**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Образовательной программы бакалавриата «Программная инженерия»

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Научный руководитель, доцент департамента  программной инженерии  факультета компьютерных наук, канд. техн. наук    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.Л. Макаров  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. | УТВЕРЖДАЮ  Академический руководитель  ОП «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. |

|  |  |
| --- | --- |
| *Подп. и дата* |  |
| *Инв. № дубл.* |  |
| *Взам. инв. №* |  |
| *Подп. и дата* |  |
| *Инв. № подл* |  |

**2D-игра “Adventure”  
 на Unity**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.****04.01-01 12 01-1-ЛУ**

Исполнитель:  
студент группы БПИ198  
\_\_\_\_\_\_\_\_\_\_ / Здор А. М. /  
«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Москва 2020**

УТВЕРЖДЕН

RU.17701729.04.01-01 12 01-1-ЛУ

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

**2D-игра “Adventure”  
 на Unity**

**Текст программы**

**RU.17701729.04.01-01 12 01-1**

**Листов 48**

**Москва 2020**

**Содержание**

[1. ТЕКСТ ПРОГРАММЫ 4](#_Toc40645586)

[1.1 Класс BlockInfo.cs 4](#_Toc40645587)

[1.2 Класс Camera.cs 4](#_Toc40645588)

[1.3 Класс Chest.cs 5](#_Toc40645589)

[1.4 Класс Coin.cs 6](#_Toc40645590)

[1.5 Класс Colya.cs 7](#_Toc40645591)

[1.6 Класс Deth.cs 9](#_Toc40645592)

[1.7 Класс Enemy.cs 10](#_Toc40645593)

[1.8 Класс Falling.cs 13](#_Toc40645594)

[1.9 Класс Floor.cs 14](#_Toc40645595)

[1.10 Класс GameoverMenu.cs 14](#_Toc40645596)

[1.11 Класс Gun.cs 15](#_Toc40645597)

[1.12 Класс GunAmmo.cs 16](#_Toc40645598)

[1.13 Класс HealthBonus.cs 17](#_Toc40645599)

[1.14 Класс HealthImproving.cs 18](#_Toc40645600)

[1.15 Класс HealthSystem.cs 19](#_Toc40645601)

[1.16 Класс ImproveSceneManager.cs 21](#_Toc40645602)

[1.17 Класс Info.cs 22](#_Toc40645603)

[1.18 Класс LaserGun.cs 22](#_Toc40645604)

[1.19 Класс LevelGenerator.cs 25](#_Toc40645605)

[1.20 Класс LoadInfoGame.cs 27](#_Toc40645606)

[1.21 Класс Magnet.cs 27](#_Toc40645607)

[1.22 Класс MagnetImproving.cs 28](#_Toc40645608)

[1.23 Класс MainMenu.cs 29](#_Toc40645609)

[1.24 Класс Obstacle.cs 31](#_Toc40645610)

[1.25 Класс PauseMenu.cs 31](#_Toc40645611)

[1.26 Класс player\_money.cs 35](#_Toc40645612)

[1.27 Класс player\_score.cs 35](#_Toc40645613)

[1.28 Класс PlayerController.cs 36](#_Toc40645614)

[1.29 Класс SaveLoadManager.cs 42](#_Toc40645615)

[1.30 Класс SetInfoGame.cs 44](#_Toc40645616)

[1.31 Класс SoundColya.cs 44](#_Toc40645617)

[1.32 Класс SoundScript.cs 44](#_Toc40645618)

[1.33 Класс Teleport.cs 45](#_Toc40645619)

[2. ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ 46](#_Toc40645620)

[ПРИЛОЖЕНИЕ 1 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ 47](#_Toc40645621)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 49](#_Toc40645622)

1. ТЕКСТ ПРОГРАММЫ
   1. Класс BlockInfo.cs

using UnityEngine;

public class BlockInfo : MonoBehaviour

{

// Изменение координат конца относительно

// начала платформы.

public int X;

public int Y;

}

* 1. Класс Camera.cs

using UnityEngine;

public class Camera : MonoBehaviour

{

// Игрок за которым надо следовать.

public GameObject player;

// Коефициент для Lerp.

public float koef = 0.1f;

// Расстояние на которое может камера отклониться от

// player.

public float delta = 0;

// Умер ли игрок.

public bool IsDie = false;

// Смещение камеры по Z относительно player.

private Vector3 t = new Vector3(0, 0, -10);

private void Start()

{

// Установка начального положения камеры.

gameObject.transform.position = player.transform.position + t;

}

void Update()

{

// Если игрок жив.

if (!IsDie)

{

// Если игрок выбежал за максимальное отклонение.

if (Mathf.Abs(Vector3.Distance(player.transform.position + t, gameObject.transform.position)) > delta)

{

// Перемещаем камеру к игроку.

transform.position = Vector3.Lerp(gameObject.transform.position, player.transform.position + t, koef);

}

}

}

}

* 1. Класс Chest.cs

using UnityEngine;

public class Chest : MonoBehaviour

{

// Аниматор сундука.

public Animator animator;

// Тригер.

private BoxCollider2D bc;

// Вознаграждение за открытие.

private int cost;

private void Start()

{

// Инициализация аниматора и установка анимации "CloseChest".

animator = GetComponent<Animator>();

animator.SetBool("IsOpen", false);

// Инициализация BoxCollider 2D.

bc = GetComponent<BoxCollider2D>();

}

/// <summary>

/// Обработка открытия сундука.

/// </summary>

/// <param name="col">

/// Collider объекта, на который

/// среагировал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка на то, что тригер сработал именно на игрока.

PlayerController pl = col.GetComponent<PlayerController>();

if (pl != null)

{

// Установка анимации "OpenChest".

animator.SetBool("IsOpen", true);

// Воспроизведение звука монетки и открытия сундука.

GameObject.FindGameObjectWithTag("Sounds").GetComponent<SoundScript>().GetCoinSound();

GetComponent<AudioSource>().Play();

// Выбор вознаграждения в зависимости от пройденного

// расстояния.

if (player\_score.current\_score <= 50)

cost = 5;

else if (player\_score.current\_score > 50 && player\_score.current\_score <= 150)

cost = 10;

else

cost = 15;

// Получение вознагрождения и обновление счетчика монет.

player\_money.coin += cost;

GameObject.FindGameObjectsWithTag("Player")[0].GetComponent<player\_money>().TextMoney.text = player\_money.coin.ToString();

// Отключение тригера.

bc.isTrigger = false;

}

}

}

* 1. Класс Coin.cs

using UnityEngine;

public class Coin : MonoBehaviour

{

// Монеты разных ценностей.

public Sprite red\_coin;

public Sprite green\_coin;

private int cost = 1;

private void Start()

{

// Установка ознаграждение за монету.

if (player\_score.current\_score > 50 && player\_score.current\_score < 100)

{

GetComponent<SpriteRenderer>().sprite = red\_coin;

cost = 5;

}

else if (player\_score.current\_score > 100)

{

GetComponent<SpriteRenderer>().sprite = green\_coin;

cost = 10;

}

}

/// <summary>

/// Обработка соприкосновения с игроком.

/// </summary>

/// <param name="col">

/// // Коллайдер на который сработал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка на принадлежность коллайдера игроку.

