УДК 004.946

Разработка 2D игры «Космический бой» в системе трехмерного моделирования Unity3D

Ковалева Ирина Валерьевна
Приамурский государственный университет им. Шолом-Алейхема
студент

Баженов Руслан Иванович
Приамурский государственный университет имени Шолом-Алейхема
к.п.н., доцент, зав. кафедрой информационных систем, математики и методик обучения

Аннотация
В статье рассматривается создание простейшей двухмерной игры в системе трехмерного моделирования Unity3D. В данной работе описано создание игровой сцены, подробно расписаны программные коды для управления игрой. Для иллюстрации получившихся результатов, представлены скриншоты работы.
Ключевые слова: Unity3D, 2D игра, игровой движок, скрипт

Development of 2D game «Space Battle» in Unity3D three-dimensional modeling system

Kovaleva Irina Valerievna
Sholom-Aleichem Priamursky State University
student

Bazhenov Ruslan Ivanovich
Sholom-Aleichem Priamursky State University
Candidate of pedagogical sciences, associate professor, Head of the Department of Information Systems, Mathematics and teaching methods

Abstract
This article discusses the creation of the simplest two-dimensional game in Unity3D three-dimensional modeling system. In this paper we describe how to create game scenes and painted in detail the program codes to control the game. To illustrate the obtained results that the screenshots presented work.
Keywords: Unity3D, 2D games, game engine, script

В наше время, компьютерные игры все больше становятся популярными. Но есть игры, в которые можно играть только платно. Поэтому существует возможность создать свою компьютерную игру и для этого использовать платформу разработки для создания многоплатформенных 2D и 3D игр – Unity3D.

Система Unity3D становится, все больше, популярна среди ученых, а так же и среди студентов. О.С. Ходос и Р.И. Баженов применили трехмерное моделирование в Unity3D [1]. Е.И.Сальникова показала особенности разработки персонажей для двумерных компьютерных игр[2]. А.С.Винокуров и Р.И.Баженов создали проект «Танк на острове» в Unity3D[3], а так же А.А. Ковтун и Е.Ю.Тихонов разработали игровое приложение «Paperman» [4]. Есть и те, кто занимался созданием не только компьютерных игр, но и созданием мобильных игр и приложений. Например, А.С. Сеидова и В.С. Сухоплюева разработали мобильные игры с помощью UNITY3D[5]. Н.С.Галкин, Е.А. Ромин занимаются созданием трехмерной модели местности в Unity 3D [6]. M. Labschütz и др. создали контент для 3D-игры с Майа и Unity3D [7]. G.A.N. Jian-song спроектировал дизайн внутреннего основанного блуждания на Unity3D [8].

Задача создания данной игры состоит в том, чтобы научиться работать с двухмерной графикой, научиться создавать скрипты на языке C#, а так же понять, в чем состоит механизм создания 2D игр. Данная 2D игра рассчитана на детей младшего школьного возраста. Цель игры: в космосе управлять космическим кораблем и стрелять в появляющихся врагов.

Вначале следует создать новый проект и отметить на панели двухмерную игру.

После того, как открылось окно Unity, нужно создать 4 папки: Prefabs (массивы объектов), Scripts (программный код), Sounds (звуковое сопровождение игры) и Sprites (изображения игровых объектов) (рис.1).

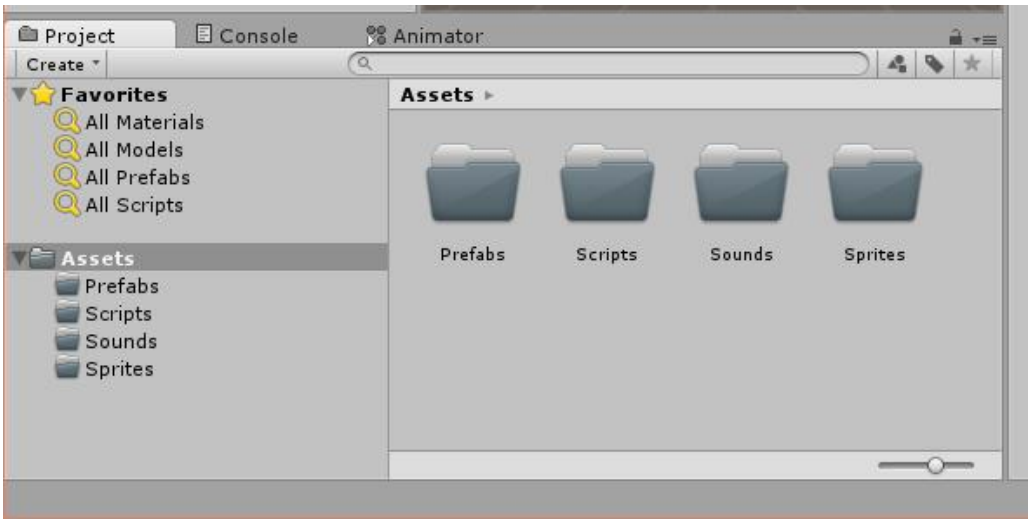


Рисунок 1 – Создание папок в окне Project

Создание игровой сцены.

