

Разработка мобильной игры «Найди пары» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель данной статьи – создание мобильной игры «Найди пары» для android. Для создания игры была использована кроссплатформенная среда для создания интерактивных 2D- и 3D-приложений Unity, которая позволяет создавать 2D-игры, используя мощный физический движок и визуальные инструменты. В результате работы была создана мобильная игра «найди пары»

Ключевые слова: Unity, игра, соответствие, пары.

Development of the mobile game "Find pairs" on Unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create a mobile game "Find pairs" for Android. To create the game, we used the cross-platform environment for creating interactive 2D and 3D applications Unity, which allows you to create 2D games using a powerful physics engine and visual tools. As a result, we created a mobile game "Find the Pairs"

Keywords: Unity, game, matching, pairs.

1 Введение

1.1 Актуальность

В современном мире мобильные игры прочно вошли в повседневную жизнь миллионов людей, став не только способом развлечения, но и мощной индустрией с многомиллиардными оборотами. Среди всего разнообразия жанров стабильной и массовой популярностью пользуются казуальные головоломки, сочетающие простоту освоения с интуитивно понятным геймплеем. Одной из таких классических и вечных механик является жанр «Найди пары», который остается востребованным у самой широкой аудитории, от детей до взрослых.

Движок Unity, являясь одним из лидеров в среде разработки игр, предоставляет идеальный набор инструментов для быстрой и эффективной реализации подобных проектов. Его мощная и гибкая система управления ассетами (ресурсами), встроенные возможности для работы с 2D-графикой, а

также кроссплатформенность позволяют быстро создавать прототипы и выпускать готовый продукт на все популярные мобильные платформы (iOS и Android), что критически важно для достижения коммерческого успеха.

Таким образом, разработка мобильной игры «Найди пары» на Unity представляет собой актуальную и практически значимую задачу. Проект соответствует рыночному спросу на доступный и интеллектуальный досуг, обладает значительным потенциалом для развития памяти, внимания и концентрации у игроков. Являясь отличным примером применения современных инструментов игровой индустрии, данный проект позволяет не только освоить ключевые аспекты геймдева на Unity, но и создать востребованный продукт, ориентированный на массовую аудиторию.

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

2 Материалы и методы

Для создания игры «найди пары» в качестве игрового движка был выбран Unity. Для редактирования скриптов использовалась интегрированная среда разработки Visual Studio Community.

3 Результаты

Для создания мобильной игры «найди пары» необходимо настроить новый проект в Unity. Для этого в лаунчере Unity Hub необходимо выбрать шаблон для двухмерной разработки Universal 2D, задать название проекту и создать его (см. рис. 1).

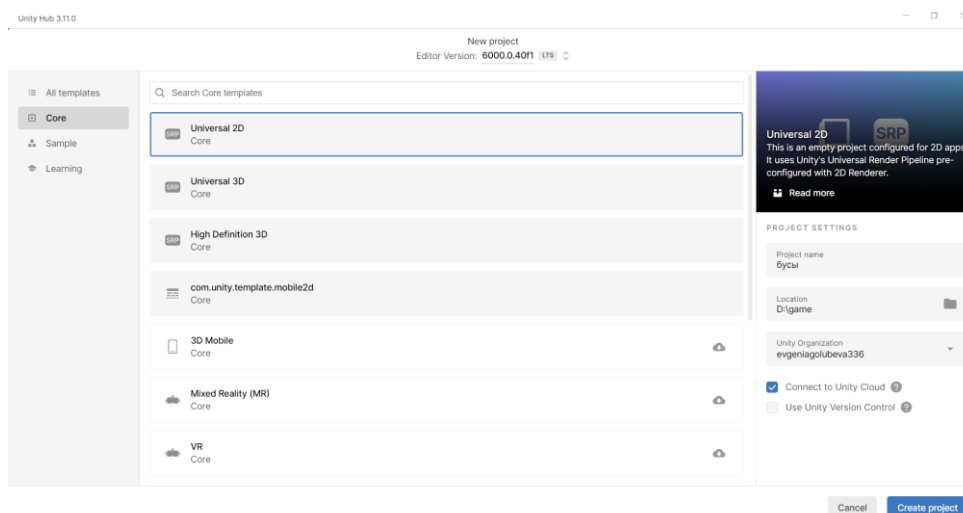


Рисунок-1 Создание проекта

После создания проекта на основе шаблона Universal 2D, необходимо провести базовую настройку под мобильную платформу. В первую очередь, в настройках построения (File -> Build Profiles) следует переключить платформу на Android (см. рис. 2).

