

Создание мобильной игры «Неваляшки» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Целью статьи является создание мобильной игры «Неваляшки» 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 «Неваляшки».

Ключевые слова: Unity, игра, неваляшки, пара.

Creating a Tilt-A-Whirl Mobile game with unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create a mobile game called "Tumblers" 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 "Tumblers" was created.

Keywords: Unity, game, tumblers, couple.

1 Введение

1.1 Актуальность

Мобильные игры прочно вошли в современную цифровую культуру, представляя собой одну из самых динамичных и финансово успешных отраслей индустрии развлечений. Особое место в этом сегменте занимают казуальные проекты, которые благодаря простым правилам, лаконичному дизайну и увлекательной механике привлекают самую широкую аудиторию.

Разработка игры «Неваляшки», построенной на механике поиска и совпадения пар одинаковых персонажей, полностью соответствует текущим рыночным трендам. Такой проект служит не только идеальным развлечением для коротких игровых сессий, но и выполняет полезную функцию, способствуя тренировке внимания, зрительной памяти, скорости реакции и концентрации. Это определяет его ценность как в игровой, так и в развивающей сфере, где он может выступать в роли эффективного тренажера для игроков всех возрастов.

Реализация проекта в среде Unity является наиболее эффективным и современным решением. Этот движок предоставляет мощный и гибкий инструментарий для кроссплатформенной разработки, позволяя создавать яркую, стилизованную графику, очаровательных анимированных неваляшек с уникальным дизайном и интуитивные интерфейсы. Unity позволяет сосредоточиться на качестве контента, плавности геймплея и «отзывчивости» управления, а также обеспечивает высокую производительность, простую интеграцию систем подсчета очков и прогресса, и лёгкую публикацию в основных магазинах приложений. Благодаря встроенным возможностям для 2D-разработки и работе с анимацией, создание привлекательных и живых неваляшек становится прямой и выполнимой задачей.

Таким образом, разработка мобильной игры «Неваляшки» на Unity — это актуальная задача, объединяющая популярный игровой жанр (пазл/поиск пар), полезный для когнитивных навыков геймплей и современные технологии. Проект имеет практическую значимость, соответствует рыночному спросу на лёгкий и увлекательный контент и открывает широкие возможности для дальнейшего расширения за счёт добавления новых уровней сложности, коллекций неваляшек в различных тематических стилях, специальных бонусов 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 «Неваляшки», используя возможности среды разработки Unity для создания интерактивных 2D/3D-приложений.

2 Материалы и методы

Для разработки игры «Неваляшки» выбран движок Unity. Решение 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

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. 1).

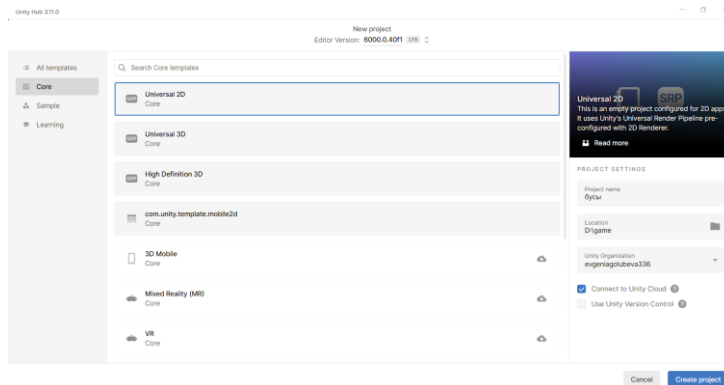


Рисунок 1 – Создание проекта

Следующий этап разработки — подготовка проекта для запуска на мобильных устройствах. Для этого в Unity необходимо открыть окно настроек сборки (File → Build Settings), а затем в списке доступных платформ указать Android в качестве целевой (см. рис. 2).

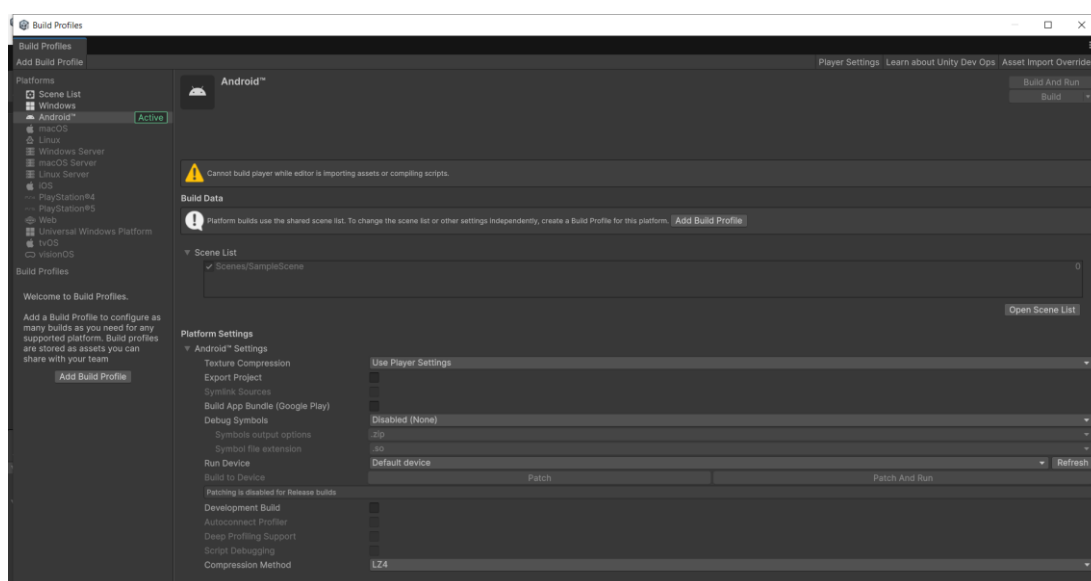


Рисунок 2 – Настройка проекта

Когда базовая настройка проекта завершена, следующим логичным этапом является построение игровой сцены. Её реализацию следует начать с настройки камеры, так как она определяет область видимости для игрока (см. рис. 3).

