

Создание мобильной игры «Сортировка» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

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

Ключевые слова: Unity, игра, сортировка, картинка, цвет.

Creating a Sorting 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 "Sorting" 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 "Sorting"

Keywords: Unity, game, sorting, picture, color.

1 Введение

1.1 Актуальность

Игра «Сортировка» — это современная интерпретация классической головоломки, которая превращает простое развлечение в инструмент для тренировки логики, скорости реакции и цветовосприятия. Такой формат пользуется стабильной популярностью, так как предлагает позитивный психологический эффект: снятие стресса через концентрацию на ясных задачах и удовлетворение от моментальной визуальной обратной связи при правильной сортировке.

Разработка этой игры на движке Unity — это практичный и эффективный подход. Движок предоставляет идеальный набор инструментов для работы с 2D-графикой, создания интуитивного интерфейса с перетаскиванием предметов и точной физикой их падения, а также быстрой сборки проекта под все мобильные платформы. Это значительно ускоряет путь от прототипа, где ключевая механика проверяется за считанные часы, до готового продукта в магазинах приложений.

Таким образом, проект по созданию игры «Сортировка» на Unity решает сразу несколько задач: он позволяет на практике освоить ключевые этапы мобильной разработки, включая работу с touch-интерфейсом и игровой логикой состояний, и создает востребованный продукт, который оценит массовая аудитория, ищущая одновременно расслабляющий и динамичный 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]. Максимов Н.В. 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# Любич А.А. [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 «Сортировка» для ОС Android при помощи кроссплатформенной среды для создания интерактивных 2D и 3D приложений Unity.

2 Материалы и методы

Разработка игры «Сортировка» 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 «Сортировка», 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).

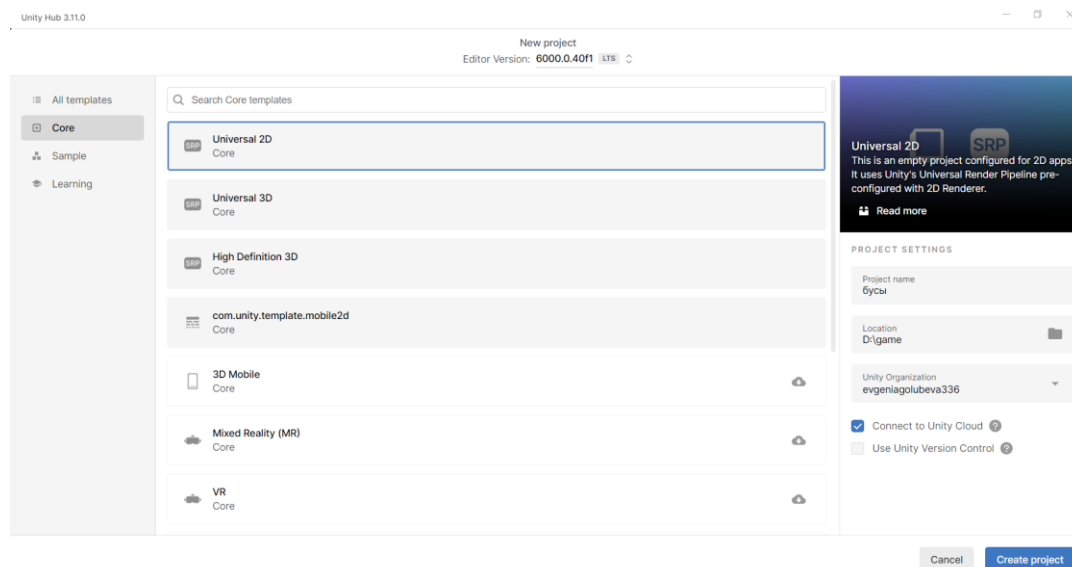


Рисунок 1 – Создание проекта

После создания проекта на основе шаблона Universal 2D необходимо выполнить его базовую настройку под мобильные устройства. Основное действие — изменить целевую платформу на Android. Это делается через меню File -> Build Settings (см. рис. 2).

