

Создание мобильной игры «Найди животных» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Целью статьи является создание мобильной игры «Найди животных» для android. Разработка игры была осуществлена на кроссплатформенной платформе, оснащённой мощным физическим движком и интуитивно понятным визуальным редактором, что делает её оптимальным выбором для создания 2D-игр. В результате работы была создана мобильная игра «Найди животных».

Ключевые слова: Unity, игра, животные, поиск.

Creating a Find the Animals 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 called "Find the Animals" for Android. The game was developed using a cross-platform platform that includes a powerful physics engine and an intuitive visual editor, making it an optimal choice for creating 2D games. As a result, the mobile game "Find the Animals" was created.

Keywords: Unity, game, animals, search.

1 Введение

1.1 Актуальность

Мобильные игры стали неотъемлемой частью современной цифровой культуры, представляя собой одну из самых динамичных и финансово успешных отраслей индустрии развлечений. Особое место в этом сегменте занимают казуальные проекты, которые благодаря интуитивному управлению, лаконичному дизайну и увлекательной механике привлекают широкую аудиторию.

Создание игры «Найди животных», построенной на механике поиска и идентификации, полностью соответствует текущим рыночным трендам. Такой проект служит не только развлечением, но и выполняет просветительскую функцию, способствуя изучению фауны, тренировке внимания, зрительной памяти и концентрации. Это определяет его ценность как в игровой, так и в образовательной сфере, где он может выступать в роли интерактивного пособия для знакомства с миром животных у детей и взрослых.

Реализация проекта в среде Unity является наиболее эффективным и современным решением. Этот движок предоставляет мощный, гибкий инструментарий для кроссплатформенной разработки, позволяя создавать яркую и детализированную графику, анимированных персонажей и интуитивные интерфейсы. Unity позволяет сосредоточиться на качестве контента и удобстве геймплея, а также обеспечивает высокую производительность, простую интеграцию обучающих элементов и лёгкую публикацию в основных магазинах приложений.

Таким образом, разработка мобильной игры «Найди животных» на Unity — это актуальная задача, объединяющая популярный игровой жанр, образовательный потенциал и современные технологии. Проект имеет практическую значимость, соответствует рыночному спросу на познавательный и развивающий контент и открывает широкие возможности для дальнейшего расширения за счёт добавления новых локаций, видов животных и игровых режимов.

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# А.А. Любич [4]. Д.В. Морозова в рамках работы разработала развивающую 2D игра-платформер для детей, написанная на Unity [5].

1.3 Цель исследования

Цель исследования — разработать под операционную систему Android мобильную игру «Найди животных», используя возможности среды разработки Unity для создания интерактивных 2D/3D-приложений.

2 Материалы и методы

Для разработки игры «Найди отличия» был выбран движок Unity. Это решение было обусловлено его ключевыми достоинствами: доступностью, интуитивным интерфейсом и поддержкой языка C#, что оптимально для создания игровой логики. Код проекта разрабатывался в среде Visual Studio Community.

3 Результаты

Первым шагом является создание проекта в Unity Hub: выбирается шаблон «2D (URP)», задаётся имя проекта и подтверждается его создание (см. рис. 1).

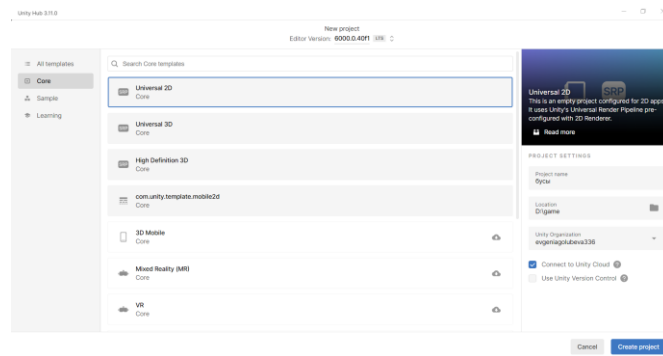


Рисунок 1 – Создание проекта

Следующим шагом после создания проекта является его конфигурация для мобильных платформ. Для этого в окне Build Settings (File → Build Settings) необходимо выбрать Android в качестве целевой платформы (см. рис. 2).

