

Создание компьютерной игры «Обыкновенные дроби и смешанные числа» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель статьи – создание компьютерной игры «Обыкновенные дроби и смешанные числа» на Unity. 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 компьютерная игра «Обыкновенные дроби и смешанные числа».

Ключевые слова: Unity, игра, дроби.

Creating a computer game «Common Fractions and Mixed Numbers» using Unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create a computer game "Common Fractions and Mixed Numbers" using Unity. To create the game, we used Unity, a cross-platform environment for creating interactive 2D and 3D applications, which allows us to create 2D games using a powerful physics engine and visual tools. As a result, we created a computer game called "Common Fractions and Mixed Numbers."

Keywords: Unity, game, fractions.

1 Введение

1.1 Актуальность

Одной из важнейших тенденций развития современного образования является его цифровизация, предполагающая применение цифровых и смешанных технологий, разработку интерактивных форм обучения и проектирование новых цифровых средств обучения. В контексте обучения математике цифровизация позволяет сделать образовательный процесс более интересным и доступным благодаря практически неограниченным возможностям визуализации абстрактных понятий. В качестве эффективного средства обучения математике в школе всё чаще предлагается использовать цифровые дидактические игры, обладающие значительным потенциалом в формировании у школьников математических и логических умений, а также

познавательной мотивации. Актуальность данного направления обусловлена необходимостью разработки таких средств обучения, которые не только способствуют развитию математических умений, но и соответствуют запросам учащихся цифрового поколения к форме подачи информации.

Многочисленные исследования подтверждают эффективность применения игровых и контекстных технологий в образовании для получения определённых знаний, умений и навыков, а также для развития логического мышления. Игровые практики способствуют формированию устойчивых когнитивных структур и повышают вовлечённость учащихся в учебную деятельность. Данные международных обзоров свидетельствуют о том, что игровое обучение (Game-Based Learning) значительно улучшает когнитивные результаты, включая концептуальное понимание, навыки решения проблем и критическое мышление. Особенно перспективным представляется использование компьютерных игр, которые позволяют визуализировать абстрактные математические понятия, обеспечивают мгновенную обратную связь и создают среду для непроизвольного обучения через игровую деятельность.

Выбор платформы Unity как инструмента разработки образовательных игр. Среди инструментов разработки образовательных игровых приложений особое место занимает игровой движок Unity. Данная платформа зарекомендовала себя как эффективное средство создания многоуровневых образовательных игр, прошедших валидацию по критериям валидности, практичности и эффективности. Unity предоставляет широкие возможности для визуализации учебного материала, разработки интерактивных механик и создания иммерсивных сред, способствующих повышению мотивации учащихся. Опыт разработки образовательных игр на Unity демонстрирует высокий уровень вовлечённости пользователей - например, в одном из проектов общий показатель удовлетворённости игровым процессом достиг 85,6%.

Несмотря на широкий спектр существующих цифровых инструментов для обучения математике - электронных курсов, игровых платформ, образовательных квестов - наблюдается дефицит комплексных игровых решений, целенаправленно разработанных для отработки навыков работы с обыкновенными дробями и смешанными числами. Существующие разработки часто представляют собой отдельные тестовые задания или презентации, а не полноценные игровые приложения, интегрирующие образовательное содержание в увлекательный игровой процесс. Таким образом, актуальность настоящего исследования определяется необходимостью разработки и создания компьютерной игры по теме «Обыкновенные дроби и смешанные числа» на платформе Unity, которая позволит восполнить существующий пробел в инструментарии цифрового обучения математике, повысить мотивацию и качество усвоения материала у школьников, а также внести вклад в развитие теории и практики геймификации математического образования.

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 компьютерной игры «Обыкновенные дроби и смешанные числа» при помощи кроссплатформенной среды для создания интерактивных 2D и 3D приложений Unity.

2 Материалы и методы

Для создания игры «Обыкновенные дроби и смешанные числа» в качестве игрового движка был 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

Для того, чтобы приступить к созданию игры, необходимо создать новый проект в Unity. Для этого необходимо выбрать в unity hub рендер-пайплайн Universal 2D и нажать кнопку создать проект (см. рис. 1).

