

Создание мобильной игры «Пазл» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель данной статьи – создание мобильной игры «Пазл» 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 «Пазл»

Ключевые слова: Unity, игра, пазл, картинка.

Creating a Puzzle Mobile Game in Unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create a mobile game "Puzzle" for android. To create the game, we used a 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 of the work, a mobile game "Puzzle" was created.

Keywords: Unity, game, puzzle, picture.

1 Введение

1.1 Актуальность

Мобильные игры стали неотъемлемой частью жизни, превратившись из простого развлечения в многомиллиардную индустрию. Среди множества жанров особой популярностью пользуются пазлы — они совмещают в себе интеллектуальный вызов, релаксацию и приносят удовлетворение от решения задачи. Эта классическая механика привлекает самую широкую аудиторию, от детей, развивающих логику, до взрослых, которые снимают стресс и тренируют внимание.

Unity, как один из ведущих игровых движков, идеально подходит для таких проектов. Его мощные инструменты для работы с 2D-графикой, встроенные компоненты для перетаскивания и кроссплатформенность позволяют быстро создавать прототипы и выпускать готовые игры на iOS и Android, что является ключом к коммерческому успеху.

Разработка мобильного пазла на Unity — это актуальная и практичная задача. Такой проект отвечает спросу на игры с позитивным психологическим

эффектом, развивающие логику и пространственное мышление. Это отличная возможность освоить геймдев на практике и создать востребованный продукт для массовой аудитории, ценящей качественный досуг.

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 «Пазл» для ОС Android при помощи кроссплатформенной среды для создания интерактивных 2D и 3D приложений Unity.

2 Материалы и методы

Разработка игры «Пазл» была реализована на игровом движке Unity, а для написания и редактирования программного кода применялась интегрированная среда разработки Visual Studio Community.

3 Результаты

Чтобы приступить к созданию мобильной игры «Пазл», требуется инициализировать новый проект в Unity. Для этого в интерфейсе Unity Hub нужно выбрать шаблон **Universal 2D** для двумерной графики, присвоить проекту название и подтвердить его создание. (см. рис. 1).

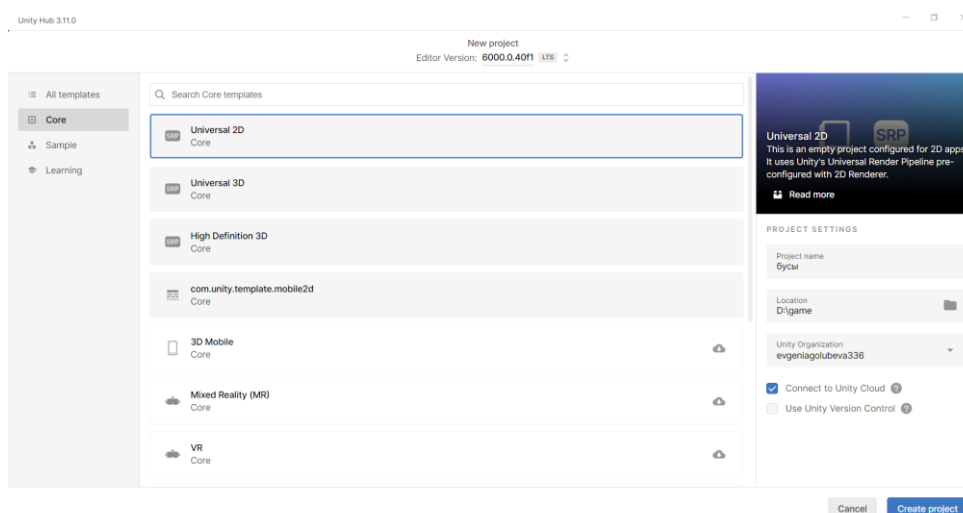


Рисунок 1- Создание проекта

После инициализации проекта с использованием шаблона Universal 2D требуется выполнить его базовую конфигурацию для мобильной платформы. Ключевым шагом является смена целевой платформы на Android через меню File -> Build Profiles (см. рис. 2).

