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 2D-платформера «Хроники Ржавого» по ДПП ПП «****Основы Gamedev и VR разработки»**

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Определение основных механик и прототипирование. | Почёмин И.М.,  Черепанов М.В... | 02.06.2025 – 13.06.2025 |
| 2 | Разработка базового уровня и врагов | 14.06.2025 – 18.06.2025 |
| 3 | Реализация системы навыков и улучшение боевой системы | 19.06.2025 – 22.06.2025 |
| 4 | Тестирование прототипа | 22.06.2025 – 24.06.2025 |
| 5 | Подготовка отчета и видео-презентации | 24.06.2025 – 25.06.2025 |

Руководитель проекта   
преподаватель 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 2D-платформера «Хроники Ржавого»**

Исполнители:

Черепанов М.В.  
Почёмин И.М.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта

преп. Осыкин Д.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

# Цель проекта

Целью проекта является разработка 2D платформера с элементами RPG в постапокалиптическом сеттинге, предлагающего игрокам геймплей, включающий боевую систему, систему навыков и паркур. Проект направлен на создание прототипа игрового продукта.

# Задачи проекта и исполнители

Для достижения поставленной цели требовалось решить ряд задач:

1. Определение основных механик и прототипирование
2. Разработка базового уровня и врагов
3. Реализация системы навыков и улучшение боевой системы.
4. Тестирование прототипа.

# Актуальность и востребованность проекта

Проект "Хроники Ржавого" актуален и востребован, поскольку он ориентирован на существующий интерес к доступным и увлекательным игровым продуктам. Вот несколько ключевых аспектов, подтверждающих это:

Постоянный интерес к 2D платформерам: Несмотря на развитие графики, 2D платформеры сохраняют свою популярность. Они не требуют мощного оборудования, что расширяет потенциальную аудиторию. Многие игроки ценят этот жанр за возможность сосредоточиться на геймплее и проработке механик, а не на реалистичности графики.

Сбалансированный геймплей с элементами RPG: Включение боевой системы, паркура и системы навыков не просто добавляет глубины, но и предлагает игрокам разнообразный опыт. Это отличает игру от простых платформеров и делает её интересной для тех, кто ценит тактические элементы и развитие персонажа. Наличие различных типов врагов и способностей увеличивает реиграбельность.

В целом, "Хроники Ржавого" представляют собой проект, который опирается на проверенные временем игровые концепции и популярные жанры, предлагая их в новом исполнении. Это делает его потенциально привлекательным для широкого круга игроков.

# Общие сведения о проделанной работе

Разработка "Хроник Ржавого" — это результат длительных усилий и командной работы, которая потребовала значительного вложения времени и ресурсов. Мы привлекли сторонних специалистов, чтобы справиться с объёмом задач и обеспечить качество на всех этапах.

Вот как мы подошли к процессу:

Проектирование и планирование: В самом начале мы продумали, какой должна быть игра. Определили основные игровые механики, нарисовали, как будет выглядеть уровень и составили список врагов и навыков..

Реализация базовых механик: Затем мы перешли к "скелету" игры. Были реализованы основные движения персонажа — ходьба, бег, прыжки. Мы также создали начальные версии паркура и боевой системы, чтобы убедиться, что базовое управление работает.

Создание и интеграция контента: Мы уделили много внимания визуальной части. С помощью Piskel были отрисованы все спрайты персонажа и врагов, а также создана их анимация. Для окружения мы искали и адаптировали подходящие ассеты из Unity Asset Store, чтобы создать атмосферу, не тратя время на рисование всего с нуля. Также были спроектирован и построен игровой уровень, на котором можно продемонстрировать внедрённые в игру механики.

Внедрение ключевых систем: По мере готовности базовых механик, мы интегрировали более сложные системы. Это включало систему скиллов с огненным шаром, поражающим первого врага на пути, молнией с автонаведением и исцелением персонажа. Для каждого типа врагов — обычных зомби, стреляющих врагов и босса с комбинированными атаками и преследованием — был разработан свой искусственный интеллект, чтобы они вели себя реалистично и представляли вызов.

Техническое обеспечение: Мы использовали Unity как основной инструмент разработки. Для совместной работы и контроля за изменениями в коде и ассетах мы активно применяли Unity DevOps. Это позволило команде работать слаженно, быстро интегрировать новые элементы и избегать путаницы.

Тестирование и доработка: На протяжении всего процесса мы постоянно тестировали игру, выявляя и исправляя ошибки. Этот итеративный подход помог нам довести игру до работоспособного состояния, обеспечив стабильность.

В итоге, "Хроники Ржавого" — это 2D платформер, в котором реализованы все заявленные механики, от паркура до боевой системы и системы навыков, а также разнообразные враги.

# Результаты проекта

В ходе выполнения проекта было проведено большое количество испытаний. Так, например, на рисунке 1 представлены первые шесть поколений, в ходе выполнения программы, было замечено существенное продвижение искусственного интеллекта по трассе, ему понадобилось всего 6 поколений, чтобы стабильно ездить по трассе и не врезаться в препятствия.



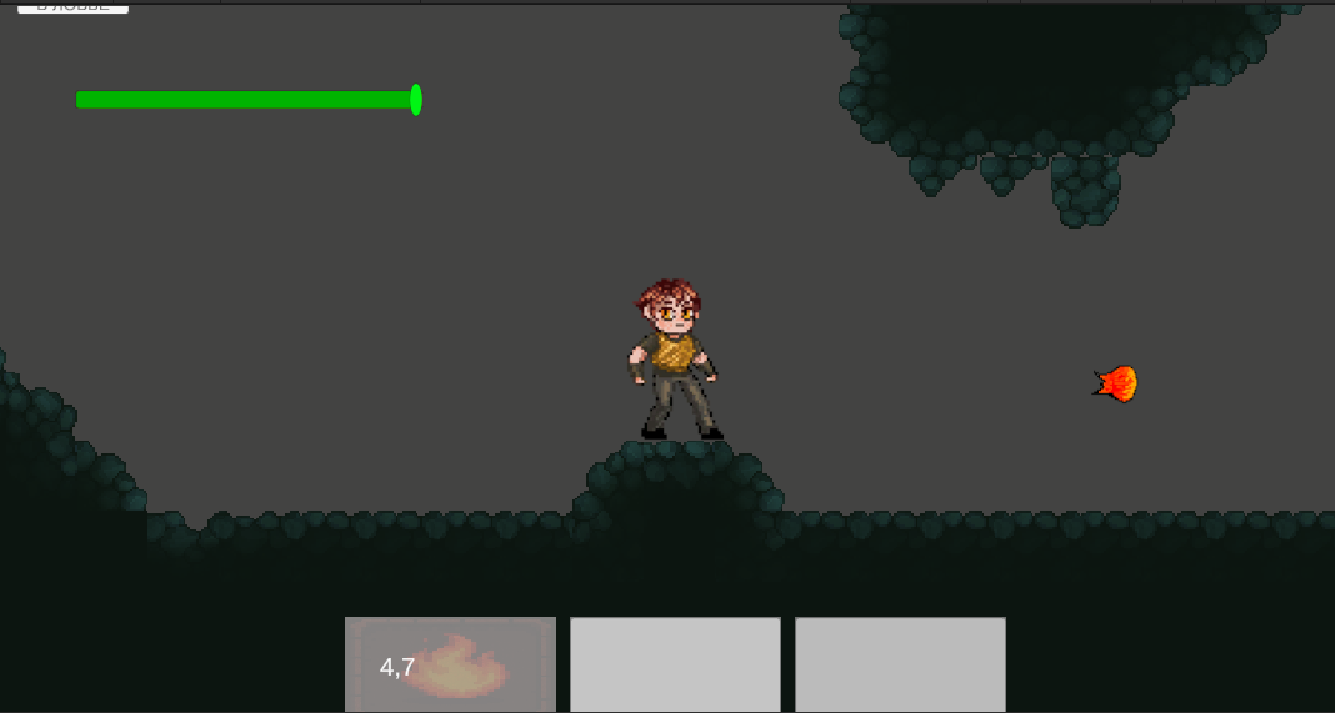
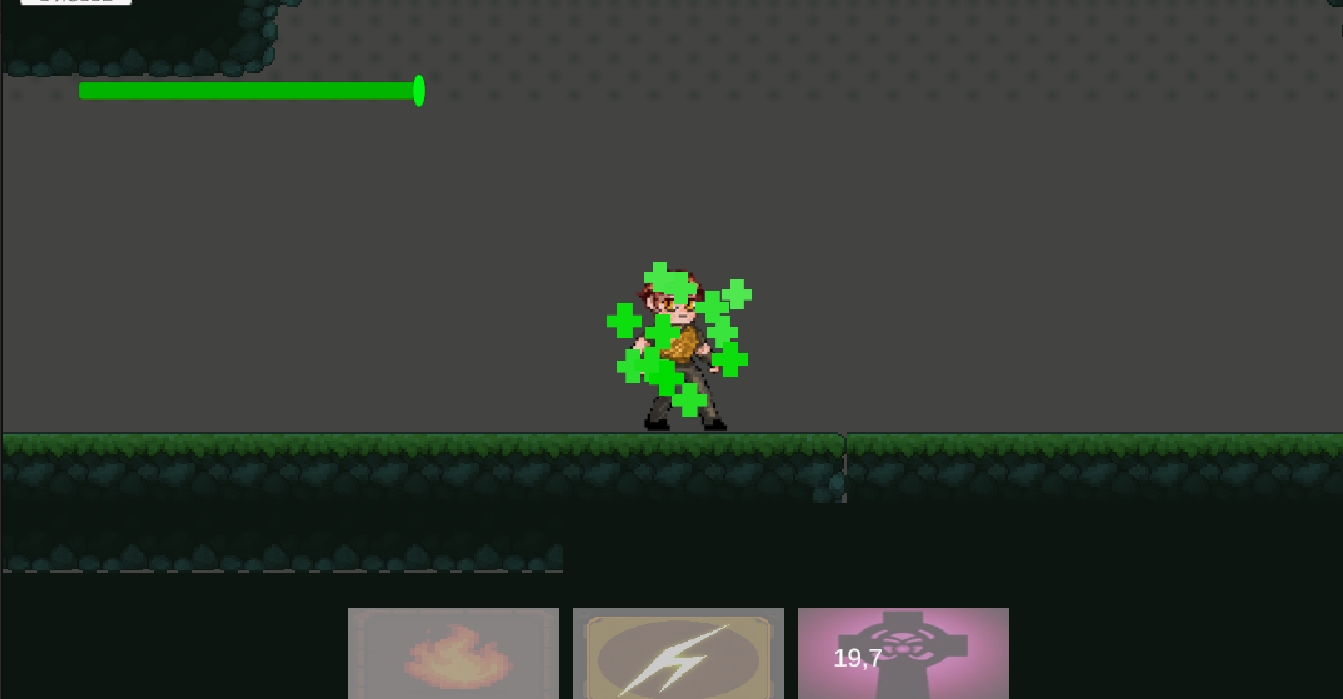