Так как изображение корабля уже существует, при этом изображение должно быть в формате png, следует перетащить его в окно Unity, в папку Sprites. Затем нужно выделить изображение и в окне «Inspector» в строчке «Texture Type» выбрать значение «Sprite (2D and UI)». И после этого нажать кнопку применить (Apply). С окна Project перетащить изображение корабля в

окно Scene. В окне Inspector поменять координаты корабля и переместить его в центр сцены.

Создание фона, на котором будет находиться космический корабль.

В программе Paint нужно нарисовать звездное небо, размер изображения 100x100, сохранить его в формате png. Все те же действия с изображением корабля сделать и с изображение неба, координаты фона следует поменять так, чтобы фон находился за кораблем. Так как фон нельзя растянуть, поэтому со сцены нужно удалить фон и добавить куб, растянуть его до размера игровой сцены. Придать ему текстуру нашего фона. (рис. 2)

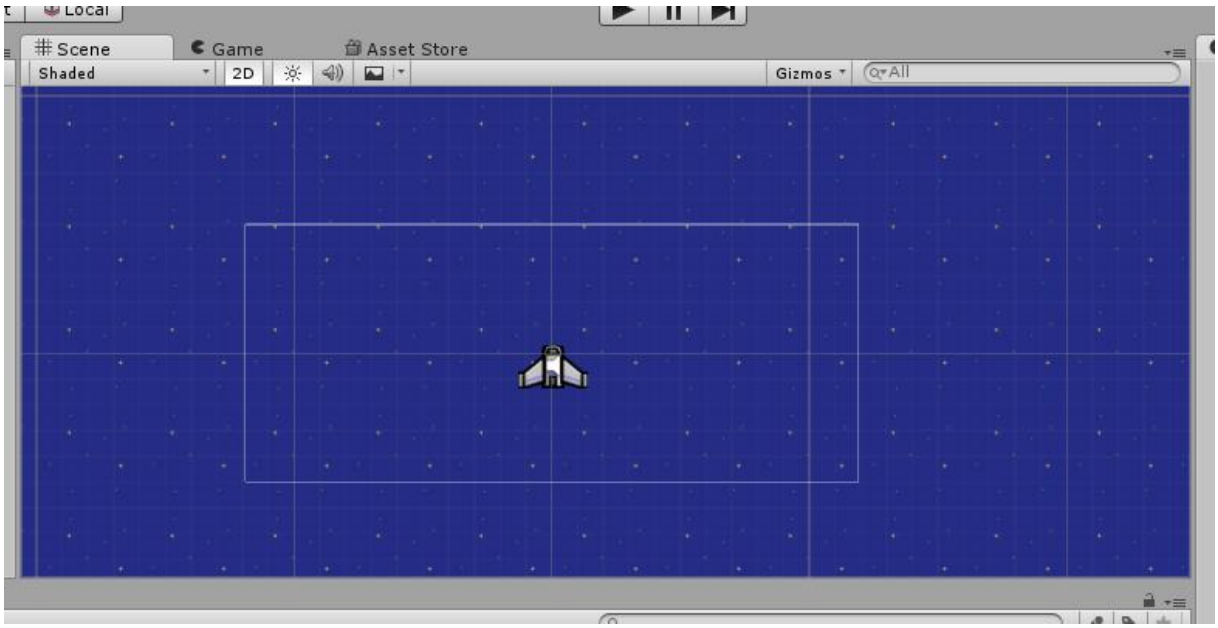


Рисунок 2 – Создание игровой сцены

Для того чтобы кораблем можно было управлять, следует создать Script и привязать его к кораблю. (рис. 3, а,б)

PlayerScript ▸ No selection

```
1  using UnityEngine;
2  using System.Collections.Generic;
3
4  public class PlayerScript : MonoBehaviour
5  {
6
7      // Изменение скорости перемещения героя
8      public float playerSpeed = 2.0f;// Текущая скорость перемещения
9  private float currentSpeed = 0.0f;
10
11 // Создание переменных для кнопок
12 public List<KeyCode> upButton;
13 public List<KeyCode> downButton;
14 public List<KeyCode> leftButton;
15 public List<KeyCode> rightButton;
16
17 // Сохранение последнего перемещения
18 private Vector3 lastMovement = new Vector3();
19
20     // Update is called once per frame
21 void Update()
22 {
23     // Поворот героя к мышке
24     Rotation();
25     // Перемещение героя
```

