

Разработка игры-головоломки типа «пазл» на Unity

Матвеев Сергей Сергеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Статья посвящена разработке и реализации функционального прототипа игры-головоломки типа «пазл» на игровом движке Unity. Актуальность работы обусловлена устойчивым рыночным спросом на жанр, образовательным потенциалом подобных проектов и широкими возможностями кроссплатформенного движка Unity, идеально подходящего для данной задачи. В исследовании поставлена цель — создать прототип с базовой механикой сборки, адаптивным интерфейсом и системой уровней, а также проанализировать ключевые аспекты разработки. Методология основана на итеративном прототипировании с использованием Unity и языка C#. В результате был разработан рабочий прототип, включающий главное меню, алгоритмы динамического создания фрагментов пазла, их интерактивного перетаскивания и корректной сборки. Работа подтверждает эффективность Unity для создания головоломок и служит практическим руководством для начинающих разработчиков, демонстрируя типовые решения и выделяя направления для дальнейшей оптимизации и развития проекта.

Ключевые слова: Unity, игра-головоломка, пазл, разработка игр, C#, пользовательский интерфейс.

Development of a puzzle-type game "Jigsaw Puzzle" on Unity

Matveev Sergey Sergeevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article focuses on the development and implementation of a functional prototype of a jigsaw puzzle-type game using the Unity game engine. The relevance of the work is justified by the stable market demand for the genre, the educational potential of such projects, and the broad capabilities of the cross-platform Unity engine, which is ideally suited for this task. The study aims to create a prototype with core assembly mechanics, an adaptive interface, and a level system, as well as to analyze key aspects of the development process. The methodology is based on iterative prototyping using Unity and the C# language. As a result, a working prototype was developed, including a main menu, algorithms for dynamically generating puzzle pieces, their interactive drag-and-drop functionality, and correct assembly. The work confirms the effectiveness of Unity for creating puzzle games and serves as a

practical guide for novice developers, demonstrating typical solutions and highlighting areas for further optimization and project development.

Keywords: Unity, puzzle game, game development, C#, user interface.

1. Введения

1.1 Актуальность

Актуальность разработки игры-головоломки типа «пазл» на Unity обусловлена комплексом факторов, охватывающих технологические, рыночные и образовательные аспекты. Во-первых, жанр пазлов остается одним из самых популярных и демократичных в гейм-деве, стабильно востребованным широкой аудиторией всех возрастов благодаря интуитивному геймплею, развивающему логику и пространственное мышление, что обеспечивает проекту устойчивый коммерческий и пользовательский потенциал. Во-вторых, движок Unity, как кроссплатформенный и доступный инструмент, идеально подходит для создания таких проектов, позволяя относительно быстро реализовать механику сборки, адаптировать игру под мобильные устройства, ПК и веб, а также интегрировать современные фишки вроде тач-контроля, динамической генерации уровней или систем прогрессии, что соответствует текущим трендам индустрии. В-третьих, такой проект служит отличной практической задачей для обучения геймдев-специалистов, так как охватывает ключевые этапы разработки: от проектирования UI/UX и работы с 2D-графикой до реализации игровой логики и оптимизации под разные разрешения экранов. Наконец, в контексте растущей конкуренции на рынке казуальных игр, именно пазлы, дополненные оригинальной художественной стилистикой, сюжетом или нестандартной механикой (например, гибрид с другой жанровой механикой), демонстрируют высокие шансы на успех, что делает тему практически значимой для разработчиков, стремящихся создать востребованный и качественный продукт с умеренными затратами ресурсов.

1.2 Обзор исследования

Основываясь на исследовании Н.А. Базеевой и Д.С. Лебедева в области компьютерных наук, в статье языки программирования для создания игр представляет собой анализ зарождения игровой индустрии и сравнительную оценку языков программирования для выявления наиболее универсального и востребованного инструментария в кроссплатформенной игровой разработке [1]. В статье Т.Н. Астаховой, А.А. Бердниковой, С.В. Кривоногова и А.С. Лямина представлена разработка и экспериментальное обоснование специализированной компьютерной игры, направленной на развитие логического мышления у детей старшего дошкольного и младшего школьного возраста (5–6 классы), где подтверждается положительное влияние игровых методов на когнитивные способности и предлагаются конкретные технологии для их интеграции в образовательный процесс [2]. На основе исследования М.Р. Степченко, представленного в статье, было разработано и проанализировано многоуровневое мобильное приложение «Mindle» в жанре

головоломки, где обоснован выбор движка Unity для реализации пяти уникальных мини-игр, описаны ключевые архитектурные и интерфейсные решения, подтверждающие его эффективность для развития логики и внимания пользователя [3]. В исследовании А.А. Любич детально описан процесс создания 2D-платформера на Unity, включая реализацию ключевых механик управления персонажем, системы здоровья, разрушаемого окружения с помощью Tilemap, а также разнообразных врагов с уникальным поведением, что демонстрирует практическое применение C# и встроенных инструментов движка для разработки игр данного жанра [4]. В своей работе К.А. Матюнин рассматривает процесс разработки игр на Unity, акцентируя внимание на ключевых этапах создания проектов и необходимом наборе навыков для начинающих гейм дизайнеров, что подчеркивает доступность движка для непрофессиональных пользователей и его роль в качестве образовательного инструмента [5].