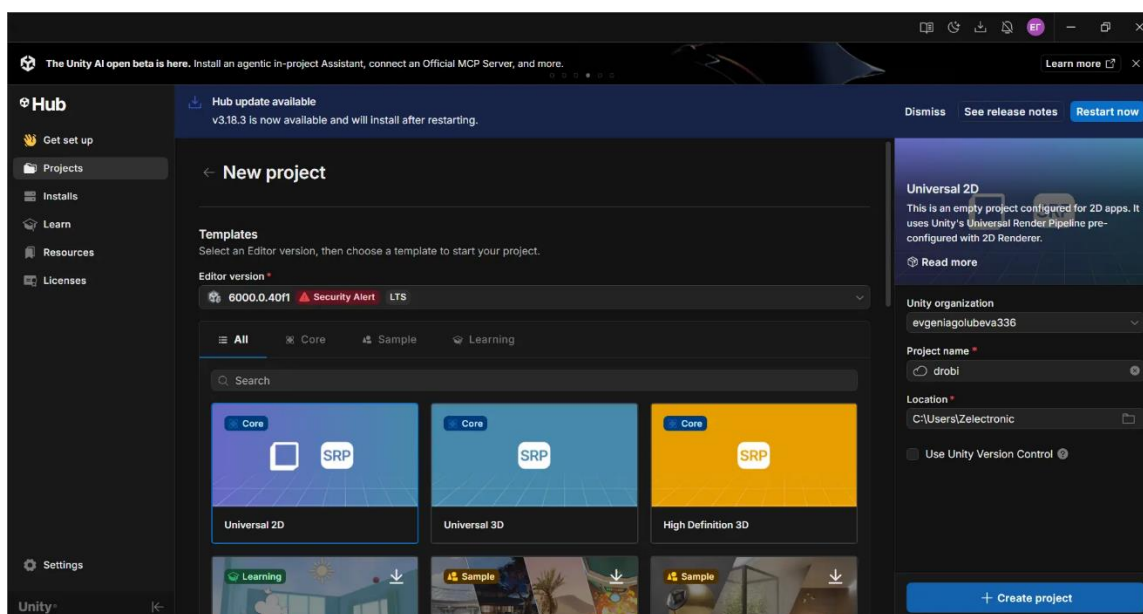


Рисунок 1 – Создание проекта

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

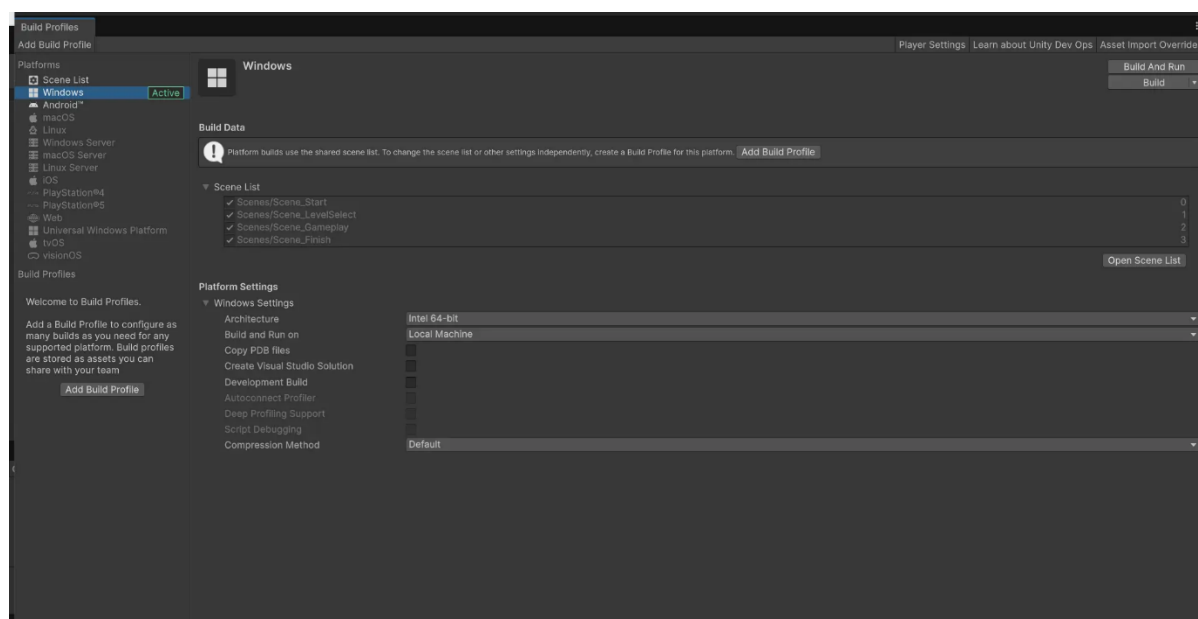


Рисунок 2 – Настройка проекта

Далее необходимо создать четыре сцены игры. Для этого в папке Scene необходимо нажать правой кнопкой мыши по папке, в появившемся окне нажать Create, и выбрать Scene, и так повторить три раза, так как первая сцена уже создана автоматически. Созданные сцены необходимо переименовать. Название первой сцены Scene_Start, второй сцены Scene_LevelSelect, третьей сцены Scene_Gameplay и четвертой сцены Scene_Finish (см. рис. 3).

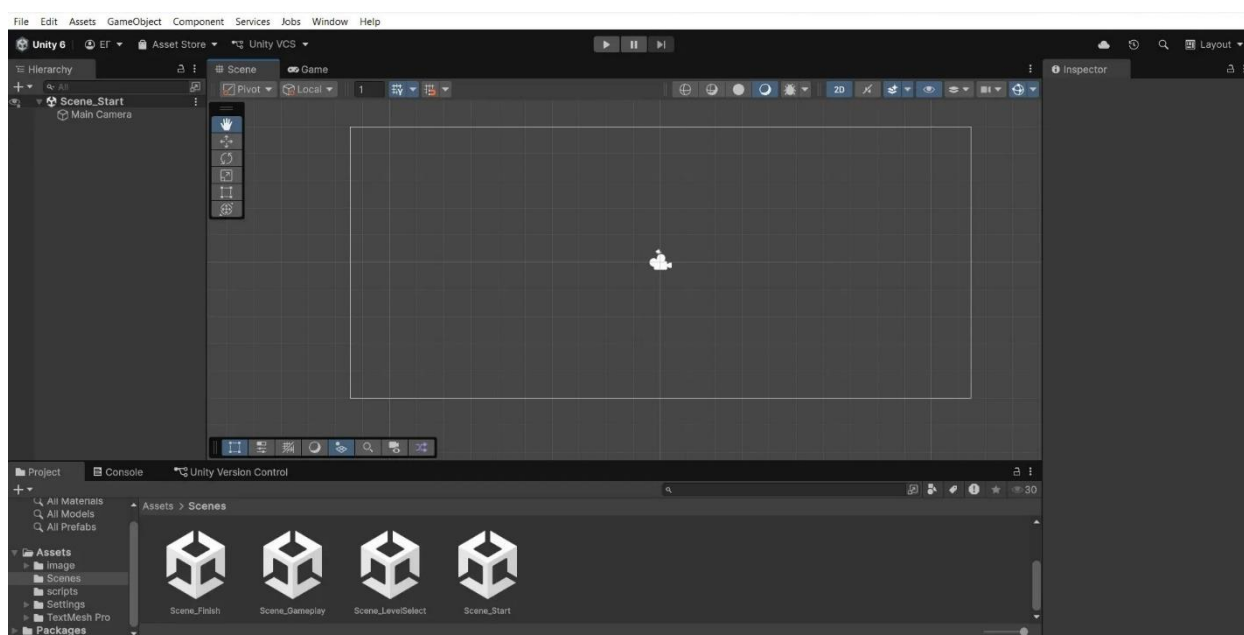


Рисунок 3 – Создание сцен игры

В папку image необходимо добавить заранее подготовленные изображения для игры (см. рис. 4).

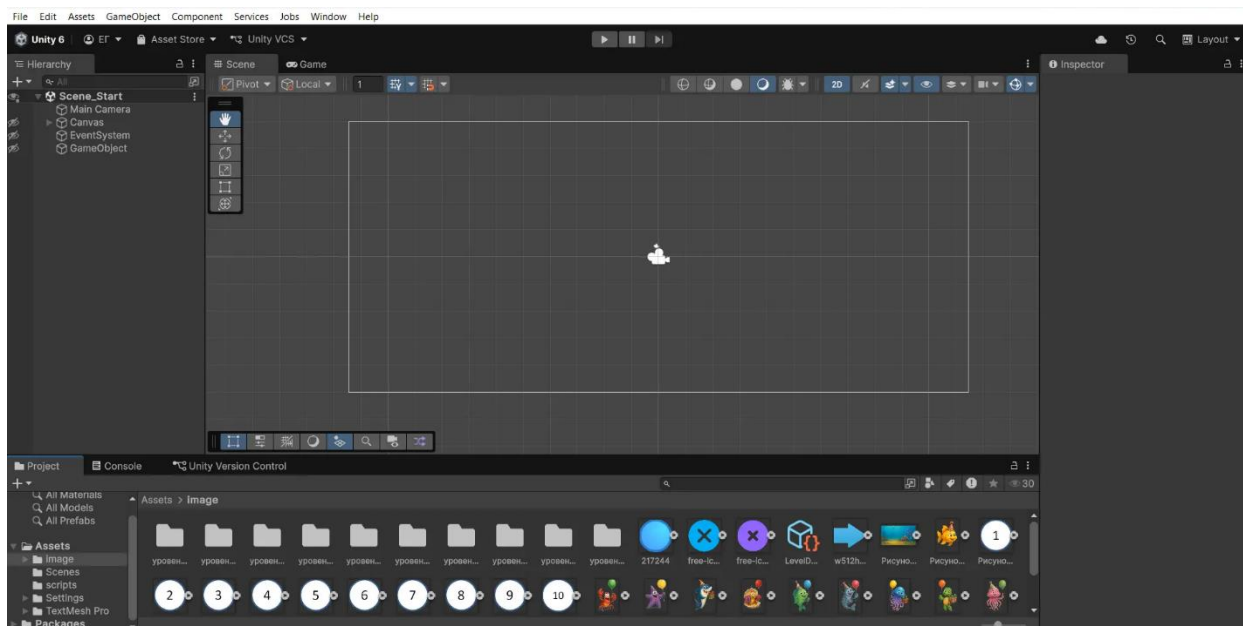


Рисунок 4 – Добавление изображений

Следующим шагом необходимо открыть сцену Scene_Start. Данная сцена содержит название игры, обращение персонажа к игроку, правила игры и кнопку начать игру. Для начала необходимо создать объект Canvas который служит основой для создания пользовательского интерфейса. В Hierarchy необходимо нажать правой кнопкой мыши для вызова меню в пространстве иерархии и выбрать UI -> Canvas. Для того, чтобы интерфейс работал корректно Canvas необходимо настроить. В разделе inspector необходимо выбрать следующие пункты: в Render Mode выбрать Screen Space – Camera, в Render Camera перенести объект камеры из Hierarchy, в Order in Layer ввести 1, а в Reference Resolution установить размеры 1920 на 1080 (см. рис. 5).