Рисунок 3а – Программный код для управления кораблем

```
PlayerScript ► laser
// Поворот героя к мышке
void Rotation()
{
    // Показываем игроку, где мышка
    Vector3 worldPos = Input.mousePosition;
    worldPos = Camera.main.ScreenToWorldPoint(worldPos);
    // Сохраняем координаты указателя мыши
    float dx = this.transform.position.x - worldPos.x;
    float dy = this.transform.position.y - worldPos.y;
    // Вычисляем угол между объектами «Корабль» и «Указатель»
    float angle = Mathf.Atan2(dy, dx) * Mathf.Rad2Deg;
    // Трансформируем угол в вектор
    Quaternion rot = Quaternion.Euler(new Vector3(0, 0, angle + 90));
    // Изменяем поворот героя
    this.transform.rotation = rot;
}

// Движение героя к мышке
void Movement()
{
    // Необходимое движение
    Vector3 movement = new Vector3();
    // Проверка нажатых клавиш
    movement += MoveIfPressed(upButton, Vector3.up);
    movement += MoveIfPressed(downButton, Vector3.down);
    movement += MoveIfPressed(leftButton, Vector3.left);
    movement += MoveIfPressed(rightButton, Vector3.right);
    // Если нажато несколько кнопок, обрабатываем это
    movement.Normalize();
    // Проверка нажатия кнопки
    if (movement.magnitude > 0)
    {
        // После нажатия двигаемся в этом направлении
        currentSpeed = playerSpeed;
        // После нажатия двигаемся в этом направлении
        currentSpeed = playerSpeed;
        this.transform.Translate(movement * Time.deltaTime * playerSpeed, Space.World);
        lastMovement = movement;
    }
    else
    {
        // Если ничего не нажато
        this.transform.Translate(lastMovement * Time.deltaTime * currentSpeed, Space.World);
        // Замедление со временем
        currentSpeed *= 0.9f;
    }
}

// Возвращает движение, если нажата кнопка
Vector3 MoveIfPressed(List<KeyCode> keyList, Vector3 Movement)
{
    // Проверяем кнопки из списка
    foreach (KeyCode element in keyList)
    {
        if (Input.GetKey(element))
        {
            // Если нажато, покидаем функцию
            return Movement;
        }
    }
    // Если кнопки не нажаты, то не двигаемся
    return Vector3.zero;
}
```

Рисунок 3б – Программный код для управления кораблем

Теперь нужно добавить возможность стрелять. Для этого в скрипте необходимо прописать, в верхней части кода, где прописываются переменные, следующие строки. (рис. 4) В функции «Update()» добавить код для обработки вышепрописанных пременных: (рис.5)

```
// Переменная для лазера
public Transform laser;

// Как далеко от центра корабля будет появляется лазер
public float laserDistance = 0.2f;

// Задержка между выстрелами (кулдаун)
public float timeBetweenFires = 0.3f;

// Счетчик задержки между выстрелами
private float timeTilNextFire = 0.0f;

// Кнопка, которая используется для выстрела
public List<KeyCode> shootButton;

// Переменная для звука выстрела лазером
public AudioClip shootSound;
```

Рисунок 4 – Программный код для возможности стрелять

```
// Update is called once per frame
void Update()
{
    // Поворот героя к мышке
    Rotation();
    // Перемещение героя
    Movement();

    foreach (KeyCode element in shootButton)
    {
        if (Input.GetKey(element) && timeTilNextFire < 0)
        {
            timeTilNextFire = timeBetweenFires;
            ShootLaser();
            break;
        }
    }
    timeTilNextFire -= Time.deltaTime;

}
```

Рисунок 5 – Программный код для обработки переменных

В самом низу скрипта прописать функцию создания пули: (рис. 6)

```
// Создание лазера
void ShootLaser()
{
    // Высчитываем позицию корабля
    float posX = this.transform.position.x + (Mathf.Cos((transform.localEulerAngles.z - 90) * Mathf.Deg2Rad) * -laserDistance);
    float posY = this.transform.position.y + (Mathf.Sin((transform.localEulerAngles.z - 90) * Mathf.Deg2Rad) * -laserDistance);
    // Создаём лазер на этой позиции
    Instantiate(laser, new Vector3(posX, posY, 0), this.transform.rotation);
        // Воспроизвести звук выстрела лазером
        GetComponent<AudioSource>().PlayOneShot(shootSound);
}
}
```

Рисунок 6 – Программный код для создания пули

Создание изображение для пули. Для этого необходимо создать изображение лазера, который будет уничтожать врагов, и создать изображение врага. Файлы добавить в игровую сцену. Специально для лазера создать новый Script и прописать в коде следующие строки: (рис. 7) Перетащить скрипт в свойства к лазеру.