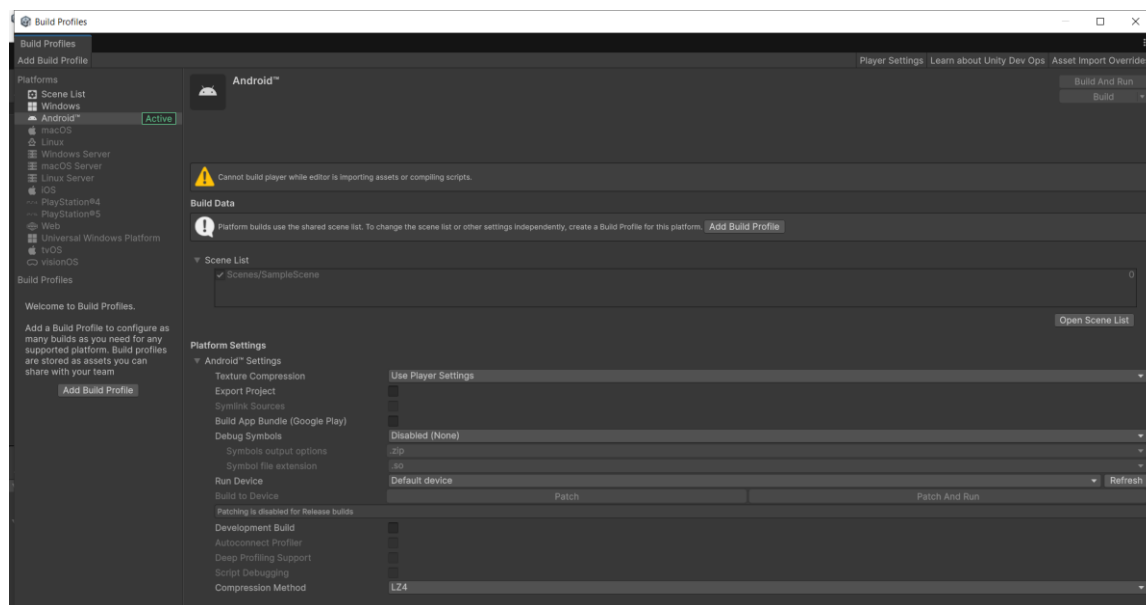


Рисунок 2 – Настройка проекта

Следующий шаг — создание игровой сцены. Первоочередное внимание следует уделить настройке камеры, так как от её параметров зависит область видимости и общее визуальное представление игрового мира (см. рис. 3).

