPrefs**.
- **ResetScore()**: Цей метод скидає поточний рахунок на нуль. Він встановлює значення **score** на 0 та оновлює відображення поточного рахунку на екрані.

До того ж, механіка балів реалізована у об'єкті **Pigs**. **private ScoreManager scoreManager** відповідає за посилання на **ScoreManager**. **scoreManager.AddScore(points)** додає бали при знищенні свині.

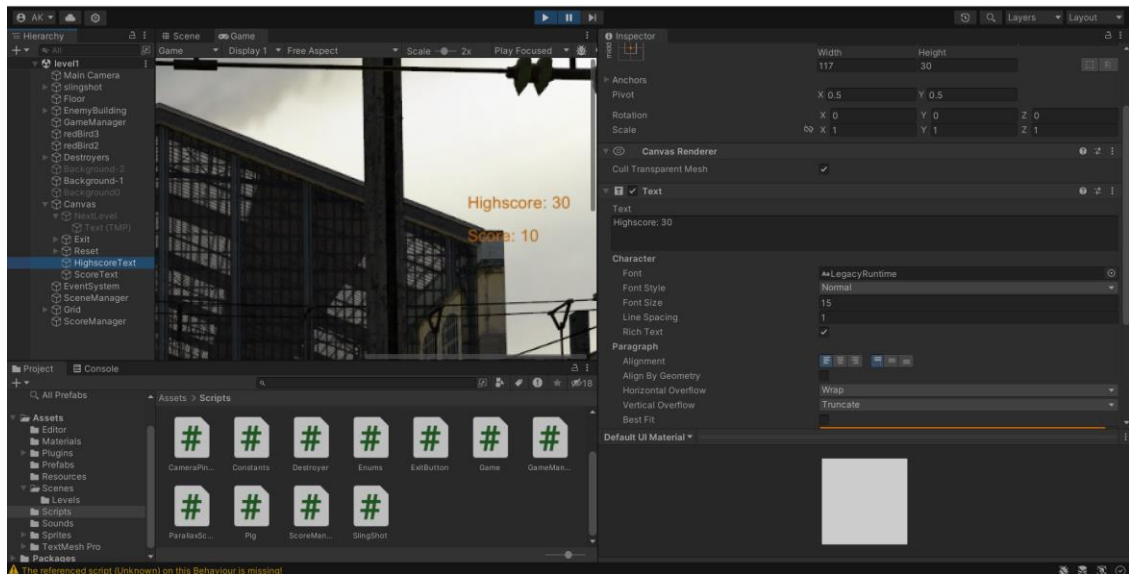


Рис.2.3. “Highscore” та “Score”, що змінюються у момент тестування сцени

2.4. Оптимізація та тестування

У рамках оптимізації продуктивності гри були внесені наступні зміни:

- Використання маловагових текстур: Замість великих та важких за розміром текстур були використані текстури низької роздільної здатності та оптимізовані формати файлів. Це сприяло зменшенню обсягу пам'яті, використовуюваного для зберігання графічних ресурсів гри, та покращило завантаження гри на різних пристроях.
- Ефективне управління об'єктами на сцені: Для зниження навантаження на систему в момент гри було впроваджено ефекти зникнення об'єктів з сцени. Це означає, що птахи після приземлення та свині після їх знищення відображають анімацію зникнення, що допомагає зменшити кількість активних об'єктів на сцені.
- Використання тайлмепів для побудови сцени: Для оптимізації продуктивності гри було використано тайлмапи для побудови об'єктів на

сцені. Це дозволило скоротити кількість окремих об'єктів, представлених на сцені, і зменшити навантаження на систему.

- Ці зміни допомогли покращити продуктивність гри та забезпечити плавний та безперебійний геймплей для гравців.