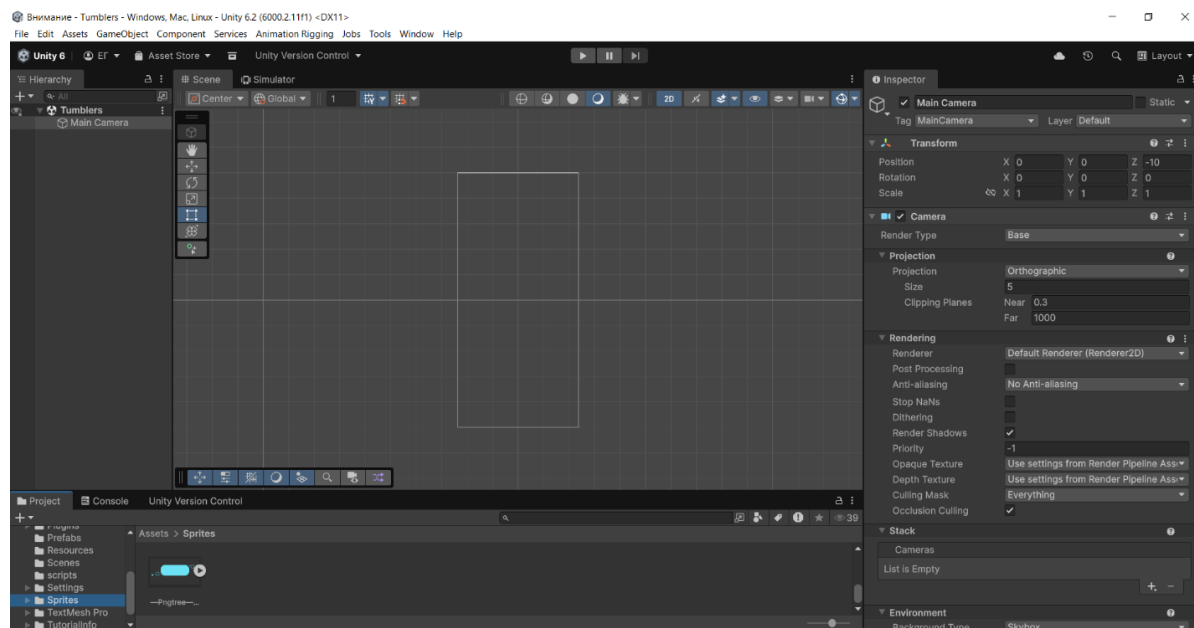


Рисунок 3 – Настройка камеры

Следующим шагом является организация графических ресурсов. Необходимо разместить подготовленные спрайты в предназначенную для них директорию Sprites (см.рис.4).

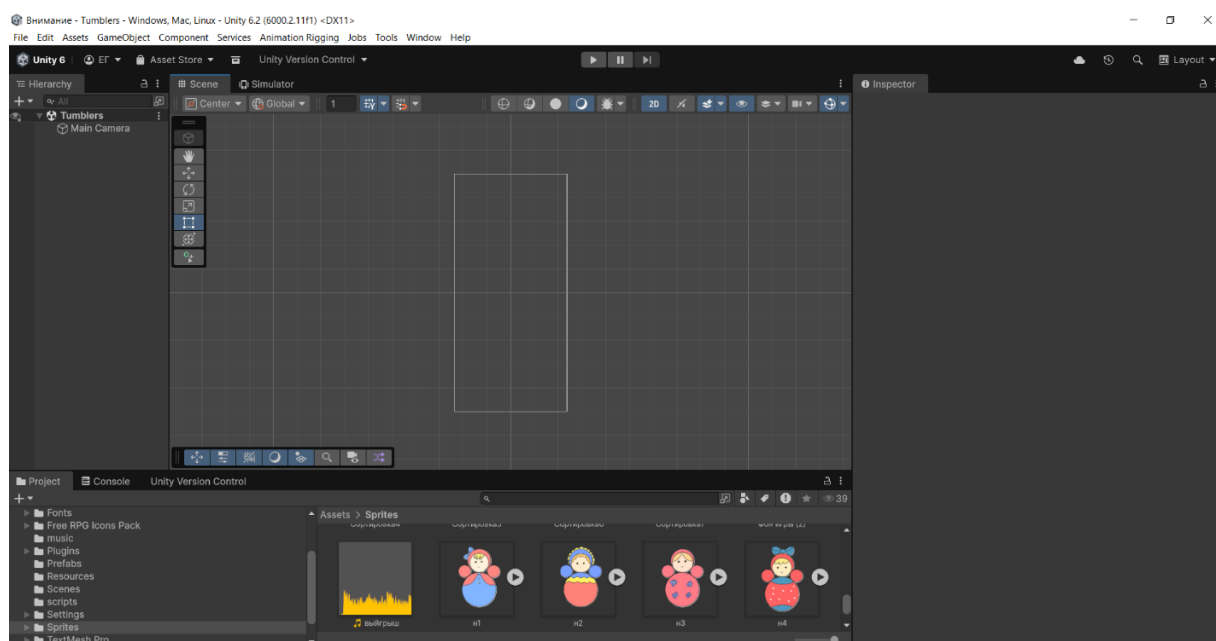


Рисунок 4 – Добавление изображений

Далее необходимо создать фон 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).

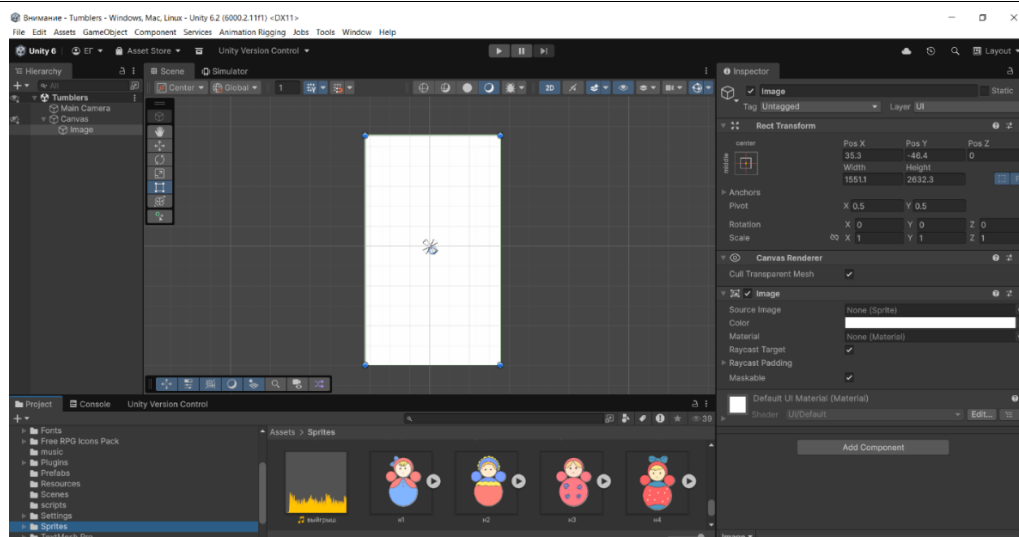


Рисунок 5 – Настройка фона игры

После создание фона сцены, необходимо разместить изображения неваляшек. Для того, чтобы разместить неваляшек на фон, необходимо создать шесть пустых объектов, которые будут служить контейнерами для изображений неваляшек. В Hierarchy нажимаем правой кнопкой мыши, и в контекстном меню выбираем Create empty и переименовываем в 1, 2 3, 4, 5 и 6. После создание объектов необходимо в созданные объекты добавить компонент Button и Create empty который необходимо переименовать в DollSprite, и так необходимо сделать и в остальных объектах. На объект DollSprite следует добавить компонент Image и назначить ему соответствующий спрайт неваляшки из папки Assets/Sprites. Важно настроить параметры Rect Transform для каждого контейнера и вложенного DollSprite, чтобы задать их размер, позицию и привязку на экране, а также настроить свойства кнопки — визуальные состояния (Normal, Highlighted, Pressed цвета) и привязать метод обработки нажатия в секции On Click(). Эта структура позволит удобно управлять неваляшками как самостоятельными интерактивными элементами и легко масштабировать игру за счёт добавления новых пар (см.рис.6).

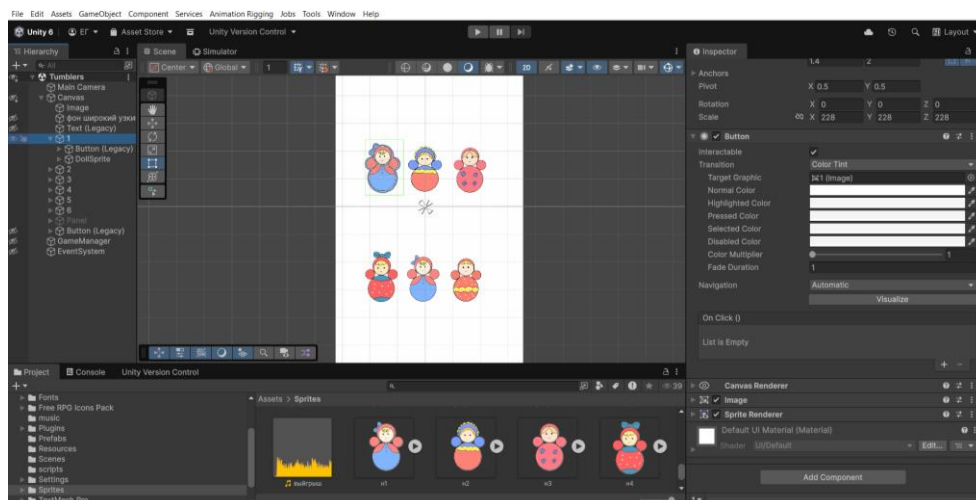


Рисунок 6 – Добавление объектов для неваляшек и настройка

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, 1080 на 369,3 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.7).

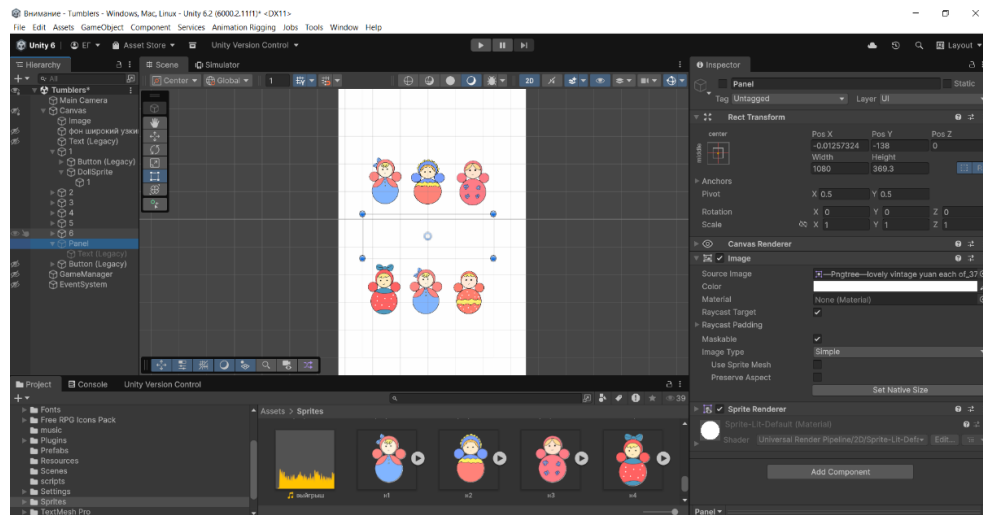


Рисунок 7 – Настройка панели

Для вывода уведомления «Правильно!» выполняется детальная настройка текстового элемента. В компоненте Text (Legacy) выбирается шрифт PlaypenSans-VariableFont_wght с начертанием Bold и устанавливается размер 129 пунктов. В параметрах Rect Transform задаются размеры: ширина 891,3 и высота 262,8 пикселей, и якоря (Anchors) фиксируется в позиции (0.5, 0.5) (см.рис. 8).

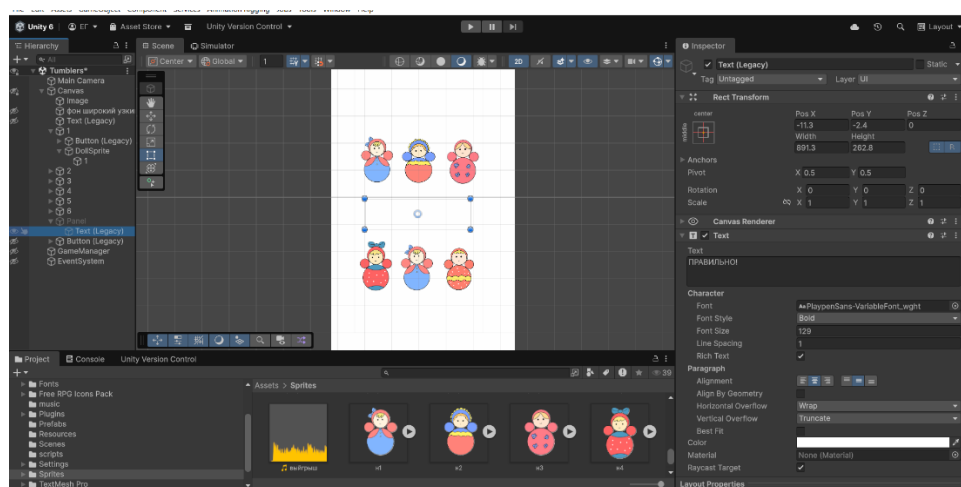


Рисунок 8 – Настройка текстового элемента панели сообщений

Для настройки текстового элемента выбирается шрифт Pangolin-Regular с жирным начертанием Bold и размер шрифта 80 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.9).

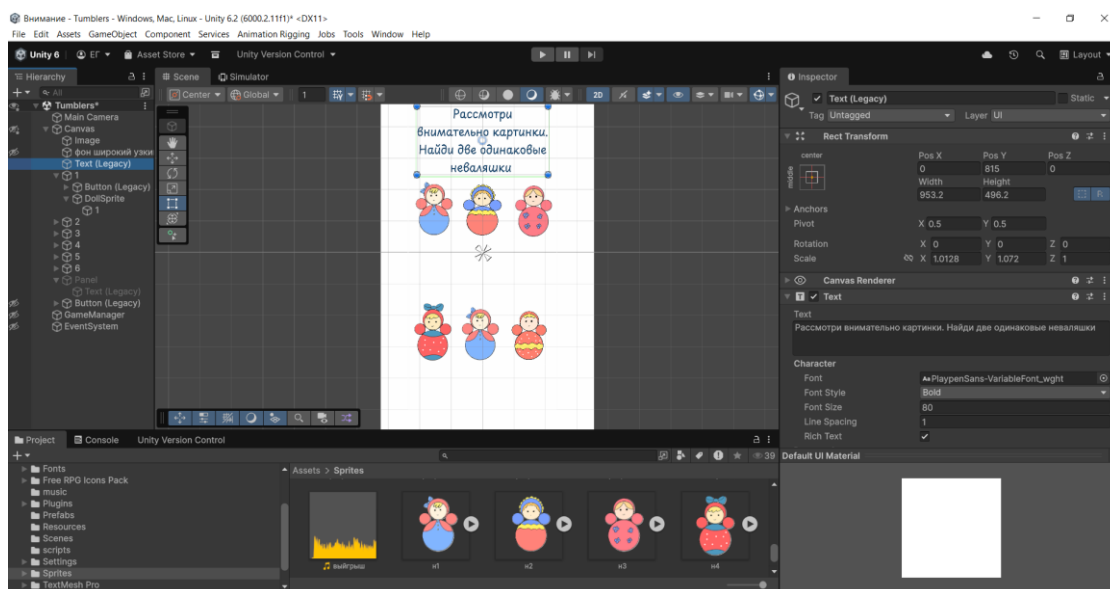


Рисунок 9 – Настройка инструкции игры

Скрипт DollGameController.cs управляет всей логикой игры по нахождению одинаковых неваляшек. После инициализации синглтона и настройки аудио запускается корутина, ожидающая полной загрузки всех игровых объектов. В методе InitializeDolls() двум правильным неваляшкам назначаются одинаковые спрайты и ID, а остальным четырём — уникальные.

Основная механика реализована в SelectDoll(): при выборе двух неваляшек рисуется соединительная линия и проверяется совпадение их идентификаторов. При успешном совпадении воспроизводится звук победы, пара отмечается как найденная и проверяется условие завершения игры. При несовпадении состояние сбрасывается.

При нахождении необходимой пары активируется панель победы с последующей автоматической загрузкой меню. Скрипт также включает функции сброса игры, отладки и проверки состояния игровых объектов.