1.3 Цель исследования

Цель исследования — разработать и реализовать на игровом движке Unity функциональный прототип игры-головоломки типа «пазл», включающий базовую механику сборки, адаптивный пользовательский интерфейс и систему прогрессии уровней, с последующим анализом и оценкой ключевых аспектов разработки: производительности, удобства управления и масштабируемости архитектуры кода. Реализация проекта направлена на практическую демонстрацию типовых решений для жанра, а его анализ — на выявление оптимальных подходов и потенциальных проблем при создании подобных игр, что в итоге позволит сформулировать методические рекомендации для начинающих разработчиков.

2. Материалы и методы

Разработка проводилась в Unity 2019 LTS с использованием 2D-инструментов и C#. Методология основана на итеративном прототипировании: сначала реализовано базовое ядро (перетаскивание и «притягивание» фрагментов), затем система уровней, UI и оптимизация (пулы объектов, атласы). Тестирование выполнялось в редакторе и на компьютере для оценки производительности.

3. Результат

Для начала создаем проект в Unity (рис. 1).

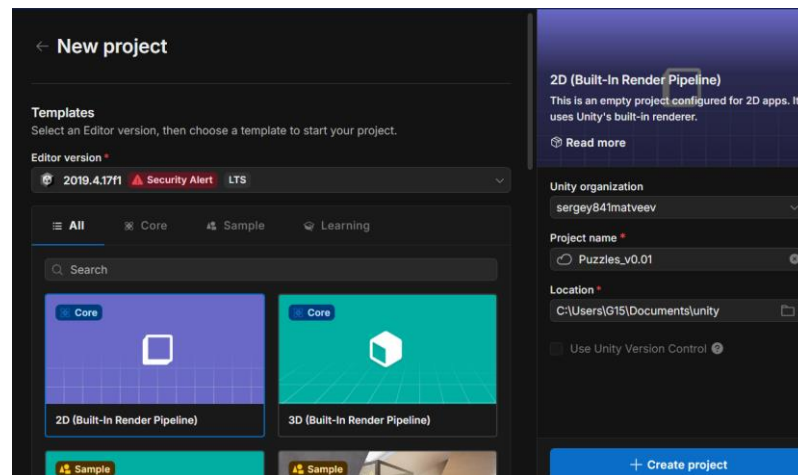


Рисунок 1 – Окно создания проекта

Далее добавляем объект “Canvas” для отображения игрового интерфейса (рис. 2).

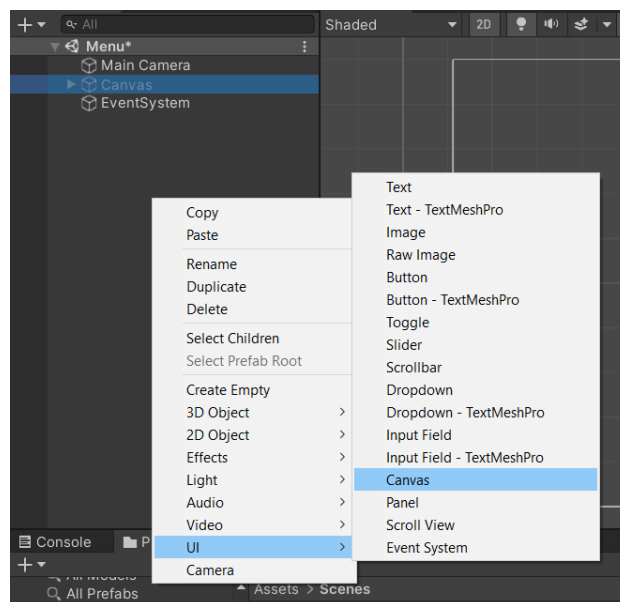


Рисунок 2 – Создания “Canvas”

И в настройках “Canvas” в поле “Render Camera” перетащим туда объект “Main Camera” для того чтобы камера захватывала весь “Canvas” то есть игровой интерфейс (рис. 3).

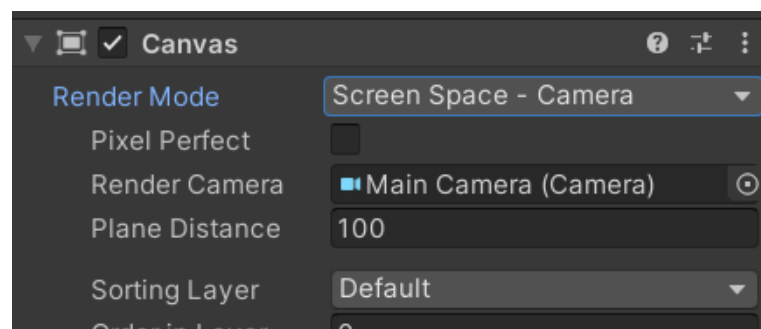


Рисунок 3 – Настройка “Canvas”

Потом в “Canvas” добавим объект “Image” и два объекта “Button”, объект “Image” задаёт фон, а объекты “Button” для размещения кнопок (рис. 4).

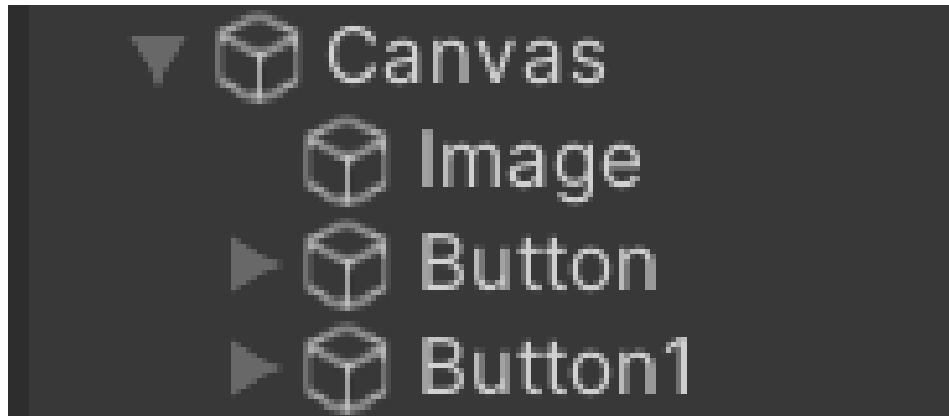


Рисунок 4 –Элементов главного меню

В после создания данных объектов и их настроек меню должно выглядеть так (рис. 5).

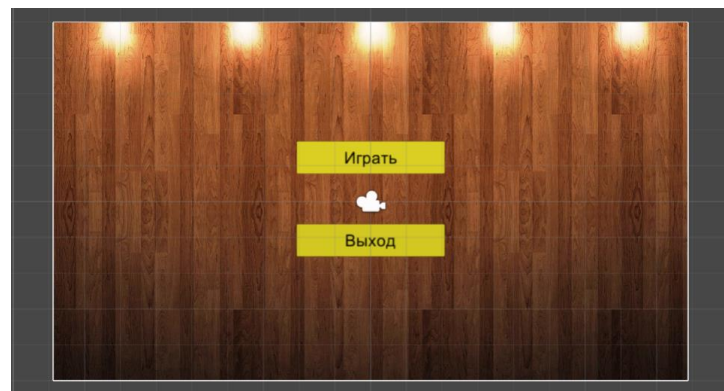


Рисунок 5 – Вид меню

Код кнопок для начала игры и для выхода из нее (рис. 6).