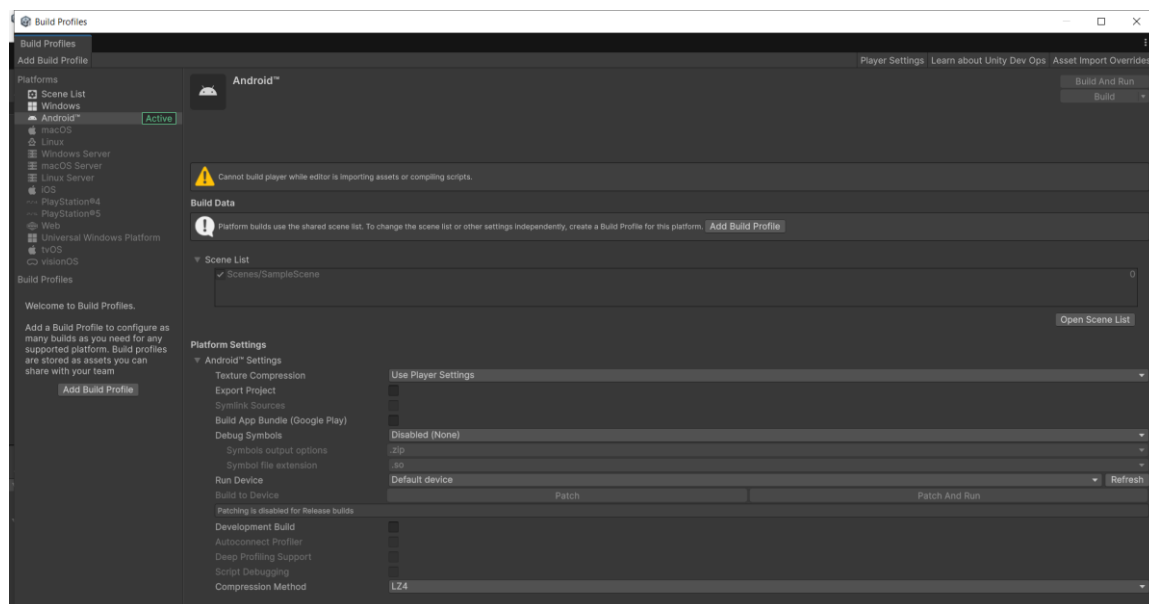


Рисунок 2 - Настройка проекта

Следующим этапом является создание игровой сцены. Настройка камеры выступает в качестве первоочередной задачи, поскольку именно она определяет область видимости и визуальное восприятие игрового пространства (см. рис. 3).

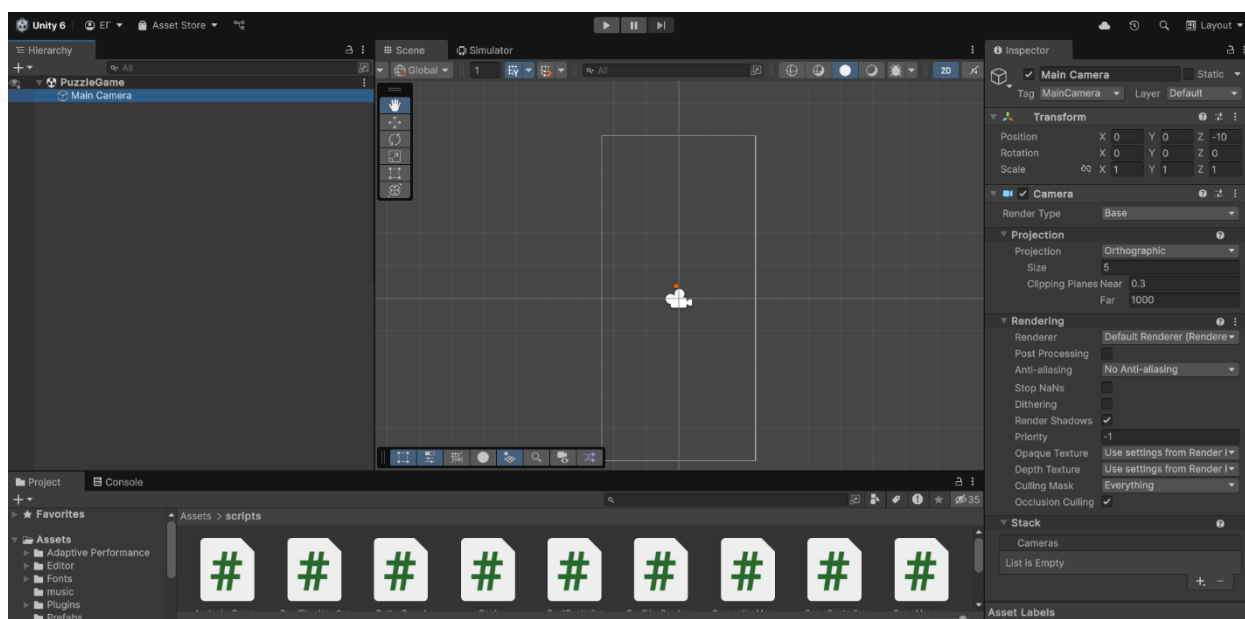


Рисунок 3 - Настройка камеры

Для дальнейшей работы следует поместить заранее подготовленные спрайты в соответствующую папку Sprites (см.рис.4).

