**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Научный руководитель,  Научный сотрудник МЛ ИССА  факультета компьютерных наук, канд.техн.наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Максименкова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия», канд.техн.наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** | **RU.17701729.04.01-01 12 01-1** | | **КОМПЬЮТЕРНАЯ ИГРА: СКАЗАНИЯ О МОНСТРЕ**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 12 01-1-ЛУ** | | |
|  |  | |
| Исполнитель:  студент группы БПИ192  \_\_\_\_\_\_\_\_\_\_\_\_ / Глущенко З. С./  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |
|  | | |
|  | |  |

**2020**

|  |  |  |  |
| --- | --- | --- | --- |
| УТВЕРЖДЕНО  RU.17701729.04.01-01 12 01-1-ЛУ | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** | RU.17701729.04.01-01 12 01-1 | | **КОМПЬЮТЕРНАЯ ИГРА: СКАЗАНИЯ О МОНСТРЕ**  **Текст программы**  **RU.17701729.507140-01 12 01-1**    **Листов 46** | | | | |
|  |  | | | |
|  | | | |
|  | | | | |

**2020**

**Аннотация**

В данном документе приведет текст «Компьютерная игра: сказания о монстре». Программа разработана на языке C# 7.2 и Unity версии 2019.3.0f6.   
Среда разработки - Microsoft Visual Studio Community 2019.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов [1];

2) ГОСТ 19.102-77 Стадии разработки [2];

3) ГОСТ 19.103-77 Обозначения программ и программных документов [3];

4) ГОСТ 19.104-78 Основные надписи [4];

5) ГОСТ 19.105-78 Общие требования к программным документам [5];

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [6];

7) ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению [7].

Изменения к данному документу оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-78 [9].

Перед прочтением данного документа рекомендуется ознакомиться с терминологией, приведенной в Приложении 1.

СОДЕРЖАНИЕ

[1 Текст программы 4](#_Toc40472817)

[**1.1** **Код класса AudioManager.cs** 4](#_Toc40472818)

[**1.2** **Код класса CanvasManager.cs** 5](#_Toc40472819)

[**1.3** **Код DataManager.cs** 10](#_Toc40472820)

[**1.4** **Код класса EnemyManager.cs** 11](#_Toc40472821)

[**1.5** **Код класса GameManager.cs** 14](#_Toc40472822)

[**1.6** **Код класса InputManager.cs** 24](#_Toc40472823)

[**1.7** **Код класса SkillsManager.cs** 26](#_Toc40472824)

[**1.8** **Код класса AbstractCharacter.cs** 27](#_Toc40472825)

[**1.9** **Код класса AbstractEnemy.cs** 31](#_Toc40472826)

[**1.10** **Код класса RagdollScript.cs** 34](#_Toc40472827)

[**1.11** **Код класса SaveSlotButton.cs** 34](#_Toc40472828)

[**1.12** **Код класса ScrollingText.cs** 35](#_Toc40472829)

[**1.13** **Код класса Singleton.cs** 36](#_Toc40472830)

[**1.14** **Код класса SmoothFollowTarget.cs** 36](#_Toc40472831)

[**1.15** **Код класса Player.cs** 37](#_Toc40472832)

[**1.16** **Код класса Goblin.cs** 40](#_Toc40472833)

[**1.17** **Код класса GoblinBoss.cs** 41](#_Toc40472834)

[**1.18** **Код класса GoblinData.cs** 41](#_Toc40472835)

[**1.19** **Код класса Cutscene\_dock.cs** 42](#_Toc40472836)

[**1.20** **Код класса Cutscene\_dungeon.cs** 43](#_Toc40472837)

[**1.21** **Код интерфейса ICharacter.cs** 44](#_Toc40472838)

[ПРИЛОЖЕНИЕ 1 45](#_Toc40472839)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 46](#_Toc40472840)

# **1 Текст программы**

## **Код класса AudioManager.cs**

using UnityEngine;

/// <summary>

///

/// </summary>

public class AudioManager : Singleton<AudioManager>

{

#pragma warning disable 649

[SerializeField] AudioClip[] Clips; // Массив, содержащий музыку, для воспроизведения

#pragma warning restore 649

public AudioSource AudioSource => GetComponent<AudioSource>();

MyAudioClips CurrentClip;

private void FixedUpdate()

{

if (!AudioSource.isPlaying)

{

PlayNextClip();

}

}

/// <summary>

/// Проигрывает следующий клип

/// </summary>

public void PlayNextClip()

{

AudioSource.clip = Clips[(int)CurrentClip];

//AudioSource.PlayDelayed(1);

AudioSource.Play();

CurrentClip = CurrentClip == MyAudioClips.LastClip ? MyAudioClips.StartClip : CurrentClip + 1;

}

/// <summary>

/// Перечисление клипов

/// </summary>

public enum MyAudioClips

{

StartClip, // Winds of Winter

RedSwan,

HiroyukiSavano,

LastClip

}

}

## **Код класса CanvasManager.cs**

using UnityEngine;

using UnityEngine.UI;

/// <summary>

/// Класс, реализующий работу UI

/// </summary>

public class CanvasManager : Singleton<CanvasManager>

{

#pragma warning disable 649

[Header("StartScene")]

[SerializeField] GameObject \_startSceneUI; // Контейнер для объектов стартового меню

[Header("Settings")]

[SerializeField] Image \_settingsPanel; // Ссылка на меню настроек

[SerializeField] Image \_pauseImage; // Ссылка на меню паузы

[SerializeField] Slider \_musicVolume; // Ссылка на ползунок громкости в меню настроек

[SerializeField] TMPro.TextMeshProUGUI \_FPSLimit\_TMP;

[SerializeField] Slider \_FPSVolume;

[Header("Save&Load")]

[SerializeField] Image \_loadMenu; // Ссылка на меню загрузки сохранений

[SerializeField] Image \_saveMenu; // Ссылка на меню создания сохранений

public SaveSlotButton loadOne; // Ссылка на класс, реализующий работу с кнопками сохранения

public SaveSlotButton loadTwo; // Ссылка на класс, реализующий работу с кнопками сохранения

public SaveSlotButton loadThree; // Ссылка на класс, реализующий работу с кнопками сохранения

public SaveSlotButton saveOne; // Ссылка на класс, реализующий работу с кнопками сохранения

public SaveSlotButton saveTwo; // Ссылка на класс, реализующий работу с кнопками сохранения

public SaveSlotButton saveThree; // Ссылка на класс, реализующий работу с кнопками сохранения

[Header("HUD")]

[SerializeField] GameObject HUD; // Контейнер для объектов пользовательского интерфейса

[SerializeField] Image \_healthImage; // Полоска здоровья героя

[SerializeField] Image \_manaImage; // Полоска маны героя

[SerializeField] Image \_expImage; // Полоска опыта героя

[SerializeField] TMPro.TextMeshProUGUI \_lvlText; // Ссылка на текст, отвечающий за показ уровня героя

[SerializeField] TMPro.TextMeshProUGUI \_messageText; // Ссылка на текст, для инструкций при смене локаций

public Button Skill1, Skill2; // Ссылка на способности героя

[SerializeField] Image \_storyImage; // Ссылка на изображение с предысторией

[SerializeField] Image \_victoryPanel; // Ссылка на изображение, появляющиеся после победы над боссом

[Header("Players Death")]

[SerializeField] Image \_DeathPanel; // Ссылка на меню смерти

[HideInInspector] public bool needLoad; // отвечает за информирование о необходимости загрузки сохранения

[HideInInspector] public string path; // Путь до папки с сохранением

#pragma warning restore 649

/// <summary>

/// Обновляет значения здоровья, маны, опыта, уровня

/// </summary>

public void UpdateHUD()

{

Player player = Player.instance;

int lvl = player.Level;

\_healthImage.fillAmount = player.Health / DataManager.Stats.Player.Health[lvl];

\_manaImage.fillAmount = player.Manapool / DataManager.Stats.Player.Manapool[lvl];

\_expImage.fillAmount = player.Experience / DataManager.Stats.Player.Experience[lvl];

\_lvlText.text = $"Level: {player.Level}";

}

/// <summary>

/// Отобразить сообщение в левой части экрана

/// </summary>

/// <param name="message"></param>

public void UpdateMessage(string message)

{

\_messageText.text = message;

}

/// <summary>

/// Включает / выключает элементы интерфейса.