PlayerScript ▸ laser

```
    foreach (KeyCode element in keyList)
    {
        if (Input.GetKey(element))
        {
            // Если нажато, покидаем функцию
            return Movement;
        }
    }
    // Если кнопки не нажаты, то не двигаемся
    return Vector3.zero;
}

// Создание лазера
void ShootLaser()
{
    // Высчитываем позицию корабля
    float posX = this.transform.position.x + (Mathf.Cos((transform.localEulerAngles.z - 90) * Mathf.Deg2Rad) * -laserDistance);
    float posY = this.transform.position.y + (Mathf.Sin((transform.localEulerAngles.z - 90) * Mathf.Deg2Rad) * -laserDistance);
    // Создаём лазер на этой позиции
    Instantiate(laser, new Vector3(posX, posY, 0), this.transform.rotation);
        // Воспроизвести звук выстрела лазером
        GetComponent<AudioSource>().PlayOneShot(shootSound);
}
}
```

Рисунок 7 – Отдельный скрипт для создания функций для лазера

Отдельно следует создать новый Script для врага и в программном коде добавить врагу функции, при которых враг будет находить объект корабля и двигаться по траектории к нему (рис. 8,9). Поместить скрипт в свойства к врагу.

```
using UnityEngine;

public class EnemyScript : MonoBehaviour
{
    // Сколько раз нужно попасть во врага, чтобы уничтожить его
    public int health = 2;
    // Анимация при уничтожении объекта
    public Transform explosion;

    // Переменная для звука во время попадания лазера
    public AudioClip hitSound;
    void OnCollisionEnter2D(Collision2D theCollision)
    {
        //Проверяем коллизию с объектом типа «лазер»
        if(theCollision.gameObject.name.Contains("laser"))
        {
            LaserScript laser = theCollision.gameObject.GetComponent("LaserScript") as LaserScript;
            health -= laser.damage;
            Destroy (theCollision.gameObject);
            // Воспроизвести звук попадания выстрела
            GetComponent<AudioSource>().PlayOneShot(hitSound);

            GameController controller = GameObject.FindGameObjectWithTag("GameController").GetComponent("GameController") as GameController;
            controller.KilledEnemy();
            controller.IncreaseScore(10);
        }
        if (health <= 0)
        {

            // Срабатывает при уничтожении объекта
            if (explosion)
            {
                GameObject exploder = ((Transform)Instantiate(explosion, this.transform.position, this.transform.rotation)).gameObject;
                Destroy(exploder, 2.0f);
```

Рисунок 8 – Программный код, с помощью которого враг будет двигаться по траектории к кораблю

```
        }
        if (health <= 0)
        {

            // Срабатывает при уничтожении объекта
            if (explosion)
            {
                GameObject exploder = ((Transform)Instantiate(explosion, this.transform.position, this.transform.rotation)).gameObject;
                Destroy(exploder, 2.0f);
            }


            Destroy (this.gameObject);
        }
```

Рисунок 9 – Программный код, с помощью которого враг будет двигаться по траектории к кораблю

На рис. 10 показано, как должна выглядеть игровая сцена после выполненных всех действий.