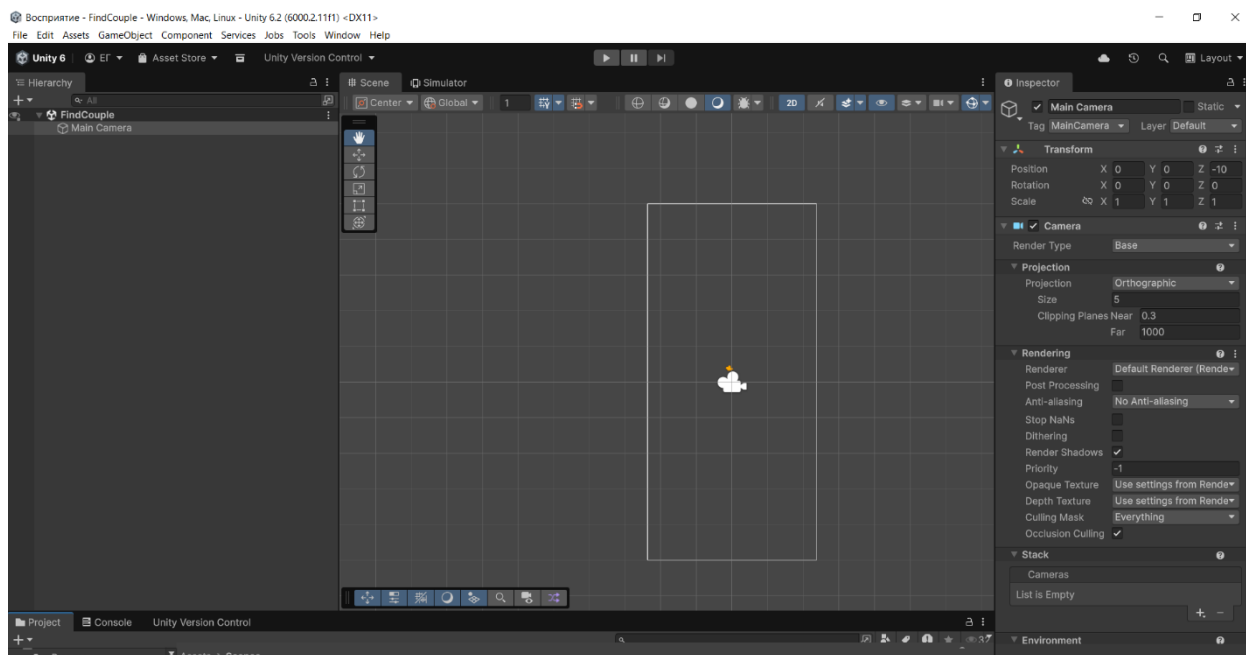


Рисунок 3 – Настройка камеры

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

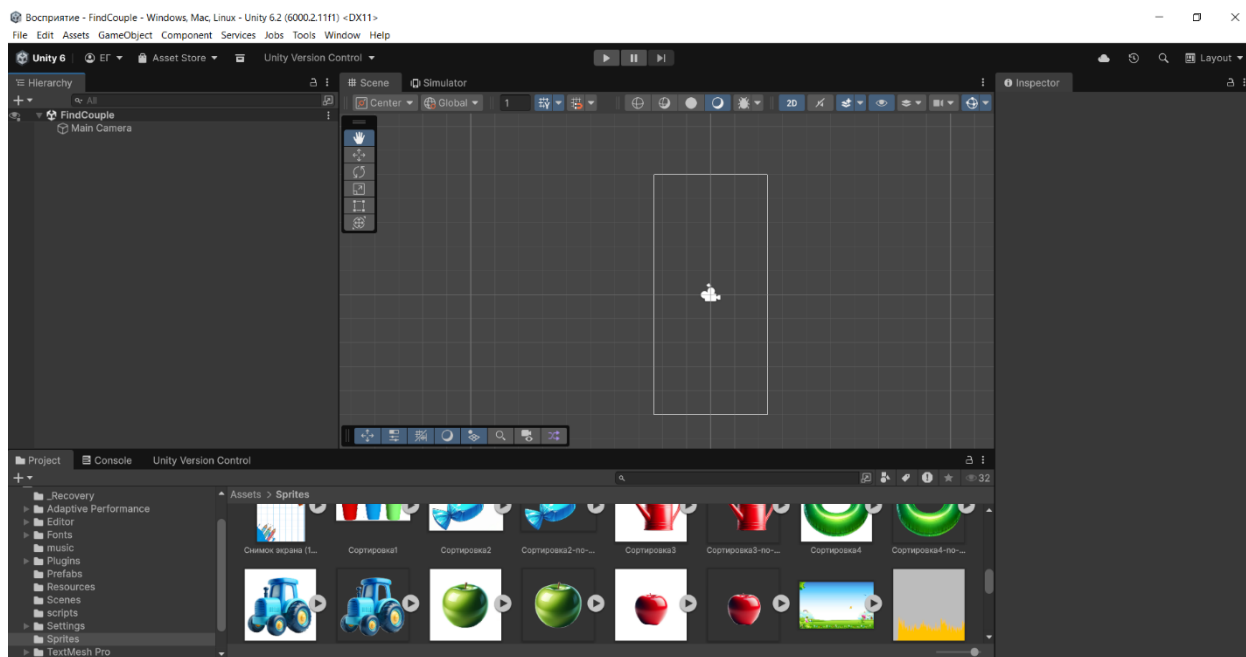


Рисунок 4 – Добавление изображений

Чтобы создать фон, необходимо переместить спрайт background из папки Sprites прямо на сцену. Затем отрегулировать его настройк, чтобы целиком закрывал всю видимую игровую область (см.рис.5).

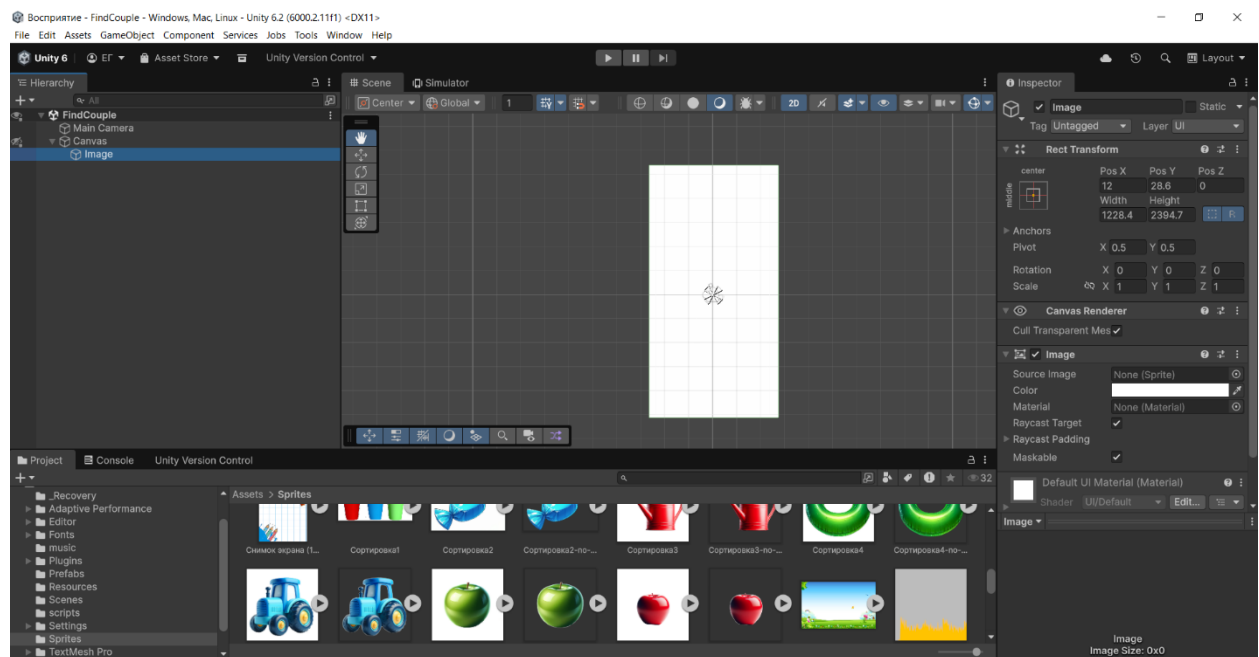


Рисунок 5 – Настройка фона игры

Следующим этапом разработки является добавление изображений стаканов и предметов, для этого необходимо перенести подготовленные спрайты из окна Project в область Hierarchy. После добавления спрайтов на сцену, необходимо их правильно расположить на сцене (см. рис. 6).