/// </summary>

/// <param name="showHUD"> Показывать HUD</param>

/// <param name="showStartMenu"> Показывать стартовое меню</param>

/// <param name="showPauseMenu">Показать меню паузы</param>

/// <param name="showDeathMenu">Показать меню смерти</param>

public void ActivateUI(bool showHUD = false, bool showStartMenu = false, bool showPauseMenu = false, bool showDeathMenu = false)

{

HUD.SetActive(showHUD);

\_startSceneUI.SetActive(showStartMenu);

\_pauseImage.gameObject.SetActive(showPauseMenu);

\_DeathPanel.gameObject.SetActive(showDeathMenu);

\_loadMenu.gameObject.SetActive(false);

\_saveMenu.gameObject.SetActive(false);

\_settingsPanel.gameObject.SetActive(false);

\_victoryPanel.gameObject.SetActive(false);

}

#region GameState / Handlers

/// <summary>

/// Обработчик загрузки сцены

/// </summary>

public void StartHandler()

{

GameManager.Instance.StartGame();

}

/// <summary>

/// Ставит игру на паузу или возобновляет и показывает/скрывает меню паузы.

/// </summary>

public void PauseHandler()

{

GameManager.Instance.TogglePause();

ActivateUI(HUD.activeSelf, showPauseMenu: !\_pauseImage.gameObject.activeSelf);

}

/// <summary>

/// Начинает игру с загруженным сохранением

/// </summary>

/// <param name="button">Нажатая кнопка сохранения</param>

public void MakeSave(SaveSlotButton button)

{

GameManager.Instance.SaveGame(button);

ActivateSaveMenu();

}

/// <summary>

/// Загружает сохранение по определенному пути.

/// </summary>

/// <param name="path">Путь сохранения</param>

public void LoadSave(string path)

{

this.path = path;

ActivateUI();

if (GameManager.Instance.CurrentState == GameState.Pregame)

{

needLoad = true;

StartHandler();

}

else

{

GameManager.Instance.CheckCurrentScene(path);

ActivateUI(showHUD: true, showPauseMenu: true);

}

}

/// <summary>

/// Открывает / Закрывает меню загрузки

/// </summary>

public void ActivateLoadMenu()

{

\_loadMenu.gameObject.SetActive(!\_loadMenu.gameObject.activeSelf);

}

/// <summary>

/// Открывает / Закрывает меню сохранений

/// </summary>

public void ActivateSaveMenu()

{

\_saveMenu.gameObject.SetActive(!\_saveMenu.gameObject.activeSelf);

}

/// <summary>

/// Открывает / Закрывает меню настроек

/// </summary>

public void ActivateSettings()

{

\_settingsPanel.gameObject.SetActive(!\_settingsPanel.gameObject.activeSelf);

}

/// <summary>

/// Обрабатывает действия в меню настроек

/// </summary>

/// <param name="action">Индекс изменение в меню настроек</param>

public void SettingsHandler(int action = 0)

{

var audioSource = AudioManager.Instance.AudioSource;

switch (action)

{

case 0: audioSource.volume = \_musicVolume.value; break;

case 1: AudioManager.Instance.PlayNextClip(); break;

case 2: GameManager.Instance.LimitFPS((int)\_FPSVolume.value); \_FPSLimit\_TMP.text = $"FPS Limit {(int)\_FPSVolume.value}"; break;

default:

break;

}

}

/// <summary>

/// Показывает / Скрывает меню с предысторией.

/// </summary>

public void ShowStory()

{

\_storyImage.gameObject.SetActive(!\_storyImage.gameObject.activeSelf);

}

/// <summary>

/// Обработчик перезапуска игры / выхода в главное меню

/// </summary>

public void RestartHandler()

{

ActivateUI(showStartMenu: true);

GameManager.Instance.RestartGame();

}

/// <summary>

/// Обработчик меню смерти

/// </summary>

public void DeathHandler()

{

\_DeathPanel.gameObject.SetActive(true);

GameManager.Instance.UpdateGameState(GameState.Death);

}

/// <summary>

/// Обработчик победы над главным злодеем

/// </summary>

public void Victory()

{

\_victoryPanel.gameObject.SetActive(true);

GameManager.Instance.TogglePause();

}

/// <summary>

/// Обработчик завершения игры

/// </summary>

public void QuitHandler()

{

GameManager.Instance.QuitGame();

}

#endregion

#region Spells

/// <summary>

/// Обработчик способности 1.

/// </summary>

public void SpellOne()

{

// Хорошо было бы сделать отображение нажатия...

SkillsManager.Instance.FirstSkill();

}

/// <summary>

/// Обработчик способности 2

/// </summary>

public void SpellTwo()

{

// Хорошо было бы сделать отображение нажатия...

SkillsManager.Instance.SecondSkill();

}

#endregion

/// <summary>

/// Обновляет доступные сохранения

/// </summary>

public void UpdateSaves()

{

loadOne.UpdateButton();

loadTwo.UpdateButton();

loadThree.UpdateButton();

saveOne.UpdateButton(true);

saveTwo.UpdateButton(true);

saveThree.UpdateButton(true);

}

/// <summary>

/// Обновляет отображение доступных навыков

/// </summary>

public void UpdateSkills()

{

Skill1.gameObject.SetActive(GameManager.Instance.avaibleFirstSkill);

Skill2.gameObject.SetActive(GameManager.Instance.avaibleSecondSkill);

}

}

## **Код DataManager.cs**

using System.Collections.Generic;

/// <summary>

/// Пространство имен, отвечающее за хранение игровых данных

/// </summary>

namespace DataManager

{

/// <summary>

/// Содержит классы с данными о персонажах

/// </summary>

namespace Stats

{

/// <summary>

/// Содержит характеристики Героя

/// </summary>

static class Player

{

public static readonly List<int> Health = new List<int>() { 0, 100, 125, 170, 200, 250, 300, 350, 400, 450, 500 };

public static readonly List<int> Manapool = new List<int>() { 0, 100, 125, 150, 175, 200, 225, 250, 275, 300, 350 };

public static readonly List<int> Experience = new List<int>() { 0, 20, 40, 80, 150, 200, 300, 400, 500, 600, 1000 };

public static readonly List<int> Damage = new List<int>() { 0, 20, 25, 30, 35, 40, 50, 60, 75, 85, 100 };

}

/// <summary>

/// Содержит характеристики Гоблина

/// </summary>

static class Goblin

{

// первый элемент 0 для удобного обращения

public static readonly List<int> Health = new List<int>() { 0, 50, 60, 70, 85, 100, 125, 150, 200, 300, 500 };

public static readonly List<int> Manapool = new List<int>() { 0, 100, 125, 150, 200, 250, 300, 350, 400, 450, 500 };

public static readonly List<int> Experience = new List<int>() { 0, 100, 125, 150, 200, 250, 0, 0, 0, 0, 0 };

public static readonly List<int> Damage = new List<int>() { 0, 10, 15, 20, 30, 40, 50, 60, 70, 80, 150 };

public static readonly float ExperienceMultiplier = 10;

}

/// <summary>

/// Заглушка

/// </summary>

static class Default

{

public static readonly List<int> Health = new List<int>() { 0, 0 };

public static readonly List<int> Manapool = new List<int>() { 0, 0 };

public static readonly List<int> Experience = new List<int>() { 0, 0 };

public static readonly List<int> Damage = new List<int>() { 0, 0 };

public static readonly float ExperienceCost = 0;

}

}

}

## **Код класса EnemyManager.cs**

using System.Collections;

using System.Collections.Generic;

using System.Runtime.Serialization;

using UnityEngine;

/// <summary>

/// Класс, реализующий работу с врагами.

/// </summary>

public class EnemyManager : Singleton<EnemyManager>

{

#region Object Pooling

#pragma warning disable 649

[SerializeField] GameObject enemySample; // Образец создоваемых врагов

[SerializeField] Transform[] \_spawnPositions; // Позиции для создания противников

[SerializeField] int poolStartLength = 20; // Минимальное количество противников в массиве. Для Object Pooling

[SerializeField] float \_spawnRadius = 50; // Минимальное расстояние между героем и созданным противником.

