МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

КАФЕДРА СИСТЕМ АВТОМАТИЗАЦИИ УПРАВЛЕНИЯ

Отчет по дисциплине

«Научно-исследовательская работа»

Выполнил: _____ /Лебедева А.А./
Студент гр. ПИб-3301-01-00 (дата, подпись)

Проверил: _____ /Земцов М.А. /
Преподаватель кафедры САУ (дата, подпись)

Киров 2018

1. Обозначение проблемы

Игровая индустрия, как и любая другая индустрия развлечений растет с каждым годом благодаря техническому прогрессу. У разработчиков появляется все больше новых возможностей. Те идеи, что раньше не могли быть реализованы становятся доступны и осуществимы. Каждый год в свет выходят все новые и новые игры. Может и похожие, но и абсолютно разные, уникальные. Хотя большинство игр направлено на развлечение игроков, но они не представляют собой исключительно развлекательное шоу. Они рассказывают историю, учат тебя новому, тому, чего ты не знал, заставляют думать не линейно, проявлять фантазию в решении вопросов, сопереживать, чувствовать радость победы, стремиться к новым рекордам. Это не просто набор скриптов и анимаций, объединенный между собой и приправленный красивой картинкой, а целый мир, придуманный и реализованный сотнями людей, вложивших свои силы в этот проект.

Разработка игр долгий и трудоемкий процесс. Он подразумевает под собой не только владение языками программирования, но и умение пользоваться средствами их разработки. Определение платформы, игрового движка, оптимизация кода и создание понятной игровой логики и интерфейса не менее важно, чем разработка концепции игры, написание сюжета, проработка персонажей.

Так как я хочу создать игру, но не имею необходимых для этого навыков *основной проблемой* является отсутствие необходимых навыков разработки игр. Решением будет получение этих навыков с дальнейшим использованием в проекте.

2. Методы решения проблемы

Я выделила несколько вариантов решения проблемы:

- Найти соответствующие курсы и записаться на них. Однако в Кирове курсов с данной тематикой не проводят, а ехать в другой город возможности нет. Онлайн курсы не подходят из-за дороговизны.
- Изучать разработку самостоятельно на примере чего-либо. В данной ситуации могут возникнуть проблемы с пониманием материала. Отсутствие нужной информации. Так же весь материал придется искать самостоятельно.
- Попросить знакомых помощи с изучением. Но среди моего окружения нет людей с подобными знаниями.

Самый подходящий вариант для меня *второй*, так как в моем случае он является единственно возможным.

3. Реализация выбранного пути решения

Следующим пунктом после выбора пути решения стал выбор того, с чем я буду работать. Игровой движок – важная составляющая любой игры. Без него игры не будет.

Существует множество различных игровых движков, позволяющих создавать игры. Все они отличаются по своему функционалу, возможностям и сложности. Мировыми фаворитами являются Unity (на нем сделаны Hearthstone, Firewatch, Syberia 3), Unreal Engine (Fortnite, Dead by Daylight), CryEngine, Corona, Phaser и другие.

Unity универсален, относительно прост в изучении, поддерживает кроссплатформенность и является бесплатным, поэтому выбор пал на него.

Так как основной задачей моего проекта было изучение работы с Unity, я решила не просто читать соответствующую литературу, а создать простую и небольшую игру, чтобы отследить как работают отдельные части проекта и увидеть конечный результат проделанной работы.

Моей идеей была классическая «endless runner» игра. Ведьма летит на метле по платформам и собирает монетки. (рис.1) За время полета игрок набирает баллы, чем дальше он летит и чем больше монеток соберет, тем выше результат. На пути игрока будут встречаться препятствия в виде шипов. При падении с платформы и столкновении с шипами персонаж умирает (рис.2).

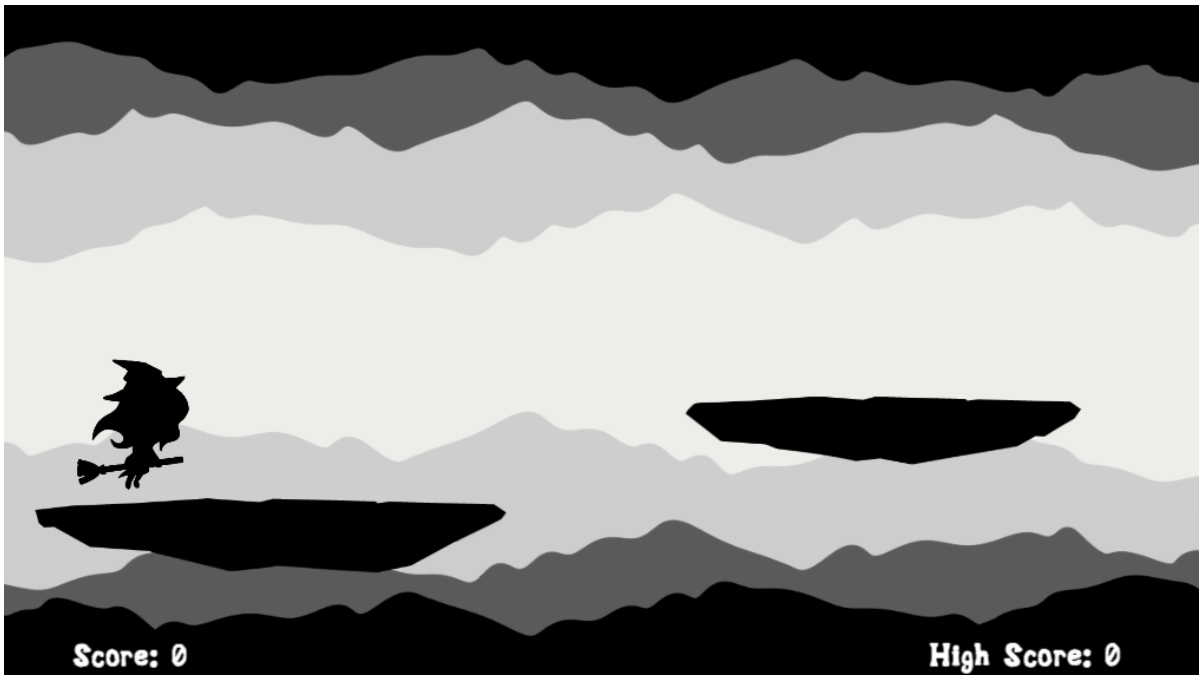

Рисунок 1

Рисунок 2

Следующим шагом стала разработка персонажа и окружения. Для упрощения и ускорения работы я предпочла использовать минималистический стиль и использовать только черный, белый и серый цвета. На Рис.3 представлены концепт персонажа, платформ, шипов и заднего окружения.

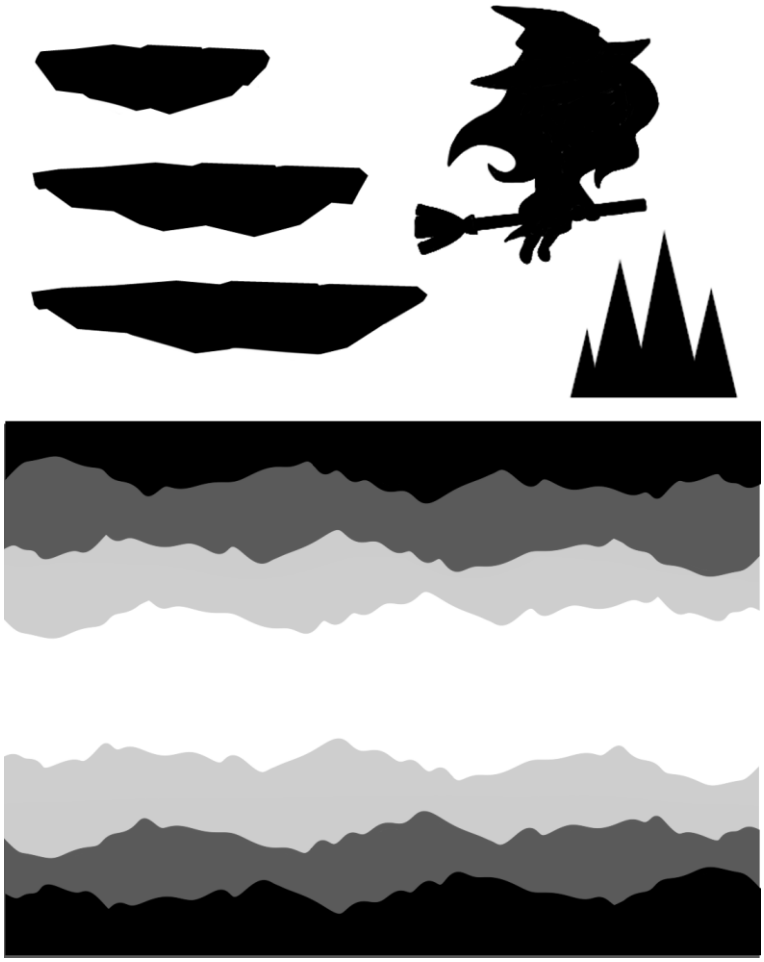

Рисунок 3

Далее началась работа с Unity. В начале я изучила работу основных вкладок. Таким образом в вкладке Hierarchy находится иерархия (перечень) всех обьектов данной сцены, Game показывает как будет выглядеть игра для игрока, Scene показывает среду разработки и как все это видит разработчик, Project содержит перечень всех элементов используемых проекте начиная от спрайтов и заканчивая скриптами, Inspector отражает свойства и функции обьекта разработки. Существуют и другие вкладки, однако эти являются основными и работать без них невозможно.

Разработку игры я начала с размещения начальных платформ на сцене, им были присвоены свойства BoxCollider2D для определения их как физических обьектов (в моем случае это было сделано чтобы платформа воспринималась как «земля» и персонаж не мог пролетать сквозь нее). Всего было создано 3 различных по длине платформы, они были добавлены в папку Prefabs (в данной папке находятся прототипы генерируемых обьектов)

Далее был создан обьект игрок и добавлена покадровая анимация для него (Рис.4) и начальный вариант скрипта игрока PlayerController. Позже были добавлены возможность двойного прыжка, изменение скорости перемещения, ограничения в прыжках, определение наличия «земли под ногами» (для возможности прыжка только с платформы, а не в воздухе), а так же возможность прыжка как по клавише пробел, так и по ЛКМ (Приложение 1.1)

Был создан скрипт управления камерой CameraController (Приложение 1.2). Среди обьектов находится игрок и его позиция присваивается камере. Таким образом при передвижении игрока камера следует за ним.

Для автоматизации процесса создания платформ была создана точка генерации платформ и для нее создан скрипт PlatformGenerator, который в дальнейшем также использовался для генерации монет и шипов (Приложение 1.3). Скрипт определяет максимальную и минимальную высоту генерации платформ исходя из полученных данных об их ширине (значении y), так же он определяет длину этих платформ для избегания наложения их друг на друга и случайное расстояние между платформами. Для генерации монет он определяет середину платформы, для шипов случайную позицию на

платформе с заданным расстоянием до конца платформы (чтобы избежать «свисания» шипов).

Чтобы избежать нагрузки на память устройства была создана точка удаления (деактивации) платформ и скрипт PlatformDestroyer (Приложение 1.4). Таким образом после того, как обьект покинет предел видимости игрока он будет деактивирован.

Так же чтобы не нагружать процесс создания обьектов был создан скрипт ObjectPooler (Приложение 1.5) определяющий обьекты генерации и устанавливающий предел генерируемых обьектов (например, по 3 на каждую платформу). Для данного скрипта было создано три точки генерации платформ (по 1 на каждый вид платформы) Таким образом платформа активируется, проходит игровую зону, деактивируется и активируется вновь, когда стает нужна. Для монет (их предел 10) и шипов (предел 5) алгоритм тот же.

Для создания ряда из трех монет создан специальный обьект и скрипт CoinGenerator (Приложение 1.6). который забирает монету из массива (из 10 монет) и генерирует ее (после чего PlatformGenerator ставит ее на платформу) и определяет расстояние между монетками.

В игре реализована система набора очков. Был создан обьект ScoreManager (Приложение 1.7) и скрипт для него названный так же. В скрипте устанавливается кол-во баллов за пройденную секунду, производится подсчет этих баллов, есть условие добавления баллов (при проигрыше счетчик останавливается) и условия превышения текущего счета игрока, предыдущим высшим счетом (то есть текущий счет стает новым рекордом). Также реализовано запоминание счета. Поэтому при выходе из игры рекорд будет оставаться.

Скрипт PickUpPoints (Приложение 1.8) определяет достиг ли игрок монетки. Если да, то обьект деактивируется, а игроку добавляются дополнительные очки.

Обьект и скрипт GameManager (Приложение 1.9) определяет начальную точку игрока, управляет остановкой и запуском счетчика баллов, остановкой и запуском музыки, запуском экрана проигрыша. Так же данный скрипт позволяет стереть все обьекты предыдущего прохождения (то есть если игрок проиграл, то при рестарте игры счетчик баллов обнулится и запустится снова, игрок вернется в начало, а платформы (кроме 2 начальных), монетки и шипы предыдущего прохождения будут удалены)

Так же был создан обьект DeathPlatform он так же движется вместе с камерой (под всеми платформами внизу) и игроком и находится вне зоны видимости игрока. При столкновении с ним игра заканчивается.

При проигрыше на экран выводится обьект DeathMenu, содержащий кнопки Restart и Main Menu. Обьект управляется скриптом DeathMenu Приложение 1.10)

Обьект SoundController запускает музыку во время игры. При проигрыше она останавливается и начинается заново при запуске игры.

Было создано UI то есть интерфейс. Во время игры в нижней части экрана высвечивается Score и High Score игрока. Эти значения изменяет скрипт ScoreManager.

В последнюю очередь было создано главное меню игры. Это отдельная сцена, которая устанавливается (в настройках проекта Unity) начальной и ее видит игрок при запуске. Она представлена названием игры, двумя кнопками Play и Exit и версией игры на сером фоне. Кнопка Play запускает следующую по счету сцену, то есть непосредственно игру, Exit закрывает приложение (Рис.4).

4. Вывод и итоги

В конечном итоге игра была создана и протестирована как на компьютере, так на телефоне. В обоих случаях она работала. На скриншотах (Приложение 2) представлена среда разработки.

Создание данного проекта позволило мне не только узнать о том, как пишется код и создаются игры на Unity, но и освоить создание на собственном примере. Я обучилась основам реализации игр и их оптимизации. Это позволит мне изучать дальнейший более сложный материал и создавать собственные игры.

Приложения

Приложение 1.1

```
public class PlayerController : MonoBehaviour {

    public float moveSpeed;
    private float moveSpeedStore;
    public float speedMultiplier;
    public float speedIncreaseMilestone;
    private float speedIncreaseMilestoneStore;
    private float speedMilestoneCount;
    private float speedMilestoneCountStore;
    public float jumpForce;
    public float jumpTime;
    private float jumpTimeCounter;
    public bool grounded;
    public LayerMask whatIsGround;
    public Transform groundCheck;
    public float groundCheckRadius;
    private bool stoppedJumpig;
    private Rigidbody2D rb;
    public GameManager theGameManager;
    private bool canDoubleJump;

       void Start ()
    {
        rb = GetComponent<Rigidbody2D>();
        jumpTimeCounter = jumpTime;
        speedMilestoneCount = speedIncreaseMilestone;
        moveSpeedStore = moveSpeed;
        speedMilestoneCountStore = speedMilestoneCount;
        speedIncreaseMilestoneStore = speedIncreaseMilestone;
        stoppedJumpig = true;

     }

       void Update ()
    {
        grounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius,
whatIsGround);

        if (transform.position.x>speedMilestoneCount)
        {
            speedMilestoneCount += speedIncreaseMilestone;
            speedIncreaseMilestone = speedIncreaseMilestone * speedMultiplier;
            moveSpeed = moveSpeed * speedMultiplier;
        }

        rb.velocity = new Vector2(moveSpeed, rb.velocity.y);

        if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0) )
        {
            if (grounded)
            {
                rb.velocity = new Vector2(rb.velocity.x, jumpForce);
                stoppedJumpig = false;
            }
            if(!grounded &&canDoubleJump)
            {
                rb.velocity = new Vector2(rb.velocity.x, jumpForce);
                jumpTimeCounter = jumpTime;
                stoppedJumpig = false;
```

```
                canDoubleJump = false;
            }
        }
        if ((Input.GetKey (KeyCode.Space) || Input.GetMouseButton(0)) && !stoppedJumpig)
        {
            if (jumpTimeCounter > 0)
            {
                rb.velocity = new Vector2(rb.velocity.x, jumpForce);
                jumpTimeCounter -= Time.deltaTime;
            }
        }
       if(Input.GetKeyUp(KeyCode.Space) || Input.GetMouseButtonUp(0))
        {
            jumpTimeCounter = 0;
            stoppedJumpig = true;
        }
        if (grounded)
        {
            jumpTimeCounter = jumpTime;
            canDoubleJump = true;
        }
       }

    void OnCollisionEnter2D (Collision2D other)
    {
        if(other.gameObject.tag =="DeathBox")
        {
            theGameManager.RestartGame();
            moveSpeed = moveSpeedStore;
            speedMilestoneCount = speedMilestoneCountStore;
            speedIncreaseMilestone = speedIncreaseMilestoneStore;
        }
    }

       }
```

Приложение 1.2

```
public class CameraController : MonoBehaviour {

    public PlayerController thePlayer;
    private Vector3 lastPlayerPosition;
    private float distanceToMove;


       void Start () {
        thePlayer = FindObjectOfType<PlayerController>();

        lastPlayerPosition = thePlayer.transform.position;
    }

       void Update () {
        distanceToMove = thePlayer.transform.position.x - lastPlayerPosition.x;

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

        lastPlayerPosition = thePlayer.transform.position;
       }
       }
```

Приложение 1.3

```
public class PlatformGenerator : MonoBehaviour {

    public GameObject thePlatform;
    public Transform generationPoint;
    public float distanceBetween;
    private float platformWidth;
    public float distanceBetweenMin;
    public float distanceBetweenMax;
    private int platformSelector;
    private float[] platformWidths;
    public ObjectPooler[] theObjectPools;
    private float minHeight;
    public Transform maxHeightPoint;
    private float maxHeight;
    public float maxHeightChange;
    private float heightChange;

    private CoinGenerator theCoinGenerator;
    public float randomCoinThreshold;

    public float randomSpikeThreshold;
    public ObjectPooler SpikePool;

    void Start ()
    {
        platformWidths = new float[theObjectPools.Length];
        for (int i = 0; i< theObjectPools.Length; i++)
        {
            platformWidths[i] =
theObjectPools[i].pooledObject.GetComponent<BoxCollider2D>().size.x;
        }
        minHeight = transform.position.y;
        maxHeight = maxHeightPoint.position.y;
        theCoinGenerator = FindObjectOfType<CoinGenerator>();
       }

       void Update ()
    {
             if(transform.position.x<generationPoint.position.x)
        {
            distanceBetween = Random.Range(distanceBetweenMin, distanceBetweenMax);
            platformSelector = Random.Range(0,theObjectPools.Length);
            heightChange = transform.position.y + Random.Range(maxHeightChange, -
maxHeightChange);

            if(heightChange > maxHeight)
            {
                heightChange = maxHeight;
            }
            else if (heightChange<minHeight)
            {
                heightChange = minHeight;
            }
            transform.position = new Vector3(transform.position.x +
(platformWidths[platformSelector] / 2) + distanceBetween, heightChange,
transform.position.z);

            GameObject newPlatform = theObjectPools[platformSelector].GetPooledObject();
            newPlatform.transform.position = transform.position;
            newPlatform.transform.rotation = transform.rotation;
            newPlatform.SetActive(true);

            if (Random.Range(0f, 100f) < randomCoinThreshold)
            {
```

```
                theCoinGenerator.SpawnCoins(new Vector3(transform.position.x,
transform.position.y + 1f, transform.position.z));
            }
            if (Random.Range(0f, 100f) < randomSpikeThreshold)
            {
                GameObject newSpike = SpikePool.GetPooledObject();

                float spikeXPosition = Random.Range(-platformWidths[platformSelector] /
2f +1f, platformWidths[platformSelector] / 2f -1f);
                Vector3 spikePosition = new Vector3(spikeXPosition, 0.3f, 0f);

                newSpike.transform.position = transform.position +spikePosition;
                newSpike.transform.rotation = transform.rotation;
                newSpike.SetActive(true);
            }
                transform.position = new Vector3(transform.position.x +
(platformWidths[platformSelector] / 2), transform.position.y, transform.position.z);
        }
    }
}
```

Приложение 1.4

```
public class PlatformDestroyer : MonoBehaviour {

    public GameObject platformDestructionPoint;

       void Start () {
        platformDestructionPoint = GameObject.Find("PlatformDestructionPoint");
       }

       void Update () {
              if (transform.position.x< platformDestructionPoint.transform.position.x)
        {
            gameObject.SetActive(false);

        }
       }
       }
```

Приложение 1.5

```
public class ObjectPooler : MonoBehaviour {

    public GameObject pooledObject;
    public int pooledAmount;

    List<GameObject> pooledObjects;

       void Start () {
        pooledObjects = new List<GameObject>();

        for (int i = 0; i < pooledAmount; i++)
        {
            GameObject obj = (GameObject)Instantiate(pooledObject);
            obj.SetActive(false);
            pooledObjects.Add(obj);
        }
       }

    public GameObject GetPooledObject()
    {
        for (int i = 0; i < pooledObjects.Count; i++)
```

```
        {
            if (!pooledObjects[i].activeInHierarchy)
            {
                return pooledObjects[i];
            }
        }
        GameObject obj = (GameObject)Instantiate(pooledObject);
        obj.SetActive(false);
        pooledObjects.Add(obj);
        return obj;
    }
    }
```

Приложение 1.6

```
public class CoinGenerator : MonoBehaviour {

    public ObjectPooler coinPool;
    public float distanceBetweenCoins;

    public void SpawnCoins(Vector3 startPosition)
    {
        GameObject coin1 = coinPool.GetPooledObject();
        coin1.transform.position = startPosition;
        coin1.SetActive(true);

        GameObject coin2 = coinPool.GetPooledObject();
        coin2.transform.position = new Vector3(startPosition.x-
distanceBetweenCoins,startPosition.y,startPosition.z);
        coin2.SetActive(true);

        GameObject coin3 = coinPool.GetPooledObject();
        coin3.transform.position = new Vector3(startPosition.x + distanceBetweenCoins,
startPosition.y, startPosition.z);
        coin3.SetActive(true);
    }

    }
```

Приложение 1.7

```
public class ScoreManager : MonoBehaviour {

    public Text scoreText;
    public Text highScoreText;
    public float scoreCount;
    public float highScoreCount;
    public float pointsPerSecond;
    public bool scoreIncreasing;

    void Start () {
          if (PlayerPrefs.HasKey("HighScore"))
        {
            highScoreCount = PlayerPrefs.GetFloat("HighScore");
        }
    }

    void Update () {
        if (scoreIncreasing)
        {
            scoreCount += pointsPerSecond * Time.deltaTime;
        }

        if (scoreCount > highScoreCount)
```

```
        {
            highScoreCount = scoreCount;
            PlayerPrefs.SetFloat("HighScore", highScoreCount);
        }
        scoreText.text = "Score: " + Mathf.Round(scoreCount);
        highScoreText.text = "High Score: " + Mathf.Round(highScoreCount);
    }
    public void AddScore (int pointsToAdd)
    {
        scoreCount += pointsToAdd;
    }
    }
```

Приложение 1.8

```
public class pickupPoints : MonoBehaviour {

    public int scoreToGive;

    private ScoreManager theScoreManager;

    void Start () {
     theScoreManager = FindObjectOfType<ScoreManager>();
    }

    void Update () {

    }
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.name=="Player")
        {
            theScoreManager.AddScore(scoreToGive);
            gameObject.SetActive(false);
        }
    }

    }
```

Приложение 1.9

```
public class GameManager : MonoBehaviour {

    public Transform platformGenerator;
    private Vector3 platformStartPoint;
    public PlayerController thePlayer;
    private Vector3 playerStartPoint;
    private PlatformDestroyer[] platformList;
    private ScoreManager theScoreManager;
    private GameObject Music;
    public DeathMenu theDeathScreen;
    void Start ()
    {
        platformStartPoint = platformGenerator.position;
        playerStartPoint = thePlayer.transform.position;

        theScoreManager = FindObjectOfType<ScoreManager>();
        Music = GameObject.Find("SoundController");
    }

    void Update ()
    {

    }
```

```
    public void RestartGame()
    {
        theScoreManager.scoreIncreasing = false;
        thePlayer.gameObject.SetActive(false);

        theDeathScreen.gameObject.SetActive(true);
        Music.gameObject.SetActive(false);

    }
    public void Reset()
    {
        theDeathScreen.gameObject.SetActive(false);
        Music.gameObject.SetActive(true);
        platformList = FindObjectsOfType<PlatformDestroyer>();
        for (int i = 0; i < platformList.Length; i++)
        {
            platformList[i].gameObject.SetActive(false);
        }
        thePlayer.transform.position = playerStartPoint;
        platformGenerator.position = platformStartPoint;
        thePlayer.gameObject.SetActive(true);

        theScoreManager.scoreCount = 0;
        theScoreManager.scoreIncreasing = true;
    }
```

Приложение 1.10

```
public class DeathMenu : MonoBehaviour {

    public string mainMenuLevel;

    public void RestartGame()
    {
        FindObjectOfType<GameManager>().Reset();
    }

    public void QuitToMain()
    {
        Application.LoadLevel(mainMenuLevel);
    }
   }
```

Приложение 2 – Скриншоты среды разработки