

Создание мобильной игры «Заплатки» на Unity

Голубева Евгения Павловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Целью данной статьи является разработка мобильной игры «Заплатки» 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 разработана мобильная игра «Заплатки».

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

Creating a mobile game "Patches" on Unity

Golubeva Evgeniya Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to develop a mobile game "Patches" 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 developed the mobile game "Patches".

Keywords: Unity, game, patches, image, color.

1 Введение

1.1 Актуальность

Игра «Заплатки» — это современная цифровая головоломка, которая превращает простую механику подбора в инструмент для развития внимания, зрительно-моторной координации и ассоциативного мышления. Такой формат сохраняет популярность благодаря позитивному психологическому эффекту: он способствует снятию стресса через сосредоточение на понятных визуальных задачах и приносит удовлетворение от моментального результата, когда заплатка идеально совпадает с изображением.

Разработка этой игры на движке Unity является оптимальным и эффективным решением. Движок предоставляет полный набор инструментов для работы с 2D-графикой, создания интуитивного интерфейса с перетаскиванием объектов и отзывчивой физикой их «прилипания», а также для быстрой публикации проекта на мобильные платформы. Это позволяет за

короткое время пройти путь от прототипа, где проверяется основа игрового процесса, до готового продукта в магазинах приложений.

Таким образом, проект по созданию игры «Заплатки» на Unity решает несколько практических задач: он даёт опыт в ключевых аспектах мобильной разработки, включая реализацию тактильного управления и логику соответствия объектов, и создаёт востребованный продукт. Игра ориентирована на массовую аудиторию, ищущую одновременно успокаивающий и интуитивно понятный досуг с мгновенной визуальной наградой.

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

Целью данного исследования является разработка мобильной игры «Заплатки» под операционную систему Android с использованием кроссплатформенной среды для создания интерактивных 2D и 3D приложений Unity.

2 Материалы и методы

Разработка мобильной игры «Заплатки» для ОС Android была реализована в кроссплатформенном игровом движке Unity, который выступил в качестве основного фреймворка для создания 2D-графики, реализации физики объектов, построения пользовательского интерфейса и управления игровыми сценами, а вся программная логика, включая механику перетаскивания заплаток и проверку их соответствия, была написана на языке C# в интегрированной среде разработки Visual Studio Community.

3 Результаты

Разработка начинается с инициализации проекта в среде Unity. Через лаунчер Unity Hub необходимо создать новый проект, выбрав специализированный шаблон для 2D-разработки (Universal 2D). После указания названия проекта и его расположения в файловой системе процесс инициализации завершается нажатием соответствующей кнопки (см. рис. 1).

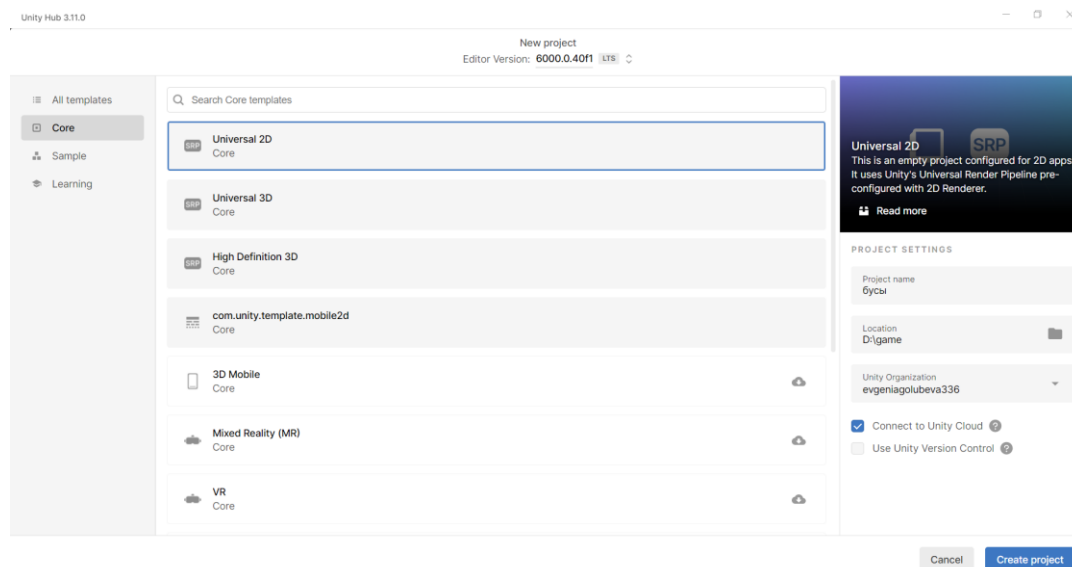


Рисунок 1 – Создание проекта

После создания проекта по шаблону Universal 2D, необходимо настроить его для сборки под Android. Для этого необходимо перейти в меню File -> Build Settings. В открывшемся окне открыть платформу Android в списке и нажать на кнопку Switch Platform. Unity начнёт процесс перенастройки проекта, что может занять некоторое время (см. рис. 2).