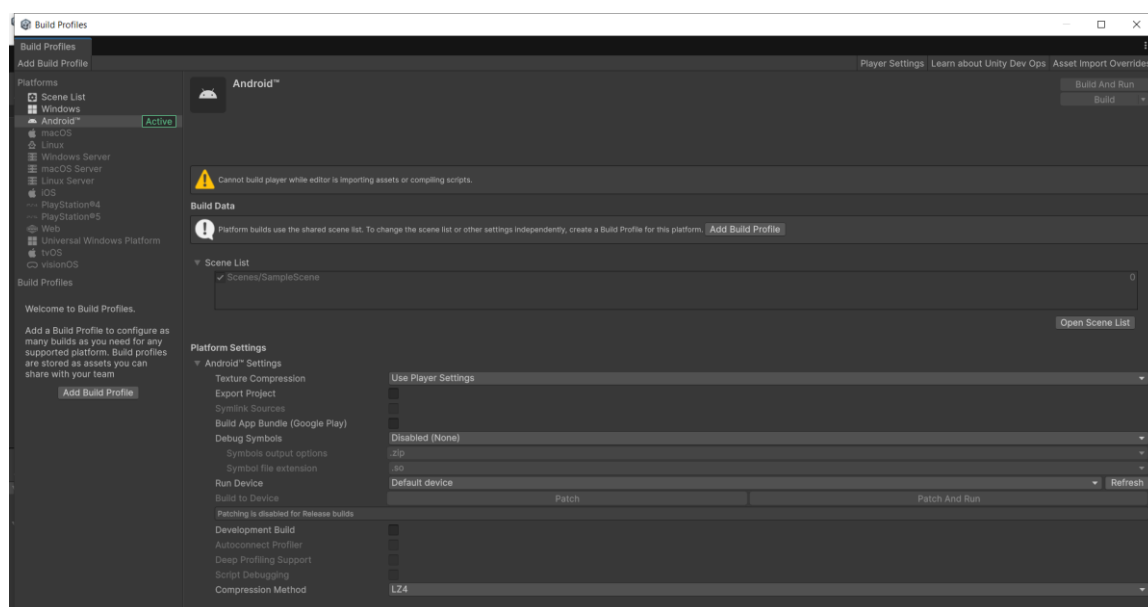


Рисунок 2 - Настройка проекта

Далее приступаем к созданию сцены игры. Первым шагом необходимо настроить камеру (см. рис. 3).