PlayerController pl = col.GetComponent<PlayerController>();

// Инициализация магнита.

Magnet magnete = col.GetComponent<Magnet>();

if (pl != null)

{

// Прибавляем деньги если это игрок.

player\_money.coin += cost;

// Воспроизводим звук монетки.

GameObject.FindGameObjectWithTag("Sounds").GetComponent<SoundScript>().GetCoinSound();

// Меняем надпись кол-ва собранных монет.

GameObject.FindGameObjectsWithTag("Player")[0].GetComponent<player\_money>().TextMoney.text = player\_money.coin.ToString();

// Уничтожаем объект.

Destroy(gameObject);

}

else if (magnete != null)

{

// Если это магнит то добавляем объект в поле магниту.

magnete.AddToField(gameObject);

}

}

}

* 1. Класс Colya.cs

using System.Collections;

using UnityEngine;

public class Colya : MonoBehaviour

{

// Коллайдер кольев.

public BoxCollider2D bc;

// Начальная задержка.

public float start\_delay;

// Таймер.

private float timer;

// Активированы ли колья.

private bool IsAttack = false;

// Аниматор кольев.

private Animator animator;

public AudioSource source;

void Start()

{

// Установка таймера.

timer = 2;

// Отключение тригера.

bc.isTrigger = false;

// Инициализация аниматора.

animator = gameObject.GetComponentInParent<Animator>();

}

void Update ()

{

// Начальная задержка.

if (start\_delay > 0)

{

start\_delay -= Time.deltaTime;

animator.speed = 0f;

}

else

{

animator.speed = 1f;

// Периодическое включение и выключение.

if (!IsAttack)

{

if (timer > 0)

{

timer -= Time.deltaTime;

}

else

{

source.Play();

bc.isTrigger = true;

timer += 1;

timer -= Time.deltaTime;

IsAttack = true;

}

}

else

{

if (timer > 0)

{

timer -= Time.deltaTime;

}

else

{

source.Play();

bc.isTrigger = false;

timer += 2;

timer -= Time.deltaTime;

IsAttack = false;

}

}

}

}

/// <summary>

/// Обработка косания игрока.

/// </summary>

/// <param name="col">

/// Коллайдер с которым столкнулся тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка коллайдера на принадлежность игроку.

PlayerController pl = col.GetComponent<PlayerController>();

if (pl != null)

{

// Нанесение урона.

GameObject.FindGameObjectsWithTag("Player")[0].GetComponent<HealthSystem>().GetDamage();

}

}

}

* 1. Класс Deth.cs

using UnityEngine;

using UnityEngine.Audio;

public class Deth : MonoBehaviour

{

// Игрок.

public PlayerController player;

// Камера.

public GameObject camera;

// Миксер Master(родитель всех остальных миксеров).

public AudioMixerGroup master;

// Скорость.

public float speed;

// Буферное расстояние между объектом и игроком.

public float delay;

void Start()

{

player = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();

camera = GameObject.FindGameObjectWithTag("MainCamera");

// Установка задержки по y.

transform.position = player.transform.position - new Vector3(0, delay, 0);

}

void Update()

{

// При приближении на 7 блоков начинаем изменять аудио фильтр LowPas

// для эффекта погружения подводу.

if (Vector2.Distance(player.transform.position, transform.position) < 7)

master.audioMixer.SetFloat("MasterLowPassFreq", Mathf.Lerp(300, 5000, (Vector2.Distance(player.transform.position, transform.position) - 3) / 4f));

else

master.audioMixer.SetFloat("MasterLowPassFreq", 5000);

// Если игрок жив.

if (player != null)

{

// Если игрок отбежал от объекта больше чем на delay.

if ((player.transform.position.y - transform.position.y) > delay)

{

// Телепортируем объект ближе к игроку.

transform.position = player.transform.position - new Vector3(0, delay, 0);

}

// Установка объекта по центру камеры.

transform.position = new Vector3(camera.transform.position.x, transform.position.y, transform.position.z);

// Движение объекта.

transform.position += new Vector3(0, speed, 0) \* Time.deltaTime;

}

}

/// <summary>

/// Обработка соприкосновения.

/// </summary>

/// <param name="collision">

/// Коллайдер который задействовал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D collision)

{

// Проверка коллайдера на принадлежность игроку.

PlayerController pl = collision.GetComponent<PlayerController>();

if (pl != null)

{

// Смерть игрока.

player.Die(0);

}

}

}

* 1. Класс Enemy.cs

using UnityEngine;

public class Enemy : MonoBehaviour

{

// Точки между которыми перемещается враг.

public Transform[] points;

// Скорость врага.

public float speed;

// Текущая точка к которой идет.

private int current\_point = 0;

// Идет от начала к концу или наоборот.

private bool IsForward = true;

// Аниматор врага.

public Animator animator;

// Направление.

private Dir direction;

enum Dir

{

Left = 0,

Right = 1,

Up = 2,

Down = 3

}

void Start()

{

//Перемещение в первую точку

transform.position = points[0].position;

//Выбор направления.

if (points[0].position.x - points[1].position.x > 0)

direction = Dir.Left;

else if (points[0].position.x - points[1].position.x < 0)

direction = Dir.Right;

else if (points[0].position.y - points[1].position.y > 0)

direction = Dir.Down;

else if (points[0].position.y - points[1].position.y < 0)

direction = Dir.Up;

// Установка анимации в зависимости от направления.

animator.SetInteger("Dir", (int)direction);

}

void Update()

{

// Дошли ли мы до текущей точки.

if ((Vector2)transform.position == (Vector2)points[current\_point].position)

{

// Это последняя точка ?

if (current\_point == points.Length - 1)

{

// Если да то надо идти обратно.

IsForward = false;

}

// Если это первая точка то идем к концу.

else if (current\_point == 0)

{

IsForward = true;

}

// Меняем направление и текущую точку.

if (IsForward)

{

current\_point++;

if (points[current\_point - 1].position.x - points[current\_point].position.x > 0)

direction = Dir.Left;

else if (points[current\_point - 1].position.x - points[current\_point].position.x < 0)

direction = Dir.Right;

else if (points[current\_point - 1].position.y - points[current\_point].position.y > 0)

direction = Dir.Down;

else if (points[current\_point - 1].position.y - points[current\_point].position.y < 0)

direction = Dir.Up;

}

else

{

current\_point--;

if (points[current\_point + 1].position.x - points[current\_point].position.x > 0)

direction = Dir.Left;

else if (points[current\_point + 1].position.x - points[current\_point].position.x < 0)

direction = Dir.Right;

else if (points[current\_point + 1].position.y - points[current\_point].position.y > 0)

direction = Dir.Down;

else if (points[current\_point + 1].position.y - points[current\_point].position.y < 0)

direction = Dir.Up;

}

// Установка анимации в зависимости от направления.

animator.SetInteger("Dir", (int)direction);

}

else

{

// Если не дошли до текущей точки то движемся к ней.

transform.position = Vector2.MoveTowards(transform.position, points[current\_point].position, speed \* Time.deltaTime);

}

}

/// <summary>

/// Обработка соприкоснавения с игроком.

/// </summary>

/// <param name="col">

/// Коллайдер на который среагировал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка на игрока.

