ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

КВАЛІФІКАЦІЙНА РОБОТА

«Розробка ігрового застосунку в жанрі Casual
на основі патерну MVC та технології Unity»

Виконав:
ст. гр. КІУКІ-21-5 Мазуренко О.Е.

Керівник: ст.викл. Фомічов О.О.

АКТУАЛЬНІСТЬ ТЕМИ

- Жанр runner є стабільно популярним у мобільному сегменті ігрової індустрії завдяки простому управлінню та швидкому геймплею;
- Створення нескладної, але функціональної гри дає змогу дослідити ключові аспекти архітектури, оптимізації й взаємодії між компонентами в ігровому застосунку;
- Використання шаблону MVC забезпечує модульність, зручність супроводу й можливість масштабування коду, що відповідає сучасним вимогам до структури програмного забезпечення;
- Реалізація ігрової логіки в Unity дозволяє інтегрувати графіку, анімацію, фізику та звук в єдине середовище без потреби у сторонніх рушіях.

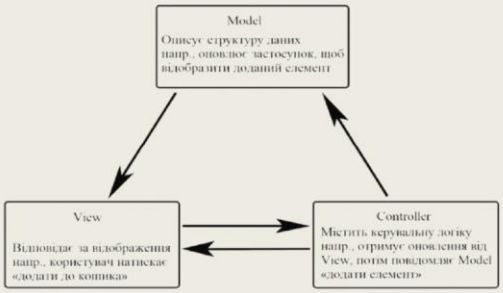
Аналіз жанру Casual та Runner

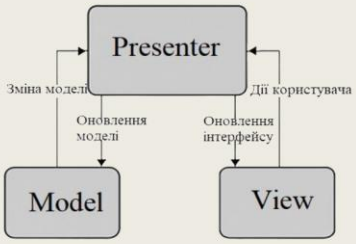
- Жанр Casual орієнтований на широку аудиторію, характеризується простим управлінням, короткими ігровими сесіями та невисоким порогом входу;
- Runner-ігри є піджанром casual, у якому персонаж автоматично рухається вперед, а гравець реагує на перешкоди шляхом свайпів або натискань;
- Основні механіки: ухилення, стрибки, бонуси, поступове ускладнення, нескінченне середовище;
- Простота реалізації та масштабування робить жанр зручним для індивідуальної або навчальної розробки в середовищі Unity.

Аналіз ігрових рушіїв

- Ігрові рушії — це програмні платформи, які забезпечують візуалізацію, фізику, обробку введення та інші базові компоненти гри;
- Найпоширеніші рушії: Unity, Unreal Engine, Godot — кожен має власні переваги та сферу застосування;
- Unity один із найпопулярніших рушіїв для 2D/3D ігор, має велику базу активів, підтримку C# і просту інтеграцію з платформами;
- Unreal Engine пропонує високий графічний рівень і використовує мову C++, проте має вищий поріг входу;
- Godot є відкритим рушієм з власною мовою GDScript, орієнтований на інді-розробників і легкі проекти;
- Вибір рушія залежить від цілей, складності проєкту, команди розробки та цільової платформи.

Порівняння архітектурних патернів





Рушій Unity

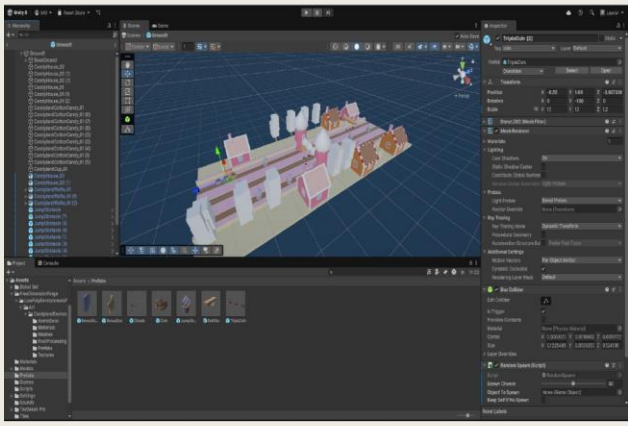
•Unity — один із найпопулярніших рушіїв для створення 2D- та 3D-ігор, підтримує кросплатформену розробку;

•Має зручний редактор сцен, систему компонентів та візуальне редагування інтерфейсу;

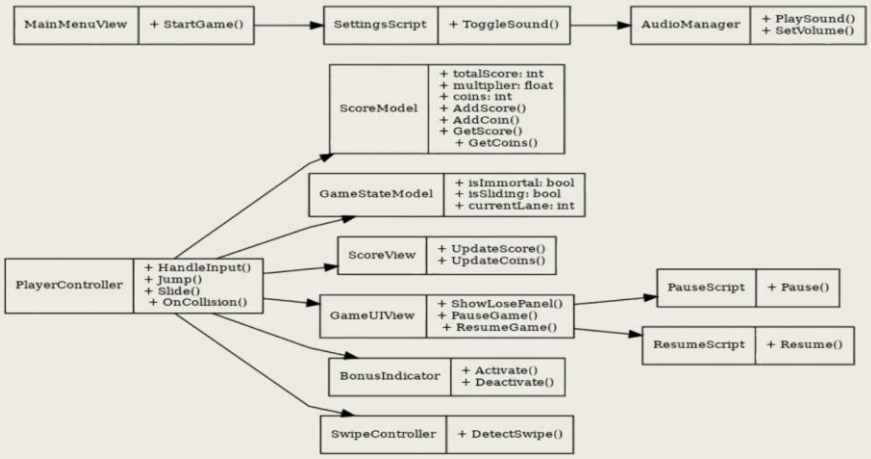
•Основною мовою програмування є C#, що поєднує високу продуктивність із простотою синтаксису;

•Велика база готових рішень і ассетів у Unity Asset Store прискорює розробку;

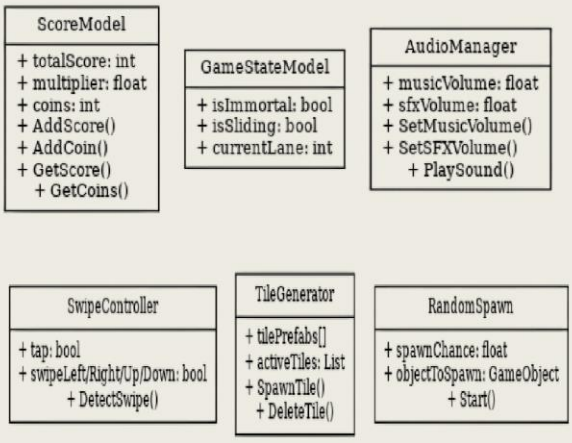
•Гнучка структура дозволяє реалізовувати як прості, так і складні архітектурні патерни, зокрема MVC.



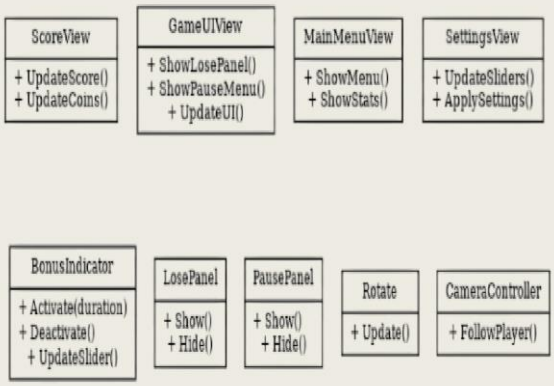
Архітектура проекту



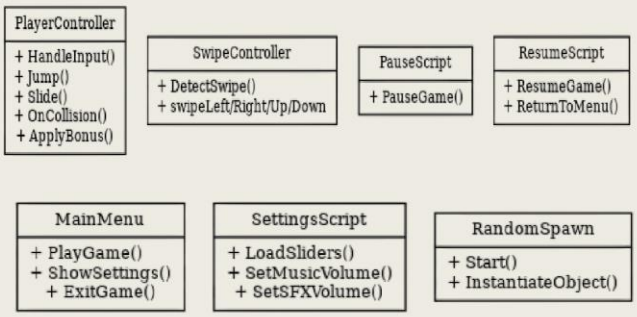
Model частина



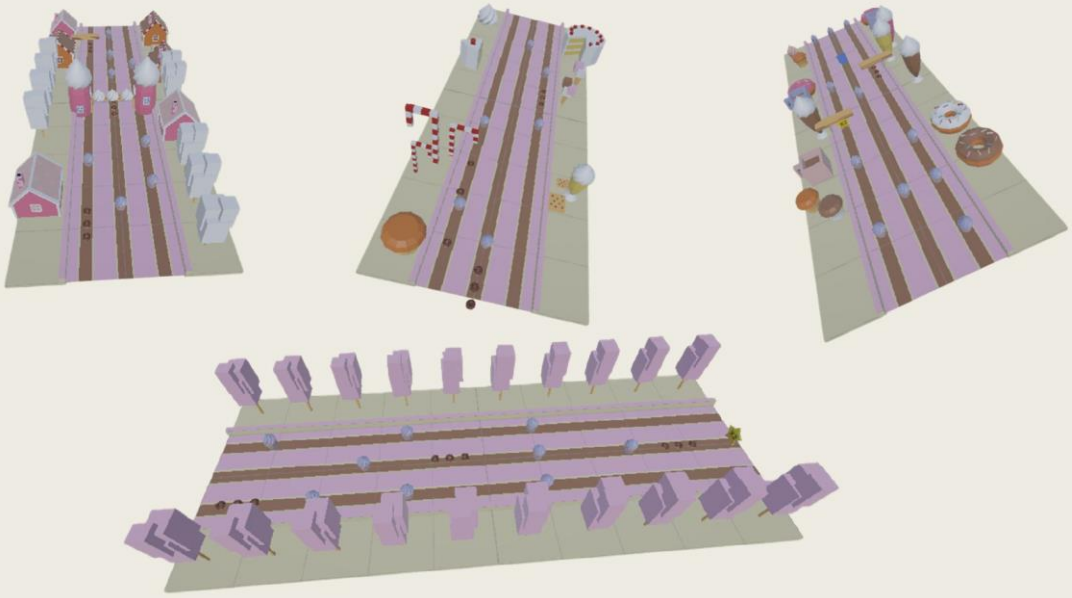
View частина



Controller частина

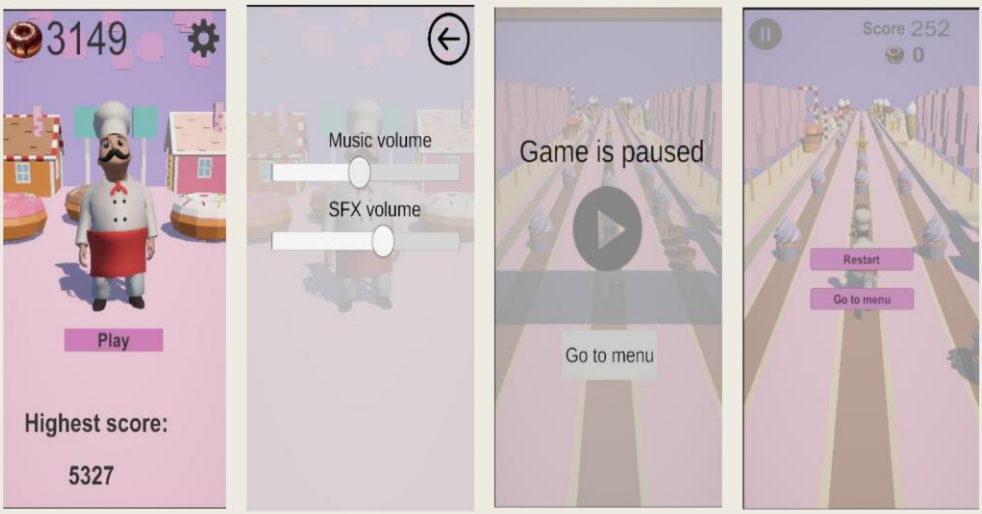


Генерація рівня



Управління персонажем

Інтерфейс користувача



Висновки

- У межах проєкту було реалізовано гру в жанрі runner з архітектурою MVC;
- Використання шаблону MVC забезпечило чітке розділення логіки, інтерфейсу та даних;
- Реалізовано управління персонажем, динамічну генерацію рівня, систему бонусів і підрахунок очок;
- Створено повноцінний інтерфейс користувача з меню, налаштуваннями, паузою та екраном поразки;
- Готовий прототип є основою для подальшого розвитку;
- Робота дозволила закріпити практичні навички в Unity, C# та організації ігрової архітектури.

ДОДАТОК Б

Лістинг 1.1

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

public class PlayerController : MonoBehaviour
{
    private CharacterController controller;
    private CapsuleCollider col;
    private Animator anim;
    private Vector3 dir;
    private ScoreModel score;
    private Vector3 originalColliderCenter;
    private float originalColliderHeight;
    private float originalColliderRadius;
    private Vector3 originalControllerCenter;
    private float originalControllerHeight;
    private int sessionCoins = 0;
    private AudioSource takeSound;
    public AudioSource backgroundMusic;

    [SerializeField] private float speed;
    [SerializeField] private float jumpForce;
    [SerializeField] private float gravity;
    [SerializeField] private int coins;
    [SerializeField] private GameObject losePanel;
    [SerializeField] private GameObject scoreText;
    [SerializeField] private Text coinText;
    [SerializeField] private ScoreModel scoreScript;

    private bool isSliding;
    private bool isImmortal;

    private int lineToMove = 1;
    public float lineDistance = 4;
    private float maxSpeed = 85;

    void Start()
    {
        anim = GetComponentInChildren<Animator>();
        controller = GetComponent<CharacterController>();
        col = GetComponent<CapsuleCollider>();
        Time.timeScale = 1;
        score = scoreText.GetComponent<ScoreModel>();
        score.scoreMultiplier = 1;
        coins = PlayerPrefs.GetInt("coins");
        coinText.text = coins.ToString();
        StartCoroutine(SpeedIncrease());

        originalColliderCenter = col.center;
```

```
        originalColliderHeight = col.height;
        originalColliderRadius = col.radius;

        originalControllerCenter = controller.center;
        originalControllerHeight = controller.height;

        sessionCoins = 0;
        coinText.text = sessionCoins.ToString();

        takeSound = GetComponent<AudioSource>();
        isImmortal = false;
    }

    private void Update()
    {
        if (!isSliding)
        {
            col.center = originalColliderCenter;
            controller.center = originalControllerCenter;
        }

        if (SwipeInputController.swipeRight && lineToMove < 2)
            lineToMove++;

        if (SwipeInputController.swipeLeft && lineToMove > 0)
            lineToMove--;

        if (SwipeInputController.swipeUp &&
controller.isGrounded)
            Jump();

        if (SwipeInputController.swipeDown && !isSliding)
        {
            if (!controller.isGrounded)
                StartCoroutine(SmoothFallToGround());

            StartCoroutine(Slide());
        }

        anim.SetBool("isRunning", controller.isGrounded &&
!isSliding);

        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;

        if (transform.position != targetPosition)
        {
            Vector3 diff = targetPosition - transform.position;
            Vector3 moveDir = diff.normalized * 25 *
```

```
Time.deltaTime;
            controller.Move(moveDir.sqrMagnitude <
diff.sqrMagnitude ? moveDir : diff);
        }
    }

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

    private void RaiseColliderForJump()
    {
        float offsetY = 0.2f;

        col.center = new Vector3(
            originalColliderCenter.x,
            originalColliderCenter.y + offsetY,
            originalColliderCenter.z
        );

        controller.center = new Vector3(
            originalControllerCenter.x,
            originalControllerCenter.y + offsetY,
            originalControllerCenter.z
        );

        Invoke(nameof(ResetColliderPosition), 0.6f);
    }

    private void ResetColliderPosition()
    {
        if (!isSliding)
        {
            col.center = originalColliderCenter;
            controller.center = originalControllerCenter;
        }
    }

    void FixedUpdate()
    {
        dir.z = speed;
        dir.y += gravity * Time.fixedDeltaTime;
        controller.Move(dir * Time.fixedDeltaTime);
    }

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

```
            if (isImmortal)
            {
                Destroy(hit.gameObject);
                isImmortal = false;
            }
            else
            {
                backgroundMusic.Stop();
                losePanel.SetActive(true);
                int lastRunScore = scoreScript.CurrentScore;
                int bestScore = PlayerPrefs.GetInt("bestScore",
0);

                if (lastRunScore > bestScore)
                    PlayerPrefs.SetInt("bestScore",
lastRunScore);

                Time.timeScale = 0;
            }
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("coin"))
        {
            coins++;
            sessionCoins++;
            PlayerPrefs.SetInt("coins", coins);
            coinText.text = sessionCoins.ToString();
            takeSound.Play();
            Destroy(other.gameObject);
        }

        if (other.gameObject.CompareTag("BonusStar"))
        {
            StartCoroutine(StarBonus());
            Destroy(other.gameObject);
        }

        if (other.gameObject.CompareTag("BonusShield"))
        {
            StartCoroutine(ShieldBonus());
            Destroy(other.gameObject);
        }
    }

    private IEnumerator SpeedIncrease()
    {
        yield return new WaitForSeconds(4);
        if (speed < maxSpeed)
        {
            speed += 0.5f;
```

```
            StartCoroutine(SpeedIncrease());
        }
    }

    private IEnumerator Slide()
    {
        isSliding = true;

        anim.ResetTrigger("isJumping");
        anim.SetBool("isRunning", false);
        anim.SetTrigger("isSliding");

        float duration = 1.0f;

        col.height = originalColliderHeight / 2f;
        col.radius = originalColliderRadius / 1.5f;

        controller.height = originalControllerHeight;

        col.center = originalColliderCenter;
        controller.center = originalControllerCenter;

        yield return new WaitForSeconds(duration);

        col.height = originalColliderHeight;
        col.radius = originalColliderRadius;
        controller.height = originalControllerHeight;

        col.center = originalColliderCenter;
        controller.center = originalControllerCenter;

        isSliding = false;
    }

    private IEnumerator StarBonus()
    {
        score.scoreMultiplier = 2;
        yield return new WaitForSeconds(5);
        score.scoreMultiplier = 1;
    }

    private IEnumerator ShieldBonus()
    {
        isImmortal = true;
        yield return new WaitForSeconds(30);
        isImmortal = false;
    }

    private IEnumerator SmoothFallToGround()
    {
        float duration = 0.5f;
        float elapsed = 0f;
        float startY = transform.position.y;
```

```
        while (elapsed < duration)
        {
            elapsed += Time.deltaTime;
            float moveY = -startY * Time.deltaTime / duration;
            controller.Move(new Vector3(0, moveY, 0));
            dir.y = 0;
            yield return null;
        }

        dir.y = 0;
    }
}
```

Лістинг 1.2

```
using UnityEngine;

public class SwipeInputController : 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;

        if (Input.GetMouseButtonDown(0))
        {
            tap = true;
            isDraging = true;
            startTouch = Input.mousePosition;
        }
        else if (Input.GetMouseButtonUp(0))
        {
            isDraging = false;
            Reset();
        }

        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();
            }
        }

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

        if (swipeDelta.magnitude > 100)
        {
            float x = swipeDelta.x;
            float y = swipeDelta.y;

            if (Mathf.Abs(x) > Mathf.Abs(y))
            {
                if (x < 0)
                    swipeLeft = true;
                else
                    swipeRight = true;
            }
            else
            {
                if (y < 0)
                    swipeDown = true;
                else
                    swipeUp = true;
            }

            Reset();
        }
    }

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

Лістинг 1.3

```
using UnityEngine;
using UnityEngine.SceneManagement;
```

```
using UnityEngine.UI;

public class GameUIView : MonoBehaviour
{
    [SerializeField] Text recordText;

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

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

Лістинг 1.4

```
using UnityEngine;
using UnityEngine.UI;

public class ScoreModel : MonoBehaviour
{
    [SerializeField] private Transform player;
    [SerializeField] public Text scoreText;
    private int totalScore;

    public int scoreMultiplier;
    public int CurrentScore => totalScore;

    private void FixedUpdate()
    {
        totalScore += scoreMultiplier;
        scoreText.text = totalScore.ToString();
    }
}
```

Лістинг 1.5

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

public class TileGenerator : MonoBehaviour
{
    public GameObject[] tilePrefabs;
    private List<GameObject> activeTiles = new
List<GameObject>();
    private float spawnPos = 0;
    private float tileLength = 100;
    [SerializeField] private GameObject initialTile;
```

```
    [SerializeField] private Transform player;
    private int startTiles = 6;

    void Start()
    {
        activeTiles.Add(initialTile);
        spawnPos += tileLength;

        for (int i = 1; i < startTiles; i++)
        {
            SpawnTile(Random.Range(0, tilePrefabs.Length));
        }
    }

    void Update()
    {
        if (player.position.z - 100 > spawnPos - (startTiles *
tileLength))
        {
            SpawnTile(Random.Range(0, tilePrefabs.Length));

            if (activeTiles.Count >= startTiles - 1)
            {
                DeleteTile();
            }
        }
    }

    private void SpawnTile(int tileIndex)
    {
        GameObject nextTile =
Instantiate(tilePrefabs[tileIndex], transform.forward *
spawnPos, transform.rotation);
        activeTiles.Add(nextTile);
        spawnPos += tileLength;
    }

    private void DeleteTile()
    {
        Destroy(activeTiles[0]);
        activeTiles.RemoveAt(0);
    }
}
```

Лістинг 1.6

```
using UnityEngine;

public class RandomSpawn : MonoBehaviour
{
    [Range(0f, 100f)]
    public float spawnChance = 50f;
    public GameObject objectToSpawn;
```

```
    public bool keepSelfIfNoSpawn = false;

    void Start()
    {
        float roll = Random.Range(0f, 100f);
        if (roll <= spawnChance)
        {
            if (objectToSpawn != null)
            {
                Instantiate(objectToSpawn, transform.position,
transform.rotation);
            }
        }
        else
        {
            if (!keepSelfIfNoSpawn)
            {
                Destroy(gameObject);
            }
        }
    }
}
```

Лістинг 1.7

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

public class CameraView : MonoBehaviour
{
    [SerializeField] private Transform player;
    private Vector3 offset;

    void Start()
    {
        offset = transform.position - player.position;
    }

    void FixedUpdate()
    {
        Vector3 newPosition = new Vector3(transform.position.x,
transform.position.y, offset.z + player.position.z);
        transform.position = newPosition;
    }
}
```

Лістинг 1.8

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MainMenuView : MonoBehaviour
{
```

```
    [SerializeField] private Text coinsText;
    [SerializeField] private Text bestScoreText;

    private void Start()
    {
        int coins = PlayerPrefs.GetInt("coins");
        coinsText.text = coins.ToString();

        int bestScore = PlayerPrefs.GetInt("bestScore", 0);
        bestScoreText.text = bestScore.ToString();
    }

    public void PlayGame()
    {
        SceneManager.LoadScene(0);
    }
}
```