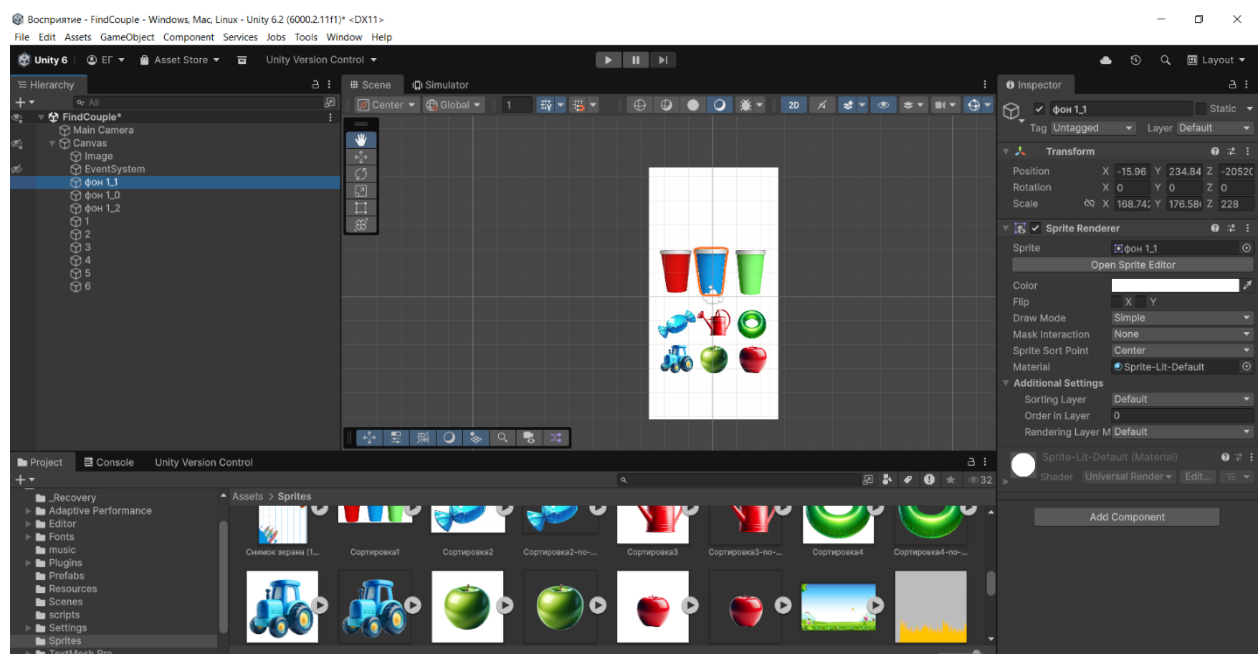


Рисунок 6 – Настройка спрайтов на сцену

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

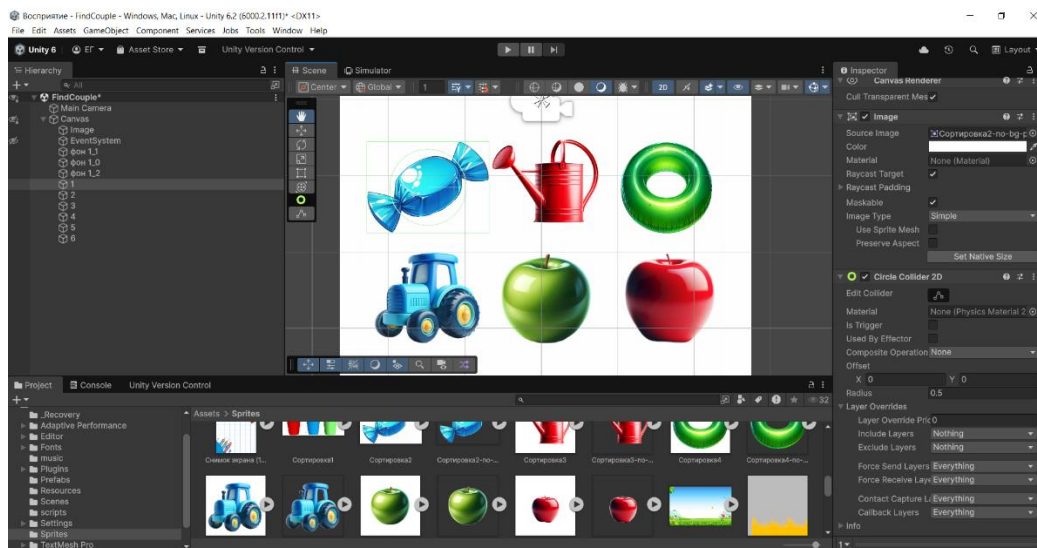


Рисунок 7 – Добавление изображений предметов на сцену и их настройка

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 1049.1 и Height 481.5 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).

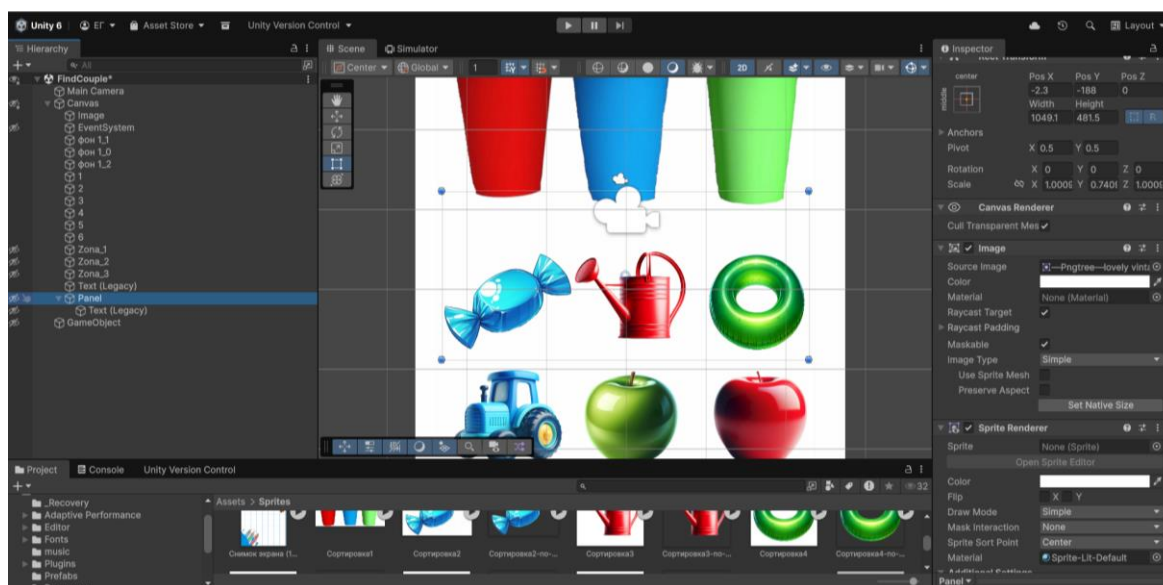


Рисунок 8 – Настройка панели

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 1041.7 и Height 347.8. 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).

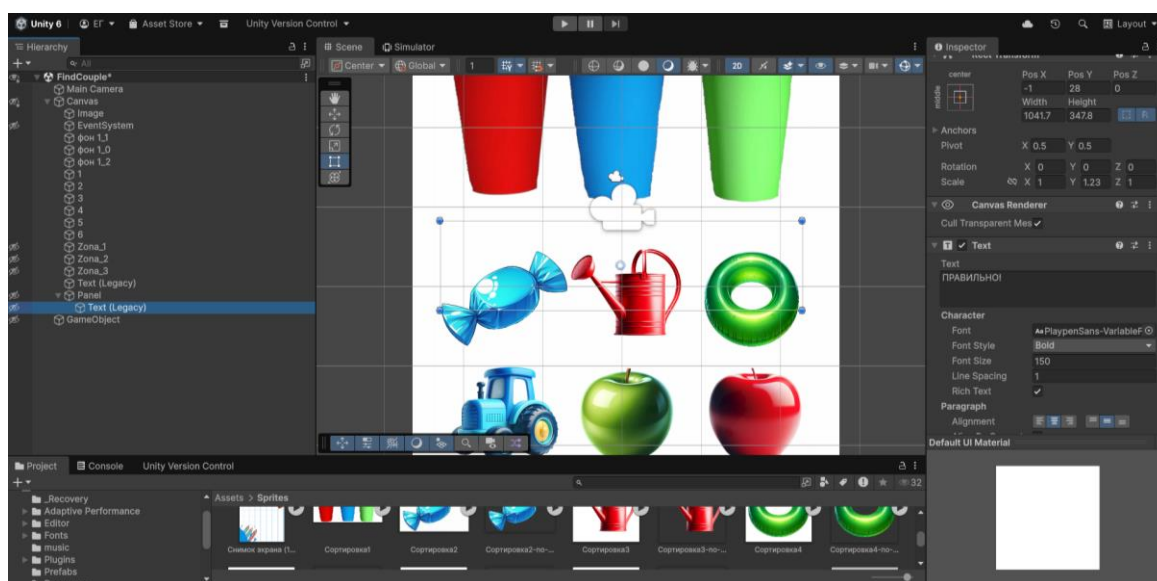


Рисунок 9 – Настройка текстового элемента панели сообщений

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).