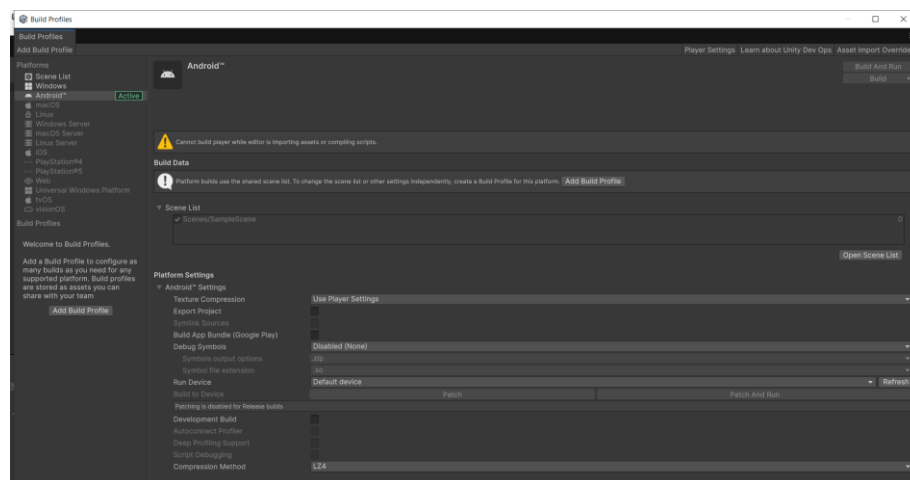


Рисунок 2 – Настройка проекта

После базовой настройки проекта следующий этап — создание основной игровой сцены, первым шагом которого является конфигурация параметров камеры (см. рис. 3).

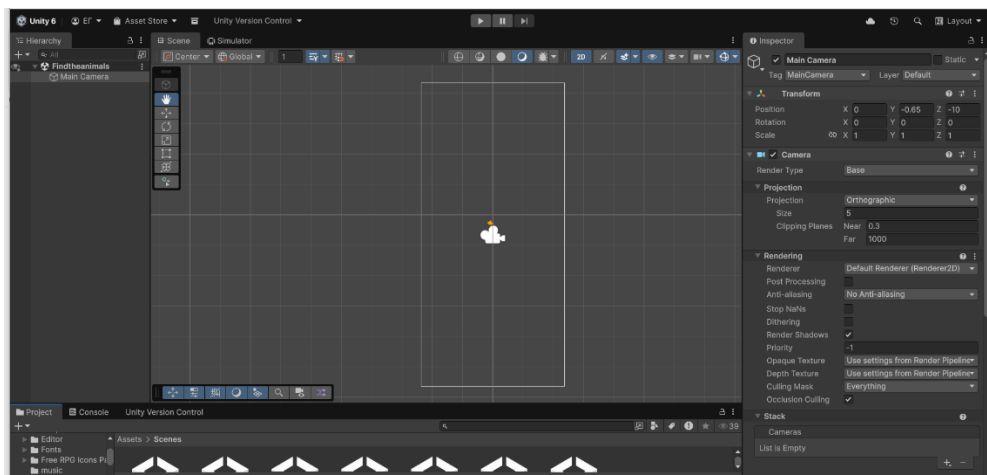


Рисунок 3 – Настройка камеры

Заранее подготовленные графические ресурсы (спрайты) для игры необходимо поместить в папку Sprites (см.рис.4).

