Приложение А

Листинг программы

AppQuit.cs (скрипт выхода из игры)

```
using UnityEngine;

public class AppQuit : MonoBehaviour
{
    public void QuitApplication()
    {
        // Выход из приложения
        Application.Quit();

        // Для отладки в редакторе
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
#endif
    }
}
```

ButtonToggle.cs(скрипт для кнопки режима в игровом меню)

```
using UnityEngine;
public class ButtonToggle : MonoBehaviour
{
    public GameObject btn1; // Кнопка Уровень 1 (всегда видна)
    public GameObject btn2; // Кнопка Уровень 2
    public GameObject btn3; // Кнопка Уровень 3
    public GameObject btn4; // Кнопка Уровень 4

    public GameObject subA; // Подуровень A
    public GameObject subB; // Подуровень B

    bool showSub = false;

    void Start()
    {
        // На старте: основные видны, подуровни скрыты
        SetActive(btn2, true);
        SetActive(btn3, true);
        SetActive(btn4, true);
        SetActive(subA, false);
        SetActive(subB, false);
    }
    // Вызывается при клике на btn1
    public void Toggle()
    {
        showSub = !showSub;

        if (showSub)
        {
            // Показываем подуровни, скрываем 2,3,4
            SetActive(btn2, false);
            SetActive(btn3, false);
            SetActive(btn4, false);
            SetActive(subA, true);
            SetActive(subB, true);
        }
        else
        {
            // Показываем 2,3,4, скрываем подуровни
            SetActive(btn2, true);
            SetActive(btn3, true);
            SetActive(btn4, true);
            SetActive(subA, false);
            SetActive(subB, false);
        }
    }
    void SetActive(GameObject obj, bool active)
    {
        if (obj != null)
            obj.SetActive(active);
    }
}
```

SimpleButtonAnimation(скрипт для анимации появления подуровней)

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

public class SimpleButtonAnimation : MonoBehaviour
{
    public Button mainButton;
    public RectTransform[] movableButtons;
    public float moveDistance = 700f;
    public float animationTime = 0.5f;

    private bool buttonsHidden = false;
    private Vector2[] startPositions;
    private CanvasGroup[] canvasGroups;

    void Start()
    {
        // Проверка на null чтобы избежать ошибок
        if (mainButton == null)
        {
            Debug.LogError("Main Button not assigned in inspector!");
            return;
        }

        if (movableButtons == null || movableButtons.Length == 0)
        {
            Debug.LogError("Movable Buttons array is empty!");
            return;
        }
        // Сохраняем стартовые позиции и настраиваем CanvasGroup
        startPositions = new Vector2[movableButtons.Length];
        canvasGroups = new CanvasGroup[movableButtons.Length];

        for (int i = 0; i < movableButtons.Length; i++)
        {
            if (movableButtons[i] != null)
            {
                startPositions[i] = movableButtons[i].anchoredPosition;

                // Добавляем или получаем компонент CanvasGroup для прозрачности
                canvasGroups[i] = movableButtons[i].GetComponent<CanvasGroup>();
                if (canvasGroups[i] == null)
                {
```

```
            canvasGroups[i] = movableButtons[i].gameObject.AddComponent<CanvasGroup>();
        }
      }
    }

    mainButton.onClick.AddListener(ToggleButtons);
  }

  void ToggleButtons()
  {
    if (buttonsHidden)
      StartCoroutine(ShowButtons());
    else
      StartCoroutine(HideButtons());

    buttonsHidden = !buttonsHidden;
  }

  IEnumerator HideButtons()
  {
    for (int i = 0; i < movableButtons.Length; i++)
    {
      if (movableButtons[i] == null) continue;

      Vector2 startPos = movableButtons[i].anchoredPosition;
      Vector2 endPos = startPos + Vector2.down * moveDistance; //

      float startAlpha = 1f;
      float endAlpha = 0f; // Полная прозрачность

      float timer = 0f;
      while (timer < animationTime)
      {
        timer += Time.deltaTime;
        float progress = timer / animationTime;

        // Двигаем позицию
        movableButtons[i].anchoredPosition = Vector2.Lerp(startPos, endPos, progress);

        // Меняем прозрачность
        if (canvasGroups[i] != null)
        {
          canvasGroups[i].alpha = Mathf.Lerp(startAlpha, endAlpha, progress);
        }

        yield return null;
      }
      // Гарантируем конечные значения
      movableButtons[i].anchoredPosition = endPos;
      if (canvasGroups[i] != null)
      {
        canvasGroups[i].alpha = endAlpha;
      }
    }
  }

  IEnumerator ShowButtons()
  {
    for (int i = 0; i < movableButtons.Length; i++)
    {
      if (movableButtons[i] == null) continue;