List<AbstractEnemy> \_abstractEnemiesPool = new List<AbstractEnemy>(); // Список противников приведенных к AbstractEnemy

public List<GameObject> ragdollPool = new List<GameObject>(); // Список ragdoll’ов, которые нужно удалить при загрузке сохранения.

[SerializeField, DataMember] List<GameObject> \_pool = new List<GameObject>(); // Список GameObject противников.

#pragma warning restore 649

/// <summary>

/// Свойство, находящее не используемого противника из пула, активирует и возвращает его.

/// </summary>

public GameObject GetEnemy

{

get

{

foreach (var item in \_pool)

{

if (!item.activeSelf)

{

item.SetActive(true);

return item;

}

}

// Если элемента не нашлось

var newEnemy = Instantiate(enemySample, transform);

\_pool.Add(newEnemy);

\_abstractEnemiesPool.Add(newEnemy.GetComponent<AbstractEnemy>());

return \_pool[\_pool.Count - 1];

}

}

/// <summary>

/// Свойство, возвращающее ссылку на пул GameObject’ов.

/// </summary>

public List<GameObject> GetPool => \_pool;

/// <summary>

/// Свойство, находящее используемых противников из пула, добавляет их в список и возвращает его.

/// </summary>

public List<AbstractEnemy> GetActiveAbstractEnemyPool

{

get

{

List<AbstractEnemy> list = new List<AbstractEnemy>();

foreach (var item in GetAbstractEnemyPool)

{

if (item.gameObject.activeSelf)

{

list.Add(item);

}

}

return list;

}

}

/// <summary>

/// Свойство, возвращающее список противников приведенных к AbstractEnemy.

/// </summary>

public List<AbstractEnemy> GetAbstractEnemyPool => \_abstractEnemiesPool;

#endregion

protected override void Awake()

{

base.Awake();

for (int i = \_pool.Count - 1; i < poolStartLength; i++)

{

var inst = Instantiate(enemySample, transform);

inst.SetActive(false);

\_pool.Add(inst);

}

foreach (var enemy in GetPool)

{

\_abstractEnemiesPool.Add(enemy.GetComponent<AbstractEnemy>());

}

}

/// <summary>

/// Удаляет все существующие ragdoll'ы со сцены

/// </summary>

public void DestroyAllRagdolls()

{

foreach (var item in ragdollPool)

{

if (item != null)

{

Destroy(item);

}

}

}

/// <summary>

/// Начинает новую "волну" атаки противников на героя

/// </summary>

/// <param name="count">Количество врагов</param>

/// <param name="enemyLevel">Уровень врагов</param>

public void StartNewWawe(int count, int enemyLevel = 4)

{

StartCoroutine(SpawnNewEnemies(count, enemyLevel));

}

/// <summary>

/// Корутина для создания новой "волны" атаки противников на героя

/// </summary>

/// <param name="count">Количество врагов</param>

/// <param name="enemyLevel">Уровень врагов</param>

/// <returns></returns>

IEnumerator SpawnNewEnemies(int count, int enemyLevel = 4)

{

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

{

yield return new WaitForSeconds(Random.Range(1.5f, 4));

var AbsEnemy = GetEnemy.GetComponent<AbstractEnemy>();

AbsEnemy.agent.SetDestination(Player.instance.transform.position);

AbsEnemy.Level = enemyLevel;

Vector3 position;

do

{

position = \_spawnPositions[Random.Range(0, \_spawnPositions.Length)].position;

} while (Vector3.Distance(Player.instance.transform.position, position) < \_spawnRadius);

AbsEnemy.transform.position = position;

}

}

}

## **Код класса GameManager.cs**

using System.Collections.Generic;

using System.IO;

using System.Runtime.Serialization.Json;

using UnityEngine;

using UnityEngine.AI;

using UnityEngine.SceneManagement;

/// <summary>

/// Игровые состояния

/// </summary>

public enum GameState

{

Pregame, // начальное состояние / главное меню

Running, // в процессе игры

Pause, // пауза

Death

}

/// <summary>

/// Содержит данные о сценах

/// </summary>

public enum Scenes

{

StartScene = 0,

MainGame = 1,

Dungeon = 2

}

/// <summary>

/// Отвечает за главную логику игры

/// </summary>

public class GameManager : Singleton<GameManager>

{

/// <summary>

/// Текущее состояние игры

/// </summary>

public GameState CurrentState;

/// <summary>

/// Текущая сцена

/// </summary>

public Scenes CurrentScene;

[HideInInspector] public bool isSaveOneAvailable; // Доступность 1 сохранения

[HideInInspector] public bool isSaveTwoAvailable; // Доступность 2 сохранения

[HideInInspector] public bool isSaveThreeAvailable; // Доступность 3 сохранения

[SerializeField] int \_deathCounter; // Счетчик убийств

public bool canTeleport; // Возможность перемещения на остров

public bool canEnterInDungeon; // Возможность перемещения в подземелье

public bool avaibleFirstSkill; // Доступность 1 способности

public bool avaibleSecondSkill; // Доступность 2 способности

/// <summary>

/// Свойство, обрабатывающее количество смертей

/// </summary>

public int DeathCounter

{

get => \_deathCounter;

set

{

\_deathCounter = value;

if (\_deathCounter == 10)

{

EnemyManager.Instance.StartNewWawe(10, 4);

}

else if (\_deathCounter == 20)

{

canTeleport = true;

}

else if (\_deathCounter == 30)

{

canEnterInDungeon = true;

}

}

}

/// <summary>

/// Встроенный метод. Используется для задавания уничтожаемых объектов, инициализации полей и проверки существующих сохранений.

/// </summary>

void Start()

{

DontDestroyOnLoad(gameObject);

DontDestroyOnLoad(CanvasManager.Instance.gameObject);

DontDestroyOnLoad(AudioManager.Instance.gameObject);

CurrentState = GameState.Pregame;

CurrentScene = Scenes.StartScene;

CheckSaves();

}

/// <summary>

/// Изменяет и обрабатывает изменение состояния игры

/// </summary>

/// <param name="state">Состояние, в которое переходит</param>

public void UpdateGameState(GameState state)

{

CurrentState = state;

switch (CurrentState)

{

case GameState.Pregame:

Time.timeScale = 0;

break;

case GameState.Running:

Time.timeScale = 1;

break;

case GameState.Pause:

Time.timeScale = 0;

break;

case GameState.Death:

Time.timeScale = 0;

break;

default:

Debug.LogError("[GameManager/UpdateGameState] default State");

break;

}

}

/// <summary>

/// Обрабатывает ограничение fps

/// </summary>

/// <param name="count">Максимальное количество FPS</param>

public void LimitFPS(int count)

{

Application.targetFrameRate = count;

}

/// <summary>

/// Сбрасывает данные GameManager

/// </summary>

void ResetGameManager()

{

\_deathCounter = 0;

canTeleport = false;

canEnterInDungeon = false;

avaibleFirstSkill = false;

avaibleSecondSkill = false;

CanvasManager.Instance.Skill1.gameObject.SetActive(false);

CanvasManager.Instance.Skill2.gameObject.SetActive(false);

}

/// <summary>

/// Возобновляет/приостанавливает игру

/// </summary>

public void TogglePause()

{

if (CurrentState == GameState.Running)

{

UpdateGameState(GameState.Pause);

}

else if (CurrentState == GameState.Pause)

{

UpdateGameState(GameState.Running);

}

}

/// <summary>

/// Начинает игру, загружая главную сцену

/// </summary>

public void StartGame()

{

if (CanvasManager.Instance.needLoad)

{

using (StreamReader sr = new StreamReader(CanvasManager.Instance.path + @"\CurrentLevel.json"))

{

CurrentScene = JsonUtility.FromJson<Scenes>(sr.ReadToEnd());

}

}

else

{

CurrentScene = Scenes.MainGame;

}

SceneManager.LoadSceneAsync((int)CurrentScene, LoadSceneMode.Additive).completed += delegate (AsyncOperation op)

