Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

**на итоговый проект «Разработка прототипа игры в жанре платформер»**

**по ДПП ПП «Основы Gamedev и VR-разработки»**

| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| --- | --- | --- | --- |
| 1 | Изучить основные механики и особенности жанра платформер, а также успешные примеры и подходы к дизайну уровней и управления персонажем. | Никулинская Я.  Демченко Е.  Цыкунова В. Федячкина К.  Маджара В.В. | 02.05.2025 – 13.05.2025 |
| 2 | Реализовать ключевые игровые элементы: прыжки, движение, сбор предметов и взаимодействие с препятствиями. | 14.05.2025 – 18.05.2025 |
| 3 | Сделать выводы исходя из результатов работы | 19.05.2025 – 26.05.2025 |
| 4 | Подготовка отчета и видео-презентации | 26.05.2025 – 30.05.2025 |

Руководитель проекта   
к. ф.-м. н, доцент Осыкин Д.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / «\_\_\_ » \_\_\_\_\_\_\_\_\_ 2025 г.

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

канд. физ.-мат. наук, доцент Козлов Д.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г.

(подпись)

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

Отчет о выполнении группового итогового проекта по ДПП ПП

«Основы Gamedev и VR-разработки»

**«Разработка прототипа игры в жанре платформер»**

Исполнители:

Никулинская Я.

Демченко Е.

Цыкунова В.

Федячкина К.

Маджара В.В.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта

к. ф.-м. н, доцент Осыкин Д.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

# Цель проекта

Целью данного проекта является разработка играбельного прототипа игры в жанре 3D платформера, демонстрирующего ключевые игровые механики, в частности управление персонажем, прыжки и преодоление препятствий, на примере приключений хомяка, который спасает свою планету от инопланетного вторжения, поднимаясь вверх по уровням игры.

# Задачи проекта и исполнители

Для реализации поставленной цели были сформулированы следующие задачи:

1. Проведение предварительного исследования и генерация идей (Никулинская Я., Демченко Е.).
2. Разработка и программирование механик движения и камеры персонажа (Маджара В.В., Цыкунова В.).
3. Создание и дизайн игровых уровней (Демченко Е., Федячкина К.).
4. Тестирование и отладка уровней и игровых механик (Федячкина К., Цыкунова В.).
5. Финальная сборка прототипа и презентация проекта (Маджара В.В., Никулинская Я.).

# Актуальность и востребованность проекта

Платформеры остаются одним из самых востребованных игровых жанров благодаря своей простоте и увлекательности. Особенно популярны они среди детской аудитории, что подтверждает высокий интерес к подобным играм среди разработчиков и образовательных учреждений. Прототип демонстрирует потенциал использования игровых механик в развлекательных, образовательных и психологических целях. Игра развивает реакцию, стратегическое мышление и пространственную ориентацию, что делает ее применимой в образовательных программах и игровых зонах детских учреждений.

# Общие сведения о проделанной работе

Этапы выполнения проекта:

Октябрь 2024 – Январь 2025:

На начальном этапе работы было проведено исследование рынка платформеров, а также разработан первоначальный концепт. В это же время началась разработка и настройка движений персонажа и камеры.

Март 2025:

В этом месяце стартовала активная разработка игровых уровней, тестирование и отладка первых рабочих механик.

Апрель – Май 2025:

В течение этого периода были завершены основные этапы разработки уровней, проведены многочисленные тестирования и отладки, направленные на улучшение игровой логики и баланса.

Конец мая 2025:

В завершающей фазе был создан итоговый играбельный прототип, после чего команда приступила к подготовке презентационных материалов.

Используемые инструменты:

\* Язык программирования: C#

\* Игровой движок: Unity

\* Среда разработки: Unity Editor

\* Дополнительные ресурсы и ассеты: официальный магазин Unity

Выбор этих технологий обусловлен удобством и мощностью Unity для быстрой разработки игровых прототипов, а также широкой поддержкой и большим количеством доступных ресурсов и документации.

# Результаты проекта

Основным результатом проекта является полнофункциональный прототип платформера, демонстрирующий основные механики управления и взаимодействия персонажа с игровым окружением. Особое внимание уделено механикам с платформами без коллайдеров, что добавило сложности и потребовало от игрока проявления наблюдательности и быстроты реакции. Несмотря на некоторые технические сложности в реализации управления персонажем, проект успешно демонстрирует все запланированные функции и механики, заложенные в начальном концепте.

**Приложение 1.**Скриншоты

На данном скриншоте изображен основной игровой персонаж — хомяк, расположенный на одной из первых платформ уровня. Представлены начальные элементы игровой среды и общий визуальный стиль игры. (Рис. 1)



Рис. 1 – Скриншот с персонажем-хомяком

Данный скриншот демонстрирует игровую механику использования телепортов. Телепорты являются ключевым элементом уровня и служат для перемещения персонажа между различными зонами игрового пространства. (Рис. 2)



Рис. 2 – Скриншот с телепортами

На изображении представлен игровой участок с головоломкой, где игроку необходимо переместить камень на специально обозначенную зону, после чего активируется движение платформы, что позволяет хомяку продвинуться дальше по уровню. (Рис. 3)

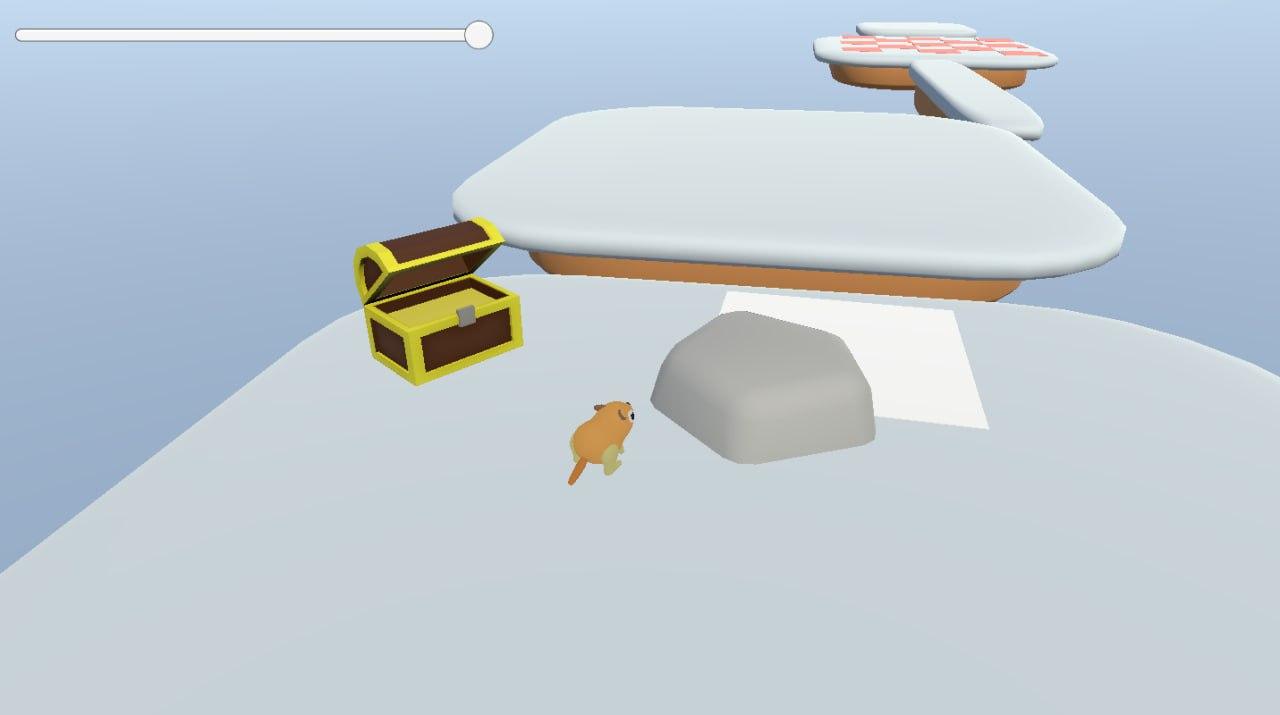


Рис. 3 – Скриншот головоломки с камнем

Скриншот демонстрирует пользовательский интерфейс, отображающийся после гибели персонажа. Игроку предоставляется возможность перезапустить уровень либо вернуться в главное меню игры. (Рис. 4)

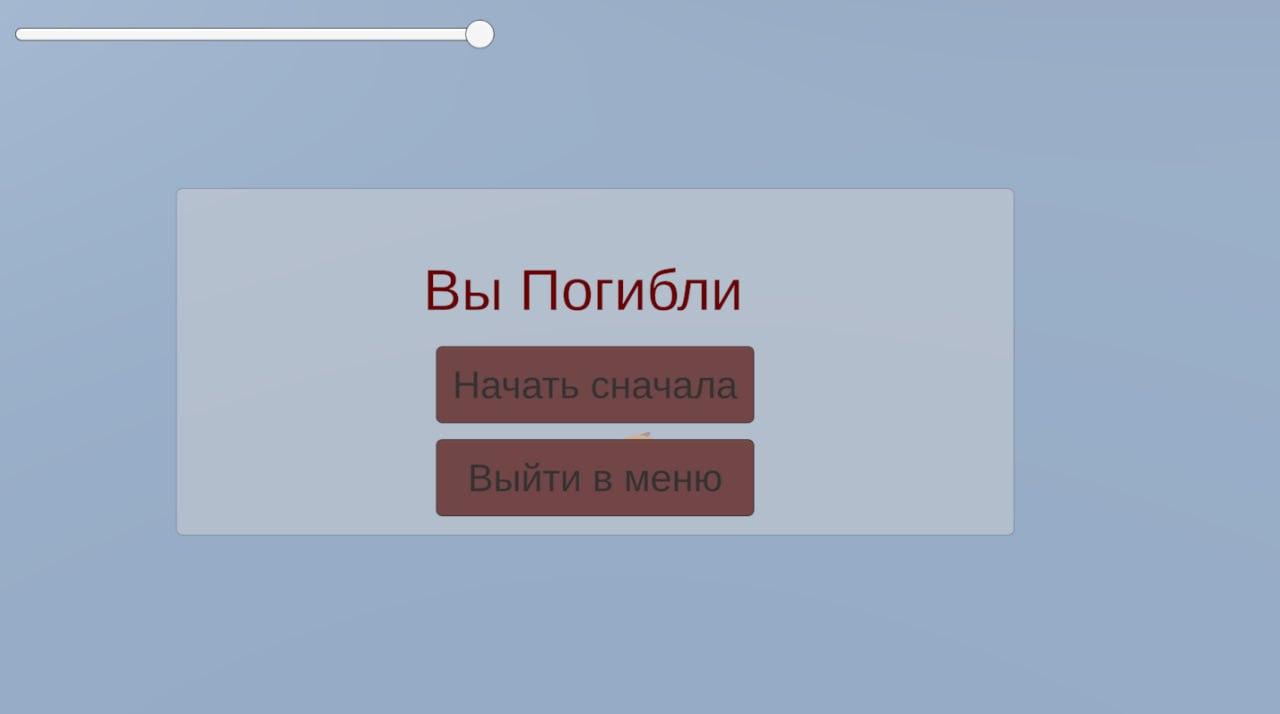


Рис. 4 – Скриншот с интерфейсом о гибели персонажа

На данном изображении представлены платформы, которые находятся в постоянном движении, усложняя прохождение и требуя от игрока расчета времени прыжков и точности передвижений. (Рис. 5)