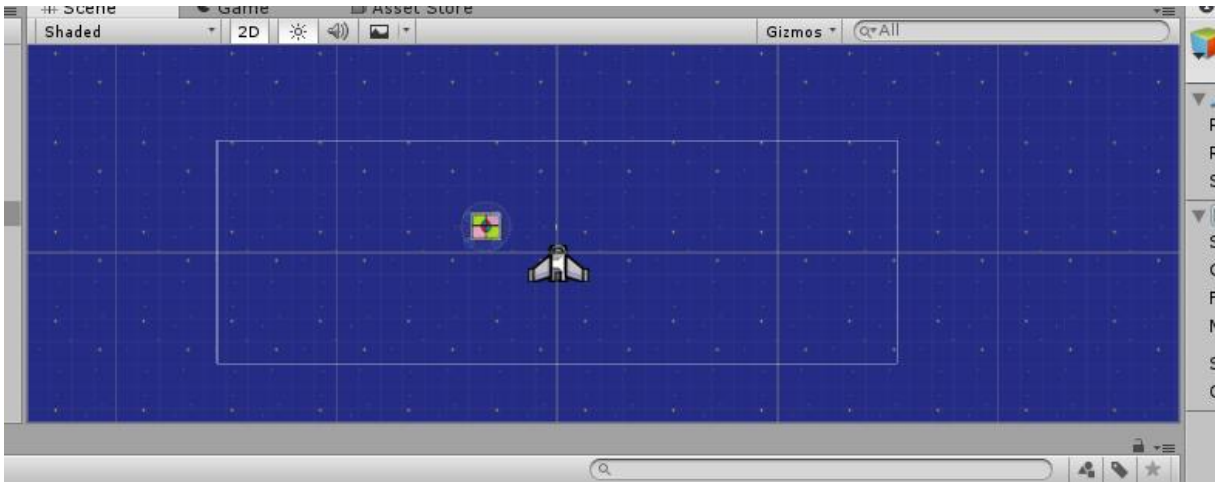
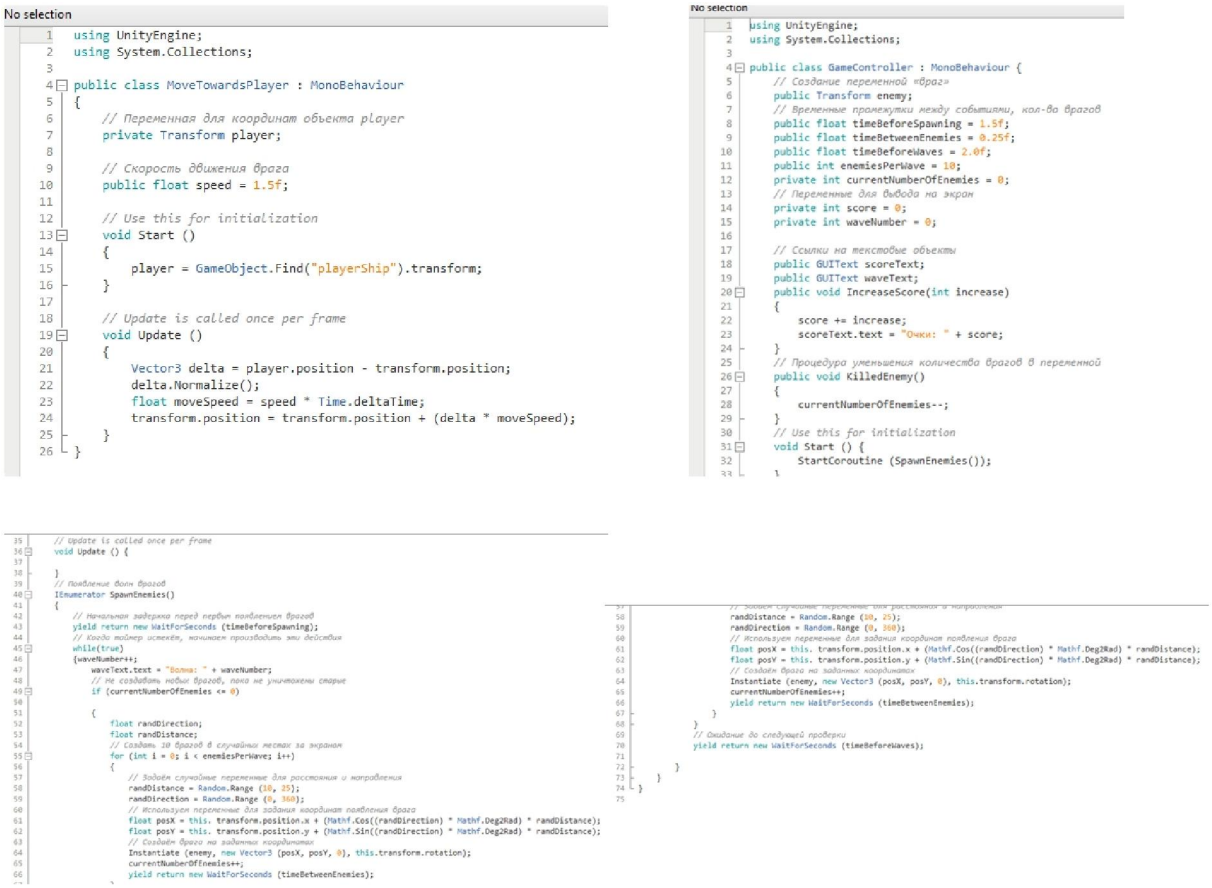