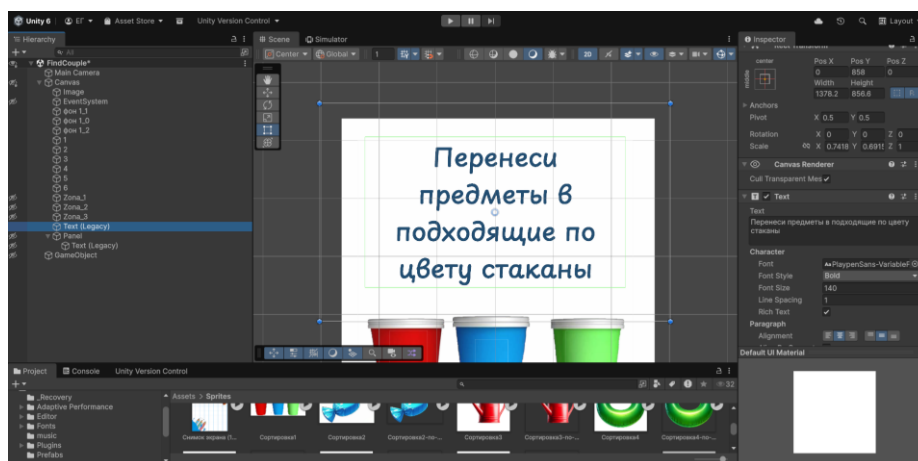


Рисунок 10 – Настройка инструкции игры

После добавление всех изображений на сцену, необходимо добавить область на стаканы, для того чтобы предметы 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 Zona_1, Zona_2, Zona_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. 11).

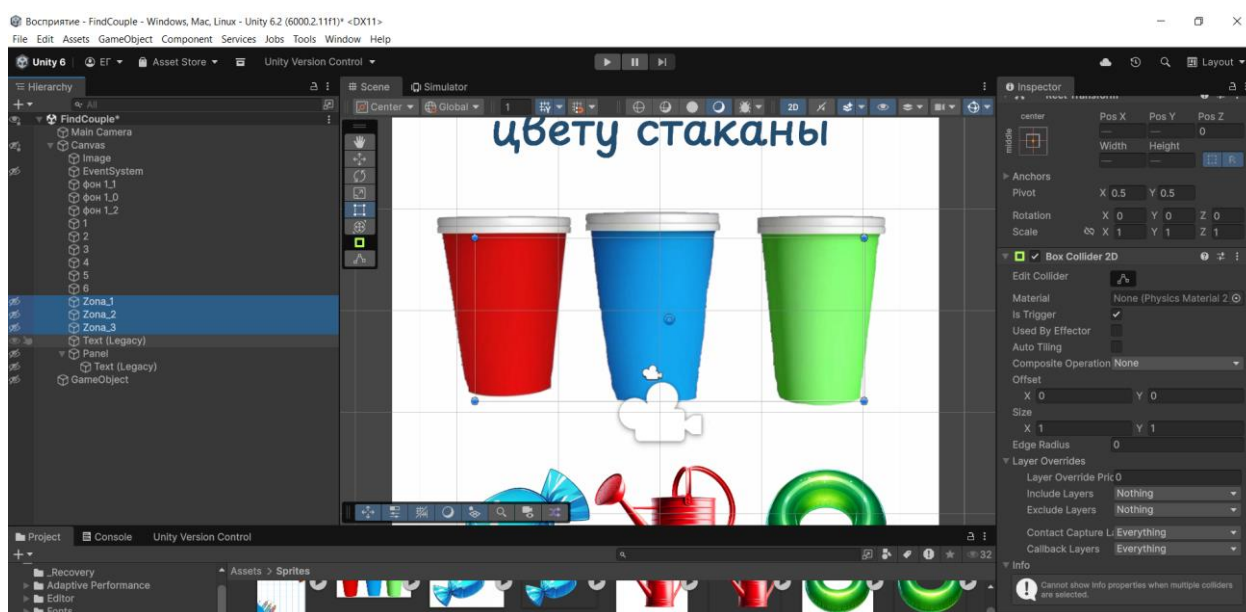


Рисунок 11 – Добавление зон для предметов и их настройка

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

Для компонента GameController необходимо создать скрипт DragAndDropGame.

Скрипт DragAndDropGame является центральным управляющим компонентом игры, реализующим всю основную логику игрового процесса. Он отвечает за координацию взаимодействия игрока с предметами, проверку условий выполнения заданий и управление состоянием уровня. Скрипт обеспечивает плавную и интуитивно понятную механику перетаскивания объектов с последующей проверкой их правильного размещения.