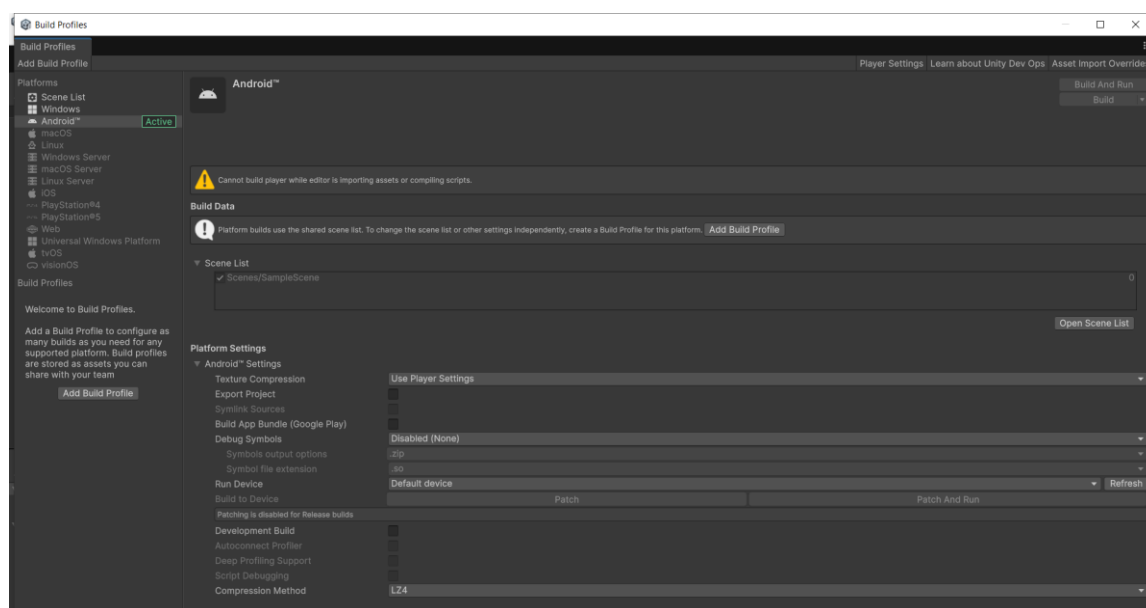


Рисунок 2 – Настройка проекта

Следующей задачей является построение игровой сцены. Приоритетной является настройка камеры, поскольку её параметры задают область видимости и фундаментальные визуальные свойства проекции игрового мира (см. рис. 3).

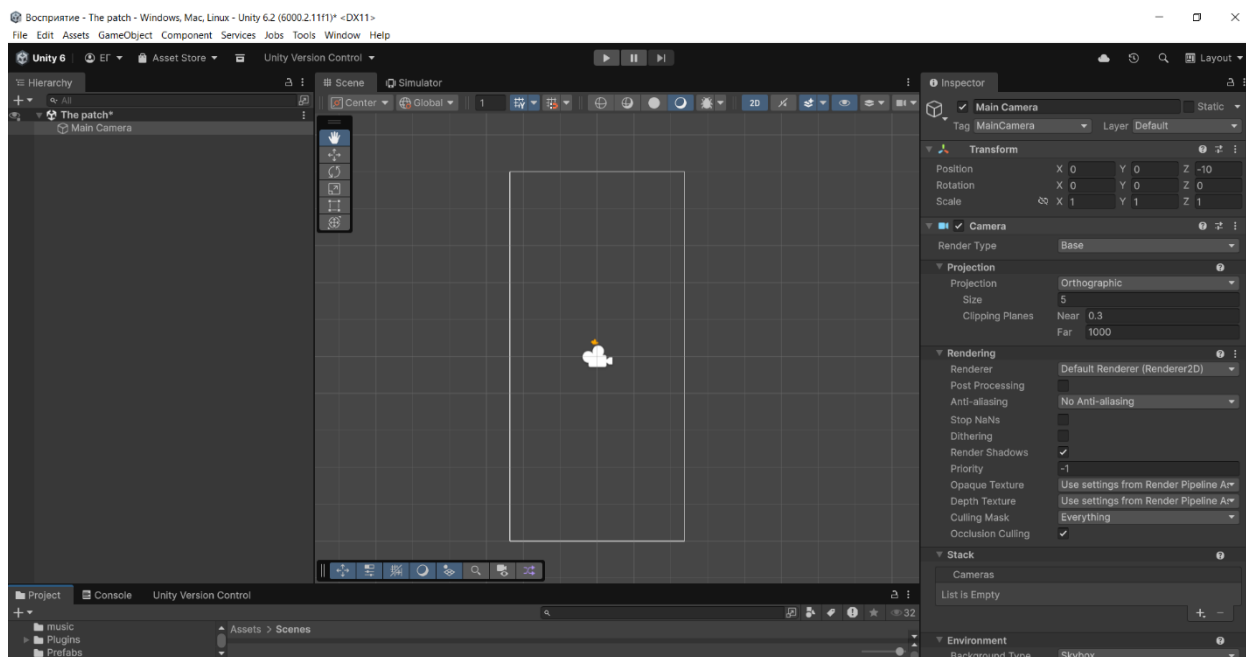


Рисунок 3 – Настройка камеры

Далее необходимо импортировать все подготовленные графические файлы (спрайты) в проект. Для этого переместить их из проводника в папку Assets/Sprites в окне Project Unity (см.рис.4).