{

if (CanvasManager.Instance.needLoad)

{

LoadGame(CanvasManager.Instance.path);

CanvasManager.Instance.needLoad = false;

}

else

{

CanvasManager.Instance.ShowStory();

ResetGameManager();

}

CanvasManager.Instance.ActivateUI(showHUD: true);

UpdateGameState(GameState.Running);

};

}

/// <summary>

/// Обрабатывает переход между игровыми сценами, с загрузкой необходимых данных

/// </summary>

/// <param name="newScene">Новая сцена</param>

/// <param name="changeLocation">Это загрузка или переход в новую локацию?</param>

public void ChangeScene(Scenes newScene, bool changeLocation = false)

{

TogglePause();

if (CurrentScene == Scenes.StartScene)

{

SceneManager.LoadSceneAsync((int)newScene, LoadSceneMode.Additive).completed += delegate (AsyncOperation obj2)

{

CurrentScene = newScene;

if (CanvasManager.Instance.needLoad)

{

LoadGame(CanvasManager.Instance.path);

CanvasManager.Instance.needLoad = false;

}

};

}

else

{

Player.PlayerData playerData = Player.instance.Clone() as Player.PlayerData;

SceneManager.UnloadSceneAsync((int)CurrentScene).completed += delegate (AsyncOperation obj)

{

SceneManager.LoadSceneAsync((int)newScene, LoadSceneMode.Additive).completed += delegate (AsyncOperation obj2)

{

CurrentScene = newScene;

if (changeLocation)

{

Player.instance.ApplyPlayerData(playerData);

}

else //(CanvasManager.Instance.needLoad) // тут желательно проверить, но вроде и так работает

{

LoadGame(CanvasManager.Instance.path);

CanvasManager.Instance.needLoad = false;

}

};

};

}

TogglePause();

}

/// <summary>

/// Перезапускает игру, выгружая текущую сцену и меняя режим интерфейса на стартовое меню.

/// </summary>

public void RestartGame()

{

SceneManager.UnloadSceneAsync((int)CurrentScene).completed += delegate (AsyncOperation asyncOperation)

{

CurrentScene = Scenes.StartScene;

UpdateGameState(GameState.Pregame);

};

}

#region Save & Load

/// <summary>

/// Проверяет существование сохранений и сообщает об их наличии

/// </summary>

public void CheckSaves()

{

CheckSave(@"Saves/Save1", CanvasManager.Instance.saveOne, CanvasManager.Instance.loadOne);

CheckSave(@"Saves/Save2", CanvasManager.Instance.saveTwo, CanvasManager.Instance.loadTwo);

CheckSave(@"Saves/Save3", CanvasManager.Instance.saveThree, CanvasManager.Instance.loadThree);

CanvasManager.Instance.UpdateSaves();

}

/// <summary>

/// Проверяет наличие сохранения

/// </summary>

/// <param name="path">Путь сохранения</param>

/// <param name="save">Скрипт на кнопке сохранения</param>

/// <param name="load">Скрипт на кнопке загрузки</param>

private void CheckSave(string path, SaveSlotButton save, SaveSlotButton load)

{

if (File.Exists($"{path}/CurrentLevel.json") && File.Exists($"{path}/EnemyGoblin.json") && File.Exists($"{path}/EnemyTransform.json") && File.Exists($"{path}/GameManager.json")

&& File.Exists($"{path}/PlayerData.json") && File.Exists($"{path}/PlayerTransform.json") && File.Exists($"{path}/SaveButton.json"))

{

using (StreamReader sr = new StreamReader($"{path}/SaveButton.json"))

{

JsonUtility.FromJsonOverwrite(sr.ReadToEnd(), save);

load.DateTMP = save.DateTMP;

load.isAvaible = true;

}

}

}

/// <summary>

/// Сохраняет игру

/// </summary>

public void SaveGame(SaveSlotButton button)

{

UpdateGameState(GameState.Pause);

button.DateTMP = System.DateTime.Now.ToString();

string path = button.path;

Directory.CreateDirectory(path);

string PlayerTransform = path + @"\PlayerTransform.json";

string PlayerPlayer = path + @"\PlayerData.json";

string EnemyTransform = path + @"\EnemyTransform.json";

string EnemyGoblin = path + @"\EnemyGoblin.json";

string ButtonPath = path + @"\SaveButton.json";

string GameManagerPath = path + @"\GameManager.json";

string CurrentLevelPath = path + @"\CurrentLevel.json";

Player player = Player.instance;

// Сохраняем GameManager

using (StreamWriter sw = new StreamWriter(GameManagerPath))

{

string data = JsonUtility.ToJson(this, true);

sw.Write(data);

}

// Сохраняем отдельно информацию о текущей сцене

using (StreamWriter sw = new StreamWriter(CurrentLevelPath))

{

string data = JsonUtility.ToJson(CurrentScene, true);

sw.Write(data);

}

// Сохраняем информацию кнопки

using (StreamWriter sw = new StreamWriter(ButtonPath))

{

string data = JsonUtility.ToJson(button, true);

sw.Write(data);

}

// Сохраняем Player героя

using (StreamWriter sw = new StreamWriter(PlayerPlayer))

{

string data = JsonUtility.ToJson(player, true);

sw.Write(data);

}

// Сохраняем transform.position героя

using (StreamWriter sw = new StreamWriter(PlayerTransform))

{

string data = JsonUtility.ToJson(player.transform.position, true);

sw.Write(data);

}

// сохраняем Transform врагов

using (FileStream fs = new FileStream(EnemyTransform, FileMode.Create))

{

List<string> enemies = new List<string>();

foreach (var item in EnemyManager.Instance.GetActiveAbstractEnemyPool)

{

enemies.Add(JsonUtility.ToJson(item.transform.position));

}

DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(List<string>));

serializer.WriteObject(fs, enemies);

}

// Сохраняем информацию (Goblin) об активных врагах

using (StreamWriter sw = new StreamWriter(EnemyGoblin))

{

foreach (var goblin in EnemyManager.Instance.GetActiveAbstractEnemyPool)

{

sw.WriteLine(JsonUtility.ToJson(new GoblinData(goblin as Goblin)));

}

}

CheckSaves();

}

/// <summary>

/// Загружает игру

/// </summary>

public void LoadGame(string path)

{

UpdateGameState(GameState.Pause);

string PlayerTransform = path + @"\PlayerTransform.json";

string PlayerPlayer = path + @"\PlayerData.json";

string EnemyTransform = path + @"\EnemyTransform.json";

string EnemyGoblin = path + @"\EnemyGoblin.json";

string GameManagerPath = path + @"\GameManager.json";

Player player = Player.instance;

// Загружаем GameManager

using (StreamReader sr = new StreamReader(GameManagerPath))

{

string data = sr.ReadToEnd();

JsonUtility.FromJsonOverwrite(data, this);

}

// Загружаем Player (статы) героя

using (StreamReader sr = new StreamReader(PlayerPlayer))

{

string data = sr.ReadToEnd();

JsonUtility.FromJsonOverwrite(data, player);

player.GetComponent<NavMeshAgent>().ResetPath();

}

// Загружаем Transform героя

using (StreamReader sr = new StreamReader(PlayerTransform))

{

string data = sr.ReadToEnd();

player.gameObject.SetActive(false);

player.transform.position = JsonUtility.FromJson<Vector3>(data);

player.gameObject.SetActive(true);

}

// Загружаем Transform и Golbin врагов

using (FileStream fset = new FileStream(EnemyTransform, FileMode.Open))

using (StreamReader swgob = new StreamReader(EnemyGoblin))

{

DataContractJsonSerializer Transformserializer = new DataContractJsonSerializer(typeof(List<string>));

List<string> enemiesTransform = Transformserializer.ReadObject(fset) as List<string>;

// Деактивируем врагов

foreach (var item in EnemyManager.Instance.GetActiveAbstractEnemyPool)

{

item.gameObject.SetActive(false);

}

for (int i = 0; i < enemiesTransform.Count; i++)

{

var enemy = EnemyManager.Instance.GetEnemy;

enemy.SetActive(false);

enemy.transform.position = JsonUtility.FromJson<Vector3>(enemiesTransform[i]);

var goblin = enemy.GetComponent<Goblin>();

GoblinData goblinData = JsonUtility.FromJson<GoblinData>(swgob.ReadLine());

goblin.ApplyData(goblinData);

goblin.gameObject.SetActive(true);

goblin.agent.ResetPath();

}

}

StartCoroutine(player.ResetRotine());

EnemyManager.Instance.DestroyAllRagdolls();

CanvasManager.Instance.UpdateHUD();

CanvasManager.Instance.UpdateSkills();

}

/// <summary>

/// Проверяет текущую сцену и загружаемую, и решает, что дальше делать.