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

Скрипт Unity | Ссылки: 0
public class MainMenu : MonoBehaviour
{
    Ссылки: 0
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
    Ссылки: 0
    public void ExitGame()
    {
        Application.Quit();
    }
}
```

Рисунок 6 – Код кнопок меню

Создаем вторую сцену, и создаем в ней “Create Empty” в котором в дальнейшем и будем прописывать весь код (рис. 7).

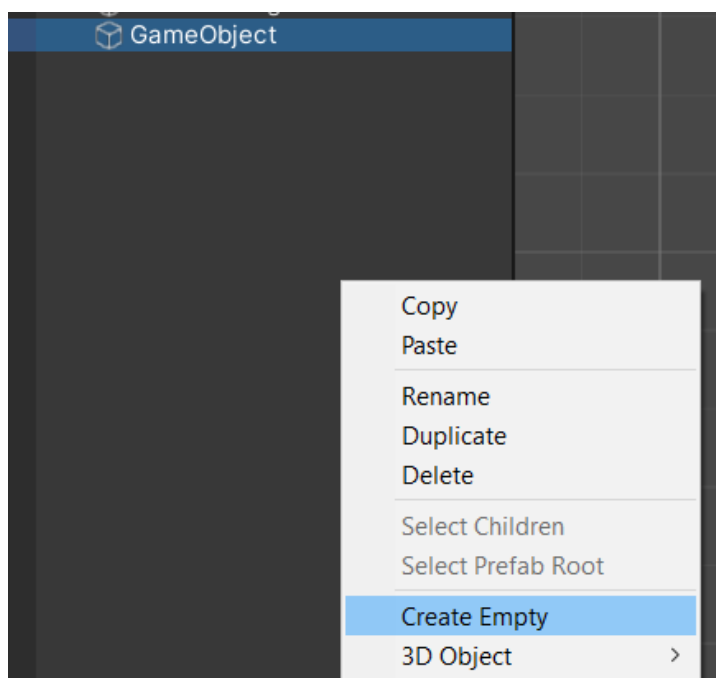


Рисунок 7 – Создания объекта “Create Empty”

В настройках Create Empty “Create Empty” добавляем компонент “New script” (рис. 8).

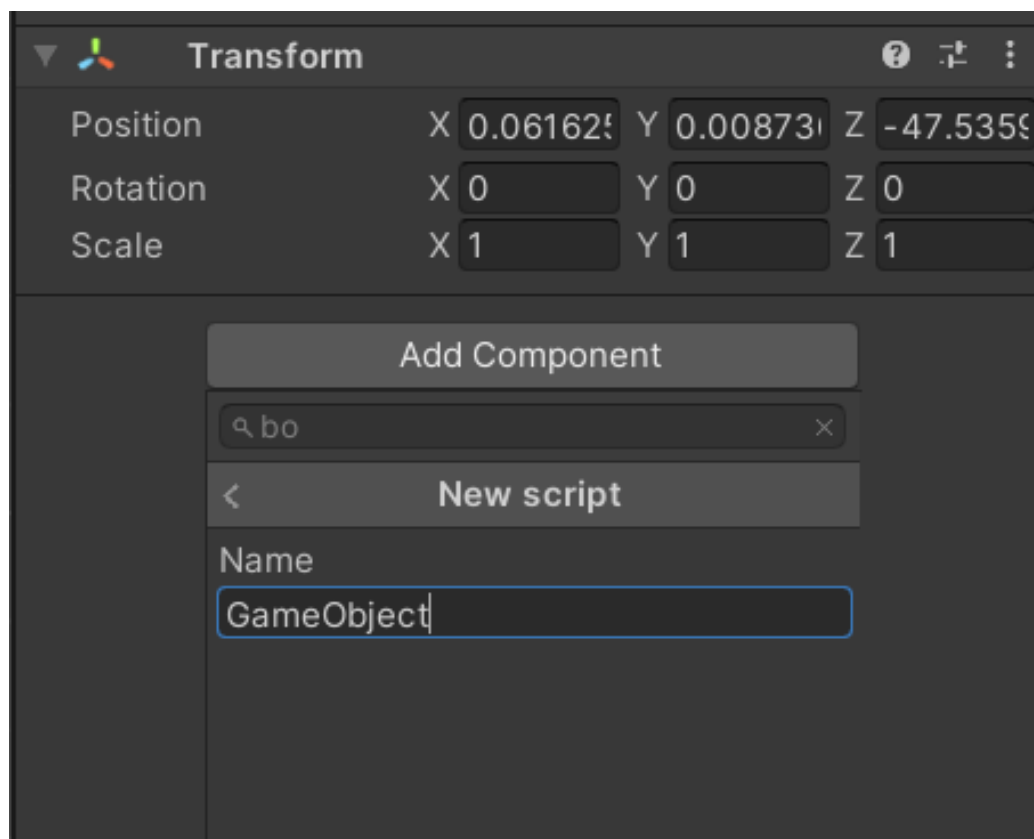


Рисунок 8 – Создания скрипта

Создадим переменные для дальнейшей работы (рис. 9).

```

public class GameManager : MonoBehaviour
{
    [Header("Game Elements")]
    [Range(2, 8)]
    [SerializeField] private int difficulty = 4;
    [SerializeField] private Transform gameHolder;
    [SerializeField] private Transform piecePrefab;

    [Header("UI Elements")]
    [SerializeField] private List<Texture2D> imageTextures;
    [SerializeField] private Transform levelSelectPanel;
    [SerializeField] private Image levelSelectPrefab;
    [SerializeField] private GameObject playAgainButton;
    [SerializeField] private GameObject vmenuButton;
    [SerializeField] private GameObject Menu;

    private List<Transform> pieces;
    private Vector2Int dimensions;
    private float width;
    private float height;

    private Transform draggingPiece = null;
    private Vector3 offset;

    private int piecesCorrect;
}

```

Рисунок 9 – Переменные

Данный фрагмент кода убирает посторонние объекты с экрана, и запускает функции для создания пазлов на основе выбранной картинки, рамки в которой и надо будет собирать эту картинку, а так же кнопки для выхода.