      Vector2 startPos = movableButtons[i].anchoredPosition;
      Vector2 endPos = startPositions[i];

      float startAlpha = 0f;
      float endAlpha = 1f; // Полная непрозрачность

      float timer = 0f;
      while (timer < animationTime)
      {
        timer += Time.deltaTime;
        float progress = timer / animationTime;

        // Двигаем позицию
        movableButtons[i].anchoredPosition = Vector2.Lerp(startPos, endPos, progress);

        // Меняем прозрачность
        if (canvasGroups[i] != null)
        {
          canvasGroups[i].alpha = Mathf.Lerp(startAlpha, endAlpha, progress);
        }

        yield return null;
      }

      // Гарантируем конечные значения
      movableButtons[i].anchoredPosition = endPos;
      if (canvasGroups[i] != null)
      {
        canvasGroups[i].alpha = endAlpha;
      }
    }
  }
}
```

Скрипт для подсказок

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

public class HintSystem : MonoBehaviour
{
  [Header("Кнопка подсказки")]
  public Button hintButton;      // Кнопка подсказки
  [Header("Настройки подсказок")]
  public int maxHints = 4;       // Максимум подсказок
  public float blinkTime = 1f;   // Время мигания
  public Color hintColor = Color.yellow; // Цвет подсказки

  [Header("Ссылка на иконки")]
  public SimpleIconGame iconGame; // Ссылка на основной скрипт
```

```
    private int hintsUsed = 0;      // Сколько подсказок использовано
    private bool isHintActive = false;
    private Coroutine hintCoroutine;
    private GameObject lastHintedIcon;
    void Start()
    {
        // Загружаем прогресс
        LoadProgress();

        // Настраиваем кнопку
        if (hintButton != null)
        {
            hintButton.onClick.AddListener(UseHint);
            UpdateHintButton();
        }
    }

    void LoadProgress()
    {
        // Загружаем использованные подсказки
        hintsUsed = PlayerPrefs.GetInt("HintsUsed", 0);
        Debug.Log($"Загружено подсказок: использовано {hintsUsed}/{maxHints}");
    }

    void SaveProgress()
    {
        // Сохраняем подсказки
        PlayerPrefs.SetInt("HintsUsed", hintsUsed);
        PlayerPrefs.Save();
    }

    // Использование подсказки
    public void UseHint()
    {
        // Проверяем, есть ли подсказки
        if (hintsUsed >= maxHints)
        {
            Debug.Log("Подсказки закончились!");
            return;
        }

        // Если нет ссылки на иконки
        if (iconGame == null)
        {
            Debug.LogError("Не назначен SimpleIconGame!");
            return;
        }
        // Получаем текущие видимые иконки
        List<GameObject> visibleIcons = GetVisibleIcons();

        if (visibleIcons.Count == 0)
        {
            Debug.Log("Нет видимых иконок для подсказки!");
            return;
        }

        // Выбираем случайную иконку
        GameObject randomIcon = visibleIcons[Random.Range(0, visibleIcons.Count)];

        // Запускаем мигание
        StartHintBlink(randomIcon);
        // Увеличиваем счетчик
        hintsUsed++;
        SaveProgress();
        // Обновляем кнопку
        UpdateHintButton();

        Debug.Log($"Использована подсказка {hintsUsed}/{maxHints}");
    }

    // Получить все видимые иконки
    List<GameObject> GetVisibleIcons()
    {
        List<GameObject> visible = new List<GameObject>();

        // Проверяем массив иконок из основного скрипта
        if (iconGame.allIcons != null)
        {
            foreach (GameObject icon in iconGame.allIcons)
            {
                if (icon != null && icon.activeSelf)
                {
                    visible.Add(icon);
                }
            }
        }
        return visible;
    }
    // Запустить мигание подсказки
    void StartHintBlink(GameObject icon)
    {
        // Останавливаем предыдущую подсказку
        StopCurrentHint();

        // Сохраняем ссылку на иконку
        lastHintedIcon = icon;

        // Запускаем мигание
        hintCoroutine = StartCoroutine(BlinkIcon(icon));
    }

    // Остановить текущую подсказку
    public void StopCurrentHint()
    {
        if (isHintActive && hintCoroutine != null)
        {
            StopCoroutine(hintCoroutine);
            isHintActive = false;

            // Возвращаем нормальный цвет
            if (lastHintedIcon != null)
            {
                Image img = lastHintedIcon.GetComponent<Image>();
                if (img != null) img.color = Color.white;
            }
```

```
        }
    }

    // Мигание иконки
    IEnumerator BlinkIcon(GameObject icon)
    {
        isHintActive = true;

        Image img = icon.GetComponent<Image>();
        if (img == null) yield break;

        Color originalColor = img.color;
        float timer = 0f;

        // Мигание
        while (isHintActive && timer < blinkTime)
        {
            timer += Time.deltaTime;

            // Пульсация
            float pulse = Mathf.PingPong(timer * 3f, 1f);
            Color pulseColor = Color.Lerp(hintColor,
originalColor, pulse);
            img.color = pulseColor;

            yield return null;
        }
        // Возвращаем оригинальный цвет
        if (img != null) img.color = originalColor;
        isHintActive = false;
        lastHintedIcon = null;
    }

    // Обновить вид кнопки
    void UpdateHintButton()
    {
        if (hintButton == null) return;

        // Текст кнопки
        Text buttonText =
hintButton.GetComponentInChildren<Text>();
        if (buttonText != null)
        {
            int remaining = maxHints - hintsUsed;
            buttonText.text = $"Подсказка ({remaining})";
        }

        // Цвет кнопки
        Image buttonImage =
hintButton.GetComponent<Image>();
        if (buttonImage != null)
        {
            if (hintsUsed >= maxHints)
            {
                // Подсказки закончились
                buttonImage.color = Color.red;
                hintButton.interactable = false;
            }
            else
            {
                // Есть подсказки
                buttonImage.color = Color.white;
                hintButton.interactable = true;
            }
        }
    }
    // Сброс подсказок
    public void ResetHints()
    {
        // Останавливаем текущую подсказку
        StopCurrentHint();

        // Сбрасываем счетчик
        hintsUsed = 0;

        // Удаляем сохранения
        PlayerPrefs.DeleteKey("HintsUsed");

        // Обновляем кнопку
        UpdateHintButton();

        Debug.Log("Подсказки сброшены!");
    }

    // Получить оставшиеся подсказки
    public int GetRemainingHints()
    {
        return maxHints - hintsUsed;
    }
}
```

Скрипт для организации игровой логики

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

public class IconGame : MonoBehaviour
{
    [Header("Все иконки (19 штук)")]
    public GameObject[] allIcons;

    [Header("Настройки игры")]
    public int iconsPerSet = 5;
    public GameObject nextLevelButton;

    [Header("Отладка")]
    public bool debugMode = true;

    private List<GameObject> currentSet = new
List<GameObject>();
    private List<GameObject> remainingIcons = new
List<GameObject>();
    private HashSet<int> collectedIds = new
HashSet<int>();
    void Start()
    {
        Log("=== ИГРА ЗАПУЩЕНА ===");

        // Скрываем все
        HideAllIcons();

        // Загружаем прогресс
        LoadProgress();
        Log($"Загружено собранных:
{collectedIds.Count}/19");
```

```
        // Создаем первый набор
        CreateNewSet();
    }

    void Log(string message)
    {
        if (debugMode) Debug.Log(message);
    }

    void HideAllIcons()
    {
        foreach (GameObject icon in allIcons)
        {
            if (icon != null) icon.SetActive(false);
        }

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

    void LoadProgress()
    {
        string saved =
PlayerPrefs.GetString("CollectedIcons", "");
        if (!string.IsNullOrEmpty(saved))
        {
            string[] ids = saved.Split(',');
            foreach (string id in ids)
            {
                if (int.TryParse(id, out int iconId))
                {
                    collectedIds.Add(iconId);
                }
            }
        }
    }
    void SaveProgress()
    {
        string ids = string.Join(",", collectedIds);
        PlayerPrefs.SetString("CollectedIcons", ids);
        PlayerPrefs.Save();
        Log($"Сохранено {collectedIds.Count} иконок");
    }
    void CreateNewSet()
    {
        Log($"=== СОЗДАНИЕ НАБОРА ===");

        if (collectedIds.Count >= allIcons.Length)
        {
            Log(" ВСЕ ИКОНКИ СОБРАНЫ!");
            ShowNextLevelButton();
            return;
        }

        // НЕ собранные иконки
        List<GameObject> available = new
List<GameObject>();

        for (int i = 0; i < allIcons.Length; i++)
        {
            if (allIcons[i] != null &&
!collectedIds.Contains(i))
            {
                available.Add(allIcons[i]);
            }
        }

        Log($"Доступно: {available.Count} иконок");

        if (available.Count == 0)
        {
            Log("Нет доступных иконок!");
            ShowNextLevelButton();
            return;
        }

        // Добавляем оставшиеся
        foreach (var icon in remainingIcons)
        {
            if (!available.Contains(icon))
            {
                available.Add(icon);
            }
        }

        // Перемешиваем
        Shuffle(available);

        // Сколько показывать
        int count = Mathf.Min(iconsPerSet,
available.Count);
        Log($"Показываем {count} иконок");

        // Очищаем текущий
        currentSet.Clear();