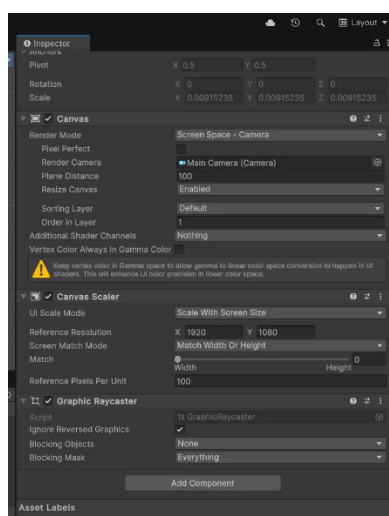


Рисунок 5 – Настройка Canvas

В Canvas требуется добавить объект image для фона игры. После этого в инспекторе необходимо настроить позицию, масштаб и выравнивание так, чтобы он полностью покрывал область, которую видит камера, а также в разделе image добавить изображение подготовленного фона (см. рис. 6).

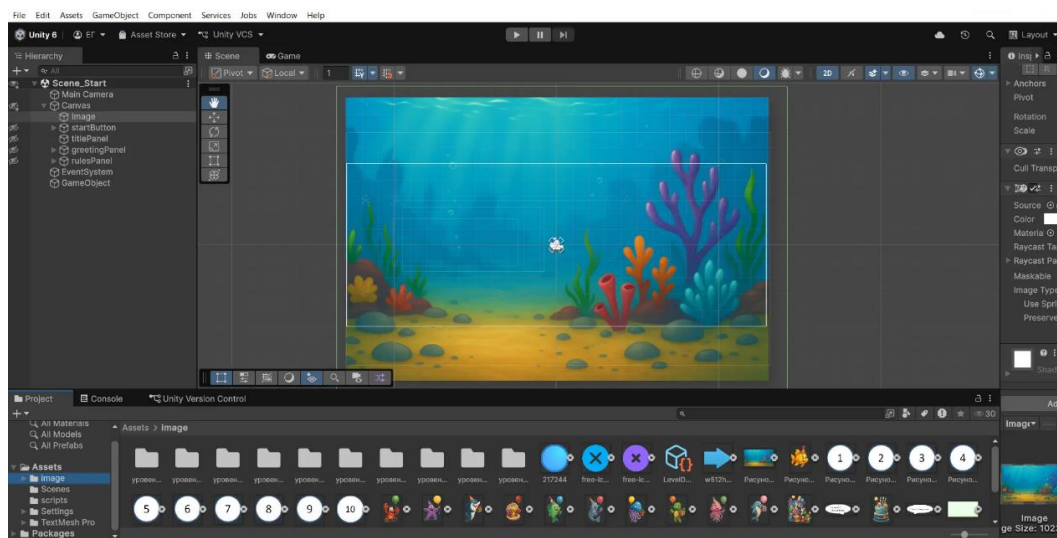


Рисунок 6 – Добавление фона

После создания фона, необходимо создать три панели, для этого в Hierarchy необходимо нажать правой кнопкой мыши для вызова меню в пространстве иерархии и выбрать UI -> Panel, и дать название панелям titlePanel, greetingPanel и rulesPanel. В каждую панель необходимо добавить подготовленные изображения, в titlePanel добавляется изображение с названием игры, в greetingPanel изображение с обращением рыбки к игроку, а rulesPanel изображение с правилами игры, а также в данную панель необходимо добавить кнопку через меню UI -> Legacy-> Button, для кнопки начать игру (см. рис. 7).

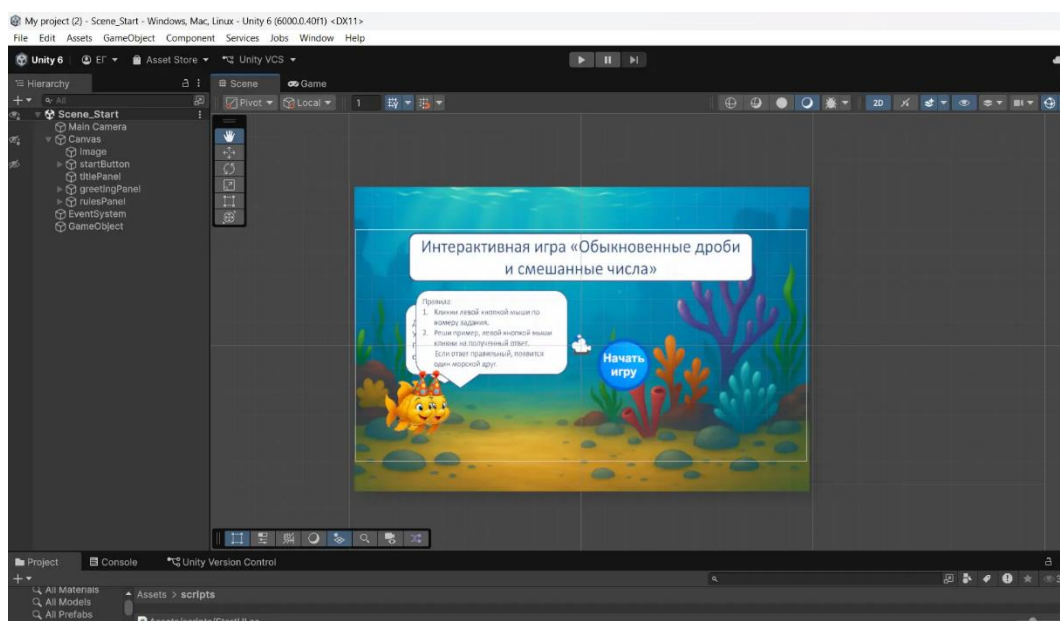


Рисунок 7 – Настройка сцены

После настройки сцены необходимо создать скрипты StartUI и GameManager. Скрипт StartUI управляет экранной последовательностью на сцене, а GameManager управляет глобальным состоянием игры, которое должно сохраняться между сценами.

Скрипт StartUI:

```
using UnityEngine;
using UnityEngine.UI;
public class StartUI : MonoBehaviour
{
    public GameObject titlePanel;
    public GameObject greetingPanel;
    public GameObject rulesPanel;
    public Button startButton;
    private bool titleShown = true;
    void Start()
    {
        titlePanel.SetActive(true);
        greetingPanel.SetActive(false);
        rulesPanel.SetActive(false);
        startButton.gameObject.SetActive(false);
    }
    public void OnScreenClick()
    {
        if (titleShown)
        {
            titlePanel.SetActive(false);
            greetingPanel.SetActive(true);
            titleShown = false;
        }
    }
    public void OnGreetingNext()
    {
        greetingPanel.SetActive(false);
        rulesPanel.SetActive(true);
        startButton.gameObject.SetActive(true);
    }
    public void OnStartButtonClick()
    {
        GameManager.Instance.StartGame();
    }
}
```

Скрипт GameManager:

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class GameManager : MonoBehaviour
{
    public static GameManager Instance;
    [Header("Progress")]
    public bool[] levelCompleted = new bool[10];
    private int selectedLevelIndex = -1;
    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            Debug.Log("GameManager инициализирован и сохранён.");
        }
        else
        {
            Destroy(gameObject);
            Debug.Log("Лишний GameManager уничтожен.");
        }
    }
    public static void EnsureInstance()
```