/// </summary>

/// <param name="path">Путь до папки с сохранениями</param>

public void CheckCurrentScene(string path)

{

string CurrentLevelPath = path + @"\CurrentLevel.json";

// Загружаем информацию о возможной сцене

using (StreamReader sr = new StreamReader(CurrentLevelPath))

{

var privScene = CurrentScene;

var newScene = JsonUtility.FromJson<Scenes>(sr.ReadToEnd());

if (privScene != newScene)

{

ChangeScene(newScene);

}

else

{

LoadGame(path);

}

}

}

#endregion

/// <summary>

/// Завершает работу игры

/// </summary>

public void QuitGame()

{

#if UNITY\_EDITOR

UnityEditor.EditorApplication.isPlaying = false;

#else

Application.Quit();

#endif

}

}

## **Код класса InputManager.cs**

using UnityEditor;

using UnityEngine;

using UnityEngine.Events;

using UnityEngine.EventSystems;

/// <summary>

/// Класс, реализующий обработку пользовательского ввода

/// </summary>

public class InputManager : Singleton<InputManager>

{

#pragma warning disable 649

[SerializeField] LayerMask clickableLayer; // layermask используется для определения кликабельных слоев

[SerializeField] Texture2D pointer; // стандартный курсор

[SerializeField] Texture2D target; // курсор в виде прицела

//public Texture2D doorway; // курсор в виде двери

[SerializeField] Texture2D sword; // курсор в виде меча

[SerializeField] float \_scrollSens = 5; // Чувствительность колесика мышки.

public EventVector3 OnClickEnviroment; // Событие для перемещения игрока с помощью клика мышки. // Подписчик в эдиторе Hero.NavMeshAgent.destination

public EventGameObject onClickAttackable; // Событие для атаки врага.

#pragma warning restore 649

void Update()

{

// Если это начало игры или смерть, то данный функционал не требуется

if (GameManager.Instance.CurrentState == GameState.Pregame || GameManager.Instance.CurrentState == GameState.Death)

{

return;

}

// ---------------------

#region Нажатие кнопок мыши

// Стреляем лучами по сцене :)

RaycastHit hit;

if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit, 70, clickableLayer.value) && !EventSystem.current.IsPointerOverGameObject())

{

// Если объект содержит интерфейс IEnemy - значит это враг, которого можно атаковать

if (hit.collider.GetComponent(typeof(AbstractEnemy)) != null /\*|| hit.collider.CompareTag("Enemy")\*/)

{

Cursor.SetCursor(sword, new Vector2(16, 16), CursorMode.Auto);

if (Input.GetMouseButtonDown(0))

{

onClickAttackable?.Invoke(hit.collider.gameObject);

}

}

else

{

Cursor.SetCursor(target, new Vector2(16, 16), CursorMode.Auto);

if (Input.GetMouseButton(1))

{

OnClickEnviroment?.Invoke(hit.point);

}

}

}

else

{

Cursor.SetCursor(pointer, Vector2.zero, CursorMode.Auto);

}

#endregion

// ---------------------

// ---------------------

#region Вращение колесика мышки

// Вращение колесика мышки - Приближение/Отдаление камеры

if (Input.GetAxis("Mouse ScrollWheel") < 0 && SmoothFollowTarget.Instance.Offset.magnitude < 30)

{

SmoothFollowTarget.Instance.Offset += SmoothFollowTarget.Instance.Offset \* Time.deltaTime \* \_scrollSens;

}

else if (Input.GetAxis("Mouse ScrollWheel") > 0 && SmoothFollowTarget.Instance.Offset.magnitude > 10)

{

SmoothFollowTarget.Instance.Offset -= SmoothFollowTarget.Instance.Offset \* Time.deltaTime \* \_scrollSens;

}

#endregion

// ---------------------

// ---------------------

#region Нажатия клавиш

if (Input.GetKeyDown(KeyCode.Escape))

{

CanvasManager.Instance.PauseHandler();

}

if (Input.GetKeyDown(KeyCode.Alpha1))

{

SkillsManager.Instance.FirstSkill();

}

if (Input.GetKeyDown(KeyCode.Alpha2))

{

SkillsManager.Instance.SecondSkill();

}

#endregion

// ---------------------

}

}

/// <summary>

/// Класс, (наследник UnityEvent<Vector3>) для создание событий с передаваемым типом Vector3.

/// </summary>

[System.Serializable]

public class EventVector3 : UnityEvent<Vector3> { }

/// <summary>

/// Класс, (наследник UnityEvent<GameObject> для создание событий с передаваемым типом GameObject.

/// </summary>

[System.Serializable]

public class EventGameObject : UnityEvent<GameObject> { }

## **Код класса SkillsManager.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

/// <summary>

///

/// </summary>

public class SkillsManager : Singleton<SkillsManager>

{

[SerializeField, Range(50, 100)] float hpHealCost = 50; // Стоимость восполнения жизни второй способностью

[SerializeField, Range(10, 20)] float agrCost = 20; // Стоимость использования первой способности

[SerializeField, Range(20, 50)] float agrRange = 20; // Радиус провокации противников

/// <summary>

/// Первая способность – провокация противников

/// </summary>

public void FirstSkill()

{

if (GameManager.Instance.CurrentState == GameState.Running && GameManager.Instance.avaibleFirstSkill && Player.instance.Manapool > hpHealCost)

{

Player.instance.Manapool -= agrCost;

foreach (var enemy in EnemyManager.Instance.GetAbstractEnemyPool)

{

if (enemy.gameObject.activeSelf && Vector3.Distance(enemy.transform.position, Player.instance.transform.position) < agrRange)

{

enemy.agent.SetDestination(Player.instance.transform.position);

}

}

}

}

/// <summary>

/// Вторая способность - восстановление хм за счет маны

/// </summary>

public void SecondSkill()

{

if (GameManager.Instance.CurrentState == GameState.Running && GameManager.Instance.avaibleSecondSkill && Player.instance.Manapool > agrCost)

{

Player.instance.Health += hpHealCost;

Player.instance.Manapool -= hpHealCost;

CanvasManager.Instance.UpdateHUD();

}

}

}

## **Код класса AbstractCharacter.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

/// <summary>

/// Абстрактный класс отвечающий за базовый функционал существа.

/// </summary>

[RequireComponent(typeof(NavMeshAgent)), RequireComponent(typeof(Animator))/\*, RequireComponent(typeof(Rigidbody))\*/]

public abstract class AbstractCharacter : MonoBehaviour, ICharacter

{

/// <summary>

/// Метод (Конструктор) для создания абстрактного персонажа.

/// </summary>

/// <param name="health">Список значений количества здоровья, соответствующий уровню героя</param>

/// <param name="mana">Список значений количества маны, соответствующий уровню героя</param>

/// <param name="experience">Список значений количества опыта, соответствующий уровню героя</param>

/// <param name="damages">>Список значений возможного урона, соответствующий уровню героя </param>

/// <param name="name">Имя персонажа</param>

/// <param name="lvl">Уровень персонажа</param>

protected void InitializeProperties(List<int> health, List<int> mana, List<int> experience, List<int> damages, string name, int lvl = 1)

{

\_listHp = health;

\_listMp = mana;

\_listExp = experience;

\_listDmg = damages;

//Level = \_lvl == 0 ? lvl : \_lvl;

Health = health[Level];

Manapool = mana[Level];

Name = name;

animator = GetComponent<Animator>();

//rbody = GetComponent<Rigidbody>();

\_scrollingTextContainer = GameObject.Find("TextContainer").transform;

// инициализация NavMeshAgent

agent = GetComponent<NavMeshAgent>();

agent.acceleration = 20;

agent.angularSpeed = 800;

agent.autoBraking = false;

agent.stoppingDistance = 1f;

}

#region Fields & properties

#pragma warning disable 649

// Компоненты

public NavMeshAgent agent; /\*{ get; protected set; }\*/ // Перемещение персонажа на сцене

protected Animator animator; // Анимирует действия персонажа

//protected Rigidbody rbody;

[SerializeField] ScrollingText \_scrollingText; // Ссылка на клонируемый объект

[SerializeField] Color \_scrollingTextColor; // Цвет вслывающего текста

Transform \_scrollingTextContainer; // Контейнер, куда будут помещаться созданные объекты.

// значения, который можно задать в инспекторе. DebugOnly! Потом удалить атрибуты

[Header("Values"), Space]

[SerializeField] float \_health; // Хранит текущее количество здоровья персонажа

[SerializeField] float \_mana; // Хранит текущее количество маны персонажа

[SerializeField] int \_lvl; // Хранит текущий уровень

[SerializeField] float \_exp; // Хранит текущее количество опыта

#pragma warning restore 649

// списки со значениями

protected List<int> \_listHp; // Список возможных значений здоровья

protected List<int> \_listMp; // Список возможных значений маны

protected List<int> \_listExp; // Список возможных значений опыта

protected List<int> \_listDmg; // Список возможных значений урона

/// <summary>

/// Имя персонажа

/// </summary>

public string Name { get; set; }

/// <summary>

/// Свойство, отвечающее за здоровье

/// </summary>

public virtual float Health

{

get => \_health;

set

{

\_health = Mathf.Clamp(value, -1, \_listHp[Level]);

if (\_health <= 0) Die();

}

}

/// <summary>

/// Свойство, отвечающее за ману.

/// </summary>

public float Manapool

{

get => \_mana;

set => \_mana = Mathf.Clamp(value, 0, \_listMp[Level]);

}

/// <summary>

/// Свойство, отвечающее за уровень.

/// </summary>

public virtual int Level

{

get => \_lvl;

set

{

\_lvl = Mathf.Clamp(value, 1, \_listExp.Count - 1);

Health = \_listHp[\_lvl];

Manapool = \_listMp[\_lvl];

}

}

/// <summary>

/// Свойство, отвечающее за опыт.

/// </summary>

public float Experience

{

get => \_exp;

set

{

while (value >= \_listExp[Level])

{

value -= \_listExp[Level];

Level++;

}

\_exp = value;

}

}

/// <summary>

/// Свойство, отвечающее за урон.

/// </summary>

public float Damage => \_listDmg[Level];

#endregion

/// <summary>

/// Атакует выбранную цель, нанося урон в итервале [Damage/2, Damage] урона.

/// </summary>

/// <param name="character">Цель атаки</param>

public virtual void Attack(ICharacter character)

{

character?.GetDamage(Random.Range(Damage / 2, Damage));

animator.SetTrigger("Attack");

}

/// <summary>

/// Обрабатывает получение урона в размере damage.

/// </summary>

/// <param name="damage">Получаемый урон</param>

public virtual void GetDamage(float damage)

{

Health -= damage;

var text = Instantiate(\_scrollingText, transform.position, Quaternion.identity, \_scrollingTextContainer);

text.SetTextAndColor(((int)damage).ToString(), \_scrollingTextColor);

}

/// <summary>

/// Восполняет здоровье и ману персонажа через определенный интервал.

/// </summary>

/// <param name="healRecovery">Скорость восстановления здоровья</param>

/// <param name="manaRecovery">Скорость восстановления маны</param>

/// <param name="delay">Задержка между лечениями</param>

/// <returns></returns>

public virtual IEnumerator Recovery(float healRecovery = 5, float manaRecovery = 5, float delay = 1)

{

while (true)

{

Health += healRecovery;

Manapool += manaRecovery;

yield return new WaitForSeconds(delay);

}

}

/// <summary>

/// Обрабатывает смерть персонажа

/// </summary>

public virtual void Die()

{

gameObject.SetActive(false);

}

}

## **Код класса AbstractEnemy.cs**

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

/// <summary>

/// Абстрактный класс, дополняющий AbstractCharacter, реализуя базовый функционал врага

/// </summary>

[RequireComponent(typeof(NavMeshAgent))]

public abstract class AbstractEnemy : AbstractCharacter

{

protected Player player; // Ссылка на главного героя

protected float \_expMultiplier; // Множитель получаемого опыта за убийство этого противника

[SerializeField] protected float \_agrRadius = 10; // Радиус, с которого враг замечает героя

[SerializeField] protected float \_attackRadius; //= 2; // Радиус, с которого враг начинает атаковать героя

[SerializeField] protected float \_attackRate = 1; // Время необходимое для совершения одной атаки

//[SerializeField] float \_multiplier = 3;

[Header("GameObject")]

[SerializeField] protected GameObject DieRagdoll; // Ragdoll’ для анимации смерти.

[SerializeField] protected GameObject healthBar; // Полоска здоровья

float timer; // Отсчитывает время прошедшее с предыдущей атаки

public override float Health

{

get => base.Health;

set

{

base.Health = value;

healthBar.transform.localScale = new Vector3(Health / \_listHp[Level], transform.localScale.y, transform.localScale.z) / 4;

}

}

/// <summary>

/// Метод (Конструктор) для создания абстрактного врага.

/// </summary>

/// <param name="health">Список значений количества здоровья, соответствующий уровню героя</param>

/// <param name="mana">Список значений количества маны, соответствующий уровню героя</param>

/// <param name="experience">Список значений количества опыта, соответствующий уровню героя</param>

/// <param name="damages">>Список значений возможного урона, соответствующий уровню героя </param>

/// <param name="name">Имя персонажа</param>

/// <param name="lvl">Уровень персонажа</param>

/// <param name="expMultiplier">Множитель получаемого количества опыта за убийство</param>

protected void InitializeProperties(List<int> health, List<int> mana, List<int> experience, List<int> damages, string name, int lvl = 1, float expMultiplier = 1)

{

base.InitializeProperties(health, mana, experience, damages, name, lvl);

\_expMultiplier = expMultiplier;

agent.stoppingDistance += .5f;

\_attackRadius = agent.stoppingDistance + .5f;

healthBar = GetComponentInChildren<SpriteRenderer>().gameObject;

}

protected virtual void Start()

{

player = Player.instance;

}

protected virtual void FixedUpdate()

{

animator.SetFloat("Speed", agent.velocity.magnitude);

healthBar.transform.LookAt(Camera.main.transform);

Move();

}

/// <summary>

/// Обработчик события

/// </summary>

public virtual void Hit()

{

transform.LookAt(Player.instance.transform);

healthBar.transform.LookAt(Camera.main.transform);

}

/// <summary>

/// Реализует перемещение врага к игроку.

/// </summary>

public virtual void Move()

{

if (Vector3.Distance(player.transform.position, transform.position) < \_agrRadius)

{

agent.SetDestination(player.transform.position);

if (Vector3.Distance(player.transform.position, transform.position) < \_attackRadius && timer > \_attackRate)

{

timer = 0;

Attack(player);

}

timer += Time.deltaTime;

}

}

/// <summary>

/// Обработчик смерти врага

/// </summary>

public override void Die()

{

player.Experience += \_expMultiplier \* Level;

CanvasManager.Instance.UpdateHUD();

GameManager.Instance.DeathCounter++;

// Todo: Настроить

var inst = Instantiate(DieRagdoll, transform.position, transform.rotation);

EnemyManager.Instance.ragdollPool.Add(inst);

var ragdoll = inst.GetComponent<RagdollScript>();

var force = transform.position - player.transform.position;

force.Normalize();

force.y += .5f;

var forceMultiplier = Mathf.Clamp(Mathf.Log(player.Level \* 1f / Level), .2f, 2);

ragdoll.StartDeath(force \* forceMultiplier, 3);

base.Die();

}

/// <summary>

/// Отображает в editor'е радиус атаки и агра противников

/// </summary>

private void OnDrawGizmos()

{

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(transform.position, \_agrRadius);

Gizmos.DrawWireSphere(transform.position, \_attackRadius);

}

/// <summary>

/// Обрабатывает получение урона.