HealthSystem pl = col.GetComponent<HealthSystem>();

if (pl != null)

{

// Нанесение урона.

pl.GetDamage();

}

}

}

* 1. Класс Falling.cs

using UnityEngine;

public class Falling : MonoBehaviour

{

// Таймер падания.

private float timer;

// Проигрывается ли анимация падения.

private bool IsPlaying = false;

/// <summary>

/// Запуск анимации падения.

/// </summary>

public void Play()

{

// Анимация начинает проигрываться.

IsPlaying = true;

// Переводим спрайт игрока на один уровень с полом.

GetComponent<SpriteRenderer>().sortingOrder = 0;

// Устанавливаем игроку значение по z.

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

// Говорим камере перестать следить за игроком.

Camera camera = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>();

camera.IsDie = true;

}

void Start()

{

// Инициализация таймера.

timer = 2;

}

void Update()

{

// Проигрывается ли анимация и не закончилась ли она.

if (IsPlaying && timer > 0)

{

// Тик таймера.

timer -= Time.deltaTime;

// Перемещаем игрока по Y вниз.

gameObject.transform.position -= new Vector3(0, Time.deltaTime\*5f, 0);

// Уменьшаем значение альфа канала.

Color tmp = GetComponent<SpriteRenderer>().color;

GetComponent<SpriteRenderer>().color = new Color(tmp.r, tmp.g, tmp.b, tmp.a - 0.03f);

}

}

}

* 1. Класс Floor.cs

using UnityEngine;

public class Floor : MonoBehaviour

{

private void Start()

{

// Позиция элемента пола по Z = Y.

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

}

}

Класс FraseScript.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class FraseScript : MonoBehaviour

{

// Source фразы.

AudioSource source;

private void Start()

{

// Инициализация source

source = GetComponent<AudioSource>();

}

/// <summary>

/// Проигрывает фразу.

/// </summary>

public void Play()

{

source.Play();

}

}

* 1. Класс GameoverMenu.cs

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.Audio;

using UnityEngine.UI;

public class GameoverMenu : MonoBehaviour

{

public Text total\_coins;

// Меню конца игры.

public GameObject gameoverMenuUI;

// Меню паузы.

public GameObject pauseMenu;

// Аудио снапшоты для игры и для паузы.

public AudioMixerSnapshot Normal;

public AudioMixerSnapshot OnPaused;

/// <summary>

/// Вызов меню конца игры.

/// </summary>

public void GameOverMenu()

{

// Обновление монет.

total\_coins.text = "Coins: " + Info.total\_coins.ToString();

// Уничтожаем меню паузы.

Destroy(pauseMenu);

// Остановка времени.

Time.timeScale = 0f;

// Переход к снапшоту паузы.

OnPaused.TransitionTo(0.5f);

// Активация меню окончания игры.

gameoverMenuUI.SetActive(true);

}

/// <summary>

/// Перезагрузка игры.

/// </summary>

public void Restart()

{

// Установка нормального времени.

Time.timeScale = 1f;

// Переход к снапшоты для игры.

Normal.TransitionTo(0.5f);

// Загрузка текущей сцены.

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

}

/// <summary>

/// Выход в главное меню.

/// </summary>

public void MainMenu()

{

// Установка нормальной скорости времени.

Time.timeScale = 1f;

// Переход к снапшоту для игры.

Normal.TransitionTo(0f);

// Загрузка сцены "MainMenu".

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);

}

}

* 1. Класс Gun.cs

using System;

using System.Collections;

using UnityEngine;

public class Gun : MonoBehaviour

{

// Снаряд.

public GameObject ammo;

// Направление стрельбы.

public Transform shotDir;

// Таймер.

private float timer;

// Игрок.

private Transform Player;

void Start()

{

// Инициализация таймера.

timer = 2;

// Инициализация игрока.

Player = GameObject.FindGameObjectsWithTag("Player")[0].transform;

}

void Update()

{

// Если игрок находится недалеко от пушки она начинает стрелять.

if (Mathf.Abs(Player.position.y - gameObject.transform.position.y ) < 10)

{

if (timer > 0)

{

timer -= Time.deltaTime;

}

else

{

timer += 2;

Shoot();

timer -= Time.deltaTime;

}

}

}

/// <summary>

/// Выстрел

/// </summary>

void Shoot()

{

// Создание объекта снаряда.

Instantiate(ammo, shotDir.position, shotDir.rotation);

}

}

* 1. Класс GunAmmo.cs

using UnityEngine;

public class GunAmmo : MonoBehaviour

{

// Скорость снаряда.

public float speed;

// Время до уничтожения.

public float destroyTime;

void Start()

{

// Уничтожить объект через destroyTime.

Invoke("DestroyAmmo", destroyTime);

}

void Update()

{

// Движение.

transform.Translate(Vector2.right \* speed \* Time.deltaTime);

}

/// <summary>

/// Уничтожение объекта.

/// </summary>

void DestroyAmmo()

{

Destroy(gameObject);

}

/// <summary>

/// Обработка касания с игроком.

/// </summary>

/// <param name="col">

/// Коллайдер на который сработал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка на принадлежность тригера игроку.

PlayerController pl = col.GetComponent<PlayerController>();

if (pl != null)

{

// Нанесение урона и уничтожения снаряда.

GameObject.FindGameObjectsWithTag("Player")[0].GetComponent<HealthSystem>().GetDamage();

Destroy(gameObject);

}

}

}

* 1. Класс HealthBonus.cs

using UnityEngine;

public class HealthBonus : MonoBehaviour

{

// Жизни игрока.

private HealthSystem player\_hp;

void Start()

{

// Инициализация.

player\_hp = GameObject.FindGameObjectWithTag("Player").GetComponent<HealthSystem>();

}

/// <summary>

/// Обработка касания игрока.

/// </summary>

/// <param name="collision">

/// Коллайдер на который сработал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

if (player\_hp == col.GetComponent<HealthSystem>())

{

player\_hp.AddHP();

Destroy(gameObject);

}

}

}

* 1. Класс HealthImproving.cs

using UnityEngine;

public class HealthImproving : MonoBehaviour

{

// Степени заполнения полоски прокачки.

public Sprite[] bars;

// Цены на прокачку.

private int[] costs;

void Start()

{

// Инициализация цен.

costs = Info.costs;

// Установка текущего уровня прокачки.

GetComponent<SpriteRenderer>().sprite = bars[Info.numberOfLifes - 2];

}

/// <summary>

/// Улучшение.

/// </summary>

public void Improve()

{

// Улучшить можно только если есть что улучшать.

if (Info.numberOfLifes != 5)

{

// Если хватает денег.

if (Info.total\_coins >= costs[Info.numberOfLifes - 2])

{

// Вычитаем плату.

Info.total\_coins -= costs[Info.numberOfLifes - 2];

// Увеличиваем максимальное число жизней.

Info.numberOfLifes++;

// Меняем спрайт уровня прокачки.

GetComponent<SpriteRenderer>().sprite = bars[Info.numberOfLifes - 2];

}

// Сохраняем игру.

