ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

СОГЛАСОВАНО

Доцент департамента программной инженерии факультета компьютерных наук, канд. техн. наук

_____ С. Л. Макаров

«___» _____ 2019 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия», канд. техн. наук, профессор

_____ В.В. Шилов

«___» _____2019 г.

Подп. и дата	
Инв. № дубл.	
Взам. Инв. №	
Подп. и дата	
Инв. № подл.	

Многоуровневая аркада в среде Unity 3D

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ
RU.17701729.04.01-01 12 01-1-ЛУ

Исполнитель

Студентка группы БПИ183

_____/ А. И. Кукина /

«___»_____2019 г.

Москва 2019

УТВЕРЖДЕН
RU.17701729.04.01-01 12 01-1 ЛУ

Многоуровневая аркада в среде Unity 3D

Текст программы

RU.17701729.04.01-01 12 01-1

Листов 41

Подп. и дата	
Инв. № дубл.	
Взам. Инв. №	
Подп. и дата	
Инв. № подл.	

Москва 2019

СОДЕРЖАНИЕ

1. ТЕКСТ ПРОГРАММЫ 3
1.1 Класс Audio.cs 3
1.2 Класс Boss.cs 4
1.3 Класс Boss_L1.cs 6
1.4 Класс Boss_L2.cs 6
1.5 Класс Boss_L3.cs 7
1.6 Класс Bullet.cs 7
1.7 Класс Bullet_ Boss2.cs 8
1.8 Класс BulletBoss.cs 9
1.9 Класс BulletEnemy.cs 10
1.10 Класс CameraController.cs 10
1.11 Класс Enemy.cs 12
1.12 Класс EnemyMelee.cs 15
1.13 Класс EnemyRange.cs 16
1.14 Класс Item.cs 18
1.15 Класс LevelGenerator.cs 19
1.16 Класс LevelInformstor.cs 25
1.17 Класс MeleeAttack.cs 26
1.18 Класс MineController.cs 27
1.19 Класс PlayerController.cs 28
1.20 Класс SceneLoader.cs 37
1.21 Класс ShowHistory.cs 39
ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 41

1. ТЕКСТ ПРОГРАММЫ

Программа написана на языке C# (использована среда разработки Microsoft Visual Studio 2017). Программа состоит из 21 класса (учтены только вручную написанные классы, полностью автоматически сгенерированные классы не учтены).

В данном документе содержится только вручную написанный исходный код программы, код из полностью автоматически сгенерированных исходных файлов в данном документе не представлен.

1.1 Класс Audio.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

/// <summary>
/// Класс для управления звуком.
/// </summary>
public class Audio : MonoBehaviour
{
    /// <summary>
    /// Источник звука.
    /// </summary>
    AudioSource audS;

    /// <summary>
    /// Слайдер, отвечающий за громкость звука.
    /// </summary>
    [SerializeField] Slider vol;


    /// <summary>
    /// Установка начальной громкости звука или загрузка сохраненной.
    /// </summary>
    void Start()
    {
        audS = gameObject.GetComponent<AudioSource>();
        if (PlayerPrefs.HasKey("Volume"))
        {
            audS.volume = PlayerPrefs.GetFloat("Volume");
            vol.value = PlayerPrefs.GetFloat("Volume");
        }
    }

    /// <summary>
    /// Обновление и сохранение громкости.
    /// </summary>
    void Update()
    {
        if (SceneManager.GetActiveScene().buildIndex == 0 && gameObject.tag == "Audio")
        {
            Destroy(gameObject);
        }
        else if (SceneManager.GetActiveScene().buildIndex != 0 && gameObject.tag != "Audio")
            Destroy(gameObject);
        audS.volume = vol.value;
        PlayerPrefs.SetFloat("Volume", vol.value);
    }
}
```

1.2 Класс Boss.cs

```
using UnityEngine;
using System.Collections;

/// <summary>
/// Абстрактный класс босса.
/// </summary>
public abstract class Boss : Enemy
{
/// <summary>
/// Пуля, которую выпускает босс.
/// </summary>
[SerializeField] protected GameObject bullet;

/// <summary>
/// Точка, на которую в очередной раз перейдет босс.
/// </summary>
protected Vector2 point = new Vector2(-7, -7);

/// <summary>
/// Начальная точка нахождения босса.
/// </summary>
protected Vector2 init;



/// <summary>
/// Задаются начальные значения. Начинается проигрывание анимации и атаки.
/// </summary>
protected void SetStartValuesForBoss()
{
boss = true;
init = transform.parent.position;
SetStartValues();
animatorController.Play("Idle");
StartCoroutine("Attack");
}

/// <summary>
/// Управление боссом. Передвижение, атака, проигрывание анимации.
/// </summary>
protected void BossController()
{
if (target != null)
{
if (!dead)
{
if (point.x.CompareTo(transform.position.x) == 0 && point.y.CompareTo(transform.position.y) == 0)
{
point = new Vector2(-7, -7);
animatorController.Play("Idle");
StartCoroutine("Attack");
}
else
{
if (point.x != -7)
transform.position = Vector2.MoveTowards(transform.position, point, speed * Time.fixedDeltaTime);
}
}
}
}
```

```
/// <summary>
/// Реализация атаки босса.
/// </summary>
/// <returns> Паузы между действиями. </returns>
protected override IEnumerator Attack()
{
animatorController.Play("Attack");
for (int i = 0; i < 5; i++)
{
base.MakeQuaternion();
yield return new WaitForSeconds(1f);
MakeShot();
}
yield return new WaitForSeconds(0.5f);
MoveOnPoint();
}

/// <summary>
/// Поворот при движении.
/// </summary>
protected override void MakeQuaternion()
{
if (point.x < transform.position.x)
transform.rotation = Quaternion.Euler(0, 180, 0);
else if (point.x > transform.position.x)
transform.rotation = Quaternion.Euler(0, 0, 0);
}

/// <summary>
/// Передвижение босса на точку.
/// </summary>
private void MoveOnPoint()
{
do
{
point = new Vector2(Random.Range(init.x - 6, init.x + 6), Random.Range(init.y - 2, init.y + 3));
} while (point.x.CompareTo(transform.position.x) == 0 && point.y.CompareTo(transform.position.y)
== 0);
MakeQuaternion();
animatorController.Play("Walk");
}

/// <summary>
/// Сделать выстрел.
/// </summary>
protected virtual void MakeShot()
{
float k = 1 / Mathf.Sqrt(2);
var b = Instantiate(bullet, transform.position, Quaternion.identity);
b.GetComponent<BulletBoss>().direction = new Vector3(1f, 0, 0);
b = Instantiate(bullet, transform.position, Quaternion.identity);
b.GetComponent<BulletBoss>().direction = new Vector3(-1f, 0, 0);
b = Instantiate(bullet, transform.position, Quaternion.identity);
b.GetComponent<BulletBoss>().direction = new Vector3(0, 1f, 0);
b = Instantiate(bullet, transform.position, Quaternion.identity);
b.GetComponent<BulletBoss>().direction = new Vector3(0, -1f, 0);
b = Instantiate(bullet, transform.position, Quaternion.identity);
b.GetComponent<BulletBoss>().direction = new Vector3(k, k, 0);
b = Instantiate(bullet, transform.position, Quaternion.identity);
b.GetComponent<BulletBoss>().direction = new Vector3(k, -k, 0);
```

```
b = Instantiate(bullet, transform.position, Quaternion.identity);
b.GetComponent<BulletBoss>().direction = new Vector3(-k, k, 0);
b = Instantiate(bullet, transform.position, Quaternion.identity);
b.GetComponent<BulletBoss>().direction = new Vector3(-k, -k, 0);
}
}
```

1.3 Класс Boss_L1.cs

```
using UnityEngine;

/// <summary>
/// Класс, управляющий боссом первого уровня.
/// </summary>
public class Boss_L1 : Boss
{
    /// <summary>
    /// Задаются начальные значения. Начинается проигрывание анимации.
    /// </summary>
    void Start()
    {
        SetStartValuesForBoss();
    }

    /// <summary>
    /// Управление боссом. Передвижение, атака, проигрывание анимации.
    /// </summary>
    void FixedUpdate()
    {
        BossController();
    }
}
```

1.4 Класс Boss_L2.cs

```
using UnityEngine;

/// <summary>
/// Класс, управляющий боссом второго уровня.
/// </summary>
public class Boss_L2 : Boss
{
    /// <summary>
    /// Задаются начальные значения. Начинается проигрывание анимации.
    /// </summary>
    void Start()
    {
        SetStartValuesForBoss();
    }

    /// <summary>
    /// Управление боссом. Передвижение, атака, проигрывание анимации.
    /// </summary>
    void FixedUpdate()
    {
        BossController();
    }

    /// <summary>
    /// Сделать выстрел.
    /// </summary>
    protected override void MakeShot()
    {
```

```
        var b = Instantiate(bullet, transform.position, Quaternion.identity);
    }
}
```

1.5 Класс Boss_L3.cs

```
using System.Collections;
using UnityEngine;

/// <summary>
/// Класс, управляющий боссом третьего уровня.
/// </summary>
public class Boss_L3 : Boss
{
    /// <summary>
    /// Противник, который будет появляться рядом с игроком.
    /// </summary>
    [SerializeField] GameObject enemy;

    /// <summary>
    /// Задаются начальные значения. Начинается проигрывание анимации.
    /// </summary>
    void Start()
    {
        SetStartValuesForBoss();
        StartCoroutine("Spawn");
    }

    /// <summary>
    /// Управление боссом. Передвижение, атака, проигрывание анимации.
    /// </summary>
    void FixedUpdate()
    {
        BossController();
    }

    /// <summary>
    /// Появление врагов каждые 10 секунд рядом с персонажем.
    /// </summary>
    /// <returns> Интервалы между появлениями. </returns>
    IEnumerator Spawn()
    {
        GameObject player = GameObject.FindGameObjectWithTag("Player");
        yield return new WaitForSeconds(10f);
        while(!dead)
        {
            Instantiate(enemy, player.transform.position, Quaternion.identity);
            yield return new WaitForSeconds(10f);
        }
    }
}
```

1.6 Класс Bullet.cs

```
using UnityEngine;

/// <summary>
/// Абстрактный класс пули.
/// </summary>
public abstract class Bullet : MonoBehaviour
{
    /// <summary>
    /// Скорость полета пули.
```

```
    /// </summary>
    [SerializeField] protected float speed = 9f;

    /// <summary>
    /// Игрок, в которого полетит пуля.
    /// </summary>
    protected GameObject player;

    /// <summary>
    /// Компонента transform игрока.
    /// </summary>
    protected Transform target;

    /// <summary>
    /// Движение пули.
    /// </summary>
    protected abstract void Move();

    /// <summary>
    /// Событие, срабатывающее, когда пуля входит в область другого объекта. Взаимодействие с
другими игровыми объектами.
    /// Если это игрок, ему наносится урон и пуля исчезает. Если это стена, пуля просто исчезает.
    /// </summary>
    /// <param name="collision"> Объект, в область которого вошла пуля. </param>
    protected void OnTriggerEnter2D(Collider2D collision)
    {
        try
        {
            if (collision.gameObject ==
GameObject.FindGameObjectWithTag("Player").transform.GetChild(0).gameObject)
            {

GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>().TakeDamage();
                Destroy(gameObject);
            }
            else if (collision.gameObject.tag == "Wall")
            {
                Destroy(gameObject);
            }
        }
        catch { }
    }
}
```

1.7 Класс Bullet_ Boss2.cs

```
using UnityEngine;

/// <summary>
/// Класс, управляющий движением пули, которая преследует игрока.
/// </summary>
public class Bullet_Boss2 : Bullet
{
    /// <summary>
    /// Точка, в которую летит пуля.
    /// </summary>
    Vector2 tr;

    /// <summary>
    /// Время, которое пуля существует.
    /// </summary>
    [SerializeField] float TimeOfLiving = 3f;
```

```
    /// <summary>
    /// Установка начальных значений.
    /// </summary>
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        target = player.transform;
    }

    /// <summary>
    /// Передвижение и счетчик времени существования.
    /// </summary>
    void Update()
    {
        if (TimeOfLiving <= 0)
            Destroy(gameObject);
        TimeOfLiving -= Time.deltaTime;
        Move();
    }

    /// <summary>
    /// Реализация движения за персонажем.
    /// </summary>
    protected override void Move()
    {
        tr = new Vector2(target.position.x, target.position.y);
        transform.position = Vector2.MoveTowards(transform.position, tr, speed * Time.deltaTime);
        if (transform.position.x == tr.x && transform.position.y == tr.y)
            Destroy(gameObject);
    }
}
```

1.8 Класс BulletBoss.cs

```
using UnityEngine;

/// <summary>
/// Класс, управляющий пулей, которая летит в определенном направлении.
/// </summary>
public class BulletBoss : Bullet
{
    /// <summary>
    /// Направление движения пули.
    /// </summary>
    public Vector3 direction = new Vector3(0, 1);

    /// <summary>
    /// Компонента Rigidbody2D пули.
    /// </summary>
    Rigidbody2D rb;

    /// <summary>
    /// Реализация движения пули.
    /// </summary>
    protected override void Move()
    {
        rb.MovePosition(transform.position + direction * speed * Time.fixedDeltaTime);
    }

    /// <summary>
    /// Установка начальных значений.
```

```
    /// </summary>
    void Start()
    {
        rb = gameObject.GetComponent<Rigidbody2D>();
    }

    /// <summary>
    /// Движение пули.
    /// </summary>
    void Update()
    {
        Move();
    }
}
```

1.9 Класс BulletEnemy.cs

```
using UnityEngine;

/// <summary>
/// Класс, управляющий пулей, которая летит в заданную точку.
/// </summary>
public class BulletEnemy : Bullet
{
    /// <summary>
    /// Точка, в которую летит пуля.
    /// </summary>
    Vector2 tr;

    /// <summary>
    /// Установка начальных значений.
    /// </summary>
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        target = player.transform;
        tr = new Vector2(target.position.x, target.position.y);
    }

    /// <summary>
    /// Движение пули.
    /// </summary>
    void Update()
    {
        Move();
    }

    /// <summary>
    /// Реализация движения пули.
    /// </summary>
    protected override void Move()
    {
        transform.position = Vector2.MoveTowards(transform.position, tr, speed * Time.deltaTime);
        if (transform.position.x == tr.x && transform.position.y == tr.y)
            Destroy(gameObject);
    }
}
```

1.10 Класс CameraController.cs

```
using UnityEngine;

/// <summary>
```

```
/// Класс, управляющий движением камеры.
/// </summary>
public class CameraController : MonoBehaviour
{
    //ширина и высота экрана
    float width = 1920 / 90f;
    float height = 1080 / 90f;

    /// <summary>
    /// Игрок, за которым следует камера.
    /// </summary>
    GameObject player;

    /// <summary>
    /// Установка начальных значений.
    /// </summary>
    private void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }

    /// <summary>
    /// Проверка позиции игрока и передвижение камеры.
    /// </summary>
    void Update()
    {
        if (player != null)
            CheckPlayerPosition();
    }

    /// <summary>
    /// Реализация передвижения камеры при переходе игрока в другую комнату.
    /// </summary>
    private void CheckPlayerPosition()
    {
        if (player.transform.position.x > transform.position.x + 9.2)
            moveCameraHorizontal(1);
        else if (player.transform.position.x < transform.position.x - 9.5)
            moveCameraHorizontal(-1);
        else if (player.transform.position.y > transform.position.y + 5.2)
            moveCameraVertical(1);
        else if (player.transform.position.y < transform.position.y - 5.3)
            moveCameraVertical(-1);
    }

    /// <summary>
    /// Горизонтальное передвижение камеры.
    /// </summary>
    /// <param name="i"> Направление передвижения. </param>
    public void moveCameraHorizontal(int i)
    {
        if (i > 0)
        {
            transform.position += new Vector3(width, 0f, 0f);
            player.transform.position += new Vector3(3f, 0f, 0f);
        }
        else
        {
            transform.position -= new Vector3(width, 0f, 0f);
            player.transform.position += new Vector3(-3f, 0f, 0f);
        }
```

```
    }

    /// <summary>
    /// Вертикальное передвижение камеры.
    /// </summary>
    /// <param name="i"> Направление передвижения. </param>
    public void moveCameraVertical(int i)
    {
        if (i > 0)
        {
            transform.position += new Vector3(0f, height, 0f);
            player.transform.position += new Vector3(0f, 2f, 0f);
        }
        else
        {
            transform.position -= new Vector3(0f, height, 0f);
            player.transform.position += new Vector3(0f, -2f, 0f);
        }
    }

}
```

1.11 Класс Enemy.cs

```
using System.Collections;
using UnityEngine;

/// <summary>
/// Абстрактный класс противника.
/// </summary>
public abstract class Enemy : MonoBehaviour
{
    /// <summary>
    /// Спрайт полоски жизни.
    /// </summary>
    [SerializeField] protected Texture2D healthBar;
    /// <summary>
    /// Координаты полоски жизни относительно противника.
    /// </summary>
    [SerializeField] protected int barX;
    [SerializeField] protected int barY;

    /// <summary>
    /// Скорость противника.
    /// </summary>
    [SerializeField] protected float speed = 5f;
    /// <summary>
    /// Максимальное количество жизни противника.
    /// </summary>
    public float health = 20;
    /// <summary>
    /// Текущее количество жизни противника.
    /// </summary>
    protected float curHealth;
    /// <summary>
    /// Количество очков за убийство противника.
    /// </summary>
    [SerializeField] protected int points = 100;

    /// <summary>
    /// Расстояние, с которого противник может атаковать.
    /// </summary>
```

```
[SerializeField] protected float distOfHitting = 1f;
/// <summary>
/// Время удара.
/// </summary>
[SerializeField] protected float timeOfShot = 1f;
/// <summary>
/// Время между ударами.
/// </summary>
[SerializeField] protected float timeBetweenShots = 1f;
/// <summary>
/// Персонаж, на которого направлена атака.
/// </summary>
protected GameObject target;
/// <summary>
/// Время, прошедшее с момента удара.
/// </summary>
protected float currentTime;

/// <summary>
/// Компонента Rigidbody2D противника.
/// </summary>
protected Rigidbody2D rb;

/// <summary>
/// Управление анимацией.
/// </summary>
protected Animator animatorController;

/// <summary>
/// Совершает ли противник удар.
/// </summary>
protected bool inAttack = false;
/// <summary>
/// Жив ли противник.
/// </summary>
public bool dead = false;
/// <summary>
/// Расстояние до игрока.
/// </summary>
protected float dist;

/// <summary>
/// Босс ли это.
/// </summary>
protected bool boss = false;

/// <summary>
/// Предметы, которые могут выпасть после смерти.
/// </summary>
[SerializeField] protected GameObject[] items = new GameObject[0];

/// <summary>
/// Установка начальных значений.
/// </summary>
protected void SetStartValues()
{
    target = GameObject.FindGameObjectWithTag("Player");
    rb = GetComponent<Rigidbody2D>();
    currentTime = -timeBetweenShots;
    curHealth = health;
    animatorController = GetComponent<Animator>();
```

```
        Physics2D.IgnoreCollision(gameObject.GetComponent<Collider2D>(),
target.GetComponent<Collider2D>());
    }

    /// <summary>
    /// Поворот в сторону игрока.
    /// </summary>
    protected virtual void MakeQuaternion()
    {
        try
        {
            if (transform.position.x > target.transform.position.x)
                transform.rotation = Quaternion.Euler(0, 180, 0);
            else
                transform.rotation = Quaternion.Euler(0, 0, 0);
        }
        catch { }
    }

    /// <summary>
    /// Получение урона и проверка на смерть.
    /// </summary>
    protected void TakeDamageEnemy()
    {
        curHealth -= 1.0f*target.gameObject.GetComponent<PlayerController>().attackForce / 100;
        if (curHealth <= 0)
        {
            if (!dead)
                StartCoroutine("Dead");
            target.GetComponent<PlayerController>().AddPointsToScore(points);
            Destroy(gameObject, 1.5f);
        }
    }

    /// <summary>
    /// Смерть противника.
    /// </summary>
    /// <returns> Промежутки между действиями. </returns>
    IEnumerator Dead()
    {
        dead = true;
        animatorController.Play("Dead");
        yield return new WaitForSeconds(1.3f);
        Instantiate(items[Random.Range(0, items.Length)], gameObject.transform.position + new
Vector3(0, -1f, 0), gameObject.transform.rotation);
        yield return new WaitForSeconds(0.1f);
        if (boss)
            PlayerController.KillThemAll();
        else
            yield return new WaitForSeconds(0.1f);

    }

    /// <summary>
    /// Получить урон при соприкосновении с миной.
    /// </summary>
    /// <param name="collision"> Объект, с которым было соприкосновение. </param>
    protected void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Mine")
```

```
        {
            Destroy(collision.gameObject);
            TakeDamageEnemy();
        }
    }

    /// <summary>
    /// Атака.
    /// </summary>
    /// <returns></returns>
    protected abstract IEnumerator Attack();

    /// <summary>
    /// Реализация полоски жизни над противником.
    /// </summary>
    protected void OnGUI()
    {
        if (curHealth > 0)
        {
            Vector3 posSqr = Camera.main.WorldToScreenPoint(transform.position);
            GUI.Box(new Rect(posSqr.x - barX, Screen.height - posSqr.y - barY, (health * 10),
10), "");
            GUI.DrawTexture(new Rect(posSqr.x - barX, Screen.height - posSqr.y - barY,
(curHealth) * 10, 10), healthBar);
        }
    }
}
```

1.12 Класс EnemyMelee.cs

```
using System.Collections;
using UnityEngine;

/// <summary>
/// Класс, управляющий противником ближнего боя.
/// </summary>
public class EnemyMelee : Enemy
{
    /// <summary>
    /// Область поражения.
    /// </summary>
    [SerializeField] GameObject hitArea;

    /// <summary>
    /// Установка начальных значений.
    /// </summary>
    private void Start()
    {
        SetStartValues();
        hitArea = transform.Find("HitArea").gameObject;
        hitArea.SetActive(false);
    }
    /// <summary>
    /// Управление противником. Передвижение, атака, проигрывание анимации.
    /// </summary>
    private void FixedUpdate()
    {
        ControlEnemy();
    }