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

public class DollGameController : MonoBehaviour
{
    public static DollGameController Instance;

    [Header("Doll References")]
    [SerializeField] private List<DollCard> dollCards = new List<DollCard>();
```

```

[SerializeField] private int correctDollIndex1 = 0;
[SerializeField] private int correctDollIndex2 = 1;
[Header("Line Settings")]
[SerializeField] private DollConnectionLine lineRenderer;
[SerializeField] private Color correctLineColor = Color.clear;
[SerializeField] private Color wrongLineColor = Color.clear;
[SerializeField] private float lineDisplayTime = 1f;
[Header("Sound Settings")]
[SerializeField] private AudioClip dollSelectSound;
[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;
[Header("Doll Settings")]
[SerializeField] private Sprite correctDollSprite;
[SerializeField] private Sprite[] uniqueDollSprites;

private List<DollCard> selectedDolls = new List<DollCard>();
private AudioSource audioSource;
private int matchedPairsCount = 0;
private const int requiredPairsToWin = 1;
private const int CORRECT_DOLL_TYPE_ID = 100;
public bool CanSelect { get; private set; } = true;
private bool isInitialized = false;
private bool isGameComplete = false;

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

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

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

void Start()
{
    StartCoroutine(InitializeGameWithDelay());
}

private IEnumerator InitializeGameWithDelay()
{
    yield return null;
    InitializeGame();
}

void InitializeGame()
{
    isGameComplete = false;

    if (lineRenderer == null)
    {
        lineRenderer = FindFirstObjectByType<DollConnectionLine>();
    }

    if (audioSource == null)
    {
    }

    if (dollCards.Count == 0)
    {
        DollCard[] foundDolls = FindObjectsByType<DollCard>(FindObjectsSortMode.None);
        if (foundDolls.Length > 0)
        {

```

```

        dollCards.AddRange(foundDolls);
        dollCards.Sort((a, b) => a.name.CompareTo(b.name));
    }
    else
    {
        return;
    }
}

if (correctDollIndex1 < 0 || correctDollIndex1 >= dollCards.Count ||
    correctDollIndex2 < 0 || correctDollIndex2 >= dollCards.Count)
{
    return;
}

if (correctDollIndex1 == correctDollIndex2)
{
    return;
}

InitializeDolls();
CheckAllDollsStatus();
isInitialized = true;
}

void InitializeDolls()
{
    if (dollCards.Count < 6)
    {
        return;
    }

    if (correctDollSprite == null)
    {
        correctDollSprite = CreateTestSprite(Color.clear, "CorrectDoll");
    }

    if (uniqueDollSprites == null || uniqueDollSprites.Length < 4)
    {
        uniqueDollSprites = new Sprite[4];
        Color[] colors = { Color.clear, Color.clear, Color.clear, Color.clear };

        for (int i = 0; i < 4; i++)
        {
            uniqueDollSprites[i] = CreateTestSprite(colors[i], $"UniqueDoll_{i}");
        }
    }

    List<Sprite> availableUniqueSprites = new List<Sprite>(uniqueDollSprites);
    ShuffleList(availableUniqueSprites);

    for (int i = 0; i < dollCards.Count; i++)
    {
        if (dollCards[i] == null)
        {
            continue;
        }

        if (i == correctDollIndex1 || i == correctDollIndex2)
        {
            dollCards[i].SetDollSprite(correctDollSprite);
            dollCards[i].SetDollTypeID(CORRECT_DOLL_TYPE_ID);
        }
        else
        {
            int spriteIndex = (i - 2) % availableUniqueSprites.Count;
            if (spriteIndex < 0) spriteIndex = 0;

            dollCards[i].SetDollSprite(availableUniqueSprites[spriteIndex]);
            dollCards[i].SetDollTypeID(i);
        }

        dollCards[i].ResetDoll();
    }
}

private void ShuffleList<T>(List<T> list)
{
    for (int i = 0; i < list.Count; i++)
    {

```

```
        T temp = list[i];
        int randomIndex = Random.Range(i, list.Count);
        list[i] = list[randomIndex];
        list[randomIndex] = temp;
    }
}

private Sprite CreateTestSprite(Color color, string name)
{
    Texture2D texture = new Texture2D(100, 100);
    Color[] pixels = new Color[100 * 100];
    for (int i = 0; i < pixels.Length; i++)
    {
        pixels[i] = color;
    }
    texture.SetPixels(pixels);
    texture.Apply();

    Sprite sprite = Sprite.Create(texture, new Rect(0, 0, 100, 100), new Vector2(0.5f, 0.5f));
    sprite.name = name;

    return sprite;
}

void Update()
{
    if (!isInitialized || isGameComplete) return;

    if (Keyboard.current != null && Keyboard.current.rKey.wasPressedThisFrame)
    {
        ResetGame();
    }

    if (Keyboard.current != null && Keyboard.current.pKey.wasPressedThisFrame)
    {
        TestVictory();
    }

    if (Keyboard.current != null && Keyboard.current.tKey.wasPressedThisFrame)
    {
        TestLine();
    }

    if (Keyboard.current != null && Keyboard.current.dKey.wasPressedThisFrame)
    {
        DebugGameState();
    }
}

public void SelectDoll(DollCard doll)
{
    if (doll == null)
    {
        return;
    }

    if (selectedDolls.Contains(doll))
    {
        return;
    }

    if (!CanSelect)
    {
        return;
    }

    if (doll.IsMatched)
    {
        return;
    }

    PlaySound(dollSelectSound);
    selectedDolls.Add(doll);

    if (selectedDolls.Count == 1)
    {
        if (lineRenderer != null)
        {
            RectTransform dollRect = doll.GetComponent<RectTransform>();
            if (dollRect != null)
```

```

        {
            Vector3[] corners = new Vector3[4];
            dollRect.GetWorldCorners(corners);
            Vector3 center = (corners[0] + corners[2]) / 2f;
            lineRenderer.StartLine(center);
        }
        else
        {
            lineRenderer.StartLine(doll.transform.position);
        }
    }
}
else if (selectedDolls.Count == 2)
{
    if (lineRenderer != null)
    {
        RectTransform doll1Rect = selectedDolls[0].GetComponent<RectTransform>();
        RectTransform doll2Rect = selectedDolls[1].GetComponent<RectTransform>();