Висновки до розділу 2

У розділі 2 ми зосередили увагу на кількох ключових аспектах розробки та оптимізації гри, включаючи аналіз коду, налаштування графічних елементів, додавання звукових ефектів та методи оптимізації продуктивності. Нижче наведені основні висновки цього розділу:

Механіка рогатки:

- **Slingshot:** Була реалізована механіка рогатки, що дозволяє гравцеві запускати птахів у напрямку свиней. Рогатка використовує фізику натягнення та відпускання для обчислення траєкторії польоту птаха. Це було реалізовано за допомогою методу `OnThrow()`, який активує звук, слід польоту та змінює властивості птаха для запуску.
- **Аналіз та налаштування ігрових об'єктів:**
- **Bird:** Ми детально розглянули мету, який відповідає за запуск звукового ефекту при кидку птаха. Також було вивчено механізм знищення птаха після зниження його швидкості до мінімуму.
- **Pig:** Аналіз показав, що об'єкт свині має змінний спрайт, який змінюється при зменшенні рівня здоров'я. Було застосовано методи, що відповідають за затримку перед видаленням свині з сцени, а також за нарахування очок гравцеві.
- **Налаштування графічних елементів:**
- **Усі графічні елементи гри виконані у 2D-стилі.** Було надано спрайти для різних об'єктів, таких як птахи, свині та будівлі. Особливу увагу було приділено реалістичному відображенню руйнування дерев'яних елементів та створенню текстур фундаменту для додання ефекту непробивності.

Додавання та управління звуковими ефектами:

- Було реалізовано відтворення звукових ефектів у відповідні моменти гри, зокрема при кидку птаха та знищенні свині. Ці звукові ефекти додали гри динаміки та зробили ігровий процес більш захопливим та реалістичним.
- Методи оптимізації продуктивності гри:
- Зменшення розміру текстур було важливим кроком для покращення продуктивності гри. Оптимізація графічних ресурсів дозволила скоротити час завантаження гри та зменшити використання пам'яті.
- Управління об'єктами на сцені також відіграло ключову роль у підвищенні ефективності гри. Ісчезнення об'єктів, таких як птахи після приземлення та свині після знищення, зменшило кількість активних елементів на сцені, що значно поліпшило продуктивність.
- Використання тайлмепів для побудови ігрових рівнів дозволило оптимізувати процес відображення графіки та зменшити навантаження на систему.

Підсумовуючи, реалізація зазначених заходів з оптимізації та налаштування значно покращила загальну продуктивність гри, забезпечивши плавний ігровий процес та високу якість графіки. Ці зміни сприяли створенню більш захопливого та динамічного ігрового досвіду, що підвищило задоволення гравців та привабливість гри.

ВИСНОВКИ

Ця робота охопила всі етапи розробки мобільної гри на платформі Android, від теоретичних основ до практичної реалізації та оптимізації. Перший розділ надав глибоке розуміння основ розробки, вибору інструментів та технологій, а другий розділ зосередився на практичних аспектах створення гри, включаючи налаштування графіки, звуку та оптимізацію продуктивності. Завдяки системному підходу до розробки та використанню передових інструментів, таких як Unity, вдалося створити високоякісний ігровий продукт, здатний конкурувати на ринку мобільних ігор.

В початковому етапі досягнуто розуміння теоретичних основ:

- Було проведено детальний аналіз теоретичних аспектів розробки мобільних ігор на платформі Android. Це включало вивчення структури платформи Android, інструментів для розробки та основних принципів програмування.
- Проведено аналіз популярних ігор, таких як Subway Surfers, Angry Birds і Doodle Jump, що дозволило визначити ключові фактори їх успіху, зокрема, захоплюючий геймплей, просте управління та яскравий візуальний стиль.
- Було обрано та обґрунтовано використання таких інструментів, як Android Studio, Firebase та Unity. Вибір Unity як основного рушія для розробки гри дозволив скористатися його універсальністю, кросплатформеністю та підтримкою як 2D, так і 3D графіки.
- Практичний момент дозволив отримати досвід розробки багатофункціонального проєкту:
- Використовуючи Unity, вдалося реалізувати всі етапи розробки гри, включаючи створення графічних елементів, налаштування механік гри, додавання звукових ефектів та оптимізацію продуктивності. Було

розроблено інтерактивний інтерфейс користувача, використовуючи принципи матеріального дизайну (Material Design) від Google.