Рисунок 10 – Игровая сцена с кораблем и врагом

Для того чтобы на игровой сцене были поля, отвечающие за то, с какой скоростью, и через какой промежуток времени будут перемещаться враги, а так же отображались игровые очки, следует создать скрипт-контроллер. (рис. 11 - 14)



```
using UnityEngine;
using System.Collections;

public class MoveTowardsPlayer : MonoBehaviour
{
    // Переменная для координат объекта player
    private Transform player;

    // Скорость движения врага
    public float speed = 1.5f;

    // Use this for initialization
    void Start ()
    {
        player = GameObject.Find("playerShip").transform;
    }

    // Update is called once per frame
    void Update ()
    {
        Vector3 delta = player.position - transform.position;
        delta.Normalize();
        float moveSpeed = speed * Time.deltaTime;
        transform.position = transform.position + (delta * moveSpeed);
    }
}
```

```
using UnityEngine;
using System.Collections;

public class GameController : MonoBehaviour {
    // Создание переменной «враг»
    public Transform enemy;
    // Временные промежутки между событиями, кол-во врагов
    public float timeBeforeSpawning = 1.5f;
    public float timeBetweenEnemies = 0.25f;
    public float timeBeforeWaves = 2.0f;
    public int enemiesPerWave = 10;
    private int currentNumberOfEnemies = 0;
    // Переменные для вывода на экран
    private int score = 0;
    private int waveNumber = 0;

    // Ссылки на текстовые объекты
    public GUIText scoreText;
    public GUIText waveText;
    public void IncreaseScore(int increase)
    {
        score += increase;
        scoreText.text = "Очки: " + score;
    }
    // Процедура уменьшения количества врагов в переменной
    public void KilledEnemy()
    {
        currentNumberOfEnemies--;
    }
    // Use this for initialization
    void Start () {
        StartCoroutine (SpawnEnemies());
    }
```

```
    // Update is called once per frame
    void Update () {

    }
    // Появление волн врагов
    IEnumerator SpawnEnemies()
    {
        // Начальная задержка перед первым появлением врагов
        yield return new WaitForSeconds (timeBeforeSpawning);
        // Когда таймер истекёт, начинает производить эти действия
        while(true)
        {waveNumber++;
            waveText.text = "Волна: " + waveNumber;
            // Не создавать новых врагов, пока не уничтожены старые
            if (currentNumberOfEnemies <= 0)

            {
                float randDirection;
                float randDistance;
                // Создать 10 врагов в случайных местах за экраном
                for (int i = 0; i < enemiesPerWave; i++)
                {
                    // Задаём случайные переменные для расстояния и направления
                    randDistance = Random.Range (10, 25);
                    randDirection = Random.Range (0, 360);
                    // Используем переменные для задания координат появления врага
                    float posX = this. transform.position.x + (Mathf.Cos((randDirection) * Mathf.Deg2Rad) * randDistance);
                    float posY = this. transform.position.y + (Mathf.Sin((randDirection) * Mathf.Deg2Rad) * randDistance);
                    // Создаём врага по заданным координатам
                    Instantiate (enemy, new Vector3 (posX, posY, 0), this.transform.rotation);
                    currentNumberOfEnemies++;
                    yield return new WaitForSeconds (timeBetweenEnemies);
```

```
                    randDistance = Random.Range (10, 25);
                    randDirection = Random.Range (0, 360);
                    // Используем переменные для задания координат появления врага
                    float posX = this. transform.position.x + (Mathf.Cos((randDirection) * Mathf.Deg2Rad) * randDistance);
                    float posY = this. transform.position.y + (Mathf.Sin((randDirection) * Mathf.Deg2Rad) * randDistance);
                    // Создаём врага по заданным координатам
                    Instantiate (enemy, new Vector3 (posX, posY, 0), this.transform.rotation);
                    currentNumberOfEnemies++;
                    yield return new WaitForSeconds (timeBetweenEnemies);
                }
            }
            // Ожидание до следующей проверки
            yield return new WaitForSeconds (timeBeforeWaves);
        }
    }
}
```

Рисунок 11 – Программный код скрипт-контроллера

Чтобы украсить игру, необходимо добавить в нее спецэффектов, таких как звуковые эффекты при уничтожении врага и музыку, так же анимацию взрыва врагов. Инструмент для создания спецэффектов – это система частиц. Эта система состоит из двух типов объектов: источник частиц и сами частицы. Частицы – это небольшие объекты, которые живут в игре строго ограниченное время. Примерами частиц могут быть языки пламени в огне, клубы дыма. (Рис. 12,13)