SaveLoadManager sv = new SaveLoadManager();

sv.SaveGame();

}

}

}

* 1. Класс HealthSystem.cs

using System.Collections;

using UnityEngine.UI;

using UnityEngine;

public class HealthSystem : MonoBehaviour

{

// Может ли игрок быть атакован.

public bool CanBeDamaged = true;

// Текущее здоровье.

public int health;

// Возможное количество жизней.

public int numberOfLifes;

// Картинки сердечек.

public Image[] lifes;

// Спрайты пустого и полного сердца.

public Sprite fullLife;

public Sprite emptyLife;

// Таймер для неуязвимости после получения урона.

private float timer;

void Start()

{

// Инициализация жизней и их максимального кол-ва.

health = Info.numberOfLifes;

numberOfLifes = Info.numberOfLifes;

}

void Update()

{

// Пробегаем по массиву картинок.

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

{

// Если текущщая картинка по счету меньше

// health то делаем ее fullLife.

if (i < health)

{

lifes[i].sprite = fullLife;

}

// Иначе делаем ее emptyLife.

else

{

lifes[i].sprite = emptyLife;

}

// Если текущее сердце по счету меньше

// чем макс кол-во жизней то отображаем его.

if (i < numberOfLifes)

{

lifes[i].enabled = true;

}

// Иначе нет.

else

{

lifes[i].enabled = false;

}

}

// Если игрок жив и неуязвим.

if (CanBeDamaged == false && health > 0)

{

// Тик таймера.

timer -= Time.deltaTime;

// Изменение албфа канала по синусоиде для эффекта

// моргания персоонажа.

Color tmp = GetComponent<SpriteRenderer>().color;

GetComponent<SpriteRenderer>().color = new Color(tmp.r, tmp.g, tmp.b, 0.5f \* Mathf.Sin(timer \* Mathf.PI / 0.25f) + 0.5f);

}

}

/// <summary>

/// Получение урона.

/// </summary>

public void GetDamage()

{

// Проверка на возможность нанести урон.

if (CanBeDamaged)

{

// Уменьшение хп.

health--;

// Перевод в режим неуязвимости.

CanBeDamaged = false;

// Установка таймера неуязвимости.

timer = 1;

// Запуск корутина задержки.

StartCoroutine(AttackingResist());

}

// Смерть при окончании жизней.

if (health <= 0)

gameObject.GetComponent<PlayerController>().Die(0);

}

/// <summary>

/// Увеличение жизней бонусом.

/// </summary>

public void AddHP()

{

if (health < numberOfLifes)

{

health++;

}

}

/// <summary>

/// Корутин для конца неуязвимости.

/// </summary>

IEnumerator AttackingResist()

{

// Задержка 1 сек.

yield return new WaitForSeconds(1f);

// Перевод в режим нанесения урона.

CanBeDamaged = true;

// Восстановление альфа канала.

Color tmp = GetComponent<SpriteRenderer>().color;

GetComponent<SpriteRenderer>().color = new Color(tmp.r, tmp.g, tmp.b, 1f);

}

}

* 1. Класс ImproveSceneManager.cs

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class ImproveSceneManager : MonoBehaviour

{

// Кол-во монет.

public Text coins;

// Стоимость прокачки магнита.

public Text cost\_magnet;

// Стоимость прокачки жизней.

public Text cost\_hp;

// Кнопка улучшения.

public Button improve\_health;

// Кнопка улучшения магнита.

public Button improve\_magnet;

private void Update()

{

// Обнавление кол-ва денег.

coins.text = ": " + Info.total\_coins.ToString();

// Обновление стоимостей.

if (Info.numberOfLifes - 2 < 3)

cost\_hp.text = "Cost: " + Info.costs[Info.numberOfLifes - 2].ToString();

else

{

cost\_hp.text = "Max upgrade";

improve\_health.gameObject.SetActive(false);

}

if ((int)Info.magneteLevel < 3)

cost\_magnet.text = "Cost: " + Info.costs[(int)Info.magneteLevel].ToString();

else

{

cost\_magnet.text = "Max upgrade";

improve\_magnet.gameObject.SetActive(false);

}

}

/// <summary>

/// Выход в главное меню.

/// </summary>

public void MainMenu()

{

// Загрузка сцены "MainMenu".

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 2);

}

}

* 1. Класс Info.cs

public static class Info

{

// Собрано денег.

public static int total\_coins = 0;

// Лучший рекорд.

public static int best\_record = 0;

// Максимальное кол-во жизней.

public static int numberOfLifes = 2;

// Стоимости прокачек.

public static int[] costs = new int[]{ 100, 500, 1000};

// Уровень магнита.

public static float magneteLevel = 0;

}

* 1. Класс LaserGun.cs

using UnityEngine;

public class LaserGun : MonoBehaviour

{

// Аниматор объекта LaserGun

public Animator animator;

// Задержка перед началом работы.

public float start\_delay;

// Таймер для периодического включения пушки.

private float timer;

// Находится ли пушка в состоянии подготовки

// перед выстрелом.

private bool IsPrepare = true;

// Коллайдер.

private BoxCollider2D bc;

// Звук.

public AudioSource source;

void Start()

{

// Инициализация коллайдера и аниматора.

animator = GetComponent<Animator>();

bc = gameObject.GetComponent<BoxCollider2D>();

// Установка таймера.

timer = 2;

// Изначально пушка не стреляет.

bc.isTrigger = false;

}

void Update()

{

// Начальная задержка.

if (start\_delay > 0)

{

start\_delay -= Time.deltaTime;

// При начальной задержке анимация не проигрывается.

animator.speed = 0f;

}

else

{

// После оконцания задержки запускаем анимацию.

animator.speed = 1f;

// Готовится ли пушка.

if (IsPrepare)

{

// Если да то включаем таймер подготовки.

if (timer > 0)

{

timer -= Time.deltaTime;

}

else

{

// Как только таймер закончился.

// Инициализируем таймер выстрела.

timer += 1;

// Переводим в режим атаки.

IsPrepare = false;

// Включаем тригер.

bc.isTrigger = true;

// Первый тик таймера.

timer -= Time.deltaTime;

// Включаем звук.

source.Play();

}

}

else

{

// Во время атаки.

// Ход таймера.

if (timer > 0)

{

timer -= Time.deltaTime;

}

else

{

// По окончании таймера.

// Инициализируем таймер подготовки.

timer += 2;

// Включаем режим подготовки.

IsPrepare = true;

// Отключаем тригер.

bc.isTrigger = false;

// Первый тик таймера.

timer -= Time.deltaTime;

// Выключаем звук.

source.Stop();

}

}

}

}

/// <summary>

/// Обработка соприкоснавения с лазером.

/// </summary>

/// <param name="col">

/// Коллайдер на который сработал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка принадлежности коллайдера игроку.

HealthSystem player = col.GetComponent<HealthSystem>();

if (player != null)

{

// Нанесение урона.

player.GetDamage();

}

}

}

* 1. Класс LevelGenerator.cs

using System.Collections.Generic;

using UnityEngine;

public class LevelGenerator : MonoBehaviour

{

// Префабы кусков дороги.

public GameObject[] Roads;

// Стартовый кусок дороги.

public GameObject StartBlock;

// Координаты точки в которой создается следующий префаб.

float X, Y;

// Размер буфера.

int ColSpawn = 6;

// Координата Y последнего элемента дороги.

float Ylast;

// Уничтожен ли стартовый блок.

private bool WasStartBlockDeleted = true;

// Игрок.

public Transform PlayerTransf;

// Буфер префабов

List<GameObject> CurrentBlocks = new List<GameObject>();

void Start()

{

// Инициализация координат.

