

Создание мобильной игры «Что подходит?» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель данной статьи – создание мобильной игры «Что подходит?» 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 «что подходит?»

Ключевые слова: Unity, игра, соответствие, предмет.

Creating a mobile game "What's the right choice?" using Unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create a mobile game "What's the match?" 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 "what's the match?"

Keywords: Unity, game, match, object.

1 Введение

1.1 Актуальность

В современном мире мобильные игры прочно вошли в повседневную жизнь миллионов людей, став не только способом развлечения, но и мощной индустрией с многомиллиардными оборотами. Среди всего разнообразия жанров стабильной популярностью пользуются казуальные головоломки, сочетающие простоту освоения с глубоким и увлекательным геймплеем.

Движок Unity, являясь одним из лидеров в среде разработки игр, предоставляет мощные и в то же время доступные инструменты для создания кроссплатформенных проектов. Это делает его идеальным выбором для реализации подобных проектов, позволяя быстро создавать прототипы и выпускать готовый продукт на все популярные платформы.

Таким образом, разработка мобильной игры «Что подходит?» — головоломки на логику и ассоциации — представляет собой актуальную и

практически значимую задачу. Проект соответствует рыночному спросу на интеллектуальный досуг, обладает значительным образовательным потенциалом для развития мышления и является отличным примером применения современных инструментов игровой индустрии для решения конкретных дизайнерских задач.

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

2 Материалы и методы

Для создания игры «Что подходит?» в качестве игрового движка выбран Unity, поскольку он отличается низким порогом входа, интуитивным интерфейсом и возможностью написания кода на C#. Для редактирования скриптов использовалась интегрированная среда разработки visual studio community.

3 Результаты

Первый этап создания мобильной игры «Что подходит?» — настройка нового проекта в Unity. Для этого в лаунчере Unity Hub необходимо выбрать шаблон для двухмерной разработки Universal 2D, задать название проекту и создать его (см. рис. 1).

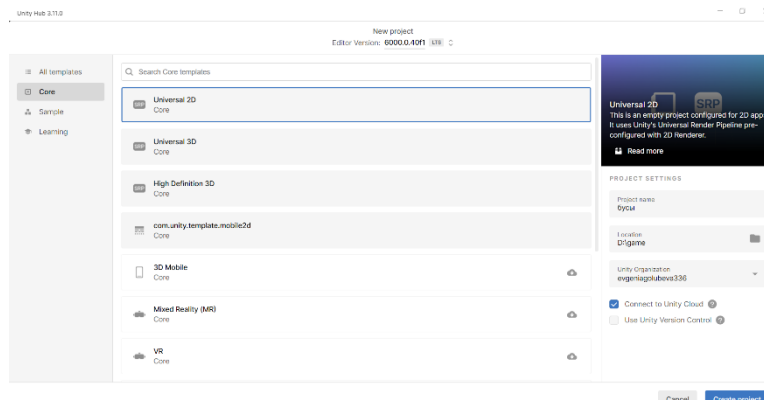


Рисунок-1 Создание проекта

После создания проекта на основе шаблона Universal 2D, необходимо провести базовую настройку под мобильную платформу. В первую очередь, в настройках построения (File -> Build Profiles) следует переключить платформу на Android (см. рис. 2).