- Успішно реалізовано ключові механіки гри, такі як рогатка для запуску птахів, управління об'єктами на сцені та налаштування поведінки ігрових об'єктів. Це включало використання фізичних обчислень для забезпечення реалістичної поведінки об'єктів у грі.
- Проведено оптимізацію графічних ресурсів, що дозволило зменшити час завантаження гри та використання пам'яті. Це включало зменшення розміру текстур та управління об'єктами на сцені для підвищення ефективності гри.
- Регулярне тестування гри на різних пристроях та версіях Android допомогло виявити та виправити помилки, забезпечуючи стабільну та надійну роботу додатку.
- Розробка інтуїтивно зрозумілого та привабливого інтерфейсу користувача, використовуючи матеріальний дизайн, зробила гру зручною та привабливою для користувачів.
- Завдяки ретельному проектуванню, розробці та оптимізації, вдалося створити високоякісний ігровий продукт, який може конкурувати на сучасному ринку мобільних ігор. Гра забезпечує захоплюючий та динамічний ігровий досвід, що підвищує задоволення гравців та привабливість гри.
- Вдалося успішно поєднати 2 Всесвіти різних ігор, що надає можливість отримати велику кількість завантажень з боку фанатів обох сторін.

Загалом, виконана робота продемонструвала успішну реалізацію всіх етапів розробки мобільної гри на платформі Android. Це включало як теоретичне обґрунтування вибору інструментів та технологій, так і практичну реалізацію гри з налаштуванням графіки, звуку та оптимізацією продуктивності. Отримані

результати свідчать про високий рівень підготовки та можливість створення конкурентоспроможного ігрового продукту на сучасному ринку мобільних ігор.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Java URL: <https://www.oracle.com/cis/java/>
2. Офіційна документація Android для C++ URL: <https://developer.android.com/ndk/guides>
3. Підручник з C++ URL: <https://www.learncpp.com/>
4. Про середовище програмування C#. URL: <https://foxminded.ua/seredovyshche-prohramuvannia-si-sharp/>
5. Підручник з C# URL: <https://learn.microsoft.com/en-us/dotnet/csharp/>
6. Android Developers. URL: <https://developer.android.com/guide/platform>
Kernel на Android - офіційна документація URL: <https://source.android.com/docs/core/architecture/kernel?hl=ru>
7. Libraries in Android - офіційна документація URL: https://developer.android.com/ndk/guides/stable_apis
8. Application Framework - офіційна документація URL: <https://developer.android.com/guide/components/activities/intro-activities>
9. Основні компоненти Android. URL: <https://developer.android.com/guide/components/fundamentals>
10. Офіційний сайт Android Studio URL: <https://developer.android.com/studio>
11. Офіційний сайт Firebase URL: <https://firebase.google.com/>
12. What is the Unity Hub? URL: <https://unity.com/ru/unity-hub>
13. Unity's Asset Store URL: <https://docs.unity3d.com/Manual/AssetStore.html>
14. Офіційний сайт Unreal Engine URL: <https://www.unrealengine.com/en-US>
15. Офіційний сайт Cocos2d URL: <https://www.cocos.com/en>

- 16.Розробка мобільних додатків від А до Я: повний гайд URL: :
<https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkov-vid-a-do-ja-povnij-gajd/>
- 17.Subway Surfers на Google Play. URL:
<https://play.google.com/store/apps/details/Subway+Surfers?id=com.kiloo.subwaysurf&hl=uk&gl=US>
- 18.Angry Birds 2. URL: <https://www.angrybirds.com/games/angry-birds-2/>
- 19.How Angry Birds Took Flight URL:
<https://www.ign.com/articles/2010/09/06/how-angry-birds-took-flight>
- 20.Doodle Jump на Google Play. URL:
<https://play.google.com/store/apps/details?id=com.lima.doodlejump&hl=ru>
- 21.Half-Life 2 URL: https://store.steampowered.com/app/220/HalfLife_2/
- 22.Unity Documentation. URL: <https://docs.unity.com/>
- 23.Уроки з Unity: встановлення рушія, налаштування інтерфейсу та робота з об'єктами URL: <https://www.tinkerteens.com/blog-post/uroky-z-unity-vstanovlennia-rushiia-nalashtuvannia-interfeisu-ta-robota-z-obiektamy/>