X = StartBlock.transform.position.x;

Y = StartBlock.transform.position.y;

// Заполнение буфера.

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

{

SpawnBlock();

}

}

void Update()

{

CheckForSpawn();

}

/// <summary>

/// Проверка на возможность спавна нового блока и

/// уничтожение последнего блока буфера.

/// </summary>

void CheckForSpawn()

{

if (PlayerTransf != null && PlayerTransf.position.y > Ylast + 25)

{

// Уничтожение стартого блока, если надо

if (WasStartBlockDeleted)

{

Destroy(StartBlock.gameObject);

WasStartBlockDeleted = false;

}

// Спавн нового и удаление последнего.

SpawnBlock();

DestroyBlock();

}

}

/// <summary>

/// Спавн нового блока буфера.

/// </summary>

void SpawnBlock()

{

// Создание нового блока.

GameObject block = Instantiate(Roads[Random.Range(0, Roads.Length)], transform);

// Изменение координат для нового спавна.

if (CurrentBlocks.Count == 0)

{

X += StartBlock.GetComponent<BlockInfo>().X;

Y += StartBlock.GetComponent<BlockInfo>().Y;

}

else

{

X += CurrentBlocks[CurrentBlocks.Count - 1].GetComponent<BlockInfo>().X;

Y += CurrentBlocks[CurrentBlocks.Count - 1].GetComponent<BlockInfo>().Y;

}

// Установка координат созданного блока.

block.transform.position = new Vector3(X, Y, 2);

// Добавление нового блока в буфер.

CurrentBlocks.Add(block);

// Обновление координат последнего блока.

Ylast = CurrentBlocks[0].gameObject.transform.position.y;

}

/// <summary>

/// Уничтожение последнего блока.

/// </summary>

void DestroyBlock()

{

Destroy(CurrentBlocks[0].gameObject);

CurrentBlocks.RemoveAt(0);

}

}

* 1. Класс LoadInfoGame.cs

using UnityEngine;

public class LoadInfoGame : MonoBehaviour

{

void Start()

{

// Загрузка игры.

SaveLoadManager sv = new SaveLoadManager();

sv.LoadGame();

}

}

* 1. Класс Magnet.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Magnet : MonoBehaviour

{

// Монеты попавшие в поле.

private List<GameObject> CoinsInField = new List<GameObject>();

// Игрок.

private GameObject player;

// Мощность притягивания.

public float power;

// Коллайдер магнита.

public BoxCollider2D bc;

private void Start()

{

// Инициализация игрока и радиуса магнита

player = GameObject.FindGameObjectWithTag("Player");

bc.edgeRadius = Info.magneteLevel;

}

void Update()

{

// Перемещаем магнит за игроком.

transform.position = player.transform.position;

// Прохожим по списку монет в поле.

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

{

// Некоторые монеты достигнут игрока и уничтожатся

// до того как мы до них дойдем. Поэтому удалим их

// из списка.

if (CoinsInField[i] == null)

{

CoinsInField.RemoveAt(i);

}

else

{

// Остальные монеты притянем к игроку.

CoinsInField[i].transform.position = Vector3.Lerp(CoinsInField[i].transform.position, player.transform.position, power);

}

}

}

/// <summary>

/// Добавление монеты в поле.

/// </summary>

/// <param name="obj">

/// Объект монеты.

/// </param>

public void AddToField(GameObject obj)

{

CoinsInField.Add(obj);

}

}

* 1. Класс MagnetImproving.cs

using UnityEngine;

public class MagnetImproving : MonoBehaviour

{

// Степени прокачки магнита.

public Sprite[] bars;

// Стоимости прокачки.

private int[] costs;

void Start()

{

// Инициализация стоимостей.

costs = Info.costs;

// Установка текущего уровня прокачки.

GetComponent<SpriteRenderer>().sprite = bars[(int)Info.magneteLevel];

}

/// <summary>

/// Улучшение.

/// </summary>

public void Improve()

{

// Если можем улучшить.

if (Info.magneteLevel != 3)

{

// Если хватает денег.

if (Info.total\_coins >= costs[(int)Info.magneteLevel])

{

// Вычитаем плату.

Info.total\_coins -= costs[(int)Info.magneteLevel];

// Улучшаем.

Info.magneteLevel++;

// Меняем степень прокачки.

GetComponent<SpriteRenderer>().sprite = bars[(int)Info.magneteLevel];

}

// Сохраняем игру.

SaveLoadManager sv = new SaveLoadManager();

sv.SaveGame();

}

}

}

* 1. Класс MainMenu.cs

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.Audio;

using UnityEngine.UI;

public class MainMenu : MonoBehaviour

{

// Миксер.

public AudioMixer mixer;

// Меню настроек.

public GameObject option\_menu;

// Меню.

public GameObject menu;

// Сдайдеры музыки, общей громкости и звуков.

public Slider slider\_music;

public Slider slider\_master;

public Slider slider\_sound;

private void Start()

{

// Убираем эфекты.

mixer.SetFloat("MasterLowPassFreq", 5000);

// Установка слайдеров в зависимости от текущих

// настроек звука

float tmp;

mixer.GetFloat("MasterVolume", out tmp);

slider\_master.value = Mathf.Lerp(1, 0, -tmp / 60);

mixer.GetFloat("SoundVolume", out tmp);

slider\_sound.value = Mathf.Lerp(1, 0, -tmp / 40);

mixer.GetFloat("MusicVolume", out tmp);

slider\_music.value = Mathf.Lerp(1, 0, -tmp / 60);

}

/// <summary>

/// Начать игру.

/// </summary>

public void PlayGame()

{

// Загрузка сцены "Game".

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

/// <summary>

/// Перейти к улучшениям.

/// </summary>

public void Improve()

{

// Загрузка сцены "Improving".

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);

}

/// <summary>

/// Выход из игры.

/// </summary>

public void Exit()

{

// Закрытие приложения.

Application.Quit();

}

/// <summary>

/// Открывает меню настроек.

/// </summary>

public void Options()

{

menu.SetActive(false);

option\_menu.SetActive(true);

}

/// <summary>

/// Закрывает меню настроек.

/// </summary>

public void Back()

{

menu.SetActive(true);

option\_menu.SetActive(false);

}

/// <summary>

/// Изменение значения общей громкости.

/// </summary>

/// <param name="volume">

/// Значение слайдера slider\_master.

/// </param>

public void ChangeTotalVolume(float volume)

{

mixer.SetFloat("MasterVolume", Mathf.Lerp(-60, 0, volume));

}

/// <summary>

/// Изменение значения громкости звуков.

/// </summary>

/// <param name="volume">

/// Значение слайдера slider\_sound.

/// </param>

public void ChangeSoundVolume(float volume)

{

mixer.SetFloat("SoundVolume", Mathf.Lerp(-40, 0, volume));

}

/// <summary>

/// Изменение значения громкости музыки.

/// </summary>

/// <param name="volume">

/// Значение слайдера slider\_music.

/// </param>

public void ChangeMusicVolume(float volume)

{

mixer.SetFloat("MusicVolume", Mathf.Lerp(-60, 0, volume));

}

}

* 1. Класс Obstacle.cs

using UnityEngine;

public class Obstacle : MonoBehaviour

{

/// <summary>

/// Взаимодействие с игроком.

/// </summary>

/// <param name="col">

/// Коллайдер на который сработал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка на принадлежность коллайдера игроку.

PlayerController pl = col.GetComponent<PlayerController>();

if (pl != null)

{

// Возврат игрока на начальную точку.

GameObject.FindGameObjectsWithTag("Player")[0].GetComponent<PlayerController>().GoToStart();

}

}

}

* 1. Класс PauseMenu.cs

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.Audio;

using UnityEngine.UI;

public class PauseMenu : MonoBehaviour

{

// Стоит ли игра на паузе.

public static bool GameIsPaused = false;

// Все меню паузы (родительский объект).

public GameObject pauseMenuUI;

// Аудио снапшоты в игре и при паузе.

public AudioMixerSnapshot Normal;

public AudioMixerSnapshot OnPaused;

// Миксер (Master).

public AudioMixerGroup mixer;

// Сдайдеры музыки, общей громкости и звуков.

public Slider slider\_music;

public Slider slider\_master;

public Slider slider\_sound;

// Меню настройки звука.

public GameObject settings\_menu;

// Меню паузы без настроек (дочерний объект pauseMenuUI).

public GameObject menu;

private void Start()

{

// Установка слайдеров в зависимости от текущих

// настроек звука

float tmp;

mixer.audioMixer.GetFloat("MasterVolume",out tmp);

slider\_master.value = Mathf.Lerp(1, 0, -tmp / 60);

mixer.audioMixer.GetFloat("SoundVolume", out tmp);

slider\_sound.value = Mathf.Lerp(1, 0, -tmp / 40);

mixer.audioMixer.GetFloat("MusicVolume", out tmp);

slider\_music.value = Mathf.Lerp(1, 0, -tmp / 60);

}

void Update()

{

// При нажатии ESC и не окончании игры (при вызове gameoverMenu

// puseMenuUI уничтожается).

if (Input.GetKeyDown(KeyCode.Escape) && pauseMenuUI != null)

{

if (GameIsPaused)

{

Resume();

}

else

{

Pause();

}

}

}

/// <summary>

/// Продолжение игры.

/// </summary>

public void Resume()

{

// Переход к снапшоту Normal.

Normal.TransitionTo(0.5f);

// Выключение pauseMenuUI.

pauseMenuUI.SetActive(false);

// Включение дочернего меню.

menu.SetActive(true);

// Отключение меню настроек.

settings\_menu.SetActive(false);

// Скорость течения времени обычная.

Time.timeScale = 1f;

// Снятие с паузы скрипта PlayerController.

GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>().IsGamePaused = false;

// Снятие игры с паузы.

GameIsPaused = false;

}

/// <summary>

/// Установка паузы.

/// </summary>

void Pause()

{

// Переход к снапшоту OnPaused.

OnPaused.TransitionTo(0.3f);

// Активирование пауз меню.

pauseMenuUI.SetActive(true);

// Остановка времени.

Time.timeScale = 0f;

// Остановка PlayerController.

GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>().IsGamePaused = true;

// Установка паузы.

GameIsPaused = true;

}

/// <summary>

/// Возвращение в главное меню.

/// </summary>

public void MainMenu()

{

// Преход к снапшоту Normal.

Normal.TransitionTo(0.5f);

// Запись рекорда если он был побит.

if (Info.best\_record < player\_score.current\_score)

{

Info.best\_record = player\_score.current\_score;

}

// Сохранение собранных монет.

Info.total\_coins += player\_money.coin;

// Сохранение игры.

SaveLoadManager sv = new SaveLoadManager();

sv.SaveGame();

// Установка нормальной скорости течения времени.

Time.timeScale = 1f;

// Переход к счене MainMenu.

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);

}

/// <summary>

/// Переход в меню настроек.

/// </summary>

public void Options()

{

// Скрытие пауз меню.

menu.SetActive(false);

// Активация меню настроек.

settings\_menu.SetActive(true);

}

/// <summary>

/// Изменение значения общей громкости.

/// </summary>

/// <param name="volume">

/// Значение слайдера slider\_master.

/// </param>

public void ChangeTotalVolume(float volume)

{

mixer.audioMixer.SetFloat("MasterVolume", Mathf.Lerp(-60, 0, volume));

}

/// <summary>

/// Изменение значения громкости звуков.

/// </summary>

/// <param name="volume">

/// Значение слайдера slider\_sound.

/// </param>

public void ChangeSoundVolume(float volume)

{

mixer.audioMixer.SetFloat("SoundVolume", Mathf.Lerp(-40, 0, volume));

}

/// <summary>

/// Изменение значения громкости музыки.

/// </summary>

/// <param name="volume">

/// Значение слайдера slider\_music.

/// </param>

public void ChangeMusicVolume(float volume)

{

mixer.audioMixer.SetFloat("MusicVolume", Mathf.Lerp(-60, 0, volume));

}

/// <summary>

/// Возращение из меню настроек в пауз меню.

/// </summary>

public void BackButton()

{

// Скрытие меню настроек.

settings\_menu.SetActive(false);

// Активация меню паузы.

menu.SetActive(true);

}

}

* 1. Класс player\_money.cs

using UnityEngine;

using UnityEngine.UI;

public class player\_money : MonoBehaviour

{

// Деньги заработанные в текущем забеге.

static public int coin = 0;

// Счетчик монет.

public Text TextMoney;

void Start()

{

// Начальная инициализация.

coin = 0;

}

}

* 1. Класс player\_score.cs

using UnityEngine;

using UnityEngine.UI;

public class player\_score : MonoBehaviour

{

// Очки в текущем забеге.

static public int current\_score = 0;

// Счетчик очков.

public Text TextScore;

void Start()

{

// Обнуление очков.

current\_score = 0;

}

}

* 1. Класс PlayerController.cs

susing UnityEngine;

public class PlayerController : MonoBehaviour

{

/// <summary>

/// Направление персоонажа.

/// </summary>

enum Dir

{

Up,

Left,

Down,

Right

}

// Был ли побит рекорд?

private bool IsRecordWasBitten = false;

// Находится ли персоонаж на земле.

private bool IsOnGround = true;

// Движется ли игрок.

public bool IsMoving;

// Поставлена ли игра на паузу.

public bool IsGamePaused;

// Открыто ли меню конца игры.

private bool IsGameOverMenuOpen = false;

// Меню после смерти.

public GameoverMenu gameoverMenu;

// Аниматор героя.

public Animator animator;

// Щаг персоонажа.

private Vector2 step;

// Позиция в которую персоонаж перемщается.

public Vector2 new\_position;

// Начальная позиция при загрузке уровня.

private Vector2 start\_position;

// Единичные вектора для всех направлений.

private Vector2 up\_vector = new Vector2(0, 1);

private Vector2 down\_vector = new Vector2(0, -1);

private Vector2 left\_vector = new Vector2(-1, 0);

private Vector2 right\_vector = new Vector2(1, 0);

// Направление персоонажа.

private Dir direction = Dir.Up;

// Таймер задержки после смерти.

private float timer;

// Коефицент для Lerp при перемещении.

public float koef;

// Радиус прекращения действия Lerp при перемещении.

public float delta;

// Коллайдер, физическое тело и магнит.

public Rigidbody2D rb;

public BoxCollider2D bc;

public GameObject magnet;

void Start()

{

// Игра не на паузе.

IsGamePaused = false;

// Инициализация тела и коллайдера.

bc = GetComponent<BoxCollider2D>();

rb = GetComponent<Rigidbody2D>();

// Изначально new\_position равна текущей.

new\_position = rb.position;

//Инициализация стартовой позиции.

start\_position = rb.position;

// Персоонаж стоит.

IsMoving = false;

// Инициализация таймера выхода.

timer = 1;

}

void Update()

{

// Если игра не на паузе и игрок жив(при смерти уничтожается коллайдер).

if (!IsGamePaused && bc != null)

{

if (!IsMoving)

{

// Если стоим на месте, то ждем ввода.

GetInput();

}

else

{

// Иначе движимся.

Move();

}

}

// Если игрок умер и меню еще не выпвало.

else if (bc == null && !IsGameOverMenuOpen)

{

// Обрабатываем таймер.

EndGame();

}

}

/// <summary>

/// По окончании таймера включаем

/// меню конца игры.

/// </summary>

public void EndGame()

{

if (timer > 0)

timer -= Time.deltaTime;

else

{

IsGameOverMenuOpen = true;

gameoverMenu.GameOverMenu();

}

}

/// <summary>

/// Движение персоонажа.

/// </summary>

public void Move()

{

// Следующая точка перемещения за пределами радиуса Lerp(delta)?

if (Mathf.Abs(Vector2.Distance(Vector3.Lerp(rb.position, new\_position, koef), new\_position)) <= delta)

{

// Если за пределами то перемещаем персоонада в точку

// назначиния.

rb.MovePosition(new\_position);

// Проверяем находится ли он на земле.

if (!IsOnGround)

{

// Если нет вызываем смерть.

Die(1);

}

// Конец движения.

IsMoving = false;

}

else

{

// Иначе перемещаем в следующую точку с помощью Lerp.

rb.MovePosition(Vector3.Lerp(rb.position, new\_position, koef));

}

}

/// <summary>

/// Считывание ввода игрока.

/// </summary>

private void GetInput()

{

// Шаг персоонажа (обнуляем).

step = new Vector2(0, 0);

// Проверка на нажатия стрелок на клавиатуре.

if (Input.GetKey(KeyCode.UpArrow))

{

// Изменение направления.

direction = Dir.Up;

// Установка шага.

step = up\_vector;

// Изменение конечной точки.

new\_position += step;

// Сброс флага.

IsOnGround = false;

// Начало движения.

IsMoving = true;

// Подсчет очков.

if (rb.position.y - start\_position.y >= player\_score.current\_score)

{

// Прибавление очков.

player\_score.current\_score = (int)(rb.position.y - start\_position.y) + 1;

// Обновление текстового поля.

GetComponent<player\_score>().TextScore.text = player\_score.current\_score.ToString();

// Проверка на побитие рекорда.

if (player\_score.current\_score > Info.best\_record && Info.best\_record != 0 && !IsRecordWasBitten)

{

// Запуск фразы "Dominating" в случае побитя рекорда.

GameObject.FindGameObjectWithTag("Frases").GetComponent<FraseScript>().Play();

// В этой игре уже был установлен рекорд.

IsRecordWasBitten = true;

}

}

}

else if (Input.GetKey(KeyCode.DownArrow))

{

// Начало движения.

IsMoving = true;

// Изменение направления.

direction = Dir.Down;

// Установка шага.

step = down\_vector;

// Установка конечной точки.

new\_position += step;

// Сброс флага.

IsOnGround = false;

}

else if (Input.GetKey(KeyCode.LeftArrow))

{

// Начало движения.

IsMoving = true;

// Изменение направления.

direction = Dir.Left;

// Установка шага.

step = left\_vector;

// Установка конечной точки.

new\_position += step;

// Сброс флага.

IsOnGround = false;

}

else if (Input.GetKey(KeyCode.RightArrow))

{

// Начало движения.

IsMoving = true;

// Изменение направления.

direction = Dir.Right;

// Установка шага.

step = right\_vector;

// Установка конечной точки.

new\_position += step;

// Сброс флага.

IsOnGround = false;

}

// Изменение анимации движения в зависимости от направления.

animator.SetInteger("Dir", (int)direction);

}

/// <summary>

/// Проверка на соприкосновения с землёй.

/// </summary>

/// <param name="col">

/// Коллайдер объекта с которым соприкаслись.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка принадлежности коллайдера полу.

Floor f = col.GetComponent<Floor>();

if (f != null)

{

IsOnGround = true;

}

}

/// <summary>

/// Смерть персоонажа.

/// </summary>

/// <param name="code">

/// 0 - смерть из-за окончания жизней.

/// 1 - смерть из-зи падания.

/// </param>

public void Die(int code)

{

// Вызов соответсвующей анимации в зависимоти от типа смерти.

if (code == 0)

animator.Play("Death");

else

GetComponent<Falling>().Play();

// Уничтожение тела и коллайдера и магнита.

Destroy(bc);

Destroy(rb);

Destroy(magnet);

// Остановка игры.

IsGamePaused = true;

// Сохранение нового рекорда, если таковой был установлен.

if (Info.best\_record < player\_score.current\_score)

{

Info.best\_record = player\_score.current\_score;

}

// Сохранение собранных монет.

Info.total\_coins += player\_money.coin;

// Сохраниение игры.

SaveLoadManager sv = new SaveLoadManager();

sv.SaveGame();

}

/// <summary>

/// Возвращение к start\_position.

/// Вызываетс в случае столкновения с объектом Chest.

/// </summary>

public void GoToStart()

{

// Начало движения.

IsMoving = true;

// Разворот шага на противоположный.

step \*= -1;

// Пересчет новой конечной точки движения.

new\_position = new\_position + step;

// Изменение направления на противоположное.

if (direction == Dir.Down)

{

direction = Dir.Up;

}

else if (direction == Dir.Up)

{

direction = Dir.Down;

}

else if (direction == Dir.Right)

{

direction = Dir.Left;

}

else if (direction == Dir.Left)

{

direction = Dir.Right;

}

// Установка флага т.к. мы вернулись откуда пришли.

IsOnGround = true;

}

}

* 1. Класс SaveLoadManager.cs

using UnityEngine;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using System;

public class SaveLoadManager : MonoBehaviour

{

// Путь к файлу сохранения.

private string filePath = "save.gamesave";

private void Start()

{

// Если нет файла сохранения, то сохраняем игру.

if (!File.Exists("save.game"))

{

SaveGame();

}

}

/// <summary>

/// Сохранение игры.

/// </summary>

public void SaveGame()

{

// Создание форматераю.

BinaryFormatter bf = new BinaryFormatter();

try

{

// Открываем поток для записи файла.

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

{

// Создаем объект сохранения.

Save save = new Save(Info.total\_coins, Info.best\_record, Info.numberOfLifes, Info.costs, Info.magneteLevel);

// Сериализуем его.

bf.Serialize(fs, save);

}

}

catch(Exception ex)

{

print(ex.Message);

}

}

/// <summary>

/// Загрузка игры.

/// </summary>

public void LoadGame()

{

// Если файла нет, то сохраняем игру и выходим.

if (!File.Exists(filePath))

{

SaveGame();

return;

}

// Создание форматера.

BinaryFormatter bf = new BinaryFormatter();

try

{

// Открываем поток для чтения файла.

using (FileStream fs = new FileStream(filePath, FileMode.Open))

{

// Десериализуем объект Save.

Save save = (Save)bf.Deserialize(fs);

// Устанавливаем его значения в Info.

Info.best\_record = save.best\_record;

Info.total\_coins = save.total\_coins;

Info.numberOfLifes = save.numOfLifes;

Info.costs = save.costs;

Info.magneteLevel = save.magneteLevel;

}

}

catch(Exception ex)

{

print(ex.Message);

}

}

}

// Объект сохранения.

[System.Serializable]

public class Save

{

// Конструктор.

public Save(int tc, int br, int numOfLifes, int[] costs, float magneteLevel)

{

total\_coins = tc;

best\_record = br;

this.numOfLifes = numOfLifes;

this.costs = costs;

this.magneteLevel = magneteLevel;

}

// Уровень магнита.

public float magneteLevel;

// Стоимость прокачек.

public int[] costs;

// Кол-во накопленных монет.

public int total\_coins;

// Рекорд.

public int best\_record;

// Максимальное кол-во жизней.

public int numOfLifes;

}

* 1. Класс SetInfoGame.cs

using UnityEngine.UI;

using UnityEngine;

public class SetInfoGame : MonoBehaviour

{

// UI элемент текста рекорда.

public Text record;

void Start()

{

// Установка текущего рекорда.

record.text = Info.best\_record.ToString();

}

}

* 1. Класс SoundColya.cs

using UnityEngine;

public class SoundColya : MonoBehaviour

{

// Пол на котором находится звук.

public GameObject pref;

void Start()

{

// Компенсация сдвига пола.

transform.position += new Vector3(0, 0, -pref.transform.position.z);

}

}

* 1. Класс SoundScript.cs

using UnityEngine;

public class SoundScript : MonoBehaviour

{

// AudioSource со звуком монеты.

AudioSource coins;

void Start()

{

// Инициализация.

coins = GetComponent<AudioSource>();

}

/// <summary>

/// Воспроизводит звук сбора монеты.

/// </summary>

public void GetCoinSound()

{

// Увеличиваем pitch пока он < 1.5,

// затем сбрасываем.

coins.pitch += 0.05f;

if (coins.pitch > 1.5)

coins.pitch = 1;

// Воспроизведение звука.

coins.Play();

}

}

* 1. Класс Teleport.cs

using UnityEngine;

public class Teleport : MonoBehaviour

{

// Точка куда телепортируются

public Floor finishPoint;

// Точка откуда телепортируются.

public Floor startPoint;

/// <summary>

/// Обработка соприкосновения с игроком.

/// </summary>

/// <param name="col">

/// Коллайдер на который сработал тригер.

/// </param>

private void OnTriggerEnter2D(Collider2D col)

{

// Проверка на принадлежность коллайдера игроку.

PlayerController pl = col.GetComponent<PlayerController>();

if(pl != null)

{

// Меняем новую конечную позицию игрока на конечную точку.

pl.new\_position += (Vector2)(finishPoint.transform.position - startPoint.transform.position);

// Перемещаем игрока на конечню точку.

pl.transform.position += (finishPoint.transform.position - startPoint.transform.position);

}

}

}

1. ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ
2. ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
11. ГОСТ 15150-69 Машины, приборы и другие технические изделия. Исполнения для различных климатических районов. Категории, условия эксплуатации, хранения и транспортирования в части воздействия климатических факторов внешней среды. – М.: Изд-во стандартов, 1997.
12. Unity Learn URL: <https://learn.unity.com/tutorials> (дата обращения: 27.11.2019)
13. Unity Documentation URL: <https://docs.unity3d.com/ru/530/Manual/> (дата обращения: 27.11.2019)
14. System requirements for Unity 2019.3 <URL:https://docs.unity3d.com/Manual/system-requirements.html> (дата обращения 04.05.2020)

# ПРИЛОЖЕНИЕ 1 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ

|  |  |
| --- | --- |
| Классы | Назначение |
| BlockInfo | Класс для хранения смещения координат начала и конца данного префаба. |
| Camera | Класс для слежения камеры за игроком. |
| Chest | Класс сундука, обработка открытия и информация о содержимом. |
| Coin | Класс монеты, обработка подбора, обработка взаимодействия с магнитом, информация о ценности. |
| Colya | Класс кольев, обработка нанесения урона игроку. |
| Death | Класс демона, движение за игроком, обработка убийства игрока. |
| Enemy | Класс врага, движение между точками, обработка столкновения с игроком. |
| Falling | Класс анимации падения игрока, после выхода за пределы поля. |
| Floor | Класс ячейки пола, выставляет координату по Z относительно координаты по Y. |
| FraseScript | Класс фразы при побитии рекорда, воспроизведение фразы. |
| GameOverMenu | Класс меню окончания игры, обработка нажатий на кнопки изменение звуковых настроек. |
| Gun | Класс пушки, периодическая стрельба объектом GunAmmo. |
| GunAmmo | Класс снаряда пушки, движения, обработка касания игрока. |
| HealthBonus | Класс бонуса жизней, обработка подбора игроком. |
| HealthImproving | Класс улучшения жизней, обработка попытки улучшить кол-во жизней. |
| HealthSystem | Класс системы жизней игрока, отображение информации о кол-ве жизней, обработка получения урона и лечения. |
| ImproveSceneMeneger | Класс управления сценой улучшения, обработка нажатий кнопок и обновление текстовой информации на сцене. |
| Info | Статический класс для хранения информации об игроке во время работы программы. |
| LaserGun | Класс лазерной пушки, анимация, периодическая работа, обработка касания игрока. |
| LevelGenerator | Класс генерации раунда. |
| LoadInfoGame | Класс загрузки информации об игроке при запуске программы. |
| Magnet | Класс магнита, обработка захваченных в поле монет, добавление и удаление монет из поля. |
| MagnetImproving | Класс улучшения магнита, обработка попытки улучшения магнита. |
| MainMenu | Класс главного меню, обработка работы кнопок и слайдеров. |
| Obstacle | Класс препятствия, отталкивает игрока на начальную позицию при попытке пройти сквозь. |
| PauseMenu | Класс меню паузы, обработка работы кнопок. |
| Player\_money | Класс отвечающий за подсчет денег игрока во время забега. |
| Player\_score | Класс отвечающий за подсчет очков игрока. |
| PlayerController | Класс игрока отвечающий за управление, движение, смерть персонажа. |
| SaveLoadManager | Класс отвечающий за сохранение и загрузку информаций об игроке в и из памяти компьютера |
| SetInfoGame | Класс установки текущего рекорда при начале раунда. |
| SoundColya | Класс отвечающи за компенсацию положения объекта звука по оси Z из-за смещения клеток пола. |
| SoundScript | Класс отвечает за воспроизведения звука монеты и изменения ее pitch при сборе монет подряд. |
| Teleport | Класс телепорта, отвечает за обработку телепортации игрока. |

# 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 | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |