

Создание мобильной игры «Найди одинаковые картинки» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель данной статьи – создание мобильной игры «Найди одинаковые картинки» 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 одинаковые картинки»

Ключевые слова: Unity, игра, соответствие, пары, изображения.

Creating a mobile game "Find the same pictures" using Unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create a mobile game "Find the Same Pictures" 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 Same Pictures"

Keywords: Unity, game, matching, pairs, images.

1 Введение

1.1 Актуальность

Мобильные игры превратились из простого развлечения в глобальную культурную и экономическую силу, формируя многомиллиардную индустрию и новый цифровой досуг. В этом разнообразии жанров особой и неизменной популярностью пользуются казуальные головоломки, предлагающие идеальный баланс простоты и интеллектуального вызова. Одной из таких классических и вечных механик является игра «Найди одинаковые картинки», которая остается востребованной 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 одинаковые картинки» 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 одинаковые картинки» для ОС Android при помощи кроссплатформенной среды для создания интерактивных 2D и 3D приложений Unity.

2 Материалы и методы

Для создания игры «Найди одинаковые картинки» в качестве игрового движка был выбран Unity. Для редактирования скриптов использовалась интегрированная среда разработки visual studio community.

3 Результаты

Для создания мобильной игры «Найди одинаковые картинки» 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).

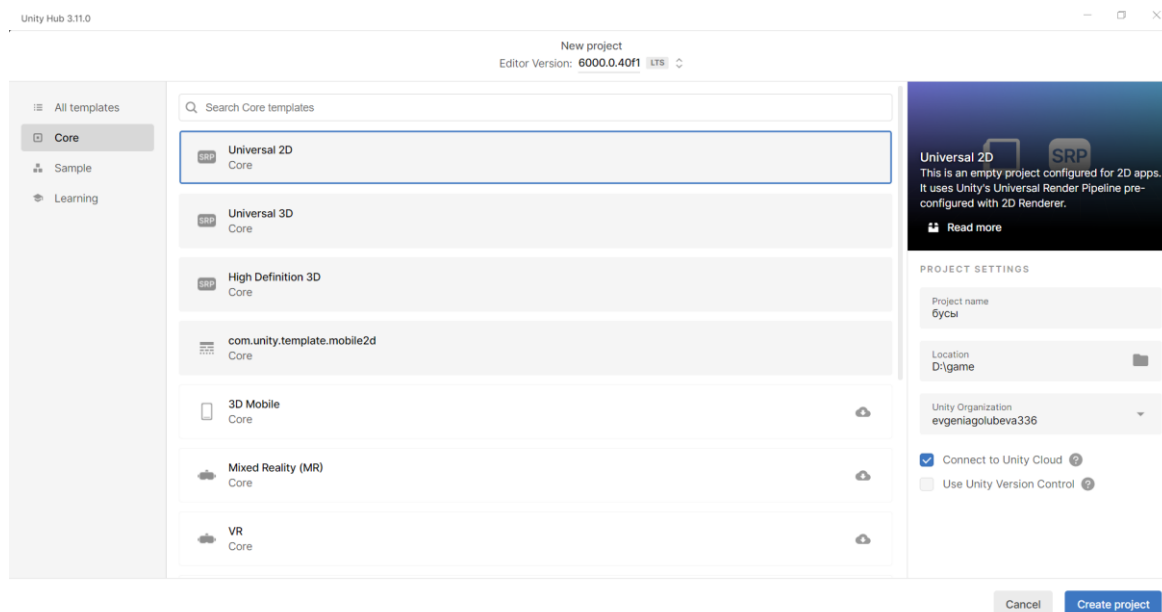


Рисунок 1 – Создание проекта

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).

