МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ТРИВИМІРНОЇ ГРИ «РАННЕР» З МОЖЛИВІСТЮ МОНЕТИЗАЦІЇ»**

Виконав: студент 2 курсу, групи 8.1223
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Я. О. Падалко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.–м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" 05 " березня 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Падалку Ярославу Олександровичу

1. Тема роботи Розробка тривимірної гри "Раннер" з можливістю монетизації

керівник роботи Матвіїшина Надія Вікторівна, к.т.н., доцент
(прізвище, ім'я та по–батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 384-с

2. Строк подання студентом роботи 25.11.2024

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити)

1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка гри.
4. Налаштування та тестування гри.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 04.03.2024

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етпів кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1.	Розробка плану роботи	10.03.2024	
2.	Збір вихідних даних	20.05.2023	
3.	Обробка методичних та	15.06.2023	
	теоретичних джерел		
4.	Розробка першого та другого розділів	12.08.2023	
5.	Розробка третього розділу	16.11.2023	
6.	Оформлення та нормокнтроль	20.11.2024	
	кваліфікаційної роботи бакалавра		
7.	Захист кваліфікаційної роботи	12.12.2024	

Студент _____ Я.О. Падалко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.В. Матвіїшина
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка тривимірної гри «Раннер» з можливістю монетизації»: 48 с., 36 рис., 10 джерел.

ГРА, РАНЕР, СЦЕНАРІЙ ГРИ, UNITY, C#, 3D моделювання.

Об’єкт дослідження – процес розробки гри, що охоплює управління проєктом, створення концепції ігрового світу, впровадження механік монетизації та налагодження взаємодії з користувачами.

Мета роботи: розробка 3D гри жанру раннер у середовищі Unity з впровадженням елементів монетизації.

Метод дослідження – аналіз, описовий, порівняльний.

В рамках кваліфікаційної роботи була розроблена гра жанру раннер у популярному середовищі розробці комп'ютерних ігор Unity. Було створено деталізований ігровий світ, використання елементів монетизації, а також різноманітними інтерактивними елементами, такими як персонаж, перешкоди та бонуси. Для забезпечення динамічної взаємодії з цими елементами були розроблені спеціальні скрипти на мові програмування C#.

SUMMARY

Master's Qualification Thesis "Development of a three-dimensional game "Runner" with the possibility of monetization": 48 p., 36 figures, 10 references.

GAME, RUNNER, GAME SCRIPT, UNITY, C#, 3D modeling.

Purpose of the work: development of a 3D runner game in the Unity environment with the introduction of monetization elements.

The research method is analysis, descriptive, comparative.

As part of the qualification work, a runner game was developed in the popular Unity computer game development environment. A detailed game world was created, using monetization elements, as well as various interactive elements, such as a character, obstacles and bonuses. To ensure dynamic interaction with these elements, special scripts were developed in the C# programming language.

ЗМІСТ

Завдання на кваліфікаційну роботу …… ..2
Реферат 4
Summary 5
Вступ 7
1 Створення комп’ютерних ігор 8
1.1 Ігрова індустрія в реаліях сьогодення 8
1.2 Ігрова механіка гри у стилі раннер 9
2 Процес створення гри 12
2.1 Особливості рушія Unity 12
2.1.1 Створення ігрового оточення та управління персонажем 12
2.1.2 Створення перешкод та системи програшу 14
2.1.3 Реалізація головного героя 17
2.1.4 Розробка та застосування текстур в ігровому проекті 21
2.1.5 Створення головного меню і системи рекорду 23
2.1.6 Система бонусів 29
3 Монетизація у іграх 34
3.1 Різновиди монетизації у іграх 35
3.2 Реалізація монетизації у проекті 40
Висновки 47
Перелік посилань 48

ВСТУП

Сучасні технології відкривають нові можливості для створення захоплюючих ігрових світів. Мета робота присвячена розробці тривимірної гри "раннер", яка використовує останні досягнення в галузі 3D–графіки,та геймдизайну. Особлива увага приділена системі монетизації, що дозволить створити унікальний ігровий досвід для користувачів.

Unity – це кросплатформне середовище розробки програмного забезпечення, яке використовується для створення ігор та інтерактивних додатків. Він дозволяє розробникам втілювати свої ідеї в життя, створюючи 2D та 3D ігри для різних платформ, включаючи ПК, мобільні пристрої, консолі та навіть пристрої віртуальної реальності.

Unity це потужний і гнучкий інструмент для створення ігор і програм, який підходить як для розробників–початківців, так і для досвідчених професіоналів. Якщо створювати свої власні ігри або програми, Unity найкращій вибір.

Жанр "раннер" є класичним прикладом того, як прості механіки можуть забезпечити годинник захоплюючого геймплею. Ця кваліфікаційна робота поєднує теоретичні знання про розробку ігор з практичним досвідом створення 3D–раннера на платформі Unity. Будуть розглянуті питання дизайну гри, оптимізації гри та монетизації. Результатом роботи стане функціональна гра, яка продемонструє глибоке розуміння принципів геймдизайну та програмної інженерії.

1 СТВОРЕННЯ КОМП'ЮТЕРНИХ ІГОР

1.1 Ігрова індустрія в реаліях сьогодення

Ігрова індустрія – це один із найбільш динамічних і перспективних секторів сучасної економіки, який впливає не лише на розваги, а й на культуру, технології, соціальні комунікації та навіть освіту. Її розвиток є відображенням загального прогресу цифрових технологій, змін у сприйнятті розваг та зростаючого попиту на інтерактивний контент[4].

По–перше, що хочеться відмітити це технологічний прорив. Сьогодні ігрова індустрія використовує передові технології, зокрема:

- доповнена та віртуальна реальність (AR і VR), які дозволяють гравцям зануритися у максимально реалістичний світ. Це знаходить застосування не лише в розвагах, а й у медицині, навчанні та військовій підготовці;
- штучний інтелект (AI) – для створення складніших персонажів, адаптивних сюжетів і персоналізованого ігрового досвіду;
- хмарні технології – ігри більше не обмежуються платформою. Гравці можуть отримувати доступ до високоякісних ігор через стрімінгові сервіси, зокрема Google Stadia, Xbox Cloud Gaming та інші.

По–друге, це економічний фактор який зростає кожен рік. Ігрова індустрія перетворилася на багатомільярдний бізнес. У 2023 році обсяги світового ринку відеоігор перевищили 200 мільярдів доларів, що ставить її попереду кіноіндустрії та музичного бізнесу. Зростає популярність моделей монетизації, таких як:

- free–to–play (F2P), де основний дохід отримується від мікротранзакцій;

– кіберспорт, який став окремою галуззю, привертаючи величезні інвестиції, мільйони глядачів і спонсорів. У 2025 році відбудуться перші олімпійські ігри в Саудівській Аравії.

По–третє, це культурний і соціальний вплив індустрії. Ігри вже давно перестали бути просто розвагою. Культурний феномен який вони створили, формують субкультури які впливають на мистецтво, моду та кінематограф. Також важливий аспект це соціальні комунікації. Онлайн–ігри створюють спільноти, де люди з різних куточків світу взаємодіють, розвиваючи командний дух і соціальні навички. Один із найважливіших моментів це проблеми залежності. Одночасно з перевагами, ігри стають джерелом соціальної ізоляції, особливо серед молоді, та викликають обговорення питання ігрової залежності.

Також факт використання ігор у освіта і тренування. Ігри дедалі частіше застосовуються як навчальний інструмент. Гейміфікація використовується у школах та університетах для підвищення мотивації, а також у професійних сферах для тренувань персоналу. Наприклад, симулятори для пілотів чи хірургів базуються на принципах ігрових механік.

Ігрова індустрія в сучасному світі – це не просто про ігри, а про глобальний вплив на різні аспекти життя. Вона продовжує розвиватися з величезною швидкістю, інтегруючи новітні технології, змінюючи уявлення про розваги та створюючи нові можливості для економіки та суспільства.

1.2 Ігрова механіка гри у стилі раннер

Сценарій гри у стилі Ранер визначає ключові елементи ігрового процесу, такі як механіка, об'єкти, їхня поведінка та умови, що впливають на перебіг гри. Основний принцип такої гри полягає в тому, що гравець керує персонажем, який постійно біжить вперед, уникаючи перешкод. У сценарії необхідно описати, які об'єкти будуть виступати як перешкоди, які можна

буде збирати, а також які бонуси та нагороди гравець отримуватиме за досягнення певних результатів. Сценарій також визначає способи керування персонажем: зазвичай це сенсорний екран мобільного пристрою або клавіатура на комп'ютері. У ньому можуть бути передбачені різні рівні складності та ігрові режими, наприклад, "виживання" чи "на час". Графіка та звукові ефекти також є важливими елементами сценарію гри у стилі Ранер. Сценарій повинен визначати, які графічні елементи будуть використовуватися для персонажа, перешкод та бонусів, а також які звукові ефекти використовуватимуться для створення атмосфери гри[5].

Також у сценарії гри потрібно визначити різні умови, які впливають на ігровий процес, наприклад швидкість руху персонажа, частоту появи перешкод, кількість життів, час, за який потрібно пройти рівень, та інші параметри, які можуть змінюватись в залежності від рівня складності гри. Всі ці параметри та умови повинні бути налаштовані таким чином, щоб створити цікавий та захоплюючий ігровий процес, який викликатиме бажання гравця продовжувати грати та покращувати свої результати.

Сценарій гри в стилі Ранер є важливою складовою процесу розробки. Як приклад, розглянемо гру Subway Surfers – популярний мобільний Ранер, створений компанією Kiloo. У цій грі гравці управляють персонажем на ім'я Джейк, який біжить по залізничних коліях, ухиляється від поїздів і перешкод, збирає монети та бонуси. Гра доступна на різних платформах, включаючи iOS, Android і Windows.

Приклад сценарію для *Subway Surfers*:

- початок гри: Гравець стартує на залізничній станції, керуючи персонажем Джейком, який біжить по рейках;
- основні дії: Гравець повинен уникати поїздів, стрибати через перешкоди та збирати монети;
- тривалість гри: Гра триває, доки персонаж не зіткнеться з поїздом або перешкодою;
- система балів: Зібрані монети збільшують рахунок гравця;

- магазин: Монети можна використовувати для придбання нових персонажів, посилень та оновлень, що допомагають продовжити ігровий процес;
- локації: У грі є різноманітні місцевості, такі як міські вулиці чи тунелі метро, кожна з яких має унікальний дизайн і власні перешкоди.
- соціальні елементи: Гравець може змагатися з іншими, здобувати досягнення та підніматися у рейтингах;

Цей сценарій визначає ключові елементи ігрового процесу та механіку, створюючи захопливий досвід для гравців.

2 ПРОЦЕС СТВОРЕННЯ ГРИ

2.1 Особливості рушія Unity

Unity – це кросплатформний ігровий рушій, розроблений компанією Unity Technologies. Він дозволяє створювати ігри для широкого спектру платформ, включаючи комп'ютери, мобільні пристрої, консолі та віртуальну реальність. Підтримка різних мов програмування: Unity підтримує кілька мов програмування, таких як C#, C++, JavaScript та інші. Ще у цій платформі велика спільнота: Unity має величезну спільноту розробників, де можна знайти відповіді на багато питань та підказки щодо розробки ігор.

Unity є одним із найпопулярніших ігрових платформ для створення ігор у світі завдяки своїм широким можливостям і простоті у використанні. Його зрозумілий інтерфейс робить платформу доступною навіть для початківців у розробці ігор. Однією з головних переваг Unity для новачків є використання мови програмування C#. Ця мова вважається однією з найбільш зручних і легких для освоєння, що дозволяє швидко почати створювати власні ігри. Крім того, Unity має потужну спільноту користувачів, яка створила величезну базу навчальних матеріалів. Це значно полегшує процес навчання і допомагає швидко вирішувати труднощі, які можуть виникнути під час розробки.

2.1.1 Створення ігрового оточення та управління персонажем

Цей скрипт забезпечує управління персонажем у грі ранері. Персонаж рухається вперед із постійною швидкістю, а гравець може змінювати його положення між трьома смугами: лівою, середньою та правою.

За допомогою свайпів вліво або вправо гравець змінює поточну смугу, на яку має переміститися персонаж. Позиція персонажа обчислюється залежно від вибраної смуги: якщо це ліва смуга, додається зсув ліворуч; якщо права – зсув праворуч. Початково персонаж знаходиться на середній смузі (рис. 2.1).

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

public class PlayerController : MonoBehaviour
private CharacterController controller;
private Vector3 dir;
[SerializeField] private float speed;

private int lineToMove = 1;
public float lineDistance = 4;

void Start()
{
    controller = GetComponent<CharacterController>();
}
private void Update()
{
    if (SwipeController.swipeRight)
    {
        if (lineToMove < 2)
            lineToMove++;
    }

    if (SwipeController.swipeLeft)
    {
        if (lineToMove > 0)
            lineToMove--;
    }

    Vector3 targetPosition = transform.position.z * transform.forward + transform.position.y * transform.up;
    if (lineToMove == 0)
        targetPosition += Vector3.left * lineDistance;
    else if (lineToMove == 2)
        targetPosition += Vector3.right * lineDistance;
}
void FixedUpdate()
{
    dir.z = speed;
}
```

Рисунок 2.1 – Скріпт Player Controller

Скрипт також задає напрямок руху вперед із фіксованою швидкістю. Для управління фізикою персонажа використовується компонент Character Controller.

Для управління головним героєм був розроблений скрипт Swipe Controller (рис. 2.2). Цей скрипт реалізує систему обробки свайпів (жестів) для управління грою, як на мобільних пристроях, так і на ПК. Він визначає напрямок свайпу: вліво, вправо, вгору або вниз, а також підтримує можливість фіксації простого тапу. Скрипт обробляє жести користувача (свайпи) і конвертує їх у відповідні події (swipeLeft, swipeRight, swipeUp, swipeDown), які можуть бути використані іншими компонентами гри для управління персонажем або об'єктами.

Скрипт працює як універсальний обробник введення для мобільних і десктопних платформ. Логіка підходить для будь–яких ігор, які вимагають

свайпів або жестів для керування. Він підтримує мультиплатформеність: коректно працює і з дотиками, і з натисканням миші.

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

    private void Update()
    {
        tap = swipeDown = swipeUp = swipeLeft = swipeRight = false;
        #region ПК - версия
        if (Input.GetMouseButtonDown(0))
        {
            tap = true;
            isDraging = true;
            startTouch = Input.mousePosition;
        }
        else if (Input.GetMouseButtonUp(0))
        {
            isDraging = false;
            Reset();
        }
        #endregion

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

Рисунок 2.2 – Частина скрипта Swipe Controller

2.1.2 Створення перешкод та системи програшу

У ранерах програш зазвичай настає, коли гравець не долає певний рівень, стикається з перешкодою або не встигає виконати необхідну дію вчасно. Ігри цього жанру відрізняються швидким темпом, що вимагає миттєвої реакції на події на екрані. Невдача може статися через те, що гравець не встиг зробити стрибок, не помітив перешкоди або обрав неправильну дію в критичний момент.

Також причиною програшу можуть бути технічні збої, як–от зависання гри або проблеми з управлінням. Це може бути спричинено помилками апаратного чи програмного забезпечення пристрою, на якому запущена гра. Загалом, програш у ранерах зазвичай пов’язаний із нездатністю гравця вчасно та точно реагувати на динамічні виклики гри.

У цій грі перешкода через яку буде вважатися програш реалізовані в текстурах автомобілів (рис. 2.3). Вони були реалізовані у сервісі Asset Forge. Asset Forge – це інтуїтивно зрозумілий інструмент, який дозволяє створювати

3D–моделі та 2D–спрайти без глибоких знань у 3D–моделюванні. По суті це конструктор, де збирається модель з готових блоків, як з LEGO.

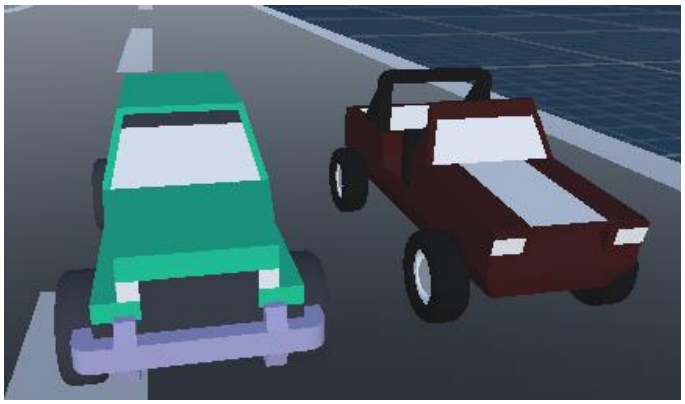


Рисунок 2.3 – Два об’єкти гри

Програш гри реалізований під час зіткнення головного героя з автомобілем. Для них додана колізія об'єкту.

У скрипті Player Controller треба додати деякі фрагменти коду, який відповідає за плавне переміщення об'єкта до заданої цільової позиції (рис. 2.4).

```
if (transform.position == targetPosition)
    return;
Vector3 diff = targetPosition - transform.position;
Vector3 moveDir = diff.normalized * 25 * Time.deltaTime;
if (moveDir.sqrMagnitude < diff.sqrMagnitude)
    controller.Move(moveDir);
else
    controller.Move(diff);
```

Рисунок 2.4– Фрагмент коду

Перелік виконаних дій кодом:

– якщо поточна позиція об'єкта (transform.position) збігається з цільовою позицією (targetPosition), метод просто завершується (return), оскільки переміщення вже не потрібне;

– обчислення вектора різниці.Vector3 diff обчислює відстань між поточною позицією об'єкта та цільовою позицією (targetPosition – transform.position);
– нормалізація та створення вектора руху. moveDir – це напрямок руху об'єкта, нормалізований (тобто з довжиною 1), помножений на швидкість (25) та час, що пройшов із попереднього кадру (Time.deltaTime). Це забезпечує плавність руху незалежно від частоти кадрів;
– перевірка допустимої довжини руху. Якщо довжина (sqrMagnitude) вектора moveDir (довжина передбачуваного руху) менша за довжину diff (відстань до цілі), об'єкт переміщується на цю відстань (controller.Move(moveDir)).Якщо довжина moveDir перевищує відстань до цілі (тобто ми рухаємося далі, ніж потрібно), об'єкт переміщується рівно на відстань до цільової позиції (controller.Move(diff)), щоб точно потрапити в точку.

Для створення системи програшу спочатку потрібно реалізувати систему підрахунку очок. Для цього створено скрипт «Score» (рис 2.5). Призначення скрипта це відстеження очок. Постійно збільшує загальний рахунок гравця. Також відображення рахунку. Оновлює текстовий елемент інтерфейсу (UI) для відображення актуального рахунку.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5
6   public class Score : MonoBehaviour
7   {
8       [SerializeField] private Transform player;
9       [SerializeField] public Text scoreText;
10      private int totalScore;
11
12      public int scoreMultiplier;
13
14
15      private void FixedUpdate()
16      {
17          totalScore += scoreMultiplier;
18          scoreText.text = totalScore.ToString();
19      }
20  }
```

Рисунок 2.5– Скрипт «Score»

Основні елементи:

- player: посилання на об'єкт гравця (хоча в цьому коді воно не використовується, можливо, зарезервовано для майбутнього розширення);
- scoreText: текстовий UI–елемент, у якому відображається рахунок;
- totalScore: цілочислова змінна для збереження загальної кількості набраних очок;
- scoreMultiplier: множник, який визначає, наскільки швидко зростає рахунок.

Метод FixedUpdate викликається на кожному фіксованому кадрі гри (корисно для фізики та регулярних обчислень). Збільшує totalScore, додаючи значення scoreMultiplier. Оновлює текст у scoreText, щоб відобразити поточний рахунок гравця.

Особливості його полягає у збільшення рахунку. Система дозволяє динамічно змінювати швидкість приросту рахунку через змінну scoreMultiplier. Наприклад, рахунок може зростати швидше на складніших рівнях.

Простий UI: Сценарій напряму оновлює текстовий елемент, забезпечуючи плавне відображення змін.

Після зіткнення головного героя з предметом, який ініціює програш гри, потрібна панель для перезапуску гри.

На цій панелі буде реалізована кнопка рестарту гри і в лівому куті підрахунок очок за спробу користувача. Вона з'являтиметься коли персонаж зіткнеться з об'єктом, при цьому процес гри буде припинений для зменшення навантаження на оперативну пам'ять пристрою, тому що не відбуватиметься генерація наступних об'єктів програшу.

2.1.3 Реалізація головного героя

Для створення головного героя було використано сервіс Mixamo. Mixamo – це онлайн–сервіс, який суттєво спрощує процес створення анімації

для 3D–моделей. Він пропонує велику бібліотеку готових анімацій, які можна легко застосовувати до власних персонажів. Цей сервіс має більше функціоналу, порівняно з конкурентами. Що Mixamo може запропонувати:

- автоматичний ригінг: Завантажте свою 3D–модель у людській подобі, і Mixamo автоматично створить для неї скелет, необхідний для анімації. Це значно економить час, який зазвичай витрачається на ручне налаштування ригінгу;
- бібліотека анімацій: Виберіть із тисяч готових анімацій, від простих ходьби та бігу до складних бойових дій та виразних емоцій. Анімації створені професійними моушен–акторами та можуть бути адаптовані під ваші потреби;
- простий інтерфейс: Навіть якщо ви не є досвідченим аніматором, ви зможете легко розібратися в інтерфейсі Mixamo та застосовувати анімації до своїх моделей;
- підтримка різних форматів: Mixamo підтримує широкий спектр форматів 3D–моделей і дозволяє експортувати готові анімації до різних форматів для використання в різних програмах, таких як ігрові движки, 3D–редактори та програми для створення відео;

Для створення головного героя потрібна його 3–D модель для подальших додавань анімацій [6]. Для додавання анімацій у сервісі Mixamo виставляються точки на моделі для плавності анімацій як показано на рисунку 2.6. Після обробки моделі сервісом у запропонованому списку вибираються відповідні анімації для програми. Саме для цієї програми потрібно додати стрибок і перекат для уникнення перешкод.

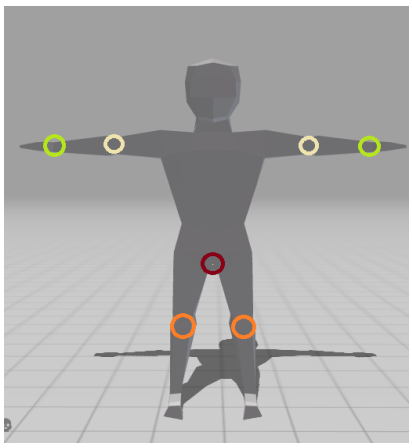


Рисунок 2.6– 3D модель

Для імпорту анімацій до проекту потрібно використовувати функцію Animator Controller (рис. 2.7). У ньому додаємо усі створенні анімації з Mixamo та налаштовуємо ієрархію виконання дії персонажем.

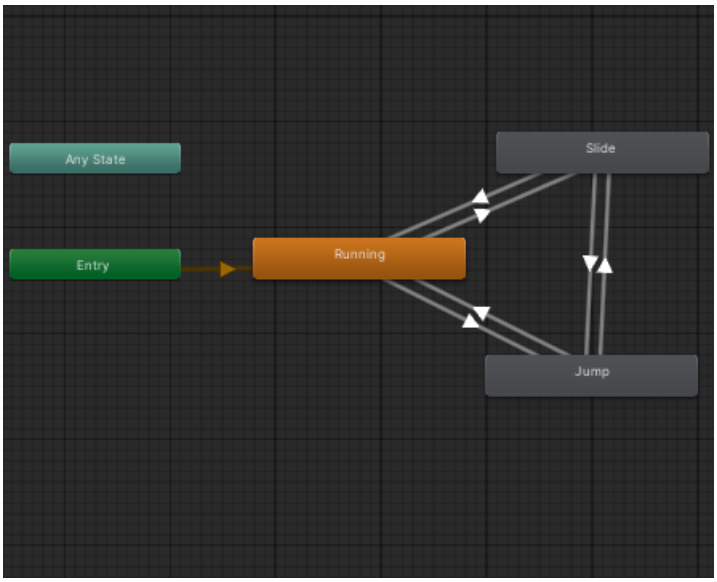


Рисунок 2.7– Функція Animator

Щоб анімації працювали, також потрібно створити аватар персонажа. Для цього в Unity є функція створення аватара, яка створює його з будь–якої доданої раніше анімації персонажа.

У скрипті «Player Controller» потрібно реалізувати створені анімації за допомогою коду (рис 2.8).

```
private bool isSliding;

  if (controller.isGrounded && !isSliding)
      anim.SetBool("isRunning", true);
  else
      anim.SetBool("isRunning", false);

private void Jump()
{
    dir.y = jumpForce;
    anim.SetTrigger("isJumping");
}
```

```
private IEnumerator Slide()
{
    col.center = new Vector3(0, -0.4f, 5f);
    col.height = 2;
    isSliding = true;
    anim.SetBool("isRunning", false);
    anim.SetTrigger("isSliding");

    yield return new WaitForSeconds(1);

    col.center = new Vector3(0, 0.108f, 0);
    col.height = 4.420422f;
    isSliding = false;
}
```

Рисунок 2.8 – Нові елементи у скрипту

Метод Slide() є корутиною, яка тимчасово змінює параметри ігрового об'єкта (наприклад, колайдер) та анімацію, щоб виконати ефект "ковзання" персонажа. Це може бути використане, наприклад, в іграх, щоб персонаж ковзав під перешкодами або виконував інші динамічні дії.

Ініціалізація ковзання:

– col.center = новий Vector3(0, –0.4f, 5f) – зміщує центр колайдера (col) вниз та трохи вперед. Це потрібно для того, щоб колайдер збігався із новою позою персонажа під час ковзання;
– col.height = 2 –зменшує висоту колайдера, щоб він підходив для пози ковзання;
– isSliding = true – встановлює прапор, який сигналізує, що персонаж перебуває у стані ковзання. Це може бути корисним для блокування інших дій персонажа під час виконання ковзання;
– Анімація:
– anim.SetBool("isRunning", false) – вимикає анімацію бігу;
– anim.SetTrigger("isSliding") – активує анімацію ковзання.

Затримка:

– yield return new WaitForSeconds(1) – визначає паузу на 1 секунду. Протягом цього часу персонаж залишатиметься у стані ковзання.

Завершення ковзання:

– col.center = новий Vector3(0, 0.108f, 0) – повертає центр колайдера у вихідне положення;

– col.height = 4.420422f – відновлює вихідну висоту колайдера;
– isSliding = false – скидає прапор, сигналізуючи про завершення ковзання.

Після всіх проведених операцій виходить повноцінний головний герой гри (рис 2.9).

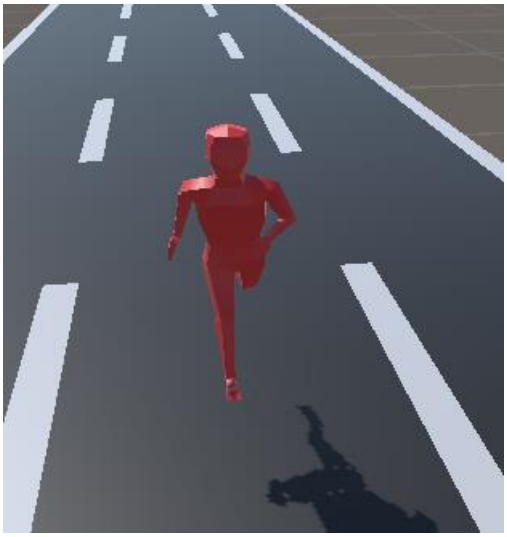


Рисунок 2.9 – Персонаж гри

2.1.4 Розробка та застосування текстур в ігровому проекті

Текстури в іграх – це двовимірні зображення, які накладаються на поверхню тривимірних об'єктів для надання їм деталізації, кольору та реалістичності. Текстури використовуються для створення візуального оформлення ігрових елементів, таких як персонажі, довкілля, дороги, перешкоди та інші об'єкти.

У раннерах, як жанр ігор (наприклад, Subway Surfers, Temple Run), текстури відіграють ключову роль, оскільки вони задають зовнішній вигляд елементів, з якими взаємодіє гравець.

Приклади використання текстур у раннерах:

– дорога: асфальт текстури, плитки, землі або трави. Може містити лінії, тріщини, тіні чи інші елементи, що створюють відчуття реалізму;

– оточення: будівлі, дерева, тунелі або скелі оформляються за допомогою текстур, щоб надати їм різноманітності та впізнаваності. Часто текстури повторюються (tileable textures) для оптимізації;
– об'єкти та перешкоди: каміння, бар'єри, рейки або автомобілі оформлюються текстурами, щоб вони виглядали деталізованими;
– персонажі та їх одяг: текстури використовуються для відображення шкіри, волосся, одягу чи аксесуарів персонажа;
– ефекти та бонуси: Монети, прискорювачі, підсвічування або магніти мають текстури, які роблять їх яскравими та помітними.

Усі текстури у грі були зроблені у сервісі Blender. Blender – це безкоштовне та відкрите програмне забезпечення для 3D–моделювання, анімації, рендерингу та композитингу. Воно поширюється під ліцензією GNU General Public License (GPL), що дозволяє вільно використовувати, змінювати та розповсюджувати програму[8].

Щоб реалізувати задуману ідею гри для початку варто створити платформу, по якій переміщатиметься головний герой. У цій грі це буде асфальтна дорога, яка має розмітку в три смуги, на яких будуть розташовані проблеми (рис. 2.10).

Наступний етап – це додавання у довільному форматі перешкод, які були створені раніше у вигляді машин. Кожна нова створена ділянка дороги повинна мати різну географію об'єктів.

Щоб урізноманітнити кількість перешкод у проекті було додати новий об'єкт шлагбаум [7].

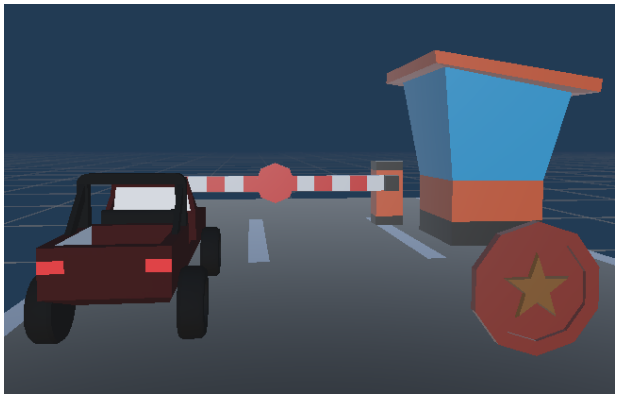


Рисунок 2.10 – Вигляд частини дороги

Щоб гра стала повноцінною в аспекті текстуризації, слід додати текстури, які будуть задавати атмосферу гри.

У цьому проекті буде реалізовано текстури у вигляді будинків, які логічно підходять в ідею гри та гармонійно виглядатимуть з іншими об'єктами (рис. 2.11). Вони також будуть розроблені в сервісі Blender[1].



Рисунок 2.11 – Нові текстури у грі

Текстури мають середню роздільну здатність і рівень деталізації, щоб зменшити навантаження на процесор користувача. Висвітлення відіграє у візуалізації текстур. У цьому випадку висвітлення вибрано таким чином, щоб підкреслити текстуру матеріалів і створити приємний візуальний ефект.

Такі текстури зазвичай використовуються для мінімалістичних сцен, щоб наголосити на стилізації low–poly. Low–poly – це стиль 3D–моделювання, у якому використовуються моделі з мінімальною кількістю полігонів. Цей підхід робить об'єкти простіше за формою та текстурою, що надає їм характерного "спрощеного" або "стилізованого" виглядур[9].

2.1.5 Створення головного меню і системи рекорду

Для створення головного меню необхідно створити нову сцену гри. На новій сцені буде реалізовано кнопку запуску гри (рис 2.12). Щоб почати гру, потрібно кнопка для запуску процесу гри.

У новій сцені потрібно видалити непотрібні елементи у її створенні, а це: Player, Swipe Controller, Tile Generator (рис 2.13).

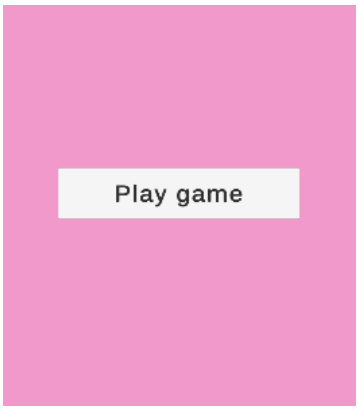


Рисунок 2.12 – Кнопка початку гри



Рисунок 2.13 – Дві сцени у грі

Щоб кнопка запуску гри стала активною, потрібно створити скрипт «MainMenu» (рис 2.14). У скрипті використовуються нові бібліотеки:

– unityEngine.SceneManagement: використовується для керування сценами, наприклад, їх завантаження;
– unityEngine.UI: дозволяє працювати з UI–елементами, такими як текст, кнопки та зображення.

Для майбутніх функцій гри потрібно створити елемент для підрахунку монет: private Text coinsText: посилання на UI–елемент Text, який використовується для відображення кількості монет гравця на екрані.

Опис методів у скрипті:

– метод Start(). Виконується один раз під час запуску сцени (в даному випадку головного меню). PlayerPrefs.GetInt("coins"). Завантажує

збережену кількість монет із сховища PlayerPrefs, яке використовується для збереження даних між сесіями гри. coinsText.text = coins.ToString(). Встановлює текстове поле (coinsText) рівним кількості монет, перетвореному на рядок;

– метод PlayGame(). Цей метод викликається, коли гравець натискає кнопку "Play" (або її аналог) у головному меню. SceneManager.LoadScene(0). Завантажує сцену з індексом 0 із Build Settings. Зазвичай це ігрова сцена.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5   using UnityEngine.UI;
6
7   public class MainMenu : MonoBehaviour
8   {
9       [SerializeField] private Text coinsText;
10      private void Start()
11      {
12          int coins = PlayerPrefs.GetInt("coins");
13          coinsText.text = coins.ToString();
14      }
15
16      public void PlayGame()
17      {
18          SceneManager.LoadScene(0);
19      }
20  }
```

Рисунок 2.14 – Скрипт MainMenu

У сцені "Menu" потрібно видалити всі скрипти з "Main Camera" і накласти лише "MainMenu". У налаштуваннях проекту потрібно додати створену сцену для її відображення у грі. Для цього потрібно перейти у вкладку Build Settings та додати зроблену раніше сцену (рис. 2.14).

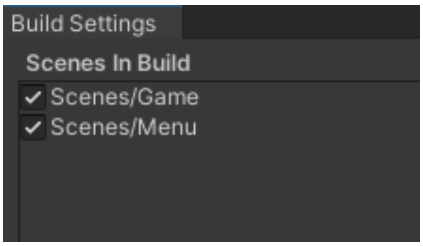


Рисунок 2.15 – Build Settings

Щоб реалізувати систему рекордів потрібно для початку створити елемент гри, який буде "лічильником прогресу" (рис. 2.16). У цьому додатку

це будуть монети, які збиратиме гравець. Для реалізації монетки було використано сервіс Blender[2].

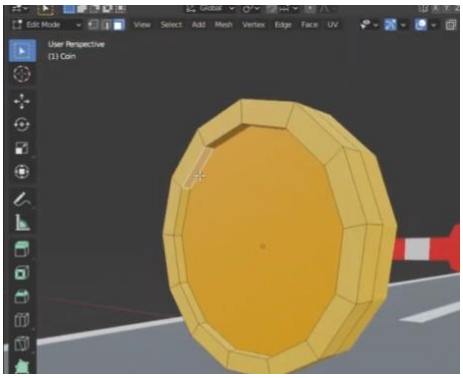


Рисунок 2.16 – Текстура монети у Blander

Далі створюємо їй префаб. Префаб у Unity – це, по суті, шаблон чи прототип ігрового об'єкта. Це як креслення, яким можна створювати безліч однакових або схожих об'єктів на ігровій сцені.

Розміщення монет на ділянках дороги відбувається у довільному вигляді. Для самої монети потрібно налаштувати її колізію, щоб гравець підбирав її (рис. 2.17).

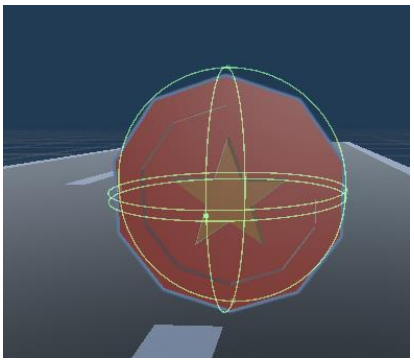


Рисунок 2.17 – Колізія монети у грі

Ще для краси програми було реалізовано обертання самої монети за допомогою додавання до скрипта «Coin» (рис. 2.18).

Щоб реалізувати систему рекордів, для цього потрібно створити панель програшу. На якій буде і реалізовано виведення найкращого рахунку гравця.

Також на ній варто розмістити кнопки перезапуску гри та переадресацію на головне меню, останнє кількості набраних очок.

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

public class Coin : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(0, 40 * Time.deltaTime, 0);
    }
}
```

Рисунок 2.18 – Скрипт «Coin»

Для цього треба створити новий скрипт «LosePanel» який буде виконувати наступні дії (рис. 2.19). Скрипт управляє панеллю програшу (LosePanel) у грі. Скрипт виконує такі функції за допомогою деяких методів. Перший це Start(). При запуску сцени (або об'єкта) отримує з PlayerPrefs два значення:

– lastRunScore: Результат останнього проходження;
– recordScore: Рекордний результат.

```
public class LosePanel : MonoBehaviour
{
    [SerializeField] Text recordText;

    private void Start()
    {
        int lastRunScore = PlayerPrefs.GetInt("lastRunScore");
        int recordScore = PlayerPrefs.GetInt("recordScore");

        if (lastRunScore > recordScore)
        {
            recordScore = lastRunScore;
            PlayerPrefs.SetInt("recordScore", recordScore);
            recordText.text = recordScore.ToString();
        }
        else
        {
            recordText.text = recordScore.ToString();
        }
    }


    public void RestartLevel()
    {
        SceneManager.LoadScene(0);
    }

    public void ToMenu()
    {
        SceneManager.LoadScene(1);
        Time.timeScale = 1;
    }
}
```

Рисунок 2.19 – Скрипт «Lose Panel»

Якщо результат останнього проходження (lastRunScore) більший за поточний рекорд (recordScore), то новий рекорд оновлюється в PlayerPrefs. Відображається новий рекорд у текстовому полі recordText.

Якщо рекорд не побито, в recordText просто відображається існуючий рекорд. Другий метод RestartLevel(). Завантажує сцену з індексом 0, щоб розпочати гру заново.

Третій метод ToMenu(). Завантажує сцену з індексом 1 (головне меню). Скидає час у грі (Time.timeScale = 1), щоб усунути можливі ефекти паузи.

Також потрібно додати нову частину коду до скрипту «Player Controller» (рис. 2.20). Завдання коду отримати текст з scoreText(де відображається поточний рахунок гравця, перетворити його на ціле число і зберегти це значення в змінній lastRunScore[10].

```
losePanel.SetActive(true);
int lastRunScore = int.Parse(scoreScript.scoreText.text.ToString());
Time.timeScale = 0;
```

Рисунок 2.20 – Новий елемент коду

Для того щоб все спрацювало треба створити новий елемент у проекті який буде нараховувати бали гравця. Він реалізовано за допомогою елементу «Text», який рахує кількість зібраних монет і певну дистанцію (рис. 2.21).

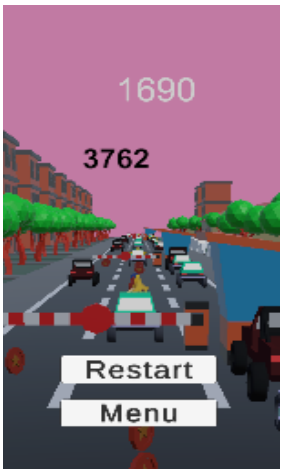


Рисунок 2.21 – Фінальний вигляд панелі

2.1.6 Система бонусів

Система бонусів в іграх жанру раннер – це механіка, яка дозволяє гравцям збирати корисні предмети або досягнення, що підвищують інтерес до гри, різноманітність та прогрес. Такі бонуси мотивують гравців та роблять ігровий процес більш захоплюючим.

Основні види бонусів у раннерах:

Підсилювачі. Підсилювачі тимчасово дають гравцеві перевагу:

– магніт: Приваблює монети навколо гравця;
– подвійні очки: Подвоює зароблені очки за певний час;
– швидкісний буст: Збільшує швидкість персонажа, автоматично уникаючи перешкод;
– щит: Захищає від зіткнень на певний час;

Колекційні предмети. Гравці збирають рідкісні об'єкти для виконання завдань або відкривання додаткових рівнів. Приклад: Букви для складання слова (наприклад, "RUN"), які дають приз при завершенні.

Подієві бонуси. Проводяться під час тимчасових подій чи спеціальних акцій:

– сезонні бонуси (наприклад, зимові подарунки);
– рідкісні скіни або підсилювачі;

Другий шанс. Дозволяє гравцеві продовжити гру після поразки, використовуючи внутрішньоігрову валюту або перегляд реклами.

У цій грі вже реалізовано систему збору монет, але так само тут для більш захоплюючого ігрового процесу буде реалізовано систему щита, який даватиме 2 життя, а також систему нарахування очок у подвійному режимі.

Для створення системи нарахування подвійних очок необхідно зробити текстуру об'єкта (рис. 2.22). Вона буде у формі зірки. Також варто намалювати її зовнішній вигляд. Для цього буде використано сервіс

Photoshop. Photoshop (повна назва Adobe Photoshop) – це професійне програмне забезпечення для роботи із зображеннями, розроблене компанією Adobe Systems. Воно є одним з найпопулярніших і найпотужніших інструментів для редагування растрової графіки та створення цифрового контенту.



Рисунок 2.22 – Зовнішній вид зірки

Сама її текстура все також реалізована за допомогою Blender (рис. 2.23).

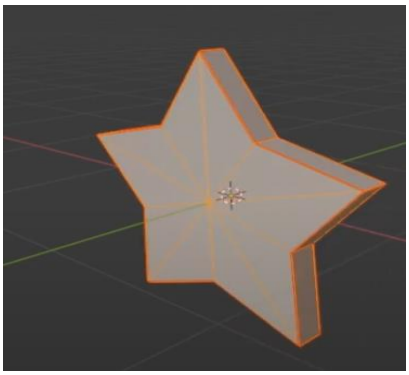


Рисунок 2.23 – Текстура зірки

Цей об'єкт також матиме функцію обертання. А також для неї потрібно додати компонент Box Collider щоб гравець міг її підбирати.

У скрипті «Player Controller» потрібно додати нову умову на знаходження наших зірочок. Щоб було коректний підрахунок очок, потрібно

налаштувати деякі аспекти в декількох скриптах. В першу чергу потрібно створити нове поле типу GameObject – "[SerializeField] private GameObject scoreText;", куди і буде вміщено новий об'єкт, який підраховуватиме набрані очки.

Створений скрипт "Score" поміщаємо на елемент виведення очко, який налаштовуємо як показано на рисунку 2.24.

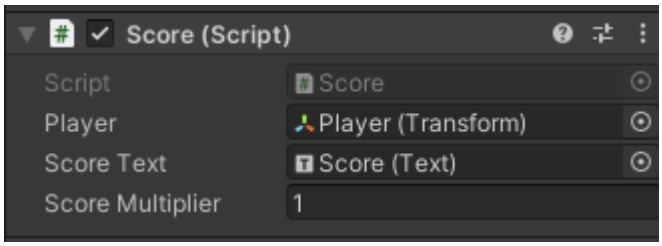


Рисунок 2.24 – Налаштування об’єкту

Ще потрібно додати фрагмент коду, який буде тимчасово збільшувати множник очок гравця і потім повертає його в нормальний стан через 5 секунд. Цей скрипт можна викликати, наприклад, коли гравець підбирає бонусний предмет у грі, який тимчасово подвоює зароблені очки.

На рисунку 2.25 є елемент Score Multiplier, цей елемент виконує дію щоб при підборі зірки гравцю завжди надавало однаковий бонус, тому що комусь може на старті гри видати великий помножувач очок. Це виконано у скрипті «Player Controller» у методі Start завдяки рядку «score.scoreMultiplier = 1;».

```
private IEnumerator StarBonus()
{
    score.scoreMultiplier = 2;

    yield return new WaitForSeconds(5);

    score.scoreMultiplier = 1;
}
```

Рисунок 2.25 – Нова частина скрипта

Також у грі реалізовано систему "2 життя", тобто якщо гравець підбирає елемент Щит, то у нього про зіткнення з перешкодою гра не закінчується, а триває далі.

Цей елемент також створений у програмі Blender. З ним проводяться ті ж дії, що і з зіркою, тобто додається колізія і розташування на ігровій зоні.

Щоб реалізувати його дії в грі, то все в тому ж скрипті "Player Controller" потрібно додати булеву змінну: private bool isImmortal. Вона відображатиме безсмертний гравець чи ні. У методі Start потрібно визначити, що гравець спочатку не безсмертний: «isImmortal = false;».

У методі OnControllerColliderHit (рис. 2.26) потрібно додати новий елемент, який if (hit.gameObject.tag == "obstacle") перевіряється, чи об'єкт, з яким сталося зіткнення, має тег "obstacle" (перешкода). Якщо це перешкода, виконуються дії, що описані в тілі умови. Також «if (isImmortal)» перевіряється стан змінної isImmorta. Якщо «isImmortal == true:» – гравець перебуває у стані "безсмертя". І тут перешкода знищується з допомогою Destroy(hit.gameObject). Якщо гравець не безсмертний (isImmortal == false), то виконуються інші дії (протягом коду).

```
private void OnControllerColliderHit(ControllerColliderHit hit)
{
    if (hit.gameObject.tag == "obstacle")
    {
        if (isImmortal)
            Destroy(hit.gameObject);
```

Рисунок 2.26 – Метод OnControllerColliderHit

Так само, як і з елементом зірки, потрібно додати IEnumerator (рис. 2.27). Цей фрагмент коду є корутиною, яка тимчасово активує стан "безсмертя" у гравця на 5 секунд. Приклад роботи у грі:

Гравець підбирає бонус "щит":

– стан "безсмертя" активується.

– гравець захищений від перешкод або втрати протягом 5 секунд;

Через 5 секунд:

– стан "безсмертя" вимикається;

– гравець знову стає вразливим.

```
private IEnumerator ShieldBonus()
{
    isImmortal = true;

    yield return new WaitForSeconds(5);

    isImmortal = false;
}
```

Рисунок 2.27 – Новий IEnumerator

3 МОНЕТИЗАЦІЯ У ІГРАХ

Монетизація є ключовим аспектом у розробці та підтримці сучасних відеоігор. Вона охоплює різноманітні методи отримання доходів розробниками, видавцями або платформами шляхом використання певних механік у грі. Ефективна монетизація має враховувати баланс між інтересами розробників та комфортом гравців. Агресивні підходи, як надмірна реклама або "pay–to–win" моделі, можуть негативно вплинути на репутацію гри та довіру користувачів.

Моделі монетизації є центральним елементом гри, який визначає її фінансовий успіх. Вибір правильної стратегії монетизації слід робити ще на початкових етапах розробки, адже від цього залежить загальний результат проєкту. Розробники прагнуть максимізувати прибуток, тоді як гравці надають перевагу безкоштовному доступу до ігор. Проте, надмірна жадібність деяких розробників може зашкодити користувацькому досвіду, викликати розчарування у гравців і навіть змусити їх видалити гру.

Гравці, усвідомлюючи потребу розробників у доходах, очікують наявності елементів монетизації, особливо у мобільних іграх. Важливо знайти баланс: розумний підхід дозволяє зберегти лояльність користувачів і перетворити їх на платників[3].

Покупки всередині гри є провідною моделлю монетизації для мобільних ігор. За даними Google AdMob, 79% мобільних ігор використовують внутрішньоігрові покупки (IAP) як основне джерело доходу. У freemium–іграх користувачі отримують базовий доступ безкоштовно, але для розширеного досвіду, наприклад, додаткових життів, внутрішньоігрової валюти, спеціального спорядження, вони можуть здійснювати оплати в додатку. Центром майже кожної ігрової економіки є внутрішньоігровий магазин, де гравці можуть придбати різні товари.

Внутрішньоігрова реклама – це друга за популярністю модель монетизації, коли розробники заробляють гроші, показуючи рекламу гравцям.

У такій моделі гравці отримують всі функції гри безкоштовно, але мають переглядати рекламу під час гри.

Основні формати реклами:

– interstitial ads: Ці оголошення займають весь екран і часто з'являються в природних паузах геймплей, наприклад, між рівнями;
– rewarded video ads: Гравці можуть добровільно переглядати рекламу, щоб отримати винагороду, наприклад, ігрові предмети або валюту (рис. 3.1);
– banner ads: Це статичні прямокутні оголошення, які з'являються у певних місцях на екрані гравця;



Рисунок 3.1 – Приклад Rewarded Video Ads

3.1 Різновиди монетизації у іграх

Монетизація у іграх охоплює різноманітні моделі, які дозволяють розробникам отримувати дохід і забезпечувати фінансову стабільність проєкту. Розглянемо основні різновиди монетизації у іграх.

Перше це продаж самої гри. Гравці сплачують разову фіксовану суму за доступ до гри і з цього вже їде прибуток розробнику. Ця модель використовується переважно для консольних та ПК–ігор. Продаж гри охоплює кілька етапів: від створення продукту та маркетингу до її доступності для придбання. Гравець платить повну вартість гри один раз і отримує доступ до всього контенту. Прикладом можуть бути AAA–ігри (приклади: The Witcher 3, Elden Ring)або інді–проекти (приклади: The Binding of Isaac).

Друге це підписка у іграх. Гравці платять регулярну абонентську плату за доступ до гри або її окремих функцій. Використання підписки в іграх може бути різноманітним. Наприклад, в онлайн–іграх типу MMORPG підписка дає змогу грати на серверах гри, отримувати оновлення, брати участь у великих подіях і отримувати преміальні ресурси. Такий підхід часто зустрічається у великих проектах, де потрібні значні кошти на підтримку серверів та створення нового контенту.

Інший варіант – підписка як доступ до великого каталогу ігор. У такій моделі, наприклад, працюють сервіси Xbox Game Pass або PlayStation Plus. Гравці сплачують фіксовану суму і отримують доступ до колекції ігор, які вони можуть завантажувати і грати протягом дії підписки. Ця модель зручна, оскільки дозволяє пробувати безліч ігор без необхідності купувати кожну окремо. У мобільних іграх підписка часто надає доступ до додаткових функцій, таких як відсутність реклами, прискорення прогресу, унікальні предмети чи скіни, які не можна отримати інакше. Наприклад, гравець може отримувати щоденні бонуси, знижки на внутрішньоігрові покупки або навіть ексклюзивні режими гри.

Популярність підписок у іграх пояснюється тим, що вони вигідні для обох сторін. Гравці отримують зручність і постійний доступ до контенту без

необхідності великих разових витрат, тоді як розробники забезпечують собі стабільний дохід, який допомагає покривати витрати на підтримку гри та створення оновлень.

Третій варіант який розповсюджений у кожній грі це донати. Донати у іграх – це система добровільних фінансових внесків гравців, яка стала одним із ключових способів монетизації в індустрії відеоігор. Вона дозволяє гравцям витрачати реальні гроші на внутрішньоігрові предмети, бонуси, косметичні елементи чи інші переваги, які можуть зробити їхній досвід у грі більш цікавим або комфортним. Найчастіше донати використовуються у безкоштовних іграх (Free–to–Play), де сам процес гри не вимагає оплати, але додаткові функції пропонуються за гроші.

Гравці можуть купувати, наприклад, віртуальну валюту, яку потім обмінюють на ексклюзивні предмети, скіни, покращення або навіть прискорення прогресу в грі. Донати можуть також включати покупку боєприпасів, енергії чи ресурсів, які необхідні для проходження складних рівнів. Вони часто розраховані на різні категорії гравців: від тих, хто хоче трохи полегшити гру, до "донаторів" – людей, готових витрачати значні суми заради унікального досвіду чи статусу у грі.

У багатьох іграх донати мають лише косметичний характер, тобто не впливають на баланс гри. Це може бути унікальний вигляд персонажа, декоративні елементи чи спеціальні емоції. У таких випадках вони сприймаються більш позитивно, адже не дають конкурентної переваги над іншими гравцями. Однак у деяких іграх донати можуть мати "Pay–to–Win" характер, коли гравці, які вкладають гроші, отримують значну перевагу, що часто викликає критику від спільноти.

Донати також використовуються як форма підтримки розробників. Деякі гравці охоче жертвують гроші, щоб підтримати створення нових ігор або оновлень, особливо якщо вони насолоджуються безкоштовним контентом. У мобільних іграх донати можуть включати плату за відключення реклами чи отримання преміальних функцій.

Загалом донати стали важливим елементом сучасного ігрового ринку, дозволяючи розробникам отримувати дохід і підтримувати свої проєкти, а гравцям – обирати, чи хочуть вони витрачати гроші для покращення свого досвіду. Однак їх використання має бути обережним, щоб уникнути ситуацій, коли фінансові внески впливають на справедливість гри або відлякують частину гравців.

Ще один із варіантів монетизації у гри це сезонні пропозиції та Battle Pass. Це популярні механіки монетизації та залучення гравців у сучасних відеоіграх. Вони поєднують у собі обмежені в часі акції, ексклюзивні нагороди та мотивацію для активного залучення до ігрового процесу протягом певного періоду.

Сезонні пропозиції, як правило, прив’язані до конкретного сезону чи події в грі, наприклад, святкових періодів (Різдва, Хелловін) або запуску нового контенту. Під час таких подій гравцям пропонуються унікальні предмети, скіни, завдання або режими гри, які доступні лише протягом обмеженого часу. Це створює унікальне відчуття ексклюзивності та стимулює гравців повертатися в гру, щоб не пропустити можливість отримати щось особливе. Сезонні пропозиції часто супроводжуються знижками або наборами, які включають вигідні комбінації преміум-контенту.

Battle Pass (бойовий пропуск) – це система прогресії, яка розділяє ігровий сезон на рівні або ступені, кожен з яких пропонує нагороди за виконання завдань і досягнення певних результатів. Гравці можуть отримувати винагороди за гру як безкоштовно, так і через придбання платного варіанта Battle Pass, який відкриває доступ до преміальних нагород. У багатьох випадках ці нагороди включають косметичні предмети, віртуальну валюту, емоції чи інші ексклюзивні бонуси, які неможливо отримати іншим способом. Платний Battle Pass також зазвичай надає більше завдань або збільшує кількість доступних рівнів, що дозволяє гравцям отримати ще більше нагород.

Особливість Battle Pass у тому, що він мотивує гравців регулярно заходити в гру, щоб виконувати завдання та досягати нових рівнів до завершення сезону. Це забезпечує довгострокове залучення аудиторії та створює відчуття досягнення, адже кожен рівень – це результат докладених зусиль. Крім того, завдяки сезонному оновленню, гравці завжди отримують щось нове, що допомагає уникнути одноманітності.

Обидві механіки успішно використовуються у таких іграх, як Fortnite, Apex Legends чи Call of Duty, і дозволяють одночасно утримувати гравців, мотивувати їх до повернення та забезпечувати стабільний дохід для розробників. Вони базуються на ідеї обмеженості в часі, яка створює у гравців бажання не пропустити унікальні можливості, зберігаючи інтерес до гри протягом тривалого часу.

Модель базується на продажу тимчасового контенту. Гравці купують доступ до "сезонного абонемента", що пропонує унікальні нагороди, місії чи косметичні предмети. Ця модель стимулює утримання користувачів і активність протягом сезону.

Останній вид монетизації у грі який розглянуто це розповсюдження DLC та розширень. Розповсюдження DLC (Downloadable Content, завантажуваний контент) та розширень – це спосіб підтримки та розширення ігор після їх релізу. Ці додаткові матеріали дозволяють розробникам продовжувати заробляти на вже випущеній грі, водночас пропонуючи гравцям новий контент і можливості, які збагачують ігровий досвід.

DLC часто включають нові місії, рівні, персонажів, зброю, скіни чи косметичні предмети, які можна додати до основної гри. Їх розробка зазвичай починається після завершення основного проєкту, хоча іноді вони плануються заздалегідь, особливо для великих ААА-ігор. Розширення, у свою чергу, є більшими за обсягом і масштабом. Вони можуть включати значні зміни в геймплеї, додавання нових сюжетних ліній, великих карт чи навіть цілих режимів гри, які суттєво змінюють або доповнюють гру.

Розробники розповсюджують DLC та розширення через цифрові платформи, такі як Steam, Epic Games Store, PlayStation Store чи Xbox Store. Для мобільних ігор їх можна знайти в App Store чи Google Play. Деякі DLC можуть бути безкоштовними, особливо якщо вони випущені для підтримки спільноти чи як подяка гравцям, однак більшість із них продаються окремо або в складі сезонних пропозицій чи спеціальних видань гри. Іноді розробники пропонують "Season Pass" – пакет, який включає всі майбутні DLC за нижчою ціною, ніж якби їх купували окремо.

Популярність DLC пояснюється можливістю подовжити життя гри та підтримувати інтерес гравців. Наприклад, багатокористувацькі ігри часто отримують регулярні оновлення з новими картами, зброєю чи персонажами, щоб урізноманітнити ігровий процес. У сюжетних іграх розширення можуть додати нові історії, які дозволяють глибше зануритися у всесвіт гри.

Проте не всі ставляться до DLC однаково позитивно. Деякі гравці критикують розробників за "відрізання" частини контенту з основної гри, щоб продати його окремо, або за високі ціни на DLC, які не виправдовують очікувань. У відповідь на це багато студій намагаються балансувати, пропонуючи безкоштовні оновлення поряд із платним контентом.

Розповсюдження DLC та розширень стало невід'ємною частиною сучасної розробки ігор. Воно дозволяє розробникам залишатися на плаву навіть після завершення розробки основної гри, забезпечує стабільний дохід і дає гравцям можливість отримувати новий контент, продовжуючи насолоджуватися улюбленими проектами.

3.2 Реалізація монетизації у проекті

Для того, щоб реалізувати монетизацію в грі буде використовуватись сервіс Google AdMob. Насамперед потрібно заповнити платіжну інформацію,

інакше ніяка реклама крім тестової не буде показуватися. Далі потрібно додати новий додаток, тобто раніше створену гру (рис. 3.2).

Після додавання гри обслуговування потрібно вибрати який тип реклами буде реалізований у проекті. У випадки мобільного додатку найкращими варіантами є "Міжсторінкове оголошення" та "Реклама з винагородою". Міжсторінкове оголошення та реклама з винагородою є одними з найефективніших видів реклами для ігор в стилі раннер з кількох причин.

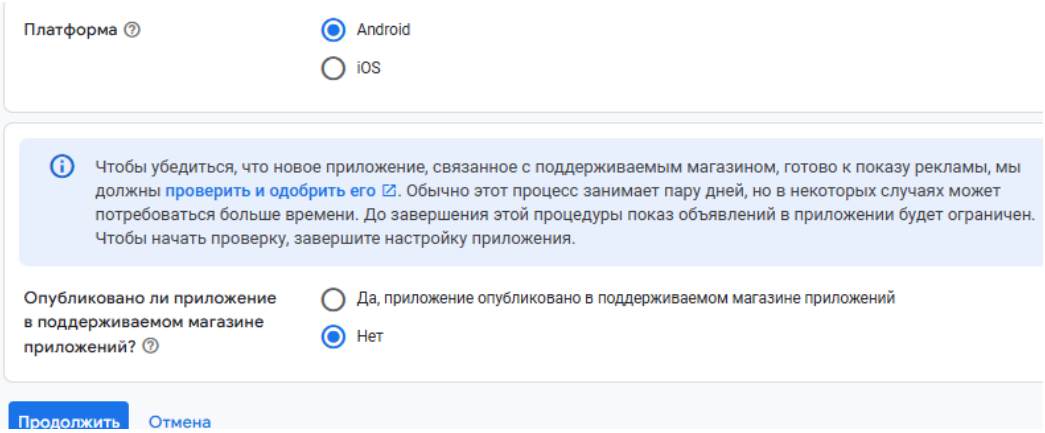


Рисунок 3.2 – Додавання гри у Google AdMob

Міжсторінкове оголошення — це реклама, яка з'являється між рівнями або етапами гри, коли гравець переходить від одного рівня до іншого. Цей формат є ненав'язливим, оскільки він не перериває геймплей в середині рівня, але в той же час дозволяє розробникам отримати дохід. Ігри в стилі раннер часто мають короткі сесії, і між етапами гравець може легко відпочити від активного процесу гри, що створює ідеальну ситуацію для демонстрації реклами. Це дозволяє забезпечити хороший баланс між утриманням інтересу гравця і монетизацією гри.

Реклама з винагородою — це форма реклами, в якій гравець отримує певну винагороду (наприклад, бонуси, монети або інші ігрові предмети) за перегляд рекламного відео або взаємодію з рекламою. Для ігор в жанрі раннер це є ідеальним варіантом, оскільки такі ігри часто базуються на

швидких сесіях, і гравці можуть бути зацікавлені в додаткових бонусах або ресурсах, які допоможуть їм прогресувати в грі. Винагорода стимулює гравців дивитися рекламу, оскільки вони отримують щось корисне для свого досвіду в грі. Це створює взаємовигідну ситуацію: розробники заробляють на рекламі, а гравці отримують додаткові можливості для просування в грі.

Обидва ці формати реклами є дуже популярними в безкоштовних іграх, особливо в мобільних іграх жанру раннер, оскільки вони дозволяють монетизувати гру без того, щоб перешкоджати гравцям насолоджуватися основним процесом гри. Вони зберігають баланс між геймплейем і рекламою, що робить їх не настільки нав'язливими для користувачів і ефективними для розробників.

Щоб додати та перевірити працездатність доданої реклами, потрібно встановити пакет плагінів AdMob для Unity. Ці плагіни є у загальному доступі в інтернеті. Щоб їх додати, потрібно перенести завантажений файл у проект і програма його автоматично встановить. Щоб перевірити коректну установку пакета плагінів у спливаючому меню Assets, та побачити новий елемент у меню (рис. 3.3).

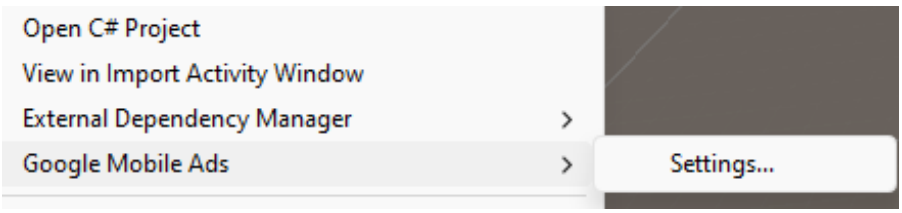


Рисунок 3.3 – Плагін Google Mobile Ads

Потрібно створити скрипт, який ініціалізуватиме рекламу (рис. 3.4).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using GoogleMobileAds.Api;

public class AdInitialize : MonoBehaviour
{
    private void Awake()
    {
        MobileAds.Initialize(initStatus => { });
    }
}
```

Рисунок 3.4 – Ініціалізатор реклами

Цей скрипт призначений для ініціалізації бібліотеки Google Mobile Ads у програмі. Скрипт включає необхідні простори імен, включаючи стандартні колекції Unity, простір GoogleMobileAds.Api та базові компоненти Unity. У класі AdInitialize реалізовано метод Awake, який автоматично викликається Unity під час запуску сцени. У цьому методі викликається статична функція MobileAds.Initialize, яка відповідає за ініціалізацію SDK AdMob. Як параметр передається лямбда-функція з порожнім тілом (initStatus => { }), що дозволяє завершити налаштування SDK без виконання додаткових дій. Цей скрипт рекомендується додати на об'єкт у сцені, який є протягом всього часу роботи програми, щоб забезпечити коректну роботу рекламного SDK.

Перша реклама, яка буде реалізована це міжсторінкове оголошення. Для зручнішої навігації на сервісі її назва буде «After 6 lose» (рис. 3.5).

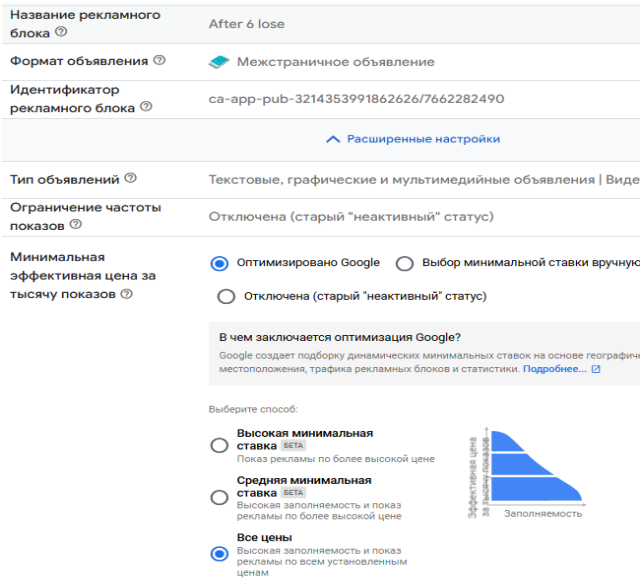


Рисунок 3.5 – Налаштування рекламного блоку

Щоб міжсторінкова реклама заробила для неї теж потрібен скрипт, який її запускатиме (рис. 3.6).

У ньому підключені необхідні простори імен, такі як UnityEngine для базової роботи Unity та GoogleMobileAds.Api для взаємодії з AdMob. У класі

InterAd створюється змінна interstitialAd, що представляє об'єкт міжсторінкової реклами, і задається ідентифікатор рекламного блоку у вигляді рядка interstitialUnitId, яка використовується для завантаження реклами (у даному випадку використовується тестовий ID, наданий AdMob). Коли об'єкт, на якому розміщено цей скрипт, стає активним (метод OnEnable), створюється новий екземпляр міжсторінкової реклами з використанням вказаного ідентифікатора. Потім формується запит завантаження реклами з допомогою класу AdRequest, який будується з допомогою методу Build(). Після цього реклама завантажується із викликом LoadAd(adRequest).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using GoogleMobileAds.Api;

public class InterAd : MonoBehaviour
{
    private InterstitialAd interstitialAd;

    private string interstitialUnitId = "ca-app-pub-3940256099942544/1033173712";

    private void OnEnable()
    {
        interstitialAd = new InterstitialAd(interstitialUnitId);
        AdRequest adRequest = new AdRequest.Builder().Build();
        interstitialAd.LoadAd(adRequest);
    }

    public void ShowAd()
    {
        if (interstitialAd.IsLoaded())
            interstitialAd.Show();
```

Рисунок 3.6 – Скріпт для міжсторінкової реклами

Скрипт також включає метод ShowAd, який дозволяє перевірити, чи завантажена реклама за допомогою методу IsLoaded() і відобразити її, якщо вона доступна, викликавши метод Show(). Цей метод може бути викликаний з інших частин програми, наприклад, при завершенні рівня або перед зміною сцени (рис. 3.7).

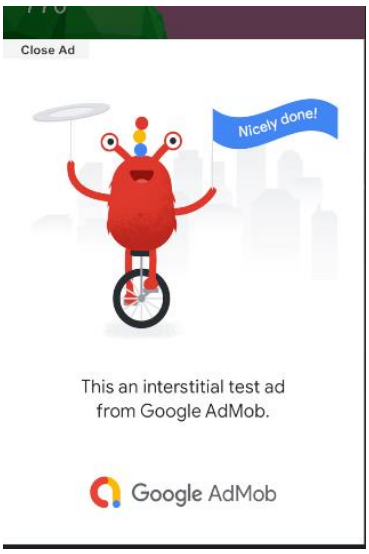


Рисунок 3.7 – Тестова реклама

Реклама з винагородою буде реалізована на сцені «Menu» за допомогою кнопки і після натискання на неї з'являтиметься реклама. Для неї так само потрібен скрипт, який її запускатиме (рис 3.8).

```
public class RewAd : MonoBehaviour
{
    private string RewardedUnitId = "ca-app-pub-3940256099942544/5224354917";

    private RewardedAd rewardedAd;
    private void OnEnable()
    {
        rewardedAd = new RewardedAd(RewardedUnitId);
        AdRequest adRequest = new AdRequest.Builder().Build();
        rewardedAd.LoadAd(adRequest);
        rewardedAd.OnUserEarnedReward += HandleUserEarnedReward;
    }

    private void HandleUserEarnedReward(object sender, Reward e)
    {
        int coins = PlayerPrefs.GetInt("coins");
        coins += 150;
        PlayerPrefs.SetInt("coins", coins);
    }

    public void ShowAd()
    {
        if (rewardedAd.IsLoaded())
            rewardedAd.Show();
    }
}
```

Рисунок 3.8 – Скрипт для реклами з винагородою

Цей скрипт реалізує функціональність рекламу з винагородаю «Rewarded Ads» (рис 3.8). Обробка подій Клас RewAd містить змінні для ідентифікатора рекламного блоку RewardedUnitId (у даному випадку використовується тестовий ID) та об'єкта винагороджуваної реклами rewardedAd. При активації об'єкта, на якому знаходиться даний скрипт (метод OnEnable), створюється новий екземпляр RewardedAd із

використанням ідентифікатора рекламного блоку. після успішного перегляду реклами обробник HandleUserEarnedReward.

Метод HandleUserEarnedReward додає користувачеві 150 монет після перегляду реклами. Для цього використовується система PlayerPrefs, в якій поточний баланс монет витягується, збільшується на 150 і зберігається назад.

Метод ShowAd перевіряє, чи завантажується реклама за допомогою методу IsLoaded() та відображає її викликом методу Show(), якщо реклама доступна.

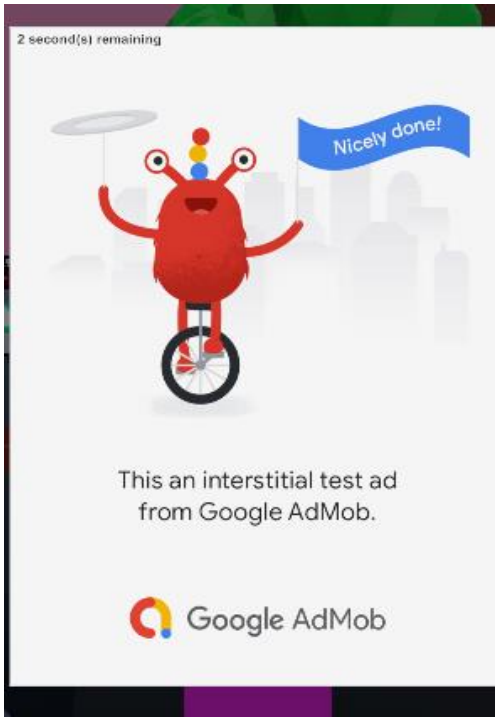


Рисунок 3.8 – Реклама з винагородою

ВИСНОВКИ

У межах кваліфікаційної роботи було створено сценарій та реалізовано гру в жанрі "ранер" на платформі Unity. У ході розробки були вивчені й застосовані ключові концепції та інструменти, які використовуються при створенні ігор у Unity, зокрема: розробка ігрових об'єктів, додавання компонентів, налаштування фізичних параметрів, застосування анімацій, написання скриптів на мові C# та інші аспекти розробки.

Особлива увага приділялася впровадженню можливостей монетизації, таких як реклама, бонуси за перегляд реклами, що відповідає сучасним тенденціям розвитку мобільних ігор. У процесі роботи було використано рушій Unity, що дозволило ефективно реалізувати графіку, анімації та інтерактивність гри, а також забезпечити кросплатформенність.

Розглянуто деякі концепції, що використовуються в іграх Ранер, такі як генерація рівнів, набір очок. Створення гри Ранер на Unity – це цікавий і захоплюючий процес, який дозволяє отримати досвід у роботі з ігровими концепціями та інструментами, а також навчитися використовувати різні аспекти розробки ігор, такі як анімації, звукові та візуальні ефекти, управління ігровими об'єктами та багато інші.

ПЕРЕЛІК ПОСИЛАНЬ

1. Руан Лоттер. Blender: новий рівень майстерності. Івано-Франківськ : Print2print, 2023. 452 с.

2. Jason Van Gamster. Blender All-in-One For Dummies. New Jersey : Wiley. John Wiley & Sons, LTD, 2024. 768 p.

3. Ryan Holiday. Growth Hacker Marketing. London: Profile Books. 2014. 144 p.

4. Joe Hocking. Unity in Action: Multiplatform Game Development in C#. New York : Manning Publications, 2015. 352 p.

5. А. Ковальов. Unity. Розробка ігор для мобільних пристроїв. Київ : English Student, 2021. 496 с.

6. Офіційний форум від Unity для створення анімацій персонажа. URL : https://discussions.unity.com/lists/animation (дата звернення 15.10.2024).

7. Лаптон Е., Коул-Філліпс Д. Графічний дизайн. Нові основи. Київ : ArtHuss, 2020. 264 с.

8. Ресурс IT-форум. URL : https://dou.ua/forums/ (дата звернення 06.11.2024).

9. McLean K., Sharp D. Games, Design and Play: A Detailed Approach to Iterative Game Design. Boston : Addison-Wesley Professional. 2016. 288 p.

10. Cleary S. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming. California : O'Reilly Media. 2019. 251 p.