        // Показываем
        for (int i = 0; i < count; i++)
        {
            currentSet.Add(available[i]);
            available[i].SetActive(true);
            AddClickHandler(available[i]);
            Log($"Показана: {available[i].name}");
        }

        // Оставшиеся
        remainingIcons = available.Skip(count).ToList();
        Log($"Осталось: {remainingIcons.Count}
иконок");
    }

    void AddClickHandler(GameObject icon)
    {
        // НАХОДИМ ИНДЕКС
        int iconIndex = -1;
        for (int i = 0; i < allIcons.Length; i++)
        {
            if (allIcons[i] == icon)
            {
                iconIndex = i;
                break;
            }
        }

        if (iconIndex == -1) return;
```

```
        Button btn = icon.GetComponent<Button>();
        if (btn == null) btn = icon.AddComponent<Button>();

        btn.onClick.RemoveAllListeners();
        btn.onClick.AddListener(() => OnIconClick(icon, iconIndex));
    }

    void OnIconClick(GameObject icon, int iconId)
    {
        Log($"Клик по иконке ID: {iconId}");
        StartCoroutine(IconClickAnimation(icon, iconId));
    }

    IEnumerator IconClickAnimation(GameObject icon, int iconId)
    {
        // Мигание
        Image img = icon.GetComponent<Image>();
        if (img != null)
        {
            Color original = img.color;

            for (int i = 0; i < 2; i++)
            {
                img.color = new Color(1, 1, 1, 0.3f);
                yield return new WaitForSeconds(0.1f);
                img.color = original;
                yield return new WaitForSeconds(0.1f);
            }

            // Исчезновение
            float timer = 0f;
            while (timer < 0.3f)
            {
                timer += Time.deltaTime;
                float alpha = 1f - (timer / 0.3f);
                img.color = new Color(original.r, original.g, original.b, alpha);
                yield return null;
            }
        }

        // Скрываем
        icon.SetActive(false);
        currentSet.Remove(icon);

        // Сохраняем
        collectedIds.Add(iconId);
        SaveProgress();

        Log($"Собрано: {collectedIds.Count}/19, осталось в наборе: {currentSet.Count}");

        // Новый набор если пусто
        if (currentSet.Count == 0)
        {
            Log("Набор пуст, создаем новый...");
            yield return new WaitForSeconds(0.5f);
            CreateNewSet();
        }
    }

    void ShowNextLevelButton()
    {
        if (nextLevelButton != null)
        {
            nextLevelButton.SetActive(true);
            Log("Кнопка 'Далее' показана");
        }
    }

    void Shuffle<T>(List<T> list)
    {
        for (int i = 0; i < list.Count; i++)
        {
            int r = Random.Range(i, list.Count);
            T temp = list[i];
            list[i] = list[r];
            list[r] = temp;
        }
    }

    public void ResetGame()
    {
        collectedIds.Clear();
        currentSet.Clear();
        remainingIcons.Clear();
        PlayerPrefs.DeleteAll();

        HideAllIcons();
        CreateNewSet();

        Log("Прогресс сброшен");
    }
}
```

Скрипт таймера для интро

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

public class IntroLoader : MonoBehaviour
{
    [Header("Settings")]
    public float introDuration = 5f;  // Длительность интро в секундах
    public int nextSceneIndex = 1;  // Индекс сцены для перехода

    void Start()
    {
        // Запускаем таймер перехода
        StartCoroutine(LoadNextSceneAfterDelay());
    }

    IEnumerator LoadNextSceneAfterDelay()
    {
        // Ждем указанное время
        yield return new WaitForSeconds(introDuration);

        // Переходим на следующую сцену
        SceneManager.LoadScene(nextSceneIndex);
    }
```

```
    // Метод для пропуска интро по клику
    public void SkipIntro()
    {
        StopAllCoroutines();  // Останавливаем таймер
        SceneManager.LoadScene(nextSceneIndex);
    }
}
```

Скприт для смены сцен

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneLoader : MonoBehaviour
{
    // Загрузка по индексу сцены
    public void LoadSceneByIndex(int sceneIndex)
    {
        SceneManager.LoadScene(sceneIndex);
    }

    // Загрузка по имени
    public void LoadSceneByName(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }

    // Перезагрузка текущей сцены
    public void ReloadCurrentScene()
    {
        int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(currentSceneIndex);
    }

    // Выход из игры
    public void ExitGame()
    {
        Application.Quit();
    }
}
```

Скрипт для воспроизведения музыки

```
using UnityEngine;

public class MusicManager : MonoBehaviour
{
    public static MusicManager Instance;

    [Header("Настройки музыки")]
    public AudioClip backgroundMusic;
    public AudioSource audioSource;

    private bool isMusicOn = true;

    void Awake()
    {
        // Синглтон pattern
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            // Настройка аудио
            if (audioSource == null)
                audioSource = GetComponent<AudioSource>();

            audioSource.clip = backgroundMusic;
            audioSource.loop = true;

            // Загружаем настройки
            LoadMusicState();

            // Запускаем музыку согласно настройкам
            if (isMusicOn && !audioSource.isPlaying)
                audioSource.Play();
        }
        else
        {
            Destroy(gameObject);
        }
    }

    // Публичные методы для управления музыкой
    public void SetMusic(bool enabled)
    {
        isMusicOn = enabled;

        if (isMusicOn)
        {
            audioSource.volume = 1f;
            audioSource.mute = false;
            if (!audioSource.isPlaying)
                audioSource.Play();
        }
        else
        {
            audioSource.volume = 0f;
            audioSource.mute = true;
        }

        SaveMusicState();
    }

    public bool IsMusicOn()
    {
        return isMusicOn;
    }

    void SaveMusicState()
    {
        PlayerPrefs.SetInt("MusicEnabled", isMusicOn ? 1 : 0);
        PlayerPrefs.Save();
    }

    void LoadMusicState()
    {
        if (PlayerPrefs.HasKey("MusicEnabled"))
        {
            isMusicOn = PlayerPrefs.GetInt("MusicEnabled") == 1;
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;


public class PlayAudioOnImageClick : MonoBehaviour
{
    [Header("Аудио настройки")]
    public AudioClip audioClip;          // Аудиофайл для воспроизведения
    public float volume = 1.0f;        // Громкость (0-1)
    public bool loop = false;          // Зациклить аудио
    public bool playOnAwake = false;   // Воспроизвести при старте

    [Header("Дополнительные настройки")]
    public bool stopOnSecondClick = false; // Остановить при повторном клике
    public bool allowMultiplePlays = false; // Разрешить несколько одновременных воспроизведений

    private AudioSource audioSource;
    private bool isPlaying = false;

    void Start()
    {
        // Создаем AudioSource если его нет
        audioSource = GetComponent<AudioSource>();
        if (audioSource == null)
        {
            audioSource = gameObject.AddComponent<AudioSource>();
        }

        // Настраиваем AudioSource
        audioSource.clip = audioClip;
        audioSource.volume = volume;
        audioSource.loop = loop;
        audioSource.playOnAwake = playOnAwake;

        // Делаем картинку кликабельной
        MakeImageClickable();
    }

    void MakeImageClickable()
    {
        // Если это UI Image
        Image image = GetComponent<Image>();
        if (image != null)
        {
            // Добавляем Button компонент если его нет
            Button button = GetComponent<Button>();
            if (button == null)
            {
                button = gameObject.AddComponent<Button>();
                button.transition = Selectable.Transition.None; // Без визуальной обратной связи
            }

            button.onClick.AddListener(OnImageClicked);
        }
        else
        {
            // Если это SpriteRenderer (2D спрайт)
            SpriteRenderer spriteRenderer = GetComponent<SpriteRenderer>();
            if (spriteRenderer != null)
            {
                // Добавляем коллайдер для клика
                if (GetComponent<Collider2D>() == null)
                {
                    gameObject.AddComponent<BoxCollider2D>();
                }
            }
        }
    }

    // Вызывается при клике на UI Image
    void OnImageClicked()
    {
        ToggleAudio();
    }

    // Вызывается при клике на 2D спрайт
    void OnMouseDown()
    {
        ToggleAudio();
    }

    void ToggleAudio()
    {
        if (stopOnSecondClick && isPlaying)
        {
            StopAudio();
        }
        else
        {
            PlayAudio();
        }
    }

    public void PlayAudio()
    {
        if (audioClip == null)
        {
            Debug.LogError("Аудиофайл не назначен!");
            return;
        }

        if (audioSource == null)
        {
            audioSource = GetComponent<AudioSource>();
            if (audioSource == null) return;
        }

        // Если разрешено несколько воспроизведений - создаем новый AudioSource
        if (allowMultiplePlays && audioSource.isPlaying)
        {
            AudioSource newSource = gameObject.AddComponent<AudioSource>();
            newSource.clip = audioClip;
            newSource.volume = volume;
            newSource.loop = loop;
```

```
            newSource.Play();

            // Удаляем временный AudioSource после завершения
            if (!loop)
            {
                Destroy(newSource, audioClip.length + 0.1f);
            }
        }
        else
        {
            // Обычное воспроизведение
            if (!audioSource.isPlaying)
            {
                audioSource.Play();
                isPlaying = true;

                // Сбрасываем флаг после завершения (если не зациклено)
                if (!loop)
                {
                    Invoke("ResetPlayingFlag", audioClip.length);
                }
            }
            else if (!loop)
            {
                // Перезапускаем если уже играет и не зациклено
                audioSource.Stop();
                audioSource.Play();
            }
        }

        Debug.Log("Воспроизводится аудио: " + audioClip.name);
    }

    public void StopAudio()
    {
        if (audioSource != null && audioSource.isPlaying)
        {
            audioSource.Stop();
            isPlaying = false;
            Debug.Log("Аудио остановлено");
        }
    }

    public void PauseAudio()
    {
        if (audioSource != null && audioSource.isPlaying)
        {
            audioSource.Pause();
            isPlaying = false;
            Debug.Log("Аудио на паузе");
        }
    }

    public void ResumeAudio()
    {
        if (audioSource != null && !audioSource.isPlaying)
        {
            audioSource.UnPause();
            isPlaying = true;
            Debug.Log("Аудио возобновлено");
        }
    }

    void ResetPlayingFlag()
    {
        isPlaying = false;
    }

    // Публичные методы для управления из других скриптов
    public void SetVolume(float newVolume)
    {
        volume = Mathf.Clamp01(newVolume);
        if (audioSource != null)
        {
            audioSource.volume = volume;
        }
    }

    public void SetAudioClip(AudioClip newClip)
    {
        audioClip = newClip;
        if (audioSource != null)
        {
            bool wasPlaying = audioSource.isPlaying;
            audioSource.clip = newClip;
            if (wasPlaying) audioSource.Play();
        }
    }