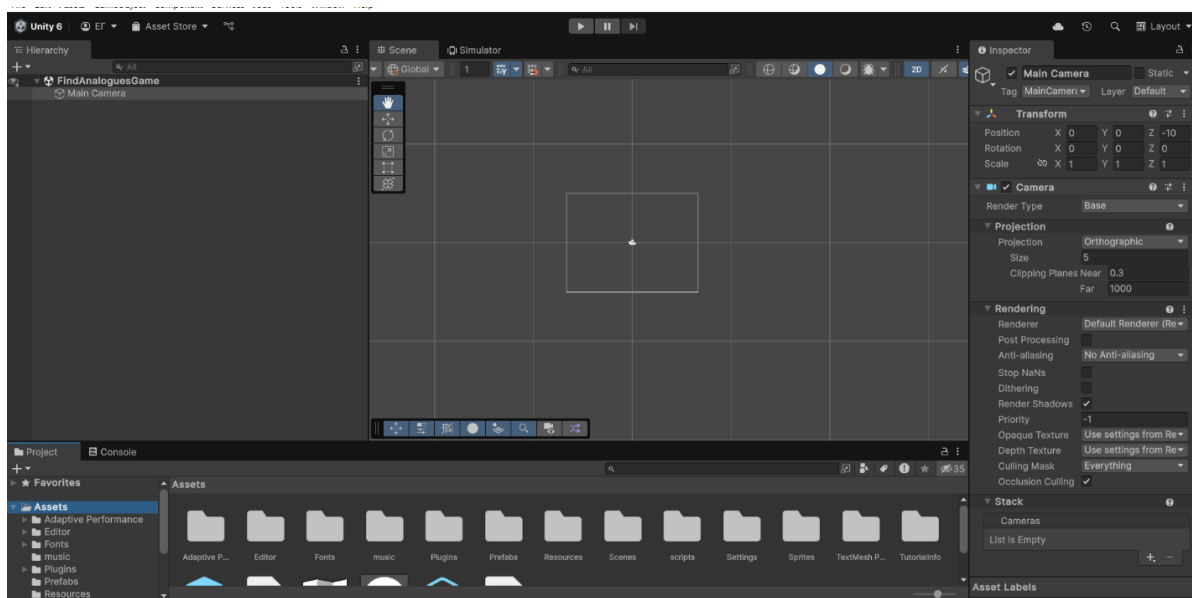


Рисунок 3 - Настройка камеры

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

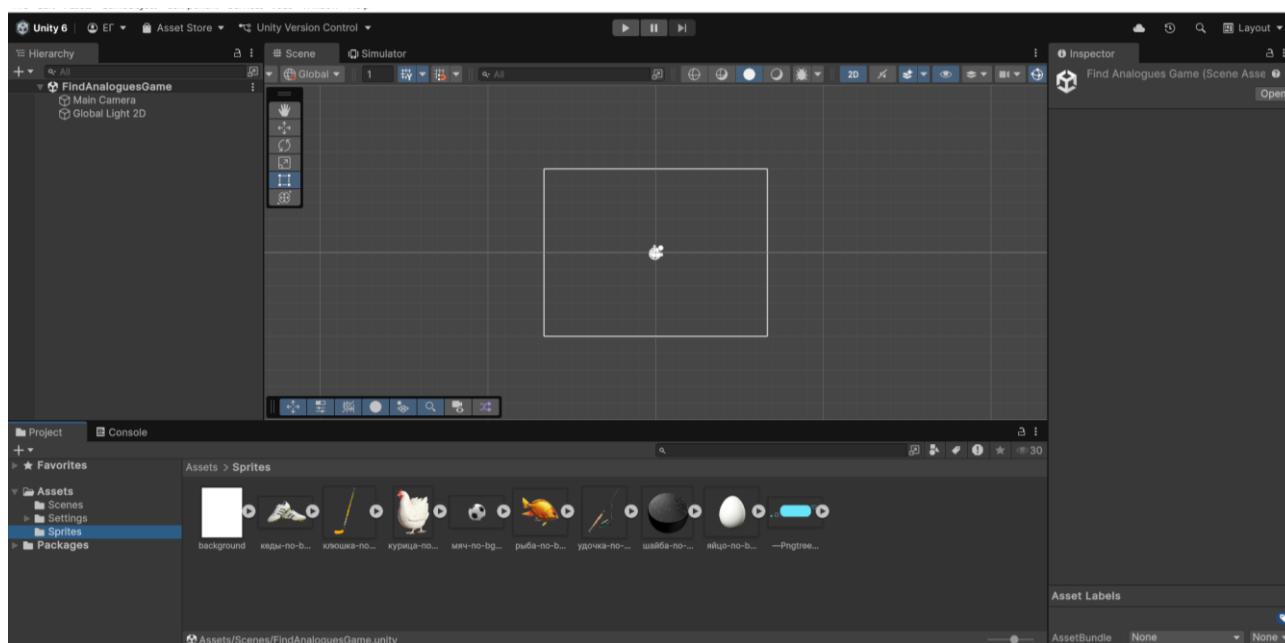


Рисунок 4 - Добавление изображений

Далее необходимо подготовить фон для игровой сцены. Для этого нужно перенести изображение background из папки Sprites на сцену и настроить его параметры так, чтобы оно полностью покрывало игровую область (см.рис.5).

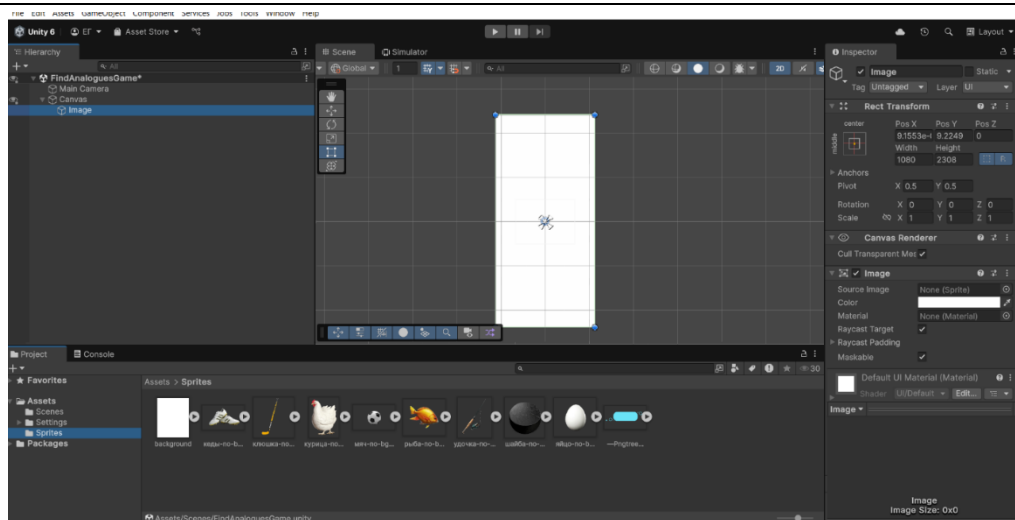


Рисунок 5 – Настройка фона игры

Следующим этапом разработки является добавление изображений предметов на игровую сцену. Необходимо перенести подготовленные изображения из окна Project в область Hierarchy, что создает новые игровые объекты типа Image. Далее в Hierarchy изображения переименовываем в Card_1, Card_2, Card_3, Card_4, Card_5, Card_6, Card_7 и Card_8. Для каждого такого объекта необходимо провести детальную настройку параметров: в компоненте Transform устанавливаются координаты позиционирования Pos X, Pos Y, задаются ширину и высоту изображения, настраиваются якоря Anchors и точка вращения Pivot для обеспечения корректного отображения на различных разрешениях экрана. В компоненте Image указывается исходная текстура в поле Source Image, выбирается тип отображения (Image Type) - "Simple". Для создания интерактивности добавляется компонент Button, где настраиваются цветовые состояния кнопки - нормальное Normal Color, выделенное Highlighted Color и нажатое Pressed Color, а также добавляется компонент Box Collider 2D, размеры которого регулируются в соответствии с видимой областью карточки для точной обработки взаимодействий (см.рис.6).

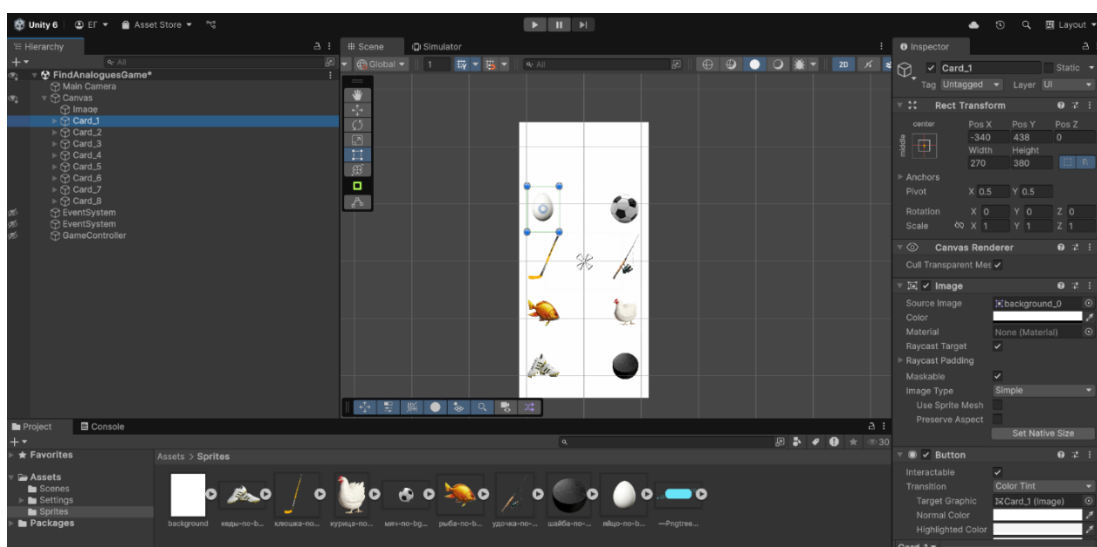


Рисунок 6 – Добавление изображений предметов на сцену

Чтобы реализовать отображение инструкции и уведомления «Правильно!», необходимо добавить на сцену пользовательский интерфейс. Через контекстное меню в Hierarchy создайте панель (UI -> Panel), а затем — текстовый блок (UI -> Text) для вывода текста инструкций.

После добавления панели и текстового элемента в интерфейс игры необходимо провести их настройку через компонент Inspector. В первую очередь следует установить якоря Anchors в центральное положение со значениями (0.5, 0.5), что обеспечивает правильное позиционирование и масштабирование элемента относительно различных разрешений экрана. Затем задаются конкретные размеры области Width 1049 и Height 481 пикселей, формирующие оптимальную рабочую зону. Для визуального оформления в компоненте Image необходимо выбрать подходящий спрайт фонового изображения в поле Source Image, настраивается цвет и прозрачность через параметр Color в соответствии с общим визуальным стилем игры, а тип отображения устанавливается в значение Simple (см.рис.7).

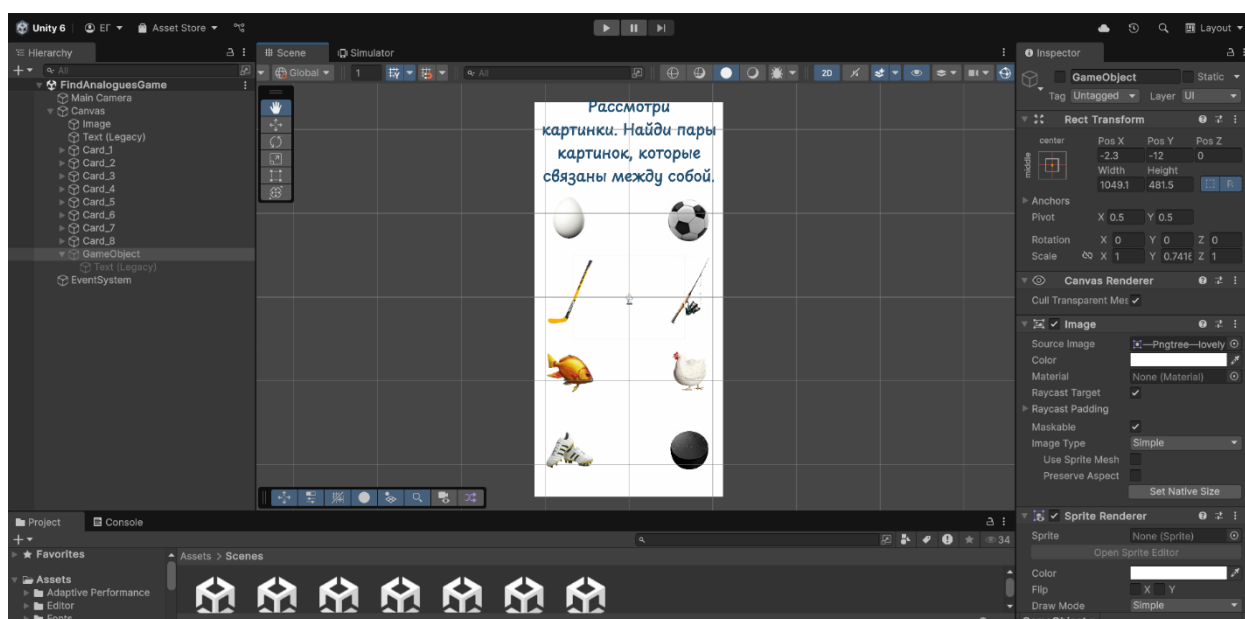


Рисунок 7 – Настройка панели

После настройки панели необходимо отредактировать текстовый блок с уведомлением "Правильно!". В компоненте Text (Legacy) необходимо выбрать шрифт PlaypenSans-VariableFont_wght с жирным начертанием Bold и установить размер 150 пунктов для обеспечения оптимальной читаемости. В разделе Rect Transform установить размеры текстового поля: Width 1041 и Height 347. Также необходимо настроить якоря Anchors с центральной точкой (0.5, 0.5), что обеспечит корректное отображение текстового элемента на экранах с различным разрешением. Далее необходимо настроить текст, для этого необходимо выбрать выравнивание текста по центру, цветовое оформление и параметры переноса текста для достижения гармоничного визуального представления в интерфейсе игры. (см.рис. 8).