Рисунок 1.





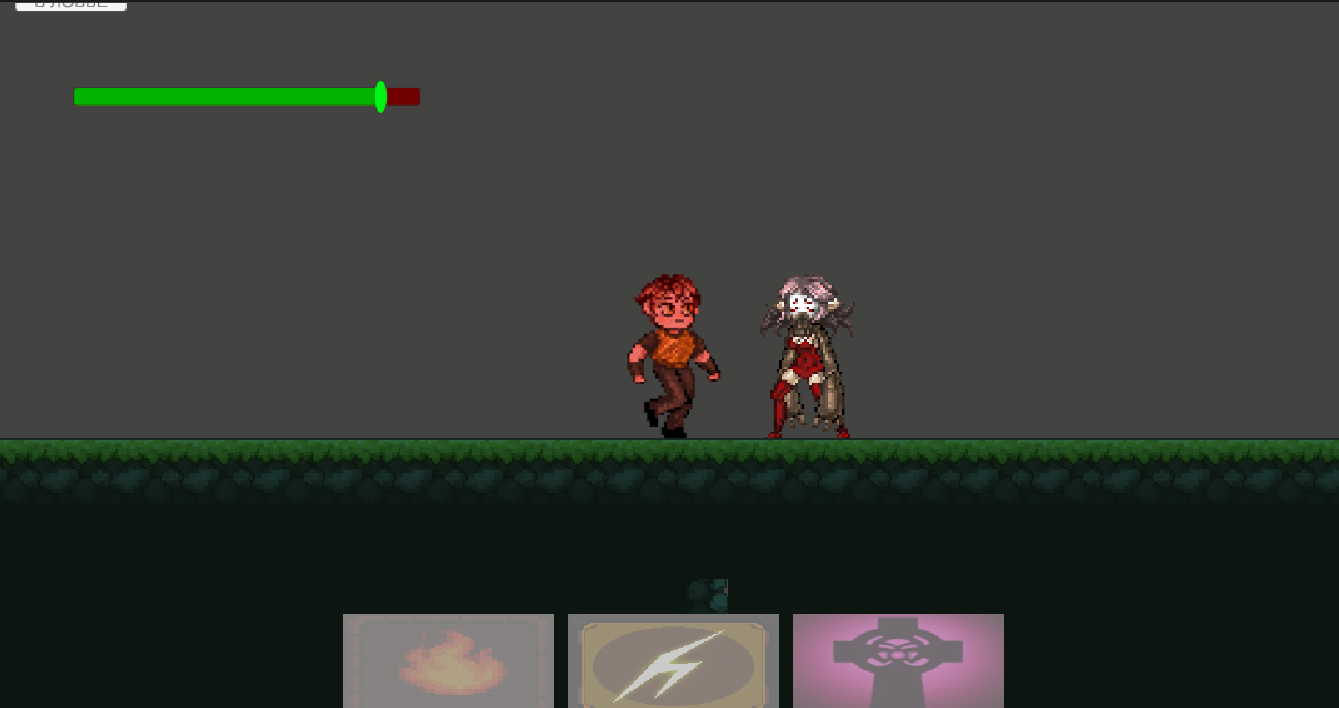


Рисунок 2.

**Приложение 1.**Основополагающие скрипты

1.1 Скрипт для передвижения персонажа (PlayerMovement.cs)

using System;

using Unity.VisualScripting;

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

[Header("Movement Settings")]

public float maxSpeed = 8f;

public float acceleration = 70f;

public float deceleration = 50f;

public float airAcceleration = 30f;

public float inertiaDuration = 0.1f;

[Header("Jump Settings")]

public float jumpForce = 12f;

public float coyoteTime = 0.1f;

public float jumpBufferTime = 0.1f;

public float fallMultiplier = 2f; // Ускорение падения

private float groundedRadius;

[Header("Wall Interaction")]

public float wallSlideSpeed = 2f; // Скорость скольжения по стене

public float wallJumpForceX = 10f; // Горизонтальная сила прыжка от стены

public float wallJumpForceY = 12f; // Вертикальная сила прыжка от стены

public float wallJumpDuration = 0.2f; // Время, когда игрок не может управлять движением после прыжка от стены4

private float leftrightRayDistance;

[Header("Dash Settings")]

public float dashSpeed = 20f;

public float dashDuration = 0.3f;

public float dashCooldown = 1f;

public int maxAirDashes = 1; // Количество рывков в воздухе

private Rigidbody2D rb;

private float moveInput;

private bool isGrounded;

private bool isTouchingWall;

private bool isWallSliding;

private bool isFacingRight = true;

private bool isDashing;

private bool canDash = true;

private int airDashesLeft;

private float lastGroundedTime;

private float lastJumpPressedTime;

private float lastWallJumpTime;

private float lastDashTime;

private float velocityXSmoothing;

public Animator anim;

private Collider2D collider1;

private void Awake()

{

rb = GetComponent<Rigidbody2D>();

airDashesLeft = maxAirDashes;

collider1 = GetComponent<Collider2D>();

groundedRadius = (collider1.bounds.size.y/2)+0.2f;

leftrightRayDistance = (collider1.bounds.size.x/2)+0.2f;

}

private void Update()

{

moveInput = Input.GetAxisRaw("Horizontal");

anim.SetFloat("IsRunning",Math.Abs(rb.linearVelocityX));

// Проверка земли и стен

CheckCollisions();

// Обработка прыжков

HandleJump();

// Обработка рывка (Dash)

if (Input.GetKeyDown(KeyCode.LeftShift))

{

TryDash();

}

// Поворот спрайта

if (moveInput > 0 && !isFacingRight)

{

Flip();

}

else if (moveInput < 0 && isFacingRight)

{

Flip();

}

anim.SetBool("WallSliding", isWallSliding);

// Ускоренное падение

if (rb.linearVelocity.y < 0 && !isGrounded)

{

rb.linearVelocity += Vector2.up \* Physics2D.gravity.y \* (fallMultiplier - 1) \* Time.deltaTime;

}

}

private void FixedUpdate()

{

if (isDashing)

{

gameObject.tag = "IgnoreThisTag";

gameObject.layer = 8;

return; // Не управлять движением во время рывка

}

else {

gameObject.layer = 0;

gameObject.tag = "Player";

}

UpdateMovement();

HandleWallSlide();

}

//=== ОСНОВНОЕ ДВИЖЕНИЕ ===//

private void UpdateMovement()

{

float targetVelocityX = moveInput \* maxSpeed;

float currentAcceleration;

// Если игрок прижат к стене и не скользит, запрещаем движение

if (isTouchingWall && !isGrounded && moveInput != 0 && Mathf.Sign(moveInput) == Mathf.Sign(transform.localScale.x))

{

targetVelocityX = 0;

}

// Выбираем ускорение в зависимости от состояния

if (isGrounded)

{

currentAcceleration = (Mathf.Sign(moveInput) != Mathf.Sign(rb.linearVelocity.x)) ? deceleration : acceleration;

}

else

{

currentAcceleration = airAcceleration;

}

// Плавное изменение скорости

float newVelocityX = Mathf.SmoothDamp(

rb.linearVelocity.x,

targetVelocityX,

ref velocityXSmoothing,

inertiaDuration,

currentAcceleration

);

rb.linearVelocity = new Vector2(newVelocityX, rb.linearVelocity.y);

}

//=== ПРЫЖКИ ===//

private void HandleJump()

{

// Обновляем таймеры

if (isGrounded)

{

lastGroundedTime = coyoteTime;

airDashesLeft = maxAirDashes; // Сброс рывков при приземлении

}

else

{

lastGroundedTime -= Time.deltaTime;

}

// Буфер прыжка

if (Input.GetButtonDown("Jump"))

{

lastJumpPressedTime = jumpBufferTime;

}

else

{

lastJumpPressedTime -= Time.deltaTime;

}

// Обычный прыжок с земли

if (lastJumpPressedTime > 0 && lastGroundedTime > 0)

{

rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);

lastJumpPressedTime = 0;

lastGroundedTime = 0;

anim.SetTrigger("JumpTrigger");

}

// Прыжок от стены

else if (lastJumpPressedTime > 0 && isWallSliding)

{

WallJump();

anim.SetTrigger("JumpTrigger");

}

}

//=== СТЕННОЕ ВЗАИМОДЕЙСТВИЕ ===//

private void HandleWallSlide()

{

if (isTouchingWall && !isGrounded && rb.linearVelocity.y < 0)

{

isWallSliding = true;

rb.linearVelocity = new Vector2(rb.linearVelocity.x, Mathf.Max(-wallSlideSpeed, rb.linearVelocity.y));

}

else

{

isWallSliding = false;

}

}

private void WallJump()

{

float direction = -Mathf.Sign(transform.localScale.x); // Прыжок в противоположную сторону от стены

rb.linearVelocity = new Vector2(direction \* wallJumpForceX, wallJumpForceY);

lastJumpPressedTime = 0;

lastWallJumpTime = wallJumpDuration;

}

//=== РЫВОК (DASH) ===//

private void TryDash()

{

if (!canDash || (isGrounded && airDashesLeft <= 0)) return;

if (!isGrounded)

{

airDashesLeft--;

}

isDashing = true;

canDash = false;

lastDashTime = dashDuration;

//anim.SetBool("IsSliding", true);

Debug.Log("Дешит");

anim.SetTrigger("SlideTrigger");

// Направление рывка (в сторону движения или просто вперёд, если стоит)

float dashDirection = moveInput != 0 ? Mathf.Sign(moveInput) : (isFacingRight ? 1 : -1);

rb.linearVelocity = new Vector2(dashDirection \* dashSpeed, 0);

Invoke(nameof(StopDash), 0.35f);

Invoke(nameof(ResetDash), dashCooldown);

}

private void StopDash()

{

//anim.SetBool("IsSliding", isDashing);

isDashing = false;

Debug.Log("Не дешит");

//anim.SetBool("IsSliding", false);

}

private void ResetDash()

{

canDash = true;

}

//=== ВСПОМОГАТЕЛЬНЫЕ МЕТОДЫ ===//

private void CheckCollisions()

{

// Проверка земли (можно заменить на Raycast или CircleCast)

isGrounded = Physics2D.Raycast(transform.position, Vector2.down, groundedRadius, LayerMask.GetMask("Ground")) || Physics2D.Raycast(transform.position, Vector2.down, groundedRadius, LayerMask.GetMask("EnemyHead"));

// Проверка стены (используем Raycast в сторону, куда смотрит игрок

Vector2 rayDirection = isFacingRight ? Vector2.right : Vector2.left;

isTouchingWall = Physics2D.Raycast(transform.position, rayDirection, leftrightRayDistance, LayerMask.GetMask("Ground"));

}

private void Flip()

{

isFacingRight = !isFacingRight;

Vector3 scale = transform.localScale;

scale.x \*= -1;

transform.localScale = scale;

}

private void OnDrawGizmos()

{

// Визуализация Raycast для стены

Gizmos.color = Color.red;

Vector2 rayDirection = isFacingRight ? Vector2.right : Vector2.left;

Gizmos.DrawLine(transform.position, transform.position + (Vector3)(rayDirection \* leftrightRayDistance));

Gizmos.DrawLine(transform.position, transform.position + (Vector3)(Vector2.down \* groundedRadius));

}

public bool IsFacingRight()

{

return isFacingRight;

}

}

1.2 Скрипт управления навыками (SkillManager.cs)

using UnityEngine;

using System.Collections.Generic;

using System.Linq; // Для использования LINQ

public class SkillManager : MonoBehaviour

{

public static SkillManager Instance; // Singleton для легкого доступа

public GameObject availableSkills;

[Header("Все доступные способности")]

public List<Skill> allAvailableSkills = new List<Skill>();

[Header("Выбранные способности")]

// Словарь для хранения выбранных способностей и их текущей перезарядки

public Dictionary<int, Skill> selectedSkills = new Dictionary<int, Skill>();

private Dictionary<Skill, float> skillCooldowns = new Dictionary<Skill, float>();

[Header("Привязки клавиш")]

public KeyCode[] skillKeys = { KeyCode.Alpha1, KeyCode.Alpha2, KeyCode.Alpha3, KeyCode.Alpha4 };

[Tooltip("Объект, который использует способности (обычно игрок)")]

public GameObject userGameObject;

void Awake()

{

if (Instance == null)

{

Instance = this;

DontDestroyOnLoad(gameObject); // Сохраняем менеджер между сценами

}

else

{

Destroy(gameObject);

}

// Инициализация словарей перезарядки

foreach (var skill in allAvailableSkills)

{

skillCooldowns.Add(skill, 0f);

}

}

void Update()

{

// Отслеживание перезарядки

List<Skill> skillsToUpdate = new List<Skill>(skillCooldowns.Keys);

foreach (var skill in skillsToUpdate)

{

if (skillCooldowns[skill] > 0)

{

skillCooldowns[skill] -= Time.deltaTime;

}

}

// Активация способностей по нажатию клавиш

for (int i = 0; i < skillKeys.Length; i++)

{

if (Input.GetKeyDown(skillKeys[i]))

{

ActivateSelectedSkill(i + 1); // +1, потому что слоты 1, 2, 3, 4

}

}

if (Input.GetKeyDown(KeyCode.Tab))

{

availableSkills.SetActive(true);

}

if (Input.GetKeyUp(KeyCode.Tab))

{

availableSkills.SetActive(false);

}

}

/// <summary>

/// Активирует выбранную способность по номеру слота (1, 2, 3, 4).

/// </summary>

/// <param name="slotNumber">Номер слота способности (1-4).</param>

public void ActivateSelectedSkill(int slotNumber)

{

if (selectedSkills.ContainsKey(slotNumber))

{

Skill skillToActivate = selectedSkills[slotNumber];

if (skillCooldowns[skillToActivate] <= 0)

{

// Проверка маны (если есть)

// if (PlayerStats.Instance.CurrentMana >= skillToActivate.manaCost)

// {

// PlayerStats.Instance.SpendMana(skillToActivate.manaCost);

// skillToActivate.Activate(userGameObject);

// skillCooldowns[skillToActivate] = skillToActivate.cooldown;

// }

// else

// {

// Debug.Log($"Недостаточно маны для использования {skillToActivate.skillName}!");

// }

// Временно без проверки маны

skillToActivate.Activate(userGameObject);

skillCooldowns[skillToActivate] = skillToActivate.cooldown;

}

else

{

Debug.Log($"Способность {skillToActivate.skillName} находится на перезарядке. Осталось: {skillCooldowns[skillToActivate]:F2} сек.");

}

}

else

{

Debug.Log($"Слот {slotNumber} пуст.");

}

}

/// <summary>

/// Добавляет или заменяет способность в выбранном слоте.

/// </summary>

/// <param name="skill">Способность для добавления.</param>

/// <param name="slotNumber">Слот, в который нужно поместить способность (1-4).</param>

public void SelectSkill(Skill skill, int slotNumber)

{

if (slotNumber >= 1 && slotNumber <= 4)

{

if (selectedSkills.ContainsKey(slotNumber))

{

Debug.Log($"Заменяем способность в слоте {slotNumber}: {selectedSkills[slotNumber].skillName} на {skill.skillName}");

selectedSkills[slotNumber] = skill;

}

else

{

Debug.Log($"Добавляем способность {skill.skillName} в слот {slotNumber}");

selectedSkills.Add(slotNumber, skill);

}

}

else

{

Debug.LogError($"Неверный номер слота: {slotNumber}. Должен быть от 1 до 4.");

}

}

/// <summary>

/// Убирает способность из выбранного слота.

/// </summary>

/// <param name="slotNumber">Слот, из которого нужно убрать способность (1-4).</param>

public void DeselectSkill(int slotNumber)

{

if (selectedSkills.ContainsKey(slotNumber))

{

Debug.Log($"Убираем способность из слота {slotNumber}: {selectedSkills[slotNumber].skillName}");

selectedSkills.Remove(slotNumber);

}

}

/// <summary>

/// Получить текущую перезарядку для способности.

/// </summary>

/// <param name="skill">Способность, для которой нужно получить перезарядку.</param>

/// <returns>Оставшееся время перезарядки.</returns>

public float GetSkillCooldown(Skill skill)

{

if (skillCooldowns.ContainsKey(skill))

{

return skillCooldowns[skill];

}

return 0f;

}

/// <summary>

/// Получить текущую перезарядку для способности в выбранном слоте.

/// </summary>

/// <param name="slotNumber">Номер слота.</param>

/// <returns>Оставшееся время перезарядки. Если слота нет или он пуст, возвращает 0.</returns>

public float GetSelectedSkillCooldown(int slotNumber)

{

if (selectedSkills.ContainsKey(slotNumber))

{

return GetSkillCooldown(selectedSkills[slotNumber]);

}

return 0f;

}

/// <summary>

/// Получить способность из выбранного слота.

/// </summary>

/// <param name="slotNumber">Номер слота.</param>

/// <returns>Объект Skill или null, если слот пуст.</returns>

public Skill GetSkillInSlot(int slotNumber)

{

if (selectedSkills.ContainsKey(slotNumber))

{

return selectedSkills[slotNumber];

}

return null;

}

}

1.3 Скрипт поведение простейшего врага (Enemy1.cs)

using System;

using UnityEditor.Rendering.Analytics;

using UnityEditorInternal;

using UnityEngine;

public class Enemy1 : MonoBehaviour

{

public string tagToIgnore = "IgnoreThisTag";

public GameObject bulletPrefab;

public float shootInterval = 2f;

public float moveSpeed = 0.75f;

public float chaseMoveSpeed = 2f;

public float patrolMoveSpeed = 2f;

public float jumpForce = 5f;

public float visionRadius = 10f;

public LayerMask groundLayer; // Для проверки земли и препятствий

private float groundCheckRadius;

public float obstacleCheckDistance = 0.5f; // Как далеко искать препятствия перед собой

public GameObject patrolPoint1 = null;

public GameObject patrolPoint2 = null;

public float attackRange = 2;

private float shootTimer = 0f;

protected GameObject player;

private Rigidbody2D rb;

private bool isGrounded;

private Collider2D collider1;

private Vector2 moveDirection = Vector2.zero;

private bool OnPatrol = true;

private float ChaseTimer;

public float ChaseForSeconds = 3f;

private Transform LastSeenPos;

private bool GoToPointA = false;

private SpriteRenderer lookdir;

public Animator anim;

private void Start()

{

player = GameObject.FindWithTag("Player");

rb = GetComponent<Rigidbody2D>();

collider1 = GetComponent<Collider2D>();

lookdir = GetComponent<SpriteRenderer>();

// Автоматически настраиваем размеры проверок по коллайдеру

groundCheckRadius = (collider1.bounds.size.y / 2) + 0.2f;

obstacleCheckDistance = (collider1.bounds.size.x / 2) + 0.3f;

}

protected virtual void Update()

{

if (player == null) return;

shootTimer += Time.deltaTime;

Vector2 direction = new Vector2(Mathf.Sign(moveDirection.x), 0f);

if (direction.x > 0f)

{

lookdir.flipX = true;

}

else

{

lookdir.flipX = false;

}

anim.SetFloat("Horizontal Velocity", Mathf.Abs(rb.linearVelocityX));

// Проверка на землю

isGrounded = Physics2D.Raycast(transform.position, Vector2.down, groundCheckRadius, groundLayer);

float distanceToPlayer = Vector2.Distance(transform.position, player.transform.position);

if (OnPatrol)

{

if (patrolPoint1 != null && patrolPoint2 != null)

{

Patrol();

}

if (CanSeePlayer())

{

OnPatrol = false;

}

}

else

{

ChasePlayer();

}

}

protected virtual void ShootAtPlayer()

{

if (bulletPrefab == null) return;

Vector2 shootDirection = (player.transform.position - transform.position).normalized;

// Спавним пулю немного перед врагом, чтобы она не исчезала

Vector2 spawnPosition = (Vector2)transform.position + shootDirection \* (obstacleCheckDistance+0.2f);

GameObject bullet = Instantiate(bulletPrefab, spawnPosition, Quaternion.identity);

Bullet bulletScript = bullet.GetComponent<Bullet>();

if (bulletScript != null)

{

bulletScript.lifeTime = attackRange/bulletScript.speed;

bulletScript.direction = shootDirection;

bulletScript.shooter = gameObject;

}

}

bool CanSeePlayer()

{

Vector2 directionToPlayer = (player.transform.position - transform.position).normalized;

float distanceToPlayer = Vector2.Distance(transform.position, player.transform.position);

RaycastHit2D hit = Physics2D.Raycast(transform.position, directionToPlayer, distanceToPlayer, groundLayer);

// Если луч не наткнулся на стены — игрок виден

return hit.collider == null && distanceToPlayer<=visionRadius;

}

bool IsObstacleAhead()

{

Vector2 direction = new Vector2(Mathf.Sign(moveDirection.x), 0f); // Право или лево, куда движется враг

RaycastHit2D hit = Physics2D.Raycast(transform.position - new Vector3(0, (collider1.bounds.size.y / 4)), direction, obstacleCheckDistance, groundLayer);

return hit.collider != null;

}

void Jump()

{

rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);

}

private void OnDrawGizmosSelected()

{

// Радиус зрения

Gizmos.color = Color.yellow;

Gizmos.DrawWireSphere(transform.position, visionRadius);

// Линия к игроку

if (player != null)

{

Gizmos.color = Color.red;

Gizmos.DrawLine(transform.position, player.transform.position);

}

}

void ChasePlayer()

{

moveSpeed = chaseMoveSpeed;

float distanceToPlayer = Vector2.Distance(transform.position, player.transform.position);

if (CanSeePlayer())

{

ChaseTimer = 0;

LastSeenPos = player.transform;

distanceToPlayer = Vector2.Distance(transform.position, player.transform.position);

// Двигаемся к игроку

moveDirection = (player.transform.position - transform.position).normalized;

rb.linearVelocity = new Vector2(moveDirection.x \* moveSpeed, rb.linearVelocity.y);

// Прыгаем через препятствия

if (isGrounded && IsObstacleAhead())

{

Jump();

}

// Стреляем по таймеру

if (shootTimer >= shootInterval && distanceToPlayer <= attackRange)

{

shootTimer = 0f;

ShootAtPlayer();

}

}

else

{

distanceToPlayer = Vector2.Distance(transform.position, LastSeenPos.position);

// Двигаемся к игроку

moveDirection = (LastSeenPos.transform.position - transform.position).normalized;

rb.linearVelocity = new Vector2(moveDirection.x \* moveSpeed, rb.linearVelocity.y);

// Прыгаем через препятствия

if (isGrounded && IsObstacleAhead())

{

Jump();

}

// Стреляем по таймеру

if (shootTimer >= shootInterval && distanceToPlayer<=attackRange)

{

shootTimer = 0f;

ShootAtPlayer();

}

ChaseTimer += Time.deltaTime;

if(ChaseTimer >= ChaseForSeconds)

{

OnPatrol = true;

}

}

}

void Patrol()

{

moveSpeed = patrolMoveSpeed;

if (GoToPointA)

{

moveDirection = (patrolPoint2.transform.position - transform.position).normalized;

rb.linearVelocity = new Vector2(Math.Sign(moveDirection.x) \* moveSpeed, rb.linearVelocity.y);

if (isGrounded && IsObstacleAhead())

{

Jump();

}

if (moveDirection.x > 0)

{

if (transform.position.x >= patrolPoint2.transform.position.x-0.5f)

{

GoToPointA = false;

}

}

else

{

if (transform.position.x <= patrolPoint2.transform.position.x + 0.5f)

{

GoToPointA = false;

}

}

}

else

{

moveDirection = (patrolPoint1.transform.position - transform.position).normalized;

rb.linearVelocity = new Vector2(Math.Sign(moveDirection.x) \* moveSpeed, rb.linearVelocity.y);

if (isGrounded && IsObstacleAhead())

{

Jump();

}

if (moveDirection.x > 0)

{

if (transform.position.x >= patrolPoint1.transform.position.x-0.5f)

{

GoToPointA = true;

}

}

else

{

if (transform.position.x <= patrolPoint1.transform.position.x+0.5f)

{

GoToPointA = true;

}

}

}

}

void OnCollisionEnter(Collision collision)

{

if (collision.gameObject.CompareTag(tagToIgnore))

{

Debug.Log("НЕА");

Physics.IgnoreCollision(GetComponent<Collider>(), collision.collider); // Игнорируем столкновение

}

}

}

**Приложение 2.**

Ссылка на презентацию https://docs.google.com/presentation/d/1KADMRz60ZTlfqKcgBVrrbDlA62BNRWmE/edit?usp=sharing&ouid=103266299067038050597&rtpof=true&sd=true

**Приложение 3.**

Ссылка на видеоотчет <https://drive.google.com/file/d/1TMlsJiX5-pAbwtMITpN-FoK4hLgqOj3W/view?usp=drive_link>

**Приложение 4.**

Скриншоты игры