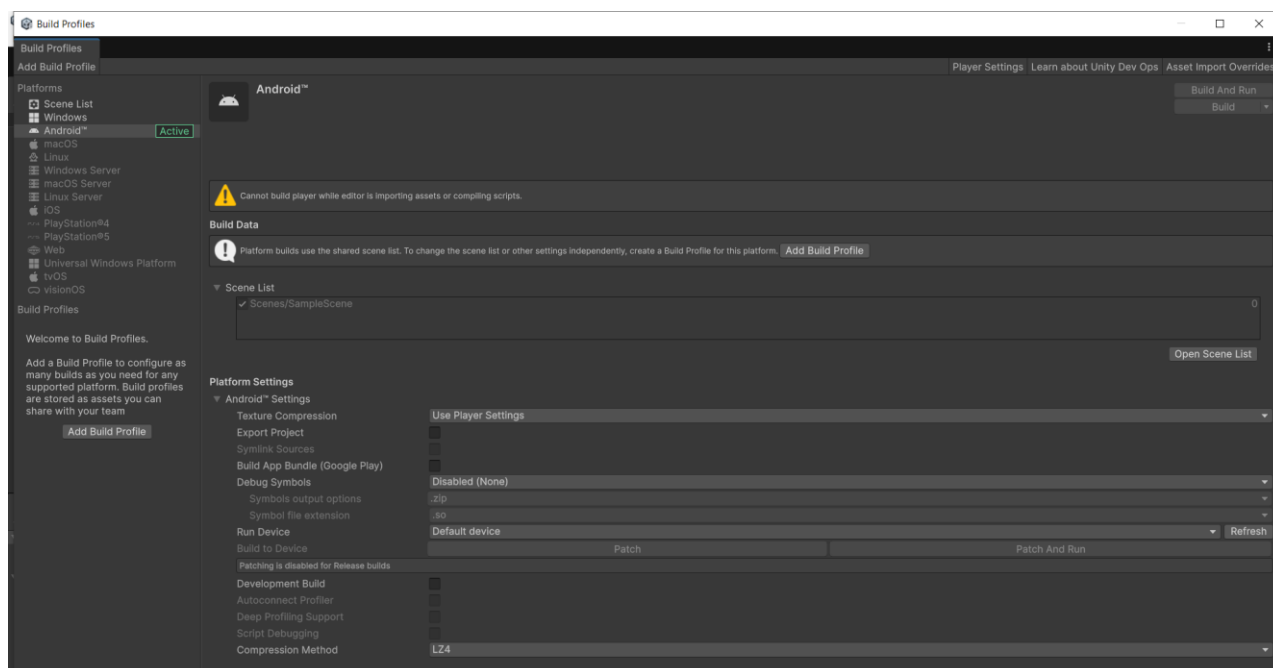


Рисунок 2 – Настройка проекта

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

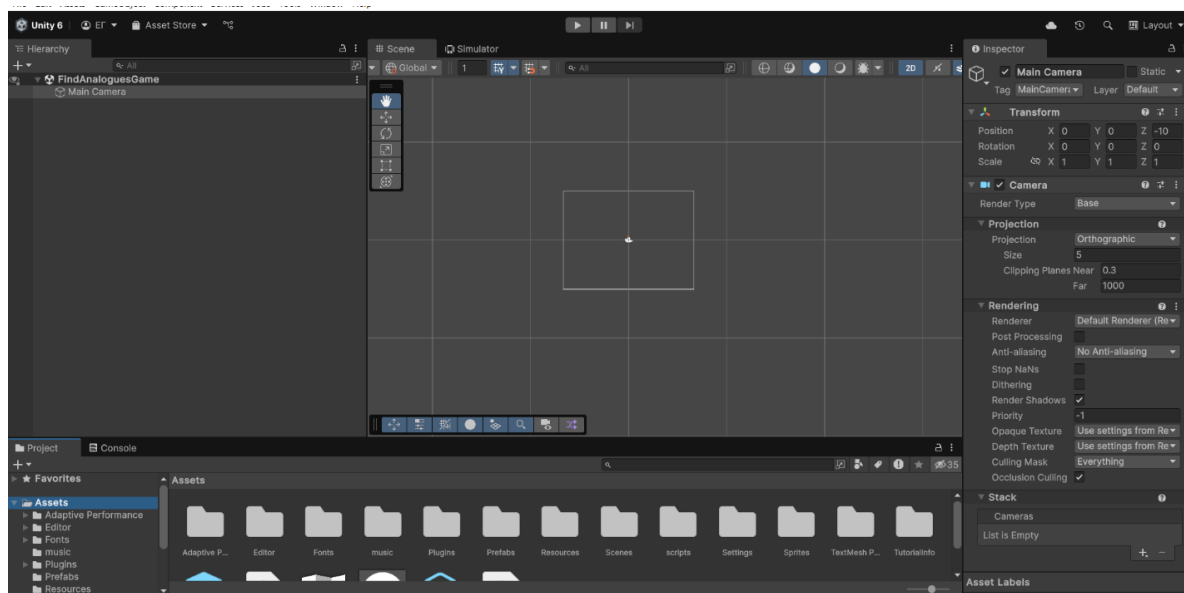


Рисунок 3 – Настройка камеры

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

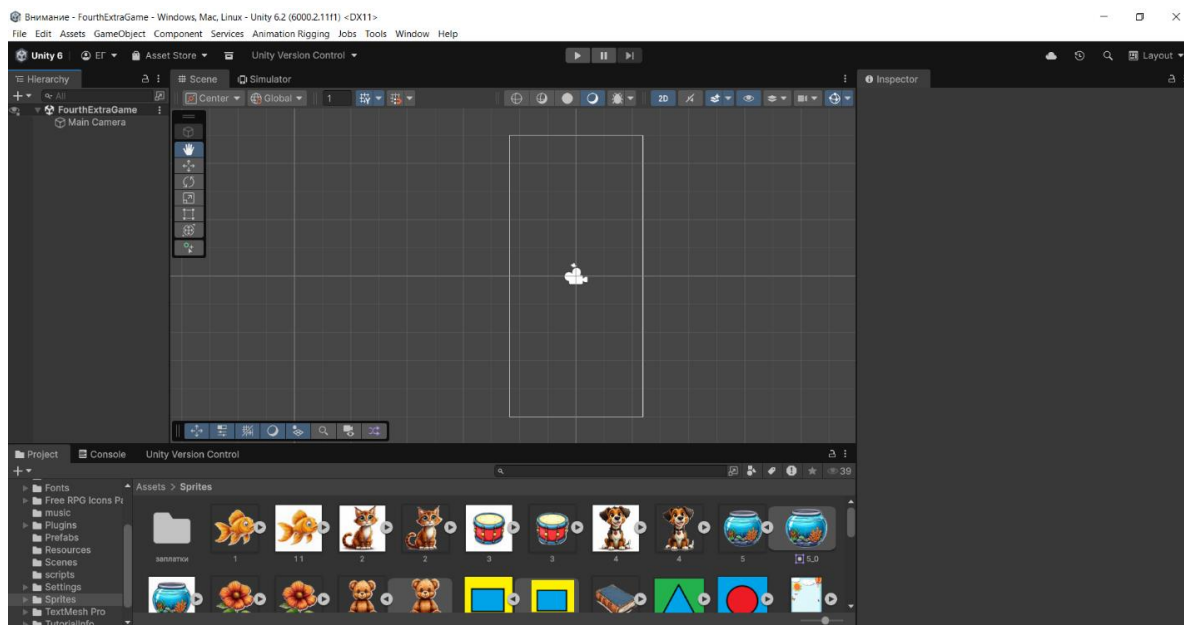


Рисунок 4 – Добавление изображений

Для создания фона игровой сцены необходимо импортировать изображение background из папки Sprites. Затем настроить его размер и положение таким образом, чтобы оно целиком заполняло игровое окно (см.рис.5).

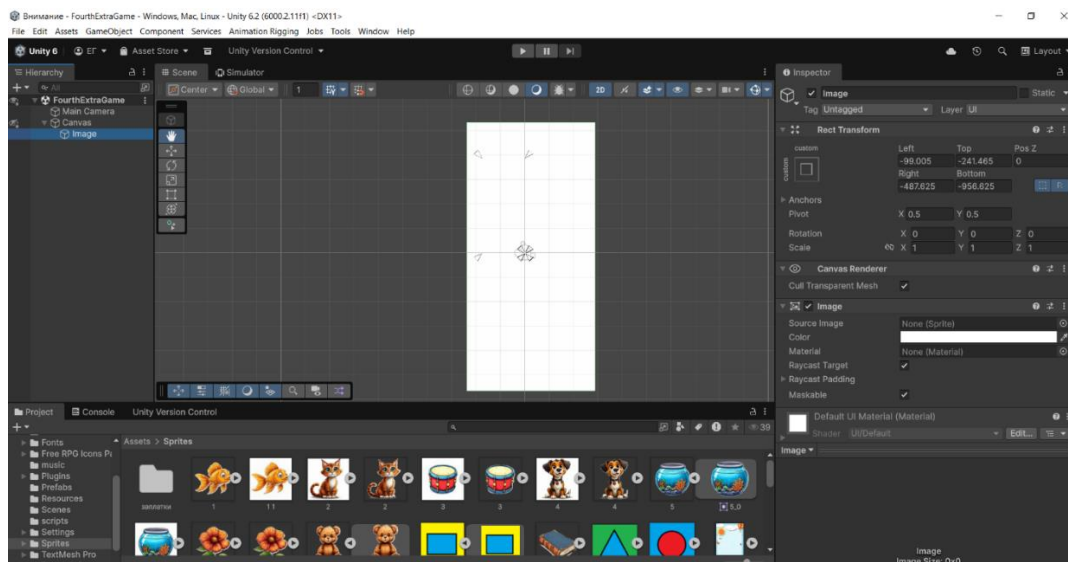


Рисунок 5 – Настройка фона игры

На следующем этапе разработки требуется разместить изображения на сцене, для чего нужно перетащить подготовленные текстуры из окна Project в Hierarchy, создав объекты типа Image, и последовательно присвоить им имена: 1_0, 2_0, 3_0, 4_0, 5_0, 6_0, 1_1, 2_2, 3_3, 4_4, 5_5, 6_6. Для каждого объекта затем выполняется детальная настройка: в компоненте Transform задаются координаты позиционирования (Pos X, Pos Y), ширина и высота, а также корректируются якоря (Anchors) и точка вращения (Pivot) для обеспечения правильного отображения на различных разрешениях экрана; в компоненте Image выбирается исходная текстура (Source Image) и устанавливается тип отображения "Simple". Для добавления интерактивности объекты дополняются компонентом Button, в котором настраиваются цветовые состояния (Normal Color, Highlighted Color, Pressed Color), а также компонент Box Collider 2D, размеры которого регулируются вручную для точного соответствия видимой области изображения и корректной обработки взаимодействий (см.рис.6).

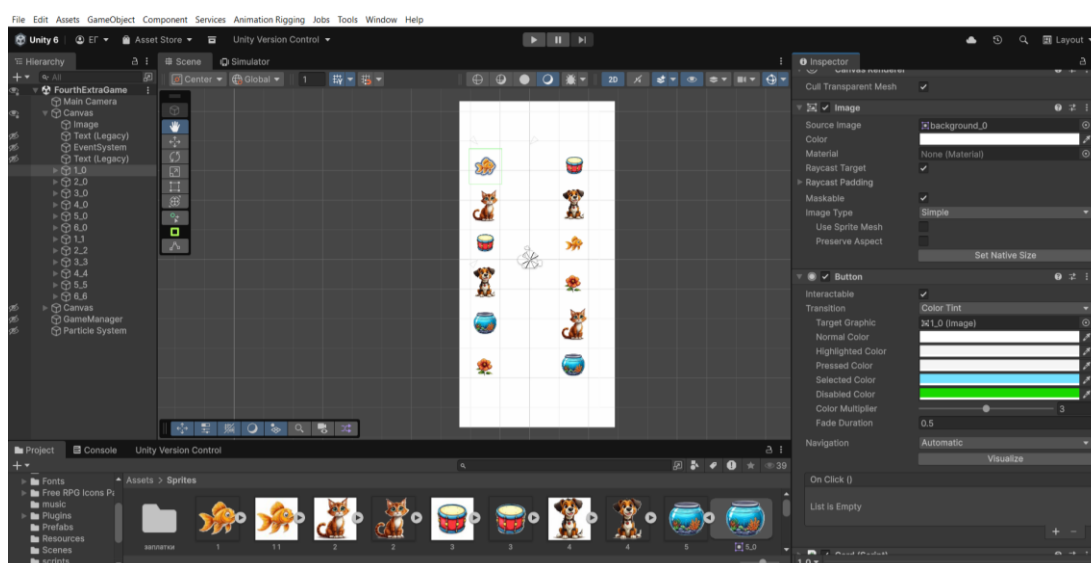


Рисунок 6 – Добавление изображений предметов на сцену

Для реализации отображения инструкции и уведомления «Правильно!» в игре необходимо добавить элементы пользовательского интерфейса. Через контекстное меню в окне Hierarchy создайте панель (UI -> Panel), а затем — текстовый элемент (UI -> Text) для вывода необходимого текста. После добавления этих объектов на сцену выполните их детальную настройку в компоненте Inspector: сначала установите якоря (Anchors) в центральное положение со значениями (0.5, 0.5), что обеспечит корректное позиционирование и адаптацию к разным разрешениям экрана; затем задайте конкретные размеры области — Width (Ширина) 1420 и Height (Высота) 601,8 пиксель. Для визуального оформления панели в компоненте Image выберите подходящий спрайт в поле Source Image, настройте цвет и прозрачность через параметр Color в соответствии со стилем игры, а тип отображения (Image Type) установите на «Simple» (см.рис.7).