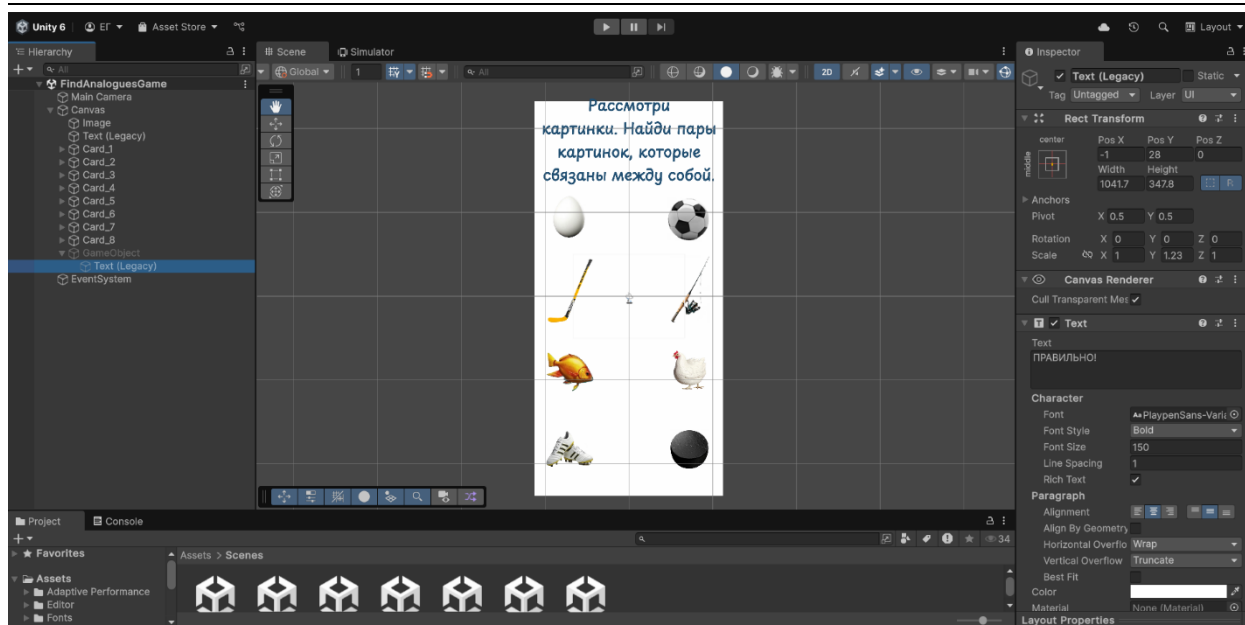


Рисунок 8 – Настройка текстового элемента панели сообщений

Для текстового элемента настраиваем шрифт Pangolin-Regular с жирным начертанием и размером 88 пунктов с жирным начертанием Bold, выравнивание по центру, включаем поддержку Rich Text для форматирования, а в Rect Transform задаем точные размеры текстового поля: Width 1080, Height 577,9, и устанавливаем якоря Anchors с центром в точке (0.5, 0.5) (см.рис.9).

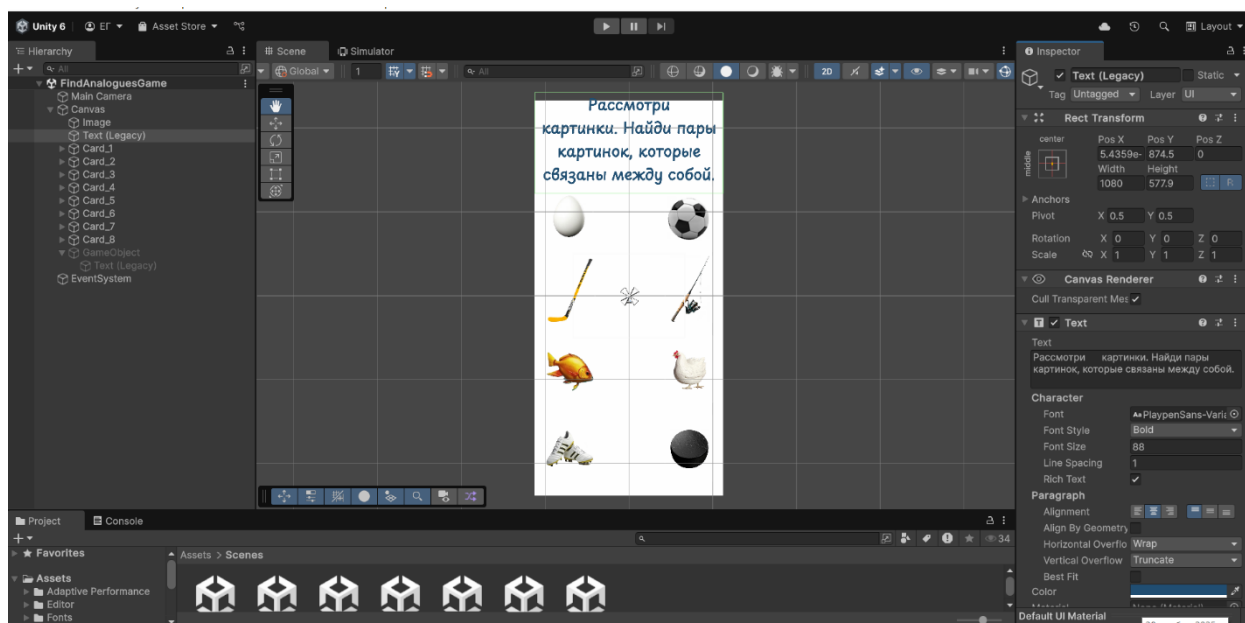


Рисунок 9 – Настройка инструкции игры

Также необходимо создать компонент GameController, который будет отвечать за механизм игры. Для этого в Hierarchy создаем пустой игровой объект через правый клик и выбираем Create Empty. Этот объект необходимо переименовать в GameController. Его основное предназначение — централизованное управление игровой логикой, включая обработку выбора карточек, проверку совпадений, визуализацию соединений между карточками,

управление звуковыми эффектами и отслеживание состояния игры (см.рис.10).