ДОДАТОК А

Вихідний код скриптів

Вихідний код скриптів, які організовують роботу програми – SlingShot, Bird, Pig, ScoreManager.

```
using UnityEngine;
using System.Collections;
using Assets.Scripts;
using System;

public class SlingShot : MonoBehaviour
{
    private Vector3 SlingshotMiddleVector;

    [HideInInspector]
    public SlingshotState slingshotState;

    public Transform LeftSlingshotOrigin, RightSlingshotOrigin;

    public LineRenderer SlingshotLineRenderer1;
    public LineRenderer SlingshotLineRenderer2;

    public LineRenderer TrajectoryLineRenderer;

    [HideInInspector]
    public GameObject BirdToThrow;

    public Transform BirdWaitPosition;

    public float ThrowSpeed;

    [HideInInspector]
    public float TimeSinceThrown;

    void Start()
    {
        SlingshotLineRenderer1.sortingLayerName = "Foreground";
        SlingshotLineRenderer2.sortingLayerName = "Foreground";
        TrajectoryLineRenderer.sortingLayerName = "Foreground";

        slingshotState = SlingshotState.Idle;
        SlingshotLineRenderer1.SetPosition(0, LeftSlingshotOrigin.position);
        SlingshotLineRenderer2.SetPosition(0, RightSlingshotOrigin.position);

        // Направлення на середню позицію двох векторів
```

```

        SlingshotMiddleVector = new Vector3((LeftSlingshotOrigin.position.x +
RightSlingshotOrigin.position.x) / 2,
        (LeftSlingshotOrigin.position.y + RightSlingshotOrigin.position.y) / 2, 0);
    }

    // Оновлюється кожен кадр
    void Update()
    {
        switch (slingshotState)
        {
            case SlingshotState.Idle:
                // Фіксація позиції птаха
                InitializeBird();
                // Відображення "струн" рогатки
                DisplaySlingshotLineRenderers();
                if (Input.GetMouseButtonDown(0))
                {
                    // Отримання точки на екрані, на яку натиснув користувач
                    Vector3 location = Camera.main.ScreenToWorldPoint(Input.mousePosition);
                    // Якщо користувач натиснув на птаха
                    if (BirdToThrow.GetComponent<CircleCollider2D>() ==
Physics2D.OverlapPoint(location))
                    {
                        slingshotState = SlingshotState.UserPulling;
                    }
                }
                break;
            case SlingshotState.UserPulling:
                DisplaySlingshotLineRenderers();

                if (Input.GetMouseButton(0))
                {
                    // Отримання позиції, на яку натиснув користувач
                    Vector3 location = Camera.main.ScreenToWorldPoint(Input.mousePosition);
                    location.z = 0;
                    // Дозвіл користувачу тягти птаха до максимальної відстані
                    if (Vector3.Distance(location, SlingshotMiddleVector) > 1.5f)
                    {
                        // Простий математичний розрахунок :)
                        var maxPosition = (location - SlingshotMiddleVector).normalized * 1.5f +
SlingshotMiddleVector;
                        BirdToThrow.transform.position = maxPosition;
                    }
                    else
                    {
                        BirdToThrow.transform.position = location;
                    }
                }
            }
        }
    }

```

```

        float distance = Vector3.Distance(SlingshotMiddleVector,
BirdToThrow.transform.position);
        // Відображення прогнозованої траєкторії на основі відстані
        DisplayTrajectoryLineRenderer2(distance);
    }
    else// Користувач відірвав торкання
    {
        SetTrajectoryLineRenderersActive(false);
        // Кидок птаха!!!
        TimeSinceThrown = Time.time;
        float distance = Vector3.Distance(SlingshotMiddleVector,
BirdToThrow.transform.position);
        if (distance > 1)
        {
            SetSlingshotLineRenderersActive(false);
            slingshotState = SlingshotState.BirdFlying;
            ThrowBird(distance);
        }
        else// Недостатньо довго тягнули, тому реконструюємо
        {
            // Довжина / 10 було визначено методом проб і помилок
            // Анімація птаха до позиції очікування
            BirdToThrow.transform.position = BirdWaitPosition.position;

        }
    }
    break;
case SlingshotState.BirdFlying:
    break;
default:
    break;
}

}

private void ThrowBird(float distance)
{
    // Отримання швидкості
    Vector3 velocity = SlingshotMiddleVector - BirdToThrow.transform.position;
    BirdToThrow.GetComponent<Bird>().OnThrow(); // Повідомлення птаха про кидок
    // Старий та альтернативний спосіб
    // BirdToThrow.GetComponent<Rigidbody2D>().AddForce
    // (new Vector2(v2.x, v2.y) * ThrowSpeed * distance * 300 * Time.deltaTime);
    // Встановлення швидкості
    BirdToThrow.GetComponent<Rigidbody2D>().velocity = new Vector2(velocity.x,
velocity.y) * ThrowSpeed * distance;
}