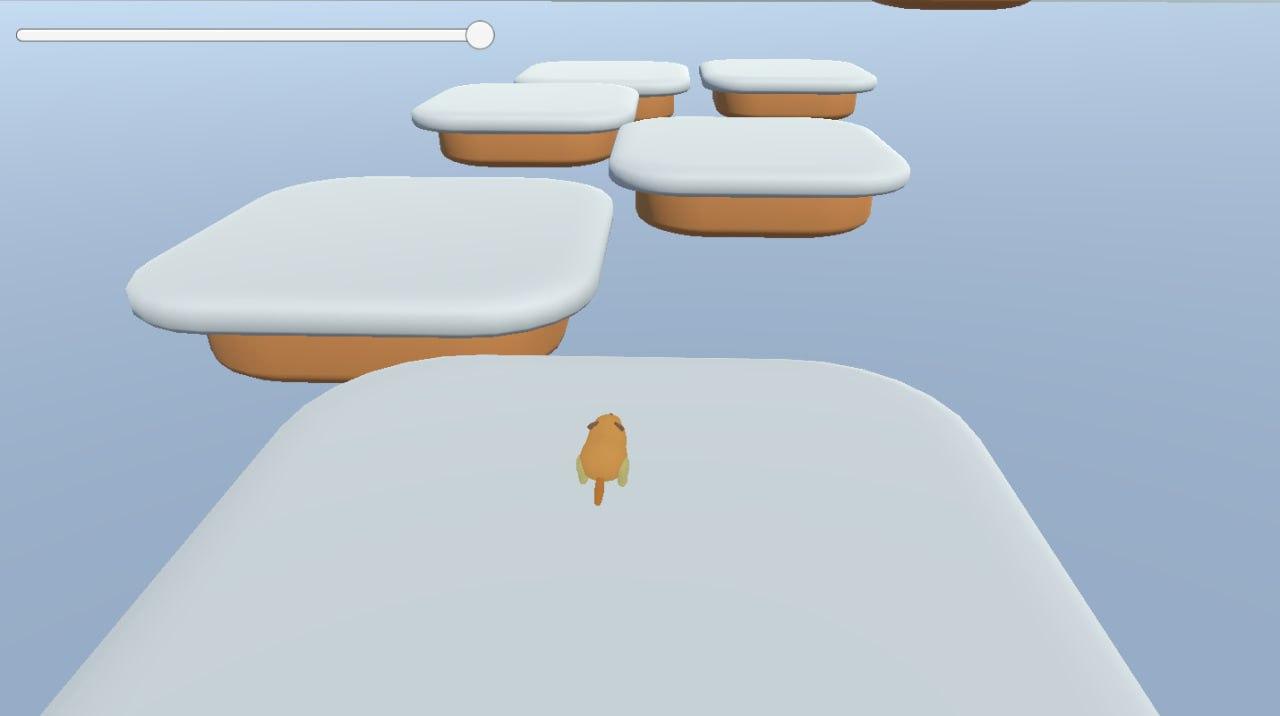


Рис. 5 – Скриншот с движущимися платформами

На заключительном скриншоте представлен пользовательский интерфейс, оповещающий игрока о успешном завершении уровня. Игроку предлагаются опции для перехода на следующий уровень, возврата в меню или выхода из игры. (Рис. 6)