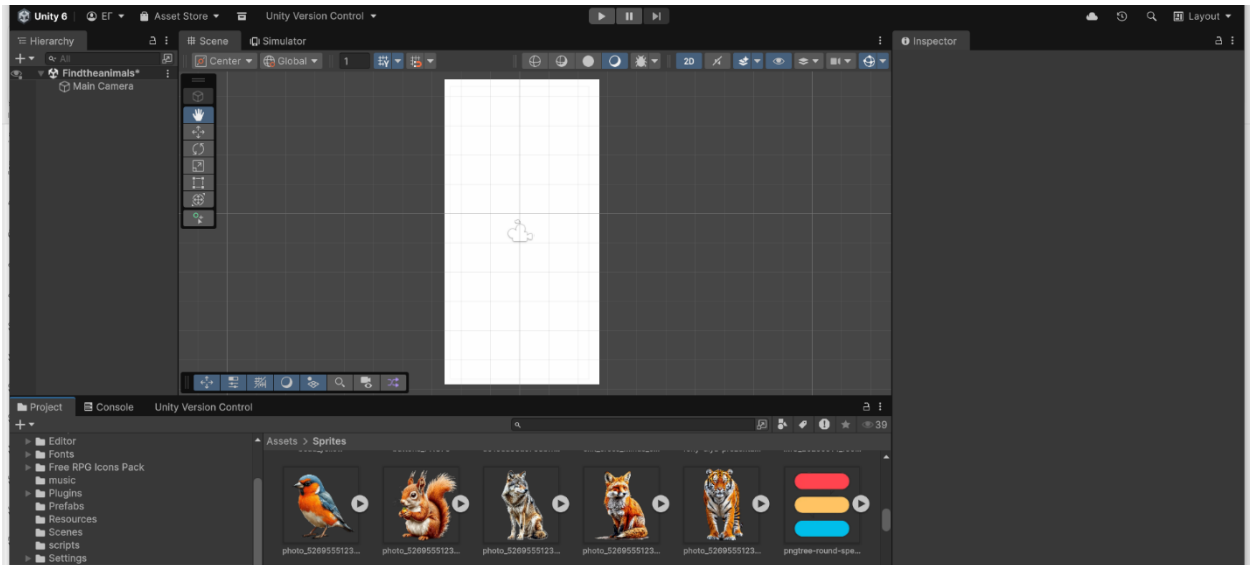


Рисунок 4 – Добавление изображений

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).

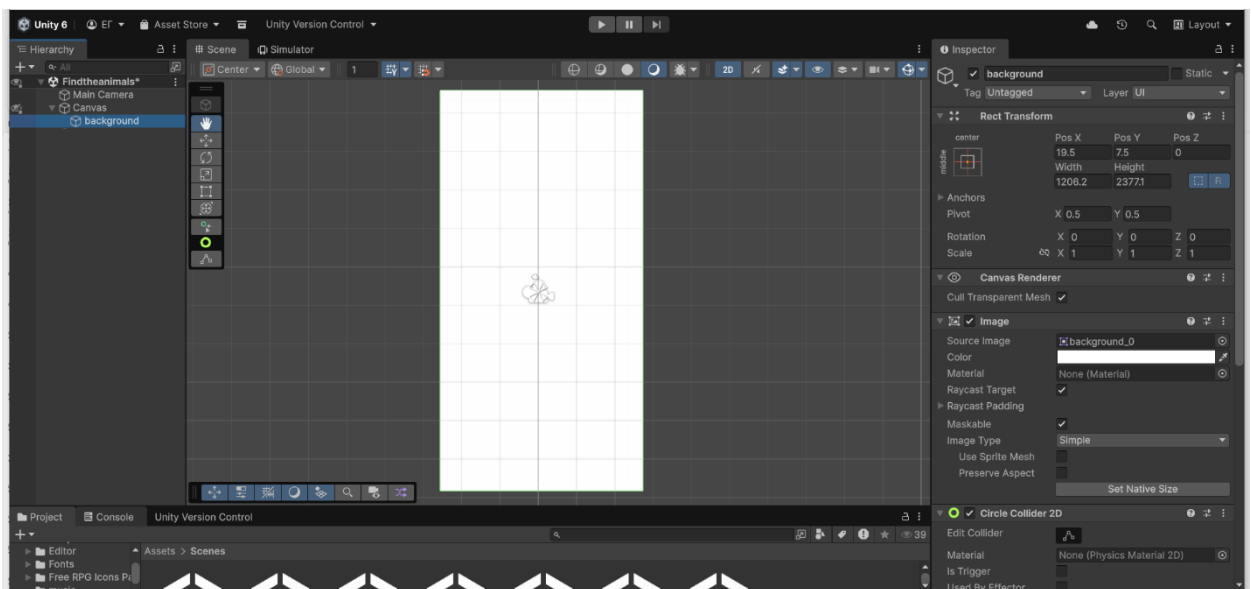


Рисунок 5 – Настройка фона игры

Далее необходимо добавить изображение леса. Для этого перемещаем изображения из папки Sprites на сцену и настраиваем параметры (см.рис.6).

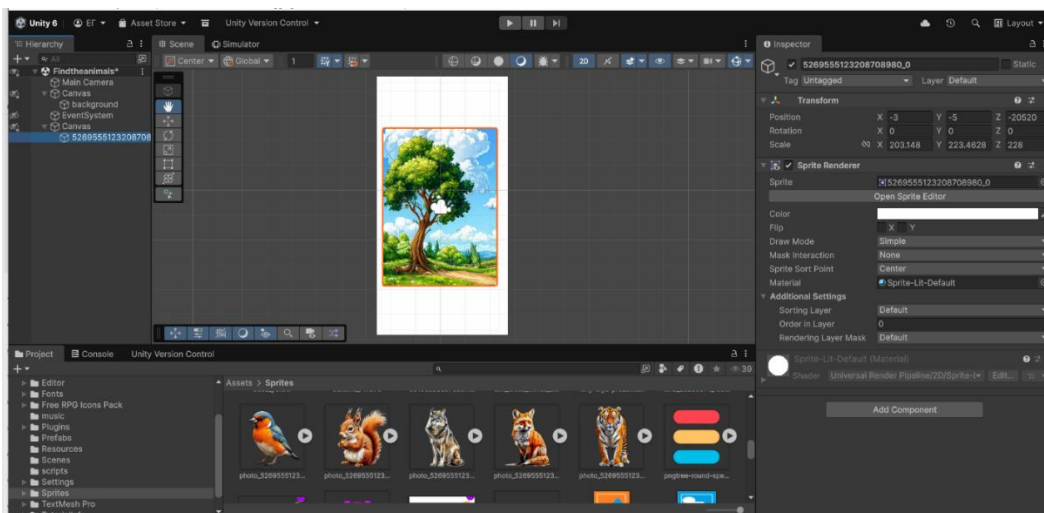


Рисунок 6 – Добавление изображения леса на сцену

Следующим шагом необходимо разместить изображение животных на фон леса, а также и на сцену для того, чтобы игрок понимал каких животных надо найти. Для того, чтобы разместить животных на фон леса, необходимо создать пять пустых объекта, которые будут служить контейнерами для изображений животных. В Hierarchy нажимаем правой кнопкой мыши, и в контекстном меню выбираем Create empty и переименовываем в Card_1, Card_2 и Card_3, Card_4, Card_5. После создание объектов необходимо их настроить (см.рис.7).

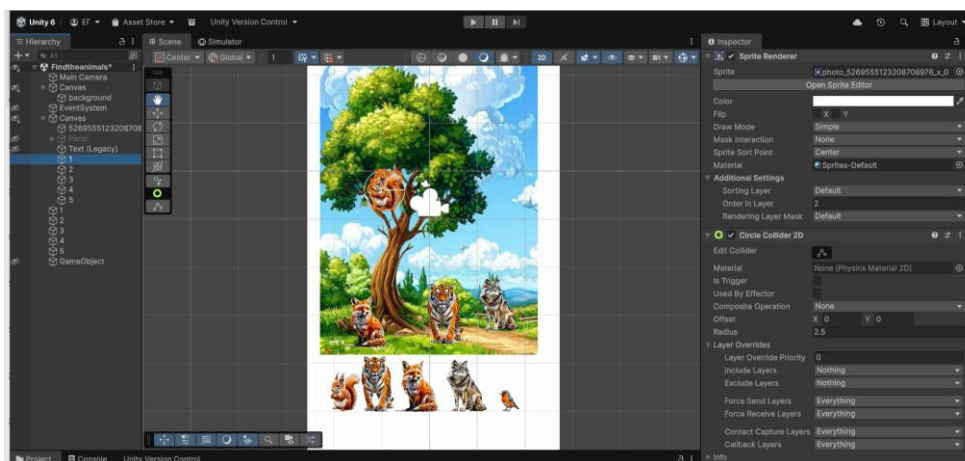


Рисунок 7 – Добавление изображений животных

Для создания интерфейса отображения сообщений в сцене необходимо добавить два основных элемента пользовательского интерфейса: панель (UI -> Panel), которая будет служить фоновой подложкой, и текстовый блок (UI -> Text) для вывода текста. После этого требуется настроить панель в окне Inspector необходимо установить центральные якоря (Anchors) со значениями (0.5, 0.5), а затем задав её конкретный размер, 1115,5 на 294,1 пикселей. Далее, в компоненте Image, нужно выбрать подходящий спрайт, настроить его цвет и уровень прозрачности, а также установить тип отображения (Image Type) в значение Simple. (см.рис.8)



Рисунок 8 – Настройка панели

Для вывода уведомления «Правильно!» выполняется детальная настройка текстового элемента. В компоненте Text (Legacy) выбирается шрифт PlaypenSans-VariableFont_wght с начертанием Bold и устанавливается размер 150 пунктов. В параметрах Rect Transform задаются размеры: ширина 1220.7 и высота 313.4 пикселей, и якоря (Anchors) фиксируется в позиции (0.5, 0.5) (см.рис. 9).

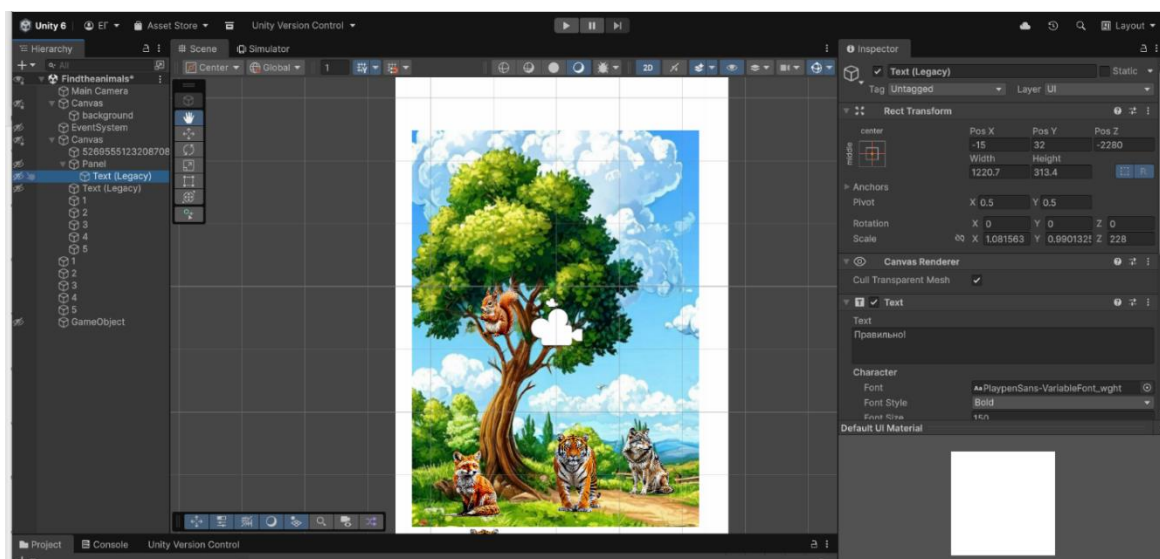


Рисунок 9 – Настройка текстового элемента панели сообщений

Для настройки текстового элемента выбирается шрифт Pangolin-Regular с жирным начертанием Bold и размер шрифта 140 пунктов для максимальной выразительности. Текст настраивается с выравниванием по центру и активируется поддержка Rich Text для возможного форматирования. В компоненте Rect Transform задаются размеры текстового поля: ширина 1050.8 и высота 473.9 пикселей, а также устанавливаются якоря (Anchors) (0.5, 0.5) (см.рис.10).



Рисунок 10 – Настройка инструкции игры

Ядром игры является скрипт `SpotTheDifferenceGame.cs`, управляющий всей логикой. При старте (`Start()`) он инициализирует камеру, звук, проверяет объекты-отличия и настраивает UI. В каждом кадре (`Update()`) обрабатываются клики: координаты касания преобразуются в игровой мир, определяется выбранный объект (с игнорированием UI). При клике на ненайденное отличие скрипт помечает его как найденное, запускает анимацию подтверждения (круг и подсветку), проигрывает звук и обновляет прогресс. Когда все отличия найдены, автоматически запускается финальная последовательность с победным звуком, показом панели успеха и возвратом в меню. Для отладки в скрипт встроены логи и инструменты редактора Unity.

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

public class SpotTheDifferenceGame : 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();
}
}

```

Далее необходимо создать пустой объект для скрипта. Для этого в Hierarchy нажимаем правой кнопкой мыши, и в контекстном меню выбираем Create empty и переименовываем в GameObject. Скрипт SpotTheDifferenceGame.cs добавляем к созданному компоненту, и настраиваем параметры игры. Настройка включает заполнение всех полей,

объявленных в скрипте, чтобы обеспечить корректную работу игровой механики, интерфейса и звукового сопровождения. (см. рис. 11).

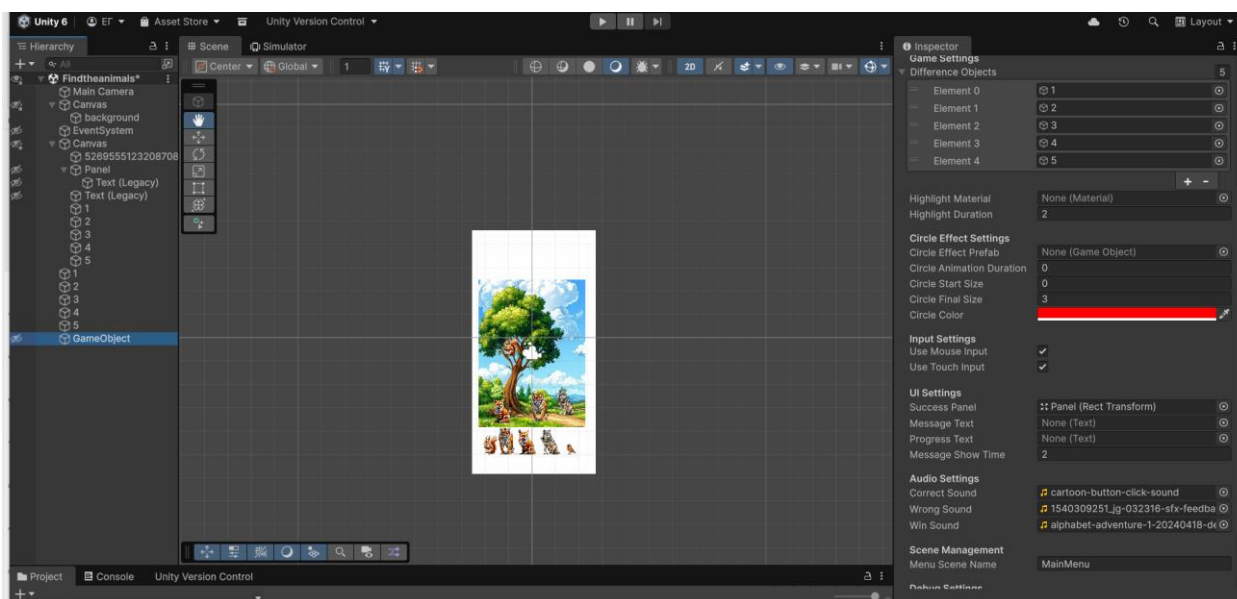


Рисунок 11 – Добавление скрипта к компоненту GameObject и настройка компонента

Итоговый результат создания игры «Найди животных» (см. рис.12).



Рисунок 12 – Готовая игра «Найди животных»

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.