Для компонента GameController необходимо написать скрипт под названием GameController, который реализует основную игровую механику перетаскивания предметов. Данный скрипт содержит несколько ключевых функциональных блоков:

Централизованное управление игровым процессом через паттерн Singleton, обеспечивающий единственный экземпляр контроллера на сцену. В методе Awake() происходит инициализация этого экземпляра, настройка AudioSource для звуковых эффектов и скрытие панели победы.

Система обработки выбора карточек через метод SelectCard(), который отслеживает выбранные игроком карточки, проверяет возможность выбора.

Механизм проверки совпадений в корутине CheckMatch(), которая сравнивает pairID выбранных карточек, определяет правильность комбинации и визуально отображает результат через цвет линии (зеленый для правильных, красный для неправильных совпадений). При успешном совпадении карточки помечаются как сопоставленные, при ошибке - возвращаются в исходное состояние.

Управление звуковыми эффектами через метод PlaySound(), который воспроизводит соответствующие аудиоклипы для различных игровых событий: выбор карточки, правильное и неправильное совпадение, победа в игре.

Система отслеживания завершения игры через метод CheckGameCompletion(), который проверяет состояние всех зарегистрированных карточек и при полном совпадении всех пар запускает процедуру завершения игры с отображением победной панели и автоматическим переходом в меню через заданное время.

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

public class GameController : MonoBehaviour
{
    public static GameController Instance;

    [Header("Line Settings")]
    [SerializeField] private ConnectionLine lineRenderer;
    [SerializeField] private Color correctLineColor = Color.green;
    [SerializeField] private Color wrongLineColor = Color.red;
    [SerializeField] private float lineDisplayTime = 1f;

    [Header("Sound Settings")]
    [SerializeField] private AudioClip cardSelectSound;
    [SerializeField] private AudioClip correctMatchSound;
    [SerializeField] private AudioClip wrongMatchSound;
    [SerializeField] private AudioClip victorySound;

    [Header("Victory Panel")]
    [SerializeField] private GameObject victoryPanel;
```

```
[SerializeField] private Text victoryText;

[Header("Scene Settings")]
[SerializeField] private string menuSceneName = "Menu";
[SerializeField] private float delayBeforeMenuLoad = 3f;

private List<Card> selectedCards = new List<Card>();
private List<Card> allCards = new List<Card>();
private AudioSource audioSource;
public bool CanSelect { get; private set; } = true;

void Awake()
{
    if (Instance == null)
    {
        Instance = this;
    }
    else
    {
        Destroy(gameObject);
    }

    audioSource = GetComponent<AudioSource>();
    if (audioSource == null)
        audioSource = gameObject.AddComponent<AudioSource>();

    if (victoryPanel != null)
        victoryPanel.SetActive(false);
}

void Start()
{
    // Дополнительная проверка компонентов
    if (lineRenderer == null)
    {
        lineRenderer = FindObjectOfType<ConnectionLine>();
        if (lineRenderer == null)
            Debug.LogWarning("LineRenderer not found in scene!");
        else
            Debug.Log("LineRenderer found automatically");
    }

    if (audioSource == null)
    {
        Debug.LogError("AudioSource is missing!");
    }
}

void Update()
{
    // Для тестирования линии - нажмите Т для создания тестовой линии
    if (Input.GetKeyDown(KeyCode.T))
    {
        TestLine();
    }

    // Для тестирования скрытия линии - нажмите Y
    if (Input.GetKeyDown(KeyCode.Y))
    {
        if (lineRenderer != null)
        {
            lineRenderer.HideLine();
            Debug.Log("Line hidden");
        }
    }
}
```

```

    }

    // Для отладки состояния линии - нажмите U
    if (Input.GetKeyDown(KeyCode.U) && lineRenderer != null)
    {
        lineRenderer.DebugLineState();
    }
}

public void SelectCard(Card card)
{
    if (selectedCards.Contains(card) || !CanSelect || card.IsMatched)
return;

    PlaySound(cardSelectSound);
    selectedCards.Add(card);

    if (selectedCards.Count == 1)
    {
        if (lineRenderer != null)
        {
            // Получаем RectTransform карточки для правильных координат
            RectTransform cardRect = card.GetComponent<RectTransform>();
            if (cardRect != null)
            {
                // Получаем мировые координаты центра карточки
                Vector3[] corners = new Vector3[4];
                cardRect.GetWorldCorners(corners);
                Vector3 center = (corners[0] + corners[2]) / 2f;

                lineRenderer.StartLine(center);
                Debug.Log($"First card selected at: {center}");
            }
            else
            {
                lineRenderer.StartLine(card.transform.position);
            }
        }
    }
    else if (selectedCards.Count == 2)
    {
        if (lineRenderer != null)
        {
            // Получаем RectTransform обеих карточек
            RectTransform card1Rect =
selectedCards[0].GetComponent<RectTransform>();
            RectTransform card2Rect =
selectedCards[1].GetComponent<RectTransform>();

            Vector3 pos1, pos2;

            if (card1Rect != null && card2Rect != null)
            {
                Vector3[] corners1 = new Vector3[4];
                Vector3[] corners2 = new Vector3[4];
                card1Rect.GetWorldCorners(corners1);
                card2Rect.GetWorldCorners(corners2);

                pos1 = (corners1[0] + corners1[2]) / 2f;
                pos2 = (corners2[0] + corners2[2]) / 2f;

                Debug.Log($"Card centers: {pos1} -> {pos2}");
            }
            else

```

```
        {
            pos1 = selectedCards[0].transform.position;
            pos2 = selectedCards[1].transform.position;
        }

        lineRenderer.SetLinePoints(pos1, pos2);
    }
    StartCoroutine(CheckMatch());
}
private IEnumerator CheckMatch()
{
    CanSelect = false;

    Card firstCard = selectedCards[0];
    Card secondCard = selectedCards[1];

    bool isMatch = firstCard.pairID == secondCard.pairID;
    Debug.Log($"Match check: {firstCard.pairID} vs {secondCard.pairID} = {isMatch}");

    if (lineRenderer != null)
    {
        Color lineColor = isMatch ? correctLineColor : wrongLineColor;
        lineRenderer.SetLineColor(lineColor);
    }
    if (isMatch)
    {
        PlaySound(correctMatchSound);
        yield return new WaitForSeconds(lineDisplayTime);

        firstCard.MarkAsMatched();
        secondCard.MarkAsMatched();

        CheckGameCompletion();
    }
    else
    {
        PlaySound(wrongMatchSound);
        yield return new WaitForSeconds(lineDisplayTime);

        firstCard.ResetVisual();
        secondCard.ResetVisual();

        if (lineRenderer != null)
            lineRenderer.HideLine();
    }
    selectedCards.Clear();
    CanSelect = true;
}

private void PlaySound(AudioClip clip)
{
    if (audioSource != null && clip != null)
    {
        audioSource.PlayOneShot(clip);
    }
    else
    {
        Debug.LogWarning("AudioSource or AudioClip is missing!");
    }
}

private void CheckGameCompletion()
{

```

```
// Проверяем, все ли карточки совпали
bool allMatched = true;
foreach (Card card in allCards)
{
    if (!card.IsMatched)
    {
        allMatched = false;
        break;
    }
}
if (allMatched)
{
    Debug.Log("All cards matched! Game complete.");
    StartCoroutine(GameComplete());
}
}

private IEnumerator GameComplete()
{
    yield return new WaitForSeconds(0.5f);

    PlaySound(victorySound);

    if (victoryPanel != null)
    {
        victoryPanel.SetActive(true);
        if (victoryText != null)
            victoryText.text = "Поздравляю!";
    }
    else
    {
        Debug.LogWarning("Victory panel is not assigned!");
    }

    // Ждем указанное время и загружаем сцену меню
    yield return new WaitForSeconds(delayBeforeMenuLoad);
    LoadMenuScene();
}

private void LoadMenuScene()
{
    if (!string.IsNullOrEmpty(menuSceneName))
    {
        Debug.Log($"Loading menu scene: {menuSceneName}");
        SceneManager.LoadScene(menuSceneName);
    }
    else
    {
        Debug.LogWarning("Menu scene name is not set!");
    }
}

public void RegisterCard(Card card)
{
    if (!allCards.Contains(card))
    {
        allCards.Add(card);
        Debug.Log($"Card registered: {card.name}, PairID: {card.pairID},
Total cards: {allCards.Count}");
    }
}

public void ResetGame()
{
    selectedCards.Clear();
    CanSelect = true;
}
```

```

        foreach (Card card in allCards)
        {
            card.ResetCard();
        }
        if (lineRenderer != null)
            lineRenderer.HideLine();
        if (victoryPanel != null)
            victoryPanel.SetActive(false);
        Debug.Log("Game reset");
    }
    // Добавляем метод для ручного перехода в меню (например, по кнопке)
    public void GoToMenu()
    {
        LoadMenuScene();
    }
    // Метод для отладки - проверка состояния игры
    public void DebugGameState()
    {
        Debug.Log($"Total cards: {allCards.Count}");
        Debug.Log($"Selected cards: {selectedCards.Count}");
        Debug.Log($"CanSelect: {CanSelect}");

        int matchedCount = 0;
        foreach (Card card in allCards)
        {
            if (card.IsMatched) matchedCount++;
        }
        Debug.Log($"Matched cards: {matchedCount}/{allCards.Count}");
    }
    // ВРЕМЕННЫЙ МЕТОД ДЛЯ ТЕСТИРОВАНИЯ ЛИНИИ
    private void TestLine()
    {
        if (lineRenderer != null)
        {
            // Тестируем с координатами в центре экрана
            Vector2 startPos = new Vector2(Screen.width * 0.3f, Screen.height
* 0.5f);
            Vector2 endPos = new Vector2(Screen.width * 0.7f, Screen.height *
0.5f);

            lineRenderer.SetLinePoints(startPos, endPos);
            lineRenderer.SetLineColor(Color.blue);

            Debug.Log($"Test line created from {startPos} to {endPos}");
            Debug.Log($"Screen dimensions: {Screen.width}x{Screen.height}");
        }
        else
        {
            Debug.LogError("LineRenderer is not assigned in GameController!");
        }
    }
}

```

После создания скрипта GameController необходимо добавить его к компоненту GameController. Для этого необходимо перетащить скрипт к компоненту в Hierarchy и его настроить его параметры через инспектор: в разделе Line Settings требуется назначить объект с компонентом ConnectionLine для визуализации связей между карточками, задать цветовую схему линий (Correct Line Color для верных соединений и Wrong Line Color для ошибочных), а также установить продолжительность отображения линии перед проверкой совпадения; в секции Sound Settings необходимо присвоить


```
public class Card : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
    public int pairID;
    public bool IsMatched { get; private set; }

    [Header("Visuals")]
    [SerializeField] private Image borderImage;
    [SerializeField] private Image contentImage;
    [SerializeField] private Color selectedColor = new Color(0.8f, 0.8f, 0.1f,
1f);

    private GameController gameController;
    private Color originalColor;
    private Button button;

    void Start()
    {
        gameController = GameController.Instance;
        originalColor = borderImage.color;
        button = GetComponent<Button>();

        if (!borderImage) borderImage = GetComponent<Image>();
        if (!contentImage && transform.childCount > 0)
            contentImage = transform.GetChild(0).GetComponent<Image>();

        gameController?.RegisterCard(this);
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        if (!IsMatched && gameController != null && gameController.CanSelect)
        {
            borderImage.DOColor(selectedColor, 0.2f);
            gameController.SelectCard(this);
        }
    }

    public void OnPointerUp(PointerEventData eventData) { }

    public void MarkAsMatched()
    {
        IsMatched = true;
        if (button != null) button.interactable = false;
        borderImage.DOColor(new Color(0.5f, 0.5f, 0.5f, 0.5f), 0.5f);
        if (contentImage != null)
            contentImage.DOFade(0.5f, 0.5f);
    }

    public void ResetVisual()
    {
        borderImage.DOColor(originalColor, 0.2f);
    }

    public void ResetCard()
    {
        IsMatched = false;
        ResetVisual();
        if (button != null)
        {
            button.interactable = true;
        }
        if (contentImage != null)
        {
            contentImage.DOFade(1f, 0.1f);
        }
    }
}
```

```
}  
}
```

Далее необходимо перетащить скрипт к каждому компоненту карточек в Hierarchy и его настроить его параметры через инспектор. Настройка скрипта Card в инспекторе включает несколько ключевых параметров, обеспечивающих корректную работу карточки в игровой механике. В поле Pair ID указывается уникальный числовой идентификатор, который связывает карточки в логические пары - одинаковые значения Pair ID обозначают соответствующие друг другу карточки. В разделе Visuals необходимо настроить визуальные компоненты: в поле Border Image перетаскивается объект с компонентом Image, отвечающий за отображение рамки карточки, а в Content Image назначается изображение с основным контентом (иллюстрацией, символом или текстом). Параметр Selected Color определяет цвет подсветки рамки при выборе карточки, визуальнo информируя игрока о текущем выборе. Для корректной работы необходимо убедиться, что все визуальные компоненты правильно связаны, а значение Pair ID согласовано с парной карточкой, что обеспечивает точное распознавание совпадений в игровой логике (см. рис. 11).

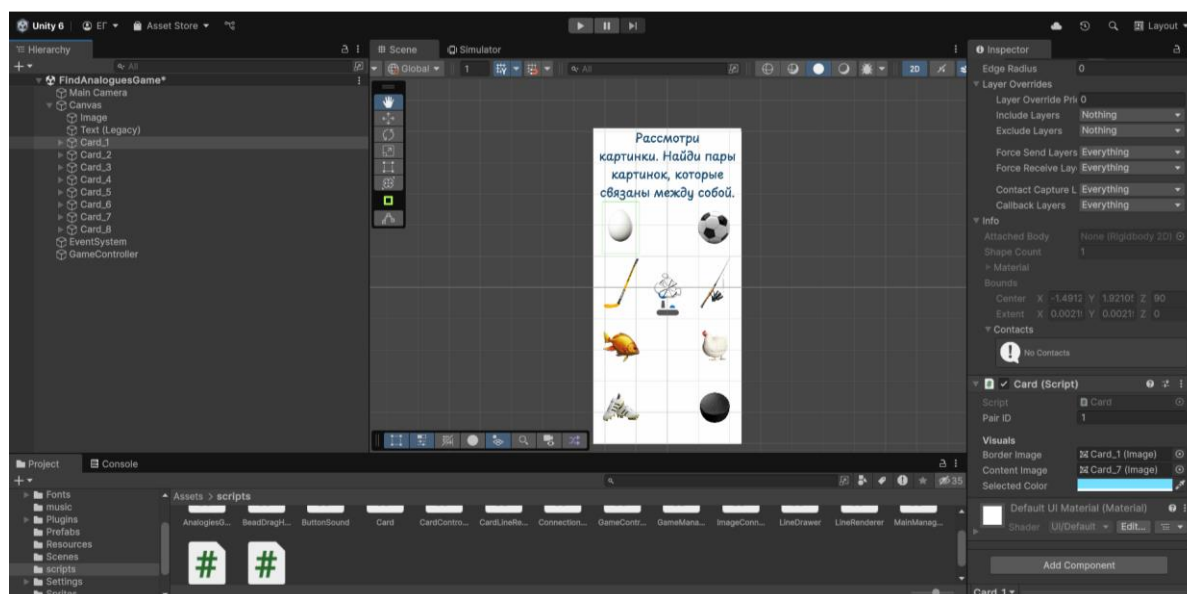


Рисунок 11 – Добавление скрипта ко всем компонентам карточек

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

Рассмотри
картинки. Найди пары
картинок, которые
связаны между собой.



Рисунок 12 – Готовая игра «Найди пары»

4. Выводы

В данной работе был показан процесс создания игры «Найди пары» 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.