    /// <summary>
    /// Управление противником.
```

```
    /// </summary>
    private void ControlEnemy()
    {
        if (target == null)
            animatorController.Play("Win");
        else
        {
            if (dead)
                animatorController.Play("Dead");
            else if (!inAttack)
            {
                dist = Vector2.Distance(transform.position, target.transform.position);
                MakeQuaternion();

                if (dist >= distOfHitting)
                {
                    transform.position = Vector2.MoveTowards(transform.position,
target.transform.position, speed * Time.deltaTime);
                    animatorController.Play("Walk");
                }
                else if (dist < distOfHitting)
                {
                    StartCoroutine("Attack");
                }
            }
        }
    }

    /// <summary>
    /// Реализация атаки.
    /// </summary>
    /// <returns> Промежутки между действиями. </returns>
    protected override IEnumerator Attack()
    {
        inAttack = true;
        animatorController.Play("Attack");
        yield return new WaitForSeconds(0.1f);
        hitArea.SetActive(true);
        yield return new WaitForSeconds(0.4f);

        hitArea.SetActive(false);
        animatorController.Play("Rest");
        yield return new WaitForSeconds(0.3f);
        inAttack = false;
    }
}
```

1.13 Класс EnemyRange.cs

```
using UnityEngine;
using System.Collections;

/// <summary>
/// Класс, управляющий противником дальнего боя.
/// </summary>
public class EnemyRange : Enemy
{
    /// <summary>
    /// Пуля, которую выпускает противник.
    /// </summary>
    [SerializeField] GameObject bullet = null;
    /// <summary>
```

```
    /// Расстояние, с которого противник начинает убегать.
    /// </summary>
    [SerializeField] float distOfLeaving = 0;

    /// <summary>
    /// Установка начальных значений.
    /// </summary>
    void Start()
    {
        SetStartValues();
    }

    /// <summary>
    /// Управление противником. Передвижение, атака, проигрывание анимации.
    /// </summary>
    void FixedUpdate()
    {
        ControlEnemy();
    }

    /// <summary>
    /// Управление противником.
    /// </summary>
    private void ControlEnemy()
    {
        if (target == null)
            animatorController.Play("Attack");
        else
        {
            if (dead)
                animatorController.Play("Dead");
            else
            {
                dist = Vector2.Distance(transform.position, target.transform.position);
                MakeQuaternion();

                if (dist >= distOfHitting)
                {
                    transform.position = Vector2.MoveTowards(transform.position,
target.transform.position, speed * Time.deltaTime);
                    animatorController.Play("Walk");
                }
                else if (dist < distOfHitting && dist >= distOfLeaving)
                {
                    animatorController.Play("Attack");
                }
                else if (dist < distOfLeaving)
                {
                    transform.position = Vector2.MoveTowards(transform.position,
target.transform.position, -speed * Time.deltaTime);
                    animatorController.Play("Walk");
                }
                if (!inAttack)
                    StartCoroutine("Attack");
            }
        }
    }

    /// <summary>
    /// Реализация атаки.
    /// </summary>
```

```
    /// <returns> Промежутки между действиями. </returns>
    protected override IEnumerator Attack()
    {
        inAttack = true;
        Instantiate(bullet, transform.position + new Vector3(0, -0.7f, 0f), Quaternion.identity);
        yield return new WaitForSeconds(3f);
        inAttack = false;
    }
}
```

1.14 Класс Item.cs

```
using UnityEngine;

/// <summary>
/// Класс, управляющий бонусами.
/// </summary>
public class Item : MonoBehaviour
{
    /// <summary>
    /// Игрок, на которого направлено действие бонусов.
    /// </summary>
    GameObject player;

    /// <summary>
    /// Тип бонуса
    /// 0 - жизнь+, 1 - защита+, 2 - атака+,
    /// 3 - жизнь-, 4 - защита-, 5 - атака-,
    /// 6 - монета, 7 - случайное
    /// </summary>
    [SerializeField] int type;

    /// <summary>
    /// Поиск игрока и определение эффекта случайного бонуса.
    /// </summary>
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        if (type == 7)
        {
            type = Random.Range(0, 7);
        }
    }

    /// <summary>
    /// Применение эффекта при вхождении игрока в область бонуса.
    /// </summary>
    /// <param name="collision"> Объект, вошедший в область бонуса. </param>
    private void OnTriggerEnter2D(Collider2D collision)
    {
        try
        {
            if (collision.gameObject.tag == "Player" || collision.transform.parent.tag ==
"Player")
            {
                player.GetComponent<PlayerController>().GetItem(type);
                Destroy(gameObject);
            }
        }
        catch { }
    }
}
```

1.15 Класс LevelGenerator.cs

```
using UnityEngine;

/// <summary>
/// Класс, отвечающий за генерацию уровня.
/// </summary>
public class LevelGenerator : MonoBehaviour
{
    /// <summary>
    /// Вероятность, с которой будет построена комната.
    /// </summary>
    [SerializeField] float probability = 1;

    /// <summary>
    /// Максимальная глубина из центра уровня к крайней комнате.
    /// </summary>
    public int maxDepth = 1;

    //Массивы с шаблонами разных комнат.
    [SerializeField] GameObject[] fightRooms = new GameObject[1];
    [SerializeField] GameObject[] bossRooms = new GameObject[1];
    [SerializeField] GameObject[] trapRooms = new GameObject[1];
    [SerializeField] GameObject[] safeRooms = new GameObject[1];
    [SerializeField] GameObject startRoom;

    //Ширина и высота комнаты.
    float width = 1920 / 90f;
    float height = 1080 / 90f;

    /// <summary>
    /// Карта уровня.
    /// </summary>
    GameObject[,] Map;

    /// <summary>
    /// Объект игрока.
    /// </summary>
    GameObject player;

    /// <summary>
    /// Устанавливаются начальные значения. Генерируется карта уровня.
    /// </summary>
    void Start()
    {
        Map = new GameObject[2 * maxDepth + 1, 2 * maxDepth + 1];
        Draw();
        AddBoss();

        player = GameObject.FindGameObjectWithTag("Player");
    }

    /// <summary>
    /// Список комнат, в которых был игрок.
    /// </summary>
    string roomNames = "";
```

```
    /// <summary>
    /// Комната, в которой в данный момент находится игрок.
    /// </summary>
    string _roomName = "";

    /// <summary>
    /// Позиция текущей комнаты на карте.
    /// </summary>
    int x = 0, y = 0;

    /// <summary>
    /// Обновление карты. Установка новых комнат, блокировка дверей.
    /// </summary>
    void Update()
    {
        try
        {
            PlaceNewRoom();
            try { if (GameObject.FindGameObjectsWithTag("Enemy")[0] == null) { }; }
            catch
            {
                Transform r = GameObject.Find(_roomName).transform;
                Transform cols = r.Find("Colliders");
                Transform doors = r.Find("Doors");
                if (doors.Find("doorT").gameObject.activeInHierarchy)

cols.Find("Top").Find("TopDoor").gameObject.GetComponent<Collider2D>().isTrigger = true;
                if (doors.Find("doorB").gameObject.activeInHierarchy)

cols.Find("Bot").Find("BotDoor").gameObject.GetComponent<Collider2D>().isTrigger = true;
                if (doors.Find("doorL").gameObject.activeInHierarchy)

cols.Find("Left").Find("LeftDoor").gameObject.GetComponent<Collider2D>().isTrigger = true;
                if (doors.Find("doorR").gameObject.activeInHierarchy)

cols.Find("Right").Find("RightDoor").gameObject.GetComponent<Collider2D>().isTrigger = true;
            }
        }
        catch { }
    }

    /// <summary>
    /// Установка комнаты в место, где стоит игрок, если комнаты еще нет.
    /// </summary>
    private void PlaceNewRoom()
    {
        if (player != null)
        {
            if (_roomName != player.GetComponent<PlayerController>().RoomName)
            {
                _roomName = player.GetComponent<PlayerController>().RoomName;
                if (!roomNames.Contains(_roomName))
                {
                    roomNames += _roomName;
                    x = int.Parse(_roomName.Substring(4, _roomName.IndexOf('_') - 4));
```

```
                    y = int.Parse(_roomName.Substring(_roomName.IndexOf('_') + 1,
_roomName.Length - _roomName.IndexOf('_') - 1));
                    var newRoom = Instantiate(Map[x, y], new Vector3((x - maxDepth) * width,
(maxDepth - y) * height, 0), transform.rotation);
                    newRoom.name = "Room" + x + "_" + y;
                    AddWallsAndDoorsToRoom(x, y);
                }
            }
        }
    }

    /// <summary>
    /// Рисует первую комнату и вызывает отрисовку последующих
    /// </summary>
    public void Draw()
    {
        Map[maxDepth, maxDepth] = startRoom;
        Draw(0, 1, 0, 1);
        Draw(1, 1, 1, 0);
        Draw(2, 1, 0, -1);
        Draw(3, 1, -1, 0);

    }

    /// <summary>
    /// Постройка очередной комнаты.
    /// </summary>
    /// <param name="direction"> В каком из четырех направлений комната будет построена. </param>
    /// <param name="curDepth"> Текущая глубина. </param>
    /// <param name="x"> Место на карте (горизонталь). </param>
    /// <param name="y"> Место на карте (вертикаль). </param>
    public void Draw(int direction, int curDepth, int x, int y)
    {
        if (curDepth <= maxDepth)
        {
            if (Random.value <= probability)
            {
                float a = Random.value;
                if (Map[maxDepth + x, maxDepth - y] == null)
                    if (a <= 0.8)
                    {
                        PlaceFightRoom(x, y);
                    }
                    else if (a <= 0.9)
                    {
                        if (safeRooms.Length > 0)
                            PlaceSafeRoom(x, y);
                        else
                            PlaceFightRoom(x, y);
                    }
                    else //if (a <= 1)
                    {
                        if (trapRooms.Length > 0)
                            PlaceTrapRoom(x, y);
                        else
                            PlaceFightRoom(x, y);
```

```
                }
            switch (direction)
            {
                case 0:
                    Draw(0, curDepth + 1, x, y + 1);
                    Draw(1, curDepth + 1, x + 1, y);
                    Draw(3, curDepth + 1, x - 1, y);
                    break;
                case 1:
                    Draw(0, curDepth + 1, x, y + 1);
                    Draw(1, curDepth + 1, x + 1, y);
                    Draw(2, curDepth + 1, x, y - 1);
                    break;
                case 2:
                    Draw(1, curDepth + 1, x + 1, y);
                    Draw(2, curDepth + 1, x, y - 1);
                    Draw(3, curDepth + 1, x - 1, y);
                    break;
                case 3:
                    Draw(0, curDepth + 1, x, y + 1);
                    Draw(2, curDepth + 1, x, y - 1);
                    Draw(3, curDepth + 1, x - 1, y);
                    break;
            }
        }
    }
}

/// <summary>
/// Поместить случайную комнату-ловушку.
/// </summary>
/// <param name="x"> Место на карте (горизонталь). </param>
/// <param name="y"> Место на карте (вертикаль). </param>
private void PlaceTrapRoom(int x, int y)
{
    Map[maxDepth + x, maxDepth - y] = trapRooms[0];
}

/// <summary>
/// Поместить случайную безопасную комнату.
/// </summary>
/// <param name="x"> Место на карте (горизонталь). </param>
/// <param name="y"> Место на карте (вертикаль). </param>
private void PlaceSafeRoom(int x, int y)
{
    Map[maxDepth + x, maxDepth - y] = safeRooms[0];
}

/// <summary>
/// Поместить случайную комнату с противниками.
/// </summary>
/// <param name="x"> Место на карте (горизонталь). </param>
/// <param name="y"> Место на карте (вертикаль). </param>
private void PlaceFightRoom(int x, int y)
{
    Map[maxDepth + x, maxDepth - y] = fightRooms[Random.Range(0, fightRooms.Length)];
```

```
}

/// <summary>
/// Установка стен и дверей на карту.
/// </summary>
private void AddWallsAndDoors()
{
    for (int i = 0; i < Map.GetLength(0); i++)
        for (int j = 0; j < Map.GetLength(1); j++)
        {
            AddWallsAndDoorsToRoom(i, j);
        }
}

/// <summary>
/// Установка стен и дверей в комнату.
/// </summary>
/// <param name="i"> Место на карте (горизонталь). </param>
/// <param name="j"> Место на карте (вертикаль). </param>
private void AddWallsAndDoorsToRoom(int i, int j)
{
    if (Map[i, j] != null)
    {
        if (i == 0 || Map[i - 1, j] == null) //Стена слева
        {
            PlaceWall(i, j, 'L');
        }
        else //Дверь слева
        {
            PlaceDoor(i, j, 'L');
        }
        if (j == 0 || Map[i, j - 1] == null) //Стена сверху
        {
            PlaceWall(i, j, 'T');
        }
        else //Дверь сверху
        {
            PlaceDoor(i, j, 'T');
        }
        if (i == Map.GetUpperBound(0) || Map[i + 1, j] == null) //Стена справа
        {
            PlaceWall(i, j, 'R');
        }
        else //Дверь справа
        {
            PlaceDoor(i, j, 'R');
        }
        if (j == Map.GetUpperBound(1) || Map[i, j + 1] == null) //Стена снизу
        {
            PlaceWall(i, j, 'B');
        }
        else //Дверь снизу
        {
            PlaceDoor(i, j, 'B');
        }
    }
```

```
}

/// <summary>
/// Установка двери в комнату.
/// </summary>
/// <param name="i"> Место на карте (горизонталь). </param>
/// <param name="j"> Место на карте (вертикаль). </param>
/// <param name="direction"> На какой стене будет дверь. </param>
private void PlaceDoor(int i, int j, char direction)
{
    GameObject room = GameObject.Find("Room" + i + "_" + j);
    Transform Doors = room.transform.Find("Doors");
    GameObject door = Doors.Find("door" + direction).gameObject;
    door.SetActive(true);
}

/// <summary>
/// Установка стены в комнату.
/// </summary>
/// <param name="i"> Место на карте (горизонталь). </param>
/// <param name="j"> Место на карте (вертикаль). </param>
/// <param name="direction"> С какой стороны будет стена. </param>
private void PlaceWall(int i, int j, char direction)
{
    GameObject room = GameObject.Find("Room" + i + "_" + j);
    Transform colliders = room.transform.Find("Colliders");
    Transform colliders2;

    Transform doors = room.transform.Find("Doors");
    doors.Find("door" + direction).gameObject.SetActive(false);

    GameObject wall;
    switch (direction)
    {
        case 'T':
            colliders2 = colliders.transform.Find("Top");
            wall = colliders2.Find("TopDoor").gameObject;
            wall.GetComponent<Collider2D>().isTrigger = false;
            break;
        case 'R':
            colliders2 = colliders.transform.Find("Right");
            wall = colliders2.Find("RightDoor").gameObject;
            wall.GetComponent<Collider2D>().isTrigger = false;
            break;
        case 'B':
            colliders2 = colliders.transform.Find("Bot");
            wall = colliders2.Find("BotDoor").gameObject;
            wall.GetComponent<Collider2D>().isTrigger = false;
            break;
        case 'L':
            colliders2 = colliders.transform.Find("Left");
            wall = colliders2.Find("LeftDoor").gameObject;
            wall.GetComponent<Collider2D>().isTrigger = false;
            break;
    }
}
```

```
    /// <summary>
    /// Установка комнаты босса на карту.
    /// </summary>
    private void AddBoss()
    {
        bool made = false;
        int k;
        int g;
        do
        {
            k = Random.Range(0, Map.GetLength(0));
            g = Random.Range(0, Map.GetLength(0));
            if (Map[k, g] == null)
            {
                try
                {
                    if (Map[k + 1, g] != null)
                    { made = true; break; }
                }
                catch { }
                try
                {
                    if (Map[k - 1, g] != null)
                    { made = true; break; }
                }
                catch { }
                try
                {
                    if (Map[k, g + 1] != null)
                    { made = true; break; }
                }
                catch { }
                try
                {
                    if (Map[k, g - 1] != null)
                    { made = true; break; }
                }
                catch { }
            }
        } while (!made);
        Map[k, g] = bossRooms[0];
    }
}
```

1.16 Класс LevelInformstor.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

/// <summary>
/// Класс, информирующий игрока о номере уровня и победе.
/// </summary>
public class LevelInformator : MonoBehaviour
{
    /// <summary>
```

```
    /// Объект, на котором отображается номер уровня.
    /// </summary>
    [SerializeField] GameObject LevelInfo;

    /// <summary>
    /// Текст с номером уровня.
    /// </summary>
    [SerializeField] Text Message;

    /// <summary>
    /// Победный экран.
    /// </summary>
    [SerializeField] GameObject WinScreen;

    /// <summary>
    /// Начать показ номера уровня.
    /// </summary>
    void Start()
    {
        StartCoroutine("Show");
    }

    /// <summary>
    /// Показать победный экран.
    /// </summary>
    public void Win()
    {
        WinScreen.SetActive(true);
    }

    /// <summary>
    /// Показывать номер уровня в течении трех секунд.
    /// </summary>
    /// <returns></returns>
    IEnumerator Show()
    {
        int a = SceneManager.GetActiveScene().buildIndex;
        Message.text = "Уровень " + a.ToString();
        LevelInfo.SetActive(true);
        yield return new WaitForSeconds(3f);
        LevelInfo.SetActive(false);
    }
}
```

1.17 Класс MeleeAttack.cs

```
using UnityEngine;

/// <summary>
/// Класс, отвечающий за нанесение урона игроку противником ближнего боя.
/// </summary>
public class MeleeAttack : MonoBehaviour
{
    /// <summary>
    /// Объект игрока.
    /// </summary>
    GameObject player;

    /// <summary>
    /// Поиск объекта игрока.
    /// </summary>
    private void Start()
```

```
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }

    /// <summary>
    /// Нанесение урона, если игрок попадает в область удара.
    /// </summary>
    /// <param name="collision"> Объект, попавший в область удара. </param>
    private void OnTriggerEnter2D(Collider2D collision)
    {

        if (collision.gameObject == player.transform.Find("PlayerField").gameObject)
        {
            player.GetComponent<PlayerController>().TakeDamage();
            gameObject.SetActive(false);
        }
    }
}
```

1.18 Класс MineController.cs

```
using UnityEngine;

/// <summary>
/// Класс, управляющий минами.
/// </summary>
public class MineController : MonoBehaviour
{
    /// <summary>
    /// Объект игрока.
    /// </summary>
    GameObject player;

    /// <summary>
    /// Находится ли игрок в области мины.
    /// </summary>
    bool inBomb = true;

    /// <summary>
    /// Поиск объекта игрока.
    /// </summary>
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }

    /// <summary>
    /// Проверка, поднимает ли игрок мину.
    /// </summary>
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.R) && inBomb)
        {
            inBomb = false;
            player.GetComponent<PlayerController>().GetMine();
            Destroy(gameObject, .2f);
        }
    }

    /// <summary>
    /// Вхождение объекта в область мины.
```

```
    /// </summary>
    /// <param name="collision"> Объект, вошедший в область мины. </param>
    private void OnTriggerEnter2D(Collider2D collision)
    {
        try
        {
            if (collision.gameObject.tag == "Player" || collision.transform.parent.tag ==
"Player")
                inBomb = true;
        }
        catch { }
    }

    /// <summary>
    /// Выход объекта из области мины.
    /// </summary>
    /// <param name="collision"> Объект, который вышел из области мины. </param>
    private void OnTriggerExit2D(Collider2D collision)
    {
        try
        {
            if (collision.gameObject.tag == "Player" || collision.transform.parent.tag ==
"Player")
            {
                inBomb = false;
            }
        }
        catch { }
    }
}
```

1.19 Класс PlayerController.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.SceneManagement;

/// <summary>
/// Класс, управляющий игроком.
/// </summary>
public class PlayerController : MonoBehaviour
{
    //Элементы интерфейса, на которых отображается статистика
    [SerializeField] Slider sliderHP;
    [SerializeField] Text numOfMinesText;
    [SerializeField] Text moneyText;
    [SerializeField] Text AttackForceText;
    [SerializeField] Text DefenceText;
    [SerializeField] GameObject DeathMenu;
    [SerializeField] GameObject GameMenu;
    [SerializeField] Text eventText;


    /// <summary>
    /// Скорость игрока.
    /// </summary>
    [SerializeField] float speed = 8f;
```

```
/// <summary>
/// Количество жизни игрока.
/// </summary>
[SerializeField] float healthPoints = 10f;

/// <summary>
/// Максимальное количество мин.
/// </summary>
[SerializeField] int maxNumOfMines = 7;

/// <summary>
/// Объект мины.
/// </summary>
[SerializeField] GameObject mine;

/// <summary>
/// Время установки мины.
/// </summary>
[SerializeField] float TimeOfMineSetting = 1f;

/// <summary>
/// Текущее время установки мины.
/// </summary>
private float CurrentTimeOfMineSetting = 0f;

/// <summary>
/// Управление анимацией.
/// </summary>
Animator animatorController;

/// <summary>
/// В какую сторону направлен игрок.
/// </summary>
char playerDirection = 'F';

/// <summary>
/// Компонента Rigidbody2D.
/// </summary>
Rigidbody2D rb;

//Ширина и высота комнат.
float width = 1920 / 90f;
float height = 1080 / 90f;
/// <summary>
/// Максимальная глубина комнат.
/// </summary>
int maxDepth;

/// <summary>
/// Определение комнаты, в которой находится игрок.
/// </summary>
public string RoomName
{
    get
    {
```

```
        string s = "Room";
        float a, b;
        if ((a = (float)(transform.position.x * 1.0 / width)) > 0)
            a = (int)(a + 0.5) + maxDepth;
        else
            a = (int)(a - 0.5) + maxDepth;

        if ((b = (float)(transform.position.y * 1.0 / height)) > 0)
            b = -1*(int)(b + 0.5) + maxDepth;
        else
            b = -1*(int)(b - 0.5) + maxDepth;

        s += a + "_" + b;
        return s;
    }
}

/// <summary>
/// Текущее количество здоровья.
/// </summary>
float _currHealthPoints;
/// <summary>
/// Текущее количество здоровья. При изменении меняет количество здоровья в интерфейсе.
/// </summary>
public float currHealthPoints
{
    get { return _currHealthPoints; }
    set
    {
        _currHealthPoints = value;
        sliderHP.value = currHealthPoints / healthPoints;
        if (currHealthPoints <= 0)
        {
            GameMenu.SetActive(false);
            DeathMenu.SetActive(true);
            Destroy(gameObject);
        }
    }
}

/// <summary>
/// Текущее количество мин.
/// </summary>
int _numberOfMines = 0;
/// <summary>
/// Текущее количество мин. При изменении изменяет количество мин в интерфейсе.
/// </summary>
public int numberOfMines
{
    get { return _numberOfMines; }
    set
    {
        _numberOfMines = value;
        numOfMinesText.text = value.ToString();
    }
}
```

```
/// <summary>
/// Количество монет.
/// </summary>
int _money = 0;
/// <summary>
/// Количество монет с отображением в интерфейсе и сохранением.
/// </summary>
public int money
{
    get { return _money; }
    set
    {
        _money = value;
        moneyText.text = value.ToString();
        PlayerPrefs.SetInt("Money", value);
    }
}

/// <summary>
/// Количество защиты.
/// </summary>
int _defence = 0;
/// <summary>
/// Количество защиты с отображением в интерфейсе.
/// </summary>
public int defence
{
    get { return _defence; }
    set
    {
        _defence = value;
        DefenceText.text = value.ToString() + "%";
    }
}

/// <summary>
/// Сила атаки.
/// </summary>
int _attackForce = 100;
/// <summary>
/// Сила атаки с отображением в интерфейсе.
/// </summary>
public int attackForce
{
    get { return _attackForce; }
    set
    {
        _attackForce = value;
        AttackForceText.text = value.ToString() + "%";
    }
}

/// <summary>
/// Установка начальных значений. Загрузка значений из сохранений.
/// </summary>
```

```
void Start()
{
    speed *= Time.fixedDeltaTime;
    transform.position = new Vector3(0, 0, 0);
    PlayerPrefs.SetInt("Level", SceneManager.GetActiveScene().buildIndex);
    rb = GetComponent<Rigidbody2D>();
    animatorController = GetComponent<Animator>();
    maxDepth = GameObject.Find("LevelGenerator").GetComponent<LevelGenerator>().maxDepth;

    LoadValues();
    sliderHP.value = currHealthPoints/healthPoints;

    CurrentTimeOfMineSetting = 0f;

    StartCoroutine("MinesAndHealthRegen");
}

/// <summary>
/// Загрузка значения из сохранений, если они есть. Иначе устанавливаются начальные значения.
/// </summary>
private void LoadValues()
{
    if (PlayerPrefs.HasKey("Mines"))
        numberOfMines = PlayerPrefs.GetInt("Mines");
    else
        numberOfMines = maxNumOfMines;
    if (PlayerPrefs.HasKey("Health"))
        currHealthPoints = PlayerPrefs.GetFloat("Health");
    else
        currHealthPoints = healthPoints;
    if (PlayerPrefs.HasKey("Defence"))
        defence = PlayerPrefs.GetInt("Defence");
    else
        defence = 0;
    if (PlayerPrefs.HasKey("Attack"))
        attackForce = PlayerPrefs.GetInt("Attack");
    else
        attackForce = 100;
    if (PlayerPrefs.HasKey("Money"))
        money = PlayerPrefs.GetInt("Money");
    else money = 0;
}

/// <summary>
/// Обновление игрока, движение, атака, анимация.
/// </summary>
void Update()
{
    if (Input.GetKeyDown(KeyCode.K))
        KillThemAll();
    if (Input.GetKeyDown(KeyCode.Escape))
        GameMenu.SetActive(!GameMenu.activeInHierarchy);
    if (CurrentTimeOfMineSetting <= 0)
    {
        MovePlayer();
        CheckAttack();
```

```
    }
    else
    {
        CurrentTimeOfMineSetting -= Time.deltaTime;
    }
}

/// <summary>
/// Получить урон.
/// </summary>
/// <param name="damage"> Количество урона (по умолчанию 1). </param>
public void TakeDamage(int damage = 1)
{
    currHealthPoints -= 1.0f*damage*(100 - defence)/100;
}

/// <summary>
/// Контролирует движение персонажа соответственно нажатым клавишам
/// </summary>
private void MovePlayer()
{
    if (Input.GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode.D))
        MoveRight();
    else if (Input.GetKey(KeyCode.LeftArrow) || Input.GetKey(KeyCode.A))
        MoveLeft();
    else if (Input.GetKey(KeyCode.UpArrow) || Input.GetKey(KeyCode.W))
        MoveBack();
    else if (Input.GetKey(KeyCode.DownArrow) || Input.GetKey(KeyCode.S))
        MoveFront();
    else
        Stay();


    if (((Input.GetKey(KeyCode.RightArrow)) & (Input.GetKey(KeyCode.UpArrow))) ||
            ((Input.GetKey(KeyCode.D)) & (Input.GetKey(KeyCode.W))))
        rb.MovePosition(rb.position + new Vector2(1, 1) * speed / (float)Math.Sqrt(2));
    else if (((Input.GetKey(KeyCode.RightArrow)) & (Input.GetKey(KeyCode.DownArrow))) ||
            ((Input.GetKey(KeyCode.D)) & (Input.GetKey(KeyCode.S))))
        rb.MovePosition(rb.position + new Vector2(1, -1) * speed / (float)Math.Sqrt(2));
    else if (((Input.GetKey(KeyCode.LeftArrow)) & (Input.GetKey(KeyCode.UpArrow))) ||
        ((Input.GetKey(KeyCode.A)) & (Input.GetKey(KeyCode.W))))
        rb.MovePosition(rb.position + new Vector2(-1, 1) * speed / (float)Math.Sqrt(2));
    else if (((Input.GetKey(KeyCode.LeftArrow)) & (Input.GetKey(KeyCode.DownArrow))) ||
            ((Input.GetKey(KeyCode.A)) & (Input.GetKey(KeyCode.S))))
        rb.MovePosition(rb.position + new Vector2(-1, -1) * speed / (float)Math.Sqrt(2));
}

/// <summary>
/// Движение вправо.
/// </summary>
private void MoveRight()
{
    rb.MovePosition(rb.position + Vector2.right * speed);
    animatorController.Play("WalkRight");
    playerDirection = 'R';
}
```

```
/// <summary>
/// Движение вниз.
/// </summary>
private void MoveFront()
{
    rb.MovePosition(rb.position - Vector2.up * speed);
    animatorController.Play("WalkFront");
    playerDirection = 'F';
}

/// <summary>
/// Движение влево.
/// </summary>
private void MoveLeft()
{
    rb.MovePosition(rb.position - Vector2.right * speed);
    animatorController.Play("WalkLeft");
    playerDirection = 'L';
}

/// <summary>
/// Движение вверх.
/// </summary>
private void MoveBack()
{
    rb.MovePosition(rb.position + Vector2.up * speed);
    animatorController.Play("WalkBack");
    playerDirection = 'B';
}

/// <summary>
/// Проигрывание анимации в состоянии покоя.
/// </summary>
private void Stay()
{
    rb.MovePosition(rb.position);
    switch (playerDirection)
    {
        case 'F':
            animatorController.Play("Idle");
            break;
        case 'R':
            animatorController.Play("IdleR");
            break;
        case 'L':
            animatorController.Play("IdleL");
            break;
        case 'B':
            animatorController.Play("IdleB");
            break;
    }
}

/// <summary>
/// Проверить, нажата ли клавиша для атаки.
```

```
/// </summary>
private void CheckAttack()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        SetMine();
    }
}

/// <summary>
/// Установить мину.
/// </summary>
private void SetMine()
{
    if (numberOfMines > 0)
    {
        CurrentTimeOfMineSetting = TimeOfMineSetting;
        animatorController.Play("Mine" + playerDirection);
        Instantiate(mine, transform.position + new Vector3(0, -0.5f, 0), transform.rotation);
        numberOfMines--;
    }
}

/// <summary>
/// Поднять мину.
/// </summary>
public void GetMine()
{
    CurrentTimeOfMineSetting = TimeOfMineSetting;
    animatorController.Play("Mine" + playerDirection);
    if (numberOfMines < maxNumOfMines)
        numberOfMines++;
}

/// <summary>
/// Поднять бонус.
/// </summary>
/// <param name="type"> Тип поднятого бонуса. </param>
public void GetItem(int type)
{
    switch (type)
    {
        case 0: //жизнь+
            if (currHealthPoints + 2 <= healthPoints)
            {
                currHealthPoints+=2;
                eventText.text = "+2 жизни";
            }
            else if (currHealthPoints < healthPoints)
            {
                currHealthPoints = healthPoints;
                eventText.text = "+2 жизни";
            }
            break;
        case 1: //защита+
            if (defence < 40)
```

```
            {
                defence += 5;
                eventText.text = "+5 защиты";
            }
            break;
        case 2: //атака+
            if (attackForce < 200)
            {
                attackForce += 5;
                eventText.text = "+5 атаки";
            }
            break;
        case 3: //жизнь-
            if (currHealthPoints > 1)
            {
                currHealthPoints--;
                eventText.text = "-1 жизнь";
            }
            break;
        case 4: //защита-
            if (defence > 0)
            {
                defence-=5;
                eventText.text = "-5 защиты";
            }
            break;
        case 5: //атака-
            if (attackForce > 50)
            {
                attackForce-=5;
                eventText.text = "-5 атаки";
            }
            break;
        case 6: //монеты
            money += 50;
            eventText.text = "+50 очков";
            break;
    }
}

/// <summary>
/// Добавить очки к рекорду.
/// </summary>
/// <param name="points"> Количество добавляемых очков. </param>
public void AddPointsToScore(int points)
{
    money += points;
    eventText.text = "+" + points.ToString() + " очков";
}

/// <summary>
/// Убить всех противников.
/// </summary>
public static void KillThemAll()
{
    GameObject[] gos = GameObject.FindGameObjectsWithTag("Enemy");
```

```
        foreach (var go in gos)
        {
            Destroy(go);
        }
    }

    /// <summary>
    /// Восстановление количество мин и здоровья.
    /// </summary>
    /// <returns></returns>
    IEnumerator MinesAndHealthRegen()
    {
        while (currHealthPoints >= 0)
        {
            for (int i = 0; i < 5; i++)
            {
                if (numberOfMines < maxNumOfMines)
                    numberOfMines++;
                yield return new WaitForSeconds(2f);
            }

            if (currHealthPoints + 1 <= healthPoints)
                currHealthPoints++;
            else if (currHealthPoints < healthPoints)
                currHealthPoints = healthPoints;
        }
    }
}
```

1.20 Класс SceneLoader.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

/// <summary>
/// Класс, отвечающий за смену сцен.
/// </summary>
public class SceneLoader : MonoBehaviour
{
    /// <summary>
    /// Дезактивация копки "Продолжить" в случае, если не была начата игра.
    /// </summary>
    private void Start()
    {
        if (SceneManager.GetActiveScene().buildIndex == 0)
        {
            if (!PlayerPrefs.HasKey("Level"))
            {
                GameObject.Find("ContinueButton").SetActive(false);
            }
        }
    }

    /// <summary>
    /// Загрузка сцены.
    /// </summary>
    /// <param name="levelNum"> Индекс сцены, которая будет загружена. </param>
    public void LoadLevel(int levelNum)
    {
        int n = SceneManager.sceneCountInBuildSettings;
```

```
        if (levelNum < n)
            SceneManager.LoadScene(levelNum);
        else
            GameObject.Find("LevelInformer").GetComponent<LevelInformator>().Win();
    }

    /// <summary>
    /// Перезапуск уровня.
    /// </summary>
    public void ReloadLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    /// <summary>
    /// Продолжить игру с последнего открытого уровня.
    /// </summary>
    public void Continue()
    {
        if (PlayerPrefs.HasKey("Level"))
        {
            SceneManager.LoadScene(PlayerPrefs.GetInt("Level"));
        }
    }

     /// <summary>
     /// Начать игру заново. Сброс всех сохранений.
     /// </summary>
    public void NewGame()
    {
        float a = -1;
        if (PlayerPrefs.HasKey("Volume"))
        {
            a = PlayerPrefs.GetFloat("Volume");
        }
        PlayerPrefs.DeleteAll();
        if (a != -1)
        {
            PlayerPrefs.SetFloat("Volume", a);
        }
        SceneManager.LoadScene(1);
    }

    /// <summary>
    /// Выйти из игры.
    /// </summary>
    public void QuitGame()
    {
        Application.Quit();
    }

    /// <summary>
    /// Загрузка следующего уровня при входе игрока в область объекта. Сохранение прогресса.
    /// </summary>
    /// <param name="collision"> Объект, вошечший в область. </param>
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "Player")
        {
            PlayerController pl =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
```

```
            PlayerPrefs.SetFloat("Health", pl.currHealthPoints);
            PlayerPrefs.SetInt("Mines", pl.numberOfMines);
            PlayerPrefs.SetInt("Defence", pl.defence);
            PlayerPrefs.SetInt("Attack", pl.attackForce);

            LoadLevel(SceneManager.GetActiveScene().buildIndex + 1);
        }
    }
}
```

1.21 Класс ShowHistory.cs

```
using System.Collections;
using UnityEngine;

/// <summary>
/// Класс, показывающий историю, когда начинается новая игра.
/// </summary>
public class ShowHistory : MonoBehaviour
{
    /// <summary>
    /// Массив картинок, включенные в историю.
    /// </summary>
    [SerializeField] Sprite[] sprites = null;

    /// <summary>
    /// Начался ли показ истории.
    /// </summary>
    bool hasStarted = false;

    /// <summary>
    /// Компонента SpriteRenderer
    /// </summary>
    SpriteRenderer sr;

    /// <summary>
    /// Объект, отвечающий за загрузку сцен.
    /// </summary>
    SceneLoader sl;

    /// <summary>
    /// Установка начальных значений, поиск компонент.
    /// </summary>
    void Start()
    {
        sr = gameObject.GetComponent<SpriteRenderer>();
        sr.color = Color.grey;
        sl = GameObject.Find("SceneLoader").GetComponent<SceneLoader>();
    }

    /// <summary>
    /// Проверка на прерывание показа истории.
    /// </summary>
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape) && hasStarted)
        {
            hasStarted = false;
            sl.NewGame();
        }
    }
```

```
    /// <summary>
    /// Начать показ истории.
    /// </summary>
    public void StartCor()
    {
        StartCoroutine("Show");
    }

    /// <summary>
    /// Реализация показа истории.
    /// </summary>
    /// <returns> Промежутки между сменами картинок. </returns>
    public IEnumerator Show()
    {
        hasStarted = true;
        sr.color = Color.white;
        foreach (Sprite sp in sprites)
        {
            sr.sprite = sp;
            yield return new WaitForSeconds(3f);
        }
        hasStarted = false;
        sl.NewGame();
    }
}
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.01-01 12				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата