

Создание мобильной игры «Найди одинаковые узоры» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель статьи – создание мобильной игры «Найди одинаковые узоры» для android. Разработка игры 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 создана мобильная игра «Найди одинаковые узоры».

Ключевые слова: Unity, игра, узоры, пара.

Creating a mobile game "Find Similar Patterns" using Unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

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

Keywords: Unity, game, patterns, pair.

1 Введение

1.1 Актуальность

Мобильные игры являются ключевым элементом современной цифровой культуры и одной из самых быстроразвивающихся и прибыльных областей в сфере развлечений. Среди них особой популярностью пользуются казуальные игры, чья доступная механика, минималистичный дизайн и увлекательный процесс привлекают широкий круг игроков.

Создание игры «Найди одинаковые узоры», в основе которой лежит механика поиска парных изображений, отвечает современным рыночным запросам. Этот формат служит не только удобным развлечением на короткий промежуток времени, но и приносит практическую пользу, развивая у игрока внимательность, зрительную память, быстроту реакции и умение концентрироваться. Благодаря этому проект обладает двойной ценностью —

развлекательной и развивающей, выступая в роли эффективного тренажера для пользователей любого возраста.

Для реализации данной игры оптимальным выбором является платформа Unity. Движок предлагает robustный инструментарий для кроссплатформенной разработки, что позволяет воплотить яркий визуальный стиль, создать детализированные узоры и понятный интерфейс. Использование Unity позволяет разработчикам сфокусироваться на отладке геймплея, качестве графики и удобстве управления, одновременно обеспечивая высокую производительность, простые решения для учёта прогресса и лёгкую публикацию в магазинах приложений. Его встроенные средства для 2D-графики значительно упрощают задачу по созданию разнообразных и привлекательных наборов паттернов.

В итоге, разработка игры «Найди одинаковые узоры» на Unity — это своевременный проект, который сочетает в себе востребованный игровой жанр (поиск совпадений), геймплей с познавательной пользой и передовые инструменты создания. Проект имеет чёткую практическую направленность, соответствует спросу на лёгкие и занимательные игры и обладает большим потенциалом для развития через введение новых уровней, тематических коллекций узоров, бонусных элементов и дополнительных игровых механик.

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 мобильную игру «Найди одинаковые узоры» под операционную систему Android, используя возможности среды разработки Unity для создания интерактивных 2D/3D-приложений.

2 Материалы и методы

Для создания игры «Найди одинаковые узоры» выбор движка Unity был основан на его основных преимуществах: доступностью, интуитивно понятным интерфейсом и поддержкой языка C#, что идеально подходит для реализации игровой логики. Разработка кода проекта велась в среде Visual Studio Community.

3 Результаты

После инициализации проекта с использованием шаблона Universal 2D требуется выполнить его базовую конфигурацию для мобильных устройств. Для этого в меню Build Profiles необходимо изменить целевую платформу на Android (см. рис. 1).

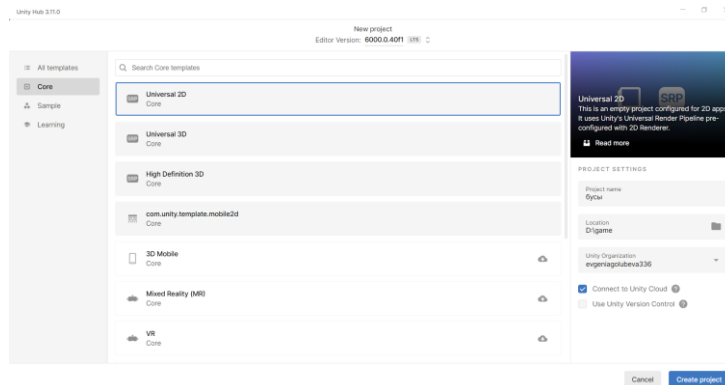


Рисунок 1 – Создание проекта

Для подготовки к запуску на мобильных устройствах следующий шаг — перейти в настройки сборки через меню File → Build Settings и выбрать Android в качестве целевой платформы для проекта (см. рис. 2).

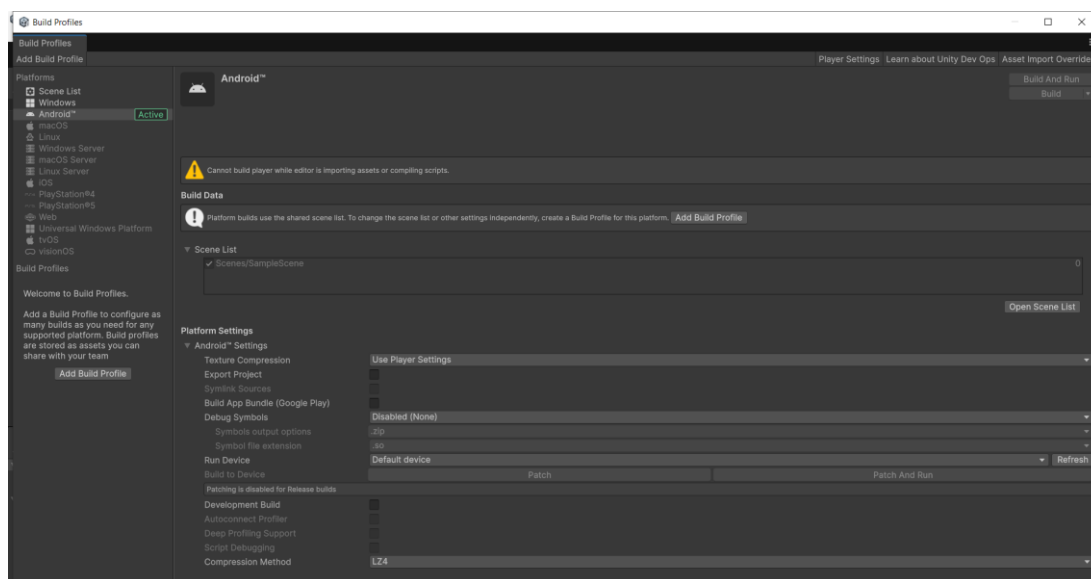


Рисунок 2 – Настройка проекта

Следующим этапом после базовой настройки является построение сцены. Начинать её реализацию необходимо с камеры, так как от её параметров зависит то, что в итоге увидит пользователь (см. рис. 3).