        Vector3 pos1 = doll1Rect != null ? GetRectCenter(doll1Rect) :
selectedDolls[0].transform.position;
        Vector3 pos2 = doll2Rect != null ? GetRectCenter(doll2Rect) :
selectedDolls[1].transform.position;

        lineRenderer.SetLinePoints(pos1, pos2);
    }

    StartCoroutine(CheckMatch());
}
else if (selectedDolls.Count > 2)
{
    selectedDolls.Clear();
}
}

private Vector3 GetRectCenter(RectTransform rect)
{
    Vector3[] corners = new Vector3[4];
    rect.GetWorldCorners(corners);
    return (corners[0] + corners[2]) / 2f;
}

private IEnumerator CheckMatch()
{
    CanSelect = false;

    DollCard firstDoll = selectedDolls[0];
    DollCard secondDoll = selectedDolls[1];

    bool isMatch = firstDoll.dollTypeID == secondDoll.dollTypeID && firstDoll.dollTypeID ==
CORRECT_DOLL_TYPE_ID;

    if (lineRenderer != null)
    {
        Color lineColor = isMatch ? correctLineColor : wrongLineColor;
        lineRenderer.SetLineColor(lineColor);
    }

    if (isMatch)
    {
        PlaySound(correctMatchSound);
        yield return new WaitForSeconds(lineDisplayTime);

        firstDoll.MarkAsMatched();
        secondDoll.MarkAsMatched();

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

        firstDoll.ResetVisual();
        secondDoll.ResetVisual();

        if (lineRenderer != null)
            lineRenderer.HideLine();
    }
}

```

```
        selectedDolls.Clear();
        CanSelect = true;
    }

    private void PlaySound(AudioClip clip)
    {
        if (audioSource != null && clip != null)
        {
            audioSource.PlayOneShot(clip);
        }
    }

    private void CheckGameCompletion()
    {
        if (matchedPairsCount >= requiredPairsToWin)
        {
            StartCoroutine(GameComplete());
        }
    }

    private IEnumerator GameComplete()
    {
        isGameComplete = true;

        yield return new WaitForSeconds(0.5f);

        PlaySound(victorySound);

        if (victoryPanel != null)
        {
            victoryPanel.SetActive(true);
            if (victoryText != null)
                victoryText.text = "Поздравляю! Нашёл одинаковые неваляшки!";
        }

        yield return new WaitForSeconds(delayBeforeMenuLoad);
        LoadMenuScene();
    }

    private void LoadMenuScene()
    {
        if (!string.IsNullOrEmpty(menuSceneName))
        {
            SceneManager.LoadScene(menuSceneName);
        }
    }

    public void ResetGame()
    {
        isGameComplete = false;
        selectedDolls.Clear();
        matchedPairsCount = 0;
        CanSelect = true;

        foreach (DollCard doll in dollCards)
        {
            if (doll != null)
            {
                doll.ResetDoll();
            }
        }

        StartCoroutine(ResetGameCoroutine());
    }

    private IEnumerator ResetGameCoroutine()
    {
        yield return null;

        if (lineRenderer != null)
            lineRenderer.HideLine();

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

        if (uniqueDollSprites != null && uniqueDollSprites.Length >= 4)
        {
            List<Sprite> sprites = new List<Sprite>(uniqueDollSprites);
            ShuffleList(sprites);
        }
    }
}
```

```

        uniqueDollSprites = sprites.ToArray();
    }

    InitializeDolls();
}

public void GoToMenu()
{
    LoadMenuScene();
}

public void DebugGameState()
{
}

private void CheckAllDollsStatus()
{
}

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.clear);
    }
}

private void TestVictory()
{
    matchedPairsCount = 1;
    CheckGameCompletion();
}

public void AddDoll(DollCard doll)
{
    if (doll != null && !dollCards.Contains(doll))
    {
        dollCards.Add(doll);
    }
}

public string GetCorrectDollsInfo()
{
    if (correctDollIndex1 < dollCards.Count && correctDollIndex2 < dollCards.Count)
    {
        return $"Correct dolls: {dollCards[correctDollIndex1].name} (Index: {correctDollIndex1}) and " +
            $"{dollCards[correctDollIndex2].name} (Index: {correctDollIndex2})";
    }
    return "Correct dolls not set properly";
}