    public bool IsPlaying()
    {
        return isPlaying;
    }
}
```

Приложение Б

Диаграмма последовательности

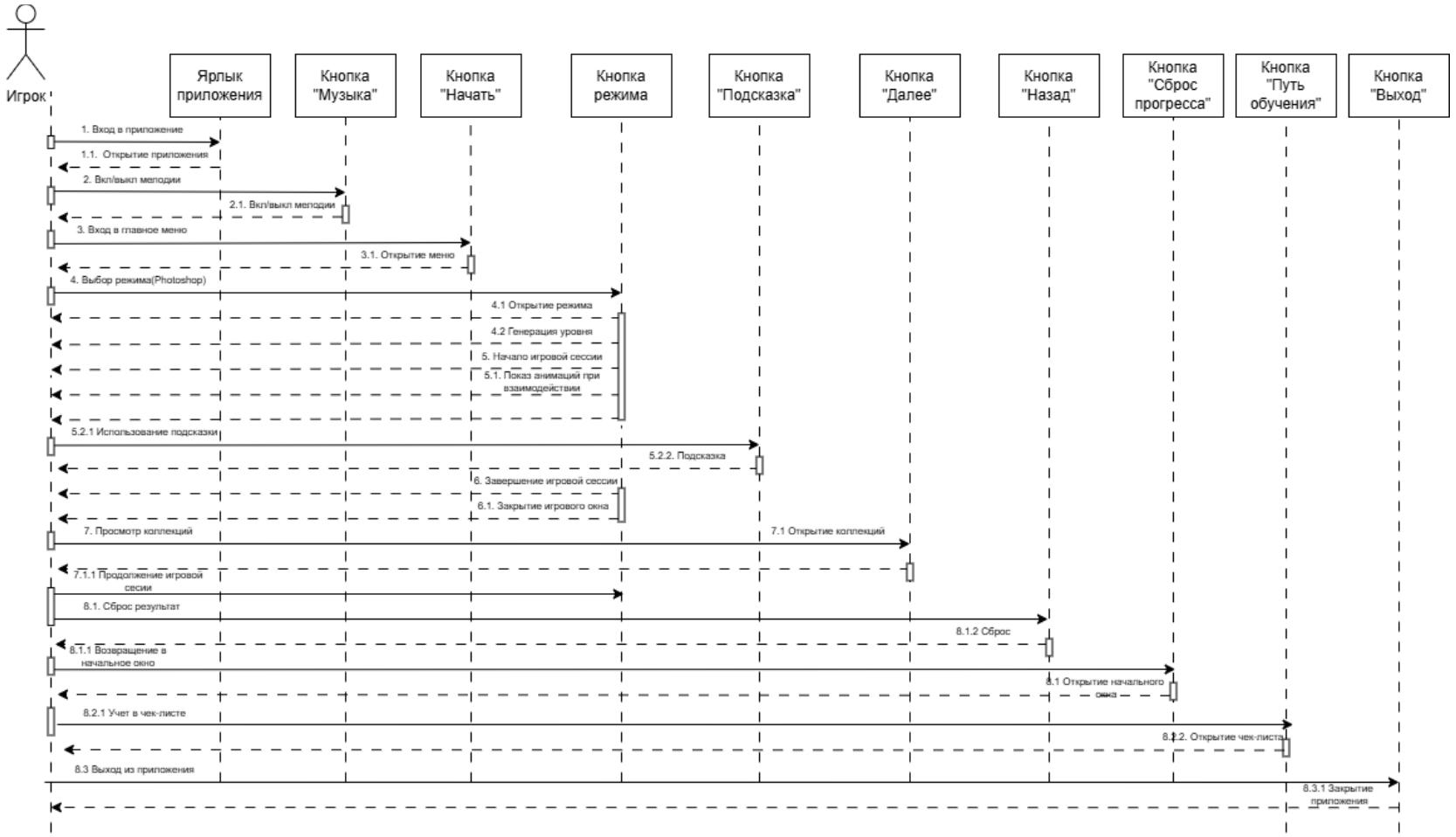

Рисунок Б.1 –Диаграмма последовательности

Приложение В

Диаграмма компонентов

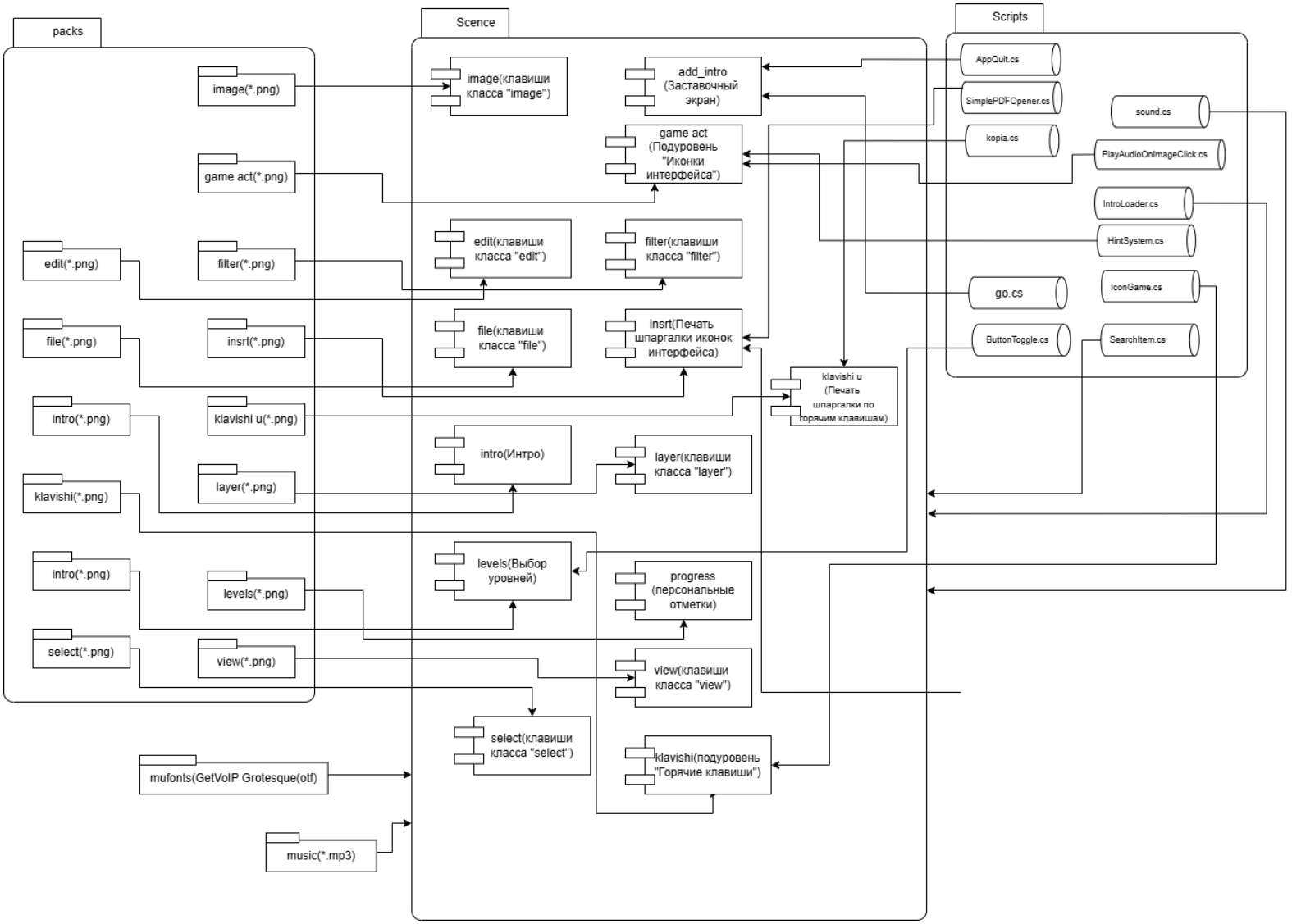

Рисунок Д.1 –Диаграмма компонентов

Приложение Г

Таблицы тест-кейсов

Приложение Д
Прототипы интерфейсов

Рисунок Д.1 – Заставка

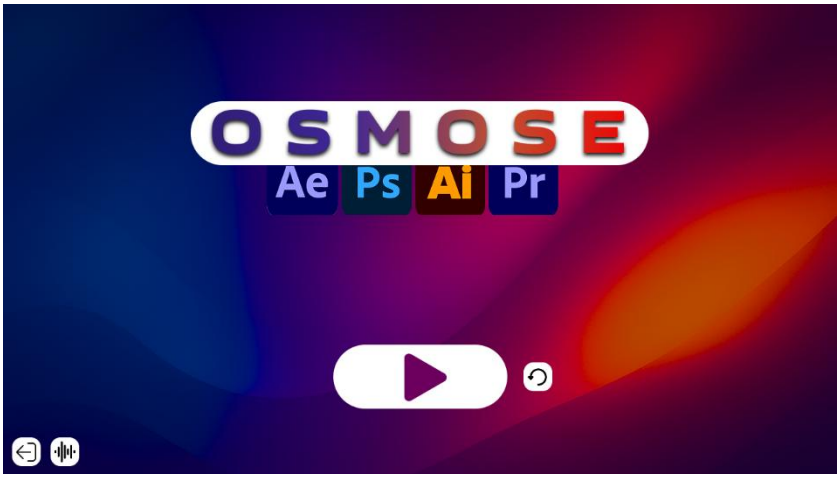

Рисунок Д.2 –Заставочный экран

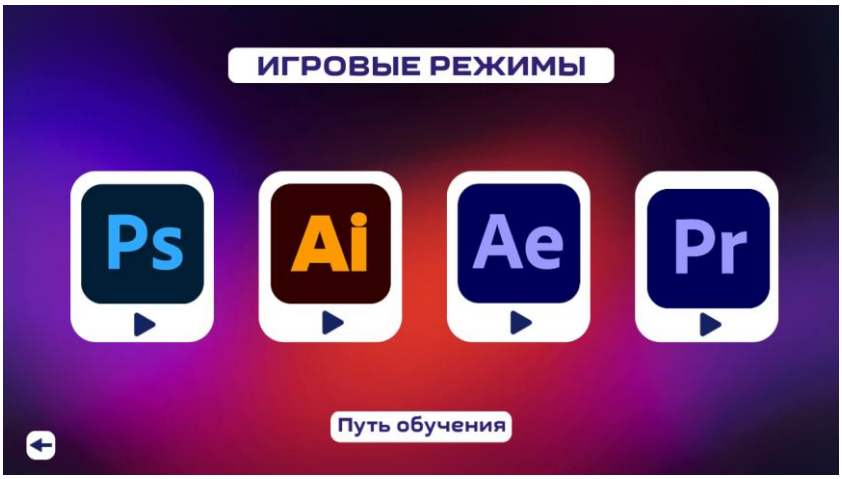

Рисунок Д.3 –Уровни

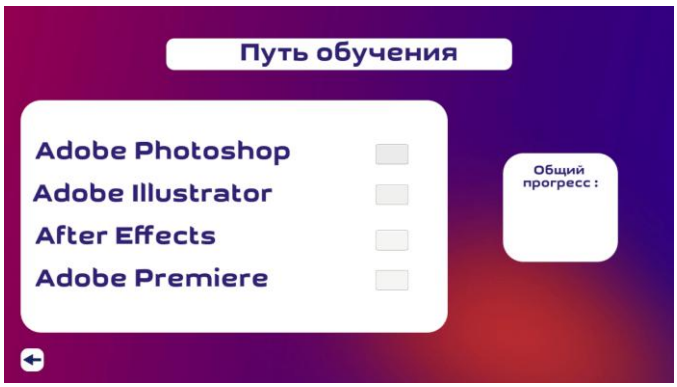

Рисунок Д. 4 –Личные отметки

Рисунок 5 – Вид подуровней

Рисунок Д.6 – Шпаргалки

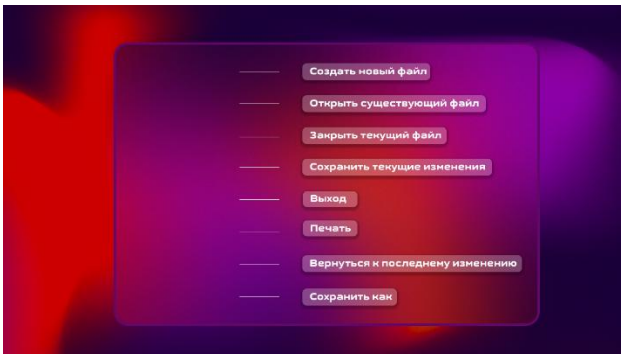

Рисунок Д.7 – Отдельные шаблоны для подгрупп горячих клавиш