```

```

        // Повідомлення, що птаха кинули
        if (BirdThrown != null)
            BirdThrown(this, EventArgs.Empty);
    }

    public event EventHandler BirdThrown;

    private void InitializeBird()
    {
        // Ініціалізація птаха готового до кидка
        BirdToThrow.transform.position = BirdWaitPosition.position;
        slingshotState = SlingshotState.Idle;
        SetSlingshotLineRenderersActive(true);
    }

    void DisplaySlingshotLineRenderers()
    {
        SlingshotLineRenderer1.SetPosition(1, BirdToThrow.transform.position);
        SlingshotLineRenderer2.SetPosition(1, BirdToThrow.transform.position);
    }

    void SetSlingshotLineRenderersActive(bool active)
    {
        SlingshotLineRenderer1.enabled = active;
        SlingshotLineRenderer2.enabled = active;
    }

    void SetTrajectoryLineRenderesActive(bool active)
    {
        TrajectoryLineRenderer.enabled = active;
    }

    /// <summary>
    /// Інший варіант (чудовий) можна знайти тут
    /// http://wiki.unity3d.com/index.php?title=Trajectory\_Simulation
    /// </summary>
    /// <param name="distance"></param>
    void DisplayTrajectoryLineRenderer2(float distance)
    {
        SetTrajectoryLineRenderesActive(true);
        Vector3 v2 = SlingshotMiddleVector - BirdToThrow.transform.position;
        int segmentCount = 15;
        float segmentScale = 2;
        Vector2[] segments = new Vector2[segmentCount];

        // Перша точка лінії - там, де знаходиться гармата гравця, тощо

```

```

segments[0] = BirdToThrow.transform.position;

// Початкова швидкість
Vector2 segVelocity = new Vector2(v2.x, v2.y) * ThrowSpeed * distance;

float angle = Vector2.Angle(segVelocity, new Vector2(1, 0));
float time = segmentScale / segVelocity.magnitude;
for (int i = 1; i < segmentCount; i++)
{
    // Виведення координат
    float time2 = i * Time.fixedDeltaTime * 5;
    segments[i] = segments[0] + segVelocity * time2 + 0.5f * Physics2D.gravity *
Mathf.Pow(time2, 2);
}

TrajectoryLineRenderer.SetVertexCount(segmentCount);
for (int i = 0; i < segmentCount; i++)
    TrajectoryLineRenderer.SetPosition(i, segments[i]);
}

}

using UnityEngine;
using System.Collections;
using Assets.Scripts;

[RequireComponent(typeof(Rigidbody2D))] // Вимагає компонент Rigidbody2D для цього
класу
public class Bird : MonoBehaviour
{
    void Start()
    {
        //trailrenderer не буде видно, поки ми не кинемо птаха
        GetComponent<TrailRenderer>().enabled = false; // Вимикає TrailRenderer до
моменту кидання птаха
        GetComponent<TrailRenderer>().sortingLayerName = "Foreground"; // Встановлює
шар сортування для TrailRenderer
        //спочатку без гравітації
        GetComponent<Rigidbody2D>().isKinematic = true; // Вимикає гравітацію,
встановлюючи isKinematic в true
        //збільшуємо колайдер, щоб полегшити торкання
        GetComponent<CircleCollider2D>().radius = Constants.BirdColliderRadiusBig; //
Збільшує радіус колайдера для полегшення торкання птаха
        State = BirdState.BeforeThrown; // Встановлює початковий стан птаха