public bool IsCorrectDoll(DollCard doll)
{
    return doll != null && doll.dollTypeID == CORRECT_DOLL_TYPE_ID;
}
}

```

Далее необходимо создать пустой объект для скрипта. Для этого в Hierarchy нажимаем правой кнопкой мыши, и в контекстном меню выбирает Create empty и переименовываем в GameManager. Скрипт DollGameController.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. 10).

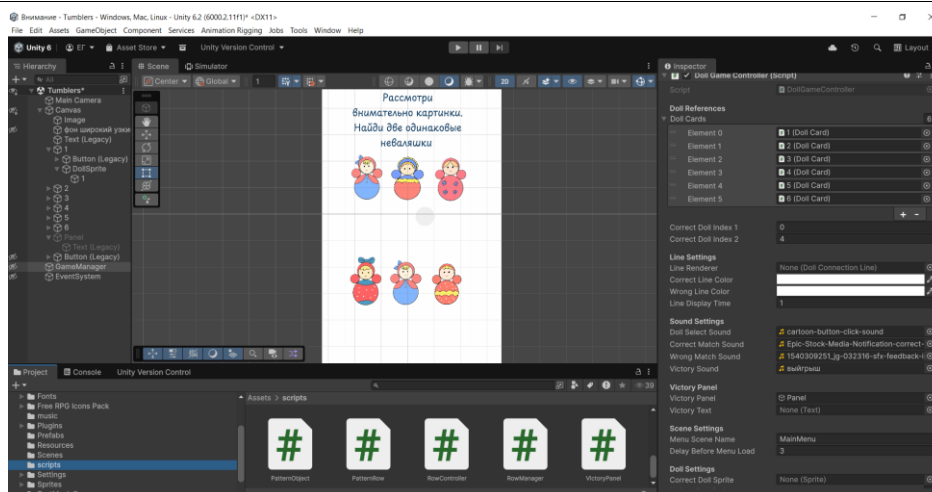


Рисунок 10 – Добавление скрипта к компоненту GameManager и настройка компонента

Скрипт DollCard.cs является фундаментальным элементом игрового взаимодействия, управляя индивидуальным поведением каждой карточки-неваляшки. Скрипт DollCard.cs обрабатывает взаимодействие с карточками-неваляшками через интерфейсы IPointerDownHandler/IPointerUpHandler, обеспечивая прямую реакцию на клики. При выборе карточки запускается анимация (масштаб и цвет рамки через DOTween), а событие передаётся в DollGameController. Уникальный dollTypeID определяет принадлежность к паре, флаг IsMatched блокирует повторный выбор после совпадения. При нахождении пары выполняется анимация исчезновения с последующим скрытием. Метод ResetDoll() полностью восстанавливает исходное состояние для перезапуска игры.

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

[RequireComponent(typeof(Image))]
public class DollCard : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
    [HideInInspector] public int dollTypeID; // ID типа неваляшки
    public bool IsMatched { get; private set; }

    [Header("Visual Components")]
    [SerializeField] private Image borderImage; // Фон/рамка
    [SerializeField] private Image dollImage; // Изображение неваляшки

    [Header("Colors")]
    [SerializeField] private Color selectedColor = new Color(0.8f, 0.8f, 0.1f, 1f);
    [SerializeField] private Color defaultColor = new Color(0.9f, 0.9f, 0.9f, 1f);

    private DollGameController gameController;
    private Vector3 originalScale;
    private bool isClickable = true;
    private float lastClickTime = 0f;
    private const float CLICK_DELAY = 0.3f;
    private Sequence matchSequence;

    void Awake()
    {
        Button[] buttons = GetComponents<Button>();
        foreach (Button btn in buttons)
        {
            Destroy(btn);
        }
    }
}
```

```
        if (borderImage == null)
        {
            borderImage = GetComponent<Image>();
        }
        if (borderImage == null)
        {
            Debug.LogWarning($"No borderImage found on {gameObject.name}. Creating one.");
            borderImage = gameObject.AddComponent<Image>();
        }
        borderImage.raycastTarget = true;
    }

    void Start()
    {
        Debug.Log($"DollCard Start called for {gameObject.name}");

        gameController = DollGameController.Instance;

        if (gameController == null)
        {
            Debug.LogError($"GameController not found for {gameObject.name}");
            return;
        }
        Image img = GetComponent<Image>();
        if (img == null)
        {
            Debug.LogError($"No Image component on {gameObject.name}! Adding one.");
            img = gameObject.AddComponent<Image>();
            img.raycastTarget = true;
        }
        else if (!img.raycastTarget)
        {
            Debug.LogWarning($"Image raycastTarget is false on {gameObject.name}. Enabling it.");
            img.raycastTarget = true;
        }

        gameController.AddDoll(this);

        InitializeVisualComponents();

        originalScale = transform.localScale;

        Debug.Log($" {gameObject.name} initialized successfully.");
    }

    private void InitializeVisualComponents()
    {
        if (dollImage == null)
        {
            Image[] childImages = GetComponentsInChildren<Image>();
            foreach (Image img in childImages)
            {
                if (img.gameObject != gameObject) // Это дочерний объект
                {
                    dollImage = img;
                    break;
                }
            }

            if (dollImage == null)
            {
                CreateDollImage();
            }
        }
    }

    private void CreateDollImage()
    {
        GameObject dollImgObj = new GameObject("DollSprite");
        dollImgObj.transform.SetParent(transform);
        dollImgObj.transform.localPosition = Vector3.zero;
        dollImgObj.transform.localScale = Vector3.one;

        dollImage = dollImgObj.AddComponent<Image>();
        dollImage.preserveAspect = true;
        dollImage.raycastTarget = false; // Дочерний не должен перехватывать клики

        RectTransform rt = dollImgObj.GetComponent<RectTransform>();
        rt.anchorMin = new Vector2(0.1f, 0.1f);
        rt.anchorMax = new Vector2(0.9f, 0.9f);
    }
}
```

```
        rt.offsetMin = Vector2.zero;
        rt.offsetMax = Vector2.zero;
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        HandleClick();
    }

    public void OnPointerUp(PointerEventData eventData)
    {
    }

    private void HandleClick()
    {
        if (Time.time - lastClickTime < CLICK_DELAY)
        {
            return;
        }

        lastClickTime = Time.time;

        if (!isClickable)
        {
            Debug.Log($"{gameObject.name} is not clickable");
            return;
        }

        if (IsMatched)
        {
            Debug.Log($"{gameObject.name} is already matched");
            return;
        }

        if (gameController == null)
        {
            Debug.LogError($"GameController is null for {gameObject.name}");
            return;
        }

        if (!gameController.CanSelect)
        {
            Debug.Log($"Cannot select {gameObject.name} - CanSelect is false");
            return;
        }

        Debug.Log($"Click handled for {gameObject.name}");

        transform.DOScale(originalScale * 1.1f, 0.2f);

        if (borderImage != null)
            borderImage.DOColor(selectedColor, 0.2f);

        gameController.SelectDoll(this);
    }

    public void SetDollSprite(Sprite sprite)
    {
        if (dollImage == null)
        {
            CreateDollImage();
        }

        if (dollImage != null)
        {
            dollImage.sprite = sprite;
            dollImage.preserveAspect = true;
            dollImage.color = Color.white;
            dollImage.enabled = true; // Убедимся, что изображение включено
            Debug.Log($"Sprite set for {gameObject.name}");
        }
        else
        {
            Debug.LogError($"Cannot set sprite for {gameObject.name}: dollImage is null");
        }
    }

    public void SetDollTypeID(int id)
    {
        dollTypeID = id;
    }
}
```

```
}

public void MarkAsMatched()
{
    Debug.Log($"Marking {gameObject.name} as matched");

    IsMatched = true;
    isClickable = false;

    if (matchSequence != null && matchSequence.IsActive())
    {
        matchSequence.Kill();
    }

    matchSequence = DOTween.Sequence();

    if (dollImage != null)
    {
        matchSequence.Append(dollImage.transform.DOScale(1.3f, 0.3f))
            .Append(dollImage.transform.DOScale(0f, 0.4f))
            .OnComplete(() => {
                if (dollImage != null)
                {
                    dollImage.enabled = false;
                }
            });
    }
    else
    {
        Debug.LogWarning($"dollImage is null for {gameObject.name} in MarkAsMatched");
    }

    if (borderImage != null)
        matchSequence.Join(borderImage.DOCOLOR(new Color(0.2f, 0.8f, 0.2f, 0.5f), 0.5f));
}

public void ResetVisual()
{
    Debug.Log($"Resetting visual for {gameObject.name}");
    transform.DOKill();
    if (borderImage != null)
        borderImage.DOKill();
    if (dollImage != null)
        dollImage.transform.DOKill();

    transform.DOScale(originalScale, 0.2f);

    if (borderImage != null)
        borderImage.DOCOLOR(defaultColor, 0.2f);
}

public void ResetDoll()
{
    Debug.Log($"Resetting doll: {gameObject.name}");

    IsMatched = false;
    isClickable = true;
    transform.DOKill();
    if (borderImage != null)
        borderImage.DOKill();
    if (dollImage != null)
    {
        dollImage.transform.DOKill();
        dollImage.DOKill();
    }

    transform.localScale = originalScale;

    if (borderImage != null)
    {
        borderImage.color = defaultColor;
        borderImage.enabled = true;
    }
    else
    {
        Debug.LogError($"borderImage is null for {gameObject.name} in ResetDoll");
    }

    if (dollImage != null)
    {

```

```

dollImage.color = Color.white;
dollImage.enabled = true;
dollImage.transform.localScale = Vector3.one;
dollImage.gameObject.SetActive(true);
}
else
{
    Debug.LogError($"dollImage is null for {gameObject.name} in ResetDoll");
}

Debug.Log($"Doll {gameObject.name} reset complete. isClickable: {isClickable}, IsMatched: {IsMatched}");
}

void OnDestroy()
{
    if (matchSequence != null && matchSequence.IsActive())
    {
        matchSequence.Kill();
    }
    transform.DOKill();
    if (borderImage != null)
        borderImage.DOKill();
    if (dollImage != null)
        dollImage.DOKill();
}
}

```

Далее необходимо добавить скрипт DollCard на каждый объект DollSprite в иерархии, перетаскив его в окно Inspector, после чего выполнить детальную настройку параметров (см.рис.11).

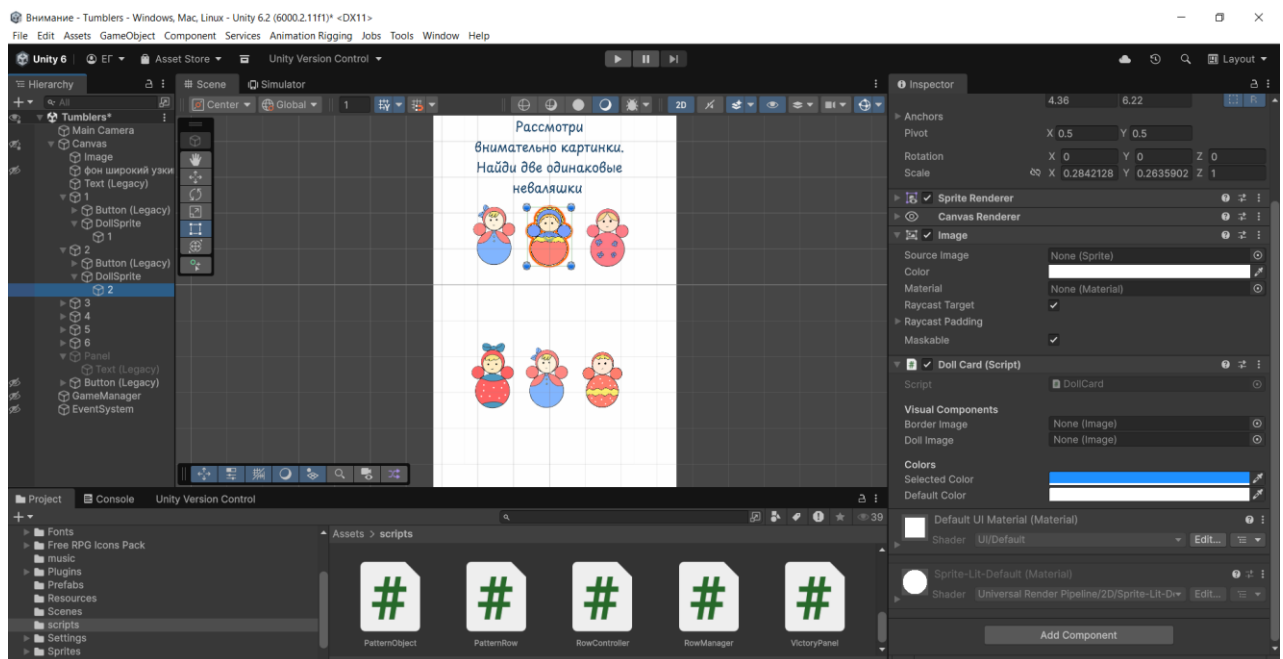


Рисунок 11 – Добавление скрипта к компоненту DollSprite и настройка компонента

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



Рисунок 12 – Готовая игра «Неваляшки»

4. Выводы

В данной работе был показан процесс создания игры «Неваляшки» 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.