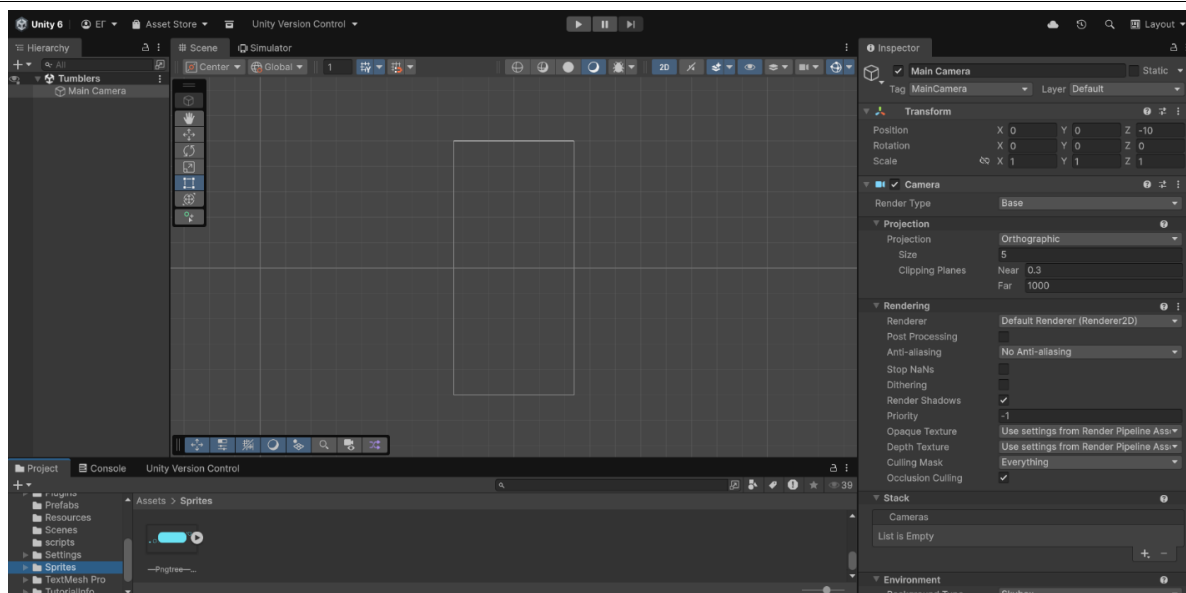


Рисунок 3 – Настройка камеры

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

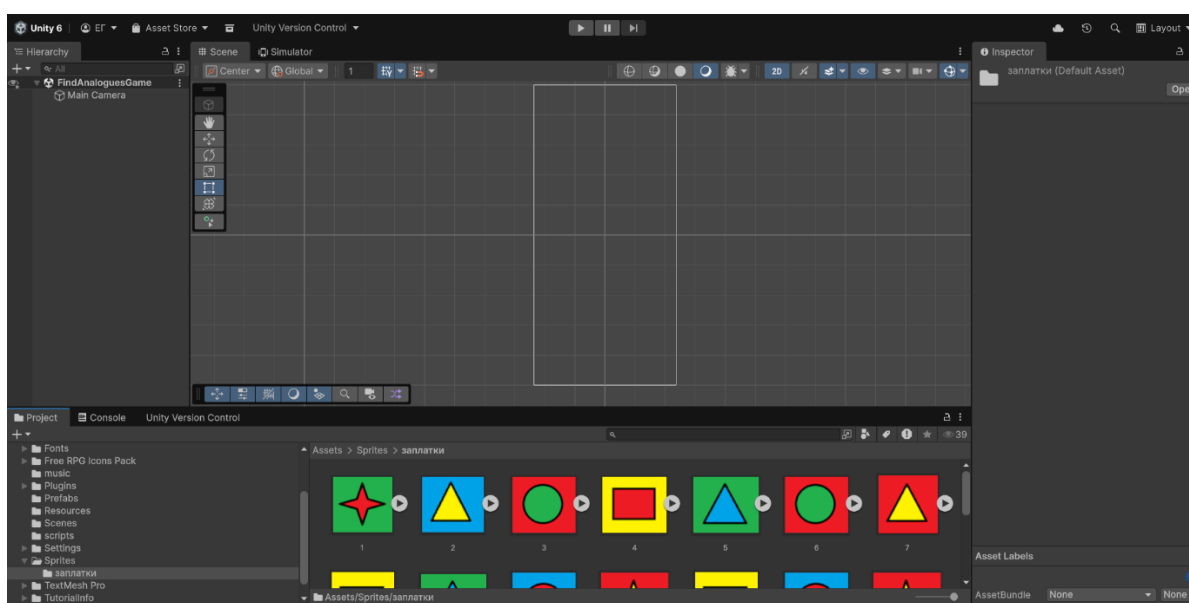


Рисунок 4 – Добавление изображений

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).