```

```

    }

    void FixedUpdate()
    {
        //якщо ми кинули птаха
        //і його швидкість дуже мала
        if (State == BirdState.Thrown &&
            GetComponent<Rigidbody2D>().velocity.sqrMagnitude <= Constants.MinVelocity)
        {
            //знищуємо птаха через 2 секунди
            StartCoroutine(DestroyAfter(2)); // Знищує птаха після 2 секунд, якщо його
швидкість дуже мала
        }
    }

    public void OnThrow()
    {
        //відтворюємо звук
        GetComponent<AudioSource>().Play(); // Відтворює звук кидання птаха
        //показуємо слід
        GetComponent<TrailRenderer>().enabled = true; // Вмикає TrailRenderer
        //дозволяємо гравітаційні сили
        GetComponent<Rigidbody2D>().isKinematic = false; // Увімкнює гравітацію,
встановлюючи isKinematic в false
        //встановлюємо нормальний розмір колайдера
        GetComponent<CircleCollider2D>().radius = Constants.BirdColliderRadiusNormal; //
Змінює розмір колайдера на нормальний
        State = BirdState.Thrown; // Встановлює стан "кинуто"
    }

    IEnumerator DestroyAfter(float seconds)
    {
        yield return new WaitForSeconds(seconds); // Затримує виконання на певну кількість
секунд
        Destroy(gameObject); // Знищує об'єкт
    }

    public BirdState State
    {
        get;
        private set;
    }
}

```

```

using UnityEngine;
using System.Collections;

public class Pig : MonoBehaviour
{
    // Поля
    public float Health = 150f; // Здоров'я свині
    public Sprite SpriteShownWhenHurt; // Спрайт, який відображається при пораненні
    private float ChangeSpriteHealth; // Здоров'я, при якому спрайт змінюється
    private AudioSource audioSource; // Аудіо джерело
    public int points = 10; // Кількість очок за знищення свині
    private ScoreManager scoreManager; // Посилання на ScoreManager

    // Викликається при старті
    void Start()
    {
        // Визначення здоров'я, при якому спрайт змінюється
        ChangeSpriteHealth = Health - 30f;
        audioSource = GetComponent<AudioSource>();
        scoreManager = FindObjectOfType<ScoreManager>(); // Знайти ScoreManager на
сцені
    }

    // Обробник зіткнень
    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.GetComponent<Rigidbody2D>() == null) return;

        // Якщо свиню вдарив птах
        if (col.gameObject.tag == "Bird")
        {
            // Відтворення звуку
            //GetComponent<AudioSource>().Play();
            // Видалення свині
            StartCoroutine(DestroyPigWithDelay());
        }
        else // Якщо її вдарило інше
        {
            // Розрахунок пошкодження
            float damage = col.gameObject.GetComponent<Rigidbody2D>().velocity.magnitude
* 10;

            // Зменшення здоров'я
            Health -= damage;
            // Відтворення звуку при великому пошкодженні
            //if (damage >= 10)
            //    GetComponent<AudioSource>().Play();

            // Зміна спрайта при досягненні певного рівня здоров'я

```

```

        if (Health < ChangeSpriteHealth)
        {
            GetComponent<SpriteRenderer>().sprite = SpriteShownWhenHurt;
        }
        // Видалення свині при смерті
        if (Health <= 0)
        {

            StartCoroutine(DestroyPigWithDelay());

        }
    }
}

// Корутина для затримки перед видаленням свині
private IEnumerator DestroyPigWithDelay()
{
    GetComponent<AudioSource>().Play();
    // Затримка на 2 секунди
    yield return new WaitForSeconds(2);
    scoreManager.AddScore(points); // Додати очки при знищенні свині
    // Видалення свині
    Destroy(gameObject);
}
}

using UnityEngine;
using UnityEngine.UI;

public class ScoreManager : MonoBehaviour
{
    public Text scoreText; // Посилання на текстовий елемент для поточного рахунку
    public Text highscoreText; // Посилання на текстовий елемент для високого рахунку

    private int score = 0; // Поточний рахунок
    private int highscore = 0; // Високий рахунок

    void Start()
    {
        // Завантажити високий рахунок з PlayerPrefs
        highscore = PlayerPrefs.GetInt("Highscore", 0);
        highscoreText.text = "Highscore: " + highscore;

        // Оновити текст поточного рахунку
        scoreText.text = "Score: " + score;
    }
}

```

```
// Метод для збільшення рахунку
public void AddScore(int points)
{
    score += points;
    scoreText.text = "Score: " + score;

    // Оновити високий рахунок, якщо поточний рахунок більший
    if (score > highscore)
    {
        highscore = score;
        highscoreText.text = "Highscore: " + highscore;

        // Зберегти новий високий рахунок
        PlayerPrefs.SetInt("Highscore", highscore);
    }
}

// Метод для скидання рахунку (наприклад, при рестарті гри)
public void ResetScore()
{
    score = 0;
    scoreText.text = "Score: " + score;
}
}
```