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 «Красная шапочка»»**

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

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Изучение механик популярных 2D-платформеров и разработка концепта игры | Руденко Е. Е. | 24.05.2025 – 31.05.2025 |
| 2 | Реализация игровых систем | 01.06.2025 – 12.06.2025 |
| 3 | Создание игрового контента | 13.06.2025 – 17.06.2025 |
| 4 | Проведение тестирования и балансировки, а также финальная доработка | 18.06.2025 – 20.06.2025 |
| 5 | Подготовка отчета, видео и презентации | 21.06.2025 – 26.06.2025 |

Руководитель проекта   
старший преподаватель кафедры культурологии и дизайна Каратаев А. 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 «Красная шапочка»»**

Исполнители:

Руденко Е. Е. *(Фамилия И.О.)*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта

старший преподаватель кафедры культурологии и дизайна

Каратаев А. А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

1. **Цель проекта**

Целью данного проекта является разработка 2D-платформера по мотивам народной европейской сказки «Красная шапочка».

1. **Задачи проекта и исполнители**

Для достижения поставленной цели требовалось решить ряд задач:

1. Изучить механики популярных 2D-платформеров (Celeste, Hollow Knight) для анализа успешных решений и вдохновения, а также разработать концепт игры с элементами инвентаря, боёв и предметов на основе сказочного сюжета.
2. Реализовать следующие игровые системы, как управление персонажем (бег, прыжки, инерция), систему здоровья и урона (лечение, получение урона), инвентарь с ограниченными слотами и функционалом использования предметов, ИИ врагов: патрулирование и преследование для волков, контактный урон и оглушение для ежей.
3. Создать игровой контент:

* Отрисовать все спрайты (персонаж, враги, предметы, окружение) в Krita;
* Анимировать движения, атаки и эффекты в Unity.

1. Провести тестирование и балансировку.
2. Выполнить финальную доработку: добавить визуальные эффекты (анимации ударов, частицы) и звуки (шаги, удары, использование предметов)

Проект разрабатывался в одиночку, что позволило полностью контролировать единство стиля от кода до графики. Глубже изучить все этапы разработки — от проектирования до тестирования. Отработать навыки многозадачности.

1. **Актуальность и востребованность проекта (разрабатываемого продукта)**

Проект основан на европейской сказке «Красная шапочка», что придаёт ему культурную глубину и актуальность. В условиях возросшего интереса к переосмыслению фольклора в современных медиа игра предлагает взрослой и подростковой аудитории интерактивный способ знакомства с классическим сюжетом через призму мрачного визуального стиля и классических игровых механик. Подобный подход соответствует тренду на интеграцию народных мотивов в современные игры, что видно на примере проектов «The Path» и «Hollow Knight».

2D-платформеры остаются одним из самых популярных жанров на инди-сцене благодаря сочетанию доступности и глубокого геймплея. Проект ориентирован на широкую аудиторию геймеров, включая пользователей со слабыми ПК.

Проект демонстрирует полный цикл разработки игры — от проектирования и программирования до создания визуального контента и тестирования. Это делает его ценным примером для: начинающих гейм-дизайнеров, желающих освоить разработку на Unity и C#; художников, интересующихся созданием спрайтов и анимаций с нуля; студентов, изучающих интеграцию программирования, дизайна и менеджмента в рамках индивидуального проекта.

Перспективы развития: Игра может послужить основой для коммерческого продукта или серии проектов по мотивам других сказок.

Итого сочетание мрачной атмосферы, культурного наследия и современных игровых механик делает проект актуальным для широкой аудитории геймеров, образовательной среды и инди-разработчиков. Востребованность подтверждается успехом аналогичных проектов, где сказочные мотивы и сложный геймплей привлекают внимание как игроков, так и профессионального сообщества.

1. **Общие сведения о проделанной работе**

**Этапы выполнения проекта:**

1. Исследование и анализ игровых механик.

На первом этапе проекта было проведено глубокое изучение популярных 2D-платформеров, таких как Celeste и Hollow Knight, с акцентом на ключевые игровые механики. Основное внимание уделялось системам управления персонажем, боевой механике и работе инвентаря. Также был проанализирован искусственный интеллект врагов, включая алгоритмы патрулирования, преследования и атаки. Дополнительно рассмотрены методы визуализации игрового процесса, включая параллакс-скроллинг, анимации персонажей и эффекты частиц, что позволило определить лучшие практики для дальнейшей реализации.

1. Проектирование архитектуры игры.

На этапе проектирования были определены ключевые системы игры:

* Движение игрока: реализована физика перемещения с учетом инерции, прыжков и взаимодействия с окружением.
* Боевая механика: разработаны системы урона, отталкивания и оглушения врагов.
* Инвентарь и предметы: продуманы функциональность и ограничения для таких предметов, как яблоки, подорожники и камни.
* Искусственный интеллект врагов: созданы модели поведения для волков (патрулирование, преследование) и ежей (контактный урон, оглушение).

Кроме того, была разработана схема взаимодействия между системами, что обеспечило согласованную работу всех компонентов игры.

1. Разработка прототипа.

На этом этапе были реализованы базовые механики:

* Управление персонажем: движение (бег, прыжки), использование предметов и взаимодействие с окружением.
* Система здоровья и урона: игрок может получать урон от врагов и восстанавливать здоровье с помощью подорожников.
* Инвентарь: ограниченное количество слотов для хранения предметов, таких как камни, палки и яблоки.

Также были созданы враги с уникальным поведением:

* Волки: патрулируют территорию, преследуют игрока и атакуют при приближении.
* Ежи: наносят урон при контакте, но могут быть оглушены с помощью предметов.

1. Тестирование и балансировка.

После разработки прототипа проведено тщательное тестирование всех систем. Были проверены:

* Корректность работы механик управления, боя и инвентаря.
* Баланс параметров: скорость передвижения игрока, урон врагов, сила отталкивания и длительность эффектов предметов.
* Исправлены ошибки, связанные с коллизиями и физикой, чтобы обеспечить плавный и предсказуемый геймплей.

1. Финальная доработка и оптимизация.

На завершающем этапе были добавлены визуальные и звуковые эффекты для улучшения игровой атмосферы:

* Анимации ударов, шагов и использования предметов.
* Система звуков, включая шаги персонажа, удары и звуковые эффекты предметов.
* Проведена оптимизация рендеринга для повышения производительности игры.

**Техническая реализация:**

Для разработки игры были использованы следующие технологии:

1. Язык программирования: C# – основной язык для Unity, обеспечивающий гибкость и высокую производительность при работе с игровой логикой.
2. Игровой движок: Unity – выбран благодаря мощным возможностям работы с 2D-графикой, физикой, анимациями, пользовательским интерфейсом и звуком.
3. Среда разработки: Visual Studio – предоставляет удобные инструменты для отладки, автодополнения кода и интеграции с Unity.
4. Krita – использовалась для создания и редактирования спрайтов, обеспечивая высокое качество визуальных элементов игры.

**Ключевые игровые системы:**

Персонаж игрока реализован с помощью нескольких классов. Класс PlayerInput.cs отвечает за управление движением персонажа, обработку нажатий клавиш для взаимодействия и использования предметов, а также содержит методы, предотвращающие бесконечные прыжки. Горизонтальное перемещение происходит с инерцией и скоростью 5 м/с, прыжки выполняются только при проверке нахождения персонажа на земле. Класс PlayerHealth.cs управляет здоровьем персонажа, устанавливая максимальное значение в 100 HP, обрабатывая получение урона от врагов (волков и ежей) и позволяя лечиться с помощью подорожника, который восстанавливает 25 HP. Для работы с инвентарём создан класс Inventory.cs, который обеспечивает хранение, использование, добавление и удаление предметов. В инвентаре можно носить камни, палки, яблоки и подорожники, при этом количество слотов ограничено (например, максимум 20 палок и 10 камней).

Враги в игре реализованы с помощью двух основных классов. Класс WolfPatrol.cs отвечает за поведение волков, которые патрулируют заданную зону, передвигаясь по маршруту с указанными точками и делая паузы перед повторением цикла. При обнаружении игрока волки переходят в режим преследования и атакуют, нанося 15 урона за удар. У каждого волка есть полоска здоровья на 30 HP, а скорость передвижения составляет 3,5 м/с. Класс HeadhogSpikes.cs реализует поведение ежей, которые неподвижны, но при контакте с игроком наносят 20 урона и отталкивают персонажа с определённой силой. Ежи могут быть оглушены ударом камня или палки.

В игре реализована система предметов с помощью класса InteractItem.cs, который управляет использованием различных типов предметов, наследующихся от базового класса. Яблоко даёт временные баффы, увеличивая скорость персонажа на 30%, силу прыжка на 20% и урон на 100% в течение 20 секунд. Подорожник служит для восстановления здоровья, восстанавливая 25 HP. Камни и палки можно использовать как метательное оружие: камень наносит 25 урона, а палка — 10 урона врагам. Визуально временные баффы отображаются на экране, а полоска здоровья игрока динамически уменьшается при снижении HP. Инвентарь показывает текущее количество предметов, что обеспечивает удобство управления ресурсами в игре.

В проекте используются несколько ключевых библиотек, обеспечивающих функциональность игры. Основная библиотека UnityEngine содержит базовые классы и функции для работы с игровыми объектами, компонентами, физикой, анимациями и другими системами движка. Она предоставляет доступ к таким компонентам, как Transform, Rigidbody2D и Collider2D, позволяет работать с физикой через Physics2D, анимациями через Animator, звуком через AudioSource, а также отвечает за рендеринг спрайтов, управление пользовательским интерфейсом и сценами. Стандартная библиотека System языка C# предоставляет базовые классы и структуры данных, такие как String, Array и Action. В проекте она используется для обработки исключений, работы с событиями и делегатами, а также для управления коллекциями, например списками и словарями. Подмодуль UnityEngine.UI отвечает за создание и управление элементами пользовательского интерфейса, включая кнопки, текст и изображения. С его помощью реализовано отображение здоровья персонажа, инвентаря и меню паузы, а также визуальная обратная связь, например изменение цвета полоски здоровья. Библиотека System.Collections используется для работы с коллекциями, такими как списки и массивы, а также для реализации корутин с помощью IEnumerator и yield return. Корутины применяются для анимаций, таймеров и пошаговой логики искусственного интеллекта. Хотя в проекте иногда используются базовые коллекции, чаще применяются их обобщённые аналоги из System.Collections.Generic, такие как List<T> и Dictionary<K,V>.

1. **Результаты проекта**

Проект завершился созданием полноценного 2D-платформера "Красная шапочка" с продуманным геймплеем. Были успешно реализованы:

1. Система управления персонажем с физикой движения;
2. Инвентарь с ограниченными слотами;
3. Боевая механика;
4. ИИ врагов (волки с патрулированием и ежи с контактным уроном).

Все графические элементы созданы в Krita, анимации и эффекты разработаны в Unity, что обеспечило единый визуальный стиль.

Тщательное тестирование и балансировка позволили добиться плавного геймплея и хорошей производительности.

Проект стал ценной практикой полного цикла разработки - от концепции до реализации. Освоены ключевые навыки работы с Unity, C# и созданием 2D-графики.

Игра обладает потенциалом для расширения новыми уровнями, врагами и предметами. Результат демонстрирует как технические навыки, так и творческий подход к интерпретации классического сюжета, представляя ценность как учебный пример и основа для коммерческого продукта.