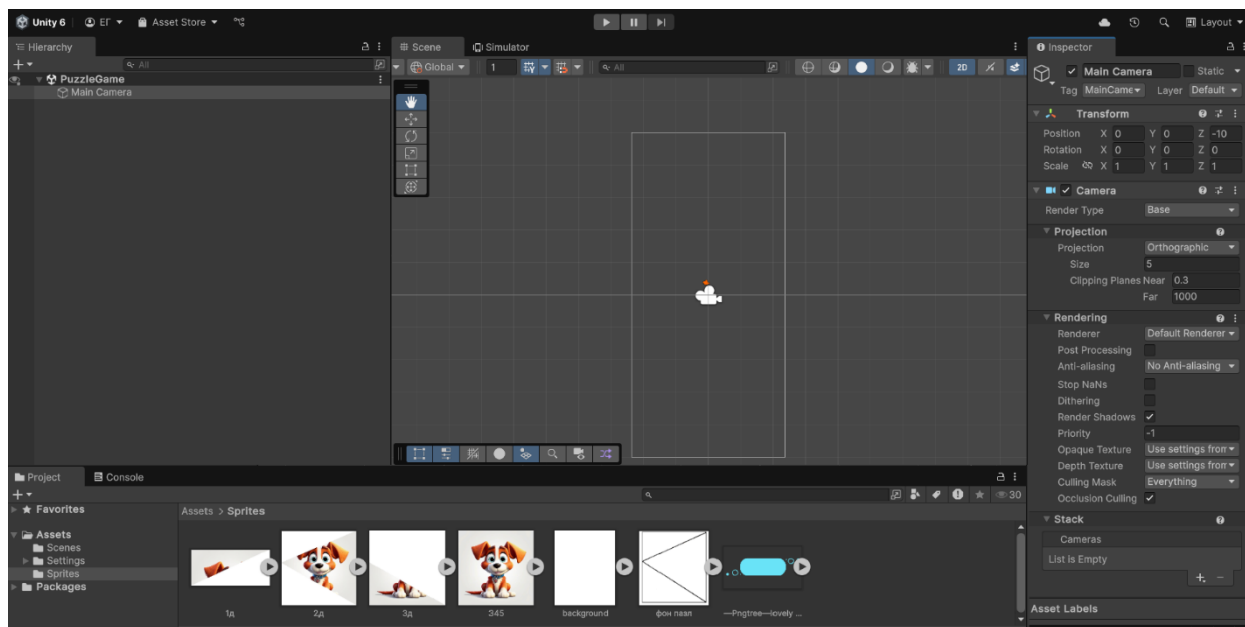


Рисунок 4 - Добавление изображений

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

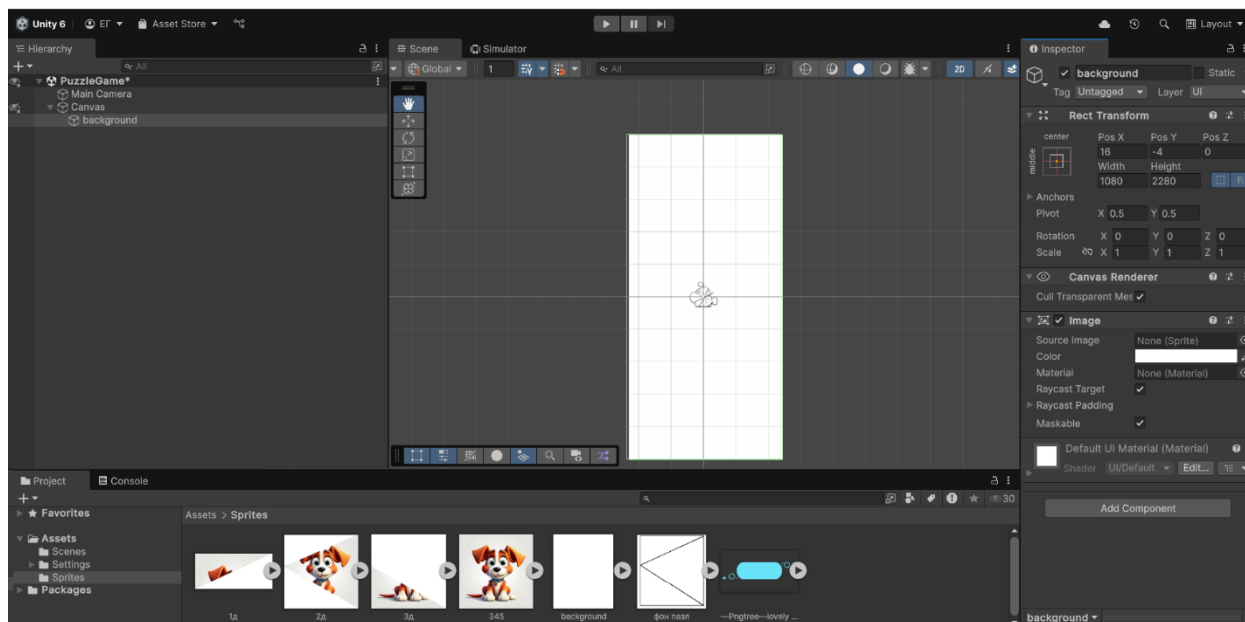


Рисунок 5 – Настройка фона игры

Следующим этапом разработки является добавление изображений пазла и рамки на игровую сцену, для чего необходимо перенести подготовленные спрайты из окна Project в область Hierarchy. После добавления элементов требуется выполнить их детальную настройку: для рамки пазла устанавливается позиция с координатами X: -0.775, Y: 501.6, Z: -2052, где

значительное отрицательное значение по оси Z обеспечивает корректное отображение в камере, и масштаб X: 224.02, Y: 176.52, Z: 228 (см. рис. 6).

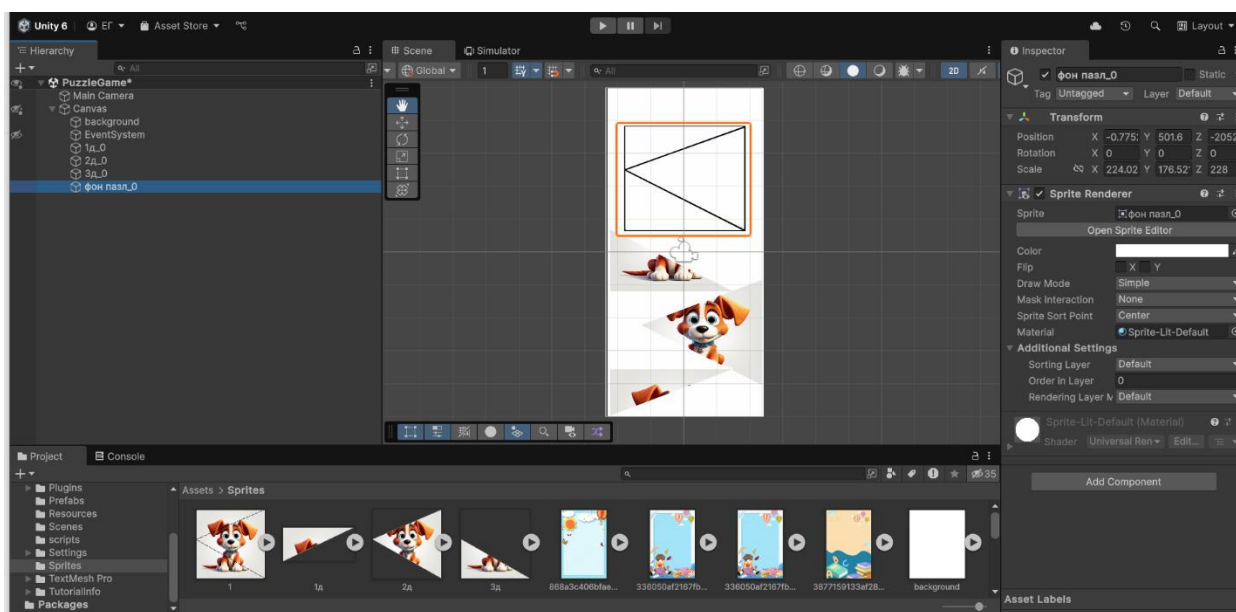


Рисунок 6 – Настройка рамки для пазлов

Для элементов пазла настраиваются индивидуальные координаты в Transform, добавляются необходимые компоненты - Sprite Renderer для отображения, и Circle Collider 2D для обработки взаимодействий (см.рис.7).

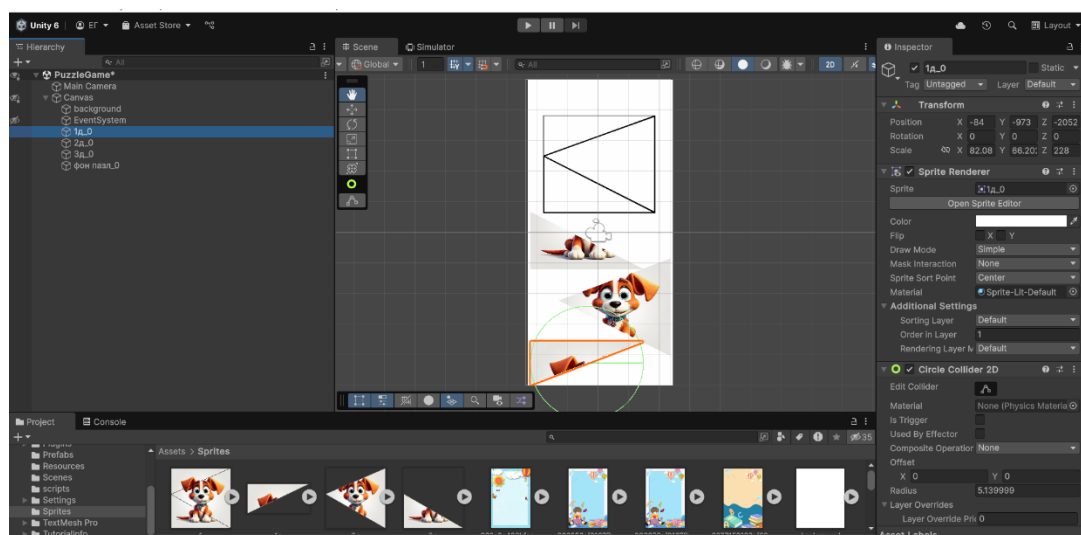


Рисунок 7 – Добавление изображений пазлов на сцену и их настройка

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

После добавления панели и текстового элемента в интерфейс игры необходимо провести их настройку через компонент Inspector. Устанавливаем якоря Anchors в центральное положение со значениями (0.5, 0.5), что

обеспечивает правильное позиционирование и масштабирование элемента относительно различных разрешений экрана. Затем задаются конкретные размеры области Width 1080 и Height 232,7 пикселей. Для визуального оформления в компоненте Image необходимо выбрать подходящий спрайт фонового изображения в поле Source Image, настраивается цвет и прозрачность через параметр Color в соответствии с общим визуальным стилем игры, а тип отображения устанавливается в значение Simple (см.рис.8).

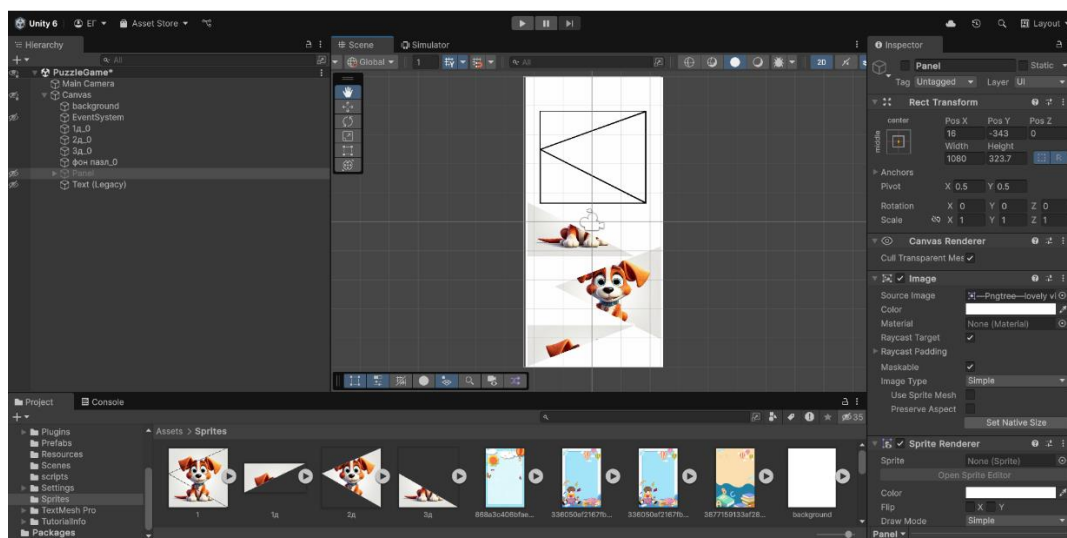


Рисунок 8 – Настройка панели

После настройки панели необходимо отредактировать текстовый блок с уведомлением "Правильно!". В компоненте Text (Legacy) необходимо выбрать шрифт PlaypenSans-VariableFont_wght с жирным начертанием Bold и установить размер 150 пунктов для обеспечения оптимальной читаемости. В разделе Rect Transform установить размеры текстового поля: Width 1073,1 и Height 247,9. Также необходимо настроить якоря Anchors с центральной точкой 0,5, 0,5. Далее необходимо настроить текст, для этого необходимо выбрать выравнивание текста по центру, цветовое оформление и параметры переноса текста для достижения гармоничного визуального представления в интерфейсе игры. (см.рис. 9).

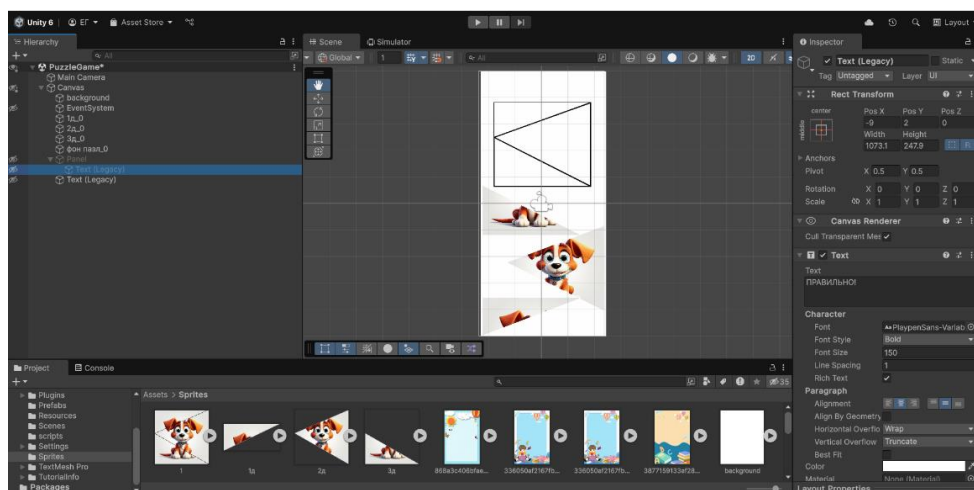


Рисунок 9 – Настройка текстового элемента панели сообщений

Для текстового элемента настраиваем шрифт Pangolin-Regular с жирным начертанием и размером 140 пунктов с жирным начертанием Bold, выравнивание по центру, включаем поддержку Rich Text для форматирования, а в Rect Transform задаем точные размеры текстового поля: Width 1064, Height 344,7, и устанавливаем якоря Anchors с центром в точке (0.5, 0.5) (см.рис.10).

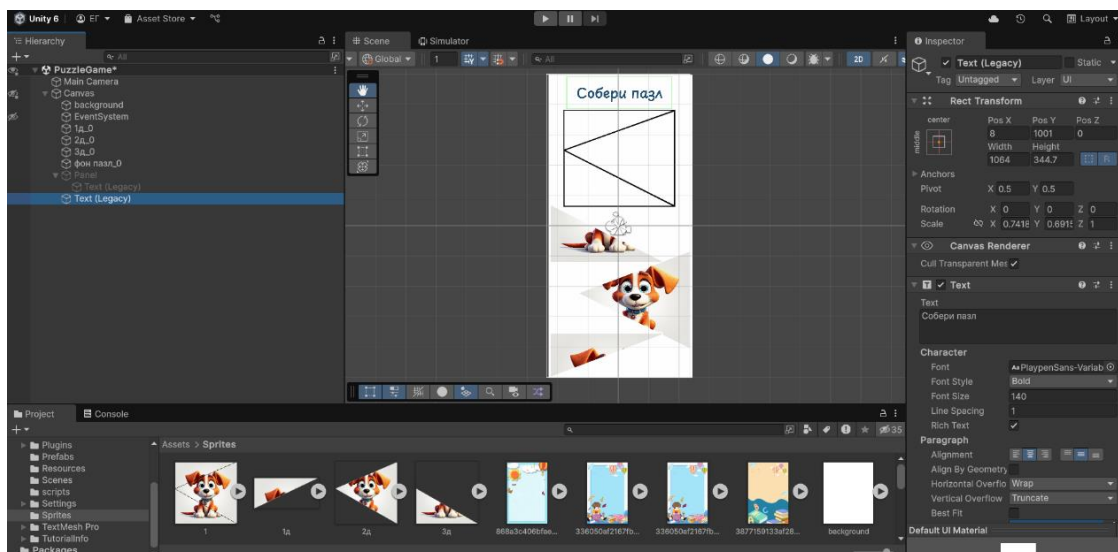


Рисунок 10 – Настройка инструкции игры

Также необходимо добавить изображение готовой картинке пазла, для того чтобы при сборке пазла появилась готовая картинка. Для этого подготовленную картинку добавляем в Hierarchy, и в inspector ее настраиваем (см. рис. 11).

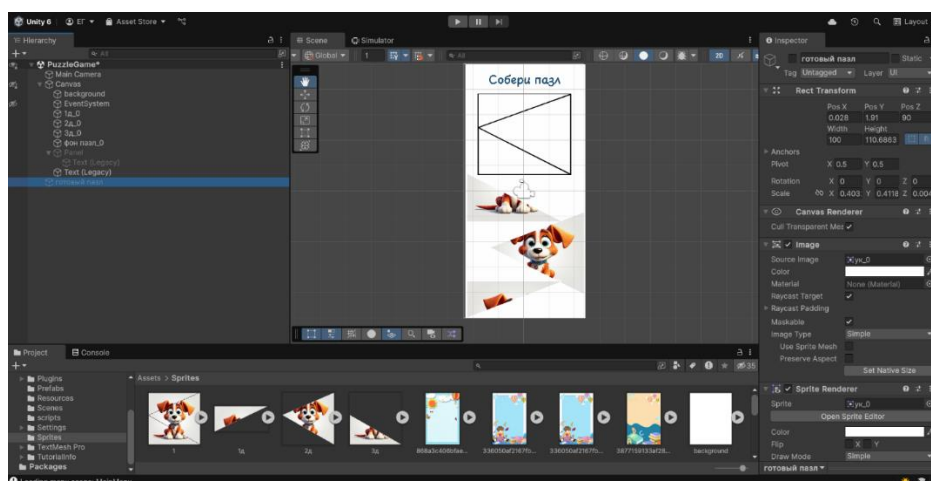


Рисунок 11 – Добавление готовой картинки пазла

После добавление всех изображений на сцену, необходимо добавить область на рамку пазла, для того чтобы пазлы могли зафиксироваться в своей ячейки. Для этого необходим в Hierarchy создать 3 пустых игровых объектов через правый клик и выбираем Create Empty. Эти объекты необходимо переименовать в Зона_1, Зона_2, Зона_3. В Inspector необходимо выполнить следующие настройки для каждого объекта: установить компоненту Transform

точные координаты позиции в соответствии с местоположением соответствующих ячеек на рамке пазла, добавить компонент Box Collider 2D и активировать опцию Is Trigger для обеспечения обработки столкновений в режиме триггера, настроить размер коллайдера в соответствии с габаритами ячеек, задать соответствующий слой и тег для организации взаимодействия с другими объектами сцены (см. рис. 12).

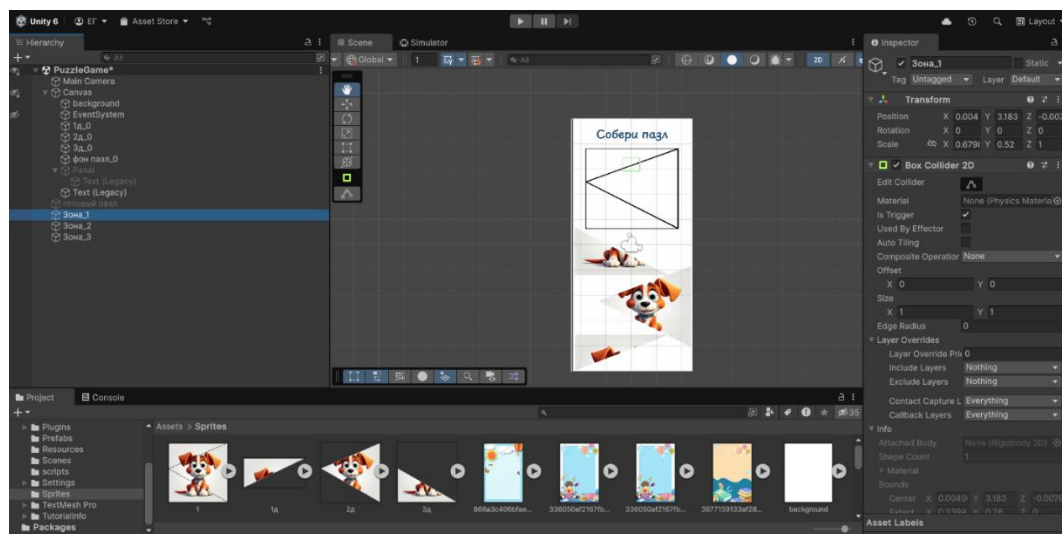


Рисунок 12 – Добавление зон для пазлов и их настройка

Далее необходимо создать GameController для управления игрой. Для этого необходим в Hierarchy создать пустой игровой объект через правый клик и выбираем Create Empty, и переименовываем в GameController.

Для компонента GameController необходимо создать скрипт PuzzleGame, который реализует основную логику управления игровым процессом пазла: он содержит систему сериализуемых классов PuzzlePiece для связи игровых объектов пазлов с их целевыми зонами, обрабатывает ввод данных с мыши и тачскрина для перемещения элементов, проверяет корректность установки пазлов в зоны фиксации с помощью расчета дистанции между объектами, управляет визуальными и звуковыми эффектами (включая воспроизведение звуков подбора, установки и завершения пазла), отслеживает общее состояние сборки головоломки через проверку всех флагов isInPlace, реализует завершение игры с отображением сообщения о победе и итогового изображения.

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

public class PuzzleGame : MonoBehaviour
{
    [System.Serializable]
    public class PuzzlePiece
    {
        public GameObject pieceObject;    // Объект пазла
```

```
        public GameObject targetArea;          // Целевая зона
        public float snapDistance = 1.0f; // Дистанция для притягивания
        [HideInInspector] public Vector2 startPosition; // Начальная позиция
        [HideInInspector] public bool isInPlace;          // Флаг правильного
положения
    }

    public List<PuzzlePiece> puzzlePieces = new List<PuzzlePiece>();
    public GameObject winMessage;
    public Image completeImage; // Изображение собранного пазла
    public float showCompleteDuration = 3f; // Время показа итоговой картинки
    public string menuSceneName = "Menu"; // Название сцены меню
    // Звуковые эффекты
    public AudioClip piecePickupSound; // Звук при взятии пазла
    public AudioClip pieceDropSound; // Звук при отпускании пазла
    public AudioClip pieceSnapSound; // Звук при правильной установке пазла
    public AudioClip puzzleCompleteSound; // Звук при полной сборке пазла
    private PuzzlePiece currentDraggedPiece;
    private Vector2 offset;
    private Camera mainCamera;
    private bool isCompleted = false;
    private AudioSource audioSource;

    void Start()
    {
        mainCamera = Camera.main;

        // Получаем или добавляем AudioSource
        audioSource = GetComponent<AudioSource>();
        if (audioSource == null)
        {
            audioSource = gameObject.AddComponent<AudioSource>();
        }
        InitializePieces();
        winMessage.SetActive(false);
        if (completeImage != null)
        {
            completeImage.gameObject.SetActive(false);
        }
    }
    void InitializePieces()
    {
        foreach (var piece in puzzlePieces)
        {
            piece.startPosition = piece.pieceObject.transform.position;
            piece.isInPlace = false;
            AddColliderIfMissing(piece.pieceObject);
        }
    }
    void Update()
    {
        if (isCompleted) return;

        HandleInput();
        UpdatePiecePosition();
    }
    void HandleInput()
    {
#if UNITY_EDITOR || UNITY_STANDALONE
        if (Input.GetMouseButtonDown(0))
        {
            TryStartDrag(Input.mousePosition);
        }
        else if (Input.GetMouseButtonUp(0))

```

```

        {
            EndDrag();
        }
    }
    #elif UNITY_ANDROID || UNITY_IOS
        if (Input.touchCount > 0)
        {
            Touch touch = Input.GetTouch(0);
            if (touch.phase == TouchPhase.Began)
            {
                TryStartDrag(touch.position);
            }
            else if (touch.phase == TouchPhase.Ended || touch.phase ==
TouchPhase.Canceled)
            {
                EndDrag();
            }
        }
    }
    #endif
    }
    void UpdatePiecePosition()
    {
        if (currentDraggedPiece != null)
        {
            Vector2 inputPos = GetInputPosition();
            Vector2 worldPos = mainCamera.ScreenToWorldPoint(inputPos);
            currentDraggedPiece.pieceObject.transform.position = worldPos +
offset;
        }
    }
    void TryStartDrag(Vector2 inputPosition)
    {
        Vector2 worldPos = mainCamera.ScreenToWorldPoint(inputPosition);
        RaycastHit2D hit = Physics2D.Raycast(worldPos, Vector2.zero);

        if (hit.collider != null)
        {
            foreach (var piece in puzzlePieces)
            {
                if (hit.collider.gameObject == piece.pieceObject &&
!piece.isInPlace)
                {
                    currentDraggedPiece = piece;
                    offset = (Vector2)piece.pieceObject.transform.position -
worldPos;

                    // Воспроизводим звук взятия пазла
                    PlaySound(piecePickupSound);
                    break;
                }
            }
        }
    }
    void EndDrag()
    {
        if (currentDraggedPiece == null) return;
        float distance = Vector2.Distance(
            currentDraggedPiece.pieceObject.transform.position,
            currentDraggedPiece.targetArea.transform.position
        );
        if (distance <= currentDraggedPiece.snapDistance)
        {
            SnapToTarget();
            CheckWinCondition();
        }
        else
    }

```

```

        {
            ResetPosition();
            // Воспроизводим звук отпускания пазла (не в целевой зоне)
            PlaySound(pieceDropSound);
        }
        currentDraggedPiece = null;
    }
    void SnapToTarget()
    {
        currentDraggedPiece.pieceObject.transform.position =
currentDraggedPiece.targetArea.transform.position;
        currentDraggedPiece.isInPlace = true;
        currentDraggedPiece.pieceObject.GetComponent<BoxCollider2D>().enabled
= false;
        // Воспроизводим звук правильной установки пазла
        PlaySound(pieceSnapSound);
    }
    void ResetPosition()
    {
        currentDraggedPiece.pieceObject.transform.position =
currentDraggedPiece.startPosition;
    }
    void CheckWinCondition()
    {
        foreach (var piece in puzzlePieces)
        {
            if (!piece.isInPlace) return;
        }
        StartCoroutine(CompletePuzzle());
    }
    IEnumerator CompletePuzzle()
    {
        isCompleted = true;
        // Воспроизводим звук завершения пазла
        PlaySound(puzzleCompleteSound);
        winMessage.SetActive(true);
        foreach (var piece in puzzlePieces)
        {
            piece.pieceObject.SetActive(false);
        }
        if (completeImage != null)
        {
            completeImage.gameObject.SetActive(true);
        }
        yield return new WaitForSeconds(showCompleteDuration);

        if (!string.IsNullOrEmpty(menuSceneName))
        {
            SceneManager.LoadScene(menuSceneName);
        }
    }
    void AddColliderIfMissing(GameObject obj)
    {
        if (obj.GetComponent<BoxCollider2D>() == null)
        {
            obj.AddComponent<BoxCollider2D>();
        }
    }
    public void RestartGame()
    {
        isCompleted = false;
        foreach (var piece in puzzlePieces)
        {
            piece.pieceObject.transform.position = piece.startPosition;

```

```

        piece.isInPlace = false;
        piece.pieceObject.GetComponent<BoxCollider2D>().enabled = true;
        piece.pieceObject.SetActive(true);
    }
    winMessage.SetActive(false);
    if (completeImage != null)
    {
        completeImage.gameObject.SetActive(false);
    }
}
Vector2 GetInputPosition()
{
#if UNITY_EDITOR || UNITY_STANDALONE
    return Input.mousePosition;
#elif UNITY_ANDROID || UNITY_IOS
    return Input.GetTouch(0).position;
#endif
}
// Метод для воспроизведения звука
void PlaySound(AudioClip clip)
{
    if (clip != null && audioSource != null)
    {
        audioSource.PlayOneShot(clip);
    }
}
}

```

После создания скрипта `PuzzleGame` необходимо добавить его к компоненту `GameController`. Для этого необходимо перетащить скрипт к компоненту в `Hierarchy` и его настроить его параметры через инспектор: в разделе `Puzzle Pieces` необходимо указать количество элементов пазла (в данном случае 3), после чего для каждого элемента последовательно задать параметры - в поле `Piece Object` перетащить соответствующий игровой объект пазла (`1A_0`, `2A_0`, `3A_0`), в поле `Target Area` назначить соответствующие зоны фиксации (`Зона_1`, `Зона_2`, `Зона_3`), а в `Snap Distance` установить дистанцию срабатывания фиксации (значение 1). Далее необходимо настроить дополнительные параметры: в поле `Win Message` указать объект с сообщением о победе, в `Complete Image` - изображение собранного пазла, выставить продолжительность показа результата в `Show Complete Duration`, и добавить соответствующие звуковые файлы в поля для звуковых эффектов (см. рис. 13).

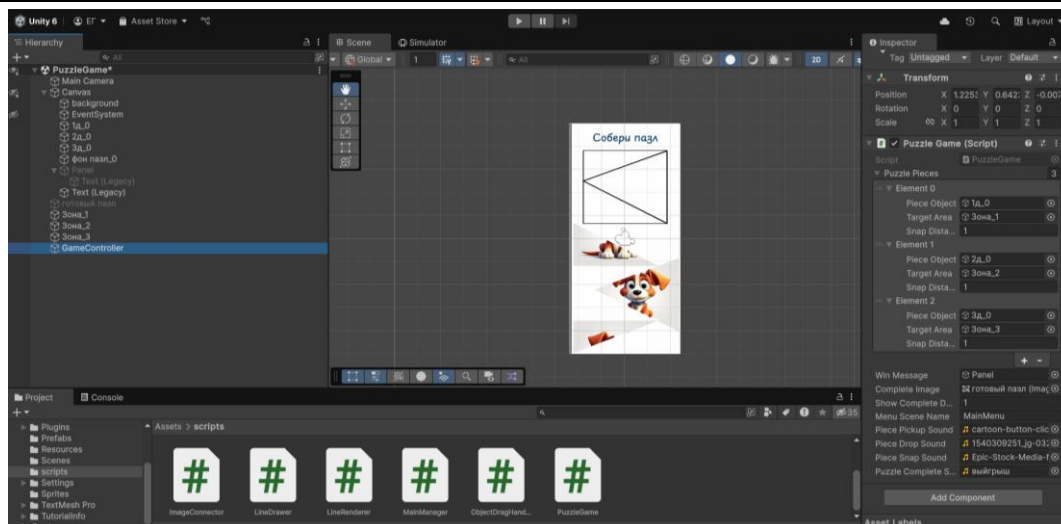


Рисунок 13 – Добавление скрипта к компоненту GameController и его настройка

Итоговый результат создания игры «Пазл» (см. рис.14).

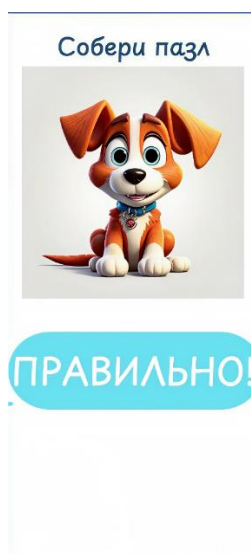


Рисунок 14 – Готовая игра «Найди пары»

4. Выводы

В данной работе был показан процесс создания игры «Пазл» с помощью кроссплатформенной среды для создания интерактивных 2D и 3D приложений. Материал статьи 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.