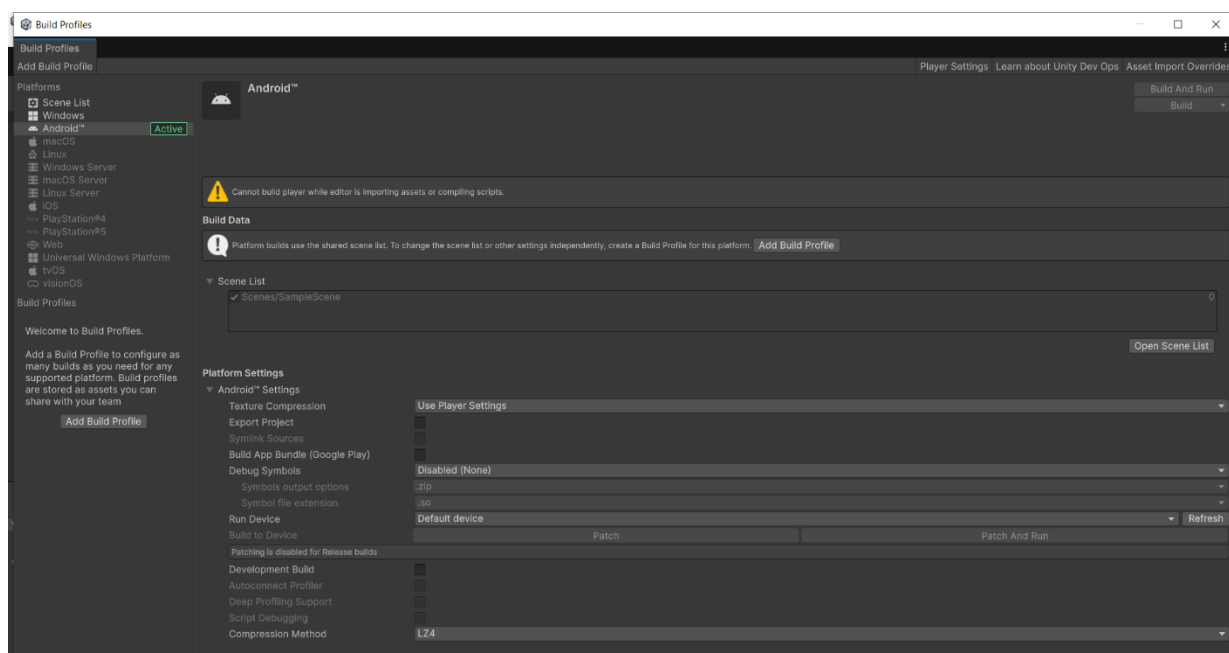


Рисунок- 2 Настройка проекта

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

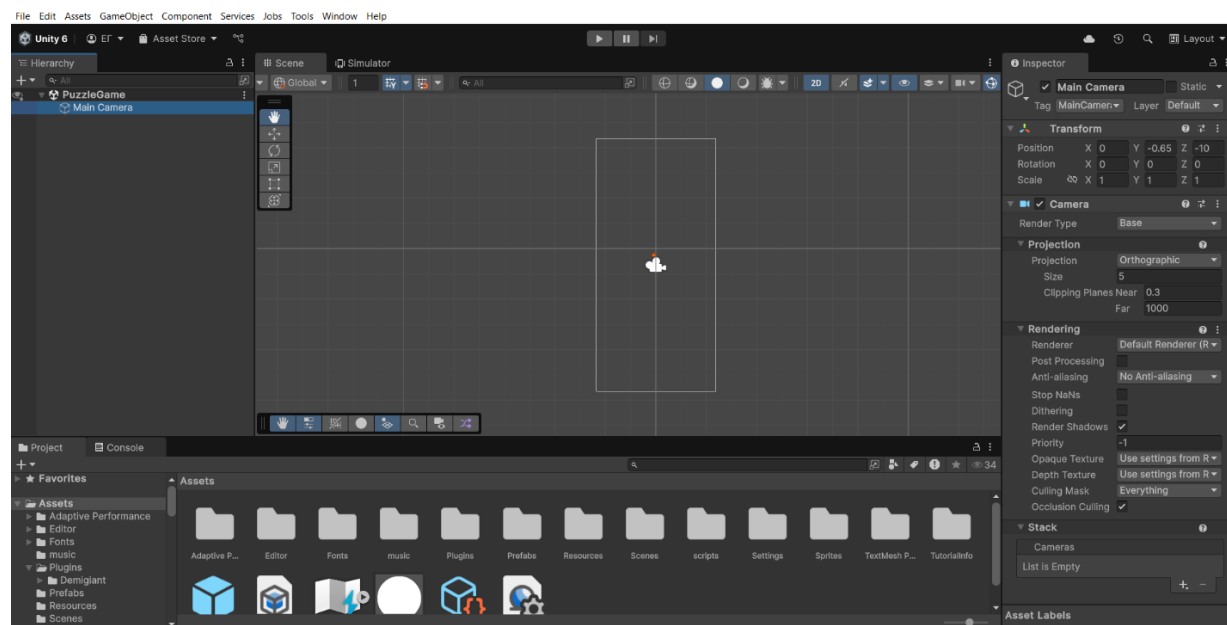


Рисунок-3 Настройка камеры

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

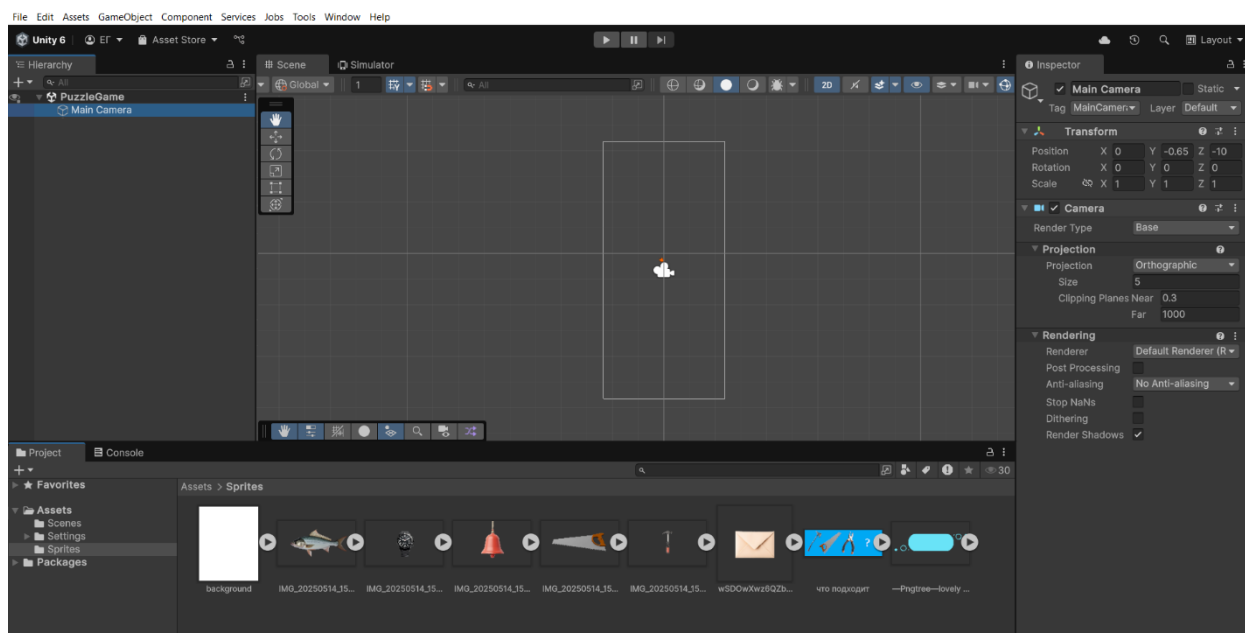


Рисунок 4 - Добавление изображений

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).

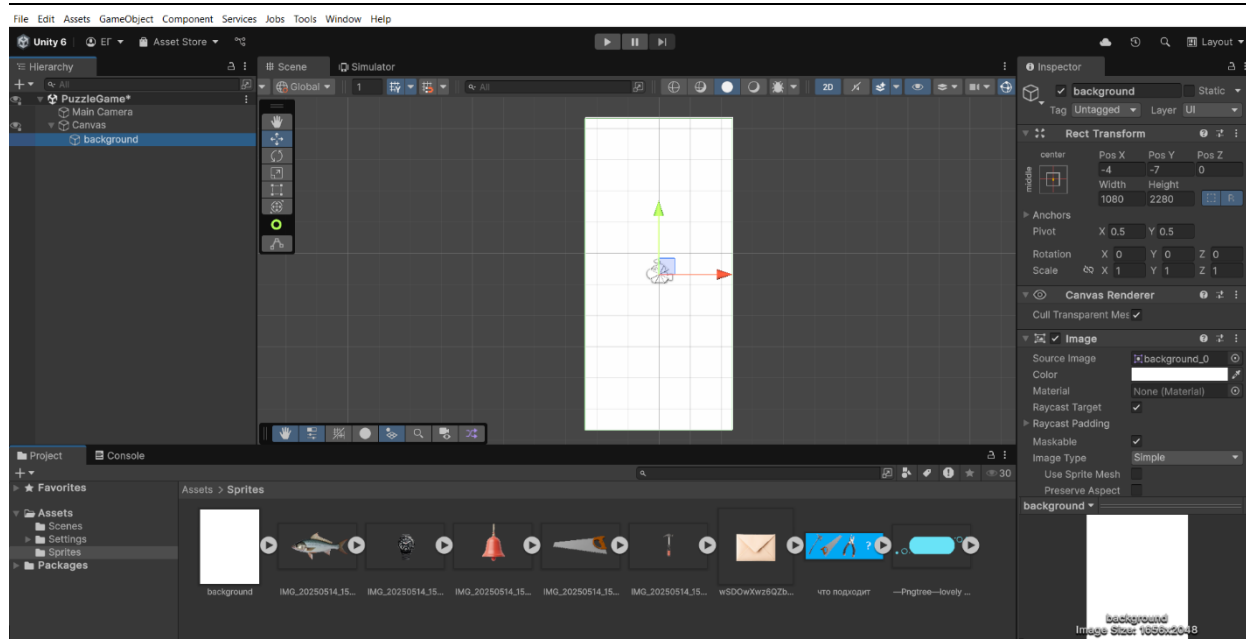


Рисунок 5 – Настройка фона игры

Следующим шагом добавляем изображения панели с предметами, куда необходимо перенести недостающий предмет. Для этого переносим изображение в Hierarchy и настраиваем параметры (см.рис.6).

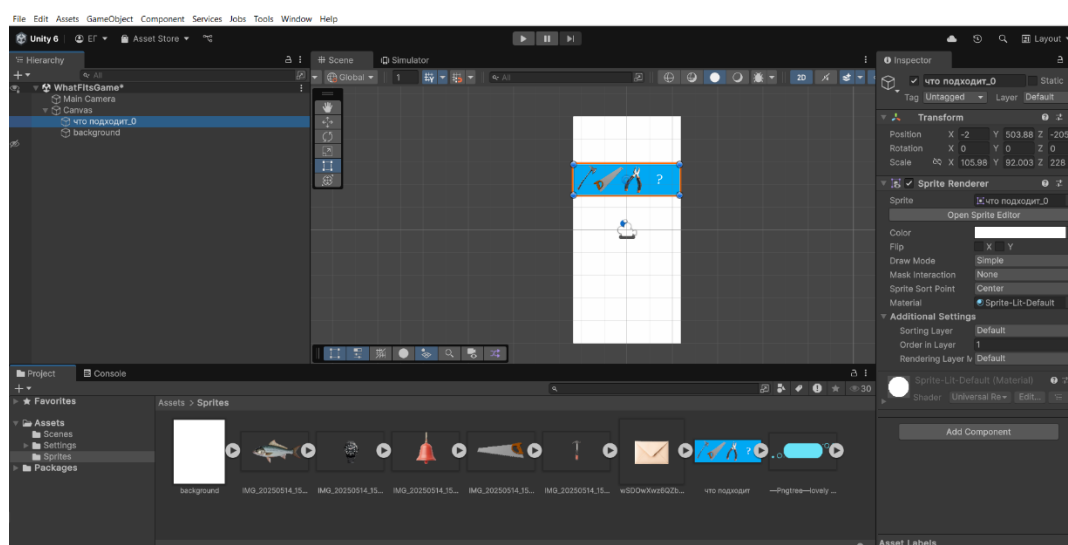


Рисунок 6 – Добавление изображения панели сцену

Далее необходимо добавить инструкцию игры и панель для отображения уведомления «Правильно!». Для этого в Hierarchy необходимо создать новый UI-объект. Правой кнопкой мыши вызываем меню в пространстве иерархии и выбираем UI -> Panel. Далее для инструкции добавляем текст UI -> Text.