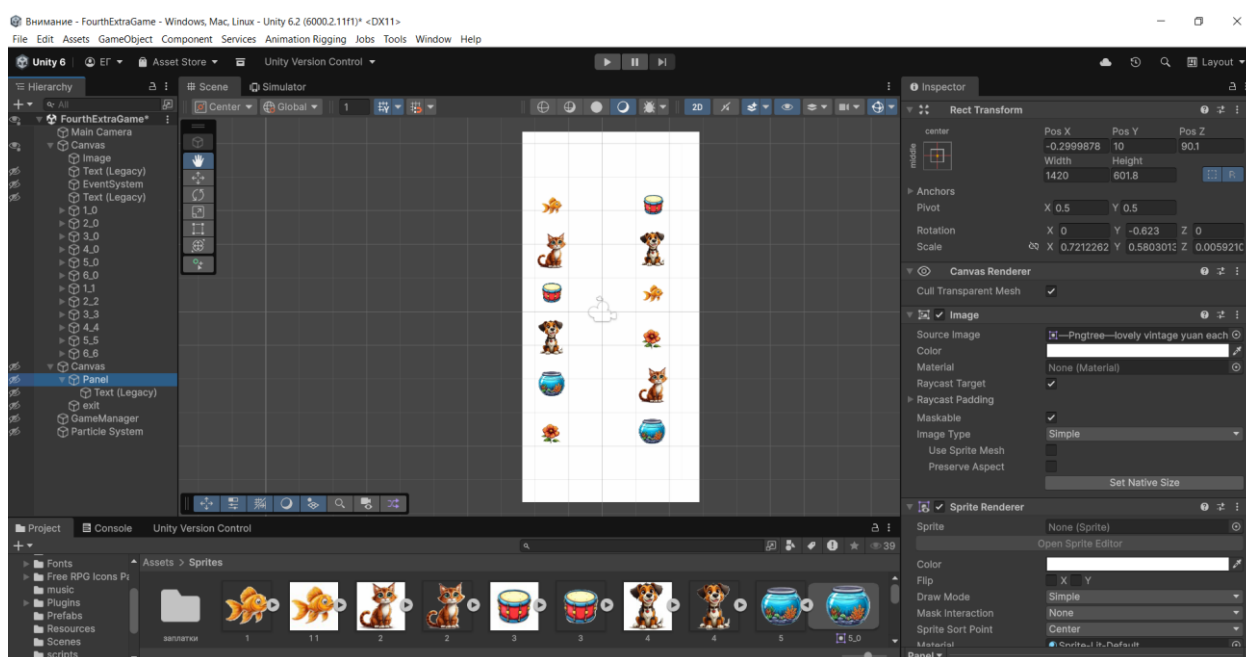


Рисунок 7 – Настройка панели

Далее необходимо настроить текстовый блок для уведомления «Правильно!». В компоненте Text (Legacy) устанавливаем шрифт PlaypenSans-VariableFont_wght с начертанием Bold и размер 100 пунктов, чтобы обеспечить хорошую читаемость. В разделе Rect Transform задайте размеры поля: Width 1041 и Height 347. Настройте якоря (Anchors) с центральной точкой (0.5, 0.5) — это гарантирует правильное отображение на экранах с разным разрешением. Затем настройте сам текст: установите выравнивание по центру, выберите цветовое оформление и настройте параметры переноса для гармоничного внешнего вида в интерфейсе игры (см.рис. 8).

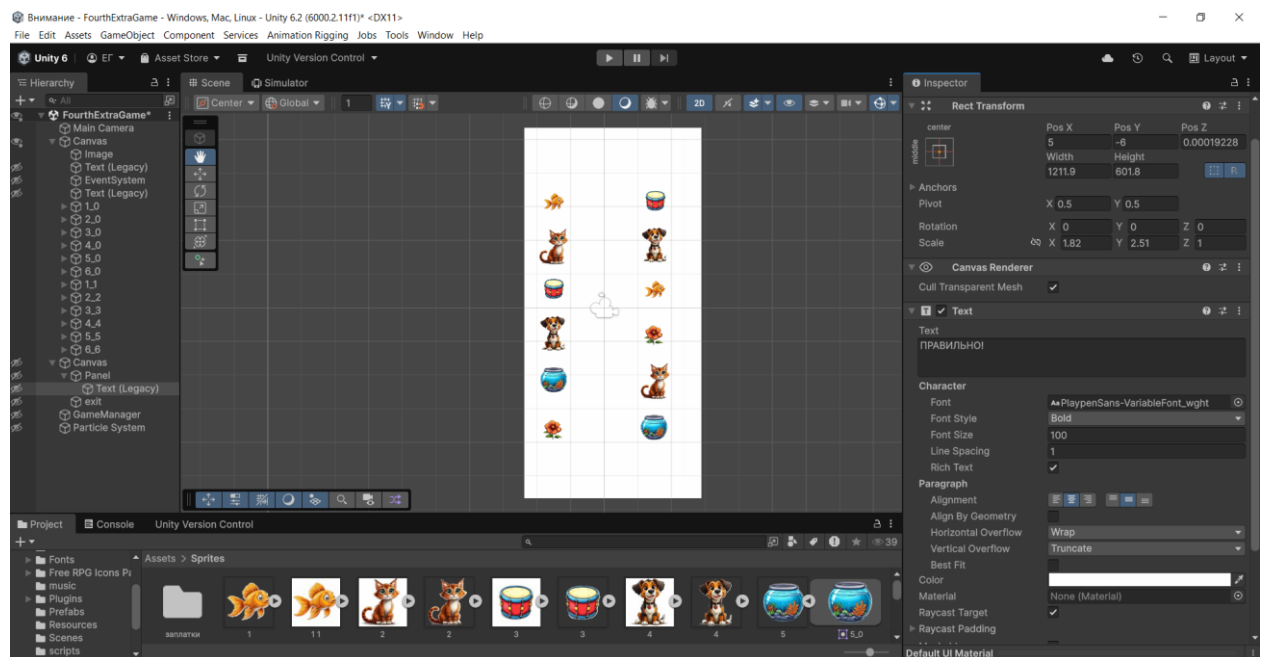


Рисунок 8 – Настройка текстового элемента панели сообщений

Для текстового элемента настраиваем шрифт Pangolin-Regular с жирным начертанием и размером 65 пунктов с жирным начертанием Bold, выравнивание по центру, включаем поддержку Rich Text для форматирования, а в Rect Transform задаем точные размеры текстового поля: Width 567,76, Height 317,38, и устанавливаем якоря Anchors с центром в точке (0.5, 0.5) (см.рис.9).

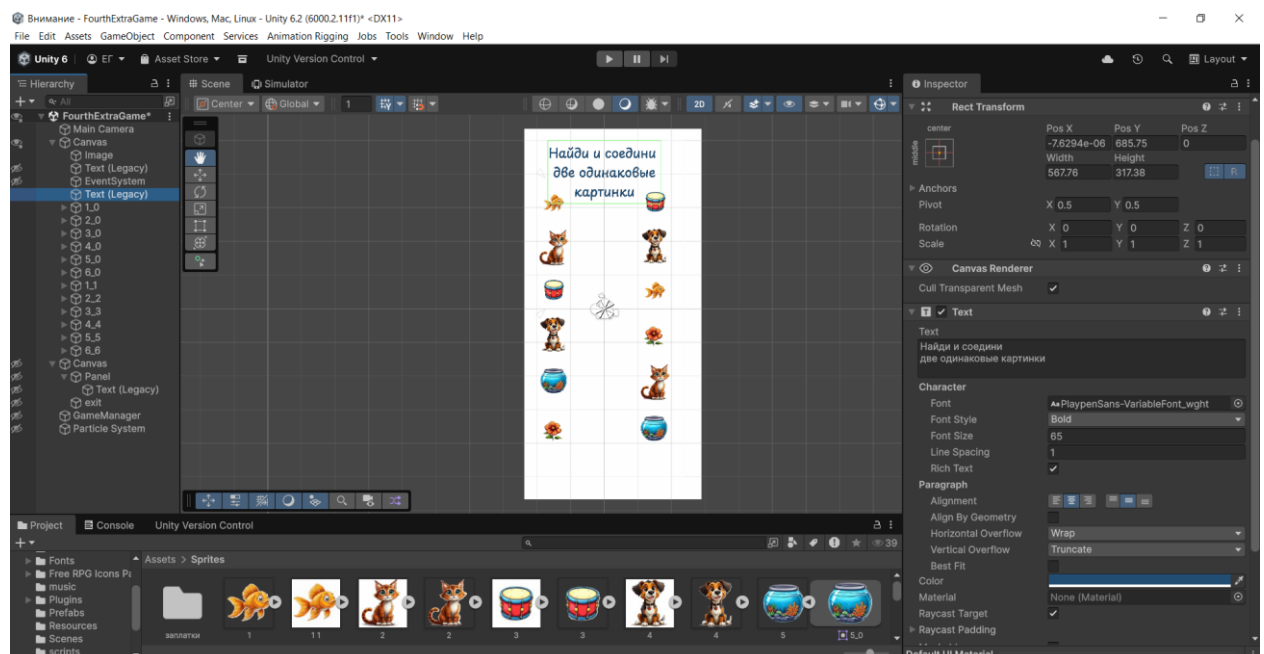


Рисунок 9 – Настройка инструкции игры

Также необходимо создать компонент GameManager, 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 GameManager. 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).

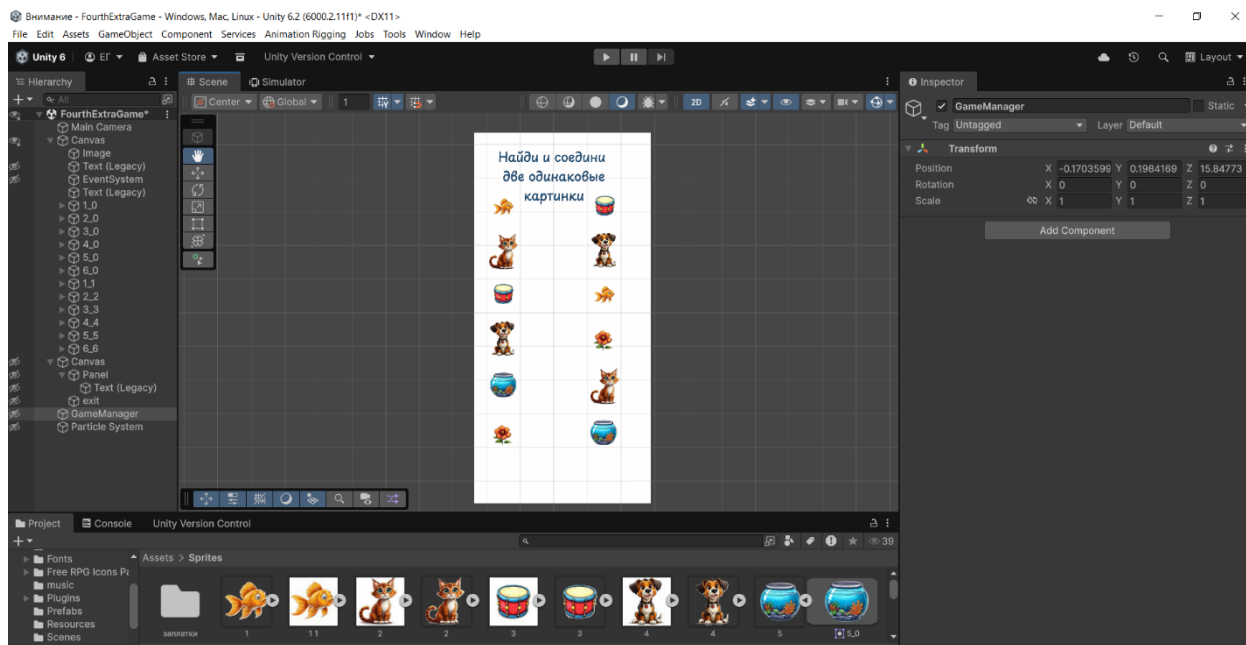


Рисунок 10 – Добавление пустого объекта

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

Центральное управление игровым процессом реализовано через паттерн Singleton, гарантирующий единственный экземпляр менеджера на сцене. В методе Awake() происходит инициализация этого экземпляра, настройка компонента AudioSource для воспроизведения звуков и скрытие панели победы.

Логика выбора карточек обрабатывается в методе SelectCard(): он отслеживает выбранные игроком карточки, проверяет допустимость выбора и управляет процессом сопоставления. Проверка совпадения выполняется в корутине CheckMatch(), где сравниваются pairID выбранных карт. В зависимости от результата линия соединения окрашивается в зелёный (правильно) или красный (ошибка) цвет, а карточки либо помечаются как найденные, либо возвращаются в исходное состояние.

Звуковое сопровождение обеспечивается методом PlaySound(), который проигрывает соответствующие аудиоклипы для различных событий: выбор карточки, успешное или неудачное совпадение, победа. Завершение игры отслеживается методом CheckGameCompletion(): после нахождения всех пар карт активируется панель победы, и через заданный интервал времени игра автоматически возвращает в меню.

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

public class GameManager : MonoBehaviour
{
    public static GameManager 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()
    {
        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()
{
    if (Input.GetKeyDown(KeyCode.T))
    {
        TestLine();
    }

    if (Input.GetKeyDown(KeyCode.Y))
    {
        if (lineRenderer != null)
        {
            lineRenderer.HideLine();
            Debug.Log("Line hidden");
        }
    }

    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)
        {
            RectTransform cardRect = card.GetComponent<RectTransform>();
            if (cardRect != null)
            {
                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)
        {
            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()
{
    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!");
        }

        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");
    }

    public void GoToMenu()
    {
        LoadMenuScene();
    }

    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}");
    }

    private void TestLine()
    {
        if (lineRenderer != null)
        {
            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 GameManager!");
        }
    }
}

```

После создания скрипта GameManager необходимо добавить его к компоненту GameManager. 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 соответствующие звуковые клипы основным игровым событиям — выбору карточки, правильному и неправильному сопоставлению, и победе в игре; в блоке Victory Panel нужно связать объекты пользовательского интерфейса — панель победы и текстовый элемент для отображения поздравительного сообщения (см.рис.11)

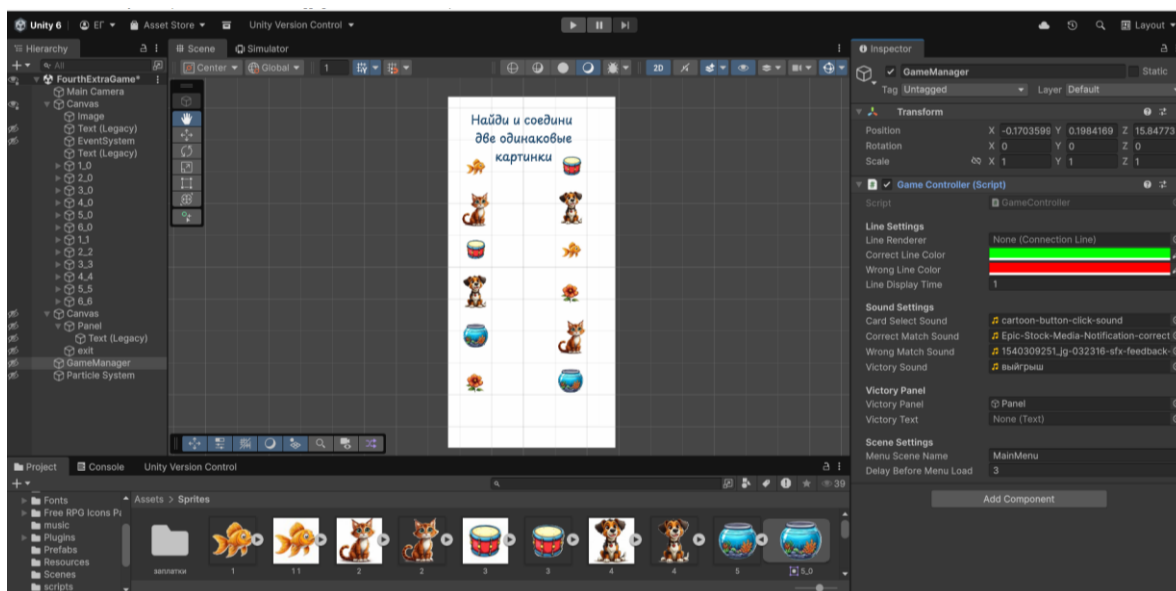


Рисунок 11 – Добавление скрипта к компоненту GameManager и его настройка

Далее необходимо создать скрипт Card, отвечающий за индивидуальное поведение каждой игровой карточки. Он реализует интерфейсы IPointerDownHandler и IPointerUpHandler для обработки нажатий и отпускания указателя, обеспечивая прямую реакцию на действия пользователя.

При взаимодействии с карточкой активируется визуальный отклик: плавное анимированное изменение цвета её рамки, реализованное с помощью DOTween. Одновременно с этим событие о выборе карточки передается в центральный управляющий класс GameManager через вызов метода SelectCard().

Каждая карточка хранит уникальный pairID (идентификатор пары), который определяет её логическую связь с парной карточкой. Флаг IsMatched отслеживает, найдена ли уже пара для данной карточки. После успешного совпадения карточка переходит в неактивное состояние: визуально затемняется за счет применения полупрозрачных цветов, что блокирует её повторный выбор и наглядно показывает игроку завершенность пары.

Скрипт также содержит метод ResetCard(), предназначенный для сброса состояния карточки. Этот метод восстанавливает исходные цветовые параметры и повторно активирует возможность взаимодействия, что обеспечивает корректную работу при перезапуске игрового раунда.

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;
using DG.Tweening;

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 добавить скрипт Card на каждый объект карточки в иерархии, перетаскив его в окно Inspector, после чего выполнить детальную настройку параметров: в поле Pair ID указывается уникальный числовой идентификатор, связывающий карточки в логические пары (одинаковые значения обозначают соответствие), а в разделе Visuals настраиваются визуальные компоненты — в Border Image назначается объект с компонентом Image для отображения рамки, в Content Image указывается изображение с основным контентом, а параметр Selected Color определяет цвет подсветки

рамки при выборе, обеспечивая визуальную обратную связь; для корректной работы важно убедиться, что все визуальные компоненты правильно связаны, а значение Pair ID согласовано с парной карточкой, что гарантирует точное распознавание совпадений в игровой логике (см. рис. 12).

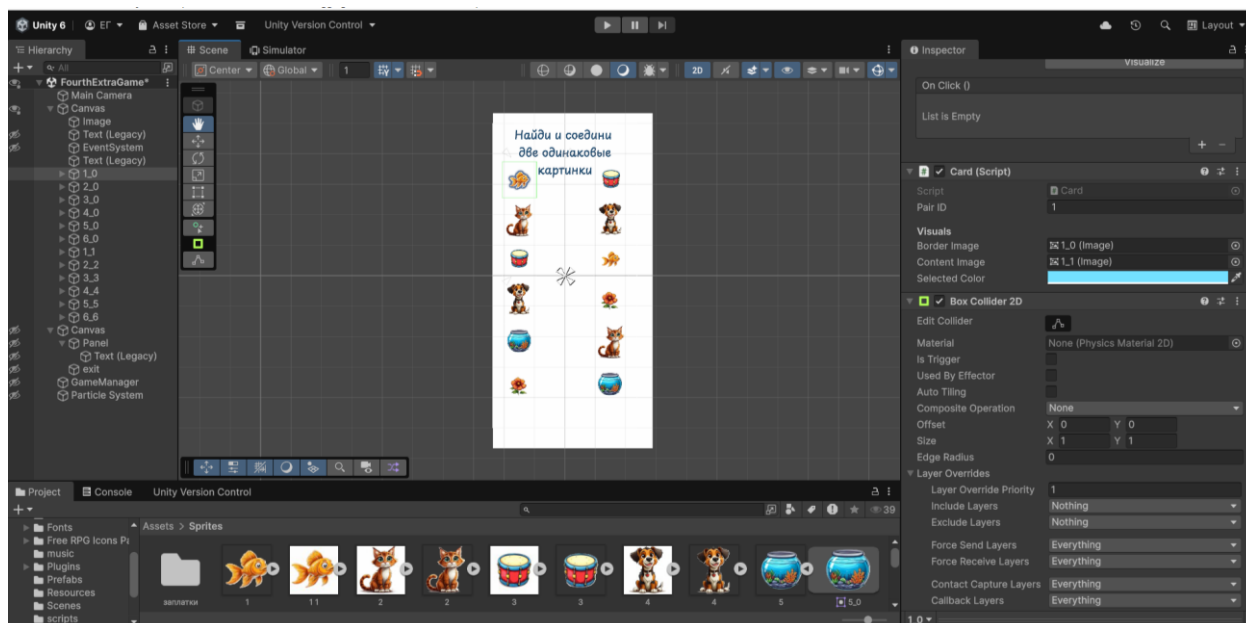


Рисунок 12 – Добавление скрипта ко всем компонентам карточек

Итоговый результат создания игры «Найди одинаковые картинки» (см. рис.13).



Рисунок 13 – Готовая игра «Найди одинаковые картинки»

4. Выводы

В данной работе был показан процесс создания игры «Найди одинаковые картинки» 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 / Я. А. Фаст // 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.