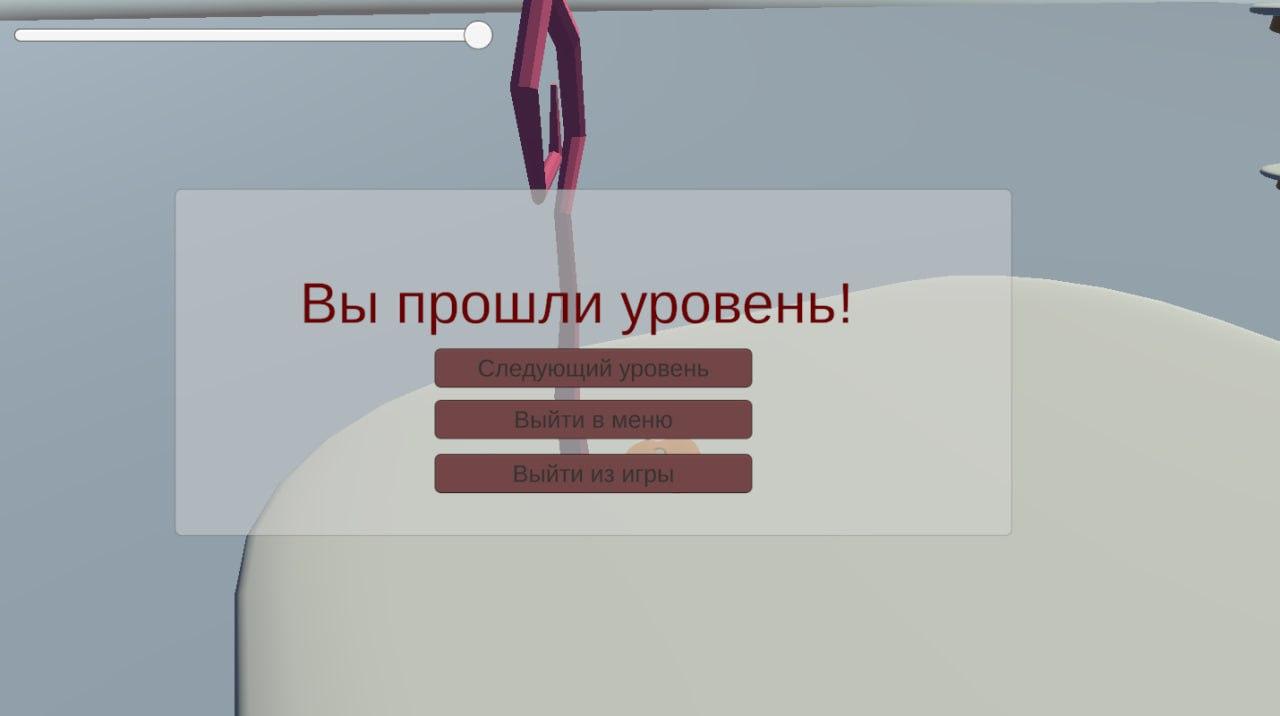


Рис. 6 – Скриншот успешного прохождения уровня

**Приложение 2.**

Программный код

PlayerController.cs  
public class PlayerController : MonoBehaviour

{

public Transform rabbitMesh;

public Camera playerCamera;

public float moveSpeed = 5f;

public float jumpForce = 5f;

public Transform groundCheck;

public LayerMask groundLayer;

public Transform projectileSpawnPoint;

private Rigidbody rb;

private bool isGrounded;

private float groundCheckRadius = 0.5f;

public float maxHealth = 100f;

private float currentHealth;

public Slider healthSlider;

private Animator animator;

public AudioClip runSound;

public AudioClip damageSound;

public AudioClip jumpSound;

private AudioSource audioSource;

private bool isRunning = false;

public GameOverUIController gameOverUIController;

private Vector3 spawnPoint;

private bool isDead = false;

void Start()

{

rb = GetComponent<Rigidbody>();

animator = transform.Find("Muskrat\_LOD0").GetComponent<Animator>();

audioSource = GetComponent<AudioSource>();

currentHealth = maxHealth;

spawnPoint = transform.position;

if (healthSlider != null)

{

healthSlider.maxValue = maxHealth;

healthSlider.value = currentHealth;

}

}

void Update()

{

if (isDead) return;

isGrounded = Physics.CheckSphere(groundCheck.position, groundCheckRadius, groundLayer);

if (animator != null)

{

animator.SetBool("IsGrounded", isGrounded);

}

float moveX = Input.GetAxis("Horizontal");

float moveZ = Input.GetAxis("Vertical");

Vector3 forward = playerCamera.transform.forward;

Vector3 right = playerCamera.transform.right;

forward.y = 0f;

right.y = 0f;

forward.Normalize();

right.Normalize();

Vector3 move = (forward \* moveZ + right \* moveX).normalized;

if (animator != null)

{

animator.SetFloat("Speed", move.magnitude \* moveSpeed);

}

if (move.magnitude > 0.1f && isGrounded && !isRunning)

{

isRunning = true;

PlayRunSound();

}

else if ((!isGrounded || move.magnitude <= 0.1f) && isRunning)

{

isRunning = false;

StopRunSound();

}

MoveCharacter(move);

if (isGrounded && Input.GetButtonDown("Jump"))

{

Jump();

if (animator != null)

{

animator.SetTrigger("Jump");

}

}

}

private void MoveCharacter(Vector3 move)

{

if (move.magnitude > 0.1f)

{

Quaternion targetRotation = Quaternion.LookRotation(move);

transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime \* 10f);

if (rabbitMesh != null)

{

rabbitMesh.rotation = Quaternion.Slerp(rabbitMesh.rotation, targetRotation, Time.deltaTime \* 10f);

}

}

Vector3 targetVelocity = move \* moveSpeed;

targetVelocity.y = rb.linearVelocity.y;

rb.linearVelocity = targetVelocity;

}

private void Jump()

{

rb.AddForce(Vector3.up \* jumpForce, ForceMode.Impulse);

if (jumpSound != null && audioSource != null)

{

audioSource.PlayOneShot(jumpSound);

}

}

public void TakeDamage(float damage)

{

if (isDead) return;

currentHealth -= damage;

if (currentHealth < 0) currentHealth = 0;

if (damageSound != null)

{

AudioSource.PlayClipAtPoint(damageSound, transform.position);

}

if (animator != null)

{

animator.SetTrigger("Hit");

Invoke("ResetHitTrigger", 0.5f);

}

if (currentHealth == 0)

{

Die();

}

if (healthSlider != null)

{

healthSlider.value = currentHealth;

}

}

private void ResetHitTrigger()

{

if (animator != null)

{

animator.ResetTrigger("Hit");

}

}

public void Heal(float healAmount)

{

currentHealth += healAmount;

if (currentHealth > maxHealth) currentHealth = maxHealth;

if (healthSlider != null)

{

healthSlider.value = currentHealth;

}

}

private void Die()

{

isDead = true;

if (animator != null)

{

animator.SetTrigger("Death");

}

if (gameOverUIController != null)

{

gameOverUIController.ShowGameOverUI();

}

rb.isKinematic = true;

}

public void Respawn()

{

transform.position = spawnPoint;

rb.isKinematic = false;

currentHealth = maxHealth;

isDead = false;

if (animator != null)

{

animator.ResetTrigger("Death");

}

if (healthSlider != null)

{

healthSlider.value = currentHealth;

}

}

public void SetSpawnPoint(Vector3 newSpawnPoint)

{

spawnPoint = newSpawnPoint;

}

private void PlayRunSound()

{

if (runSound != null && audioSource != null)

{

audioSource.clip = runSound;

audioSource.loop = true;

audioSource.Play();

}

}

private void StopRunSound()

{

if (audioSource.isPlaying)

{

audioSource.Stop();

}

}

}

HealthPotion.cs  
using UnityEngine;

public class HealthPotion : MonoBehaviour

{

public float healAmount = 20f; // Количество здоровья, которое восстанавливает этот объект

public AudioClip healSound; // Звук, который проигрывается при лечении

public AudioClip crunchSound; // Звук хруста при уничтожении

public float moveHeight = 0.5f; // Высота, на которую будет подниматься/опускаться объект

public float moveSpeed = 1f; // Скорость движения вверх/вниз

private Vector3 startPosition; // Начальная позиция аптечки

private bool movingUp = true; // Флаг, указывающий, двигается ли аптечка вверх или вниз

void Start()

{

// Сохраняем начальную позицию аптечки

startPosition = transform.position;

}

void Update()

{

// Двигаем аптечку вверх-вниз

if (movingUp)

{

transform.position = Vector3.MoveTowards(transform.position, startPosition + Vector3.up \* moveHeight, moveSpeed \* Time.deltaTime);

if (transform.position.y >= startPosition.y + moveHeight)

{

movingUp = false; // Меняем направление на опускание

}

}

else

{

transform.position = Vector3.MoveTowards(transform.position, startPosition - Vector3.up \* moveHeight, moveSpeed \* Time.deltaTime);

if (transform.position.y <= startPosition.y - moveHeight)

{

movingUp = true; // Меняем направление на поднятие

}

}

}

private void OnTriggerEnter(Collider other)

{

// Проверяем, что объект, который вошел в триггер, является игроком

if (other.CompareTag("Player"))

{

// Получаем компонент PlayerController на игроке

PlayerController player = other.GetComponent<PlayerController>();

if (player != null)

{

player.Heal(healAmount); // Восстанавливаем здоровье игрока

Debug.Log("Игрок восстановил здоровье на " + healAmount);

// Проигрываем звук, если он задан

if (healSound != null)

{

AudioSource.PlayClipAtPoint(healSound, transform.position);

}

// Проигрываем звук хруста перед уничтожением

if (crunchSound != null)

{

AudioSource.PlayClipAtPoint(crunchSound, transform.position);

}

// Уничтожаем аптечку после использования

Destroy(gameObject);

}

}

}

}

BouncePlatform.cs

using UnityEngine;

public class BouncePlatform : MonoBehaviour

{

public float bounceForce = 10f;

public AudioClip bounceSound;

private AudioSource audioSource;

void Start()

{

audioSource = GetComponent<AudioSource>();

}

private void OnCollisionEnter(Collision collision)

{

if (collision.collider.CompareTag("Player"))

{

Rigidbody playerRb = collision.collider.GetComponent<Rigidbody>();

if (playerRb != null)

{

Vector3 velocity = playerRb.linearVelocity;

velocity.y = 0f;

playerRb.linearVelocity = velocity;

playerRb.AddForce(Vector3.up \* bounceForce, ForceMode.Impulse);

if (bounceSound != null && audioSource != null)

{

audioSource.PlayOneShot(bounceSound);

}

Debug.Log("Прыжок!");

}

}

}

}

GameOverUIController.cs  
using UnityEngine;

using UnityEngine.UI;

public class GameOverUIController : MonoBehaviour

{

public GameObject gameOverPanel;

public Text gameOverText;

public Button restartButton;

public PlayerController player;

void Start()

{

gameOverPanel.SetActive(false);

restartButton.onClick.AddListener(RestartFromCheckpoint);

}

public void ShowGameOverUI()

{

gameOverPanel.SetActive(true);

gameOverText.text = "Вы погибли!";

}

private void RestartFromCheckpoint()

{

gameOverPanel.SetActive(false);

player.Respawn();

}

}

SpawnPoint.cs  
using UnityEngine;

public class SpawnPoint : MonoBehaviour

{

private bool isActivated = false;

private void OnTriggerEnter(Collider other)

{

if (!isActivated && other.CompareTag("Player"))

{

PlayerController player = other.GetComponent<PlayerController>();

if (player != null)

{

player.SetSpawnPoint(transform.position);

isActivated = true;

Debug.Log($"Спавнпоинт активирован на позиции {transform.position}");

}

}

}

}

KillZone.cs

using UnityEngine;

public class KillZone : MonoBehaviour

{

private void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Player"))

{

PlayerController player = other.GetComponent<PlayerController>();

if (player != null)

{

player.TakeDamage(player.maxHealth); // ХП в ноль

}

}

}

}

**Приложение 3.**

Репозиторий

Ссылка на репозиторий:

<https://github.com/vasyaBoss/Humster2025>

**Приложение 4.**

Презентация проекта

Ссылка на презентацию: <https://docs.google.com/presentation/d/1Uvz7JE5No-Fa26Cr6T6vISku2Wsr-6lp2bRVflr2fnA/edit?usp=sharing>