После добавления панели и текста, необходимо их настроить. Для панели устанавливаем якоря (Anchors) в центральное положение с значениями (0.5, 0.5), что обеспечивает правильное позиционирование относительно разрешения экрана. Задаем размеры области Width 1080, Height 2280. В

компоненте Image выбираем подходящий спрайт для фонового изображения, настраиваем цвет и прозрачность в соответствии с визуальным стилем игры, а также устанавливаем Image Type в значение Simple для оптимального отображения. (см.рис.7).

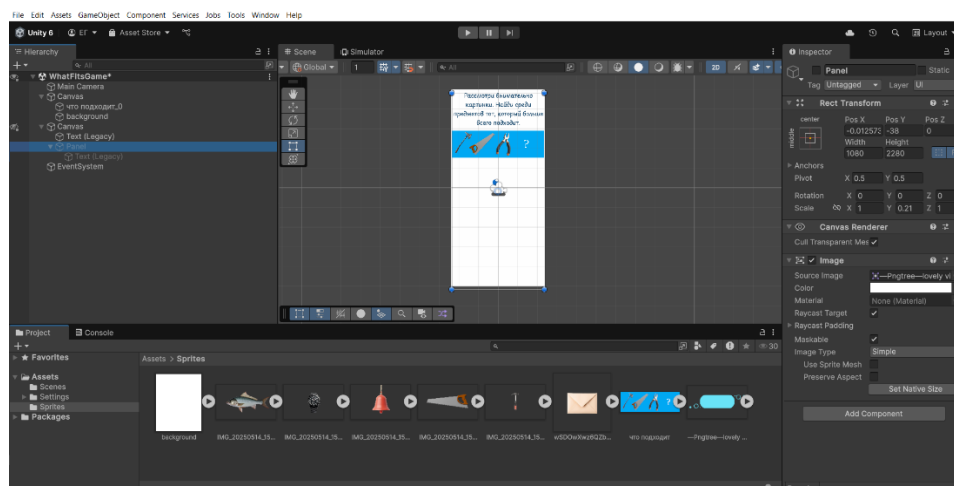


Рисунок 7 – Настройка панели

Также необходимо настроить текстовый блок с уведомлением "Правильно!". В компоненте Text (Legacy) выбираем шрифт PlaypenSans-VariableFont_wght с жирным начертанием Bold и устанавливаем размер шрифта 100 пунктов для обеспечения хорошей читаемости. В разделе Rect Transform задаем точные размеры текстового поля: Width 823,7, Height 1747,3 и позиционирование относительно родительской панели с координатами Pos X: 17, Pos Y: 36. Особое внимание уделяется настройке якорей (Anchors) с центром в точке (0.5, 0.5), что гарантирует правильное отображение текста на экранах с разным разрешением. (см.рис. 8).

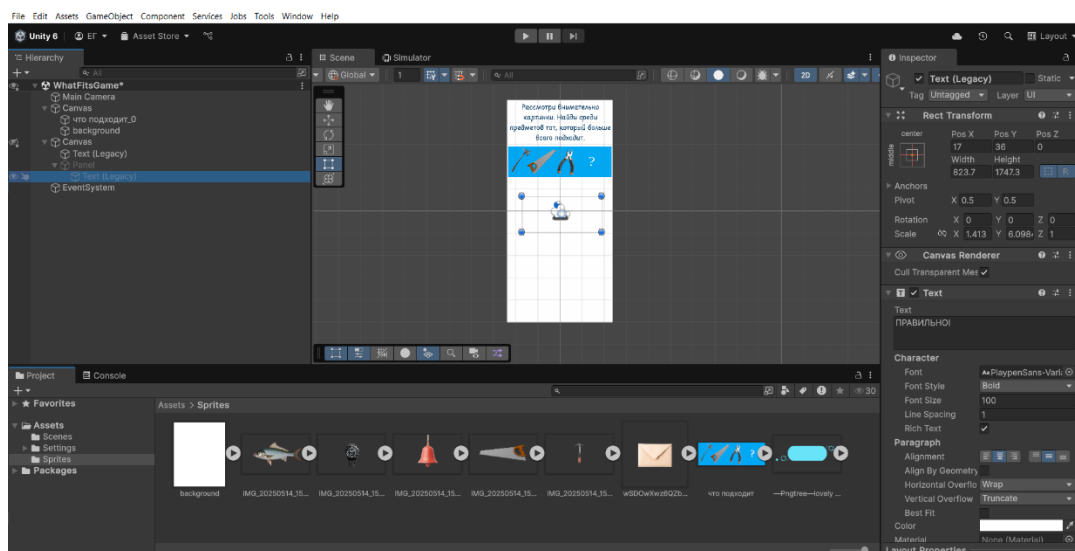


Рисунок 8 – Настройка текстового элемента панели сообщений

Для текстового элемента настраиваем шрифт Pangolin-Regular с жирным начертанием и размером 65 пунктов с жирным начертанием(Bold,

выравнивание по центру, включаем поддержку Rich Text для форматирования, а в Rect Transform задаем точные размеры текстового поля: Width 1039, Height 399,3 и позиционирование относительно родительской панели с координатами Pos X: 3, Pos Y: 913,4 и устанавливаем якоря Anchors с центром в точке (0.5, 0.5) (см.рис.9).