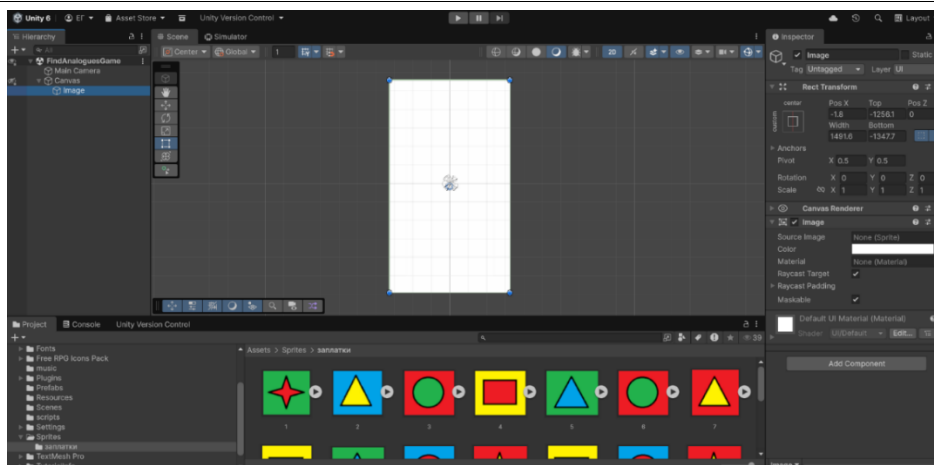


Рисунок 5 – Настройка фона игры

После создания фона сцены необходимо разместить изображения узоров в виде упорядоченной сетки. Для этого следует организовать четыре горизонтальных ряда, каждый из которых будет служить контейнером для шести игровых карточек. Создание начинается с добавления в иерархию четырёх пустых объектов, которые последовательно переименовываются в 1г, 2г, 3г и 4г. Эти объекты-ряды являются основой для компоновки, и их необходимо правильно настроить, чтобы дочерние элементы автоматически выравнивались.

Для каждого ряда требуется задать базовые параметры трансформации и добавить компонент горизонтальной компоновки. В компоненте Rect Transform часто устанавливаются якоря на растяжение, чтобы ряд занимал отведённую ширину, а позиция по Y задаётся индивидуально для вертикального расположения. Ключевым является компонент Horizontal Layout Group, где настраиваются отступы по краям, расстояние между элементами ряда, а также выравнивание — например, по верхнему краю или центру. Важно отключить опции принудительного расширения дочерних объектов, чтобы карточки сохраняли свой фиксированный или предпочтительный размер, и включить контроль над их шириной и высотой для единообразия.

Внутри каждого ряда-контейнера создаются шесть пустых дочерних объектов, которые переименовываются в PatchCard1, PatchCard2 и так далее до PatchCard6. Каждый такой объект представляет собой интерактивную карточку. Для управления её размером в сетке может быть добавлен компонент Layout Element, где указываются предпочтительные ширина и высота, после чего на сам объект добавляется компонент Button.

Затем, для отделения логики взаимодействия от графики, внутрь каждого объекта PatchCard создаётся ещё один пустой объект, именуемый PatchSprite. Этот объект будет единственным носителем визуального представления карточки. На него добавляется компонент Image, куда перетягивается спрайт узора, а в настройках важно отключить свойство Raycast Target, чтобы все клики обрабатывались родительской кнопкой, а не изображением. Финальным шагом является связывание визуального элемента

с логикой кнопки: в компоненте Button объекта PatchCard в поле Target Graphic назначается созданный объект PatchSprite, что обеспечивает корректную визуальную реакцию на взаимодействие. (см.рис.6).

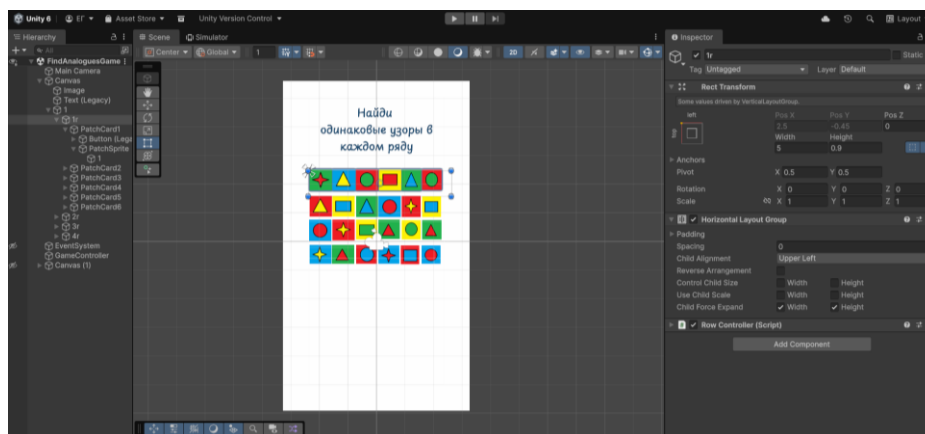


Рисунок 6 – Добавление объектов для узоров и настройка

Для отображения системных сообщений в сцене необходимо создать два базовых элемента интерфейса. Первым элементом будет панель (UI → Panel), выполняющая роль фоновой подложки, а вторым — текстовый блок (UI → Text), непосредственно предназначенный для вывода сообщения.

Настройку начинают с панели. В окне Inspector для её компонента Rect Transform устанавливают центральные якоря (Anchors) со значениями (0.5, 0.5), что позволяет позиционировать элемент относительно центра родительского контейнера. Затем задают точный размер панели, например, 1420 на 601.8 пикселей. В компоненте Image выбирают подходящий спрайт-фон, настраивают его цвет и уровень прозрачности (Alpha), а для корректного отображения устанавливают Image Type в значение Simple. (см.рис.7).

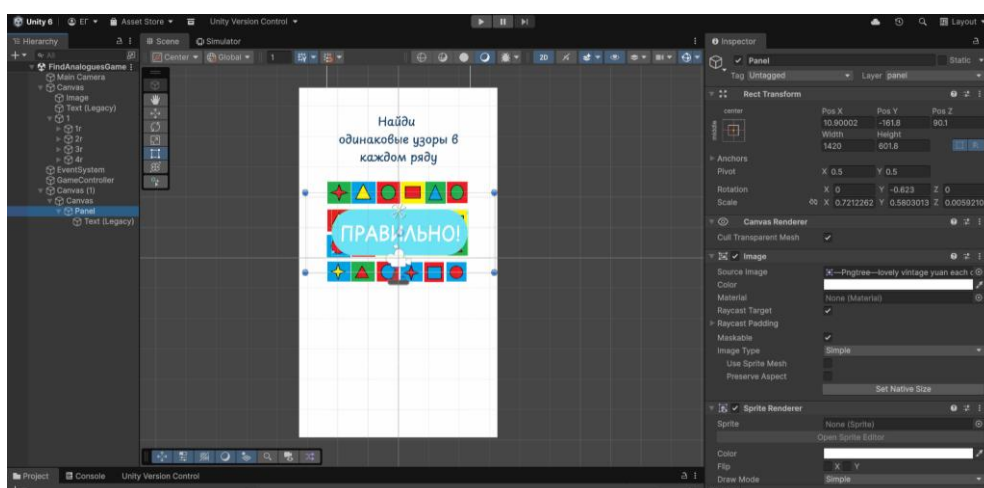


Рисунок 7 – Настройка панели

Для вывода уведомления «Правильно!» выполняется детальная настройка текстового элемента. В компоненте Text (Legacy) выбирается шрифт PlaypenSans-VariableFont_wght с начертанием Bold и устанавливается

размер 100 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).

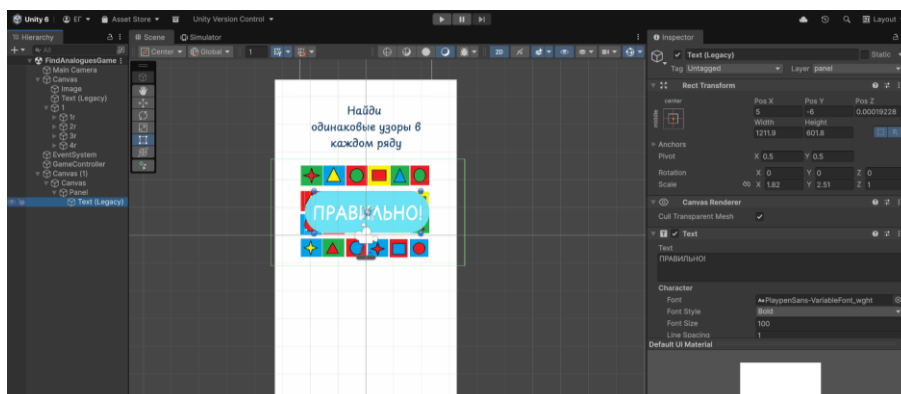


Рисунок 8 – Настройка текстового элемента панели сообщений

Для настройки текстового блока выбирается шрифт Pangolin-Regular с применённым жирным начертанием Bold и размером 88 пунктов, что обеспечивает максимальную чёткость и выразительность надписи. Параметры текста устанавливаются с выравниванием по центру по горизонтали и вертикали, а также активируется опция Rich Text, позволяющая при необходимости использовать базовое форматирование внутри строки. В компоненте Rect Transform задаётся геометрия элемента: ширина 909.1 и высота 440.1 пикселей. Для точного позиционирования относительно родительской панели устанавливаются центральные якоря (Anchors) со значениями (0.5, 0.5). (см.рис.9).

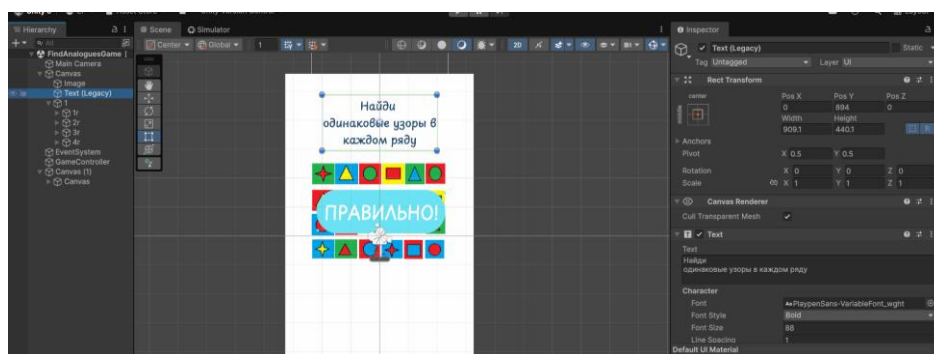


Рисунок 9 – Настройка инструкции игры

После создание сцены, необходимо приступить к написанию скриптов. Скрипт PatchGameController служит главным управляющим звеном мини-игры по сборке карточек. Он отвечает за создание и настройку четырёх игровых рядов, каждый со своим уникальным целевым изображением и правильной последовательностью слотов. Скрипт обрабатывает выбор игрока, проверяет корректность собранных комбинаций и фиксирует успешно завершённые ряды. Как только все ряды собраны верно, контроллер запускает

победную последовательность со звуковыми и визуальными эффектами, после чего возвращает игрока в меню.

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

public class PatchGameController : MonoBehaviour
{
    public static PatchGameController Instance;

    [Header("Rows Settings")]
    [SerializeField] private List<RowController> rows = new List<RowController>();
    [SerializeField] private int correctPatchesPerRow = 4;

    [Header("Correct Patch Positions")]
    [SerializeField] private int[] row1Correct = new int[] { 0, 1, 2, 3 };
    [SerializeField] private int[] row2Correct = new int[] { 1, 2, 3, 4 };
    [SerializeField] private int[] row3Correct = new int[] { 2, 3, 4, 5 };
    [SerializeField] private int[] row4Correct = new int[] { 0, 2, 4, 5 };

    [Header("Patch Sprites")]
    [SerializeField] private Sprite[] correctSprites;
    [SerializeField] private Sprite[] uniqueSprites;

    [Header("Game Settings")]
    [SerializeField] private AudioClip patchSelectSound;
    [SerializeField] private AudioClip correctMatchSound;
    [SerializeField] private AudioClip wrongMatchSound;
    [SerializeField] private AudioClip victorySound;
    [SerializeField] private bool enableFlashFeedback = true;

    [Header("UI Settings")]
    [SerializeField] private GameObject victoryPanel;
    [SerializeField] private Text victoryText;
    [SerializeField] private string menuSceneName = "Menu";
    [SerializeField] private float delayBeforeMenuLoad = 3f;

    private List<PatchCard> selectedPatches = new List<PatchCard>();
    private AudioSource audioSource;
    private int completedRowCount = 0;
    private const int REQUIRED_ROWS_TO_WIN = 4;
    private Dictionary<int, Sprite> rowCorrectSprites = new Dictionary<int, Sprite>();

    public bool CanSelect { get; private set; } = true;
    private bool isInitialized = 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()
    {
        InitializeGame();
    }

    void InitializeGame()
    {

```

```

        if (rows.Count == 0)
        {
            #if UNITY_2021_3_OR_NEWER
                RowController[] foundRows =
            FindObjectsByType<RowController>(FindObjectsInactive.Exclude, FindObjectsSortMode.None);
            #else
                RowController[] foundRows = FindObjectsOfType<RowController>();
            #endif

            if (foundRows.Length > 0)
            {
                rows.AddRange(foundRows);
                rows.Sort((a, b) => a.name.CompareTo(b.name));
            }

            if (rows.Count < 4)
            {
                Debug.LogError("Need 4 rows! Currently have: " + rows.Count);
                return;
            }

            if (correctSprites == null || correctSprites.Length < 4)
            {
                correctSprites = CreateDefaultSprites(4, "Correct");
            }

            if (uniqueSprites == null || uniqueSprites.Length < 2)
            {
                uniqueSprites = CreateDefaultSprites(2, "Unique");
            }

            InitializeRows();

            isInitialized = true;
            Debug.Log("Game Initialization Complete");
        }

        void InitializeRows()
        {
            List<int[]> rowConfigs = new List<int[]> { row1Correct, row2Correct, row3Correct,
            row4Correct };

            for (int i = 0; i < rows.Count && i < 4; i++)
            {
                if (rows[i] == null)
                {
                    Debug.LogError("Row " + i + " is null!");
                    continue;
                }

                Sprite correctSprite = correctSprites[i % correctSprites.Length];
                rowCorrectSprites[i] = correctSprite;

                int[] correctIndices = rowConfigs[i];

                rows[i].InitializeRow(i, correctSprite, correctIndices, uniqueSprites);
            }
        }

        Sprite[] CreateDefaultSprites(int count, string prefix)
        {
            Sprite[] sprites = new Sprite[count];
            Color[] colors = { Color.red, Color.blue, Color.green, Color.yellow, Color.magenta,
            Color.cyan };

            for (int i = 0; i < count; i++)
            {
                sprites[i] = CreateSimpleSprite(colors[i % colors.Length], prefix + "_" + i);
            }

            return sprites;
        }

        Sprite CreateSimpleSprite(Color color, string name)
        {
            Texture2D texture = new Texture2D(64, 64);

            for (int x = 0; x < 64; x++)
            {

```

```

        for (int y = 0; y < 64; y++)
        {
            if (x < 2 || x > 61 || y < 2 || y > 61)
            {
                texture.SetPixel(x, y, Color.black);
            }
            else
            {
                texture.SetPixel(x, y, color);
            }
        }
    }

    texture.Apply();
    Sprite sprite = Sprite.Create(texture, new Rect(0, 0, 64, 64), new Vector2(0.5f, 0.5f));
    sprite.name = name;
    return sprite;
}

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

    Keyboard keyboard = Keyboard.current;

    if (keyboard != null)
    {
        if (keyboard.rKey.wasPressedThisFrame)
            ResetGame();

        if (keyboard.dKey.wasPressedThisFrame)
            DebugGameState();

        if (keyboard.pKey.wasPressedThisFrame)
            TestVictory();

        if (keyboard.fKey.wasPressedThisFrame)
            TestFlash();
    }
}

public void SelectPatch(PatchCard patch)
{
    if (patch == null || !CanSelect || patch.IsMatched || selectedPatches.Contains(patch))
        return;

    int patchesInRow = CountSelectedPatchesInRow(patch.RowIndex);
    if (patchesInRow >= correctPatchesPerRow)
    {
        Debug.Log("Cannot select more than " + correctPatchesPerRow + " patches in row " +
patch.RowIndex);
        return;
    }

    PlaySound(patchSelectSound);
    selectedPatches.Add(patch);
    patch.SetSelected(true);

    if (enableFlashFeedback && rowCorrectSprites.TryGetValue(patch.RowIndex, out Sprite
correctSprite))
    {
        if (patch.CurrentSprite == correctSprite)
        {
            patch.FlashCorrect();
        }
        else
        {
            patch.FlashWrong();
        }
    }

    if (patchesInRow + 1 == correctPatchesPerRow)
    {
        StartCoroutine(CheckRowCompletion(patch.RowIndex));
    }
}

private int CountSelectedPatchesInRow(int rowIndex)
{
    int count = 0;

```

```

        foreach (PatchCard patch in selectedPatches)
        {
            if (patch.RowIndex == rowIndex)
                count++;
        }
        return count;
    }

    private List<PatchCard> GetSelectedPatchesInRow(int rowIndex)
    {
        List<PatchCard> patchesInRow = new List<PatchCard>();
        foreach (PatchCard patch in selectedPatches)
        {
            if (patch.RowIndex == rowIndex)
                patchesInRow.Add(patch);
        }
        return patchesInRow;
    }

    private IEnumerator CheckRowCompletion(int rowIndex)
    {
        CanSelect = false;

        List<PatchCard> rowPatches = GetSelectedPatchesInRow(rowIndex);

        if (rowPatches.Count != correctPatchesPerRow)
        {
            Debug.LogError("Expected " + correctPatchesPerRow + " patches in row " + rowIndex + ", got " + rowPatches.Count);
            CanSelect = true;
            yield break;
        }

        bool allCorrect = true;
        Sprite correctSprite = rowCorrectSprites[rowIndex];

        foreach (PatchCard patch in rowPatches)
        {
            if (patch.CurrentSprite != correctSprite)
            {
                allCorrect = false;
                break;
            }
        }

        if (allCorrect)
        {
            PlaySound(correctMatchSound);

            yield return new WaitForSeconds(0.3f);

            foreach (PatchCard patch in rowPatches)
            {
                patch.MarkAsMatched();
                selectedPatches.Remove(patch);
            }

            completedRowsCount++;
            CheckGameCompletion();
        }
        else
        {
            PlaySound(wrongMatchSound);

            yield return new WaitForSeconds(0.5f);

            foreach (PatchCard patch in rowPatches)
            {
                patch.SetSelected(false);
                selectedPatches.Remove(patch);
            }
        }

        CanSelect = true;
    }

    private void CheckGameCompletion()
    {
        if (completedRowsCount >= REQUIRED_ROWS_TO_WIN)
        {

```

```
        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! Нашёл все одинаковые узоры!";
    }

    yield return new WaitForSeconds(delayBeforeMenuLoad);

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

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

public void ResetGame()
{
    selectedPatches.Clear();
    completedRowCount = 0;
    CanSelect = true;

    foreach (RowController row in rows)
    {
        if (row != null)
            row.ResetRow();
    }

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

public void GoToMenu()
{
    if (!string.IsNullOrEmpty(menuSceneName))
    {
        SceneManager.LoadScene(menuSceneName);
    }
}

private void DebugGameState()
{
    Debug.Log("=== GAME STATE ===");
    Debug.Log("Rows: " + rows.Count);
    Debug.Log("Selected patches: " + selectedPatches.Count);
    Debug.Log("CanSelect: " + CanSelect);
    Debug.Log("Completed rows: " + completedRowCount + "/" + REQUIRED_ROWS_TO_WIN);
}

private void TestVictory()
{
    completedRowCount = REQUIRED_ROWS_TO_WIN;
    CheckGameCompletion();
}

private void TestFlash()
{
    if (rows.Count > 0 && rows[0] != null)
    {
        PatchCard[] patches = rows[0].GetComponentsInChildren<PatchCard>();
        foreach (PatchCard patch in patches)
        {
            if (patch != null)

```

```

        patch.FlashCorrect();
    }
}
}
}

```

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 GameController. Скрипт PatchGameController 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).

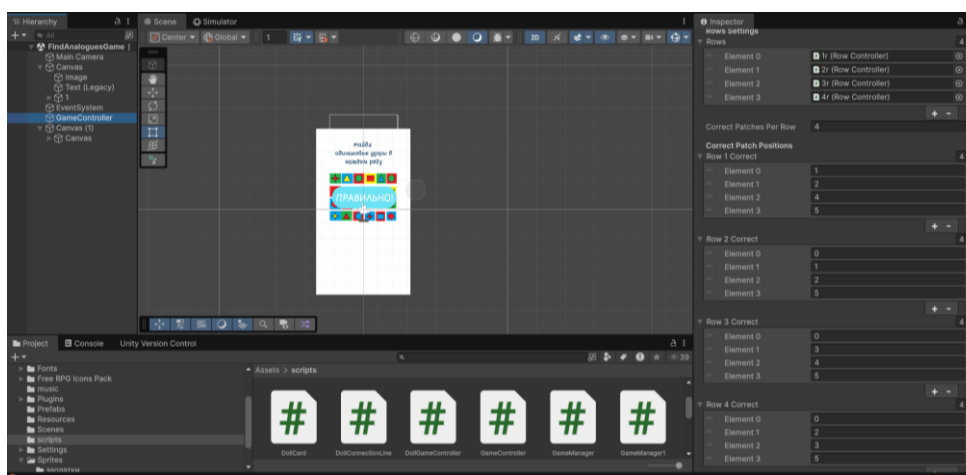


Рисунок 10 – Добавление скрипта к компоненту GameController и настройка компонента

Скрипт RowController управляет отдельным рядом карточек в игре на сопоставление: автоматически находит и сортирует все карточки в ряду, инициализирует их правильными и уникальными спрайтами согласно заданной конфигурации, обеспечивает сброс состояния ряда и предоставляет информацию о его статусе для отладки, выступая связующим звеном между главным игровым контроллером и отдельными карточками.

```

using UnityEngine;
using System.Collections.Generic;

public class RowController : MonoBehaviour
{
    [Header("Components")]
    [SerializeField] private List<PatchCard> patches = new List<PatchCard>();

    private Sprite correctSprite;
    private int[] correctIndices;

    void Start()
    {
        FindPatches();
    }

    private void FindPatches()
    {
        if (patches.Count == 0)
        {

```

```

        PatchCard[] foundPatches = GetComponentsInChildren<PatchCard>(true);
        patches.AddRange(foundPatches);

        patches.Sort((a, b)
a.transform.localPosition.x.CompareTo(b.transform.localPosition.x));
=>

        Debug.Log($"Found {patches.Count} patches in {name}");
    }
}

public void InitializeRow(int rowIndex, Sprite correctSprite, int[] correctIndices,
Sprite[] uniqueSprites)
{
    this.correctSprite = correctSprite;
    this.correctIndices = correctIndices;

    FindPatches();

    if (patches.Count < 6)
    {
        Debug.LogError($"Patch {name}: Need 6 patches, found {patches.Count}");
        return;
    }

    Debug.Log($"Initializing row {rowIndex} with {patches.Count} patches");

    for (int i = 0; i < patches.Count; i++)
    {
        if (patches[i] == null)
        {
            Debug.LogError($"Patch {i} in row {rowIndex} is null!");
            continue;
        }

        bool isCorrect = false;
        foreach (int correctIndex in correctIndices)
        {
            if (correctIndex == i)
            {
                isCorrect = true;
                break;
            }
        }

        Sprite spriteToUse;
        if (isCorrect)
        {
            spriteToUse = correctSprite;
        }
        else
        {
            int uniqueIndex = i % uniqueSprites.Length;
            spriteToUse = uniqueSprites[uniqueIndex];
        }

        patches[i].Initialize(rowIndex, spriteToUse);
    }
}

public void ResetRow()
{
    foreach (PatchCard patch in patches)
    {
        if (patch != null)
            patch.ResetCard();
    }
}

public string GetRowStatus()
{
    int matchedCount = 0;

```

```

int totalCount = patches.Count;

foreach (PatchCard patch in patches)
{
    if (patch != null && patch.IsMatched)
        matchedCount++;
}

return $"{name}: {matchedCount}/{totalCount} matched";
}

[ContextMenu("Debug Row")]
public void DebugRow()
{
    Debug.Log($"=== {name} ===");
    Debug.Log($"Correct sprite: {(correctSprite != null ? correctSprite.name : "null")}");
    Debug.Log($"Patches count: {patches.Count}");

    if (correctIndices != null)
        Debug.Log($"Correct indices: [{string.Join(", ", correctIndices)}]");

    for (int i = 0; i < patches.Count; i++)
    {
        if (patches[i] != null)
        {
            Debug.Log($"Patch {i}: {patches[i].name}, Matched: {patches[i].IsMatched}");
        }
    }
}
}

```

Далее необходимо добавить скрипт RowController на каждый объект 1г, 2г, 3г, и 4г в иерархии, перетаскив его в окно Inspector, после чего выполнить детальную настройку параметров (см.рис.11).

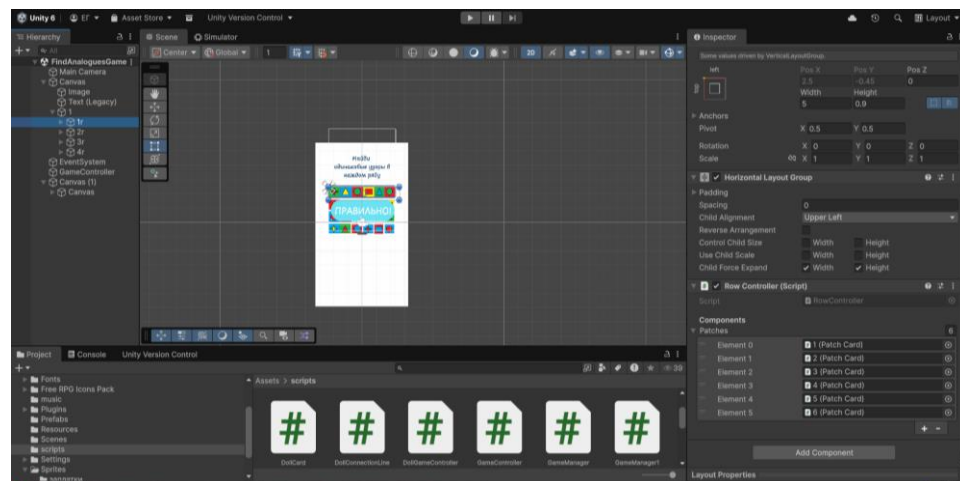


Рисунок 11 – Добавление скрипта к компоненту RowController и настройка компонента

```

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

[RequireComponent(typeof(Image))]
[RequireComponent(typeof(Button))]
public class PatchCard : MonoBehaviour, IPointerClickHandler

```

```

{
    [Header("Components")]
    [SerializeField] private Image borderImage;
    [SerializeField] private Image patchImage;

    [Header("Colors")]
    [SerializeField] private Color normalColor = Color.white;
    [SerializeField] private Color selectedColor = Color.yellow;
    [SerializeField] private Color matchedColor = Color.green;
    [SerializeField] private Color correctFlashColor = new Color(0.2f, 1f, 0.2f, 1f);
    [SerializeField] private Color wrongFlashColor = Color.red;

    [Header("Animation Settings")]
    [SerializeField] private float flashDuration = 0.2f;
    [SerializeField] private int flashCount = 2;
    [SerializeField] private float scaleFactor = 1.1f;

    public int RowIndex { get; private set; }
    public bool IsMatched { get; private set; }
    public Sprite CurrentSprite { get; private set; }

    private Button button;
    private Vector3 originalScale;
    private Sequence flashSequence;
    private Color originalBorderColor;

    void Awake()
    {
        EnsureComponents();
        originalScale = transform.localScale;

        if (borderImage != null)
            originalBorderColor = borderImage.color;
    }

    void Start()
    {
        if (button != null)
        {
            button.onClick.RemoveAllListeners();
            button.onClick.AddListener(OnClick);
        }
    }

    private void EnsureComponents()
    {
        if (borderImage == null)
            borderImage = GetComponent<Image>();

        if (borderImage != null)
        {
            borderImage.raycastTarget = true;
            borderImage.color = normalColor;
        }

        if (button == null)
            button = GetComponent<Button>();

        if (button != null)
        {
            button.transition = Selectable.Transition.None;
            Navigation nav = new Navigation();
            nav.mode = Navigation.Mode.None;
            button.navigation = nav;
        }

        if (patchImage == null)
            FindOrCreatePatchImage();
    }

    private void FindOrCreatePatchImage()

```

```
{
    Image[] childImages = GetComponentsInChildren<Image>();
    foreach (Image img in childImages)
    {
        if (img.gameObject != gameObject)
        {
            patchImage = img;
            break;
        }
    }

    if (patchImage == null)
    {
        GameObject patchObj = new GameObject("PatchSprite");
        patchObj.transform.SetParent(transform);
        patchObj.transform.localPosition = Vector3.zero;
        patchObj.transform.localScale = Vector3.one;

        patchImage = patchObj.AddComponent<Image>();
        patchImage.preserveAspect = true;
        patchImage.raycastTarget = false;

        RectTransform rt = patchObj.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;
    }

    patchImage.raycastTarget = false;
}

public void OnPointerClick(PointerEventData eventData)
{
    OnClick();
}

private void OnClick()
{
    if (IsMatched || !gameObject.activeInHierarchy)
        return;

    if (PatchGameController.Instance == null)
    {
        Debug.LogError("PatchGameController instance not found!");
        return;
    }

    PatchGameController.Instance.SelectPatch(this);
}

public void Initialize(int rowIndex, Sprite sprite)
{
    RowIndex = rowIndex;
    CurrentSprite = sprite;

    if (patchImage != null && sprite != null)
    {
        patchImage.sprite = sprite;
        patchImage.color = Color.white;
    }

    ResetCard();
}

public void SetSelected(bool selected)
{
    if (borderImage != null)
    {
        if (selected)
        {

```

```
        borderImage.color = selectedColor;
        transform.DOScale(originalScale * scaleFactor, 0.2f);
    }
    else
    {
        borderImage.color = normalColor;
        transform.DOScale(originalScale, 0.2f);
    }
}

public void FlashCorrect()
{
    StopFlashAnimation();

    flashSequence = DOTween.Sequence();

    for (int i = 0; i < flashCount; i++)
    {
        flashSequence.Append(borderImage.DOColor(correctFlashColor,
flashDuration));
        flashSequence.Append(borderImage.DOColor(selectedColor, flashDuration));
    }

    flashSequence.Append(borderImage.DOColor(selectedColor, 0.1f));
    flashSequence.Play();
}

public void FlashWrong()
{
    StopFlashAnimation();

    flashSequence = DOTween.Sequence();

    for (int i = 0; i < flashCount; i++)
    {
        flashSequence.Append(borderImage.DOColor(wrongFlashColor, flashDuration));
        flashSequence.Append(borderImage.DOColor(selectedColor, flashDuration));
    }

    flashSequence.Append(borderImage.DOColor(selectedColor, 0.1f));
    flashSequence.Play();
}

public void MarkAsMatched()
{
    IsMatched = true;

    if (button != null)
        button.interactable = false;

    StopFlashAnimation();

    Sequence matchSequence = DOTween.Sequence();
    matchSequence.Append(borderImage.DOColor(correctFlashColor, 0.3f));
    matchSequence.Append(borderImage.DOColor(matchedColor, 0.5f));

    if (patchImage != null)
    {
        matchSequence.Join(patchImage.transform.DOScale(1.2f, 0.3f));
        matchSequence.Append(patchImage.transform.DOScale(1f, 0.2f));
    }

    matchSequence.Play();
}

public void ResetCard()
{
    IsMatched = false;

    if (button != null)
```

```

        button.interactable = true;

        StopFlashAnimation();

        if (borderImage != null)
            borderImage.color = normalColor;

        transform.localScale = originalScale;

        if (patchImage != null)
            patchImage.transform.localScale = Vector3.one;
    }

    private void StopFlashAnimation()
    {
        if (flashSequence != null && flashSequence.IsActive())
        {
            flashSequence.Kill();
        }
    }

    void OnDestroy()
    {
        StopFlashAnimation();
    }
}

```

Далее необходимо добавить скрипт PathCard на каждый объект изображения узора в иерархии, перетащив его в окно Inspector, после чего выполнить детальную настройку параметров (см.рис.12).

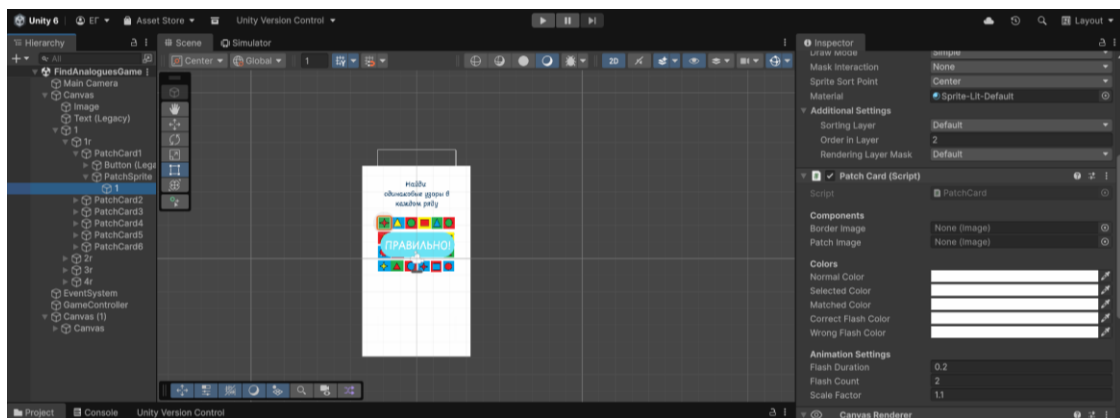


Рисунок 12 – Добавление скрипта PathCard к объектам и настройка

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



Рисунок 13 – Готовая игра «Найди одинаковые узоры»

4. Выводы

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