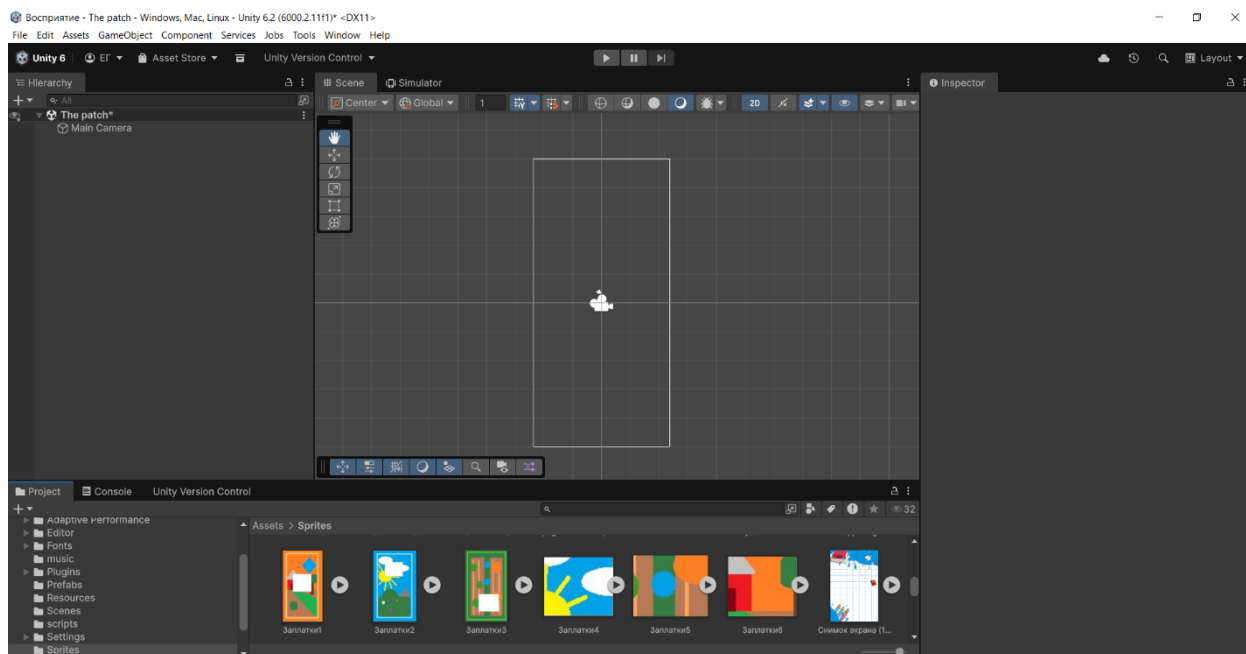


Рисунок 4 – Добавление изображений

Чтобы создать фон, необходимо переместить спрайт background из папки Sprites на сцену. После этого в инспекторе необходимо настроить позицию, масштаб и выравнивание так, чтобы он полностью покрывал область, которую видит камера (см.рис.5).

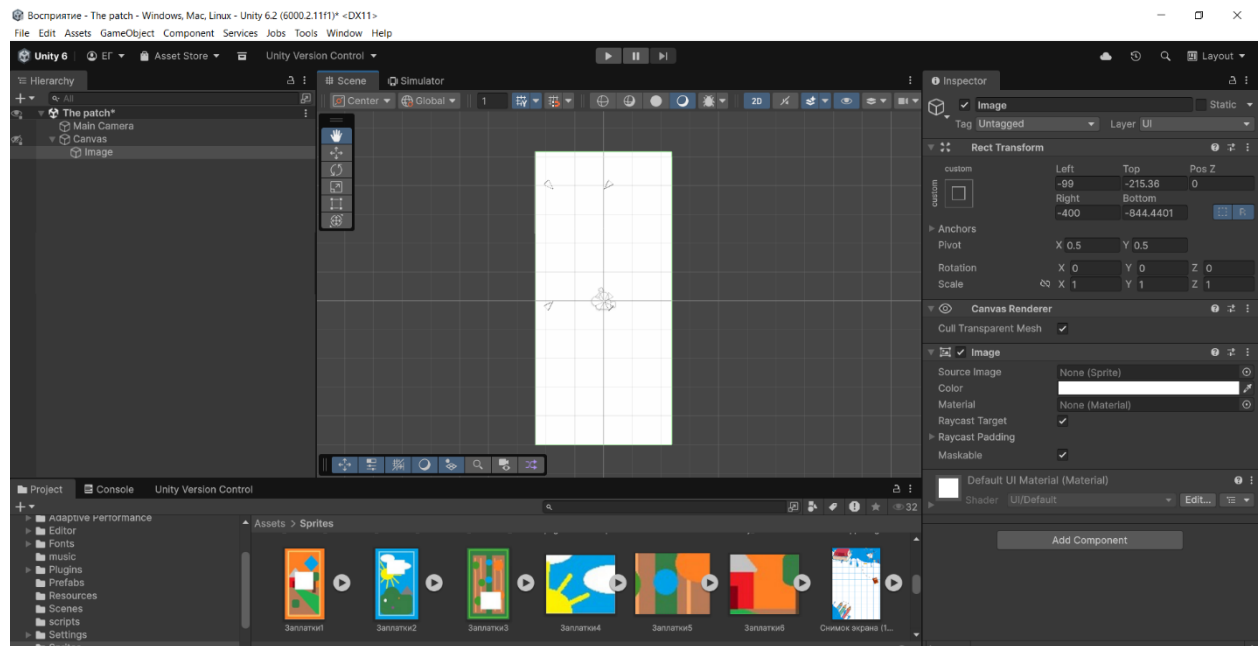


Рисунок 5 – Настройка фона игры

Далее осуществляется наполнение сцены графическими объектами. Для этого заранее созданные спрайты игровых элементов импортируются из панели Project в иерархию сцены (Hierarchy) с последующей ручной расстановкой в соответствии с игровым дизайном (см. рис. 6).

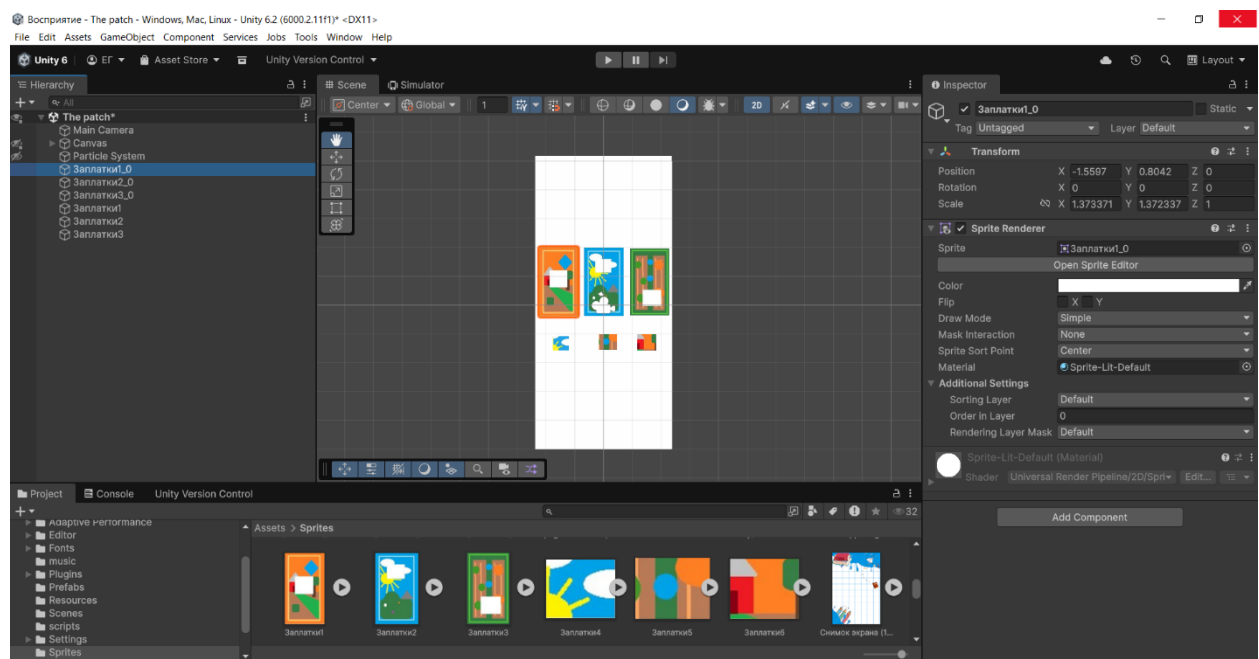


Рисунок 6 – Добавление спрайтов на сцену

Чтобы предметы корректно отображались и реагировали на действия игрока, для каждого из них вручную настраиваются координаты в Transform. После этого к объекту добавляются два основных компонента: Sprite Renderer для визуального представления и Circle Collider 2D для создания зоны физического или логического взаимодействия (см.рис.7).

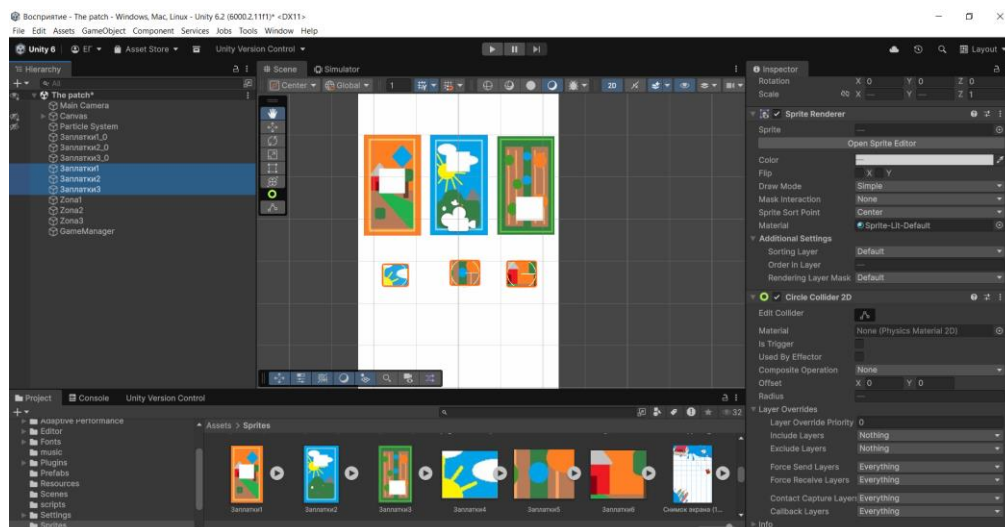


Рисунок 7 – Добавление изображений заплаток на сцену и их настройка

Для реализации отображения инструкций и уведомления о правильном действии необходимо создать элементы пользовательского интерфейса (UI). В окне Hierarchy через контекстное меню добавьте объект Panel (UI -> Panel), а затем создайте внутри него элемент Text (UI -> Text) для вывода текстовых сообщений.

После добавления этих элементов требуется их детальная настройка в Inspector. Для корректного отображения на разных экранах установите Anchors в центральное положение (0.5, 0.5). Затем задайте фиксированный размер панели, указав Width: 1049.1 и Height: 481.5. Для визуального оформления в компоненте Image выберите фоновый спрайт в поле Source Image, настройте цвет и прозрачность в параметре Color, а в качестве типа отображения установите Simple (см.рис.8).

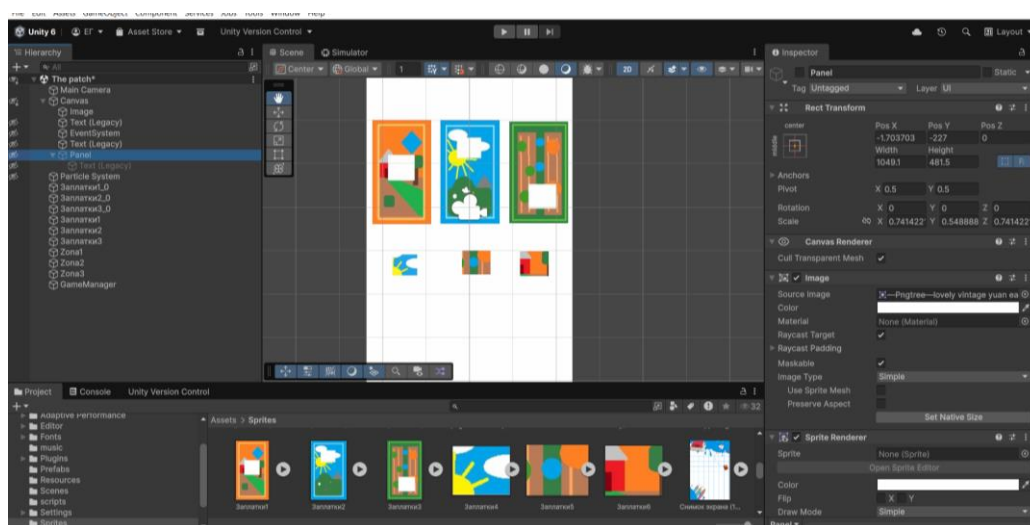


Рисунок 8 – Настройка панели

Затем выполняется детальная настройка текстового блока с уведомлением «Правильно!». В компоненте Text (Legacy) выбирается шрифт

PlaypenSans-VariableFont_wght с начертанием Bold и устанавливается размер 150 пунктов для обеспечения высокой читаемости и визуального акцента. Геометрия элемента задается в компоненте Rect Transform, где фиксируются ширина 1041.7 и высота 347.8, а якоря (Anchors) устанавливаются в центральную позицию (0.5, 0.5), что обеспечивает стабильное и централизованное расположение на любом разрешении экрана. Завершается настройка определением визуальных свойств текста: выбирается центральное выравнивание по горизонтали и вертикали, задается цвет, соответствующий общей палитре интерфейса, и при необходимости активируется перенос строк для гармоничного и сбалансированного отображения уведомления в игровом пространстве (см.рис. 9).

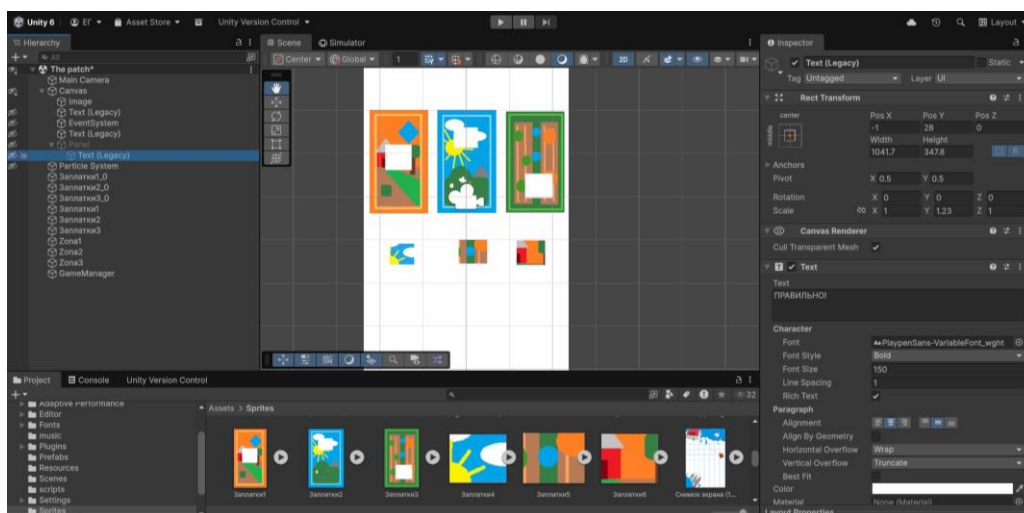


Рисунок 9 – Настройка текстового элемента панели сообщений

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 1064, Height 344,7, и устанавливаем якоря Anchors с центром в точке (0.5, 0.5) (см.рис.10).

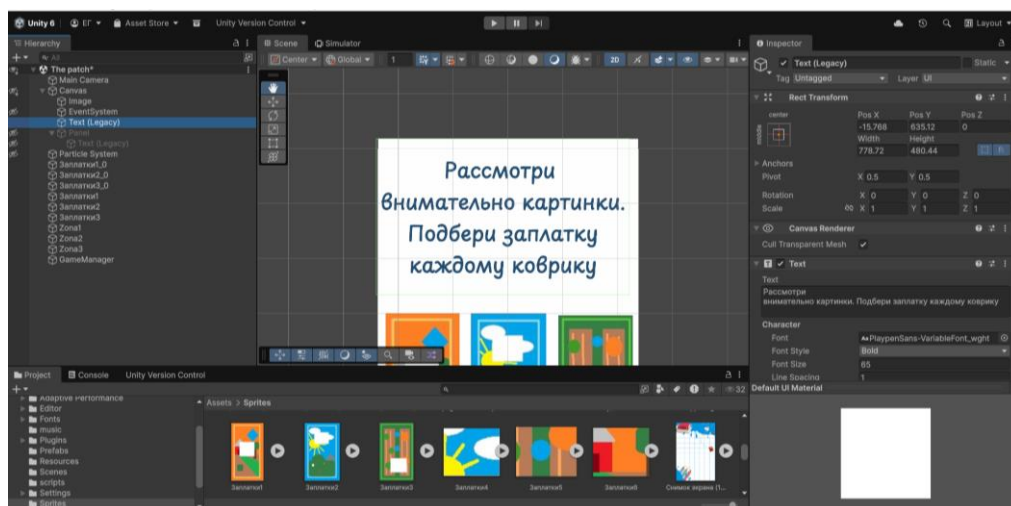


Рисунок 10 – Настройка инструкции игры

Для корректной фиксации фрагментов заплатки в своих ячейках после добавления всех изображений на сцену необходимо создать специальные триггерные зоны в местах отсутствующих заплаток. Это выполняется путём создания в иерархии (Hierarchy) трёх пустых игровых объектов с именами Zona1, Zona2 и Zona3. Для каждого из этих объектов в инспекторе (Inspector) требуется последовательная настройка: сначала задаются точные координаты позиции (Transform) в соответствии с расположением целевых ячеек на рамке пазла, затем добавляется компонент Box Collider 2D, который настраивается в режиме триггера активируется опция Is Trigger и его размер подгоняется под габариты ячейки. Для организации корректного взаимодействия в логике игры рекомендуется также назначить этим объектам отдельный слой (Layer) и уникальный тег (Tag), что упростит обработку событий столкновения в скриптах (см. рис. 11).



Рисунок 11 – Добавление зон для заплаток и их настройка

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

Для компонента GameManager необходимо создать скрипт PatchGame.

Скрипт PatchGame представляет собой полноценную игровую систему для создания интерактивной головоломки в среде Unity. Основная концепция игры строится вокруг механики "заплаток", где игроку необходимо перетаскивать элементы с вырезанными областями (дырками) на соответствующие целевые зоны для восстановления целостного изображения. Этот тип геймплея особенно популярен в образовательных и развивающих приложениях для детей, а также в казуальных играх, где требуется развитие пространственного мышления и моторики.

Игровой процесс интуитивно понятен: на экране располагаются несколько объектов с отверстиями неправильной формы, а игрок должен найти для каждой "заплатки" правильное место на основном изображении. При успешном размещении визуально проявляется скрытая часть изображения, создавая удовлетворяющий эффект завершенности. Скрипт разработан с учетом кроссплатформенности, поддерживая управление как сенсорным вводом на мобильных устройствах, так и традиционным управлением мышью на персональных компьютерах.

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

public class PatchGame : MonoBehaviour
{
    [System.Serializable]
    public class PatchItem
    {
        public GameObject patchObject;      // Объект заплатки (с дыркой)
        public GameObject targetObject;      // Целевой объект (полный спрайт)
        public float snapDistance = 1.0f;    // Дистанция для притягивания
        [HideInInspector] public Vector2 startPosition; // Начальная позиция
        [HideInInspector] public bool isInPlace; // Флаг правильного положения
    }

    [Header("Patch Elements")]
    public List<PatchItem> patches = new List<PatchItem>();
    public GameObject winMessage; // Сообщение о победе (опционально)

    [Header("Game Settings")]
    public float returnSpeed = 10f; // Скорость возврата заплатки
    public string menuSceneName = "Menu";

    [Header("Audio Settings")]
    public AudioClip patchPickupSound;      // Звук при взятии заплатки
    public AudioClip patchReturnSound;      // Звук при возврате заплатки
    public AudioClip patchCorrectSound;     // Звук при правильном размещении
    public AudioClip gameCompleteSound;     // Звук при завершении игры

    [Header("UI Elements")]
    public Text winText; // Альтернатива: используйте UI Text вместо GameObject

    private PatchItem currentDraggedPatch;
    private Vector2 offset;
    private Camera mainCamera;
    private bool isCompleted = false;
    private AudioSource audioSource;
    private bool isReturning = false; // Флаг для анимации возврата
    private Vector2 pointerPosition; // Позиция указателя

    // Ссылка на Input Actions
    private PlayerInput playerInput;
    private InputAction touchPositionAction;
    private InputAction touchPressAction;

    void Start()
    {
        mainCamera = Camera.main;

        // Получаем или добавляем AudioSource
        audioSource = GetComponent<AudioSource>();
        if (audioSource == null)
```

```
{
    audioSource = gameObject.AddComponent<AudioSource>();
}

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

// Настраиваем Input Actions
SetupInputActions();

InitializePatches();

// Скрываем сообщение о победе, если оно существует
if (winMessage != null)
{
    winMessage.SetActive(false);
}

// Скрываем текст о победе, если он существует
if (winText != null)
{
    winText.gameObject.SetActive(false);
}
}

void SetupInputActions()
{
    // Создаем простые действия
    var touchActionMap = new InputActionMap("Touch");

    // Действие для позиции касания/мышь
    touchPositionAction = touchActionMap.AddAction("Position",
        InputActionType.Value,
        "<Pointer>/position");

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

    // Подписываемся на события
    touchPressAction.started += ctx => OnTouchStarted(ctx);
    touchPressAction.canceled += ctx => OnTouchEnded(ctx);

    // Включаем действия
    touchActionMap.Enable();
}

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

    // Получаем позицию указателя
    pointerPosition = touchPositionAction.ReadValue<Vector2>();
    TryStartDrag(pointerPosition);
}

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

void InitializePatches()
{
    foreach (var patch in patches)
    {

```

```
        if (patch.patchObject != null)
        {
            patch.startPosition = patch.patchObject.transform.position;
            patch.isInPlace = false;
            AddColliderIfMissing(patch.patchObject);

            // Делаем заплатку (дырку) полностью видимой изначально
            SetPatchAlpha(patch.patchObject, 1f);

            // Убедимся, что коллайдер включен
            Collider2D collider = patch.patchObject.GetComponent<Collider2D>();
            if (collider != null)
                collider.enabled = true;
        }
        else
        {
            Debug.LogError("В списке patches есть пустой patchObject!");
        }

        // Скрываем целевой объект (полный кобрик) изначально
        if (patch.targetObject != null)
        {
            SetPatchAlpha(patch.targetObject, 0f);
        }
        else
        {
            Debug.LogError($"Заплата {patch.patchObject?.name} не имеет
назначенного targetObject!");
        }
    }
}

void Update()
{
    if (isCompleted) return;

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

    // Если перетаскиваем заплатку, обновляем ее позицию
    if (currentDraggedPatch != null && !isReturning)
    {
        UpdatePatchPosition();
    }
}

void UpdatePatchPosition()
{
    if (currentDraggedPatch != null && !isReturning)
    {
        Vector2 worldPos = mainCamera.ScreenToWorldPoint(pointerPosition);
        currentDraggedPatch.patchObject.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 patch in patches)
        {
            if (hit.collider.gameObject == patch.patchObject && !patch.isInPlace)
            {

```

```

        currentDraggedPatch = patch;
        offset = (Vector2)patch.patchObject.transform.position - worldPos;

        // Воспроизводим звук взятия заплатки
        PlaySound(patchPickupSound);
        break;
    }
}

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

    // УСОВЕРШЕНСТВОВАННАЯ ЛОГИКА: Проверяем, находится ли заплатка над своим
целевым объектом
    Vector2 patchPos = currentDraggedPatch.patchObject.transform.position;
    Vector2 targetPos = currentDraggedPatch.targetObject.transform.position;
    float distance = Vector2.Distance(patchPos, targetPos);

    // Используем коллайдер для более точного определения попадания
    Collider2D patchCollider =
currentDraggedPatch.patchObject.GetComponent<Collider2D>();
    Collider2D targetCollider =
currentDraggedPatch.targetObject.GetComponent<Collider2D>();

    bool isOverlapping = false;

    if (patchCollider != null && targetCollider != null)
    {
        isOverlapping = patchCollider.bounds.Intersects(targetCollider.bounds);
    }

    // Проверяем два условия: расстояние ИЛИ перекрытие коллайдеров
    if (distance <= currentDraggedPatch.snapDistance || isOverlapping)
    {
        PlaceOnCobra();
    }
    else
    {
        StartCoroutine(ReturnToStartPosition());
        PlaySound(patchReturnSound);
    }

    currentDraggedPatch = null;
}

void PlaceOnCobra()
{
    // Перемещаем заплатку точно в позицию целевого объекта
    currentDraggedPatch.patchObject.transform.position =
currentDraggedPatch.targetObject.transform.position;
    currentDraggedPatch.isInPlace = true;

    // Отключаем коллайдер у заплатки, чтобы ее нельзя было больше двигать
    Collider2D collider =
currentDraggedPatch.patchObject.GetComponent<Collider2D>();
    if (collider != null)
        collider.enabled = false;

    // ВОТ ИСПРАВЛЕНИЕ: Не скрываем заплатку, а оставляем ее на месте!
    // Не запускаем анимацию исчезновения заплатки
    // StartCoroutine(FadePatch(currentDraggedPatch.patchObject, 1f, 0f, 0.3f));

    // Плавно показываем целевой объект (полный кобрик)
    StartCoroutine(FadePatch(currentDraggedPatch.targetObject, 0f, 1f, 0.3f));

    // Воспроизводим звук правильного размещения
    PlaySound(patchCorrectSound);
}

```

```
// Проверяем условие победы
CheckWinCondition();
}

IEnumerator ReturnToStartPosition()
{
    isReturning = true;
    Vector2 startPos = currentDraggedPatch.startPosition;
    GameObject patch = currentDraggedPatch.patchObject;

    while (Vector2.Distance(patch.transform.position, startPos) > 0.01f)
    {
        patch.transform.position = Vector2.Lerp(
            patch.transform.position,
            startPos,
            returnSpeed * Time.deltaTime
        );
        yield return null;
    }

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

IEnumerator FadePatch(GameObject patchObject, float startAlpha, float endAlpha,
float duration)
{
    SpriteRenderer spriteRenderer = patchObject.GetComponent<SpriteRenderer>();
    if (spriteRenderer == null) yield break;

    float elapsedTime = 0f;
    Color color = spriteRenderer.color;

    while (elapsedTime < duration)
    {
        elapsedTime += Time.deltaTime;
        float alpha = Mathf.Lerp(startAlpha, endAlpha, elapsedTime / duration);
        color.a = alpha;
        spriteRenderer.color = color;
        yield return null;
    }

    color.a = endAlpha;
    spriteRenderer.color = color;
}

void SetPatchAlpha(GameObject patchObject, float alpha)
{
    SpriteRenderer spriteRenderer = patchObject.GetComponent<SpriteRenderer>();
    if (spriteRenderer != null)
    {
        Color color = spriteRenderer.color;
        color.a = alpha;
        spriteRenderer.color = color;
    }
    else
    {
        Debug.LogWarning($"Объект {patchObject.name} не имеет SpriteRenderer!");
    }
}

void CheckWinCondition()
{
    foreach (var patch in patches)
    {
        if (!patch.isInPlace) return;
    }

    StartCoroutine(CompleteGame());
}
```

```
IEnumerator CompleteGame()
{
    isCompleted = true;