**Приложение 1.**

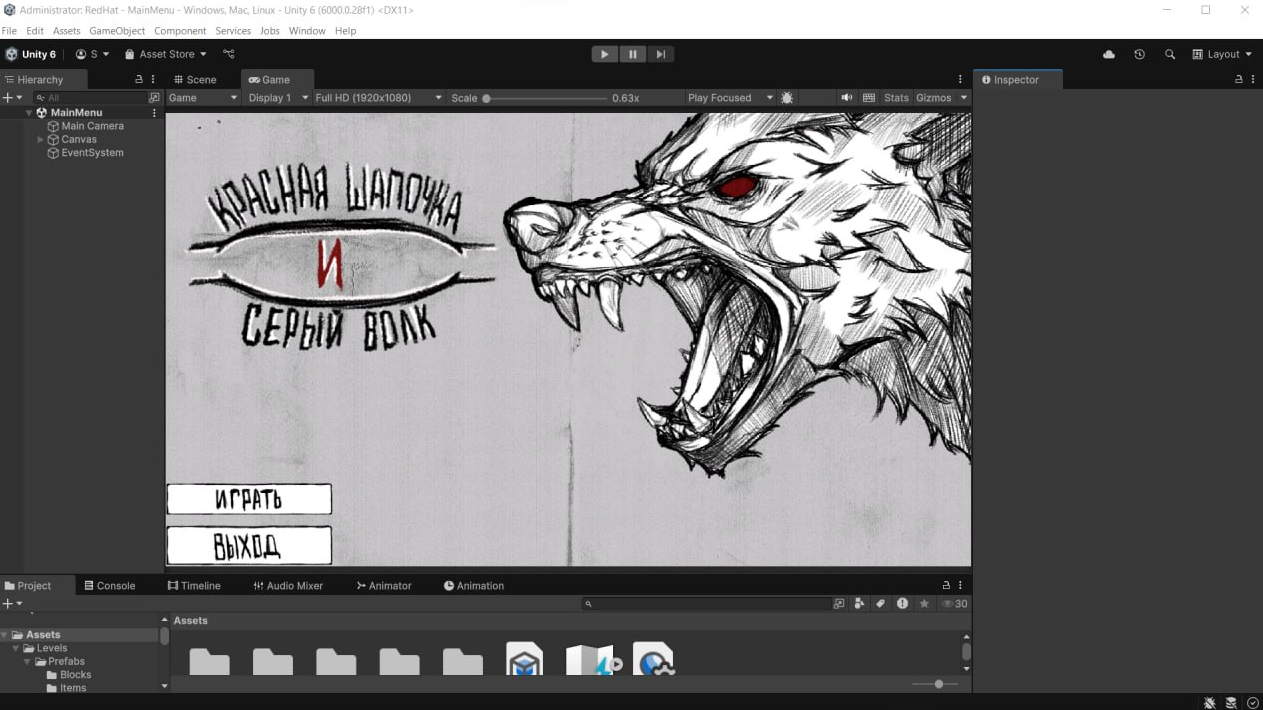
****

Рисунок 1 – Главное меню

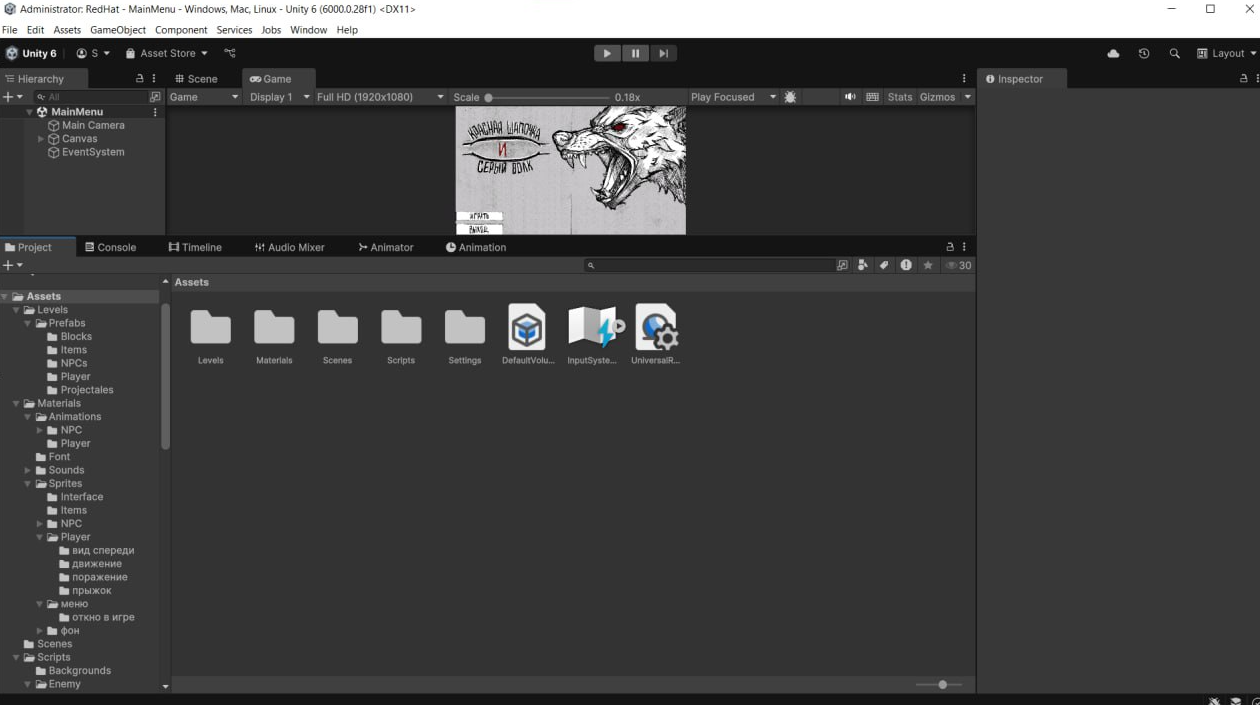


Рисунок 2 – Создание структуры папок

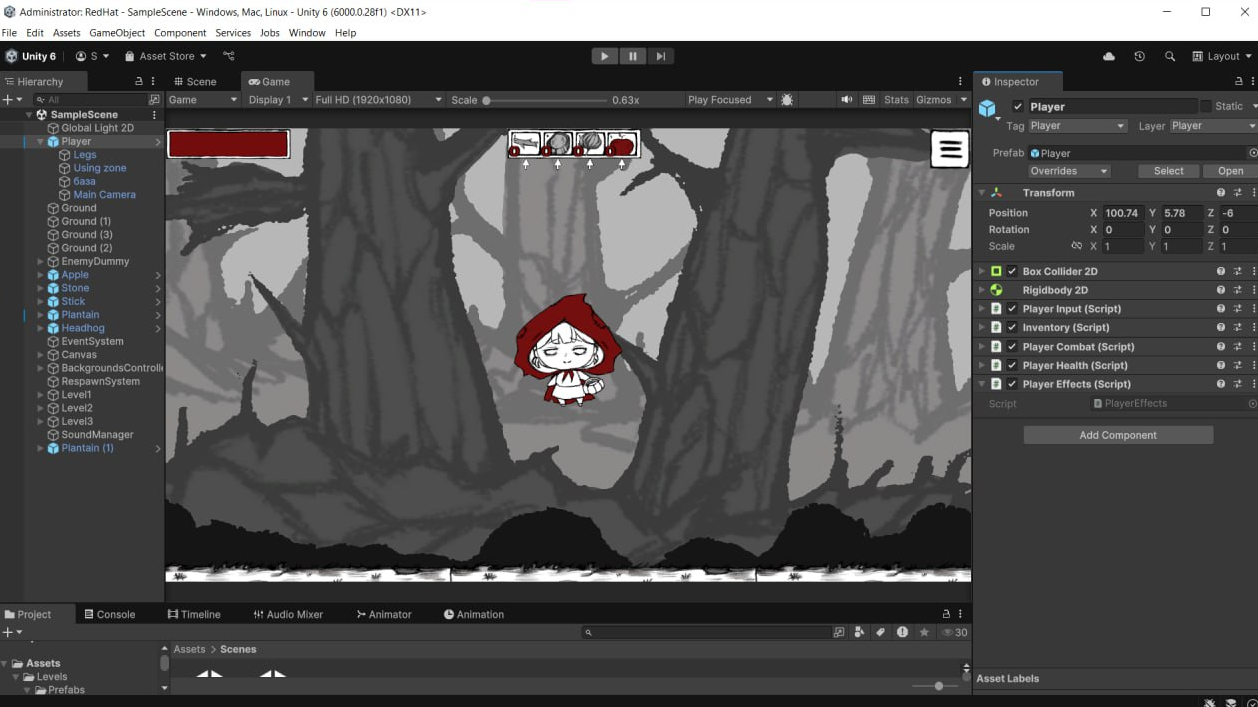


Рисунок 3 – Персонаж и его компоненты

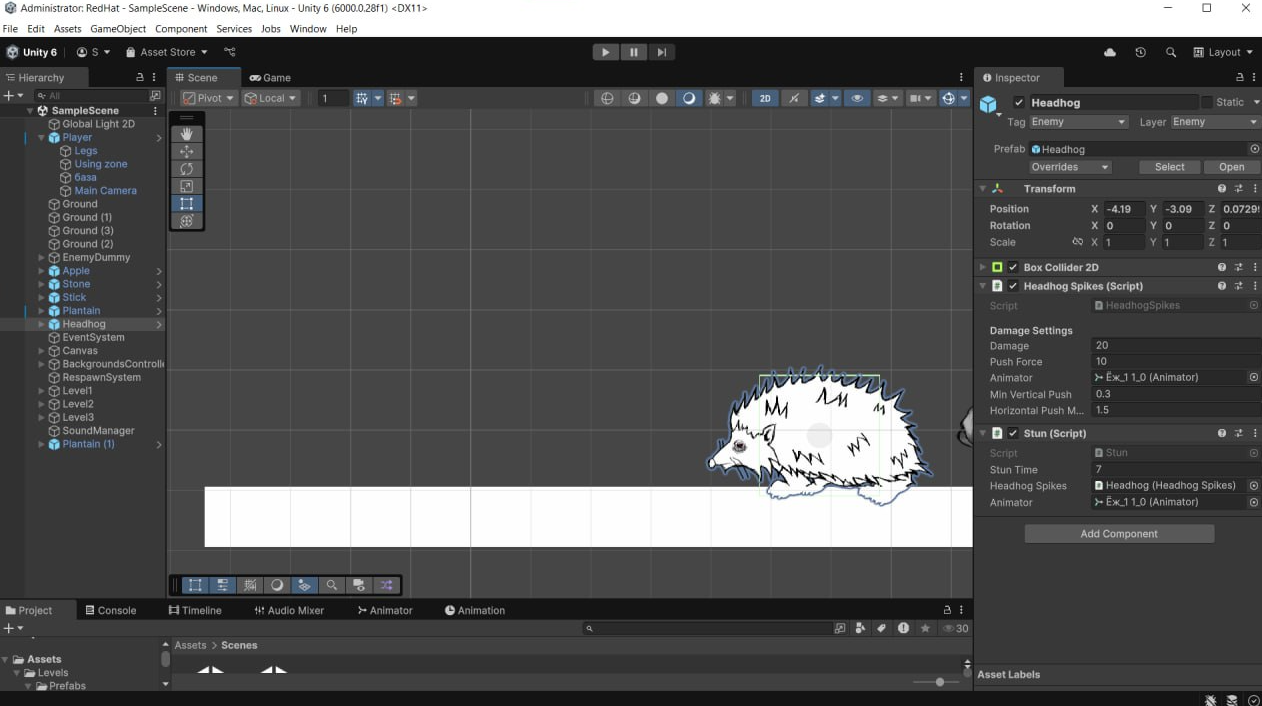


Рисунок 4 – Ёж и его компоненты



Рисунок 5 – Пример расходуемого объекта (подорожник)

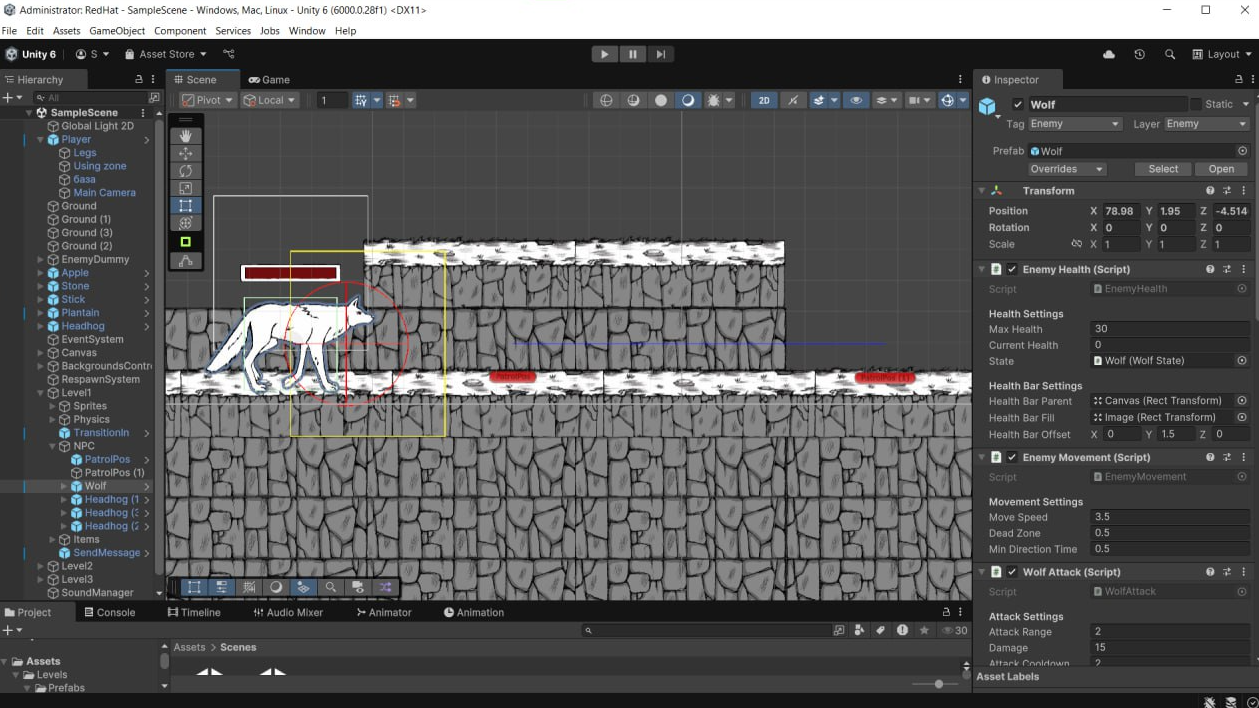


Рисунок 6 – Волк, его компоненты и система патрулирования



Рисунок 7 – Интерфейс здоровья и инвентаря

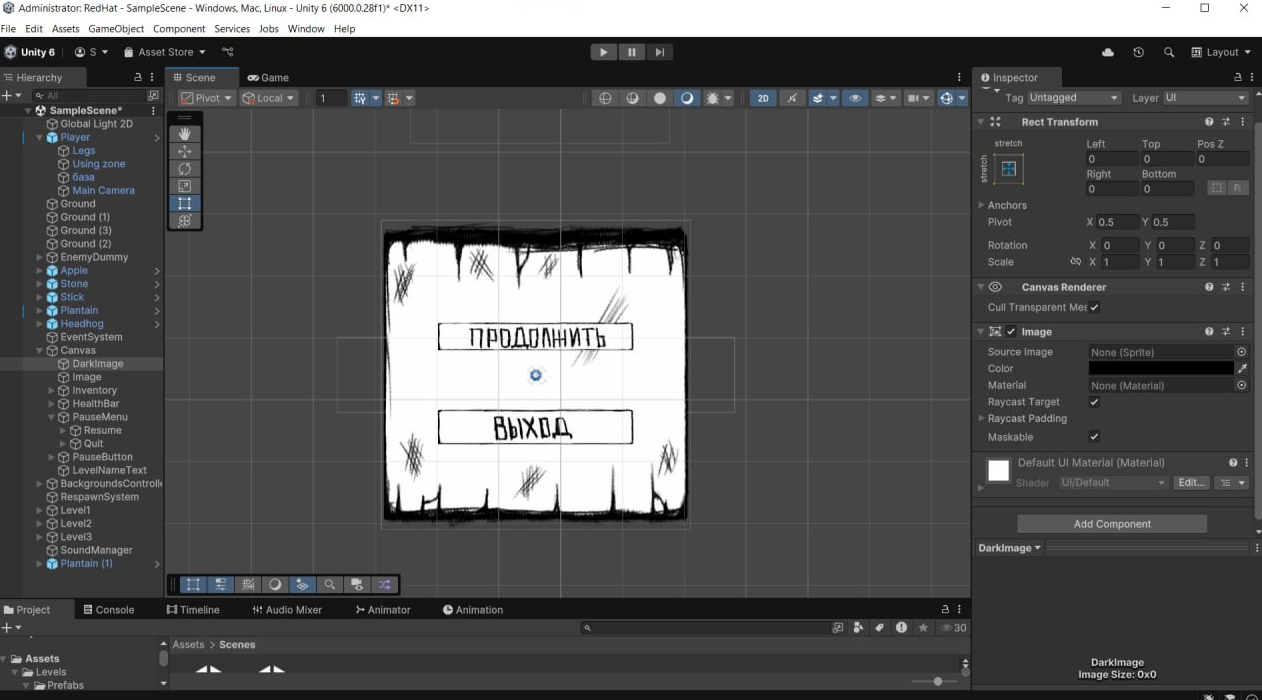


Рисунок 8 – Меню паузы

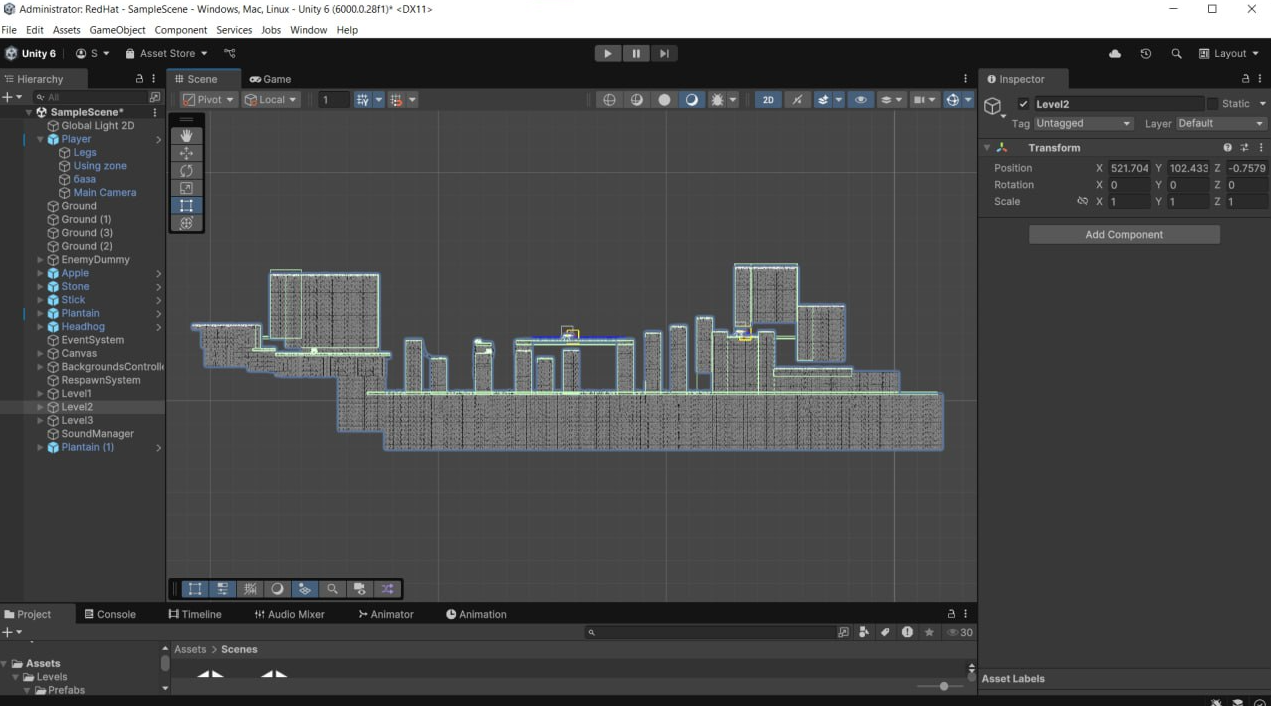


Рисунок 9 – Дизайн уровня



Рисунок 10 – менеджер звуков

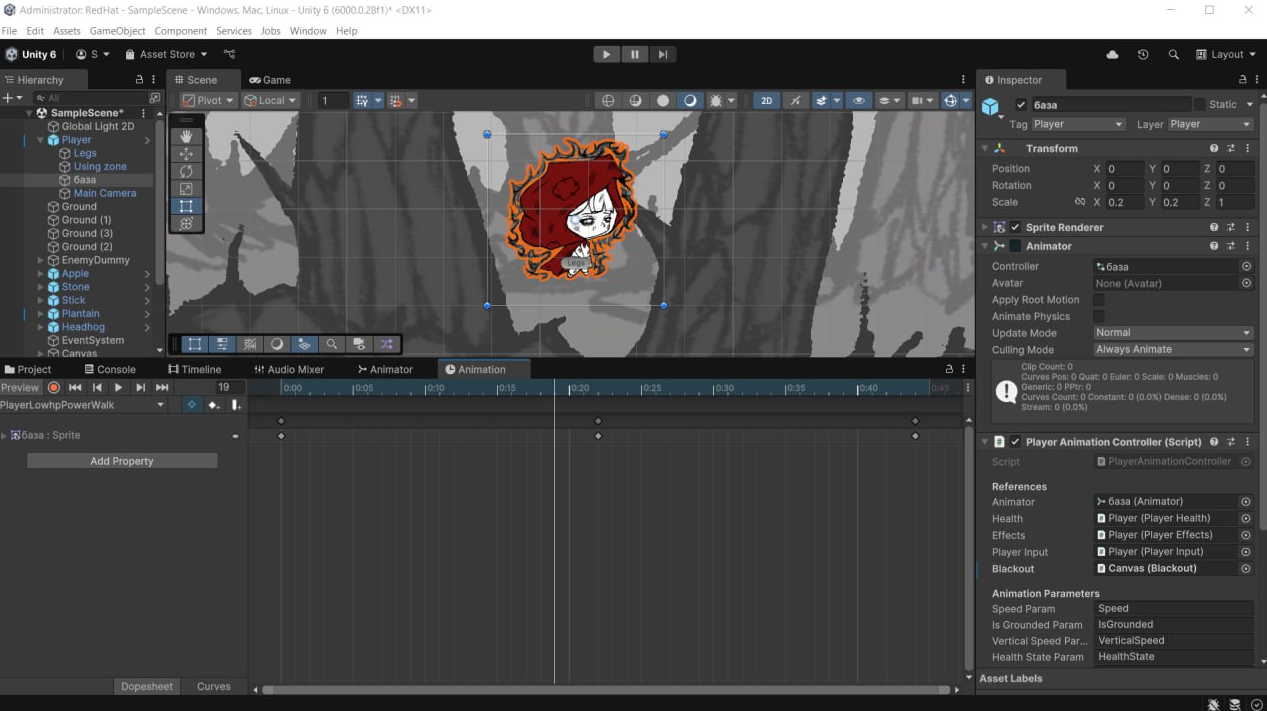


Рисунок 11 – Персонаж, настройка анимации

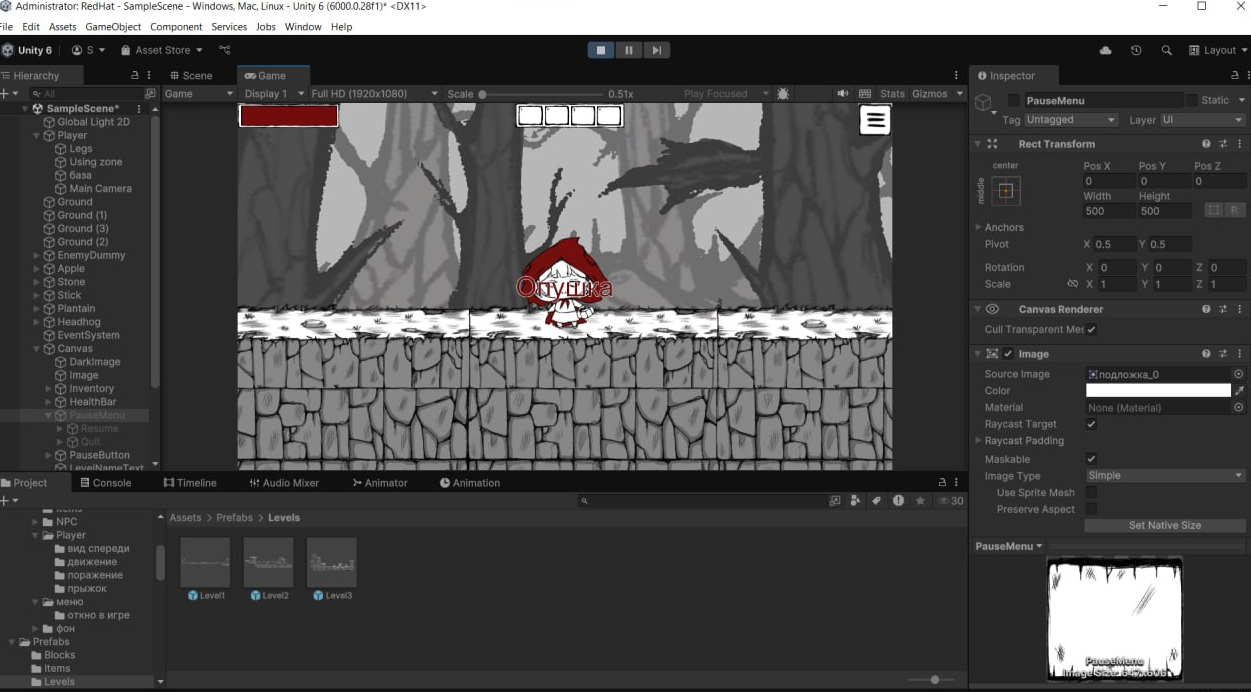


Рисунок 12 – Отображение текущего уровня

**Приложение 2.**

Программный код 1.

using UnityEngine;

public class BackgroundResizer : MonoBehaviour

{

private Camera mainCamera;

private Renderer backgroundRenderer;

void Start()

{

if (backgroundRenderer == null)

backgroundRenderer = GetComponent<Renderer>();

// Масштабируем текстуру под размер Plane

float textureAspect = (float)backgroundRenderer.material.mainTexture.width /

backgroundRenderer.material.mainTexture.height;

float screenAspect = (float)Screen.width / Screen.height;

// Если текстура уже квадратная, можно пропустить

if (Mathf.Abs(textureAspect - screenAspect) > 0.01f)

{

backgroundRenderer.material.mainTextureScale =

new Vector2(screenAspect / textureAspect, 1f);

}

}

}

Программный код 2.

using UnityEngine;

public class CameraFollower : MonoBehaviour

{

public Transform cameraTransform;

public bool followX = true;

public bool followY = true;

public bool followRotation = false;

void LateUpdate()

{

Vector3 newPosition = transform.position;

if (followX) newPosition.x = cameraTransform.position.x;

if (followY) newPosition.y = cameraTransform.position.y;

transform.position = newPosition;

if (followRotation)

{

transform.rotation = cameraTransform.rotation;

}

}

}

Программный код 3.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ParallaxController : MonoBehaviour

{

[SerializeField] Transform cam;

[SerializeField] Vector3 camStartPos;

[SerializeField] float distance;

[SerializeField] GameObject[] backgrounds;

[SerializeField] Material[] mat;

[SerializeField] float[] backSpeed;

[SerializeField] float farthestBack;

[Range(0.01f, 0.05f)]

public float parallaxSpeed;

void Start()

{

cam = Camera.main.transform;

camStartPos = cam.position;

int backCount = transform.childCount;

mat = new Material[backCount];

backSpeed = new float[backCount];

backgrounds = new GameObject[backCount];

for (int i = 0; i < backCount; i++)

{

backgrounds[i] = transform.GetChild(i).gameObject;

mat[i] = backgrounds[i].GetComponent<Renderer>().material;

}

BackSpeedCalculate(backCount);

}

void BackSpeedCalculate(int backCount)

{

for (int i = 0; i < backCount; i++)

{

if ((backgrounds[i].transform.position.z - cam.position.z) > farthestBack)

{

farthestBack = backgrounds[i].transform.position.z - cam.position.z;

}

}

for (int i = 0; i < backCount; i++)

{

backSpeed[i] = 1 - (backgrounds[i].transform.position.z - cam.position.z) / farthestBack;

}

}

private void LateUpdate()

{

distance = cam.position.x - camStartPos.x;

transform.position = new Vector3(cam.position.x, transform.position.y, 0);

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

{

float speed = backSpeed[i] \* parallaxSpeed;

mat[i].SetTextureOffset("\_MainTex", new Vector2(distance, 0) \* speed);

}

}

}

Программный код 4.

using UnityEngine;

using System.Collections;

public class HeadhogSpikes : MonoBehaviour

{

[Header("Damage Settings")]

[SerializeField] private int damage = 20;

[SerializeField] private float pushForce = 10f;

[SerializeField] private Animator animator;

[SerializeField] private float minVerticalPush = 0.3f;

[SerializeField] private float horizontalPushMultiplier = 1.5f;

SoundManager soundManager;

private void Start()

{

soundManager = FindAnyObjectByType<SoundManager>();

}

private void OnCollisionEnter2D(Collision2D collision)

{

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

{

PlayerHealth playerHealth = collision.gameObject.GetComponent<PlayerHealth>();

PlayerInput playerMovement = collision.gameObject.GetComponent<PlayerInput>();

if (playerHealth != null && playerMovement != null)

{

// Реверс направления: от игрока к ежу

Vector2 pushDirection = (transform.position - collision.transform.position).normalized;

// Усиливаем горизонтальную составляющую

pushDirection.x = Mathf.Sign(-pushDirection.x) \* Mathf.Max(Mathf.Abs(pushDirection.x), 0.5f);

// Гарантируем толчок вверх

pushDirection.y = Mathf.Max(pushDirection.y, 0.3f);

pushDirection.Normalize();

// Наносим урон с отталкиванием

playerHealth.TakeDamageWithPush(damage, pushDirection, pushForce);

soundManager.PlayHedgehogAttack();

SetAtackAnimation(true);

StartCoroutine(Attacking(0.1f));

Debug.DrawRay(collision.transform.position, pushDirection \* 3, Color.red, 2f); // Визуализация

}

}

}

IEnumerator Attacking(float time)

{

yield return new WaitForSeconds(time);

SetAtackAnimation(false);

}

public void SetAtackAnimation(bool isAttacking)

{

if (animator != null)

animator.SetBool("Atack", isAttacking);

}

}

Программный код 5.

using System.Collections;

using UnityEngine;

public class Stun : MonoBehaviour

{

[SerializeField] private float stunTime;

[SerializeField] private HeadhogSpikes headhogSpikes;

[SerializeField] private Animator animator;

// Start is called once before the first execution of Update after the MonoBehaviour is created

void Start()

{

headhogSpikes = GetComponent<HeadhogSpikes>();

}

// Update is called once per frame

void Update()

{

}

public void OnStun()

{

headhogSpikes.enabled = false;

transform.GetComponent<BoxCollider2D>().enabled = false;

animator.SetBool("IsStun", true);

StartCoroutine(Stuning(stunTime));

}

IEnumerator Stuning(float time)

{

yield return new WaitForSeconds(time);

OffStun();

}

public void OffStun()

{

headhogSpikes.enabled = true;

transform.GetComponent<BoxCollider2D>().enabled = true;

animator.SetBool("IsStun", false);

}

}

Программный код 6.

using UnityEngine;

public class EnemyMovement : MonoBehaviour

{

[Header("Movement Settings")]

[SerializeField] private float moveSpeed = 3.5f;

[SerializeField] private float deadZone = 0.5f;

[SerializeField] private float minDirectionTime = 0.5f;

private WolfPatrol patrol;

private Rigidbody2D rb;

private float currentDirection;

private float lastDirectionChangeTime;

private float lastDirection;

private void Awake()

{

patrol = GetComponent<WolfPatrol>();

rb = GetComponent<Rigidbody2D>();

}

public void MoveTo(Vector2 targetPosition)

{

float newDirection = Mathf.Sign(targetPosition.x - transform.position.x);

if (Mathf.Abs(targetPosition.x - transform.position.x) < deadZone)

{

newDirection = currentDirection != 0 ? currentDirection : lastDirection;

}

else

{

if (Time.time - lastDirectionChangeTime > minDirectionTime ||

Mathf.Sign(newDirection) != Mathf.Sign(currentDirection))

{

currentDirection = newDirection;

lastDirection = newDirection;

lastDirectionChangeTime = Time.time;

}

}

Vector2 velocity = new Vector2(currentDirection \* moveSpeed, rb.linearVelocity.y);

rb.linearVelocity = velocity;

FlipSprite(currentDirection);

}

private void FlipSprite(float direction)

{

if (direction == 0) return;

Vector3 scale = transform.localScale;

scale.x = Mathf.Abs(scale.x) \* Mathf.Sign(direction);

transform.localScale = scale;

}

}

Программный код 7.

using System.Collections;

using UnityEngine;

public class WolfAnimationController : MonoBehaviour

{

[Header("Animation References")]

[SerializeField] private Animator animator;

[SerializeField] private WolfState wolfState;

[SerializeField] private EnemyMovement movement;

[SerializeField] private WolfPatrol patrol;

[SerializeField] private WolfAttack attack;

[SerializeField] private Collider2D wolfCollider; // Добавим коллайдер для отключения

// Параметры аниматора

private const string IDLE\_PARAM = "IsIdle";

private const string WALK\_PARAM = "IsWalking";

private const string RUN\_PARAM = "IsRunning";

private const string ATTACK\_PARAM = "Attack";

private const string AFK\_PARAM = "isAFK";

private const string DEATH\_PARAM = "Death";

private const string DEATH\_STATE\_NAME = "WolfDied"; // Имя состояния смерти в аниматоре

private bool isDead = false;

private void Update()

{

if (animator == null || wolfState == null || isDead) return;

UpdateMovementAnimations();

}

private void UpdateMovementAnimations()

{

if (isDead) return;

bool isMoving = false;

bool isRunning = false;

bool isAFK = false;

switch (wolfState.currentState)

{

case WolfState.State.Patrol:

isMoving = patrol != null && patrol.enabled && !patrol.isWaiting;

break;

case WolfState.State.AFK:

isAFK = true;

break;

case WolfState.State.Chase:

isMoving = true;

isRunning = true;

break;

case WolfState.State.Attack:

// В атаке - без движения

break;

}

if (animator != null)

{

animator.SetBool(IDLE\_PARAM, !isMoving && !isRunning && !isAFK);

animator.SetBool(WALK\_PARAM, isMoving && !isRunning);

animator.SetBool(RUN\_PARAM, isRunning);

animator.SetBool(AFK\_PARAM, isAFK);

}

}

public void TriggerAttackAnimation()

{

if (animator != null && !isDead)

{

animator.SetTrigger(ATTACK\_PARAM);

}

}

public void TriggerDeathAnimation()

{

if (isDead) return;

isDead = true;

if (animator != null)

{

// Отключаем все другие состояния

animator.SetBool(IDLE\_PARAM, false);

animator.SetBool(WALK\_PARAM, false);

animator.SetBool(RUN\_PARAM, false);

// Отменяем возможную атаку

if (attack != null) attack.StopAttack();

// Запускаем анимацию смерти

animator.SetTrigger(DEATH\_PARAM);

// Отключаем все компоненты

if (wolfState != null) wolfState.enabled = false;

if (movement != null) movement.enabled = false;

if (patrol != null) patrol.enabled = false;

if (wolfCollider != null) wolfCollider.enabled = false;

// Начинаем отсчет времени до уничтожения

StartCoroutine(DestroyAfterAnimation());

}

else

{

Destroy(gameObject); // Если нет аниматора - сразу уничтожаем

}

}

private IEnumerator DestroyAfterAnimation()

{

// Ждем пока анимация смерти начнется

yield return null; // Пропускаем один кадр чтобы аниматор успел переключиться

// Получаем длину текущей анимации (смерти)

float animationLength = animator.GetCurrentAnimatorStateInfo(0).length;

// Ждем завершения анимации

yield return new WaitForSeconds(animationLength);

// Уничтожаем объект

Destroy(gameObject);

}

// Добавим метод для проверки смерти из других скриптов

public bool IsDead()

{

return isDead;

}

}

Программный код 8.

using UnityEngine;

public class WolfAttack : MonoBehaviour

{

[Header("Attack Settings")]

[SerializeField] private float attackRange = 1f;

[SerializeField] private int damage = 15;

[SerializeField] private float attackCooldown = 2f;

[SerializeField] private Transform attackPoint;

[SerializeField] private LayerMask playerLayer;

[SerializeField] private float attackDelay = 0.3f;

[SerializeField] private float pushForce = 5f; // Сила отталкивания

SoundManager soundManager;

[Header("Debug")]

[SerializeField] private bool showGizmos = true;

private float lastAttackTime;

private bool isAttacking;

private void Start()

{

soundManager = FindAnyObjectByType<SoundManager>();

}

public bool CanAttack()

{

return Time.time >= lastAttackTime + attackCooldown;

}

public bool IsPlayerInRange()

{

return Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer);

}

public void StartAttack()

{

if (!CanAttack() || isAttacking) return;

isAttacking = true;

lastAttackTime = Time.time;

Invoke(nameof(PerformAttack), attackDelay);

}

private void PerformAttack()

{

if (IsPlayerInRange())

{

Collider2D player = Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer);

if (player != null)

{

soundManager.PlayWolfAttack();

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

player.GetComponent<PlayerHealth>()?.TakeDamageWithPush(damage, pushDirection, pushForce);

Debug.Log($"Wolf attacked player! Damage: {damage}, Push: {pushForce}");

}

}

isAttacking = false;

}

public void StopAttack()

{

CancelInvoke(nameof(PerformAttack));

isAttacking = false;

}

private void OnDrawGizmosSelected()

{

if (!showGizmos || attackPoint == null) return;

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(attackPoint.position, attackRange);

// Визуализация направления атаки

if (Application.isPlaying && IsPlayerInRange())

{

Gizmos.color = Color.yellow;

Vector2 playerPos = Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer).transform.position;

Gizmos.DrawLine(attackPoint.position, playerPos);

}

}

}

Программный код 9.

using UnityEngine;

public class WolfPatrol : MonoBehaviour

{

[Header("Patrol Settings")]

[SerializeField] private Transform leftBound;

[SerializeField] private Transform rightBound;

[SerializeField] private float moveSpeed = 2f;

[SerializeField] private float waitTime = 1f;

private float waitTimer;

private bool movingRight = true;

public bool isWaiting = false;

private EnemyMovement enemyMovement;

private void Awake()

{

enemyMovement = GetComponent<EnemyMovement>();

}

private void Update()

{

if (isWaiting)

{

waitTimer -= Time.deltaTime;

if (waitTimer <= 0) isWaiting = false;

return;

}

PatrolMovement();

}

private void PatrolMovement()

{

float targetX = movingRight ? rightBound.position.x : leftBound.position.x;

float direction = Mathf.Sign(targetX - transform.position.x);

// Используем MoveTo из EnemyMovement для согласованного движения

enemyMovement.MoveTo(new Vector2(targetX, transform.position.y));

// Проверка достижения границы

if (Mathf.Abs(transform.position.x - targetX) < 0.1f)

{

movingRight = !movingRight;

isWaiting = true;

waitTimer = waitTime;

}

}

private void OnDrawGizmosSelected()

{

if (leftBound && rightBound)

{

Gizmos.color = Color.blue;

Gizmos.DrawLine(

new Vector2(leftBound.position.x, transform.position.y),

new Vector2(rightBound.position.x, transform.position.y));

}

}

}

Программный код 10.

using UnityEngine;

public class WolfState : MonoBehaviour

{

public enum State { Patrol, Chase, Attack, AFK }

public State currentState = State.Patrol;

[Header("References")]

[SerializeField] private WolfVision vision;

[SerializeField] private WolfPatrol patrol;

[SerializeField] private WolfAttack attack;

[SerializeField] private EnemyMovement movement;

[SerializeField] private WolfAnimationController animationController;

[Header("Settings")]

[SerializeField] private float chaseRange = 7f;

[SerializeField] private float attackRange = 2f;

[SerializeField] private float visionCheckInterval = 0.3f;

private Transform player;

private float lastVisionCheckTime;

private void Start()

{

player = GameObject.FindWithTag("Player")?.transform;

if (player == null)

{

Debug.LogError("Player not found! Make sure player has 'Player' tag.");

enabled = false;

}

// Автоматическое получение ссылок на компоненты

if (vision == null) vision = GetComponent<WolfVision>();

if (patrol == null) patrol = GetComponent<WolfPatrol>();

if (attack == null) attack = GetComponent<WolfAttack>();

if (movement == null) movement = GetComponent<EnemyMovement>();

if (animationController == null) animationController = GetComponent<WolfAnimationController>();

}

private void Update()

{

if (player == null) return;

if (Time.time - lastVisionCheckTime > visionCheckInterval)

{

lastVisionCheckTime = Time.time;

UpdateState();

}

ExecuteState();

}

private void UpdateState()

{

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

switch (currentState)

{

case State.Attack:

if (distanceToPlayer > attackRange \* 1.1f)

SwitchState(State.Chase);

break;

case State.Patrol:

if (patrol != null && patrol.isWaiting)

{

SwitchState(State.AFK);

}

else if (distanceToPlayer <= chaseRange && vision.CanSeeTarget(player))

{

SwitchState(State.Chase);

}

break;

case State.AFK:

if (patrol == null || !patrol.isWaiting)

{

SwitchState(State.Patrol);

}

else if (distanceToPlayer <= chaseRange && vision.CanSeeTarget(player))

{

SwitchState(State.Chase);

}

break;

case State.Chase:

if (distanceToPlayer <= attackRange)

{

SwitchState(State.Attack);

}

else if (distanceToPlayer > chaseRange \* 1.2f)

{

SwitchState(State.Patrol);

}

break;

}

}

private void ExecuteState()

{

switch (currentState)

{

case State.Patrol:

patrol.enabled = true;

movement.enabled = false;

break;

case State.Chase:

patrol.enabled = false;

movement.enabled = true;

movement.MoveTo(player.position);

break;

case State.Attack:

patrol.enabled = false;

movement.enabled = false;

attack.StartAttack();

if (animationController != null)

{

animationController.TriggerAttackAnimation();

}

break;

case State.AFK:

patrol.enabled = true;

movement.enabled = false;

break;

}

}

private void SwitchState(State newState)

{

if (currentState == newState) return;

switch (currentState)

{

case State.Attack:

attack.StopAttack();

break;

}

currentState = newState;

Debug.Log($"Wolf state changed to: {newState}");

}

public void Die()

{

if (!animationController.IsDead()) // Проверяем, не умер ли уже волк

{

animationController.TriggerDeathAnimation();

}

}

}

Программный код 11.

using UnityEngine;

public class WolfVision : MonoBehaviour

{

[Header("Vision Settings")]

[SerializeField] private float visionRange = 5f;

[SerializeField] private float visionWidth = 3f;

[SerializeField] private LayerMask obstacleLayers;

[SerializeField] private LayerMask targetLayer;

public bool CanSeeTarget(Transform target)

{

Vector2 directionToTarget = target.position - transform.position;

float distanceToTarget = directionToTarget.magnitude;

// Проверка расстояния

if (distanceToTarget > visionRange) return false;

// Проверка ширины поля зрения (проще чем угол в 2D)

if (Mathf.Abs(directionToTarget.y) > visionWidth) return false;

// Проверка препятствий

RaycastHit2D hit = Physics2D.Raycast(

transform.position,

directionToTarget,

distanceToTarget,

obstacleLayers);

if (hit.collider != null) return false;

return ((1 << target.gameObject.layer) & targetLayer) != 0;

}

private void OnDrawGizmosSelected()

{

Gizmos.color = Color.yellow;

Vector2 size = new Vector2(visionRange, visionWidth \* 2);

Gizmos.DrawWireCube(

transform.position + transform.right \* visionRange / 2,

size);

}

}

Программный код 12.

using UnityEngine;

public class EnemyHealth : MonoBehaviour

{

[Header("Health Settings")]

[SerializeField] private int maxHealth = 3;

[SerializeField] private int currentHealth;

[SerializeField] private WolfState state;

private EndGame endGame;

[Header("Health Bar Settings")]

[SerializeField] private Transform healthBarParent; // Родительский объект для полоски

[SerializeField] private Transform healthBarFill; // Красная заполняемая часть

[SerializeField] private Vector3 healthBarOffset = new Vector3(0, 1.5f, 0);

private Vector3 initialScale;

private void Awake()

{

endGame = FindAnyObjectByType<EndGame>();

currentHealth = maxHealth;

if (healthBarFill != null)

{

initialScale = healthBarFill.localScale;

UpdateHealthBarPosition();

}

}

public void TakeDamage(int damage)

{

currentHealth -= damage;

currentHealth = Mathf.Max(currentHealth, 0);

UpdateHealthBar();

if (currentHealth <= 0)

{

Die();

}

}

private void UpdateHealthBar()

{

if (healthBarFill != null)

{

float healthPercent = (float)currentHealth / maxHealth;

healthBarFill.localScale = new Vector3(

initialScale.x \* healthPercent,

initialScale.y,

initialScale.z

);

}

}

private void UpdateHealthBarPosition()

{

if (healthBarParent != null)

{

healthBarParent.position = transform.position + healthBarOffset;

healthBarParent.rotation = Quaternion.identity;

}

}

private void Die()

{

if (healthBarParent != null)

{

Destroy(healthBarParent.gameObject);

}

if(gameObject.name == "WolfBoss")

{

endGame.ShowWinMessage();

}

state.Die();

}

private void LateUpdate()

{

UpdateHealthBarPosition();

}

}

Программный код 13.

using UnityEngine;

public class AppleItem : InteractItem

{

[SerializeField] private float speedMultiplier = 1.3f;

[SerializeField] private float jumpMultiplier = 1.2f;

[SerializeField] private float damageMultiplier = 2f;

[SerializeField] private float duration = 20f;

public override void Use(PlayerInput player)

{

var effects = player.GetComponent<PlayerEffects>();

if (effects != null)

{

effects.ApplyEffect(new SpeedEffect(speedMultiplier, duration));

effects.ApplyEffect(new JumpEffect(jumpMultiplier, duration));

effects.ApplyEffect(new DamageEffect(damageMultiplier, duration));

}

}

}

Программный код 14.

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

public class GlobalItemManager : MonoBehaviour

{

public static GlobalItemManager Instance;

[Header("Settings")]

[SerializeField] private float defaultRespawnTime = 30f;

[SerializeField] private Vector2 randomTimeOffset = new Vector2(-5f, 5f);

private Dictionary<InteractItem, ItemRespawnData> items = new Dictionary<InteractItem, ItemRespawnData>();

private void Awake()

{

if (Instance == null)

{

Instance = this;

DontDestroyOnLoad(gameObject);

}

else

{

Destroy(gameObject);

}

}

public void RegisterItem(InteractItem item, Vector3 originalPosition, Transform customSpawnPoint = null)

{

if (!items.ContainsKey(item))

{

items.Add(item, new ItemRespawnData(originalPosition, customSpawnPoint));

item.OnPickup += () => StartCoroutine(RespawnItem(item));

}

}

private IEnumerator RespawnItem(InteractItem item)

{

if (!items.TryGetValue(item, out var data)) yield break;

float delay = defaultRespawnTime + Random.Range(randomTimeOffset.x, randomTimeOffset.y);

yield return new WaitForSeconds(delay);

if (item != null)

{

item.transform.position = data.customSpawnPoint != null ?

data.customSpawnPoint.position : data.originalPosition;

item.gameObject.SetActive(true);

if (item is IResettable resettable)

resettable.ResetState();

}

}

private class ItemRespawnData

{

public Vector3 originalPosition;

public Transform customSpawnPoint;

public ItemRespawnData(Vector3 position, Transform spawnPoint)

{

originalPosition = position;

customSpawnPoint = spawnPoint;

}

}

}

Программный код 15.

using UnityEngine;

using System;

public abstract class InteractItem : MonoBehaviour

{

[Header("Base Item Settings")]

[SerializeField] private string itemName;

[SerializeField] private Sprite icon;

[SerializeField] private int maxStack = 1;

[Header("Respawn Settings")]

[SerializeField] private bool respawnable = true;

[SerializeField] private Transform customRespawnPoint;

private Vector3 originalPosition;

public string ItemName => itemName;

public Sprite Icon => icon;

public int MaxStack => maxStack;

private void Start()

{

originalPosition = transform.position;

if (respawnable)

GlobalItemManager.Instance?.RegisterItem(this, originalPosition, customRespawnPoint);

}

// Событие подбора предмета

public event Action OnPickup = delegate { };

public abstract void Use(PlayerInput player);

public virtual void Pickup()

{

OnPickup?.Invoke();

gameObject.SetActive(false);

}

public virtual void Drop(Vector3 position)

{

transform.position = position;

gameObject.SetActive(true);

}

}

// Интерфейс для сброса состояния

public interface IResettable

{

void ResetState();

}

Программный код 16.

using UnityEngine;

public class PlantainItem : InteractItem

{

[SerializeField] private int healAmount = 25;

public override void Use(PlayerInput player)

{

if (player.TryGetComponent(out PlayerHealth health))

{

health.Heal(healAmount);

}

}

}

Программный код 17.

using UnityEngine;

public class StickItem : InteractItem

{

[Header("Throwing Settings")]

[SerializeField] private float throwForce = 12f;

[SerializeField] private GameObject stickProjectilePrefab;

[SerializeField] private float throwOffset = 0.5f;

[SerializeField] private int damage = 5;

public override void Use(PlayerInput player)

{

if (stickProjectilePrefab == null)

{

Debug.LogError("Stick projectile prefab is not assigned!");

return;

}

// Определяем направление броска

Vector3 throwDirection = player.isFacingRight ? Vector3.right : Vector3.left;

Vector3 spawnPosition = player.transform.position + throwDirection \* throwOffset;

// Создаем снаряд

GameObject stick = Instantiate(

stickProjectilePrefab,

spawnPosition,

Quaternion.identity

);

Destroy(stick, 1f);

// Инициализируем снаряд

StickProjectile projectile = stick.GetComponent<StickProjectile>();

if (projectile != null)

{

projectile.SetDamage(damage); // Устанавливаем урон

projectile.Throw(throwDirection, throwForce);

}

else

{

Debug.LogError("StickProjectile component missing on prefab!");

}

}

}

Программный код 18.

using UnityEngine;

public class StickProjectile : MonoBehaviour

{

[Header("Stick Settings")]

[SerializeField] private float lifeTime = 2f;

[SerializeField] private float destroyDelayAfterHit = 0.1f;

[SerializeField] private GameObject hitEffect;

[SerializeField] private int damage;

private SoundManager soundManager;

private Rigidbody2D rb;

private bool hasHit = false;

private void Awake()

{

soundManager = FindAnyObjectByType<SoundManager>();

rb = GetComponent<Rigidbody2D>();

Destroy(gameObject, lifeTime);

}

public void SetDamage(int newDamage)

{

damage = newDamage;

}

public void Throw(Vector2 direction, float force)

{

rb.AddForce(direction \* force, ForceMode2D.Impulse);

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (hasHit) return;

//if (collision.gameObject.tag ==("Player"))

//return;

//hasHit = true;

if (collision.gameObject.tag == ("Enemy"))

{

if(collision.transform.GetComponent<Stun>() != null)

{

Stun stun = collision.transform.GetComponent<Stun>();

stun.OnStun();

return;

}

var health = collision.gameObject.GetComponent<EnemyHealth>();

if (health != null)

{

health.TakeDamage(damage);

}

soundManager.PlayHit();

Destroy(gameObject);

}

//if (hitEffect != null)

{

//Instantiate(hitEffect, transform.position, Quaternion.identity);

}

rb.linearVelocity = Vector2.zero;

rb.isKinematic = true;

GetComponent<Collider2D>().enabled = false;

GetComponent<SpriteRenderer>().enabled = false;

Destroy(gameObject, destroyDelayAfterHit);

}

}

Программный код 19.

using UnityEngine;

public class StoneItem : InteractItem

{

[Header("Throwing Settings")]

[SerializeField] private float throwForce = 15f;

[SerializeField] private GameObject stoneProjectilePrefab;

[SerializeField] private float throwOffset = 0.5f;

public override void Use(PlayerInput player)

{

if (stoneProjectilePrefab == null)

{

Debug.LogError("Stone projectile prefab is not assigned!");

return;

}

// Определяем направление броска

Vector3 throwDirection = player.isFacingRight ? Vector3.right : Vector3.left;

Vector3 spawnPosition = player.transform.position + throwDirection \* throwOffset;

// Создаем снаряд

GameObject stone = Instantiate(

stoneProjectilePrefab,

spawnPosition,

Quaternion.identity

);

Destroy(stone, 1f);

// Инициализируем снаряд

StoneProjectile projectile = stone.GetComponent<StoneProjectile>();

if (projectile != null)

{

projectile.Throw(throwDirection, throwForce);

}

else

{

Debug.LogError("StoneProjectile component missing on prefab!");

}

}

}

Программный код 20.

using UnityEngine;

public class StoneProjectile : MonoBehaviour

{

[Header("Stone Settings")]

[SerializeField] private int damage = 10;

[SerializeField] private float lifeTime = 3f;

[SerializeField] private float destroyDelayAfterHit = 0.1f;

[SerializeField] private GameObject hitEffect;

private SoundManager soundManager;

private Rigidbody2D rb;

private bool hasHit = false;

private void Awake()

{

soundManager = FindAnyObjectByType<SoundManager>();

rb = GetComponent<Rigidbody2D>();

Destroy(gameObject, lifeTime);

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (hasHit) return;

if (collision.gameObject.tag == ("Enemy"))

{

if (collision.transform.GetComponent<Stun>() != null)

{

Stun stun = collision.transform.GetComponent<Stun>();

stun.OnStun();

return;

}

var health = collision.gameObject.GetComponent<EnemyHealth>();

if (health != null)

{

health.TakeDamage(damage);

}

soundManager.PlayHit();

Destroy(gameObject);

}

rb.linearVelocity = Vector2.zero;

rb.isKinematic = true;

GetComponent<Collider2D>().enabled = false;

GetComponent<SpriteRenderer>().enabled = false;

Destroy(gameObject, destroyDelayAfterHit);

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (hasHit) return;

if (collision.gameObject.tag ==("Enemy"))

{

var health = collision.gameObject.GetComponent<EnemyHealth>();

if (health != null)

{

health.TakeDamage(damage);

}

Destroy(gameObject);

}

rb.linearVelocity = Vector2.zero;

rb.isKinematic = true;

GetComponent<Collider2D>().enabled = false;

GetComponent<SpriteRenderer>().enabled = false;

Destroy(gameObject, destroyDelayAfterHit);

}

public void Throw(Vector2 direction, float force)

{

rb.AddForce(direction \* force, ForceMode2D.Impulse);

}

}

Программный код 21.

// Базовый класс эффекта

public abstract class TimedEffect

{

public float Duration { get; protected set; }

public bool IsActive => Duration > 0;

public virtual void ApplyEffect(PlayerInput player)

{

// Применяем эффект

}

public virtual void RemoveEffect(PlayerInput player)

{

// Отменяем эффект

}

public void Update(float deltaTime)

{

if (IsActive)

{

Duration -= deltaTime;

}

}

}

// Конкретные эффекты

public class SpeedEffect : TimedEffect

{

private readonly float multiplier;

private float originalSpeed;

public SpeedEffect(float multiplier, float duration)

{

this.multiplier = multiplier;

Duration = duration;

}

public override void ApplyEffect(PlayerInput player)

{

originalSpeed = player.moveSpeed;

player.moveSpeed \*= multiplier;

}

public override void RemoveEffect(PlayerInput player)

{

player.moveSpeed = originalSpeed;

}

}

public class JumpEffect : TimedEffect

{

private float multiplier;

private float originalJumpForce;

public JumpEffect(float multiplier, float duration)

{

this.multiplier = multiplier;

Duration = duration;

}

public override void ApplyEffect(PlayerInput player)

{

//originalJumpForce = player.jumpForce;

//player.jumpForce \*= multiplier;

player.ModifyJump(multiplier);

}

public override void RemoveEffect(PlayerInput player)

{

//player.jumpForce = originalJumpForce;

player.ModifyJump(1f/multiplier);

}

}

public class DamageEffect : TimedEffect

{

private readonly float multiplier;

public DamageEffect(float multiplier, float duration)

{

this.multiplier = multiplier;

Duration = duration;

}

public override void ApplyEffect(PlayerInput player)

{

}

public override void RemoveEffect(PlayerInput player)

{

}

}

Программный код 22.

using UnityEngine;

using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour

{

private SoundManager soundManager;

private void Start()

{

soundManager = FindAnyObjectByType<SoundManager>();

}

// Метод для кнопки "Играть"

public void PlayGame()

{

SceneManager.LoadScene(1); // Загружаем сцену с индексом 1 (GameScene)

soundManager.PlayButtonClick();

}

// Метод для кнопки "Выход"

public void QuitGame()

{

soundManager.PlayButtonClick();

Application.Quit(); // Закрываем приложение

#if UNITY\_EDITOR

UnityEditor.EditorApplication.isPlaying = false;

#endif

}

}

Программный код 23.

using UnityEngine;

using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour

{

[Header("UI References")]

[SerializeField] private GameObject pauseMenuUI;

[SerializeField] private GameObject pauseButton;

[SerializeField] private string mainMenuScene = "MainMenu";

private bool isPaused = false;

private SoundManager soundManager;

private void Start()

{

soundManager = FindAnyObjectByType<SoundManager>();

}

void Update()

{

if (Input.GetKeyDown(KeyCode.Escape))

{

// Проигрываем звук при открытии паузы

if (soundManager != null)

soundManager.PlayButtonClick();

if (isPaused)

Resume();

else

Pause();

}

}

public void TogglePause()

{

// Проигрываем звук при открытии паузы

if (soundManager != null)

soundManager.PlayButtonClick();

if (isPaused)

Resume();

else

Pause();

}

public void Pause()

{

pauseMenuUI.SetActive(true);

pauseButton.SetActive(false);

Time.timeScale = 0f;

isPaused = true;

AudioListener.pause = true;

// Проигрываем звук при открытии паузы

if (soundManager != null)

soundManager.PlayButtonClick();

}

public void Resume()

{

pauseMenuUI.SetActive(false);

pauseButton.SetActive(true);

Time.timeScale = 1f;

isPaused = false;

AudioListener.pause = false;

// Проигрываем звук при продолжении игры

if (soundManager != null)

soundManager.PlayButtonClick();

}

public void LoadMainMenu()

{

Time.timeScale = 1f;

// Проигрываем звук перед загрузкой меню

if (soundManager != null)

soundManager.PlayButtonClick();

SceneManager.LoadScene(mainMenuScene);

}

public void QuitGame()

{

// Проигрываем звук перед выходом

if (soundManager != null)

soundManager.PlayButtonClick();

Application.Quit();

#if UNITY\_EDITOR

UnityEditor.EditorApplication.isPlaying = false;

#endif

}

}

Программный код 24.

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class Blackout : MonoBehaviour

{

public Image darkImage;

public float fadeDuration = 1f;

private bool isFading = false;

public void StartBlackout(bool isTeleporting, Transition transition = null)

{

StartCoroutine(FadeDarkScreen(isTeleporting, transition));

}

IEnumerator FadeDarkScreen(bool isTeleporting, Transition targetTransition)

{

isFading = true;

// Затемнение

float timer = 0f;

while (timer < fadeDuration)

{

timer += Time.deltaTime;

darkImage.color = new Color(0, 0, 0, Mathf.Lerp(0, 1, timer / fadeDuration));

yield return null;

}

// Телепортация

if (isTeleporting && targetTransition != null)

{

targetTransition.Transfer();

// Осветление

timer = 0f;

while (timer < fadeDuration)

{

timer += Time.deltaTime;

darkImage.color = new Color(0, 0, 0, Mathf.Lerp(1, 0, timer / fadeDuration));

yield return null;

}

isFading = false;

}

}

}

Программный код 25.

using UnityEngine;

public class DeadZone : MonoBehaviour

{

private void OnCollisionEnter2D(Collision2D collision)

{

if(collision.transform.GetComponent<PlayerHealth>() != null)

{

PlayerHealth player = collision.transform.GetComponent<PlayerHealth>();

player.Die();

}

}

}

Программный код 26.

using System;

using UnityEngine;

public enum ItemID

{

Stick = 0,

Stone = 1,

Plantain = 2,

Apple = 3

}

[System.Serializable]

public class InventorySlot

{

public ItemID itemId;

public InteractItem itemPrefab;

public int maxStack;

public int currentAmount;

public bool IsFull => currentAmount >= maxStack;

public bool IsEmpty => currentAmount <= 0;

}

public class Inventory : MonoBehaviour

{

[SerializeField] private InventorySlot[] slots;

public event Action OnInventoryChanged;

private void Start()

{

slots = new InventorySlot[]

{

new InventorySlot { itemId = ItemID.Stick, maxStack = 20 },

new InventorySlot { itemId = ItemID.Stone, maxStack = 10 },

new InventorySlot { itemId = ItemID.Plantain, maxStack = 6 },

new InventorySlot { itemId = ItemID.Apple, maxStack = 6 }

};

}

public bool AddItem(ItemID itemId, InteractItem itemPrefab)

{

InventorySlot slot = GetSlot(itemId);

if (slot == null) return false;

if (slot.IsFull) return false;

if (slot.itemPrefab == null)

slot.itemPrefab = itemPrefab;

slot.currentAmount++;

OnInventoryChanged?.Invoke();

return true;

}

public bool RemoveItem(ItemID itemId, int amount = 1)

{

InventorySlot slot = GetSlot(itemId);

if (slot == null || slot.currentAmount < amount)

return false;

slot.currentAmount -= amount;

OnInventoryChanged?.Invoke();

return true;

}

public bool HasItem(ItemID itemId, int amount = 1)

{

InventorySlot slot = GetSlot(itemId);

return slot != null && slot.currentAmount >= amount;

}

public int GetItemCount(ItemID itemType)

{

// Реализуйте получение количества предметов

// Пример:

foreach (var slot in slots)

{

if (slot.itemId == itemType)

return slot.currentAmount;

}

return 0;

}

public void UseItem(ItemID itemId)

{

InventorySlot slot = GetSlot(itemId);

if (slot == null || slot.IsEmpty || slot.itemPrefab == null)

return;

var player = GetComponent<PlayerInput>();

var newItem = Instantiate(slot.itemPrefab);

newItem.Use(player);

RemoveItem(itemId);

Destroy(newItem, 5f);

}

private InventorySlot GetSlot(ItemID itemId)

{

foreach (var slot in slots)

if (slot.itemId == itemId)

return slot;

return null;

}

}

Программный код 27.

using UnityEngine;

using UnityEngine.UI;

using System.Collections.Generic;

public class InventoryUI : MonoBehaviour

{

[System.Serializable]

public class ItemSlot

{

public Image iconImage;

public GameObject arrow;

public Text countText;

public ItemID itemType;

[SerializeField] private Sprite itemSprite; // Добавляем поле для спрайта в инспекторе

public Sprite Sprite => itemSprite;

}

[Header("Settings")]

[SerializeField] private List<ItemSlot> itemSlots;

[SerializeField] private Sprite emptySlotSprite;

[SerializeField] private Color normalColor = new Color(1, 1, 1, 0.5f);

[SerializeField] private Color selectedColor = Color.white;

private Inventory inventory;

private ItemID? currentSelectedItem = null;

private void Start()

{

inventory = FindObjectOfType<Inventory>();

if (inventory == null)

{

Debug.LogError("Inventory not found!");

return;

}

// Подписываемся на событие изменения инвентаря

inventory.OnInventoryChanged += UpdateAllSlots;

UpdateAllSlots();

}

private void OnDestroy()

{

// Отписываемся при уничтожении объекта

if (inventory != null)

{

inventory.OnInventoryChanged -= UpdateAllSlots;

}

}

public void UpdateAllSlots()

{

foreach (var slot in itemSlots)

{

UpdateSlot(slot);

}

}

private void UpdateSlot(ItemSlot slot)

{

int itemCount = inventory.GetItemCount(slot.itemType);

bool hasItem = itemCount > 0;

bool isSelected = currentSelectedItem == slot.itemType;

// Обновляем иконку

slot.iconImage.sprite = hasItem ? GetItemSprite(slot) : emptySlotSprite;

slot.iconImage.color = hasItem ? (isSelected ? selectedColor : Color.white) : normalColor;

// Обновляем счетчик

if (slot.countText != null)

{

slot.countText.text = hasItem ? itemCount.ToString() : "";

slot.countText.gameObject.SetActive(hasItem);

}

// Обновляем стрелку

if (slot.arrow != null)

{

slot.arrow.SetActive(isSelected && hasItem);

}

}

public void SelectItem(ItemID itemType)

{

currentSelectedItem = itemType;

UpdateAllSlots();

}

private Sprite GetItemSprite(ItemSlot slot)

{

// Сначала проверяем спрайт назначенный в инспекторе

if (slot.Sprite != null)

return slot.Sprite;

// Если нет - пробуем загрузить из ресурсов

var loadedSprite = Resources.Load<Sprite>($"Items/{slot.itemType}");

if (loadedSprite != null)

return loadedSprite;

Debug.LogWarning($"Sprite for {slot.itemType} not found!");

return emptySlotSprite;

}

}

Программный код 28.

using UnityEngine;

public class PlayerAnimationController : MonoBehaviour

{

[Header("References")]

[SerializeField] private Animator animator;

[SerializeField] private PlayerHealth health;

[SerializeField] private PlayerEffects effects;

[SerializeField] private PlayerInput playerInput;

[SerializeField] private Blackout blackout;

private PauseMenu pauseMenu;

private EndGame endgame;

[Header("Animation Parameters")]

[SerializeField] private string speedParam = "Speed";

[SerializeField] private string isGroundedParam = "IsGrounded";

[SerializeField] private string verticalSpeedParam = "VerticalSpeed";

[SerializeField] private string healthStateParam = "HealthState";

[SerializeField] private string isBuffedParam = "IsBuffed";

[SerializeField] private string isDeadParam = "IsDead";

[Header("Settings")]

[SerializeField] private float lowHPThreshold = 0.5f;

private void Awake()

{

endgame = FindAnyObjectByType<EndGame>();

pauseMenu = FindAnyObjectByType<PauseMenu>();

// Автоматическое получение компонентов, если не заданы в инспекторе

if (!animator) animator = GetComponent<Animator>();

if (!health) health = GetComponent<PlayerHealth>();

if (!effects) effects = GetComponent<PlayerEffects>();

if (!playerInput) playerInput = GetComponent<PlayerInput>();

}

private void Update()

{

if (animator == null || playerInput == null) return;

UpdateMovementAnimations();

if (health != null)

{

UpdateHealthState();

}

if (effects != null)

{

UpdateBuffedState();

}

}

private void UpdateMovementAnimations()

{

// Обновляем параметры движения

animator.SetFloat(speedParam, Mathf.Abs(playerInput.CurrentVelocity.x));

animator.SetBool(isGroundedParam, playerInput.isGrounded);

animator.SetFloat(verticalSpeedParam, playerInput.CurrentVelocity.y);

}

private void UpdateHealthState()

{

// 0 = FullHP, 1 = LowHP

bool isLowHP = health.currentHealth < health.maxHealth \* lowHPThreshold;

animator.SetInteger(healthStateParam, isLowHP ? 1 : 0);

if (health.currentHealth <= 0)

{

Dead();

}

}

private void UpdateBuffedState()

{

animator.SetBool(isBuffedParam, effects.HasBuff);

}

public void Dead()

{

endgame.ShowLoseMessage();

animator.SetBool(isDeadParam, true);

blackout.StartBlackout(false);

}

public void RestartGame()

{

pauseMenu.LoadMainMenu();

}

}

Программный код 29.

using System.Collections.Generic;

using UnityEngine;

public class PlayerEffects : MonoBehaviour

{

private List<TimedEffect> activeEffects = new List<TimedEffect>();

private PlayerInput playerInput;

// Свойство, которое автоматически проверяет наличие активных баффов

public bool HasBuff => activeEffects.Count > 0;

private void Awake()

{

playerInput = GetComponent<PlayerInput>();

}

private void Update()

{

// Обрабатываем эффекты в обратном порядке

for (int i = activeEffects.Count - 1; i >= 0; i--)

{

activeEffects[i].Update(Time.deltaTime);

if (!activeEffects[i].IsActive)

{

activeEffects[i].RemoveEffect(playerInput);

activeEffects.RemoveAt(i);

Debug.Log("Effect removed. HasBuff: " + HasBuff); // Для отладки

}

}

}

public void ApplyEffect(TimedEffect effect)

{

effect.ApplyEffect(playerInput);

activeEffects.Add(effect);

Debug.Log("Effect applied. HasBuff: " + HasBuff); // Для отладки

}

}

Программный код 30.

using UnityEngine;

using UnityEngine.UI;

public class PlayerHealth : MonoBehaviour

{

[Header("Health Settings")]

public int maxHealth = 100;

public int currentHealth = 100;

[Header("UI Settings")]

[SerializeField] private Image healthBarFill; // Ссылка на Image компонент полоски здоровья

[SerializeField] private bool isHealthBarRightToLeft = false; // Направление заполнения

[SerializeField] private PlayerAnimationController playerAnimationController;

[SerializeField, HideInInspector] private PlayerInput playerInput;

[Header("Push Settings")]

[SerializeField] private float pushRecoveryTime = 0.5f;

[SerializeField] private float pushDeceleration = 15f;

[Header("Effects")]

[SerializeField] private ParticleSystem hitParticles;

[SerializeField] private AudioClip hitSound;

private Rigidbody2D rb;

private Vector2 pushVelocity;

private bool isPushed;

private float pushEndTime;

private SoundManager soundManager;

private void Awake()

{

playerInput = GetComponent<PlayerInput>();

playerAnimationController = GetComponentInChildren<PlayerAnimationController>();

rb = GetComponent<Rigidbody2D>();

InitializeHealthBar();

UpdateHealthBar();

soundManager = FindAnyObjectByType<SoundManager>();

}

public void TakeDamageWithPush(int damage, Vector2 pushDirection, float pushForce)

{

currentHealth -= damage;

currentHealth = Mathf.Max(currentHealth, 0);

UpdateHealthBar();

// Визуальные эффекты

if (hitParticles != null)

{

hitParticles.transform.position = transform.position;

hitParticles.Play();

}

if (hitSound != null)

{

AudioSource.PlayClipAtPoint(hitSound, transform.position);

}

ApplyPush(pushDirection, pushForce);

if (currentHealth <= 0)

{

Die();

}

}

private void UpdateHealthBar()

{

if (healthBarFill != null)

{

float healthPercent = (float)currentHealth / maxHealth;

healthBarFill.fillAmount = healthPercent;

if (healthPercent < 0.3f) // Порог низкого HP

{

soundManager.PlayHeartbeat();

}

}

}

private void ApplyPush(Vector2 direction, float force)

{

playerInput.isPushed = true;

pushVelocity = direction.normalized \* force;

isPushed = true;

pushEndTime = Time.time + pushRecoveryTime;

}

private void InitializeHealthBar()

{

if (healthBarFill != null)

{

// Настройка якорей и пивота с небольшим смещением вправо

float xOffset = 0.01f; // Значение от 0 до 1 (10% смещение)

healthBarFill.rectTransform.anchorMin = new Vector2(xOffset, 0.5f);

healthBarFill.rectTransform.anchorMax = new Vector2(xOffset, 0.5f);

healthBarFill.rectTransform.pivot = new Vector2(0, 0.5f);

// Дополнительное смещение через localPosition

healthBarFill.rectTransform.anchoredPosition = new Vector2(6f, 0); // Пиксельное смещение

// Для заполнения слева направо

healthBarFill.fillOrigin = (int)Image.OriginHorizontal.Left;

}

}

private void FixedUpdate()

{

if (isPushed)

{

rb.linearVelocity = pushVelocity;

pushVelocity = Vector2.MoveTowards(pushVelocity, Vector2.zero, pushDeceleration \* Time.fixedDeltaTime);

if (Time.time >= pushEndTime)

{

isPushed = false;

playerInput.isPushed = false;

rb.linearVelocity = Vector2.zero;

}

}

}

public void Heal(int amount)

{

currentHealth = Mathf.Min(currentHealth + amount, maxHealth);

UpdateHealthBar();

}

public void Die()

{

playerAnimationController.Dead();

playerInput.enabled = false;

gameObject.GetComponent<Rigidbody2D>().simulated = false;

}

}

Программный код 31.

using UnityEngine;

public class PlayerInput : MonoBehaviour

{

[Header("Movement Settings")]

[SerializeField] public float moveSpeed = 5f;

[SerializeField] public float jumpForce = 10f;

[SerializeField] private float deceleration = 5f; // Скорость замедления (5 - нормально, 10 - резко)

[Header("Ground Check")]

[SerializeField] private Transform groundCheck;

[SerializeField] private float groundCheckDistance = 0.2f;

[SerializeField] private LayerMask groundLayer;

[SerializeField] private bool drawDebugRays = true;

[SerializeField] Inventory inventory;

[SerializeField] UsingZone usingZone;

[SerializeField] InventoryUI inventoryUI;

public Vector2 CurrentVelocity => rb.linearVelocity;

private int currentItemSelected = -1; // Начинаем с -1 (ничего не выбрано)

private Rigidbody2D rb;

private SoundManager soundManager;

private float horizontalInput;

public bool isFacingRight = true;

public bool isGrounded;

public bool isPushed;

private void Awake()

{

rb = GetComponent<Rigidbody2D>();

soundManager = FindAnyObjectByType<SoundManager>();

}

private void Update()

{

TakeItem();

HandleItemSelection(); // Теперь это просто обновляет currentItemSelected

UseItem(currentItemSelected); // Используем сохраненное значение

// Остальной код без изменений...

CheckGrounded();

horizontalInput = Input.GetAxisRaw("Horizontal");

if (Mathf.Abs(horizontalInput) > 0.1f && isGrounded)

{

soundManager.PlayFootsteps();

}

else

{

soundManager.StopFootsteps();

}

if ((Input.GetKeyDown(KeyCode.Space) || Input.GetKeyDown(KeyCode.UpArrow) || Input.GetKeyDown(KeyCode.W))

&& isGrounded)

{

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

}

if (horizontalInput > 0 && !isFacingRight)

{

Flip();

}

else if (horizontalInput < 0 && isFacingRight)

{

Flip();

}

}

public void TakeItem()

{

if (Input.GetKeyDown(KeyCode.E) && usingZone.isInZone && usingZone.currentObject != null)

{

ItemID itemId = GetItemId(usingZone.currentObject);

if (inventory.AddItem(itemId, usingZone.currentObject))

{

usingZone.currentObject.Pickup();

usingZone.currentObject = null;

}

}

}

private ItemID GetItemId(InteractItem item)

{

if (item is StickItem) return ItemID.Stick;

if (item is StoneItem) return ItemID.Stone;

if (item is PlantainItem) return ItemID.Plantain;

if (item is AppleItem) return ItemID.Apple;

throw new System.Exception("Unknown item type");

}

public void UseItem(int num)

{

if (Input.GetKeyDown(KeyCode.F))

{

ItemID itemId = (ItemID)num;

if (inventory.HasItem(itemId))

{

inventory.UseItem(itemId);

soundManager.PlayThrow();

}

}

}

private void HandleItemSelection()

{

if (Input.GetKeyDown(KeyCode.Alpha1))

{

currentItemSelected = (int)ItemID.Stick;

inventoryUI.SelectItem(ItemID.Stick);

}

else if (Input.GetKeyDown(KeyCode.Alpha2))

{

currentItemSelected = (int)ItemID.Stone;

inventoryUI.SelectItem(ItemID.Stone);

}

else if (Input.GetKeyDown(KeyCode.Alpha3))

{

currentItemSelected = (int)ItemID.Plantain;

inventoryUI.SelectItem(ItemID.Plantain);

}

else if (Input.GetKeyDown(KeyCode.Alpha4))

{

currentItemSelected = (int)ItemID.Apple;

inventoryUI.SelectItem(ItemID.Apple);

}

}

public void ModifySpeed(float multiplier)

{

moveSpeed \*= multiplier;

}

public void ModifyJump(float multiplier)

{

jumpForce \*= multiplier;

}

private void FixedUpdate()

{

if (isPushed) return; // Не управляем во время толчка

// Сохраняем текущую Y-скорость (гравитация, прыжки)

float yVelocity = rb.linearVelocity.y;

// Мгновенно устанавливаем X-скорость при вводе

float xVelocity = horizontalInput \* moveSpeed;

// Плавное затухание скорости при отсутствии ввода

if (horizontalInput == 0)

{

xVelocity = Mathf.Lerp(rb.linearVelocity.x, 0, deceleration \* Time.fixedDeltaTime);

if (Mathf.Abs(xVelocity) < 0.1f) xVelocity = 0; // Полная остановка

}

rb.linearVelocity = new Vector2(xVelocity, yVelocity);

}

private void CheckGrounded()

{

RaycastHit2D hit = Physics2D.Raycast(

groundCheck.position,

Vector2.down,

groundCheckDistance,

groundLayer);

isGrounded = hit.collider != null;

if (drawDebugRays)

{

Debug.DrawRay(groundCheck.position, Vector2.down \* groundCheckDistance,

isGrounded ? Color.green : Color.red);

}

}

private void Flip()

{

isFacingRight = !isFacingRight;

Vector3 scale = transform.localScale;

scale.x \*= -1;

transform.localScale = scale;

}

}

Программный код 32.

using UnityEngine;

public class Transition : MonoBehaviour

{

public Transform Player;

public Transform nextPoint;

[SerializeField] private Blackout blackout;

private bool isActiveTransition = false;

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.CompareTag("Player") && !isActiveTransition)

{

isActiveTransition = true;

blackout.StartBlackout(true, this); // Передаем текущий Transition

}

}

public void Transfer()

{

Debug.Log("Teleporting to: " + nextPoint.position);

Player.position = new Vector3(

nextPoint.position.x,

nextPoint.position.y,

-6f

);

isActiveTransition = false;

}

}

Программный код 33.

using UnityEngine;

public class UsingZone : MonoBehaviour

{

public bool isInZone = false;

public InteractItem currentObject = null;

private void OnTriggerEnter2D(Collider2D collision)

{

var item = collision.GetComponent<InteractItem>();

if (item != null)

{

isInZone = true;

currentObject = item;

ShowPickupPrompt(true, collision);

}

}

private void ShowPickupPrompt(bool show, Collider2D collision)

{

collision.transform.GetChild(0).gameObject.SetActive(show);

}

private void OnTriggerExit2D(Collider2D collision)

{

if (collision.CompareTag("InteractItem"))

{

isInZone = false;

currentObject = null;

if (collision.transform.GetChild(0) != null)

{

ShowPickupPrompt(false, collision);

}

}

}

}

Программный код 34.

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class EndGame : MonoBehaviour

{

[SerializeField] Text textField;

private PlayerAnimationController playerAnimationController;

private void Start()

{

playerAnimationController = FindAnyObjectByType<PlayerAnimationController>();

}

public void ShowWinMessage()

{

if (textField != null)

{

textField.text = "Победа";

// Устанавливаем непрозрачный белый цвет (альфа = 1.0)

textField.color = new Color(textField.color.r, textField.color.g, textField.color.b, 1f);

StartCoroutine(RestartAfterDelay());

}

}

public void ShowLoseMessage()

{

if (textField != null)

{

textField.text = "Поражение";

// Устанавливаем непрозрачный белый цвет (альфа = 1.0)

textField.color = new Color(textField.color.r, textField.color.g, textField.color.b, 1f);

}

}

private IEnumerator RestartAfterDelay()

{

yield return new WaitForSeconds(3f);

playerAnimationController.RestartGame();

}

}

Программный код 35.

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class ShowLevelName : MonoBehaviour

{

[SerializeField] private string levelName;

[SerializeField] private Text title;

[SerializeField] private float fadeDuration = 1.5f;

[SerializeField] private float displayDuration = 3f;

private bool isAnimating = false;

private Coroutine currentAnimation;

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.CompareTag("Player") && !isAnimating)

{

if (currentAnimation != null)

{

StopCoroutine(currentAnimation);

}

currentAnimation = StartCoroutine(ShowAndHideText());

}

}

private IEnumerator ShowAndHideText()

{

isAnimating = true;

// Установка текста

title.text = levelName;

// Плавное появление

yield return FadeText(0f, 1f);

// Ждем пока текст виден

yield return new WaitForSeconds(displayDuration);

// Плавное исчезание

yield return FadeText(1f, 0f);

isAnimating = false;

Destroy(gameObject);

}

private IEnumerator FadeText(float startAlpha, float endAlpha)

{

float timer = 0f;

Color textColor = title.color;

while (timer < fadeDuration)

{

timer += Time.deltaTime;

float alpha = Mathf.Lerp(startAlpha, endAlpha, timer / fadeDuration);

title.color = new Color(textColor.r, textColor.g, textColor.b, alpha);

yield return null;

}

}

}

Программный код 36.

using UnityEngine;

public class SoundManager : MonoBehaviour

{

[Header("Player Sounds")]

public AudioClip wolfAttackSound;

public AudioClip hedgehogAttackSound;

public AudioClip footstepsSound;

public AudioClip throwSound;

public AudioClip buttonClickSound;

public AudioClip heartbeatSound;

public AudioClip hitSound;

[Header("Music")]

public AudioClip backgroundMusic;

private AudioSource musicSource;

private AudioSource sfxSource;

private AudioSource footstepsSource; // Отдельный источник для шагов

// Флаги для проверки playing

private bool isFootstepsPlaying = false;

private bool isHeartbeatPlaying = false;

private void Awake()

{

musicSource = gameObject.AddComponent<AudioSource>();

sfxSource = gameObject.AddComponent<AudioSource>();

footstepsSource = gameObject.AddComponent<AudioSource>(); // Для шагов

// Настройка параметров

musicSource.loop = true;

musicSource.clip = backgroundMusic;

footstepsSource.loop = true; // Шаги зациклены

footstepsSource.clip = footstepsSound;

}

// Обычные звуки (без проверки)

public void PlayWolfAttack() => sfxSource.PlayOneShot(wolfAttackSound);

public void PlayHedgehogAttack() => sfxSource.PlayOneShot(hedgehogAttackSound);

public void PlayThrow() => sfxSource.PlayOneShot(throwSound);

public void PlayHit() => sfxSource.PlayOneShot(hitSound);

public void PlayButtonClick() => sfxSource.PlayOneShot(buttonClickSound);

// Шаги с проверкой

public void PlayFootsteps()

{

if (!isFootstepsPlaying)

{

footstepsSource.Play();

isFootstepsPlaying = true;

}

}

public void StopFootsteps()

{

footstepsSource.Stop();

isFootstepsPlaying = false;

}

// Сердцебиение с проверкой

public void PlayHeartbeat()

{

if (!isHeartbeatPlaying)

{

sfxSource.PlayOneShot(heartbeatSound);

isHeartbeatPlaying = true;

Invoke(nameof(ResetHeartbeat), heartbeatSound.length);

}

}

private void ResetHeartbeat() => isHeartbeatPlaying = false;

// Музыка

public void PlayMusic() => musicSource.Play();

public void StopMusic() => musicSource.Stop();

}

**Приложение 3.**

Презентация: https://docs.google.com/presentation/d/1Dsi6zEsD-MJ8-suc9etiwmVePHS6ia5D/edit?usp=sharing&ouid=107201872048884474270&rtpof=true&sd=true

Видео: https://drive.google.com/file/d/1Kn68WV8oKcwNsnbLAoGLmE8WYwAY49By/view?usp=sharing