```

// Метод запуска игры с выбранной текстурой
Ссылка: 1
public void StartGame(Texture2D jigsawTexture)
{
    // Скрываем панель выбора уровня
    levelSelectPanel.gameObject.SetActive(false);
    // Включается кнопка выхода в меню
    vmenuButton.SetActive(true);
    Menu.SetActive(false);
    // Инициализируем список для хранения частей пазла
    pieces = new List<Transform>();
    // Рассчитываем размерность сетки пазла на основе текстуры и сложности
    dimensions = GetDimensions(jigsawTexture, difficulty);
    // Создаем части пазла
    CreateJigsawPieces(jigsawTexture);
    // Разбрасываем части по экрану
    Scatter();
    // Обновляем отображение границ пазла
    UpdateBorder();
    // Сбрасываем счетчик правильно собранных частей
    piecesCorrect = 0;
}

```

Рисунок 10 – Функция “StartGame”

При выборе изображением пользователя, узнает высоту и ширину изображения, и на основе этого создает сетку частей, где разрезает изображения на равные части. (рис. 11).

```
void CreateJigsawPieces(Texture2D jigsawTexture)
{
    // Вычисляем высоту и ширину одной части в локальных координатах
    height = 1f / dimensions.y;
    float aspect = (float)jigsawTexture.width / jigsawTexture.height;
    width = aspect / dimensions.x;
    // Двойной цикл для создания сетки частей
    for (int row = 0; row < dimensions.y; row++)
    {
        for (int col = 0; col < dimensions.x; col++)
        {
            Transform piece = Instantiate(piecePrefab, gameHolder);
            piece.localPosition = new Vector3(
                (-width * dimensions.x / 2) + (width * col) + (width / 2),
                (-height * dimensions.y / 2) + (height * row) + (height / 2),
                -1);
            piece.localScale = new Vector3(width, height, 1f);

            piece.name = $"Piece{(row * dimensions.x) + col}";

            pieces.Add(piece);

            float width1 = 1f / dimensions.x;
            float height1 = 1f / dimensions.y;

            Vector2[] uv = new Vector2[4];
            uv[0] = new Vector2(width1 * col, height1 * row);
            uv[1] = new Vector2(width1 * (col + 1), height1 * row);
            uv[2] = new Vector2(width1 * col, height1 * (row + 1));
            uv[3] = new Vector2(width1 * (col + 1), height1 * (row + 1));

            Mesh mesh = piece.GetComponent<MeshFilter>().mesh;
            mesh.uv = uv;

            piece.GetComponent<MeshRenderer>().material.SetTexture("_MainTex", jigsawTexture);
        }
    }
}
```

Рисунок 11 – Функции CreateJigsawPieces

Данный код, определяет рабочую область экрана, что бы пазлы не переместились за область, которая не отображается на экране, а также проверяет место положения всех пазл и переносит из в случайное место на экране (рис. 12).

```
// Разбрасывает части пазла по случайным позициям на экране
Ссылка: 1
private void Scatter()
{
    float orthoHeight = Camera.main.orthographicSize;
    float screenAspect = (float)Screen.width / Screen.height;
    float orthoWidth = (screenAspect * orthoHeight);

    float pieceWidth = width * gameHolder.localScale.x;
    float pieceHeight = height * gameHolder.localScale.y;

    orthoHeight -= pieceHeight;
    orthoWidth -= pieceWidth;

    foreach (Transform piece in pieces)
    {
        float x = Random.Range(-orthoWidth, orthoWidth);
        float y = Random.Range(-orthoHeight, orthoHeight);
        piece.position = new Vector3(x, y, -1);
    }
}
```

Рисунок 12 – Функция Scatter

Определяет границы выбранного изображения и по четырем точкам рисует границы, тем самым и создает рамку в котором надо собирать изображения (рис. 13).