    // Воспроизводим звук завершения игры
    PlaySound(gameCompleteSound);

    // Показываем сообщение о победе, если оно есть
    if (winMessage != null)
    {
        winMessage.SetActive(true);
    }
    else if (winText != null)
    {
        winText.gameObject.SetActive(true);
    }
    else
    {
        Debug.Log("Победа! Все заплатки размещены правильно!");
    }

    // Ждем 2 секунды перед переходом в меню
    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;

    // Скрываем сообщение о победе, если оно есть
    if (winMessage != null)
    {
        winMessage.SetActive(false);
    }

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

    foreach (var patch in patches)
    {
        patch.patchObject.transform.position = patch.startPosition;
        patch.isInPlace = false;

        // Включаем коллайдер у заплатки
        Collider2D collider = patch.patchObject.GetComponent<Collider2D>();
        if (collider != null)
            collider.enabled = true;

        // Восстанавливаем видимость заплатки (дырки)
        SetPatchAlpha(patch.patchObject, 1f);

        // Скрываем целевой объект (полный кобрик)
        SetPatchAlpha(patch.targetObject, 0f);
    }
}
```

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

void OnDisable()
{
    // Отписываемся от событий при отключении объекта
    if (touchPressAction != null)
    {
        touchPressAction.started -= OnTouchStarted;
        touchPressAction.canceled -= OnTouchEnded;
    }
}

// Метод для настройки через инспектор (вызывается при изменении значений в инспекторе)
void OnValidate()
{
    // Убедимся, что snapDistance не отрицательный
    foreach (var patch in patches)
    {
        if (patch.snapDistance < 0.1f)
        {
            patch.snapDistance = 0.1f;
        }
    }
}
```

После создания скрипта PatchGame необходимо добавить его к компоненту GameManager, для чего следует перетащить файл скрипта на объект GameManager в окне Hierarchy. После этого в окне Inspector появятся настраиваемые параметры скрипта, которые необходимо заполнить в соответствии с логикой игры. В разделе Patch Elements представлен массив элементов, каждый из которых включает поля Patch Object, Target Object и Snap Distance. В поле Patch Object для каждого элемента нужно перетащить из сцены тот игровой объект, который будет выполнять роль перемещаемой заплатки. В поле Target Object следует указать целевой объект, к которому должен быть присоединён соответствующий Patch Object. Числовое поле Snap Distance определяет максимальное расстояние, при котором происходит автоматическое притягивание патча к цели — меньшее значение требует более точного позиционирования игроком.

В разделе Game Settings находится параметр Return Speed, задающий скорость возврата патча на исходную позицию в случае его отпущения вдали от цели, а также поле Menu Scene Name, куда необходимо вписать точное имя сцены главного меню, например, MainMenu, для корректного перехода после завершения игры. Раздел Audio Settings содержит поля для назначения звуковых эффектов: Patch Pickup Sound для звука поднятия патча, Patch Return Sound для звука его возврата, Patch Correct Sound для сигнала успешного соединения и Game Complete Sound для воспроизведения при полном завершении уровня. В каждое из этих полей нужно перетащить соответствующий аудиоклип из папки проекта.

В разделе UI Elements требуется связать визуальные компоненты: в поле Win Message Panel следует перетащить объект панели UI, которая будет отображаться при победе, а в поле Win Text — текстовый компонент, находящийся внутри этой панели, для вывода поздравительного сообщения. После заполнения всех параметров важно убедиться, что перетащенные объекты существуют в сцене, имена сцен указаны верно, аудиоклипы подключены корректно, а значения расстояний и скоростей соответствуют желаемому игровому балансу. Затем следует сохранить сцену и провести тестирование, чтобы удостовериться в правильности работы механики присоединения патчей, воспроизведения звуков и отображения интерфейсных элементов. (см. рис. 12).

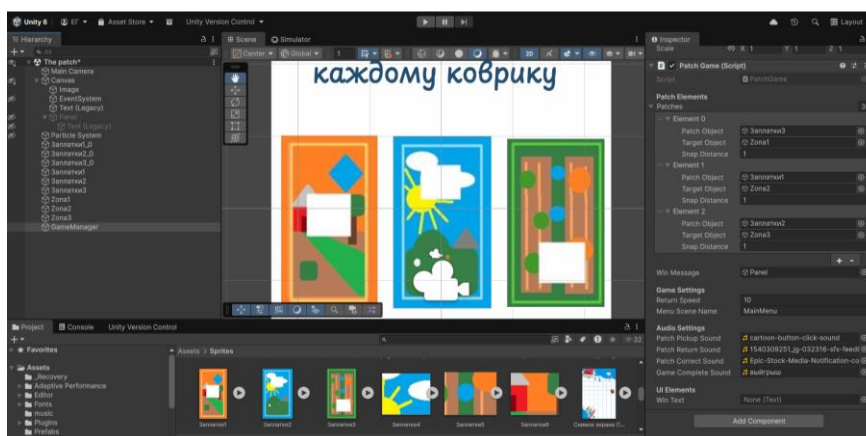


Рисунок 12 – Добавление скрипта к компоненту GameManager и его настройка

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



Рисунок 13 – Готовая игра «Заплатки»

4. Выводы

В рамках данной работы был реализован процесс создания образовательной игры «Заплатки» в кроссплатформенной среде Unity, демонстрирующий полный цикл разработки интерактивного приложения. Научная статья может быть использована в качестве основы для изучения программирования, проектирования пользовательских интерфейсов, а также как модель для создания собственных дидактических и развивающих приложений в академической и профессиональной среде.

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.