```
{
    if (Instance == null)
    {
        GameObject go = new GameObject("GameManager");
        Instance = go.AddComponent<GameManager>();
        DontDestroyOnLoad(go);
        Debug.Log("GameManager создан автоматически.");
    }
}

public void StartGame()
{
    Debug.Log("StartGame -> загружаем Scene_LevelSelect");
    SceneManager.LoadScene("Scene_LevelSelect");
}

public int GetCurrentLevel()
{
    for (int i = 0; i < levelCompleted.Length; i++)
    {
        if (!levelCompleted[i])
        {
            Debug.Log($"GetCurrentLevel: возвращаем {i} (уровень {i + 1})");
            return i;
        }
    }
    Debug.Log("GetCurrentLevel: все уровни пройдены, возвращаем 10");
    return levelCompleted.Length;
}

public bool IsLevelCompleted(int index)
{
    if (index < 0 || index >= levelCompleted.Length) return false;
    return levelCompleted[index];
}

public bool CanPlayLevel(int index)
{
    bool result = index == GetCurrentLevel();
    Debug.Log($"CanPlayLevel({index}) = {result}");
    return result;
}

public void LoadLevel(int index)
{
    Debug.Log($"LoadLevel: попытка загрузить уровень {index + 1}");
    if (CanPlayLevel(index))
    {
        selectedLevelIndex = index;
        Debug.Log($"LoadLevel: загружаем Scene_Gameplay,
selectedLevelIndex={selectedLevelIndex}");
        SceneManager.LoadScene("Scene_Gameplay");
    }
    else
    {
        Debug.Log($"LoadLevel: уровень {index + 1} не доступен.");
    }
}

public int GetSelectedLevel()
{
    return selectedLevelIndex;
}

public void CompleteLevel()
{
    if (selectedLevelIndex >= 0 && selectedLevelIndex < levelCompleted.Length)
    {
        levelCompleted[selectedLevelIndex] = true;
        Debug.Log($"CompleteLevel: уровень {selectedLevelIndex + 1} отмечен
пройденным.");
        selectedLevelIndex = -1;
    }
    if (GetCurrentLevel() >= levelCompleted.Length)
    {
        Debug.Log("CompleteLevel: все уровни пройдены, загружаем Scene_Finish");
        SceneManager.LoadScene("Scene_Finish");
    }
}
```

```

    else
    {
        Debug.Log("CompleteLevel: возвращаемся на Scene_LevelSelect");
        SceneManager.LoadScene("Scene_LevelSelect");
    }
}

public void GoToLevelSelect()
{
    Debug.Log("GoToLevelSelect");
    selectedLevelIndex = -1;
    SceneManager.LoadScene("Scene_LevelSelect");
}

public void ResetProgress()
{
    for (int i = 0; i < levelCompleted.Length; i++) levelCompleted[i] = false;
    selectedLevelIndex = -1;
    Debug.Log("Progress reset");
    SceneManager.LoadScene("Scene_LevelSelect");
}
}
}

```

Чтобы механика игры работала, необходимо создать пустой GameObject и добавить к нему скрипты StartUI и GameManager, и настроить в инспекторе (см. рис. 8).

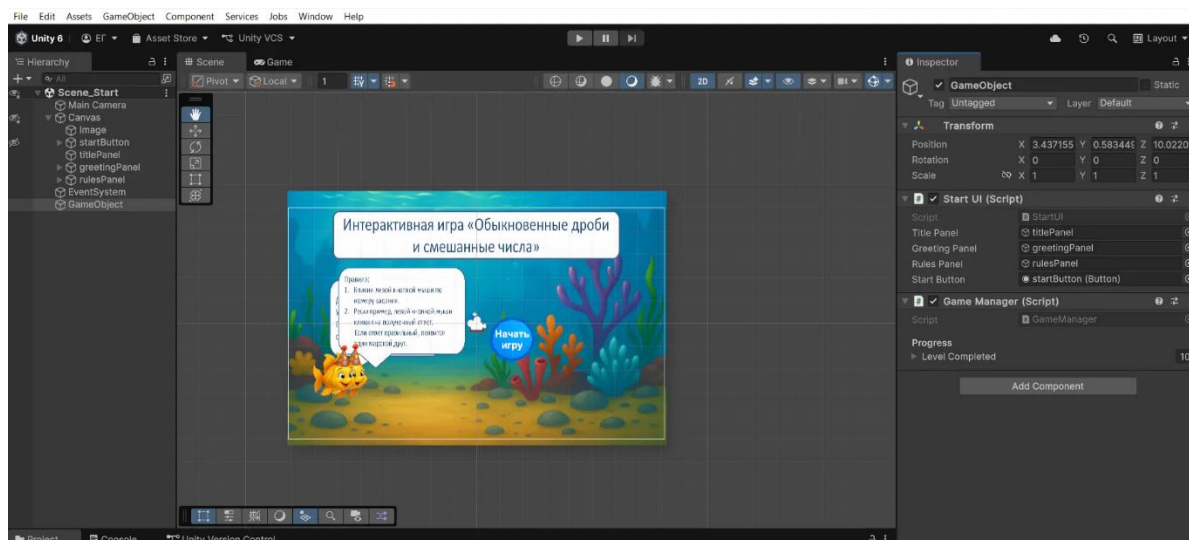


Рисунок 8 – Настройка GameObject

Для того, чтобы при нажатии на экран панели появлялись по очереди, необходимо в панель titlePanel и greetingPanel добавить компонент Button в инспекторе и настроить On Click(). В каждой панели в On Click() необходимо перенести объект GameObject в поле None (Object), но настройка в выпадающем меню справа для каждой панели разная. Для панели titlePanel необходимо выбрать StartUI -> OnScreenClick(), для greetingPanel StartUI -> OnGreetingNext(). Для панели rulesPanel не предназначены эти настройки, так как ранее создана кнопка «Начать игру», в инспекторе кнопки необходимо настроить On Click() для этого надо перенести объект GameObject в поле None (Object) и выбрать StartUI -> OnStartButtonClick().

После завершения настроек первой сцены, надо открыть сцену с уровнями Scene_LevelSelect. На сцене необходимо произвести такие же настройки Canvas, добавить фон. Игра содержит 10 уровней. Необходимо в

Hierarchy создать 10 кнопок, установить размеры 150 на 150, и к каждой кнопке добавить уже заранее подготовленные изображения каждого уровня. После настроек кнопок расположить их по всей сцене змейкой (см. рис. 9).

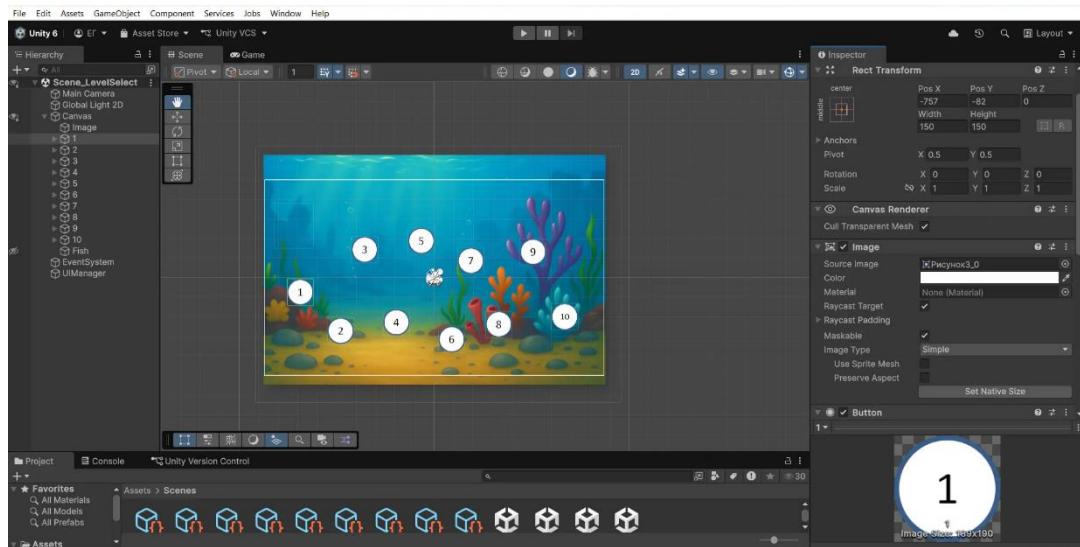


Рисунок 9 – Настройка сцены

На сцену также надо добавить персонажа рыбку, для того чтобы перемещался по уровням. Для того чтобы рыбка перемещалась необходимо убрать галочку возле Raycast Target (см. рис. 10).

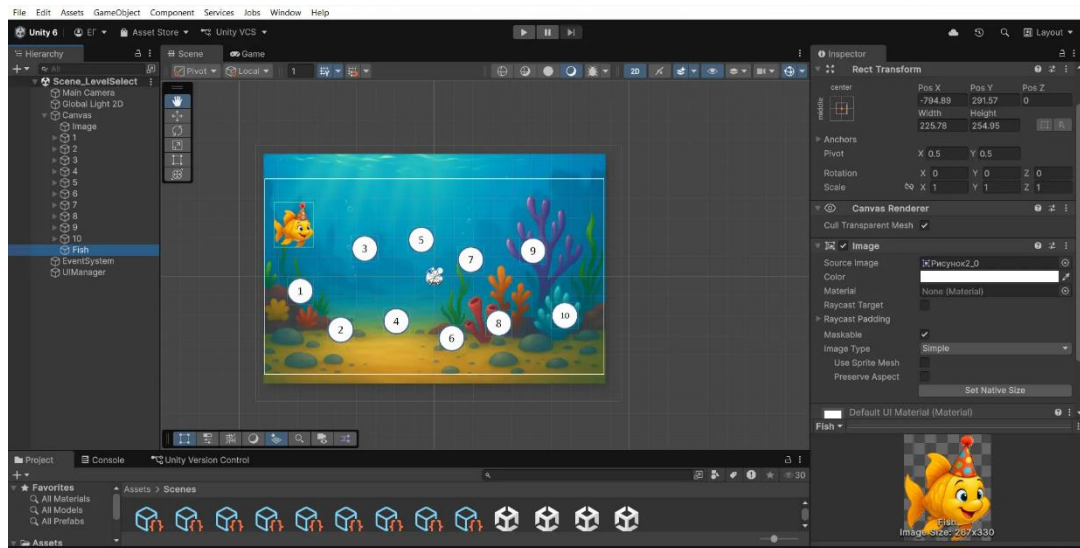


Рисунок 10 – Добавление и настройка объекта

Далее необходимо приступить к настройке механики сцены. Для этого необходимо создать скрипт LevelSelectUI который отвечает за клики по кнопкам, перемещение рыбки и синхронизацию с прогрессом игрока.

Скрипт LevelSelectUI:

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
public class LevelSelectUI : MonoBehaviour
```

```

{
    public Button[] levelButtons;
    public RectTransform fish;
    public float moveDuration = 0.5f;
    private void Start()
    {
        GameManager.EnsureInstance();
        if (levelButtons == null || levelButtons.Length == 0)
        {
            Debug.LogError("LevelSelectUI: levelButtons not assigned!");
            return;
        }
        if (fish == null)
        {
            Debug.LogError("LevelSelectUI: fish not assigned!");
            return;
        }
        for (int i = 0; i < levelButtons.Length; i++)
        {
            int index = i;
            if (levelButtons[i] != null)
            {
                levelButtons[i].onClick.AddListener(() => OnLevelClick(index));
                Debug.Log($"Кнопка {i + 1} подписана на клик.");
            }
        }
        MoveFishToCurrentLevel();
    }
    private void MoveFishToCurrentLevel()
    {
        if (GameManager.Instance == null)
        {
            Debug.LogError("GameManager.Instance is null!");
            return;
        }
        int current = GameManager.Instance.GetCurrentLevel();
        Debug.Log($"MoveFish: текущий уровень = {current}");
        if (current >= 0 && current < levelButtons.Length && levelButtons[current] !=
null)
        {
            Vector3 target = levelButtons[current].transform.position;
            StartCoroutine(SmoothMove(fish, target, moveDuration));
        }
        else
        {
            Debug.LogWarning($"Некорректный индекс текущего уровня: {current}");
        }
    }
    private IEnumerator SmoothMove(RectTransform obj, Vector3 target, float duration)
    {
        if (obj == null) yield break;
        Vector3 start = obj.position;
        float t = 0f;
        while (t < 1f)
        {
            t += Time.deltaTime / duration;
            obj.position = Vector3.Lerp(start, target, t);
            yield return null;
        }
        obj.position = target;
    }
    private void OnLevelClick(int index)
    {
        Debug.Log($"OnLevelClick вызван для уровня {index + 1}");
        if (GameManager.Instance == null)
        {
            Debug.LogError("GameManager.Instance is null!");
            return;
        }
        bool canPlay = GameManager.Instance.CanPlayLevel(index);
        Debug.Log($"CanPlayLevel({index}) = {canPlay}");
    }
}

```

```

        if (canPlay)
        {
            GameManager.Instance.LoadLevel(index);
        }
        else
        {
            Debug.Log($"Уровень {index + 1} недоступен. Текущий уровень:
{GameManager.Instance.GetCurrentLevel() + 1}");
        }
    }
}

```

Для механики, необходимо создать пустой объект и добавить скрипт LevelSelectUI. В инспекторе UIManager необходимо переместить кнопки по порядку в Level Buttons (см. рис. 11).

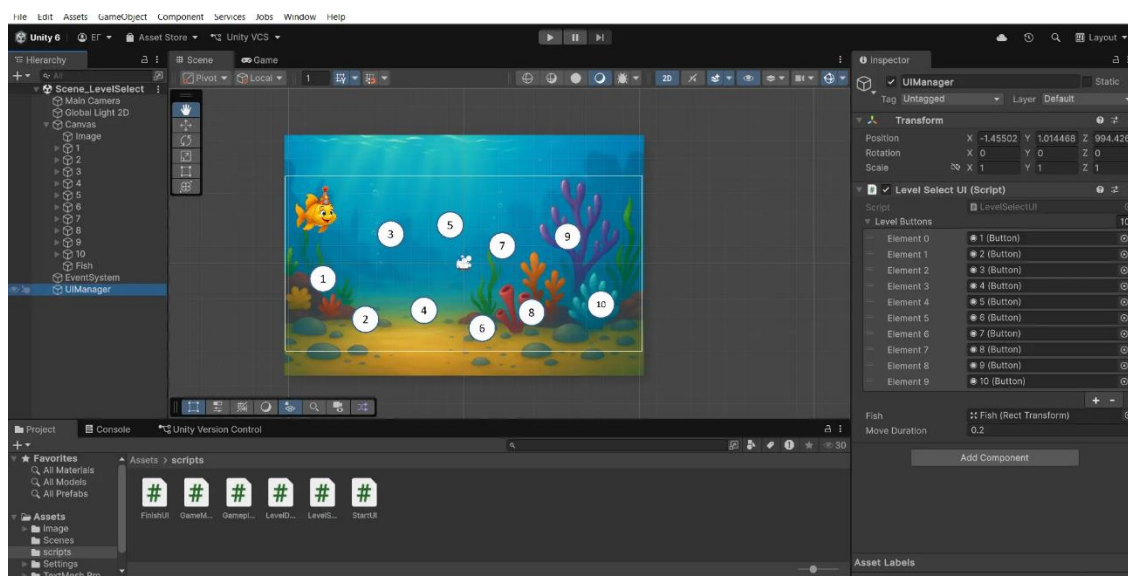


Рисунок 11 – Настройка механики сцены

Вторая сцена полностью готова, можно приступать к настройке сцены игры с заданиями. Canvas также настраивается как и на предыдущих сценах, но фон белый. Для визуализации задания необходимо добавить объект image и переименовать в QuestionImage и указать размеры 1352,92 на 150. Для визуализации ответов на вопрос необходимо создать две кнопки и дать им названия AnswerButton_0 и AnswerButton_1, указать размеры 300 на 200, а также в кнопках создать объект изображения AnswerImage_0 и AnswerImage_1, указать такие же размеры, как и у кнопок. В изображения QuestionImage, AnswerImage_0 и AnswerImage_1 добавить подготовленные изображения. При правильном ответе должна плавно появляться морской друг, для этого добавляем 10 изображений морских друзей с скрываем их, чтобы их не было видно. Также необходимо добавить кнопку для выхода на сцену с уровнями (см. рис. 12).

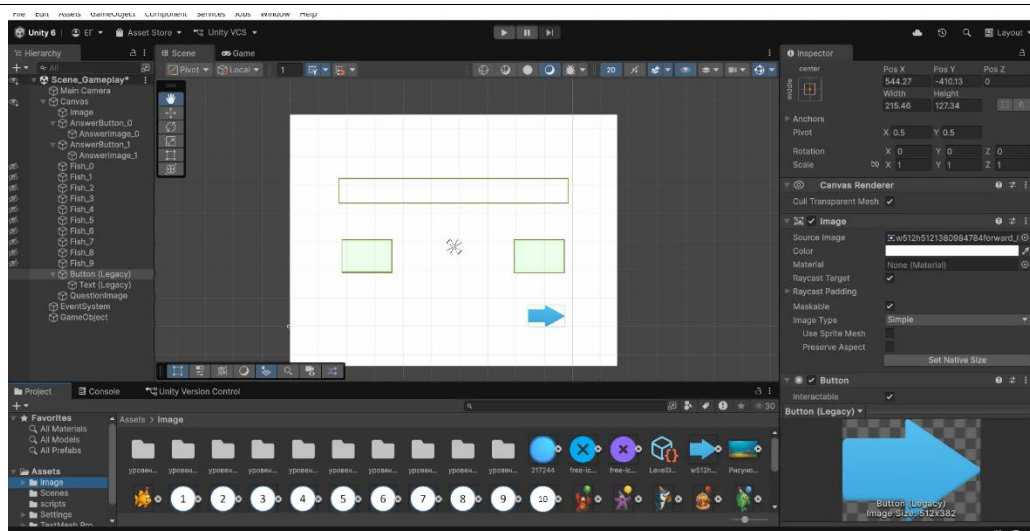


Рисунок 12 – Настройка сцены

Для того, чтобы в созданных изображения появлялись задания и ответы необходимо создать скрипт LevelData, который отвечает за уровни игры.

Скрипт LevelData:

using UnityEngine;

```
[CreateAssetMenu(fileName = "LevelData", menuName = "Game/Level Data")]
public class LevelData : ScriptableObject
{
    public Sprite questionImage;    // картинка с заданием
    public Sprite answer0;         // первый вариант ответа
    public Sprite answer1;         // второй вариант ответа
    public int correctAnswerIndex; // 0 или 1
}
```

Далее создаются уровни игры. Для этого в окне Project необходимо нажать правой кнопкой далее выбрать Create перейти в Game, выбрать Level Data. Необходимо создать 10 уровней. В инспекторе уровня перетащить картинку задания в поле Question Image, картинки ответов в Answer 0 и Answer 1, и указать правильный ответ 0 или 1 (см. рис. 13).

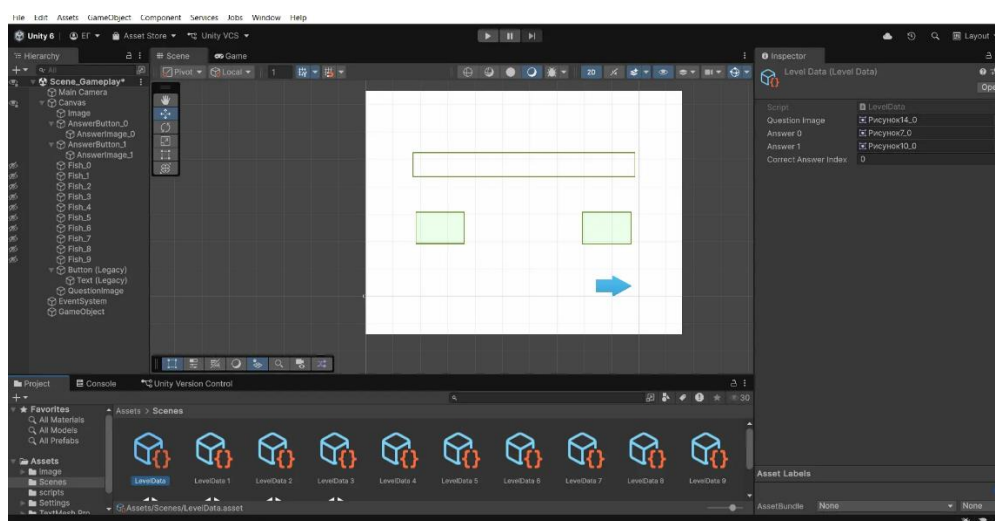


Рисунок 13 – Настройка уровней игры

Следующим шагом необходимо создать механику сцены. Для этого необходимо создать скрипт `GamePlayUI` который отвечает за управление всем, что происходит во время решения примера: отображение вопроса, варианты ответа, проверка правильности, обратная связь и переходы.

Скрипт `GamePlayUI`:

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;

public class GameplayUI : MonoBehaviour
{
    [Header("UI Elements")]
    public Image questionImage;           // вместо TMP_Text - Image для вопроса
    public Button[] answerButtons;        // две кнопки
    public Image[] answerImages;          // изображения внутри кнопок
    public Button backButton;

    [Header("Level Data")]
    public LevelData[] levelDataArray; // массив из 10 LevelData (в инспекторе
// заполняем)

    [Header("Fish Collection")]
    public GameObject[] fishObjects;
    public float delayBeforeNext = 1.5f;
    public float fishAppearDuration = 0.6f;

    private int levelIndex;
    private bool isAnswered = false;
    private Vector3[] fishTargetPositions;

    void Start()
    {
        GameManager.EnsureInstance();

        levelIndex = GameManager.Instance.GetSelectedLevel();
        if (levelIndex < 0 || levelIndex >= levelDataArray.Length)
        {
            Debug.LogError($"Некорректный индекс уровня: {levelIndex}. Возврат на
выбор уровней.");
            GameManager.Instance.GoToLevelSelect();
            return;
        }
        LevelData currentLevel = levelDataArray[levelIndex];

        if (questionImage != null && currentLevel.questionImage != null)
            questionImage.sprite = currentLevel.questionImage;
        else
            Debug.LogWarning("Не назначено изображение для вопроса уровня " +
(levelIndex+1));

        for (int i = 0; i < answerButtons.Length && i < 2; i++)
        {
            if (answerImages[i] != null)
            {
                Sprite answerSprite = (i == 0) ? currentLevel.answer0 :
currentLevel.answer1;
                answerImages[i].sprite = answerSprite;
            }
            int idx = i;
            answerButtons[i].onClick.RemoveAllListeners();
            answerButtons[i].onClick.AddListener(() => OnAnswerClick(idx));
        }

        backButton.onClick.RemoveAllListeners();
        backButton.onClick.AddListener(() => GameManager.Instance.GoToLevelSelect());
    }
}
```

```

        fishTargetPositions = new Vector3[fishObjects.Length];
        for (int i = 0; i < fishObjects.Length; i++)
        {
            if (fishObjects[i] != null)
            {
                fishTargetPositions[i] = fishObjects[i].transform.position;
                Vector3 startPos = fishTargetPositions[i];
                startPos.y -= 500f;
                fishObjects[i].transform.position = startPos;
                fishObjects[i].SetActive(false);
            }
        }
    }
    void OnAnswerClick(int selected)
    {
        if (isAnswered) return;

        int correct = levelDataArray[levelIndex].correctAnswerIndex;
        if (selected == correct)
        {
            isAnswered = true;
            if (levelIndex >= 0 && levelIndex < fishObjects.Length &&
fishObjects[levelIndex] != null)
                StartCoroutine>ShowFishWithAnimation(levelIndex));
            else
                StartCoroutine(CompleteLevelWithDelay(0f));

            foreach (var btn in answerButtons)
                btn.interactable = false;
        }
        else
        {
            Debug.Log("Неверно, попробуйте другой вариант.");
        }
    }
    IEnumerator ShowFishWithAnimation(int fishIndex)
    {
        GameObject fish = fishObjects[fishIndex];
        if (fish == null) yield break;

        fish.SetActive(true);
        Vector3 startPos = fish.transform.position;
        Vector3 targetPos = fishTargetPositions[fishIndex];
        float elapsed = 0f;
        while (elapsed < fishAppearDuration)
        {
            elapsed += Time.deltaTime;
            float t = elapsed / fishAppearDuration;
            t = t * t * (3f - 2f * t);
            fish.transform.position = Vector3.Lerp(startPos, targetPos, t);
            yield return null;
        }
        fish.transform.position = targetPos;
        yield return new WaitForSeconds(delayBeforeNext);
        GameManager.Instance.CompleteLevel();
    }
    IEnumerator CompleteLevelWithDelay(float delay)
    {
        yield return new WaitForSeconds(delay);
        GameManager.Instance.CompleteLevel();
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.LeftArrow))
            GameManager.Instance.GoToLevelSelect();
    }
}

```

Следующим шагом создается пустой объект и к нему прикрепляется скрипт `GamePlayUI`, и в инспекторе добавляются кнопки и изображения для

вывода ответов, изображение для вывода задания, кнопка для выхода на сцену уровней, созданные уровни прикрепляются по очереди, также прикрепляются и изображение рыбок к каждому уровню (см. рис. 14).

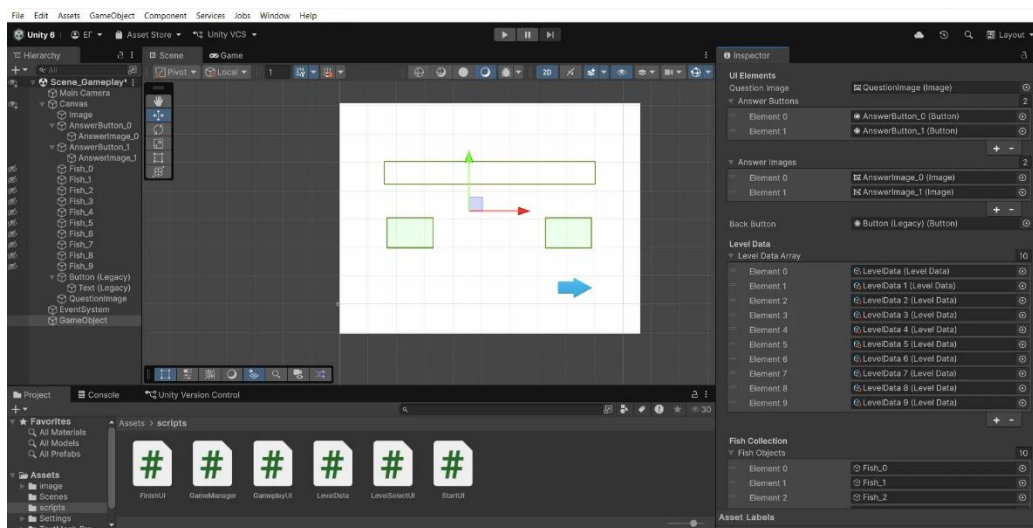


Рисунок 14 – Настройка уровней игры

Сцена с заданиями готова, завершающим шагом является настройка сцены завершения игры. Для этого, необходимо настроить фон как для первой сцены. Создать четыре рисунка в Hierarchy, и к каждому добавить компонент Canvas Group. Разместить изображения на сцене. Так же в Hierarchy добавить объект панель. В объект панель добавить две кнопки сыграть снова и выход (см. рис. 15).



Рисунок 15 – Настройка сцены

Далее необходимо создать скрипт FinishUI, для механики сцены.

Скрипт FinishUI:

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
public class FinishUI : MonoBehaviour
{
```

```

[Header("Элементы для последовательного появления")]
public CanvasGroup[] objectsToAppear
public float appearDuration = 0.8f
public float delayBetween = 0.5f;
public GameObject buttonsPanel;
public float panelDelay = 3f;
[Header("Кнопки")]
public Button replayButton;
public Button quitButton;
void Start()
{
    if (objectsToAppear != null && objectsToAppear.Length > 0)
    {
        foreach (var cg in objectsToAppear)
        {
            if (cg != null)
            {
                cg.gameObject.SetActive(true);
                cg.alpha = 0f;
                cg.interactable = false;
                cg.blocksRaycasts = false;
            }
        }
        StartCoroutine(AppearSequence());
    }
    else
    {
        Debug.LogWarning("FinishUI: нет объектов для последовательного
появления.");
    }
    if (buttonsPanel != null)
    {
        buttonsPanel.SetActive(false);
    }
    StartCoroutine>ShowButtonsPanelAfterDelay(panelDelay));
    if (replayButton != null)
        replayButton.onClick.AddListener(() =>
GameManager.Instance.ResetProgress());
    else
        Debug.LogWarning("Replay button not assigned!");
    if (quitButton != null)
        quitButton.onClick.AddListener(() => Application.Quit());
    else
        Debug.LogWarning("Quit button not assigned!");
}
IEnumerator AppearSequence()
{
    for (int i = 0; i < objectsToAppear.Length; i++)
    {
        CanvasGroup cg = objectsToAppear[i];
        if (cg == null) continue;
        Debug.Log($"Появляется объект {i + 1}: {cg.name
}");
        yield return StartCoroutine(FadeIn(cg, appearDuration));
        yield return new WaitForSeconds(delayBetween);
    }
}
IEnumerator FadeIn(CanvasGroup cg, float duration)
{
    float elapsed = 0f;
    while (elapsed < duration)
    {
        elapsed += Time.deltaTime;
        cg.alpha = Mathf.Lerp(0f, 1f, elapsed / duration);
        yield return null;
    }
    cg.alpha = 1f;
    cg.interactable = true;
}
IEnumerator ShowButtonsPanelAfterDelay(float delay)
{

```

```

yield return new WaitForSeconds(delay);
if (buttonsPanel != null)
{
    buttonsPanel.SetActive(true);
    Debug.Log("Панель с кнопками появилась!");
}
else
{
    Debug.LogWarning("buttonsPanel не назначен в инспекторе!");
}
}
}

```

Для настройки механики игры необходимо создать пустой объект, и к нему добавить скрипт FinishUI. В инспекторе добавить изображения, чтобы они плавно появлялись, а также добавить панель и кнопки (см. рис. 16).

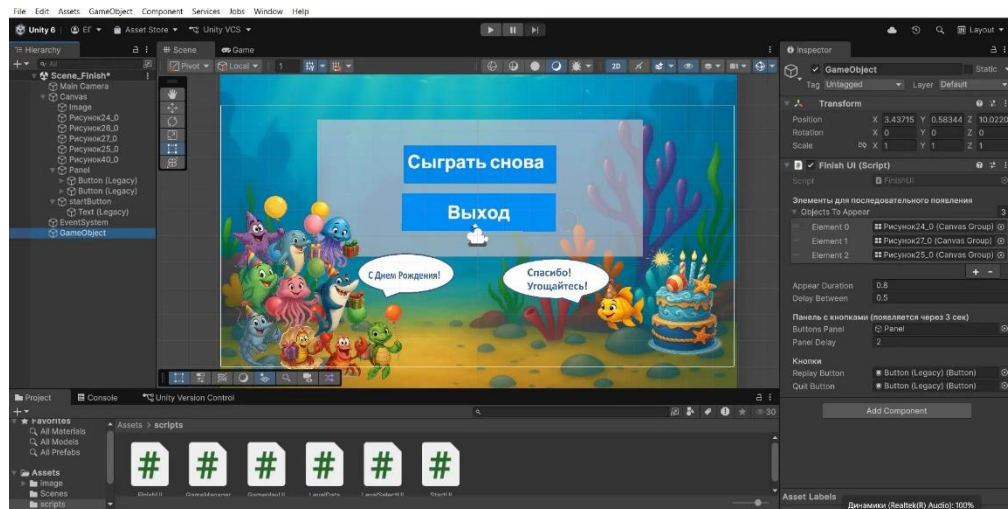


Рисунок 16 – Настройка сцены финиш

Готовая компьютерная игра (см. рис. 17.1, 17.2, 17.3, 17.4, 17.5).

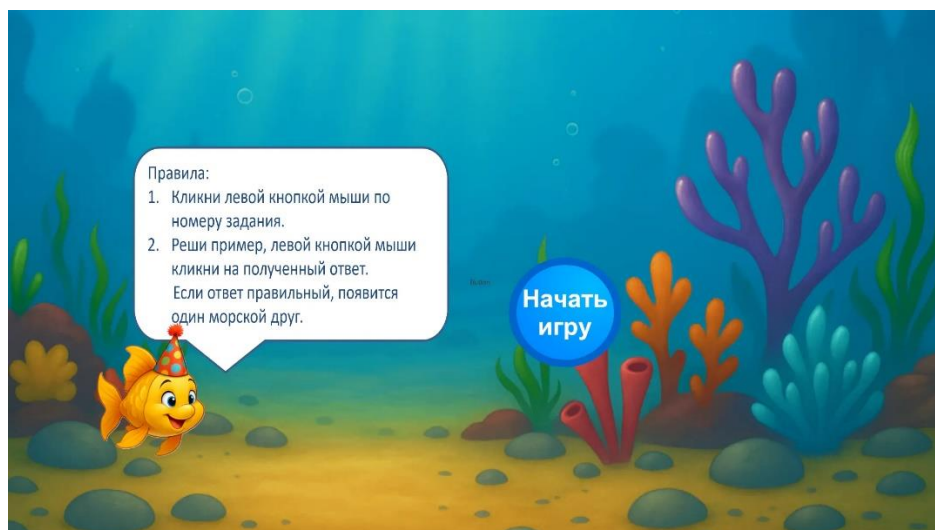


Рисунок 17.1 – Готовая игра

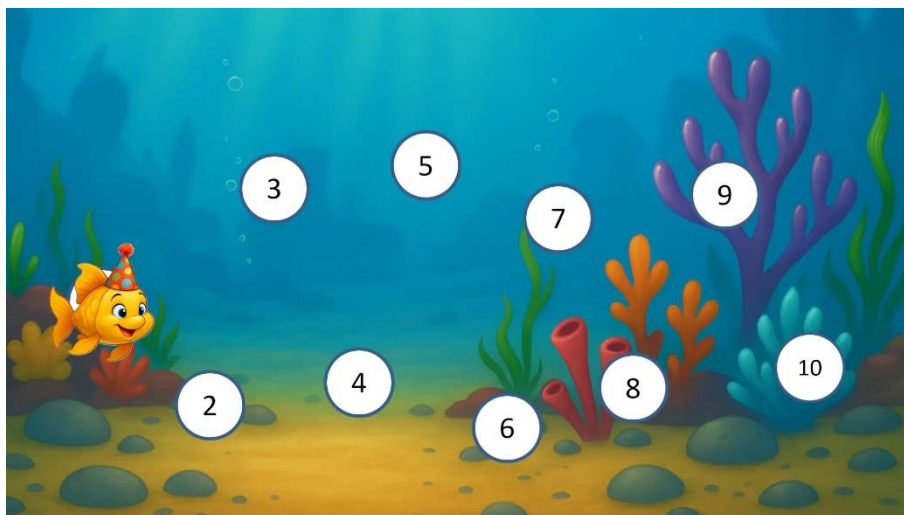


Рисунок 17.2 – Готовая игра

Переведи неправильную дробь в смешанное число: $\frac{48}{7}$.



Рисунок 17.3 – Готовая игра



Рисунок 17.4 – Готовая игра

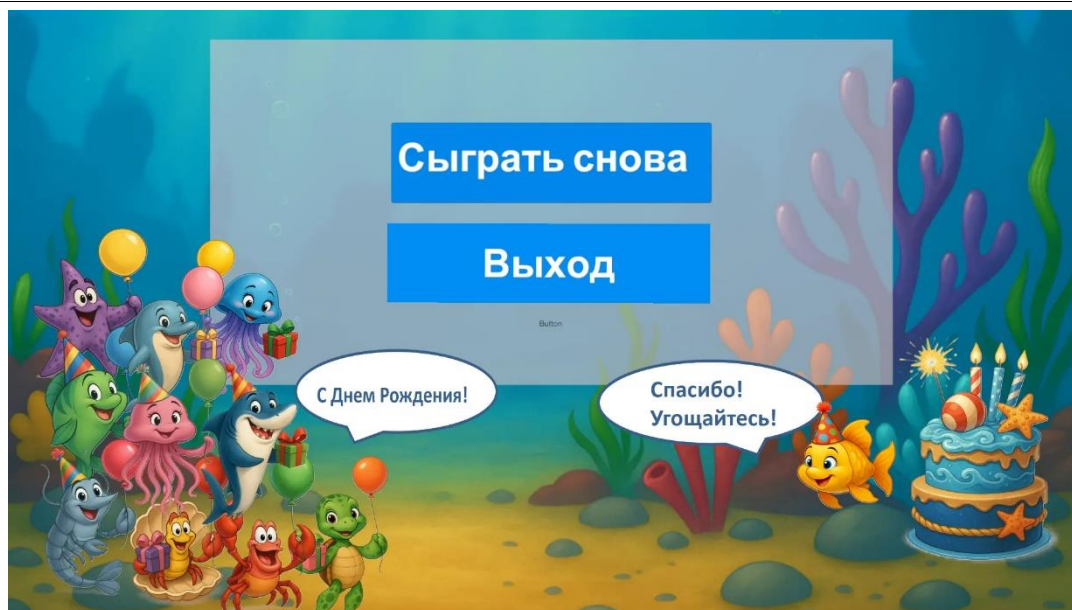


Рисунок 17.5 – Готовая игра

4. Выводы

В данной работе показан процесс создания компьютерной игры «Обыкновенные дроби и смешанные числа» с помощью кроссплатформенной среды для создания интерактивных 2D и 3D приложений. 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. Волгоград: Общество с ограниченной ответственностью "Сфера", 2024. С. 22-25. URL: <https://www.elibrary.ru/item.asp?id=68624300>
2. Фаст Я. А. Разработка 2D игр с элементами головоломок на платформе Unity // V Международная научная конференция по междисциплинарным исследованиям. Екатеринбург: Общество с ограниченной ответственностью "Институт Цифровой Экономики и Права", 2024. С. 266-269. URL: <https://www.elibrary.ru/item.asp?id=67143462>
3. Максимов Н. В. Разработка мобильной 2D игры "Bootur Jump" на Unity // Фундаментальные и прикладные аспекты развития современной науки. Уфа: Общество с ограниченной ответственностью "Научно-издательский центр "Вестник науки", 2023. С. 306-308. URL: <https://www.elibrary.ru/item.asp?id=59762421>
4. Любич А. А. Создание 2D-игры «б-у-у-м» на движке Unity // Фундаментальные и прикладные исследования молодых ученых. Новокузнецк: Кемеровский государственный университет, 2025. С. 167-169. URL: <https://www.elibrary.ru/item.asp?id=82653081>