```
// Рисует границу вокруг области сборки пазла
Ссылка: 1
private void UpdateBorder()
{
    LineRenderer lineRenderer = gameHolder.GetComponent<LineRenderer>();

    float halfWidth = (width * dimensions.x) / 2f;
    float halfHeight = (height * dimensions.y) / 2f;

    float borderZ = 0f;

    lineRenderer.SetPosition(0, new Vector3(-halfWidth, halfHeight, borderZ));
    lineRenderer.SetPosition(1, new Vector3(halfWidth, halfHeight, borderZ));
    lineRenderer.SetPosition(2, new Vector3(halfWidth, -halfHeight, borderZ));
    lineRenderer.SetPosition(3, new Vector3(-halfWidth, -halfHeight, borderZ));

    lineRenderer.startWidth = 0.1f;
    lineRenderer.endWidth = 0.1f;

    lineRenderer.enabled = true;
}
```

Рисунок 13 – Функция UpdateBorder

Код реализует базовый цикл интерактивного перетаскивания объектов с помощью мыши в Unity. Он обрабатывает три фазы взаимодействия: захват фрагмента пазла при нажатии кнопки с расчетом смещения, непрерывное обновление его позиции при удержании и завершение перетаскивания с вызовом логики проверки корректности сборки при отпускании кнопки. (рис. 14).

```
// Update вызывается каждый кадр, обрабатывает ввод и перетаскивание
Сообщение Unity | Ссылка: 0
void Update()
{
    // Проверяем нажатие левой кнопки мыши
    if (Input.GetMouseButtonDown(0))
    {
        RaycastHit2D hit = Physics2D.Raycast(Camera.main.ScreenToWorldPoint(Input.mousePosition), Vector2.zero);
        if (hit)
        {
            // Запоминаем часть, которую начали перетаскивать
            draggingPiece = hit.transform;
            // Вычисляем смещение между курсором и центром части
            offset = draggingPiece.position - Camera.main.ScreenToWorldPoint(Input.mousePosition);
            offset += Vector3.back;
        }
    }

    // Проверяем отпускание левой кнопки мыши
    if (draggingPiece && Input.GetMouseButtonUp(0))
    {
        SnapAndDisableIfCorrect();
        draggingPiece.position += Vector3.forward;
        draggingPiece = null;
    }

    // Если часть перетаскивается, обновляем ее позицию
    if (draggingPiece)
    {
        Vector3 newPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        newPosition += offset;
        draggingPiece.position = newPosition;
    }
}
```

Рисунок 14 – Функция Update

Данный метод реализует ключевую игровую механику верификации и фиксации фрагментов пазла. Алгоритм вычисляет целевую позицию для перетаскиваемого фрагмента на основе его индекса в сетке, после чего проверяет, находится ли текущее положение фрагмента в пределах допустимой погрешности; при успешной проверке фрагмент фиксируется в правильной позиции и становится не активным (рис. 15).

```
// Проверяет, находится ли часть близко к правильной позиции, и фиксирует ее
Ссылка: 1
private void SnapAndDisableIfCorrect()
{
    int pieceIndex = pieces.IndexOf(draggingPiece);
    int col = pieceIndex % dimensions.x;
    int row = pieceIndex / dimensions.x;

    Vector2 targetPosition = new Vector2(
        (-width * dimensions.x / 2) + (width * col) + (width / 2),
        (-height * dimensions.y / 2) + (height * row) + (height / 2));

    if (Vector2.Distance(draggingPiece.localPosition, targetPosition) < (width / 2))
    {
        draggingPiece.localPosition = targetPosition;
        draggingPiece.GetComponent<BoxCollider2D>().enabled = false;
        piecesCorrect++;

