

Создание мобильной игры «Найди отличия» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель данной статьи – создание мобильной игры «Найди отличия» для android. Для разработки игры использовалась кроссплатформенная платформа, предоставляющая мощный физический движок и удобные визуальные инструменты, что делает её эффективным решением для создания 2D-игр. В результате работы была создана мобильная игра «Найди отличия»

Ключевые слова: Unity, игра, отличия, поиск.

Creating a Find the Differences mobile game in Unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create a mobile game "Find the Differences" for android. To develop the game, a cross-platform platform was used, which provides a powerful physics engine and convenient visual tools, making it an effective solution for creating 2D games. As a result of the work, a mobile game "Find the Differences" was created.

Keywords: Unity, game, differences, search.

1 Введение

1.1 Актуальность

В современной цифровой среде мобильные игры прочно вошли в повседневную жизнь, представляя собой не только массовый вид развлечения, но и динамично развивающуюся многомиллиардную индустрию. Особой популярностью пользуются казуальные головоломки, которые сочетают интуитивно понятный интерфейс, минималистичный дизайн и геймплей, способствующий развитию когнитивных функций.

Разработка визуальной головоломки «Найди отличия», основанной на механике сравнения парных изображений, полностью соответствует актуальным трендам игрового рынка. Подобные игры востребованы широкой аудиторией, так как способствуют развитию внимания к деталям, зрительной памяти, концентрации и скорости восприятия. Их ценность признана не только в сфере развлечений, но и в образовательном и реабилитационном контексте,

где они используются в качестве тренажеров для улучшения наблюдательности у разных возрастных групп.

Использование игрового движка Unity для реализации данного проекта является современным и практичным решением. Unity предоставляет мощный, гибкий и доступный инструментарий для быстрой разработки кроссплатформенных приложений. Это позволяет сосредоточиться на качестве контента (создании уровней и графики) и удобстве игрового процесса, обеспечивая при этом высокую производительность и простую публикацию во всех основных магазинах приложений.

Таким образом, создание мобильной игры «Найди отличия» на Unity представляет собой актуальную задачу, находящуюся на пересечении востребованного игрового жанра, образовательно-развивающего потенциала и эффективного применения современных технологий разработки. Проект обладает четкой практической значимостью и соответствует запросам рынка на интеллектуальный, расслабляющий и развивающий концентрацию контент.

1.2 Обзор исследований

А. Е. Унарова представила разработку мобильной игры «Кустук» на Unity для дошкольников и младших школьников на якутском языке [1]. Посвятил статью обзору ключевых этапов, необходимых для разработки компьютерных игр с двухмерной графикой на платформе Unity в жанре приключения с элементами головоломок Я.А.Фаст [2]. Н.В.Максимов рассматривает метод разработки мобильной 2D игры «Bootur Jump» на платформе Unity [3]. Показал процесс разработки двухмерной платформенной игры на игровом движке Unity с использованием языка программирования C# Любич А.А. [4]. Д.В. Морозова в рамках работы разработала развивающую 2D игра-платформер для детей, написанная на Unity [5].

1.3 Цель исследования

Цель исследования – создание мобильной игры «Найди отличия» для ОС Android при помощи кроссплатформенной среды для создания интерактивных 2D и 3D приложений Unity.

2 Материалы и методы

В качестве инструментария для разработки игры «Найди отличия» был выбран движок Unity. Это решение обусловлено целым рядом преимуществ: доступностью для разработчиков, интуитивно понятным интерфейсом и поддержкой языка C#, что делает его оптимальным для реализации игровой логики. Написание кода производилось в среде разработки Visual Studio Community.

3 Результаты

Первым этапом разработки мобильной игры «Найди отличия» является инициализация проекта. В менеджере проектов Unity Hub выбирается специализированный шаблон для 2D — Universal Render Pipeline, после чего указывается название проекта и запускается процесс его генерации (см. рис. 1).

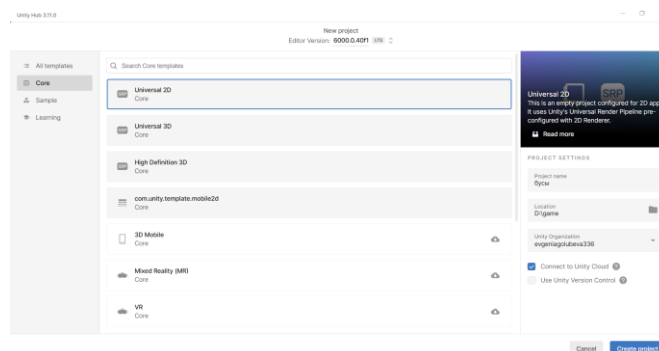


Рисунок 1 – Создание проекта

После создания проекта на основе шаблона URP (2D) его необходимо настроить для мобильных платформ. Для этого в меню File → Build Settings следует выбрать Android в качестве целевой платформы (см. рис. 2).

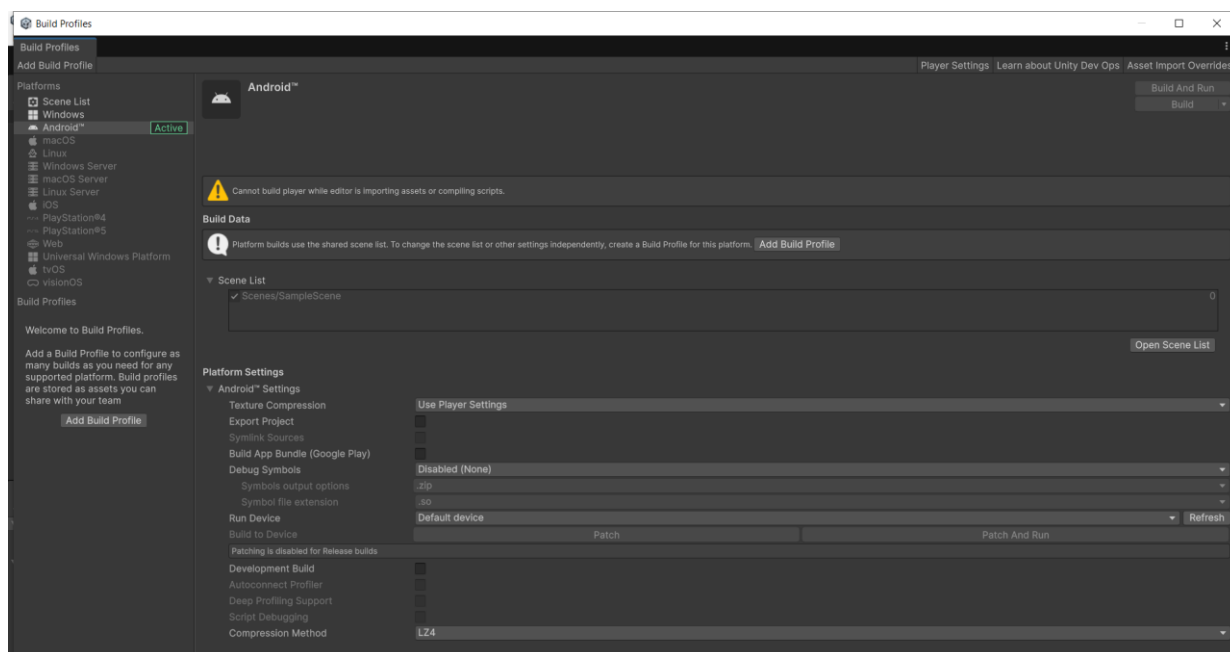


Рисунок 2 – Настройка проекта

Следующим шагом является создание основной игровой сцены, и начинается он с настройки камеры (см. рис. 3).

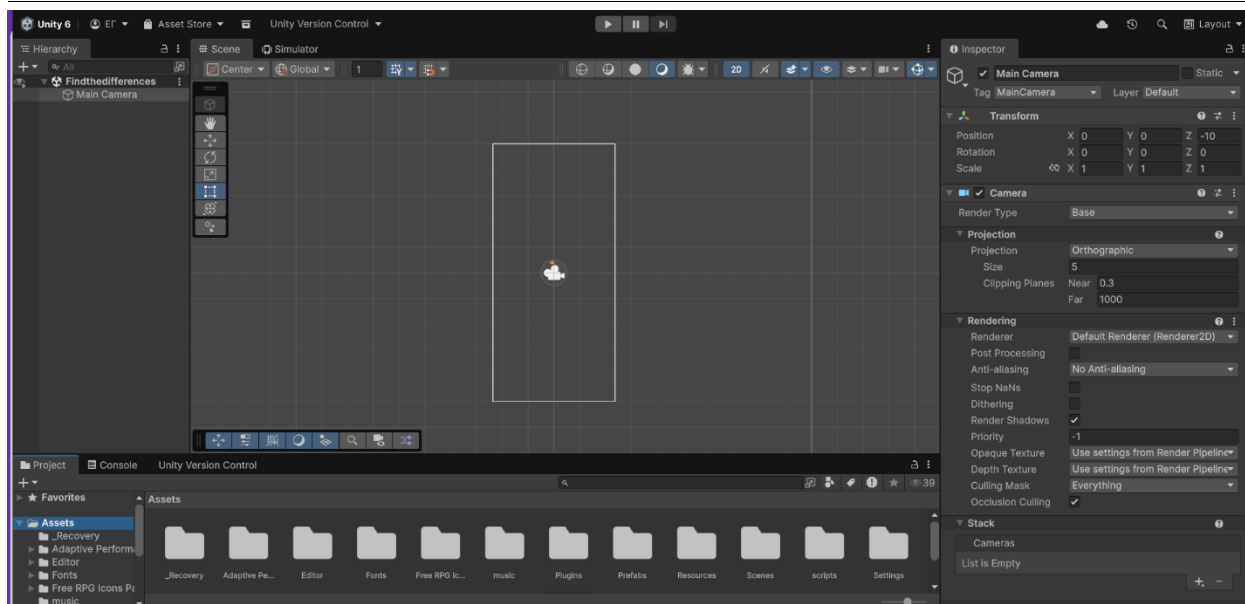


Рисунок 3 – Настройка камеры

В папку Sprites необходимо добавить заранее подготовленные изображения для игры (см.рис.4).

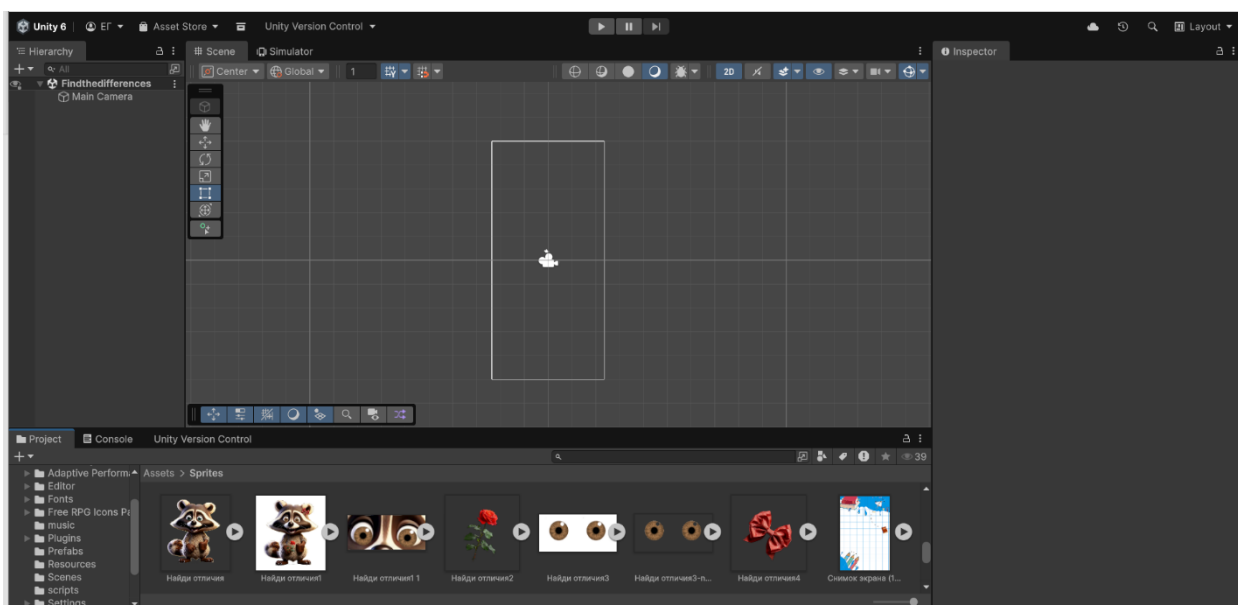


Рисунок 4 – Добавление изображений

Следующим шагом является создание фона для игровой сцены. Для этого необходимо переместить изображение background из папки Sprites в сцену и настроить его параметры трансформации и отрисовки для полного покрытия игрового окна (см.рис.5).

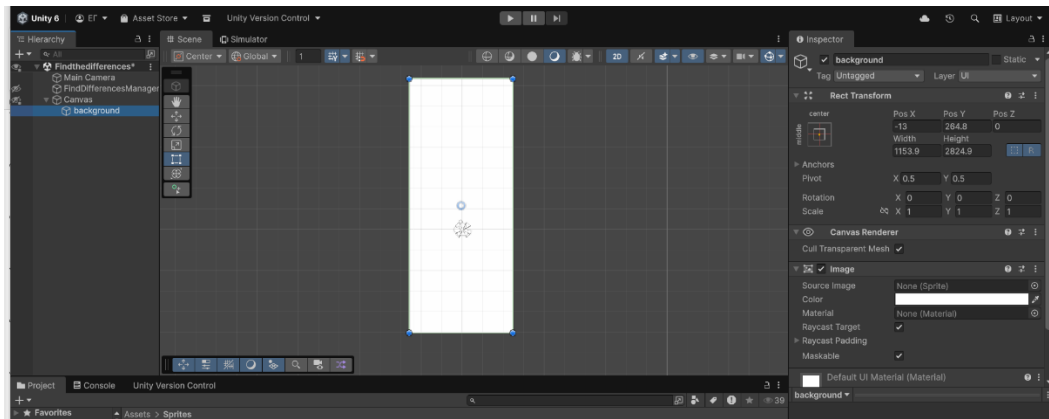


Рисунок 5 – Настройка фона игры

Следующим шагом добавляем два изображения енота, на которых необходимо найти отличия. Для этого перемещаем изображения из папки Sprites на сцену и настраиваем их параметры (см.рис.6).

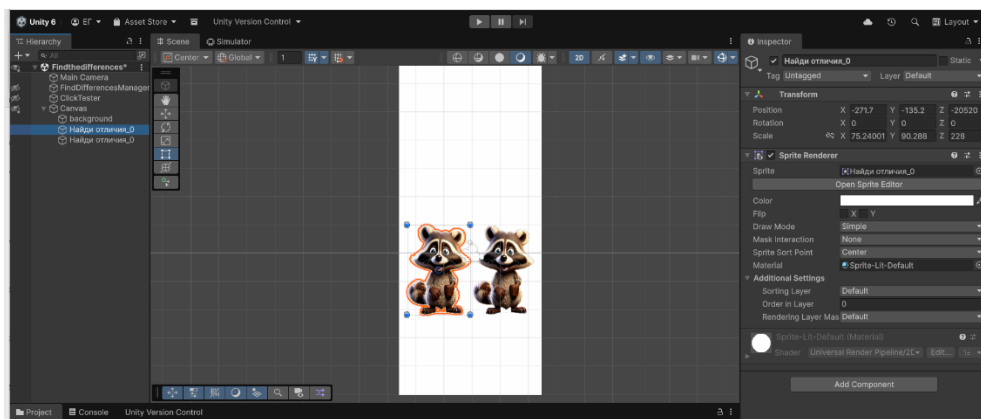


Рисунок 6 – Добавление изображений на сцену

Далее на одно изображения енота необходимо добавить отличия. Для этого требуется создать три пустых объекта, которые будут служить контейнерами для предметов. В Hierarchy нажимаем правой кнопкой мыши, и в контекстном меню выбираем Create empty и переименовываем в Card_1, Card_2 и Card_3. После создание объектов необходимо их настроить (см.рис.7).

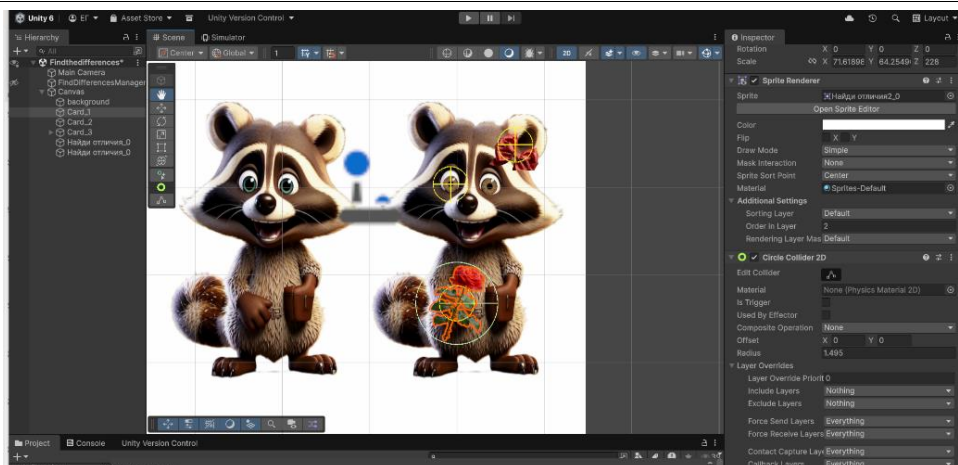


Рисунок 7 – Добавление предметов

Для отображения сообщений необходимо создать в сцене панель (UI -> Panel) как фоновую подложку и текстовый блок (UI -> Text). Панель настраивается в Inspector: устанавливаются центральные якоря (0.5, 0.5) для адаптивности, задаются размеры 1420x601.8 пикселей, выбирается спрайт и настраивается его цвет и прозрачность в компоненте Image, а тип отображения устанавливается в Simple (см.рис.8)

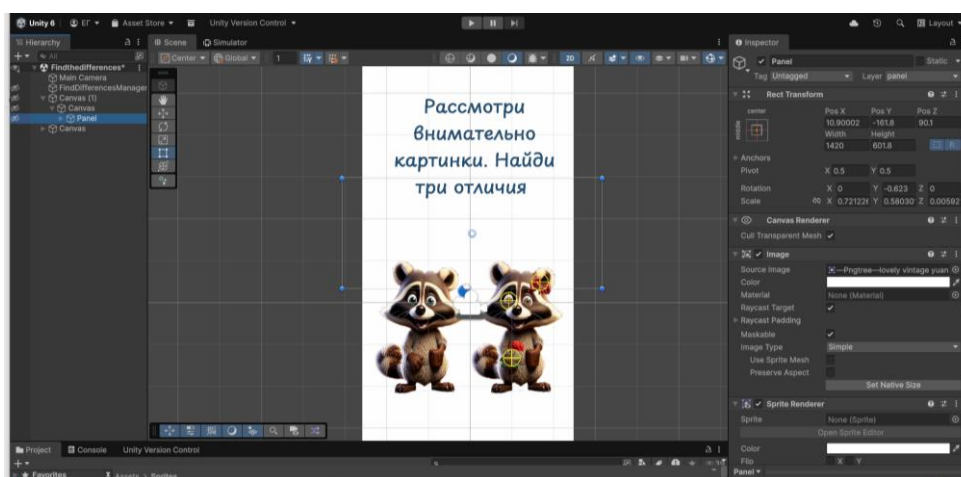


Рисунок 8 – Настройка панели

После этого выполняется настройка текстового элемента для вывода уведомления «Правильно!». В компоненте Text (Legacy) выбирается шрифт PlaypenSans-VariableFont_wght с жирным начертанием (Bold) и устанавливается размер 100 пунктов, что обеспечивает хорошую видимость и читаемость. В разделе Rect Transform задаются точные размеры текстового поля: ширина 1211.9 и высота 601.8. Также производится важная настройка якорей (Anchors) — их центральная точка устанавливается в позиции (0.5, 0.5), что гарантирует стабильное центрированное отображение текста на экранах любых разрешений (см.рис. 9).

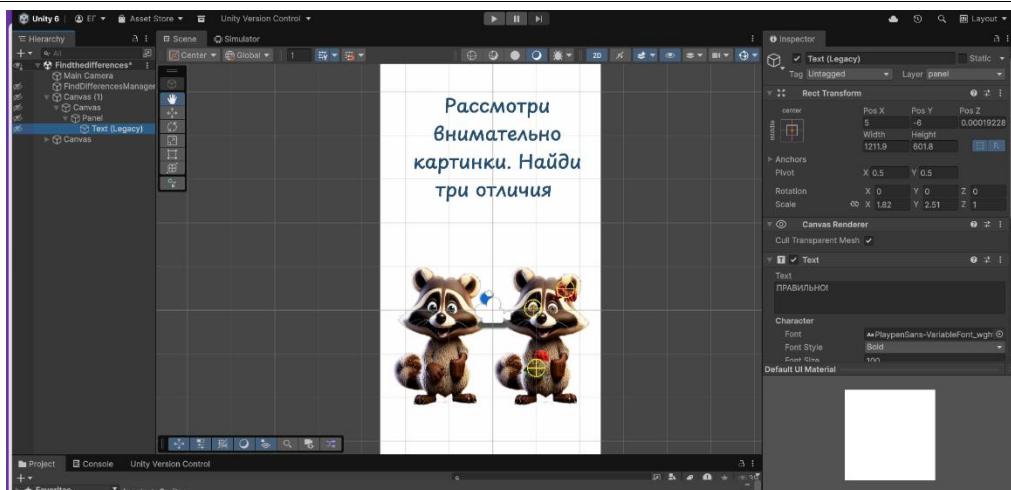


Рисунок 9 – Настройка текстового элемента панели сообщений

Для текстового элемента настраиваем шрифт Pangolin-Regular с жирным начертанием и размером 140 пунктов с жирным начертанием Bold, выравнивание по центру, включаем поддержку Rich Text для форматирования, а в Rect Transform задаем точные размеры текстового поля: Width 1437,2, Height 927,76 и устанавливаем якоря Anchors с центром в точке (0.5, 0.5) (см.рис.10).

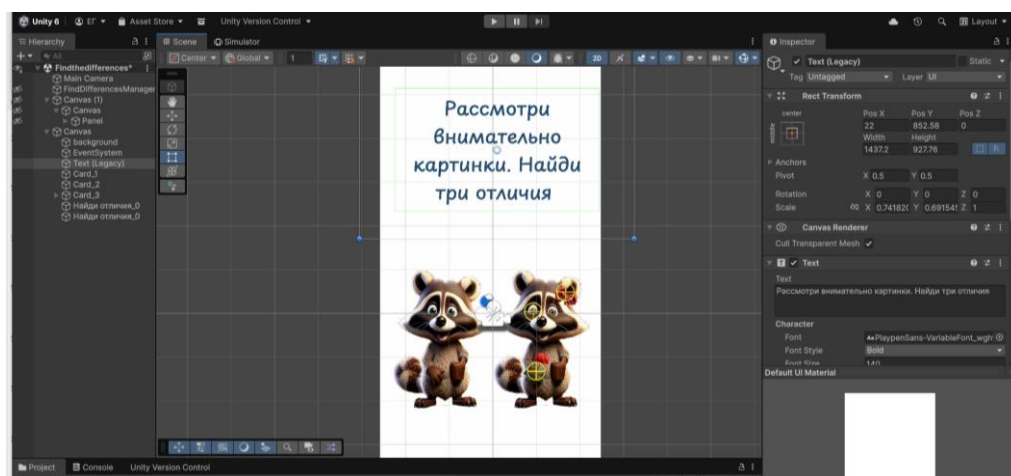


Рисунок 10 – Настройка инструкции игры

После добавления всех необходимых компонентов необходимо создать скрипт под названием differenceObjects. Скрипт differenceObjects.cs выступает основным управляющим компонентом игры, отвечающим за логику поиска отличий, обработку ввода и визуальную обратную связь. При запуске в методе Start() происходит инициализация: определяется камера, настраивается аудиосource, проверяются объекты-отличия из списка differenceObjects (при необходимости им добавляются коллайдеры), а также подготавливаются элементы интерфейса.

Обработка кликов и касаний происходит в Update(), где метод ProcessInput() преобразует экранные координаты в мировые и определяет выбранный объект с учётом исключения кликов по интерактивным

UI-элементам. Если объект является ненайденным отличием, метод `ProcessClickedObject()` добавляет его в список найденных, запускает анимированный круговой эффект и кратковременную подсветку, воспроизводит звук успеха и обновляет счётчик прогресса.

При обнаружении всех отличий автоматически запускается завершающая корутина `CompleteGame()`, которая воспроизводит звук победы, отображает и скрывает поздравительную панель, после чего возвращает игрока в главное меню. Для удобства отладки в скрипт встроена система логирования и контекстные меню, позволяющие тестировать отдельные функции в редакторе Unity.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using UnityEngine.EventSystems;
using UnityEngine.InputSystem;
using System.Collections;
using System.Collections.Generic;

public class FindDifferencesGame : MonoBehaviour
{
    [Header("Game Settings")]
    public List<GameObject> differenceObjects = new List<GameObject>();
    public Material highlightMaterial;
    public float highlightDuration = 0.5f;

    [Header("Circle Effect Settings")]
    public GameObject circleEffectPrefab;
    public float circleAnimationDuration = 0.5f;
    public float circleStartSize = 0.1f;
    public float circleFinalSize = 0.8f;
    public Color circleColor = new Color(1f, 0.5f, 0f, 0.8f);

    [Header("Input Settings")]
    public bool useMouseInput = true;
    public bool useTouchInput = true;

    [Header("UI Settings")]
    public RectTransform successPanel;
    public Text messageText;
    public Text progressText;
    public float messageShowTime = 2f;
    [Header("Audio Settings")]
    public AudioClip correctSound;
    public AudioClip wrongSound;
    public AudioClip winSound;

    [Header("Scene Management")]
    public string menuSceneName = "Menu";

    [Header("Debug Settings")]
    public bool debugMode = true;

    private int totalDifferences;
    private int foundDifferences = 0;
    private AudioSource audioSource;
    private CanvasGroup canvasGroup;
    private Dictionary<GameObject, Material> originalMaterials = new
Dictionary<GameObject, Material>();
    private List<GameObject> foundObjects = new List<GameObject>();
    private Camera mainCamera;
    private Dictionary<GameObject, GameObject> objectCircles = new
Dictionary<GameObject, GameObject>();
```

```
void Start()
{
    mainCamera = Camera.main;
    if (mainCamera == null)
    {
        Debug.LogError("Main Camera не найдена!");
        GameObject camObj = new GameObject("Main Camera");
        camObj.AddComponent<Camera>();
        camObj.tag = "MainCamera";
        mainCamera = camObj.GetComponent<Camera>();
    }
    totalDifferences = differenceObjects.Count;
    audioSource = GetComponent<AudioSource>();
    if (audioSource == null)
    {
        audioSource = gameObject.AddComponent<AudioSource>();
        audioSource.playOnAwake = false;
    }
    Log($"=== ИНИЦИАЛИЗАЦИЯ ИГРЫ ===", true);
    Log($"Всего отличий: {totalDifferences}", true);
    CreateCirclePrefabIfNeeded();
    InitializeDifferenceObjects();
    InitializeUI();

    UpdateProgress();

    StartCoroutine(DebugInfoCoroutine());
}

void Update()
{
    if (useMouseInput && Mouse.current != null &&
        Mouse.current.leftButton.wasPressedThisFrame)
    {
        ProcessInput(Mouse.current.position.ReadValue(), InputType.Mouse);
    }

    if (useTouchInput && Touchscreen.current != null)
    {
        for (int i = 0; i < Touchscreen.current.touches.Count; i++)
        {
            var touch = Touchscreen.current.touches[i];
            if (touch.press.wasPressedThisFrame)
            {
                ProcessInput(touch.position.ReadValue(), InputType.Touch);
                break;
            }
        }
    }
}

private void CreateCirclePrefabIfNeeded()
{
    if (circleEffectPrefab == null)
    {
        Log("Создаем префаб кругового эффекта...", true);

        GameObject circleObject = new GameObject("CircleEffectPrefab");

        SpriteRenderer spriteRenderer =
            circleObject.AddComponent<SpriteRenderer>();

        Texture2D circleTexture = CreateCircleTexture(256, circleColor);
        Sprite circleSprite = Sprite.Create(circleTexture,
            new Rect(0, 0, circleTexture.width, circleTexture.height),
            new Vector2(0.5f, 0.5f));

        spriteRenderer.sprite = circleSprite;
        spriteRenderer.sortingOrder = 999;

        circleEffectPrefab = circleObject;
    }
}
```

```
        circleObject.SetActive(false);

        Log("Префаб круга создан", true);
    }
}
private Texture2D CreateCircleTexture(int size, Color color)
{
    Texture2D texture = new Texture2D(size, size, TextureFormat.RGBA32, false);

    float center = size / 2f;
    float radius = size / 2f;
    float borderThickness = 10f;

    for (int x = 0; x < size; x++)
    {
        for (int y = 0; y < size; y++)
        {
            float distance = Vector2.Distance(new Vector2(x, y), new
Vector2(center, center));

            if (distance > radius - borderThickness && distance <= radius)
            {
                float alpha = 1f;
                if (distance > radius - 2f)
                {
                    alpha = 1f - ((distance - (radius - 2f)) / 2f);
                }
                texture.SetPixel(x, y, new Color(color.r, color.g, color.b,
alpha));
            }
            else
            {
                texture.SetPixel(x, y, Color.clear);
            }
        }
    }

    texture.Apply();
    return texture;
}

private void InitializeDifferenceObjects()
{
    Log("Настройка объектов-отличий:", true);

    for (int i = 0; i < differenceObjects.Count; i++)
    {
        GameObject obj = differenceObjects[i];

        if (obj == null)
        {
            Debug.LogError($"Объект-отличие {i} в списке равен null!");
            continue;
        }

        Log($"  Объект: {obj.name}", true);

        Renderer renderer = obj.GetComponent<Renderer>();
        if (renderer != null)
        {
            originalMaterials[obj] = renderer.material;
            Log($"    Материал: {renderer.material.name}", true);
        }
        else
        {
            Debug.LogWarning($"    У объекта {obj.name} нет Renderer!");
        }

        SetupCollider(obj);

        if (!obj.activeSelf)
```

```
        {
            Log($"    Объект был неактивен, активируем", true);
            obj.SetActive(true);
        }
    }
}

private void SetupCollider(GameObject obj)
{
    bool hasCollider = obj.GetComponent<Collider>() != null ||
        obj.GetComponent<Collider2D>() != null;

    if (hasCollider)
    {
        Log($"    Коллайдер уже есть", true);
        return;
    }

    SpriteRenderer spriteRenderer = obj.GetComponent<SpriteRenderer>();
    if (spriteRenderer != null)
    {
        BoxCollider2D collider2D = obj.AddComponent<BoxCollider2D>();
        if (spriteRenderer.sprite != null)
        {
            collider2D.size = spriteRenderer.sprite.bounds.size * 1.2f;
            Log($"    Добавлен BoxCollider2D, размер: {collider2D.size}", true);
        }
    }
    else
    {
        Renderer renderer = obj.GetComponent<Renderer>();
        if (renderer != null)
        {
            BoxCollider collider3D = obj.AddComponent<BoxCollider>();
            Log($"    Добавлен BoxCollider", true);
        }
        else
        {
            Debug.LogWarning($"    Не могу определить тип объекта для коллайдера:
{obj.name}");
        }
    }
}

private void InitializeUI()
{
    if (successPanel != null)
    {
        canvasGroup = successPanel.GetComponent<CanvasGroup>();
        if (canvasGroup == null)
            canvasGroup = successPanel.gameObject.AddComponent<CanvasGroup>();

        canvasGroup.alpha = 0;
        successPanel.gameObject.SetActive(false);
    }

    DisableBackgroundRaycast();
}

private void DisableBackgroundRaycast()
{
    Canvas[] allCanvases = FindObjectsByType<Canvas>(FindObjectsSortMode.None);

    foreach (Canvas canvas in allCanvases)
    {
        Image[] images = canvas.GetComponentsInChildren<Image>(true);

        foreach (Image image in images)
        {
            if (image.gameObject.name.Contains("Background") ||
                image.gameObject.name.Contains("Panel") ||
```

```
        image.gameObject.name.Contains("Фон"))
    {
        image.raycastTarget = false;
        Log($"Отключен Raycast Target у: {image.gameObject.name}", true);
    }
}

private void ProcessInput(Vector2 screenPosition, InputType inputType)
{
    Log($"{{inputType}} в позиции: {screenPosition}", true);

    if (IsPointerOverInteractiveUI(screenPosition))
    {
        Log(" Клик по интерактивному UI элементу (кнопка и т.д.), игнорируем",
true);
        return;
    }

    GameObject clickedObject = FindObjectAtPosition(screenPosition);

    if (clickedObject != null)
    {
        Log($" Найден объект: {clickedObject.name}", true);
        ProcessClickedObject(clickedObject);
    }
    else
    {
        Log(" Объект не найден", true);
        PlaySound(wrongSound);
    }
}

private bool IsPointerOverInteractiveUI(Vector2 screenPosition)
{
    if (EventSystem.current == null) return false;

    PointerEventData eventData = new PointerEventData(EventSystem.current);
    eventData.position = screenPosition;

    List<RaycastResult> results = new List<RaycastResult>();
    EventSystem.current.RaycastAll(eventData, results);

    if (results.Count == 0) return false;

    foreach (RaycastResult result in results)
    {
        Selectable selectable = result.gameObject.GetComponent<Selectable>();
        if (selectable != null && selectable.interactable)
        {
            Log($" Найден интерактивный UI: {result.gameObject.name}", true);
            return true;
        }

        Button button = result.gameObject.GetComponent<Button>();
        Toggle toggle = result.gameObject.GetComponent<Toggle>();
        Slider slider = result.gameObject.GetComponent<Slider>();

        if (button != null || toggle != null || slider != null)
        {
            Log($" Найден UI элемент с компонентом: {result.gameObject.name}",
true);
            return true;
        }
    }

    return false;
}

private GameObject FindObjectAtPosition(Vector2 screenPosition)
```

```

    {
        if (mainCamera == null) return null;

        if (mainCamera.orthographic)
        {
            Vector2 worldPoint = mainCamera.ScreenToWorldPoint(screenPosition);

            if (debugMode)
            {
                Debug.DrawRay(worldPoint, Vector2.up * 0.1f, Color.green, 1f);
                Debug.DrawRay(worldPoint, Vector2.right * 0.1f, Color.green, 1f);
            }

            RaycastHit2D hit2D = Physics2D.Raycast(worldPoint, Vector2.zero);

            if (hit2D.collider != null)
            {
                Log($" Попадание в 2D объект: {hit2D.collider.gameObject.name}",
true);
                return hit2D.collider.gameObject;
            }
        }
        else
        {
            Ray ray = mainCamera.ScreenPointToRay(screenPosition);

            if (debugMode)
            {
                Debug.DrawRay(ray.origin, ray.direction * 100, Color.red, 1f);
            }

            RaycastHit hit;
            if (Physics.Raycast(ray, out hit, Mathf.Infinity))
            {
                Log($" Попадание в 3D объект: {hit.collider.gameObject.name}", true);
                return hit.collider.gameObject;
            }
        }

        return null;
    }

private void ProcessClickedObject(GameObject clickedObject)
{
    if (differenceObjects.Contains(clickedObject))
    {
        if (!foundObjects.Contains(clickedObject))
        {
            foundObjects.Add(clickedObject);
            foundDifferences++;

            Log($" НОВОЕ ОТЛИЧИЕ НАЙДЕНО! Всего:
{foundDifferences}/{totalDifferences}", true);

            CreateCircleEffect(clickedObject);

            StartCoroutine(TemporaryHighlight(clickedObject));

            PlaySound(correctSound);

            UpdateProgress();

            if (foundDifferences >= totalDifferences)
            {
                StartCoroutine(CompleteGame());
            }
        }
        else
        {
            Log(" Это отличие уже было найдено ранее", true);
            PlaySound(wrongSound);
        }
    }
}

```

```

    }
    else
    {
        Log(" Это не отличие", true);
        PlaySound(wrongSound);
    }
}

private void CreateCircleEffect(GameObject targetObject)
{
    if (circleEffectPrefab == null)
    {
        Debug.LogWarning("Префаб круга не назначен!");
        return;
    }

    if (objectCircles.ContainsKey(targetObject))
    {
        Log($" У объекта {targetObject.name} уже есть круг", true);
        return;
    }

    GameObject circleEffect = Instantiate(circleEffectPrefab);
    circleEffect.SetActive(true);
    circleEffect.name = $"Circle_{targetObject.name}";

    circleEffect.transform.SetParent(targetObject.transform);
    circleEffect.transform.localPosition = Vector3.zero;

    circleEffect.transform.localScale = Vector3.one * circleStartSize;

    objectCircles[targetObject] = circleEffect;

    StartCoroutine(AnimateCircleAppearance(circleEffect));
}

private IEnumerator AnimateCircleAppearance(GameObject circleEffect)
{
    if (circleEffect == null) yield break;

    SpriteRenderer circleRenderer = circleEffect.GetComponent<SpriteRenderer>();
    if (circleRenderer == null) yield break;

    float elapsedTime = 0f;
    Vector3 startScale = Vector3.one * circleStartSize;
    Vector3 targetScale = Vector3.one * circleFinalSize;

    Color startColor = new Color(circleColor.r, circleColor.g, circleColor.b, 0f);
    Color targetColor = circleColor;

    circleEffect.transform.localScale = startScale;
    circleRenderer.color = startColor;

    while (elapsedTime < circleAnimationDuration)
    {
        if (circleEffect == null) yield break;

        float t = elapsedTime / circleAnimationDuration;
        float easedT = t * t;

        circleEffect.transform.localScale = Vector3.Lerp(startScale, targetScale,
easedT);
        circleRenderer.color = Color.Lerp(startColor, targetColor, t);

        elapsedTime += Time.deltaTime;
        yield return null;
    }

    if (circleEffect != null)
    {

```

```
        circleEffect.transform.localScale = targetScale;
        circleRenderer.color = targetColor;
    }
}

private IEnumerator TemporaryHighlight(GameObject obj)
{
    Renderer renderer = obj.GetComponent<Renderer>();
    if (renderer != null && highlightMaterial != null)
    {
        Material currentMaterial = renderer.material;

        renderer.material = highlightMaterial;

        yield return new WaitForSeconds(highlightDuration);

        if (originalMaterials.ContainsKey(obj))
        {
            renderer.material = originalMaterials[obj];
        }
        else
        {
            renderer.material = currentMaterial;
        }
    }
}

private void UpdateProgress()
{
    if (progressText != null)
    {
        progressText.text = $"Найдено: {foundDifferences} / {totalDifferences}";
    }
}

private IEnumerator CompleteGame()
{
    Log("=== ВСЕ ОТЛИЧИЯ НАЙДЕНЫ! ===", true);

    PlaySound(winSound);

    yield return new WaitForSeconds(0.5f);

    ShowSuccessMessage();
    StartCoroutine(LoadMenuAfterDelay(messageShowTime + 1f));
}

private void PlaySound(AudioClip clip)
{
    if (clip != null)
    {
        Log($"    Воспроизведение звука: {clip.name}", true);

        if (audioSource != null)
        {
            audioSource.PlayOneShot(clip);
        }
        else
        {
            AudioSource.PlayClipAtPoint(clip, mainCamera.transform.position, 1f);
        }
    }
    else
    {
        Debug.LogWarning("    Звуковой клип не назначен!");
    }
}

private void ShowSuccessMessage()
{
    if (successPanel == null) return;
```

```

        successPanel.gameObject.SetActive(true);
        if (messageText != null)
            messageText.text = $"Поздравляем! Вы нашли все {totalDifferences}
отличия!";

        StartCoroutine(FadePanel(0, 1, 0.5f));
        StartCoroutine(FadePanel(1, 0, 0.5f, messageShowTime));
    }

private IEnumerator FadePanel(float from, float to, float duration, float delay =
0f)
{
    if (delay > 0)
        yield return new WaitForSeconds(delay);

    float elapsed = 0f;
    while (elapsed < duration)
    {
        canvasGroup.alpha = Mathf.Lerp(from, to, elapsed / duration);
        elapsed += Time.deltaTime;
        yield return null;
    }

    canvasGroup.alpha = to;

    if (to == 0)
        successPanel.gameObject.SetActive(false);
}

private IEnumerator LoadMenuAfterDelay(float delay)
{
    yield return new WaitForSeconds(delay);
    if (!string.IsNullOrEmpty(menuSceneName))
    {
        Log($"Переход в меню: {menuSceneName}", true);
        SceneManager.LoadScene(menuSceneName);
    }
}

private IEnumerator DebugInfoCoroutine()
{
    yield return new WaitForSeconds(1f);

    Log("=== ИНФОРМАЦИЯ О СИСТЕМЕ ===", true);
    Log($"Камера: {mainCamera.name}", true);
    Log($"Тип камеры: {(mainCamera.orthographic ? "Ортографическая (2D)" :
"Перспективная (3D)})", true);
    Log($"Размер экрана: {Screen.width}x{Screen.height}", true);
    Log($"Объектов-отличий: {differenceObjects.Count}", true);
    Log($"Объектов найдено: {foundDifferences}", true);
    Log($"Mouse.current: {(Mouse.current != null ? "Найдена" : "Не найдена")}",
true);
    Log($"Touchscreen.current: {(Touchscreen.current != null ? "Найден" : "Не
найден")}", true);
    Log($"EventSystem.current: {(EventSystem.current != null ? "Найден" : "Не
найден")}", true);
    Log($"Префаб круга: {(circleEffectPrefab != null ? "Назначен" : "Не
назначен")}", true);
    Log($"Звук победы: {(winSound != null ? "Назначен" : "Не назначен")}", true);
    Log("=== Готово! ===", true);
    Log("Кликайте/касайтесь объектов-отличий", true);
}

private void Log(string message, bool forceLog = false)
{
    if (debugMode || forceLog)
    {
        Debug.Log(message);
    }
}

```

```
void OnDrawGizmos()
{
    if (!debugMode) return;

    Gizmos.color = Color.yellow;
    foreach (GameObject obj in differenceObjects)
    {
        if (obj != null)
        {
            Gizmos.DrawWireSphere(obj.transform.position, 0.2f);
        }
    }
}

private enum InputType
{
    Mouse,
    Touch
}

[ContextMenu("Тест: Воспроизвести звук успеха")]
private void TestSuccessSound()
{
    if (correctSound != null)
    {
        PlaySound(correctSound);
        Log("Тест: воспроизведен звук успеха", true);
    }
    else
    {
        Debug.LogWarning("Звук успеха не назначен!");
    }
}

[ContextMenu("Тест: Воспроизвести звук победы")]
private void TestWinSound()
{
    if (winSound != null)
    {
        PlaySound(winSound);
        Log("Тест: воспроизведен звук победы", true);
    }
    else
    {
        Debug.LogWarning("Звук победы не назначен!");
    }
}

[ContextMenu("Тест: Проверить первый объект")]
private void TestFirstObject()
{
    if (differenceObjects.Count > 0)
    {
        GameObject testObj = differenceObjects[0];
        if (testObj != null)
        {
            Log($"Тест: проверка объекта {testObj.name}", true);
            ProcessClickedObject(testObj);
        }
    }
}

[ContextMenu("Тест: Завершить игру")]
private void TestCompleteGame()
{
    foundDifferences = totalDifferences;
    StartCoroutine(CompleteGame());
}

[ContextMenu("Тест: Создать эффект круга на первом объекте")]
```

```

private void TestCircleEffect()
{
    if (differenceObjects.Count > 0)
    {
        GameObject testObj = differenceObjects[0];
        if (testObj != null)
        {
            CreateCircleEffect(testObj);
        }
    }
}

[ContextMenu("Тест: Удалить все круги")]
private void TestRemoveAllCircles()
{
    foreach (var kvp in objectCircles)
    {
        if (kvp.Value != null)
        {
            DestroyImmediate(kvp.Value);
        }
    }
    objectCircles.Clear();
    Log("Все круги удалены", true);
}

void OnDestroy()
{
    foreach (var kvp in objectCircles)
    {
        if (kvp.Value != null)
        {
            Destroy(kvp.Value);
        }
    }
    objectCircles.Clear();
}
}

```

Скрипт `differenceObjects` необходимо добавить к объектам `Card_1`, `Card_2`, `Card_3`. В каждом объекте следует настроить механизм игры. В список `Difference Objects` следует последовательно добавить игровые объекты — `Card_1`, `Card_2` и `Card_3` что позволит игровой системе идентифицировать их как цели для поиска. Далее требуется настроить визуальную и аудиообратную связь: назначить материал для временной подсветки (`Highlight Material`) и настроить параметры кругового эффекта (`Circle Effect Settings`), включая длительность анимации, начальный и конечный размер, а также цвет кольца; кроме того, важно добавить звуковые клипы для правильного и неправильного выбора, а также для победы. Для отображения прогресса и системных сообщений необходимо связать соответствующие UI-элементы: `Success Panel` для панели успеха, `Message Text` для текста уведомлений и `Progress Text` для счётчика найденных отличий (см. рис. 11).

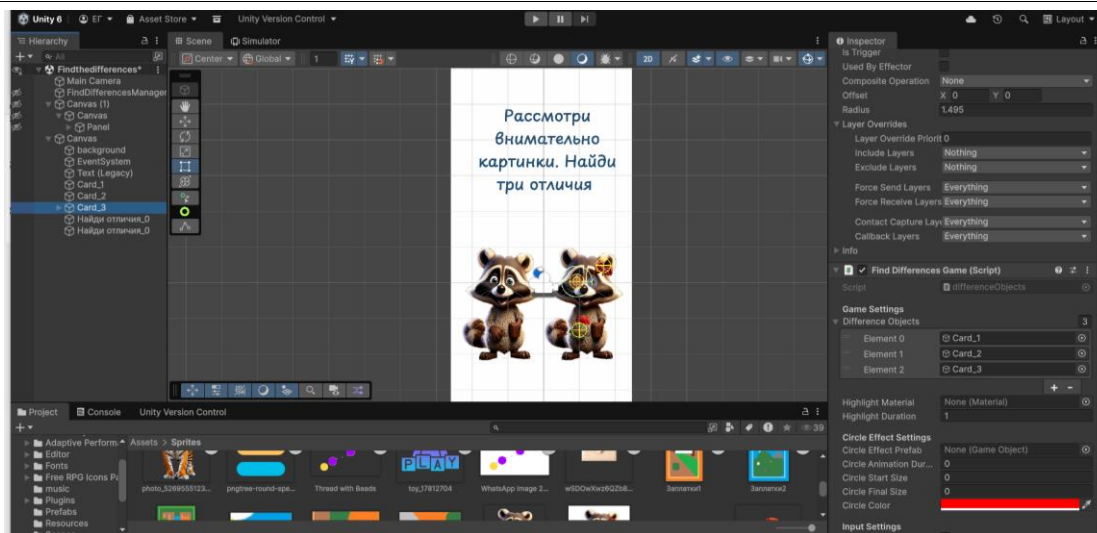


Рисунок 11 – Добавление скрипта к компоненту GameManager и настройка компонента

Итоговый результат создания игры «Четвертый лишний» (см. рис.12).

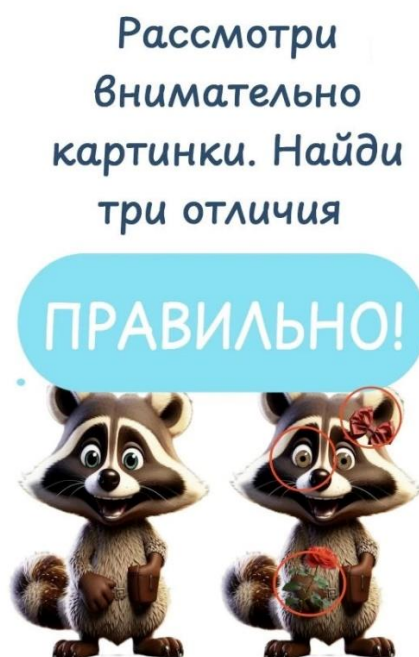


Рисунок 12 – Готовая игра «Четвертый лишний»

4. Выводы

В данной работе был показан процесс создания игры «Найди отличия» с помощью кроссплатформенной среды для создания интерактивных 2D и 3D приложений. Научная статья может служить методическим пособием в учебной деятельности.

Библиографический список

1. Унарова А. Е. Разработка игры на платформе Unity // Технические и естественные науки: вчера, сегодня, завтра: Сборник научных статей. Волгоград: Общество с ограниченной ответственностью "Сфера", 2024. С. 22-25.
2. Фаст Я. А. Разработка 2D игр с элементами головоломок на платформе Unity // V Международная научная конференция по междисциплинарным исследованиям: сборник статей V Международной научной конференции по междисциплинарным исследованиям, Екатеринбург, 15 апреля 2024 года. Екатеринбург: Общество с ограниченной ответственностью "Институт Цифровой Экономики и Права", 2024. С. 266-269.
3. Максимов Н. В. Разработка мобильной 2D игры "Bootur Jump" на Unity // Фундаментальные и прикладные аспекты развития современной науки: Сборник трудов по материалам XV Международного конкурса научно-исследовательских работ, Уфа, 04 декабря 2023 года. Уфа: Общество с ограниченной ответственностью "Научно-издательский центр "Вестник науки", 2023. С. 306-308.
4. Любич А. А. Создание 2D-игры «б-у-у-м» на движке Unity // Фундаментальные и прикладные исследования молодых ученых: Сборник научных трудов XV Всероссийской научно-практической конференции студентов, аспирантов и молодых учёных, Новокузнецк, 07–25 апреля 2025 года. Новокузнецк: Кемеровский государственный университет, 2025. С. 167-169.