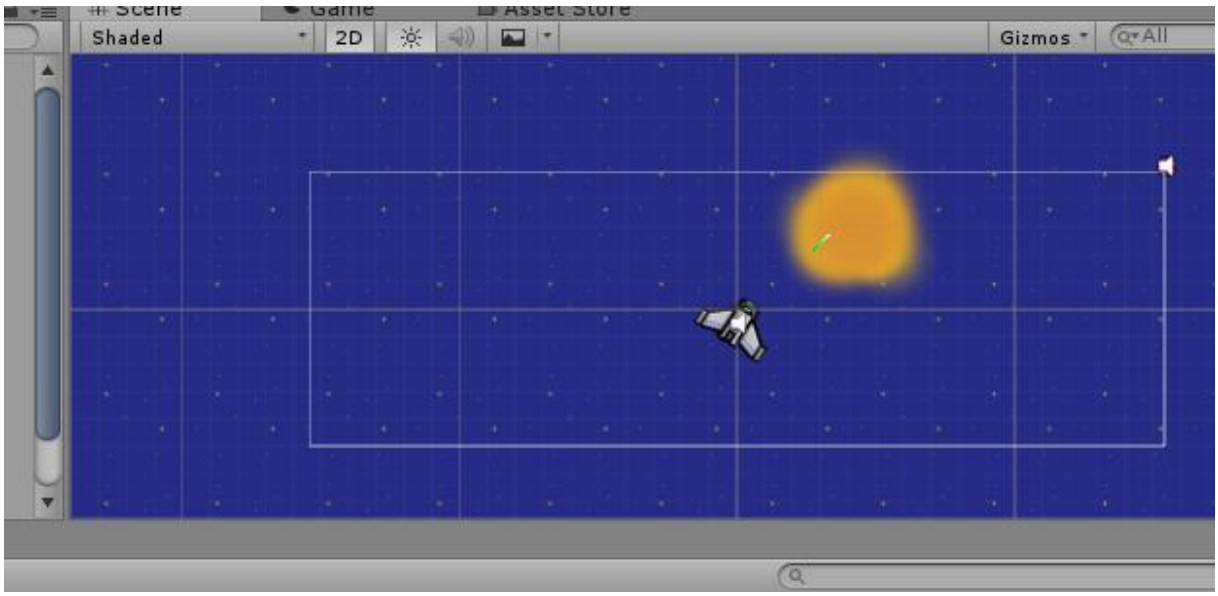


Рисунок 12 – Изображение сцены взрыва врага

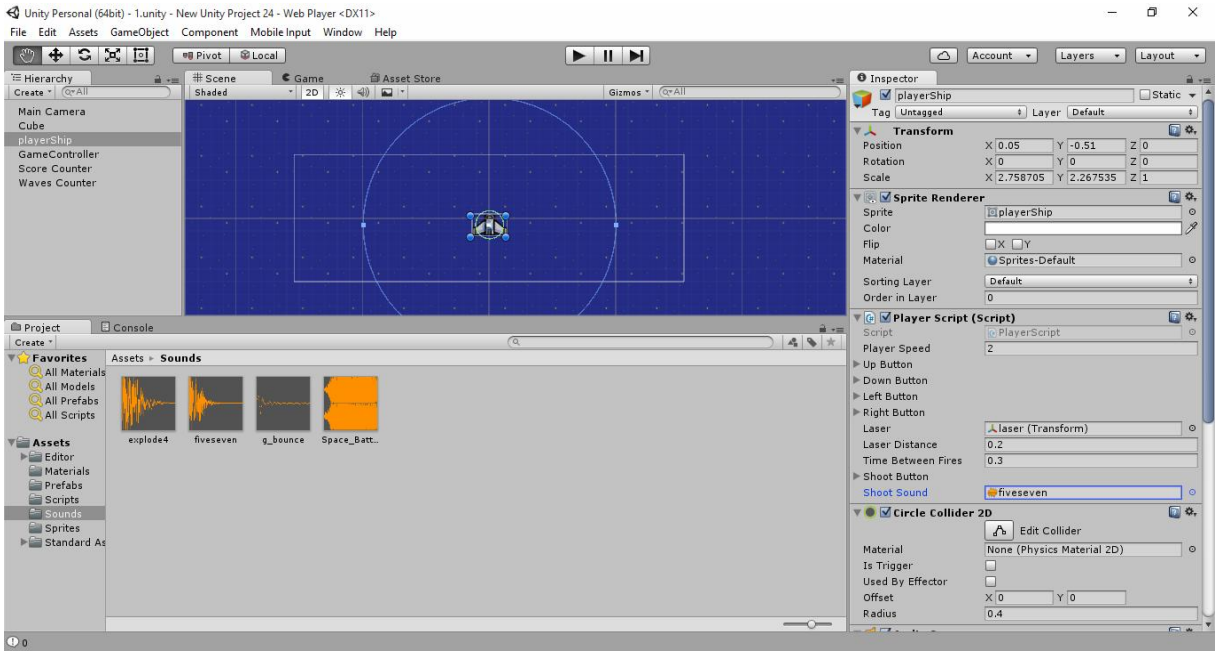


Рисунок 13 – Добавление звукового сопровождения

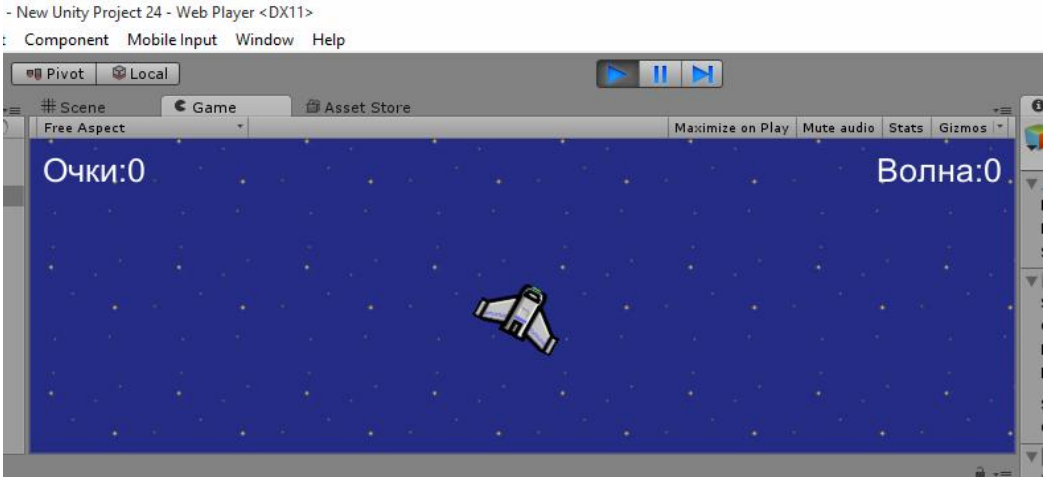


Рисунок 14 – Добавление подсчета очков и число волн приближения врагов

Сохранить созданную игр можно при нажатии на «File | Build Settings» . Затем нажать «Add current». Для начала нужно выбрать для какой платформы следует сохранить игру, поэтому нужно выбирать «Windows – x86» и нажать кнопку Build and Run. После того как пройдет компиляция, появится меню параметров для запуска игры. Выбирать развертку экрана и запустить игру, нажав «Play». Так же для игры было создано основное меню. (рис. 15)

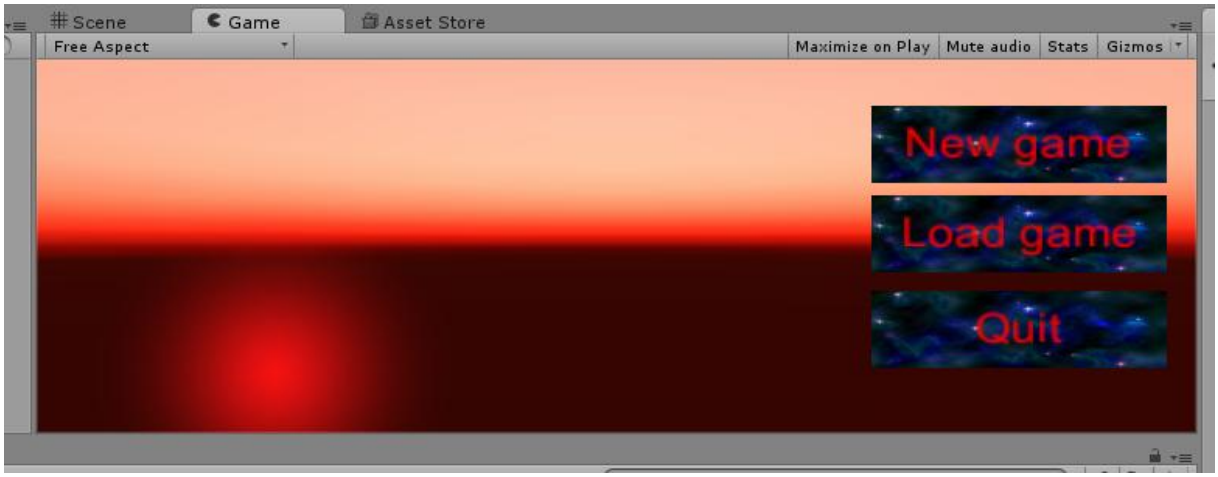


Рисунок 15 – Создание основного меню

В ходе проделанной работы была создана игра-стрелялка в виде космического корабля, который лазером уничтожает своих врагов в космосе. Игра развивает реацию и интерес уничтожить всех врагов и выиграть. Так же это хороший способ улучшить свои навыки в такой программе, как Unity 3D и наподобие создания этой игры можно создавать свои игры как с двухмерной графикой, так и с трехмерной графикой.

Библиографический список

1. Ходос О.С., Баженов Р.И. Обучение трехмерному моделированию в Unity3D// Современные научные исследования и инновации. 2014. № 6-3 (38). С. 14.
2. Сальникова Е.И. Особенности разработки персонажей для двумерных компьютерных игр.// Творчество молодых: дизайн, реклама, информационные технологии сборник трудов XIII Международной научно-практической конференции студентов и аспирантов. Научный редактор Л. М. Дмитриева. Омск, 2014. С. 129-131.
3. Винокуров А.С., Баженов Р.И. Создание проекта «танк на острове» в Unity3D// Современная техника и технологии. 2015. № 7 (47). С. 53-59.
4. Ковтун А.А., Тихонов Е.Ю. Разработка мультиплатформенного игрового приложения «Paperman»// Научные труды Кубанского государственного технологического университета. 2015. № 6. С. 296-298.
5. Сеидова А.С., Сухоплюева В.С. Разработка мобильных игр с помощью UNITY3D // Информационно-телекоммуникационные системы и технологии Материалы Всероссийской научно-практической

конференции. Кемерово, 2014. С. 294-295.
6. Галкин Н.С., Ромин Е.А. Создание трехмерной модели местности в Unity 3D // Инновационные технологии: теория, инструменты, практика. 2014. № 1. С. 311-316.
7. Labschütz M. et al. Content creation for a 3D game with Maya and Unity 3D // Institute of Computer Graphics and Algorithms, Vienna University of Technology. 2011. URL: https://www.researchgate.net/profile/Reinhold_Preiner/publication/267417785_Content_Creation_for_a_3D_Game_with_Maya_and_Unity_3D/links/554788c70cf26a7bf4d93df6.pdf (Дата обращения 16.03.2016)
8. Jian-song G. A. N. Simulating Design of Indoor Wandering Based on the Unity3d [J] //Journal of Yancheng Institute of Technology (Natural Science Edition). 2011. Т. 4. С. 015.