        if (piecesCorrect == pieces.Count)
        {
            playAgainButton.SetActive(true);
            vmenuButton.SetActive(false);
        }
    }
}
```

Рисунок 15 – Функция SnapAndDisableIfCorrect

На данном рисунке показаны все имеющиеся пазлы, которые можно собрать, пользователю надо выбрать картинку, и на ее основе создастся пазл (рис. 16).

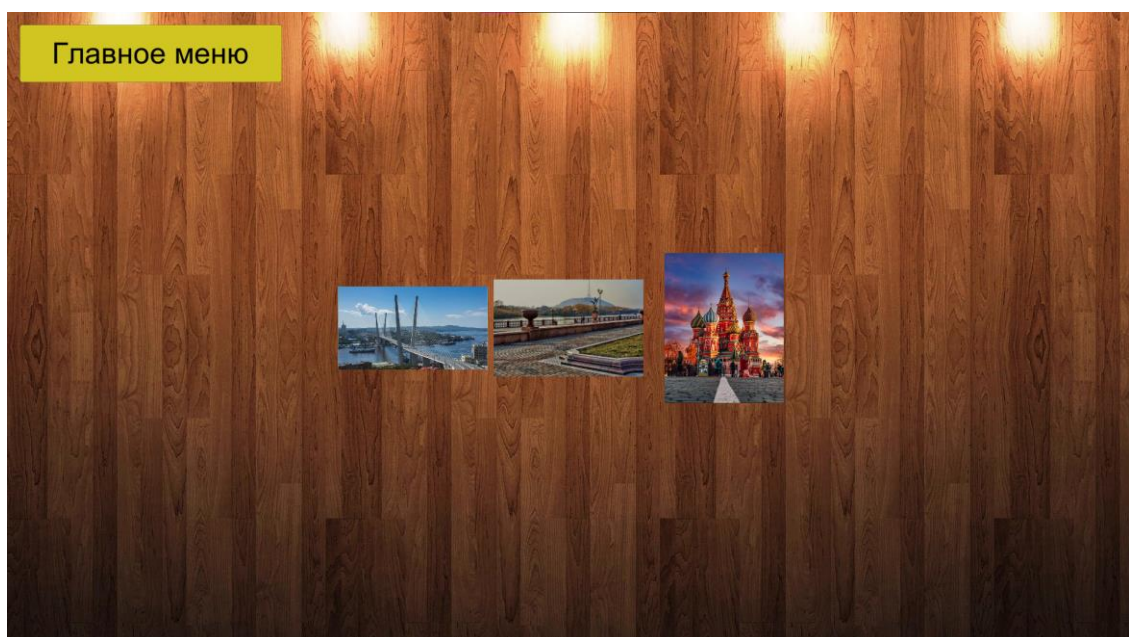


Рисунок 16 – Экран выбора пазлов

После выбора картинки, на экране создается рамка, в которой, надо собрать изображения, а также пазлы, которые созданы на основе выбранного изображения (рис. 17).

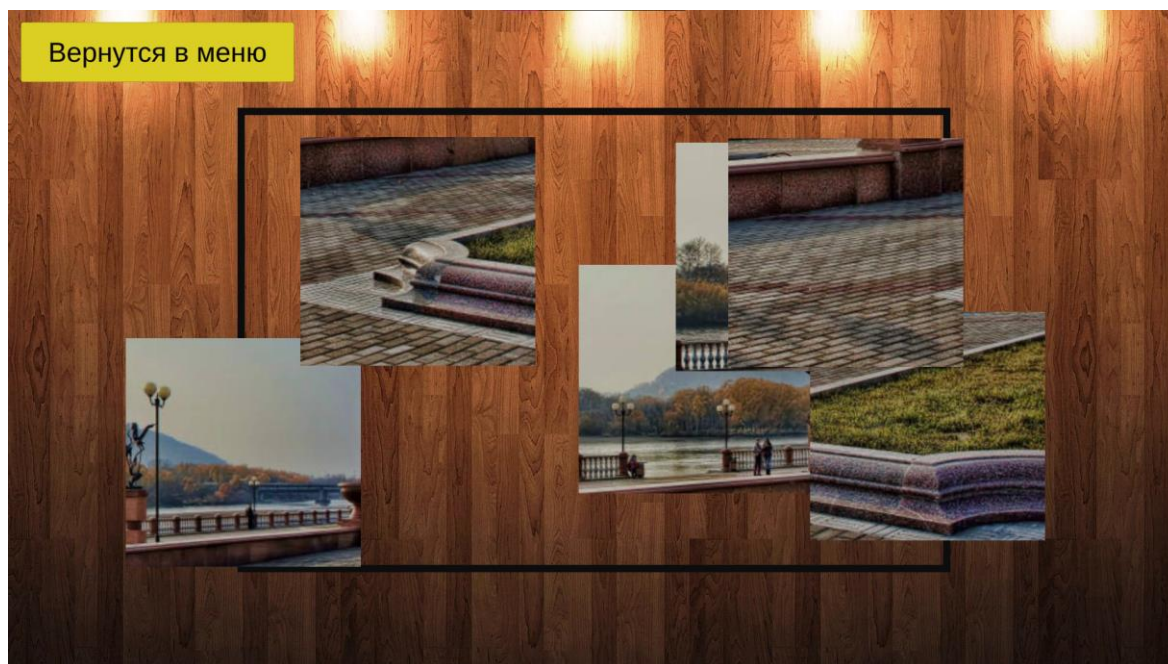


Рисунок 17 - Уровень сбора пазлов

После сбора пазл кнопка «Вернуться в меню» становится не видимой, и появляются кнопка «Закончить»

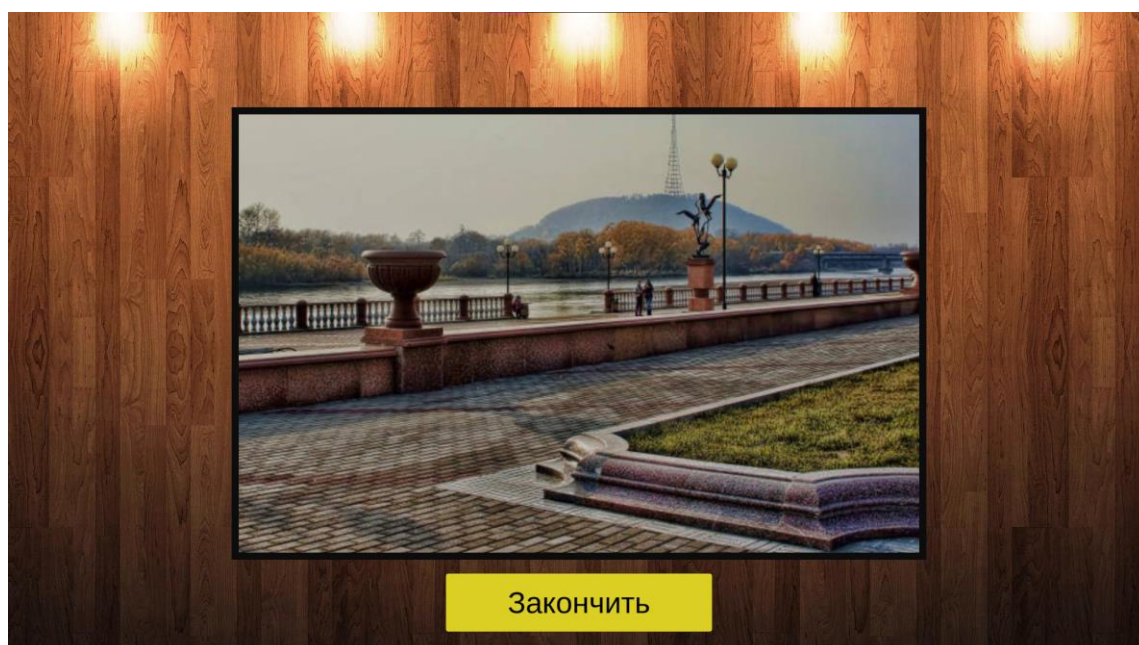


Рисунок 18 – Конец уровня

4. Вывод

В ходе работы был успешно разработан и реализован функциональный прототип игры-головоломки типа «пазл» на игровом движке Unity. Процесс

разработки продемонстрировал практическое применение ключевых инструментов и методик создания 2D-игр: от настройки пользовательского интерфейса с использованием UI-элементов до программирования базовой игровой логики на C#.

Были последовательно решены основные задачи: создано интерактивное главное меню, реализован алгоритм динамического разрезания исходного изображения на фрагменты, обеспечен механизм их случайного размещения на поле и интерактивного перетаскивания с помощью мыши. Дополнительно были созданы визуальные элементы, такие как рамка.

Полученный прототип подтверждает, что использование Unity предоставляет комплексное и эффективное решение для быстрой разработки игр-головоломок. Реализованная архитектура демонстрирует работоспособность базовых механик жанра. Таким образом, проект служит практической основой и доказательством концепции, на базе которой можно вести дальнейшую разработку полноценного игрового продукта.

Библиографический список

1. Базеева Н. А., Лебедев Д. С. Языки программирования для создания игр // E-Scio. 2019. №. 4 (31). С. 615-620. URL: <https://cyberleninka.ru/article/n/yazyki-programmirovaniya-dlya-sozdaniya-igr>
2. Астахова Т. Н., Бердникова А. А., Кривоногов С. В., Лямин А. С. Разработка компьютерной игры для развития логического мышления у детей 5-6 классов / // Информатизация образования и науки. 2025. № 2(66). С. 45-57. URL: https://www.elibrary.ru/download/elibrary_82293410_79769723.pdf
3. Степченко М. Р. Игровое 2D приложение для развития логики и внимания в жанре головоломка с использованием игрового движка Unity // Время науки: актуальные вопросы, достижения и инновации: сборник статей VII Международной научно-практической конференции. Пенза: 2025. С. 81-83. URL: https://www.elibrary.ru/download/elibrary_82435735_14613931.pdf
4. Любич А. А. Создание 2D-игры «б-у-у-м» на движке // Фундаментальные и прикладные исследования молодых ученых: Сборник научных трудов XV Всероссийской научно-практической конференции студентов, аспирантов и молодых учёных, Новокузнецк. Новокузнецк: 2025. С. 167-169. URL: https://www.elibrary.ru/download/elibrary_82653081_59295327.pdf
5. Матюнин К. А. Создание компьютерных игр и приложений на движке unity непрофессиональными пользователями // Молодые ученые в решении актуальных проблем науки: Сборник материалов Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых. Красноярск: 2019. С. 505-506 URL: https://www.elibrary.ru/download/elibrary_41477230_85961892.pdf