Основная функциональность включает управление коллекцией предметов, каждый из которых имеет целевую зону размещения, параметры притягивания и отслеживание текущего состояния. Для обработки пользовательского ввода используется современная система Input System,

обеспечивающая кроссплатформенную поддержку сенсорных устройств и компьютеров. При перетаскивании предмета постоянно отслеживается его позиция, а при отпуске определяется ближайшая возможная зона размещения.

Логика размещения предметов включает проверку расстояния до целевой зоны с использованием параметра `snapDistance`. При успешном размещении в правильной зоне предмет фиксируется и воспроизводится соответствующий звуковой эффект. В случае ошибки предмет плавно возвращается на исходную позицию с анимированным движением, скорость которого настраивается через параметр `returnSpeed`.

Скрипт также управляет состоянием завершения уровня, отслеживая правильное размещение всех предметов. При выполнении условий активируются элементы пользовательского интерфейса с поздравлением, воспроизводится победный звук и через заданный интервал происходит автоматический переход в главное меню. Дополнительно реализована возможность перезапуска текущего уровня без перезагрузки всей сцены.

Визуальная и звуковая обратная связь обеспечивается через систему аудиоэффектов, которые сопровождают ключевые действия игрока. Техническая сторона включает автоматическую инициализацию компонентов, проверку целостности данных и обработку исключительных ситуаций, что делает скрипт устойчивым к ошибкам конфигурации. Все параметры доступны для настройки через инспектор Unity, позволяя быстро адаптировать игру под различные дизайн-требования и уровни сложности.

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

public class DragPuzzleGame : MonoBehaviour
{
    [System.Serializable]
    public class DraggableItem
    {
        public GameObject itemObject;
        public GameObject targetCup;
        public float snapDistance = 1.0f;
        [HideInInspector] public Vector2 startPosition;
        [HideInInspector] public bool isInPlace;
    }

    public List<DraggableItem> items = new List<DraggableItem>();
    public List<GameObject> allCups = new List<GameObject>();
    public GameObject winMessage;

    public float returnSpeed = 10f;
    public string menuSceneName = "Menu";

    public AudioClip itemPickupSound;
    public AudioClip itemReturnSound;
    public AudioClip itemCorrectSound;
```

```
public AudioClip gameCompleteSound;

public Text winText;

private DraggableItem currentDraggedItem;
private Vector2 offset;
private Camera mainCamera;
private bool isCompleted = false;
private AudioSource audioSource;
private bool isReturning = false;
private Vector2 pointerPosition;

private PlayerInput playerInput;
private InputAction touchPositionAction;
private InputAction touchPressAction;

void Start()
{
    mainCamera = Camera.main;

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

    playerInput = GetComponent<PlayerInput>();
    if (playerInput == null)
    {
        playerInput = gameObject.AddComponent<PlayerInput>();
    }

    SetupInputActions();
    InitializeItems();

    if (winMessage != null)
    {
        winMessage.SetActive(false);
    }

    if (winText != null)
    {
        winText.gameObject.SetActive(false);
    }
}

void SetupInputActions()
{
    var touchActionMap = new InputActionMap("Touch");

    touchPositionAction = touchActionMap.AddAction("Position",
        InputActionType.Value,
        "<Pointer>/position");

    touchPressAction = touchActionMap.AddAction("Press",
        InputActionType.Button,
        "<Pointer>/press");

    touchPressAction.started += ctx => OnTouchStarted(ctx);
    touchPressAction.canceled += ctx => OnTouchEnded(ctx);

    touchActionMap.Enable();
}
```

```

void OnTouchStarted(InputAction.CallbackContext context)
{
    if (isCompleted || isReturning) return;

    pointerPosition = touchPositionAction.ReadValue<Vector2>();
    TryStartDrag(pointerPosition);
}

void OnTouchEnded(InputAction.CallbackContext context)
{
    if (currentDraggedItem == null) return;
    EndDrag();
}

void InitializeItems()
{
    foreach (var item in items)
    {
        if (item.itemObject != null)
        {
            item.startPosition = item.itemObject.transform.position;
            item.isInPlace = false;
            AddColliderIfMissing(item.itemObject);
        }
    }

    if (allCups.Count == 0)
    {
        try
        {
            GameObject[] cupsArray =
GameObject.FindGameObjectsWithTag("Cup");
            if (cupsArray.Length > 0)
            {
                allCups.AddRange(cupsArray);
            }
        }
        catch (UnityException) { }
    }

    foreach (var item in items)
    {
        if (item.targetCup == null)
        {
            Debug.LogError($"Предмет {item.itemObject?.name} не имеет
назначенного targetCup!");
        }
    }
}

void Update()
{
    if (isCompleted) return;

    if (touchPositionAction != null && touchPositionAction.enabled)
    {
        pointerPosition = touchPositionAction.ReadValue<Vector2>();
    }

    if (currentDraggedItem != null && !isReturning)
    {
        UpdateItemPosition();
    }
}

```

```

void UpdateItemPosition()
{
    if (currentDraggedItem != null && !isReturning)
    {
        Vector2 worldPos = mainCamera.ScreenToWorldPoint(pointerPosition);
        currentDraggedItem.itemObject.transform.position = worldPos +
offset;
    }
}

void TryStartDrag(Vector2 inputPosition)
{
    if (isReturning) return;

    Vector2 worldPos = mainCamera.ScreenToWorldPoint(inputPosition);
    RaycastHit2D hit = Physics2D.Raycast(worldPos, Vector2.zero);

    if (hit.collider != null)
    {
        foreach (var item in items)
        {
            if (hit.collider.gameObject == item.itemObject &&
!item.isInPlace)
            {
                currentDraggedItem = item;
                offset = (Vector2)item.itemObject.transform.position -
worldPos;
                PlaySound(itemPickupSound);
                break;
            }
        }
    }
}

void EndDrag()
{
    if (currentDraggedItem == null) return;

    bool isCorrectCup = false;
    GameObject closestCup = null;
    float closestDistance = float.MaxValue;

    foreach (var cup in allCups)
    {
        if (cup == null) continue;

        float distance = Vector2.Distance(
            currentDraggedItem.itemObject.transform.position,
            cup.transform.position
        );

        if (distance < closestDistance)
        {
            closestDistance = distance;
            closestCup = cup;
        }
    }

    if (closestCup != null && closestDistance <=
currentDraggedItem.snapDistance)
    {
        if (closestCup == currentDraggedItem.targetCup)
        {

```

```

        isCorrectCup = true;
        PlaceInCup(closestCup);
    }
    else
    {
        isCorrectCup = false;
        StartCoroutine(ReturnToStartPosition());
    }
}
else
{
    StartCoroutine(ReturnToStartPosition());
}

if (!isCorrectCup)
{
    PlaySound(itemReturnSound);
}

currentDraggedItem = null;
}

void PlaceInCup(GameObject cup)
{
    currentDraggedItem.itemObject.transform.position =
cup.transform.position;
    currentDraggedItem.isInPlace = true;
    PlaySound(itemCorrectSound);
    Destroy(currentDraggedItem.itemObject);
    CheckWinCondition();
}

IEnumerator ReturnToStartPosition()
{
    isReturning = true;
    Vector2 startPos = currentDraggedItem.startPosition;
    GameObject item = currentDraggedItem.itemObject;

    while (Vector2.Distance(item.transform.position, startPos) > 0.1f)
    {
        item.transform.position = Vector2.Lerp(
            item.transform.position,
            startPos,
            returnSpeed * Time.deltaTime
        );
        yield return null;
    }

    item.transform.position = startPos;
    isReturning = false;
}

void CheckWinCondition()
{
    foreach (var item in items)
    {
        if (!item.isInPlace) return;
    }

    StartCoroutine(CompleteGame());
}

IEnumerator CompleteGame()
{

```

```
        isCompleted = true;
        PlaySound(gameCompleteSound);

        if (winMessage != null)
        {
            winMessage.SetActive(true);
        }
        else if (winText != null)
        {
            winText.gameObject.SetActive(true);
        }

        yield return new WaitForSeconds(2f);

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

    void AddColliderIfMissing(GameObject obj)
    {
        if (obj != null && obj.GetComponent<Collider2D>() == null)
        {
            obj.AddComponent<BoxCollider2D>();
        }
    }

    public void RestartGame()
    {
        isCompleted = false;

        if (winMessage != null)
        {
            winMessage.SetActive(false);
        }

        if (winText != null)
        {
            winText.gameObject.SetActive(false);
        }

        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

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

    void OnDisable()
    {
        if (touchPressAction != null)
        {
            touchPressAction.started -= OnTouchStarted;
            touchPressAction.canceled -= OnTouchEnded;
        }
    }

    void OnValidate()
    {

```

```

foreach (var item in items)
{
    if (item.snapDistance < 0.1f)
    {
        item.snapDistance = 0.1f;
    }
}
}

```

После создания скрипта DragAndDropGame необходимо добавить его к компоненту GameController. Для этого необходимо перетащить скрипт к компоненту в Hierarchy и его настроить его параметры через инспектор: в разделе Game Objects необходимо указать количество элементов предметов, в данном случае 6, после чего для каждого элемента последовательно задать параметры - в поле All Cups перетащить соответствующий игровой объект (Zona_1, Zona_2, Zona_3), в поле Target Area назначить соответствующие зоны фиксации (Зона_1, Зона_2, Зона_3). Далее необходимо настроить дополнительные параметры: в поле Win Message указать объект с сообщением о победе, и добавить соответствующие звуковые файлы в поля для звуковых эффектов (см. рис. 12).



Рисунок 12 – Добавление скрипта к компоненту GameController и его настройка

Итоговый результат создания игры «Сортировка» (см. рис.13).



Рисунок 13 – Готовая игра «Сортировка»

4. Выводы

В данной работе показан процесс создания игры «Сортировка» 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.