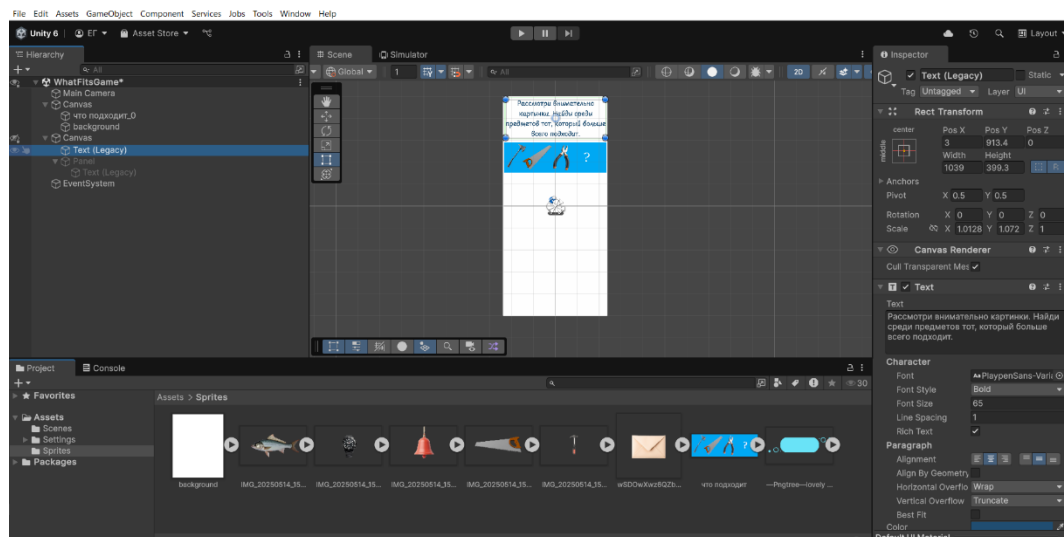


Рисунок 9 – Настройка инструкции игры

Далее добавляем изображения предметов на сцену и в hierarchy переименовываем на Frame_1, Frame_2, Frame_3, Frame_4, Frame_5 и настраиваем их, чтобы их можно было передвигать. Для этого в inspector каждого изображения предметов добавляем компонент Sprite Renderer, где назначается визуальный спрайт предметов, а в качестве материала выбирается Sprite-Lit-Default для совместимости с URP. Далее добавляем Rigidbody 2D, и выбираем режим Kinematic, который позволяет управлять ее движением через код. Для оптимизации замораживаем позицию по осям X и Y в Constraints, чтобы предмет не падал под действием гравитации. (см.рис.10).

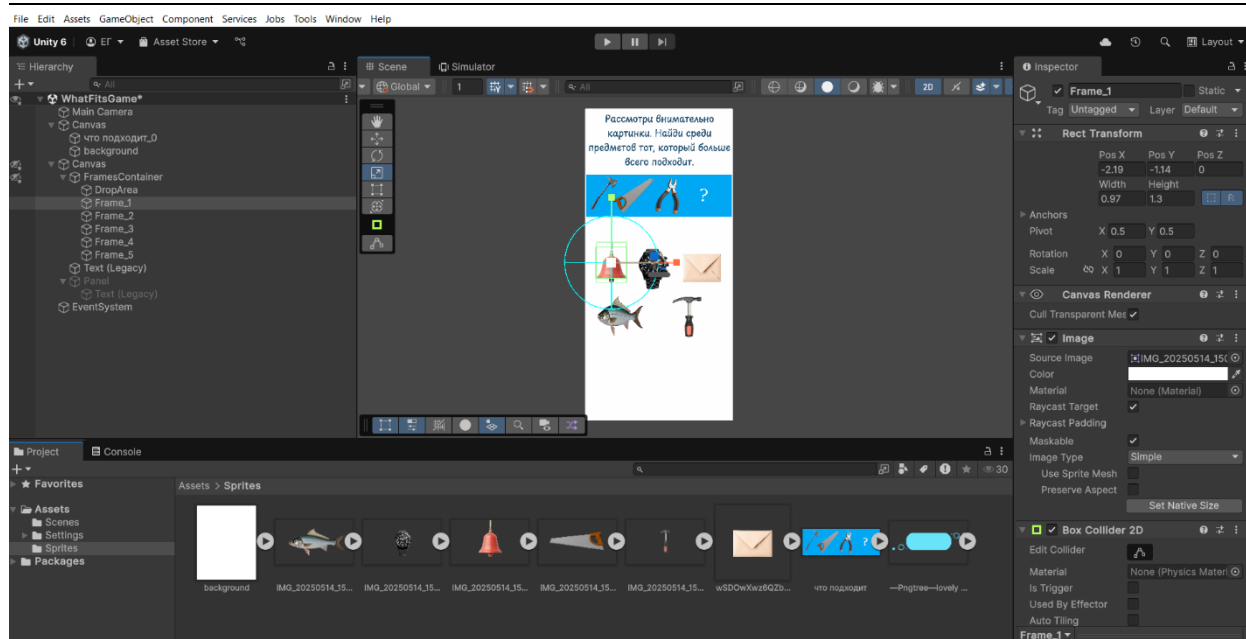


Рисунок 10 – Добавление предметов на сцену и настройка параметров

Также необходимо создать область, куда игрок должен перетащить подходящий предмет. Для этого в Hierarchy создаем пустой игровой объект через правый клик и выбираем Create Empty. Этот объект необходимо переименовать в DropArea. Его основное предназначение — отмечать на сцене точку, где должен находиться предмет. Для реализации этого функционала к DropArea необходимо добавить компонент Box Collider 2D. В настройках коллайдера обязательно надо включить опцию Is Trigger, чтобы объект не создавал физического препятствия, но мог регистрировать пересечения с другими коллайдерами. Размер коллайдера Size настраивается так, чтобы он визуально соответствовал пустому месту на нитке бус. Через компонент Transform задается точное положение объекта на сцене, обеспечивая его выравнивание с уже существующими бусинами (см.рис.11).

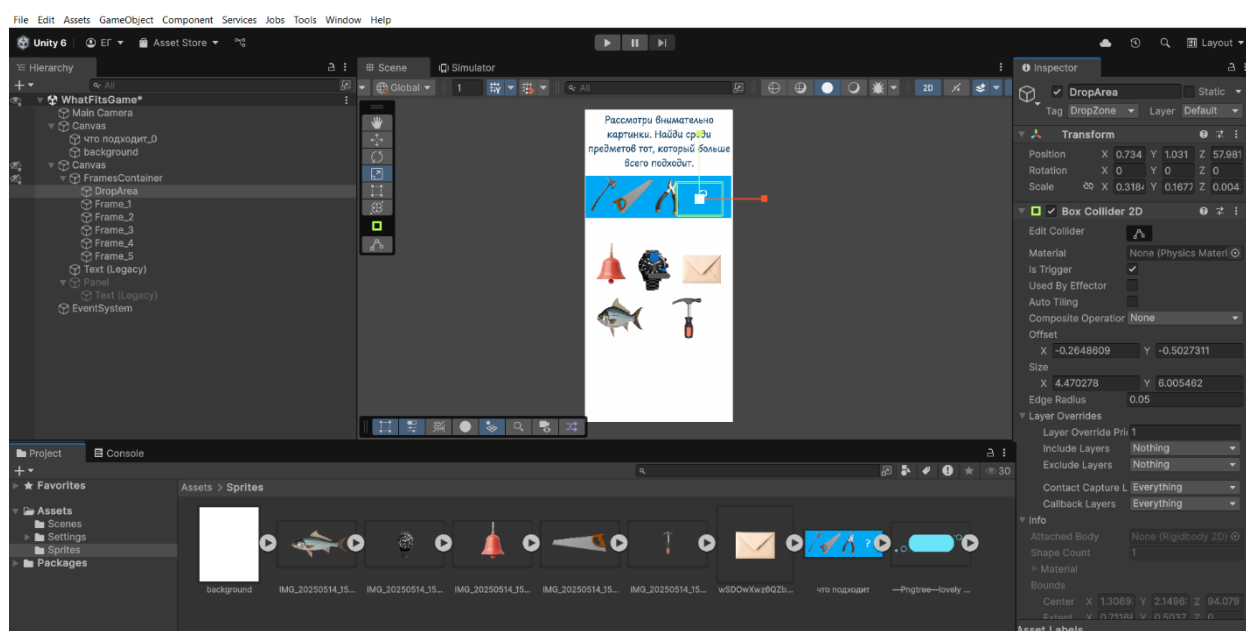


Рисунок 11 – Добавление области для предмета на сцену и настройка параметров

После добавления всех необходимых компонентов необходимо создать скрипт под названием `ObjectDragHandler`, который реализует основную игровую механику перетаскивания предметов. Данный скрипт содержит несколько ключевых функциональных блоков:

Механизм перетаскивания реализован через корутину `DragObject()`, которая отслеживает положение мыши и перемещает объект с учетом ограничений. При активации объект временно смещается по оси *Z* для визуального выделения. Система проверки размещения использует радиус `snapDistance` для определения допустимой зоны сброса. При попадании в зону с тегом `"DropZone"` объект проверяется на соответствие `correctObject`, после чего либо фиксируется (`SnapToTarget`), либо возвращается на исходную позицию через плавную анимацию (`SmoothReturn`).

Визуальная обратная связь включает панель успеха с анимированным появлением и исчезновением через изменение прозрачности `CanvasGroup`. Аудио-сопровождение состоит из трех элементов: звук перетаскивания (`dragSound`), успешного размещения (`successSound`) и ошибки (`wrongSound`). Для обработки коллизий используется `OverlapCircleAll`, а для отладки — визуализация радиуса через `Gizmos`.

Оптимизация взаимодействия обеспечивается проверкой `IsPointerOverUI()`, блокирующей перетаскивание при нахождении курсора над элементами интерфейса. При успешном завершении игры запускается автоматический переход в меню через заданную задержку.

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

public class DragDropGame : MonoBehaviour
{
    [Header("Game Settings")]
    [Tooltip("Перетащите сюда ПРАВИЛЬНЫЙ объект")]
    public GameObject correctObject; // Указываем правильный объект через Inspector
    public float snapDistance = 1.5f; // Дистанция для "прилипания" к цели

    [Header("UI Settings")]
    public RectTransform successPanel;
    public Text messageText;
    public float messageShowTime = 2f;

    [Header("Audio Settings")]
    public AudioClip successSound;
    public AudioClip wrongSound;
    public AudioClip dragSound; // Добавляем звук перетаскивания

    [Header("Scene Management")]
    public string menuSceneName = "Menu";
```

```

private Vector3 originalPosition;
private bool isCorrectlyPlaced;
private AudioSource audioSource;
private Collider2D targetCollider;
private CanvasGroup canvasGroup;
private bool isDragging; // Флаг для отслеживания состояния перетаскивания

void Start()
{
    originalPosition = transform.position;
    audioSource = GetComponent<AudioSource>();
    targetCollider = GetComponent<Collider2D>();
    isDragging = false;

    // Инициализация UI
    if (successPanel != null)
    {
        canvasGroup = successPanel.GetComponent<CanvasGroup>();
        if (canvasGroup == null)
            canvasGroup = new CanvasGroup();
        successPanel.gameObject.AddComponent<CanvasGroup>();

        canvasGroup.alpha = 0;
        successPanel.gameObject.SetActive(false);
    }
}

private void OnMouseDown()
{
    if (!isCorrectlyPlaced && !IsPointerOverUI())
    {
        StartCoroutine(DragObject());
    }
}

private bool IsPointerOverUI()
{
    return EventSystem.current.IsPointerOverGameObject();
}

private IEnumerator DragObject()
{
    // Воспроизводим звук при начале перетаскивания
    PlayDragSound();
    isDragging = true;

    // Поднимаем объект над другими
    transform.position += new Vector3(0, 0, -1);

    while (Input.GetMouseButton(0))
    {
        Vector3 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        transform.position = new Vector3(mousePos.x, mousePos.y, transform.position.z);
        yield return null;
    }

    // Завершаем перетаскивание
    isDragging = false;

    // Возвращаем Z-позицию
    transform.position = new Vector3(transform.position.x, transform.position.y, 0);
}

```

```

        CheckPlacement();
    }

    private void CheckPlacement()
    {
        Collider2D[] hits = Physics2D.OverlapCircleAll(transform.position,
snapDistance);
        bool isOnTarget = false;

        foreach (Collider2D hit in hits)
        {
            if (hit.CompareTag("DropZone"))
            {
                isOnTarget = true;
                if (gameObject == correctObject) // Проверка на правильный
объект
                {
                    SnapToTarget(hit.transform.position);
                    HandleCorrectPlacement();
                }
                else
                {
                    PlaySound(wrongSound);
                    ReturnToOriginalPosition();
                }
                break;
            }
        }

        if (!isOnTarget)
        {
            ReturnToOriginalPosition();
        }
    }

    private void SnapToTarget(Vector3 targetPos)
    {
        transform.position = targetPos;
        isCorrectlyPlaced = true;
        targetCollider.enabled = false;
    }

    private void ReturnToOriginalPosition()
    {
        StartCoroutine(SmoothReturn(originalPosition, 0.3f));
    }

    private IEnumerator SmoothReturn(Vector3 targetPos, float duration)
    {
        Vector3 startPos = transform.position;
        float elapsed = 0f;

        while (elapsed < duration)
        {
            transform.position = Vector3.Lerp(startPos, targetPos, elapsed /
duration);
            elapsed += Time.deltaTime;
            yield return null;
        }

        transform.position = targetPos;
    }

    private void HandleCorrectPlacement()

```

```
{
    PlaySound(successSound);
    ShowSuccessMessage();
    StartCoroutine(LoadMenuAfterDelay(messageShowTime + 1f));
}

// Метод для воспроизведения звука перетаскивания
private void PlayDragSound()
{
    if (dragSound != null)
    {
        if (audioSource != null)
            audioSource.PlayOneShot(dragSound);
        else
            AudioSource.PlayClipAtPoint(dragSound,
Camera.main.transform.position);
    }
}

private void PlaySound(AudioClip clip)
{
    if (clip != null)
    {
        if (audioSource != null)
            audioSource.PlayOneShot(clip);
        else
            AudioSource.PlayClipAtPoint(clip,
Camera.main.transform.position);
    }
}

private void ShowSuccessMessage()
{
    if (successPanel == null) return;

    successPanel.gameObject.SetActive(true);
    if (messageText != null)
        messageText.text = "Правильно! Молодец!";

    StartCoroutine(FadePanel(0, 1, 0.5f));
    StartCoroutine(FadePanel(1, 0, 0.5f, messageShowTime));
}

private IEnumerator FadePanel(float from, float to, float duration, float
delay = 0f)
{
    if (delay > 0)
        yield return new WaitForSeconds(delay);

    float elapsed = 0f;
    while (elapsed < duration)
    {
        canvasGroup.alpha = Mathf.Lerp(from, to, elapsed / duration);
        elapsed += Time.deltaTime;
        yield return null;
    }

    canvasGroup.alpha = to;

    if (to == 0)
        successPanel.gameObject.SetActive(false);
}

private IEnumerator LoadMenuAfterDelay(float delay)
```

```
{
    yield return new WaitForSeconds(delay);
    if (!string.IsNullOrEmpty(menuSceneName))
    {
        SceneManager.LoadScene(menuSceneName);
    }
}

void OnDrawGizmosSelected()
{
    Gizmos.color = Color.cyan;
    Gizmos.DrawWireSphere(transform.position, snapDistance);
}
}
```

После создания скрипта `ObjectDragHandler` необходимо добавить его к предметам в сцене. Для этого необходимо перетащить скрипт на каждый предмет в `Hierarchy`. После добавления скрипта требуется настроить его параметры: укажите цвет бусины (`Bead Color`) и соответствующий правильный цвет (`Correct Bead Color`), установите радиус размещения (`Placement Radius`) около 0.5. В `Message Panel` добавляем панель сообщений, а в `Message Text` текстовое поле панели, а также добавляем звуковые файлы для перетаскивания в `Drag Sound`, а для успешного размещения в `Success Sound`.

Для каждого предмета необходимо указать правильный объект (`Correct Object`) - если текущий предмет является целевым для размещения, следует перетащить его же в это поле. Радиус притяжения (`Snap Distance`) определяет расстояние, на котором предмет автоматически займет правильную позицию.

В разделе `UI Settings` настраивается визуальная обратная связь: панель успеха `Success Panel` должна содержать UI-элемент для отображения сообщения, а в поле `Message Text` указывается текстовый компонент. Время показа сообщения `Message Show Time` устанавливает продолжительность отображения панели.

Аудио настройки включают три типа звуков: успешного размещения `Success Sound`, ошибки `Wrong Sound` и перетаскивания `Drag Sound`. Каждому звуку назначается соответствующий аудиоклип из проекта. Для работы аудиосистемы необходим компонент `AudioSource` на объекте (см. рис. 12).

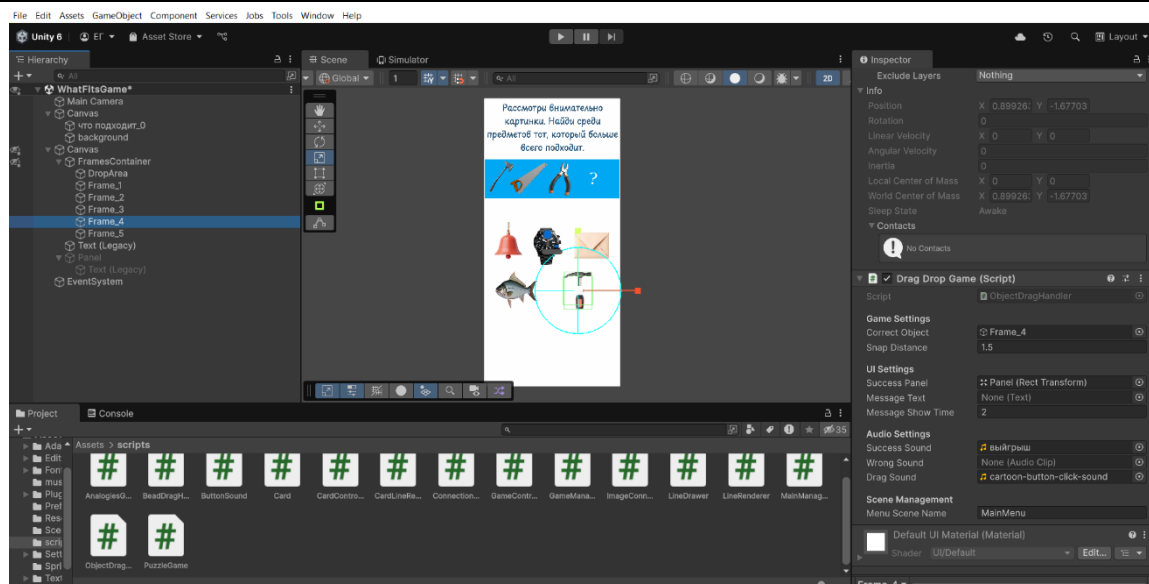


Рисунок 12 – Добавление скрипта к предметам

Итоговый результат создания игры «Что подходит?» (см. рис.13).

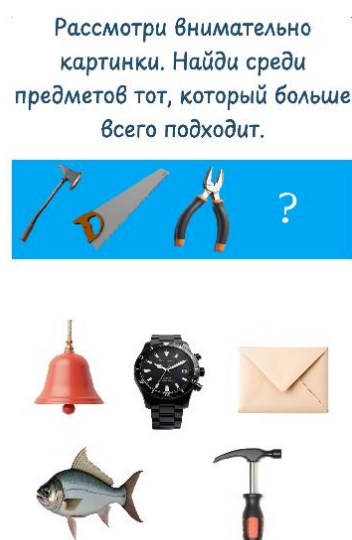


Рисунок 12 – Готовая игра «Что подходит?»

4. Выводы

В данной работе показан процесс создания игры «Что подходит?» 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.