/// </summary>

/// <param name="damage">количество урона</param>

public override void GetDamage(float damage)

{

base.GetDamage(damage);

}

}

## **Код класса RagdollScript.cs**

using UnityEngine;

/// <summary>

/// Класс, анимирующий смерть противника.

/// </summary>

public class RagdollScript : MonoBehaviour

{

#pragma warning disable 649

[SerializeField] Rigidbody rbody; // Ссылка на компонент Rigidbody

[SerializeField] float forceToAdd = 5000; // Множитель для силы отталкивания

#pragma warning restore 649

/// <summary>

/// Анимирует смерть врага.

/// </summary>

/// <param name="force"></param>

/// <param name="liveTime"></param>

public void StartDeath(Vector3 force, float liveTime)

{

rbody.AddForce(force \* forceToAdd);

Destroy(gameObject, liveTime);

}

}

## **Код класса SaveSlotButton.cs**

using UnityEngine;

/// <summary>

/// Класс, реализующий работу кнопки сохранений

/// </summary>

public class SaveSlotButton : MonoBehaviour

{

#pragma warning disable 649

public string path; // Путь до папки с сохранением

[SerializeField] TMPro.TextMeshProUGUI \_dateTMP; // Ссылка компонент с текстом

[SerializeField] string \_text; // Хранит значение для сериализации

public bool isAvaible; // Доступность сохранения

#pragma warning restore 649

/// <summary>

/// Свойство для работы с отображаемым текстом на кнопке сохранения

/// </summary>

public string DateTMP

{

get => \_text;

set

{

\_text = value;

\_dateTMP.text = value;

}

}

/// <summary>

/// Обновляет содержимое кнопки и ее отображение

/// </summary>

/// <param name="forceActivate">Принудительная активация</param>

public void UpdateButton(bool forceActivate = false)

{

gameObject.SetActive(isAvaible || forceActivate);

\_dateTMP.text = \_text;

}

}

## **Код класса ScrollingText.cs**

using UnityEngine;

/// <summary>

/// Класс, реализующий визуализацию информации об уроне.

/// </summary>

[RequireComponent(typeof(TextMesh))]

public class ScrollingText : MonoBehaviour

{

TextMesh \_textMesh; // Ссылка на главного героя

[SerializeField] float \_duration = 3; // Получаемый опыт за убийство этого врага

[SerializeField] float \_speed = 3; // Радиус, с которого враг замечает героя

float \_timer; // Время жизни объекта

void Awake()

{

\_textMesh = GetComponent<TextMesh>();

\_timer = 0;

transform.LookAt(Camera.main.transform.position);

}

/// <summary>

/// Встроенный метод, который отрабатывает, каждые Time.fixedDeltaTime.

/// </summary>

void FixedUpdate()

{

if (\_timer > \_duration)

{

Destroy(gameObject);

}

\_timer += Time.deltaTime;

transform.Translate(Vector3.up \* Time.deltaTime \* \_speed);

}

/// <summary>

/// Устанавливает текст и цвет

/// </summary>

/// <param name="damage">Количество урона</param>

/// <param name="color">Цвет</param>

public void SetTextAndColor(string damage, Color color)

{

\_textMesh.text = damage;

\_textMesh.color = color;

}

}

## **Код класса Singleton.cs**

using UnityEngine;

/// <summary>

/// Класс, реализующий паттерн Singleton.

/// </summary>

/// <typeparam name="T">Тип</typeparam>

public class Singleton<T> : MonoBehaviour where T : Singleton<T>

{

public static T Instance { get; private set; } // Свойство, хранящее ссылку на экземпляр класса.

protected virtual void Awake()

{

if (Instance != null)

{

Debug.LogError($"[Singleton] Trying to instantiate a second instance of singleton class");

}

Instance = (T)this;

}

}

## **Код класса SmoothFollowTarget.cs**

using UnityEngine;

/// <summary>

/// Отвечает за плавное движение камеры за игроком

/// </summary>

public class SmoothFollowTarget : Singleton<SmoothFollowTarget>

{

public GameObject target; // Ссылка на главного героя

public Vector3 Offset; // Расстояние до цели

/// <summary>

/// Встроенный метод, который вызывается в конце каждого кадра.

/// Используется для перемещения камеры за героем.

/// </summary>

public void LateUpdate()

{

if (target == null)

{

target = GameObject.FindGameObjectWithTag("Player");

if (target != null) // Если сделать else к if(target == null) поломается отдаление камер

{

if (GameManager.Instance.CurrentScene == Scenes.MainGame)

{

transform.position = new Vector3(1.5f, 12, -7) + target.transform.position;

Offset = transform.position - target.transform.position;

}

if (GameManager.Instance.CurrentScene == Scenes.Dungeon)

{

transform.position = new Vector3(6, 12, 6) + target.transform.position;

Offset = transform.position - target.transform.position;

}

}

return;

}

transform.position = Vector3.Lerp(transform.position, target.transform.position + Offset, Time.deltaTime \* 5);

}

}

## **Код класса Player.cs**

using System.Collections;

using UnityEngine;

/// <summary>

/// Класс, реализующий игрока

/// </summary>

[System.Serializable]

public class Player : AbstractCharacter, System.ICloneable

{

[HideInInspector] public static Player instance; // Ссылка на героя. Singleton

bool isAttacking; // Состояние атаки героя. Начата ли атака

GameObject attackTarget; //Ссылка на противника

/// <summary>

/// Свойство, отвечающее за работу с уровнем героя

/// </summary>

public override int Level

{

get => base.Level;

set

{

base.Level = value;

if (value == 3)

{

GameManager.Instance.avaibleFirstSkill = true;

CanvasManager.Instance.Skill1.gameObject.SetActive(true);

}

if (value == 6)

{

GameManager.Instance.avaibleSecondSkill = true;

CanvasManager.Instance.Skill2.gameObject.SetActive(true);

}

}

}

private void Awake()

{

InitializeProperties(DataManager.Stats.Player.Health, DataManager.Stats.Player.Manapool, DataManager.Stats.Player.Experience, DataManager.Stats.Player.Damage, "Hero");

instance = this;

StartCoroutine(Recovery(5, 5));

}

private void FixedUpdate()

{

animator.SetFloat("Speed", agent.velocity.magnitude);

}

/// <summary>

/// Класс, хранящий данные необходимые для сохранения.

/// </summary>

public class PlayerData

{

public float Health; // Хранение данных о здоровье героя

public float Manapool; // Хранение данных о мане героя

public int Level; // Хранение данных о уровне героя

public float Experience;// Хранение данных о полученном опыте героя

}

/// <summary>

/// Принимает и обрабатывает сохраненные данные

/// </summary>

/// <param name="data">Сохраненные данные</param>

public void ApplyPlayerData(PlayerData data)

{

this.Level = data.Level;

this.Health = data.Health;

this.Manapool = data.Manapool;

this.Experience = data.Experience;

}

/// <summary>

/// Корутина для атаки

/// Если Расстояние больше радиуса атаки, то герой сначала подбежит. И лишь потом ударит.

/// </summary>

/// <returns></returns>

IEnumerator AttackTarget()

{

agent.isStopped = false;

while (Vector3.Distance(transform.position, attackTarget.transform.position) > 2)

{

agent.stoppingDistance = 2;

agent.destination = attackTarget.transform.position;

yield return null;

}

// Todo: Донастроить

agent.isStopped = true;

transform.LookAt(attackTarget.transform);

Attack(attackTarget.GetComponent<ICharacter>());

}

/// <summary>

/// Обработчик события завершения атаки

/// </summary>

public void Hit()

{

agent.isStopped = false;

StopCoroutine(AttackTarget());

isAttacking = false;

}

/// <summary>

/// Обработчик события атаки

/// </summary>

/// <param name="target">Цель атаки</param>

public void AttackHandler(GameObject target)

{

if (!isAttacking)

{

isAttacking = true;

attackTarget = target;

StartCoroutine(AttackTarget());

}

}

/// <summary>

/// Обрабатывает получение урона в размере damage.

/// </summary>

/// <param name="damage">Получаемый урон</param>

public override void GetDamage(float damage)

{

base.GetDamage(damage);

CanvasManager.Instance.UpdateHUD();

}

/// <summary>

/// Восполняет здоровье и ману персонажа.

/// Переопределение: Обновляет HUD

/// </summary>

/// <param name="healRecovery">Множитель скорости восстановления здоровья</param>

/// <param name="manaRecovery">Множитель скорости восстановления маны</param>

/// <param name="delay">Задержка между лечениями</param>

/// <returns></returns>

public override IEnumerator Recovery(float healRecovery = 1, float manaRecovery = 1, float delay = 0.1f)

{

yield return null;

while (true)

{

Health += healRecovery \* \_listHp[Level] / 1000;

Manapool += manaRecovery \* \_listMp[Level] / 1000;

CanvasManager.Instance.UpdateHUD();

yield return new WaitForSeconds(delay);

}

}

/// <summary>

/// Обработчик смерти героя

/// </summary>

public override void Die(/\*float delay = 0\*/)

{

CanvasManager.Instance.DeathHandler();

}

/// <summary>

/// Клонирует данные героя

/// </summary>

/// <returns></returns>

public object Clone()

{

return new PlayerData() { Health = this.Health, Manapool = this.Manapool, Level = this.Level, Experience = this.Experience }; ;

}

/// <summary>

/// Завершает все корутины и после возобновляет Recovery

/// </summary>

/// <returns></returns>

public IEnumerator ResetRotine()

{

Hit();

StopAllCoroutines();

yield return null;

StartCoroutine(Recovery(5, 5));

}

}

## **Код класса Goblin.cs**

/// <summary>

/// Класс, реализующий противника гоблина.

/// </summary>

public class Goblin : AbstractEnemy

{

private void Awake()

{

InitializeProperties(DataManager.Stats.Goblin.Health, DataManager.Stats.Goblin.Manapool, DataManager.Stats.Goblin.Experience,

DataManager.Stats.Goblin.Damage, "Goblin goblinovich", 1, DataManager.Stats.Goblin.ExperienceMultiplier);

}

/// <summary>

/// Обрабатывает сохраненные данные

/// </summary>

/// <param name="goblinData">Сохраненные данные</param>

public void ApplyData(GoblinData goblinData)

{

Level = goblinData.Level;

Health = goblinData.Health;

Manapool = goblinData.Manapool;

}

}

## **Код класса GoblinBoss.cs**

/// <summary>

/// Класс, реализующий противника короля гоблинов.

/// </summary>

public class GoblinBoss : AbstractEnemy

{

private void Awake()

{

InitializeProperties(DataManager.Stats.Goblin.Health, DataManager.Stats.Goblin.Manapool, DataManager.Stats.Goblin.Experience,

DataManager.Stats.Goblin.Damage, "Goblin boss", 10, DataManager.Stats.Goblin.ExperienceMultiplier);

}

/// <summary>

/// Обработчик смерти босса - победа в игре.

/// </summary>

public override void Die()

{

CanvasManager.Instance.Victory();

}

}

## **Код класса GoblinData.cs**

/// <summary>

/// Класс, содержащий данные о гоблинах для сохранения.

/// </summary>

public class GoblinData

{

public GoblinData(Goblin goblin)

{

Health = goblin.Health;

Manapool = goblin.Manapool;

Level = goblin.Level;

}

public float Health; // Хранение данных о здоровье героя

public float Manapool; // Хранение данных о мане героя

public int Level; // Хранение данных об уровне героя

}

## **Код класса Cutscene\_dock.cs**

using UnityEngine;

/// <summary>

/// Класс, отвечающий за переход между локациями. Переход с причала на остров.

/// </summary>

public class Cutscene\_dock : MonoBehaviour

{

/// <summary>

/// Встроенный метод. Используется для обработки нахождения в триггере

/// </summary>

/// <param name="other">Вошедший коллайдер</param>

private void OnTriggerStay(Collider other)

{

if (other.CompareTag("Player"))

{

if (GameManager.Instance.canTeleport)

{

CanvasManager.Instance.UpdateMessage("Нажмите E для перехода в новую локацию");

if (Input.GetKey(KeyCode.E))

{

var timeScale = Time.timeScale;

Time.timeScale = 0;

Player.instance.agent.isStopped = true;

Player.instance.gameObject.SetActive(false);

Player.instance.transform.position = new Vector3(-68, 0.1f, 136);

Player.instance.gameObject.SetActive(true);

Player.instance.agent.isStopped = false;

Time.timeScale = timeScale;

StartCoroutine(Player.instance.ResetRotine());

OnTriggerExit(other);

}

}

else

{

CanvasManager.Instance.UpdateMessage("Для перехода в другую локацию необходимо убить больше гоблинов!");

}

}

}

/// <summary>

/// Встроенный метод. Используется для обработки выхода из триггера

/// </summary>

/// <param name="other">Вышедший коллайдер</param>

private void OnTriggerExit(Collider other)

{

CanvasManager.Instance.UpdateMessage(string.Empty);

}

}

## **Код класса Cutscene\_dungeon.cs**

using UnityEngine;

/// <summary>

/// Класс, отвечающий за переход между локациями. Переход с причала на остров.

/// </summary>

public class Cutscene\_dock : MonoBehaviour

{

/// <summary>

/// Встроенный метод. Используется для обработки нахождения в триггере

/// </summary>

/// <param name="other">Вошедший коллайдер</param>

private void OnTriggerStay(Collider other)

{

if (other.CompareTag("Player"))

{

if (GameManager.Instance.canTeleport)

{

CanvasManager.Instance.UpdateMessage("Нажмите E для перехода в новую локацию");

if (Input.GetKey(KeyCode.E))

{

var timeScale = Time.timeScale;

Time.timeScale = 0;

Player.instance.agent.isStopped = true;

Player.instance.gameObject.SetActive(false);

Player.instance.transform.position = new Vector3(-68, 0.1f, 136);

Player.instance.gameObject.SetActive(true);

Player.instance.agent.isStopped = false;

Time.timeScale = timeScale;

StartCoroutine(Player.instance.ResetRotine());

OnTriggerExit(other);

}

}

else

{

CanvasManager.Instance.UpdateMessage("Для перехода в другую локацию необходимо убить больше гоблинов!");

}

}

}

/// <summary>

/// Встроенный метод. Используется для обработки выхода из триггера

/// </summary>

/// <param name="other">Вышедший коллайдер</param>

private void OnTriggerExit(Collider other)

{

CanvasManager.Instance.UpdateMessage(string.Empty);

}

}

## **Код интерфейса ICharacter.cs**

using System.Collections;

/// <summary>

/// Интерфейс, отвечающий за основную механику персонажей.

/// </summary>

public interface ICharacter

{

/// <summary>

/// Получиет урон в размере damage.

/// </summary>

/// <param name="damage">Получаемый урон</param>

void GetDamage(float damage);

/// <summary>

/// Атакует выбранную цель.

/// </summary>

/// <param name="character">Цель атаки</param>

void Attack(ICharacter character);

/// <summary>

/// Обрабатывает смерть персонажа.

/// </summary>

void Die();

/// <summary>

/// Восполняет здоровье и ману персонажа.

/// </summary>

/// <param name="healRecovery">Скорость восстановления здоровья</param>

/// <param name="manaRecovery">Скорость восстановления маны</param>

/// <param name="delay">Задержка между лечениями</param>

/// <returns></returns>

IEnumerator Recovery(float healRecovery = 5, float manaRecovery = 5, float delay = 1);

}

# **ПРИЛОЖЕНИЕ 1**

**ТЕРМИНОЛОГИЯ**

|  |
| --- |
|  |

Таблица

|  |  |
| --- | --- |
| **Термин** | **Определение** |
| **HUD** | Head-Up Display. Часть визуального интерфейса игрока, отображающаяся на фоне виртуального игрового пространства. Содержит индикаторы здоровья, маны, полученного опыта, уровня и тд. |
| **NPC** | Non-Player Character. Персонаж, который находится не под контролем пользователя. |
| **UI** | User interface. Пользовательский интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы. |

# **ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированх |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |