

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пензенский государственный университет»

Политехнический институт

(наименование института полностью)

Кафедра «Системы автоматизированного проектирования»

(наименование)

09.03.01 «Информатика и вычислительная техника»

(код и наименование направления подготовки, специальности)

«Системы автоматизированного проектирования»

(направленность (профиль) / специализация)

КУРСОВАЯ РАБОТА
по курсу «Технология разработки программного обеспечения»

на тему «Разработка 2D компьютерной игры»

Обучающийся

Калыева С.К.

(Инициалы Фамилия)

(личная подпись)

Руководитель

к.т.н, доцент каф. САПР В.В. Эпп

(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Пенза 2025

Оглавление

1 Анализ предметной области	4
1.1 Жанр игры. Аналоги игр в этом жанре	4
1.2 Описание среды разработки	5
1.3 Техническое задание на разработку	5
2 Проектирования и разработка приложения.....	7
2.1 Проектирование уровней и логики игры.....	7
2.2 Описание основных классов и скриптов	8
2.3 Ресурсы проекта	15
2.4 Основные анимации игры	15
3 Тестирование приложения	21
Заключение	31
Список использованных источников	32
Приложение	34

Введение

В курсовой работе рассмотрен процесс разработки трех 2D игр различного жанра: Платформер, Кетчер и Шутер, где каждый из этих игр имеет свои уникальные особенности. Все игры должны быть разработаны в среде Unity. Каждая из этих игр должна представлять свой графический интерфейс, меню, спрайты, анимации, условия выигрыша и проигрыша, звуковые эффекты, фоновую музыку, что обеспечит полное погружение игрока в игровой процесс.

Анализ и разработка представленных игр позволит не только углубить понимание технологий, используемых в разработке программного обеспечения, но и отработать практические навыки, необходимые для создания успешных игровых проектов. На каждом этапе работы будут рассмотрены как технические, так и творческие аспекты, что в конечном итоге приведет к созданию полноценных игр, отвечающих современным требованиям и ожиданиям игроков.

1 Анализ предметной области

1.1 Жанр игры. Аналоги игр в этом жанре

Платформер – это игра, в котором персонаж может перемещаться по уровням, состоящих из платформ, врагов и разных преград, таких как деревья. В этой игре персонаж имеет такие функции, как бег, прыжки и состояние покоя. Он может уничтожать некоторых врагов, а от некоторых может потерять жизнь.

Аналоги игр в этом жанре:

Алекс Гордон – в этой игре у главного персонажа имеются такие функции, как на бег, прыжки на платформах разных уровней. Главной целью игры является собирать монеты, бежать от врагов, уничтожать врагов.

Кетчер - игра в жанре ловли, в которой игроку предстоит ловить различные предметы, зарабатывая при этом очки и жизнь. Основная цель игры заключается в том, чтобы игрок как можно больше ловил хорошие предметы, которые будут приносить очки, и избегать плохих, которые могут привести к потере очков и жизнью. Главной сложностью для игрока является то, что если игрок будет ловить плохие предметы 5 раз, то он проиграет. А если будет ловить хорошие предметы и наберет 50 очков, то выиграет. Однако, в игре также присутствуют специальные предметы, которые при касании с игроком могут принести дополнительную жизнь. Поэтому интерес этой игры заключается в быстром реагировании игрока на падающие предметы.

Аналоги игр в этом жанре:

Игра «Pou», в котором игроку нужно ловить как хорошие, так и плохие предметы. Он может заработать очки, при ловли хороших предметов, и потерять жизнь при ловли плохих.

Шутер – игра в жанре атаки, в которой игрок принимает участие в сражении против своих врагов. В этой игре, главной задачей, поставленной перед игроком, является стрельба лазером в противников, чтобы уничтожить их и заработать очки. Основными функциями игрока являются передвижение по игровому полю и стрельба лазером, что позволяет ему уклоняться от атак

врагов и эффективно атаковать их. Главной особенностью этой игры является то, что стреляет не только игрок в своих противников, но и враги имеют возможность нанести урон в игрока, отнимая у него жизнь.

Аналоги в этом жанре:

Игра «Танчики», в которой происходит битва между игроками и врагами. Цель этой игры уничтожать своих врагов, стреляя в них, чтобы не потерять жизнь.

1.2 Описание среды разработки

Все 3 игры будут разрабатываться в среде Unity, которая предоставляет разработчикам все необходимые инструменты для создания 2D, 3D игр. Используемая версия: Unity 2D 3.5.2.

Visual Studio Code — это легкий и мощный редактор кода, который идеально подходит для разработки на языке программирования C#. Он предлагает множество расширений, что делает его отличным выбором для работы с Unity.

1.3 Техническое задание на разработку

Задание для разработки платформера: Создание земли с правильным порядком в слое (order in layer), условия выигрыша и проигрыша, анимация для игрока и врагов, фон для каждой сцены, связь окончания игры с потерей жизней, меню проигрыша и выигрыша.

Задание для разработки кетчера: Реализация движения игрока, создание механик для хороших и плохих объектов, спавнер для предметов, которые будут лететь, механика поимки предметов (хороших и плохих), условия для победы и поражения, основанные на поимке предметов, анимация, добавление фонового изображения для каждой сцены, ограничение на количество жизней персонажа, меню с настройками игры (громкость звука, пауза).

Задание для разработки шутера: Игрок управляет космическим кораблем с помощью мыши, реализовать механизм стрельбы лазером, создание астероидов, движущихся по экрану, спавн астероидов с помощью

специального объекта, нанесение повреждений врагам с помощью лазера, появление игрока в рандомном месте на экране, создание врага со стрельбой пулями, разработка очков здоровья для игрока, добавление звезд для имитации движения, реализация health bar для врагов и персонажа.

2 Проектирования и разработка приложения

2.1 Проектирование уровней и логики игры

Платформер.

Концепция уровня: созданный уровень фокусируется на взаимодействии игрока с врагами и окружающей средой. Главная цель — уничтожить всех врагов, что добавляет элемент стратегии, так как игрок должен использовать доступные ему ресурсы и механики игры (прыжки, бег) для успешного выполнения задачи.

Логика игры: каждый враг имеет уникальные паттерны поведения. Это создает ситуации, требующие от игрока быстрой реакции и тактического мышления. Например, один из врагов может менять направление.

Уровень сложности: сложность игры прогрессивно возрастает с каждым новым врагом, который появляется по мере продвижения игрока по уровню, что поддерживает интерес и стимулирует игроков к улучшению навыков.

Кэтчер

Концепция уровня: акцент сделан на сбор очков и сохранение жизней. Игрок должен собрать 250 очков, избегая при этом столкновений с «плохими» предметами.

Логика игры: очки начисляются за каждый успешно собранный «хороший» объект (пончик, пиццу, лапшу).

Жизни игрока: игроку даны три жизни, что позволяет ему проявлять рискованное поведение, но также заставляет внимательно следить за состоянием персонажа.

Уровень сложности: на протяжении всей игры уровень сложности не меняется.

Шутер

Концепция уровня: игрок должен увернуться от летящих пуль врагов и одновременно уничтожать их и астероиды, чтобы увеличить очки. Логика игры: враги стреляют всегда в сторону игрока, что заставляет игрока развивать ловкость и хорошую координацию движений.


```

        dopanim();
    }
    if (isGrounded && Input.GetKeyDown(KeyCode.C))
    {
        dop2();
    }
}

```

Класс отвечает за управление персонажем в игре. Он обрабатывает ввод от игрока, позволяя персонажу двигаться, прыгать и выполнять дополнительные действия (например, анимации) в зависимости от состояния (находится ли на земле) и нажатых клавиш.

```

2)    private void Run()
    {
        if(isGrounded) State = States.Run;
        Vector3 dir = transform.right * Input.GetAxis("Horizontal");
        transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, speed * Time.deltaTime);
        sprite.flipX = dir.x < 0.0f; //поворот глаз
    }

```

Этот класс отвечает за управление движением объекта в горизонтальном направлении. В методе Run проверяется, находится ли объект на земле (isGrounded), и если да, то устанавливается состояние движения в "бег" (States.Run). Затем рассчитывается направление движения на основе ввода пользователя по горизонтали. Объект перемещается в заданном направлении с заданной скоростью. Также происходит отражение спрайта в зависимости от направления движения, что позволяет корректно визуализировать игрока при движении влево и вправо.

Кетчер:

```

1)    void Start()
    {

```

```

        rb = GetComponent<Rigidbody2D>();
        currentLives = lives.Length; // Инициализация текущих жизней
    }

```

Этот класс отвечает за инициализацию компонента при запуске игры. В методе Start происходит получение компонента Rigidbody2D, который позволяет объекту взаимодействовать с физикой в 2D-пространстве. Также происходит инициализация переменной currentLives, которая устанавливает текущее количество жизней, исходя из длины массива lives.

```

2)    void Start()
    {
        audioSrc = GetComponent<AudioSource>();
    }

```

Класс для воспроизведения музыки:

```

3)    void Start()
    {
        resolutionDropdown.ClearOptions();
        List<string> options = new List<string>();
        resolutions = Screen.resolutions;
        int currentResolutionIndex = 0;
        for (int i = 0; i < resolutions.Length; i++)
        {
            string option = resolutions[i].width + "x" + resolutions[i].height + " "
+ resolutions[i].refreshRateRatio + "Hz";
            options.Add(option);
            if (resolutions[i].width == Screen.currentResolution.width &&
resolutions[i].height == Screen.currentResolution.height)
                currentResolutionIndex = i;
        }
        resolutionDropdown.AddOptions(options);
        resolutionDropdown.RefreshShownValue();
    }

```

```
LoadSettings(currentResolutionIndex);  
}
```

Этот класс отвечает за инициализацию выпадающего списка разрешений экрана в игре. В методе Start он очищает текущие опции, получает доступные разрешения, формирует список строк с информацией о каждом разрешении и добавляет их в выпадающий список. Также он определяет текущее разрешение и загружает соответствующие настройки для правильной работы интерфейса.

Шутер:

```
1) void Start()  
{  
    rb = GetComponent<Rigidbody2D>();  
    Vector3 move = new Vector3(0, -1, 0);  
    rb.velocity = move * speed;  
}
```

Этот класс инициализирует физический объект, задавая его начальную скорость вниз при старте игры.

```
void OnTriggerEnter2D(Collider2D other)  
{  
    if (other.gameObject.tag == "Enemy")  
    {  
        other.gameObject.SendMessage("MakeDamage", damage,  
SendMessageOptions.DontRequireReceiver);  
        Destroy(gameObject);  
    }  
}
```

Этот класс позволяет игроку наносить урон врагу. OnTriggerEnter2D срабатывает при столкновении с врагом. Если объект имеет тег "Enemy", продолжается нанесение урона. Лазер уничтожается после столкновения.

Схема переходов игры показана на рисунке 2.

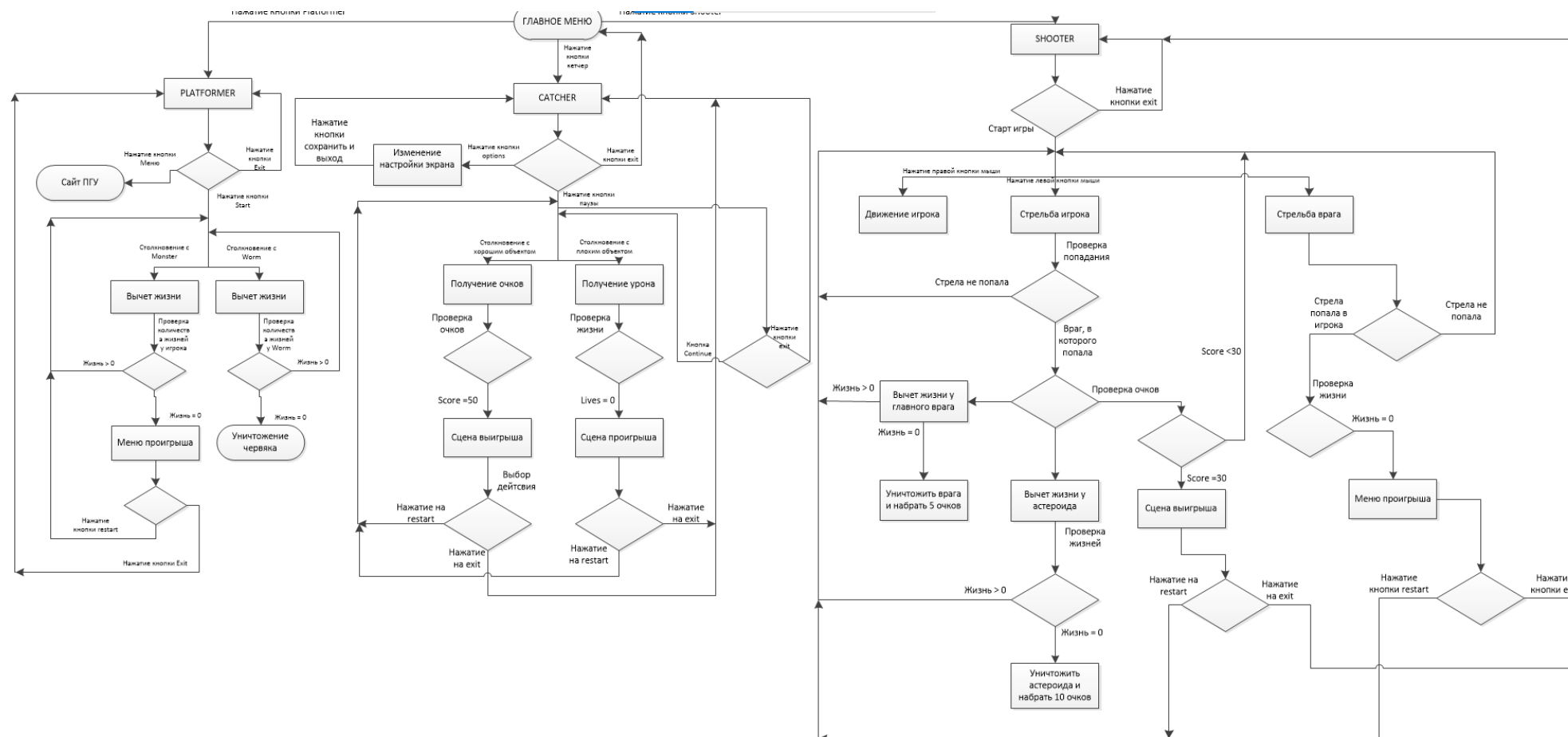
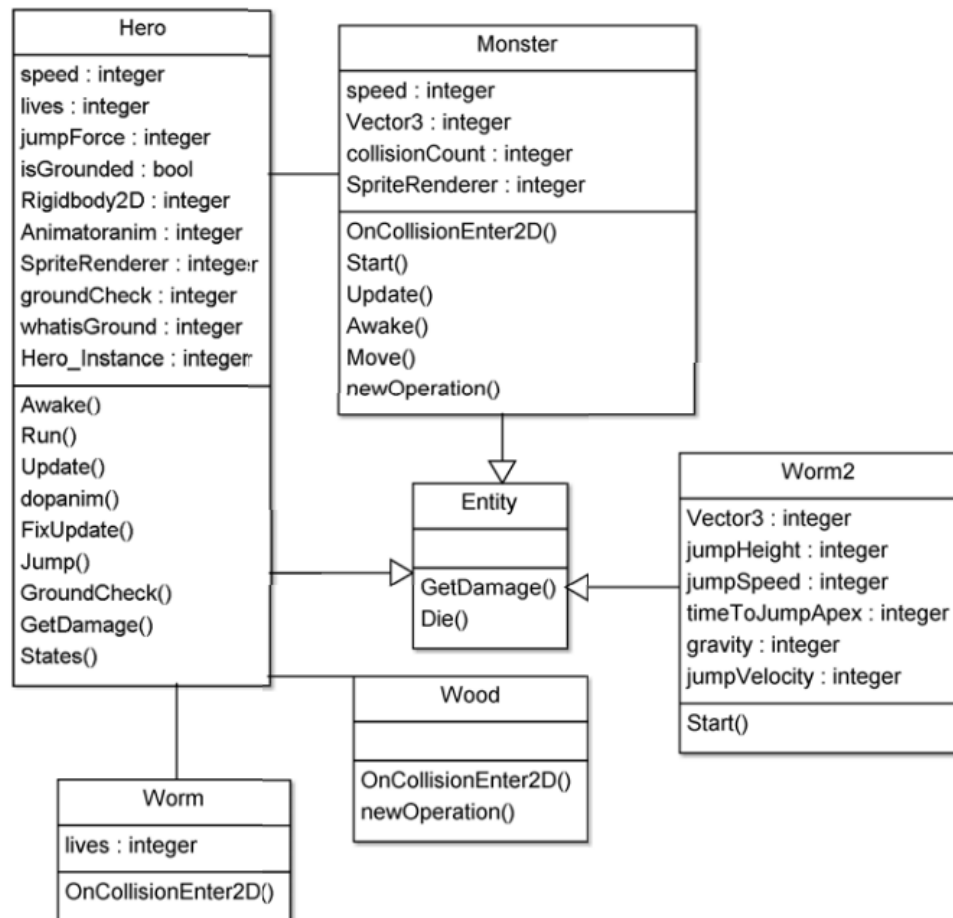


Рисунок 2 - Схема переходов

Диаграмма классов для платформера представлена на рисунке 3.



АКТИВ
11 6

Рисунок 3 - Диаграмма классов Платформера

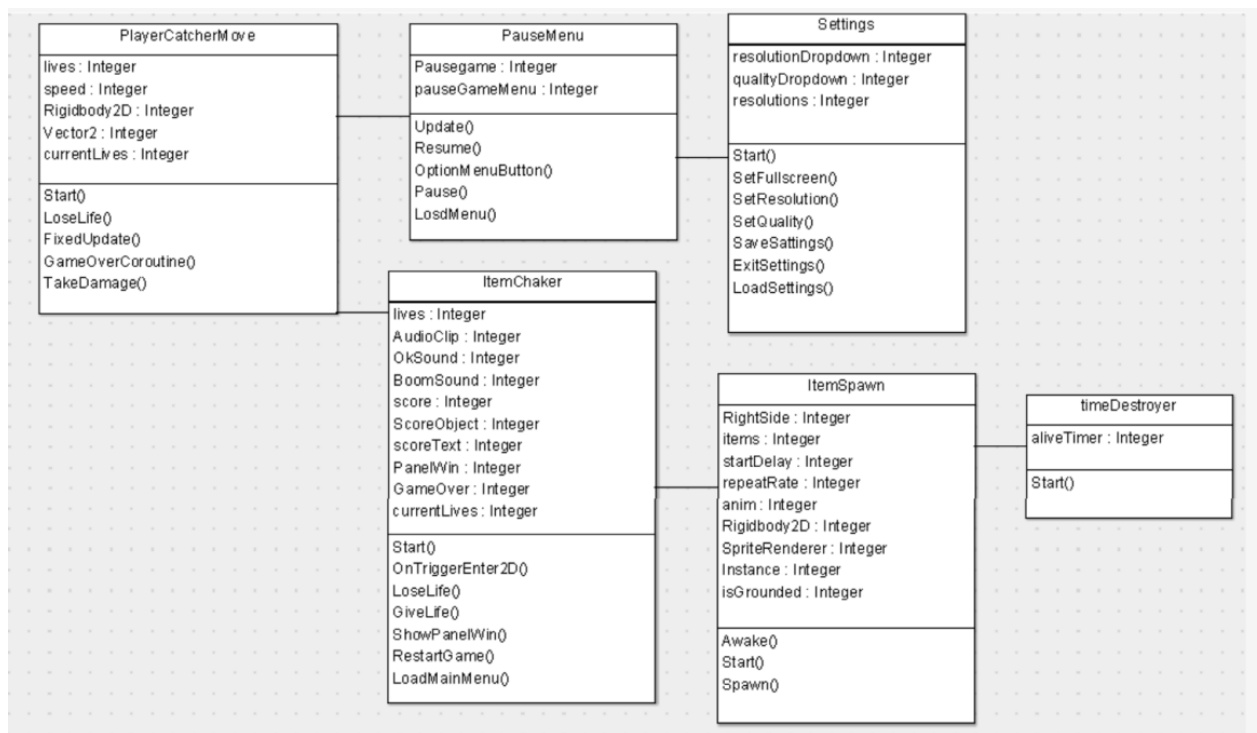


Рисунок 4 - Диаграмма классов Кетчера

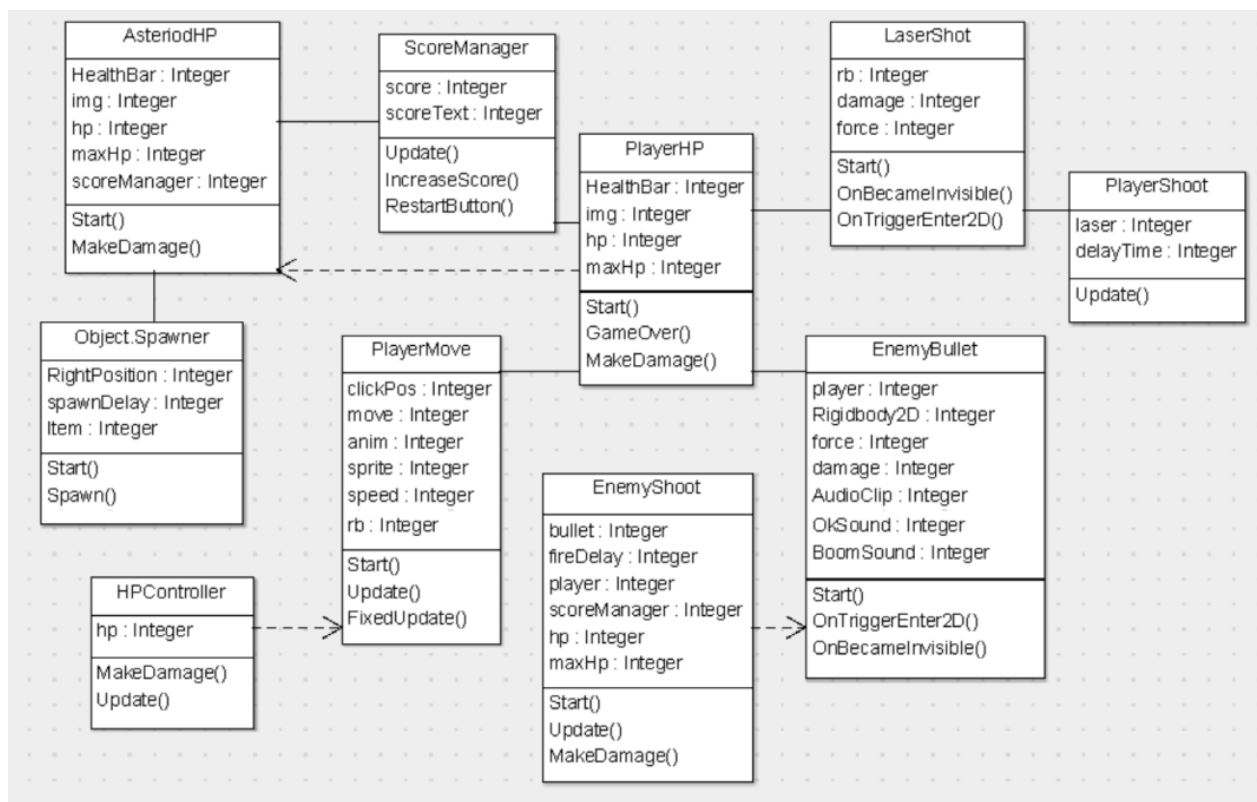


Рисунок 5 - Диаграмма классов для Шутера

2.3 Ресурсы проекта

В этом разделе представлены ресурсы, использованные в проекте, а также их описание и назначение.

2.4 Основные анимации игры

Анимации для игры Платформер:

Игрок:

Анимация «Прыжок»

Назначение: Используется, когда игрок прыгает вверх

Описание анимации: 2 кадра, персонаж делает прыжок ногами вверх и вниз

Последовательность анимации. (Рисунок 6):



Рисунок 6 - Анимация прыжка

Анимация «Бег»

Назначение: Используется, когда игрок движется по горизонтали

Описание анимации: 6 кадров, персонаж перебирает руками и ногами вправо и влево.

Условия запуска анимации: Когда игрок нажимает кнопки вправо, влево,



Рисунок 7 - Анимация бега

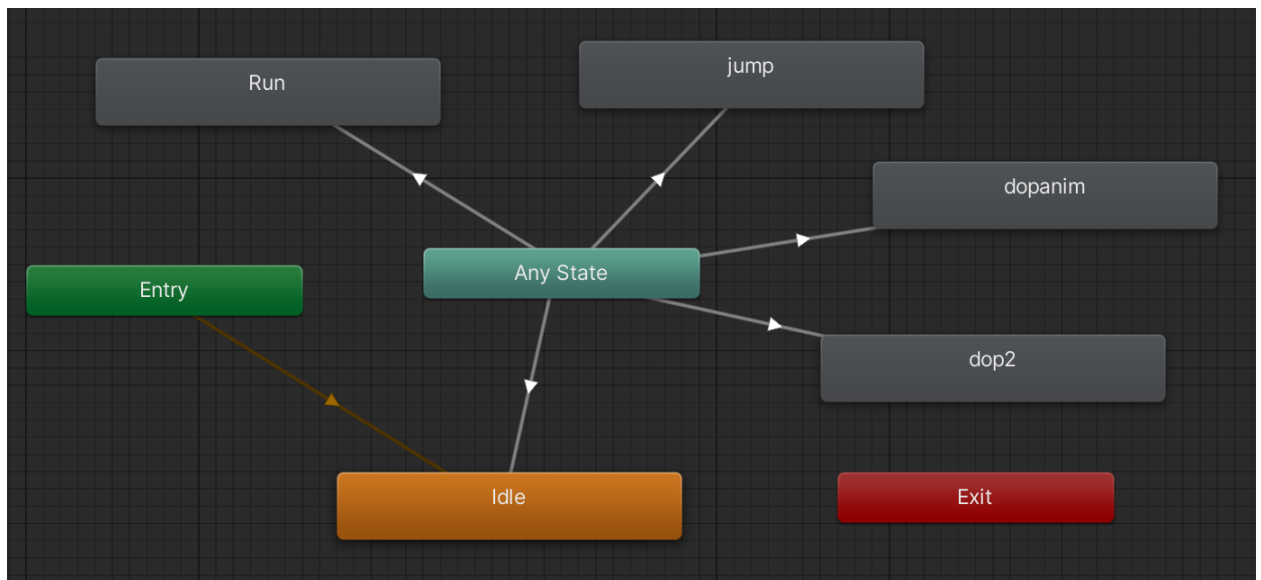


Рисунок 8 - Построение и связь анимации в Animator

Код:

```

private void Run()
{
    if(isGrounded) State = States.Run;
    Vector3 dir = transform.right * Input.GetAxis("Horizontal");
    transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, speed * Time.deltaTime);
    sprite.flipX = dir.x < 0.0f; //поворот глаз
}
private void Jump()
{
    rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
}
private void GroundCheck()
{
    Collider2D[] collider = Physics2D.OverlapCircleAll(transform.position,
0.3f);
    isGrounded = collider.Length > 1;
    if(isGrounded) State = States.Jump;;
}

```

Неподвижный анимированный враг:

Анимация «Вверх-вниз»

Назначение: Используется на протяжении всей игры, по определенно заданному временному интервалу

Описание анимации: 8 кадров, персонаж поднимается и опускается.

Условия запуска анимации: Как только запускается игра.

Последовательность анимации (Рисунок 9, 10):

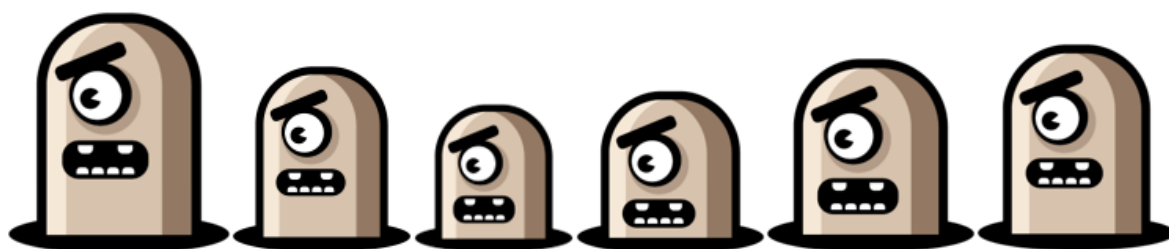


Рисунок 9 - Анимация врага

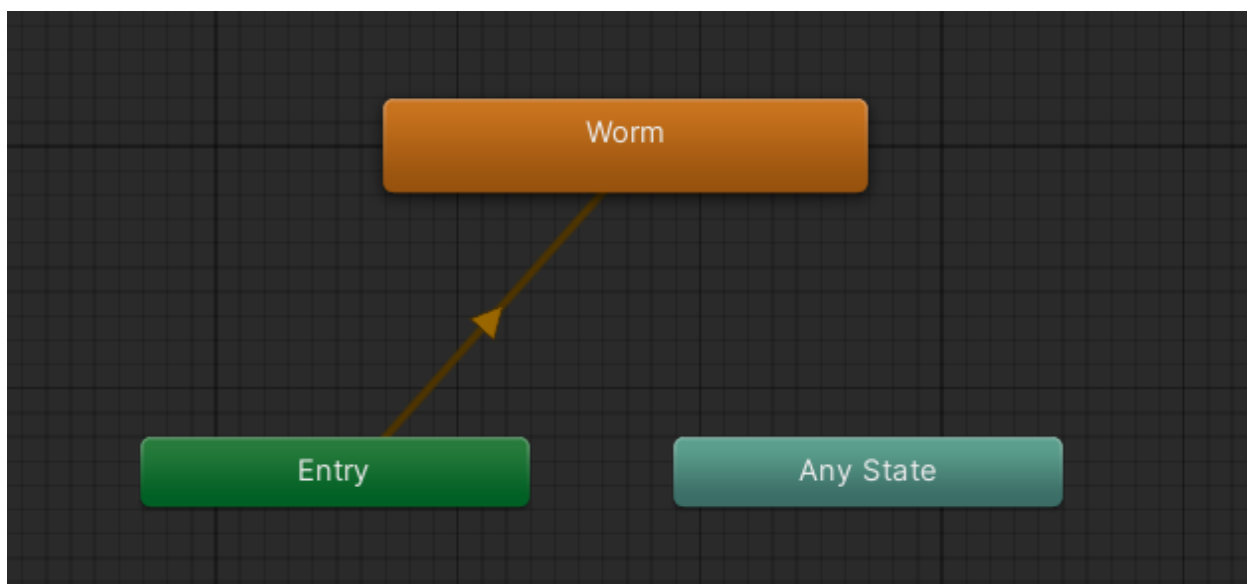


Рисунок 10 - Аниматор неподвижного врага

Анимация для игры Catcher:

Анимация «Вращение объекта»

Назначение: Используется на протяжении всей игры.

Описание анимации: анимация вращения происходит, как только начинают падать объект.

Условия запуска анимации: Как только запускается игра



Рисунок 11 - Анимация с хорошим объектом

Код:

```
public float rotationSpeed = 180f;

void Update()
{
    transform.Rotate(Vector3.forward * rotationSpeed * Time.deltaTime);
}
```

Шутер:

Анимация «Вращение»

Назначение: Используется при нажатии клавиши.

Описание анимации: анимация вращения корабля происходит, как только игрок нажмет на определенную клавишу. Корабль начинает вращаться непрерывно на 360 градусов. (Рисунок 12):



Рисунок 12 – Анимация вращения корабля

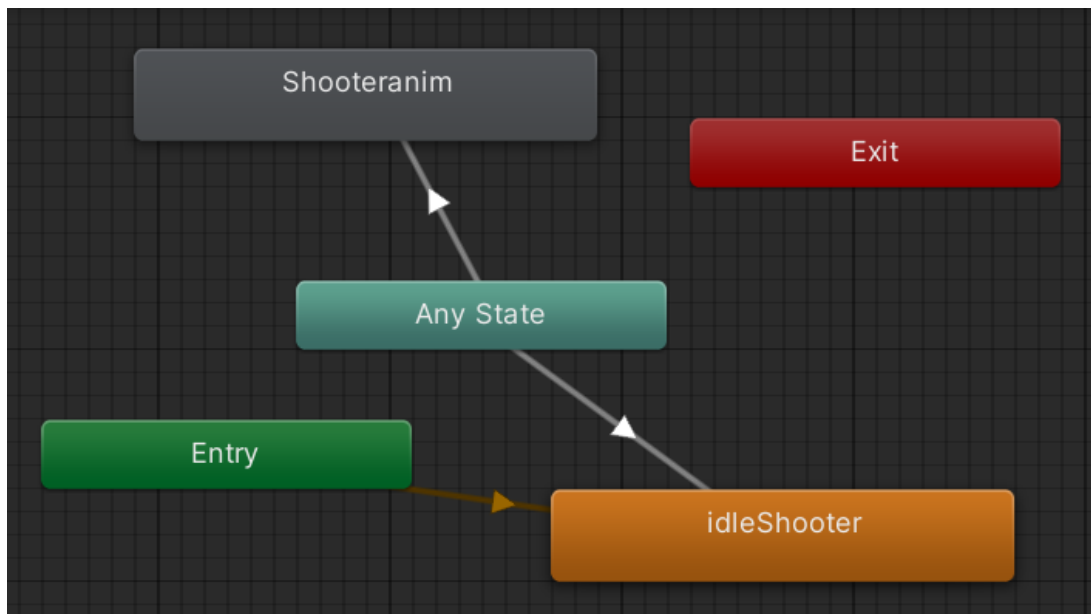


Рисунок 13 - Построение и связь анимации в Animator

Код:

```

void Update()
{
    if (Input.GetMouseButton(0))
    {
        clickPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    }
    move = clickPos - transform.position;
    if (Input.GetKeyDown(KeyCode.C))
    {
        State = States.Shooteranim;
    }
    else if (State == States.Shooteranim)
    {
        State = States.idleShooter;
    }
}

public enum States
{

```

```
    Shooteranim,  
}
```

3 Тестирование приложения

При запуске игры, открывается окно главного меню, где игроку предоставляется возможность выбрать одну из трех игр, рисунок 14.

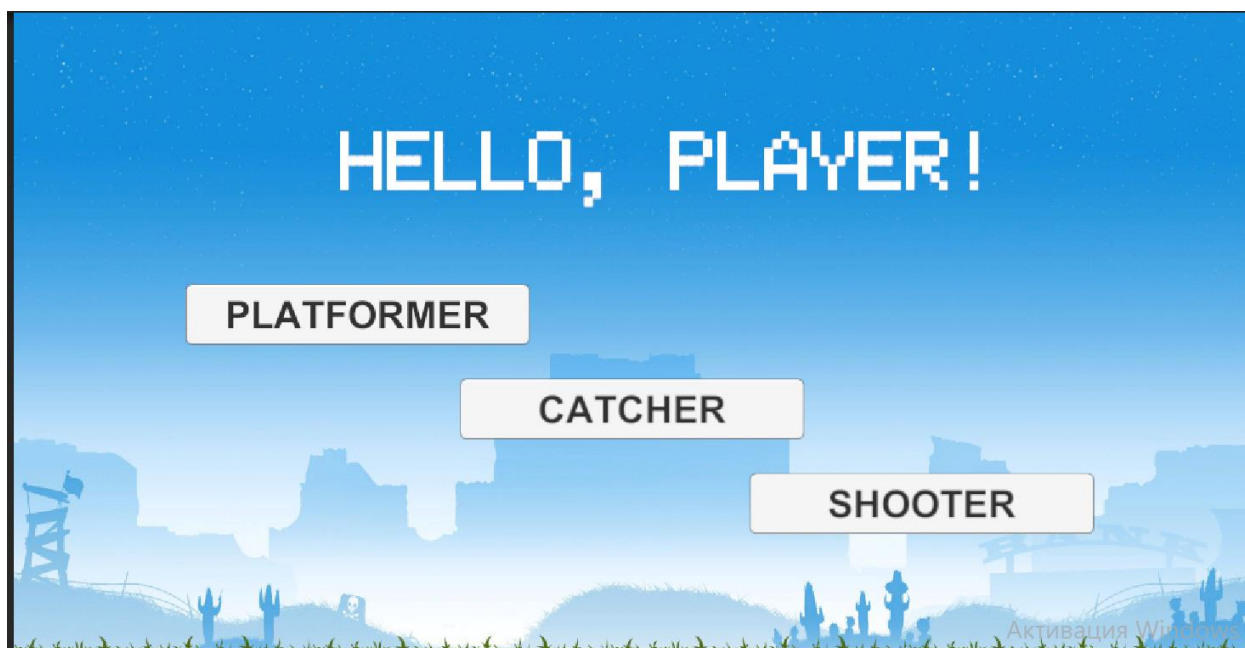


Рисунок 14 - Главное меню

При выборе игры Платформер, мы попадаем в главное окно, где у нас имеются уровни, построенные из платформ, персонаж, враги, разные объекты (деревья), которые при каждом новом запуске появляются из случайных мест, который показан на рисунке 15-16.

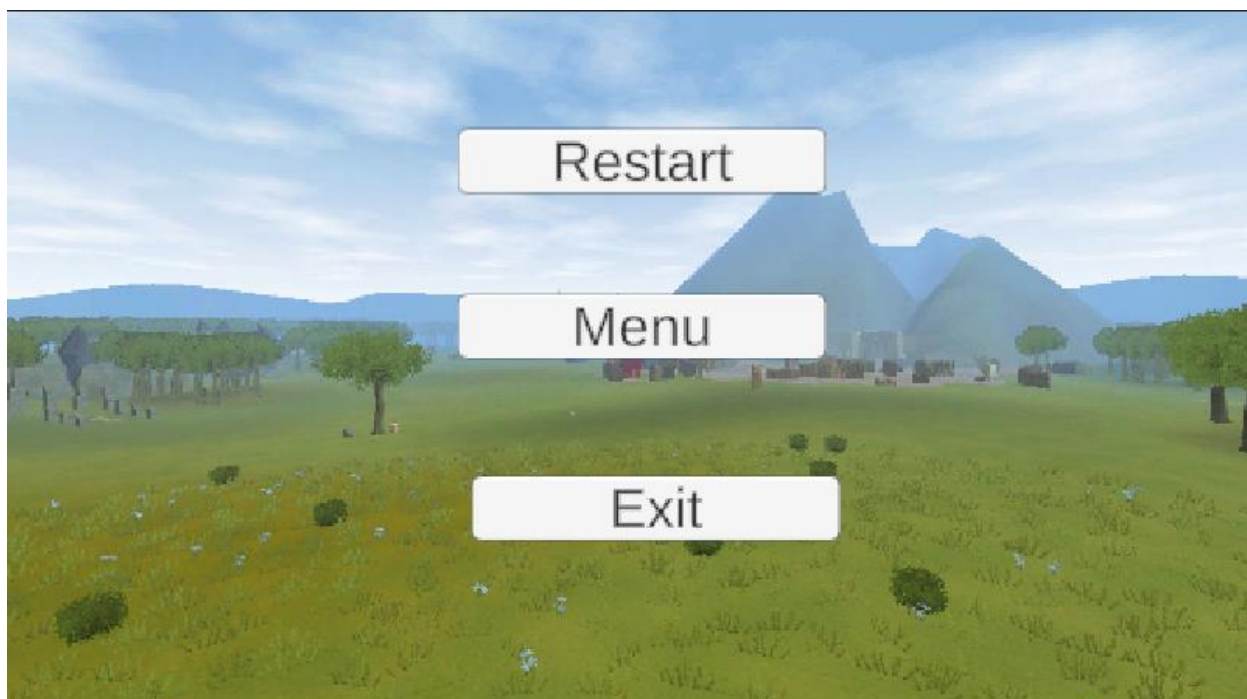


Рисунок 15 - Окно меню игры

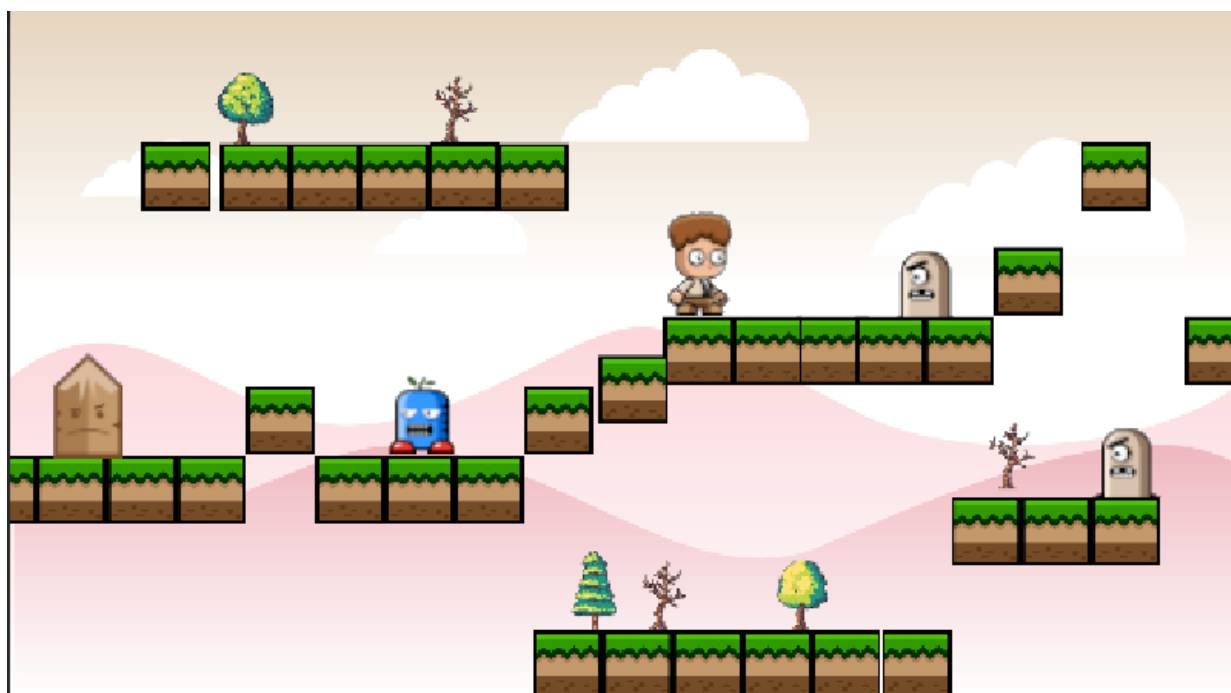


Рисунок 16 - Процесс игры

Главные функции персонажа: бег, прыжки, уничтожение врага, прыжок через препятствия. Персонаж может уничтожить врага, после трех столкновений с ним. Результат представлен на рисунке 17.

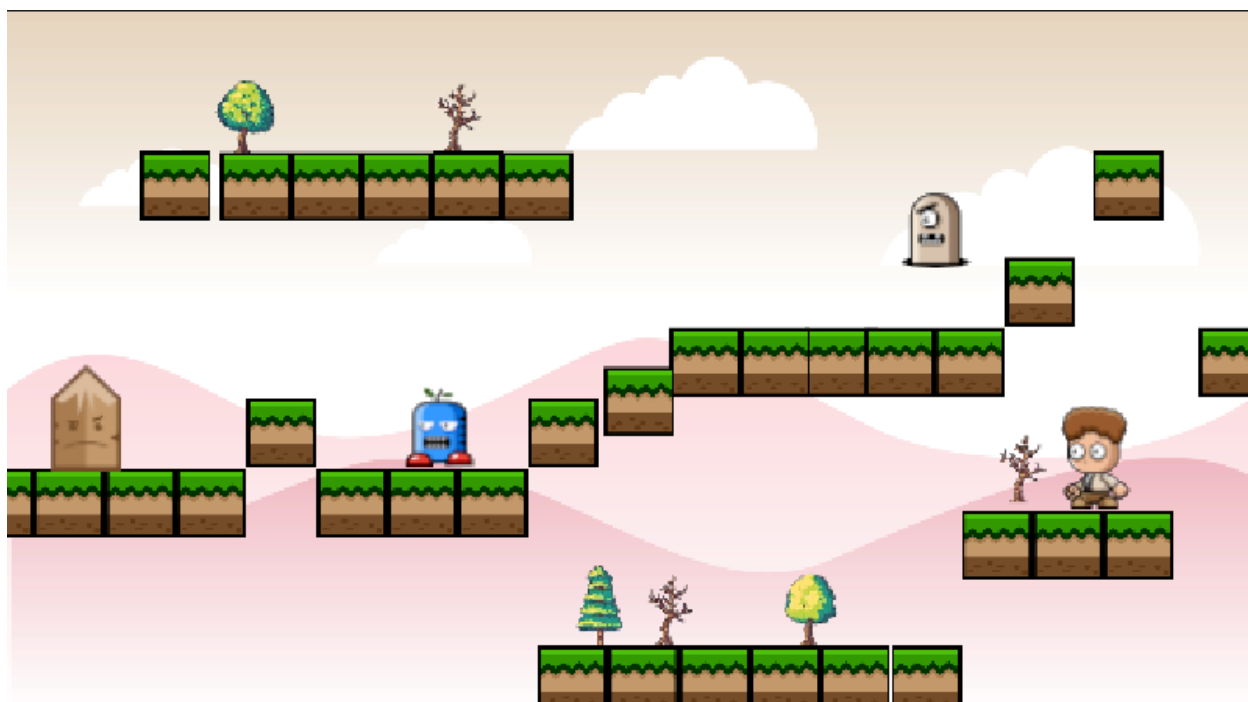


Рисунок 17 - Уничтожение врага

После 3 столкновений с синим врагом, персонаж погибает. Результат представлен на рисунке 18.

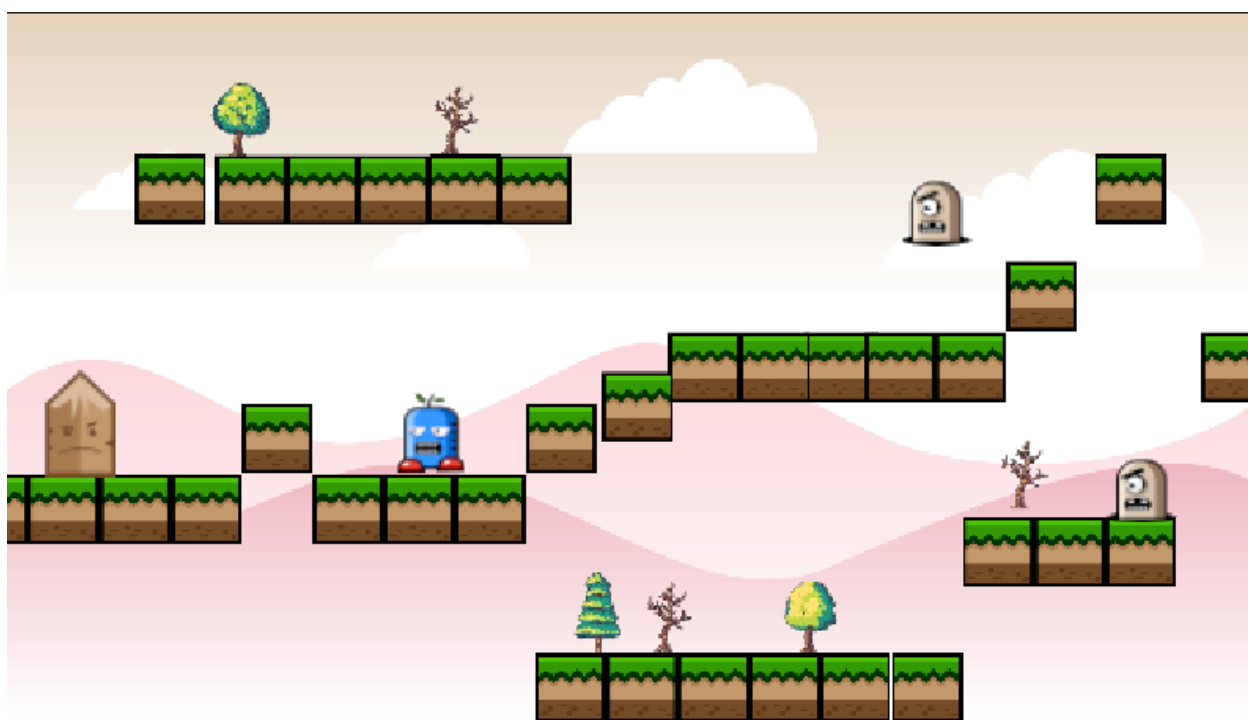


Рисунок 18 - Уничтожение персонажа

После того, как персонаж умирает, открывается окно проигрыша. Результат на рисунке 19.



Рисунок 19 - Окно проигрыша

После запуска Кетчера открывается окно меню, в котором есть три кнопки: кнопка старта, настроек и выхода.

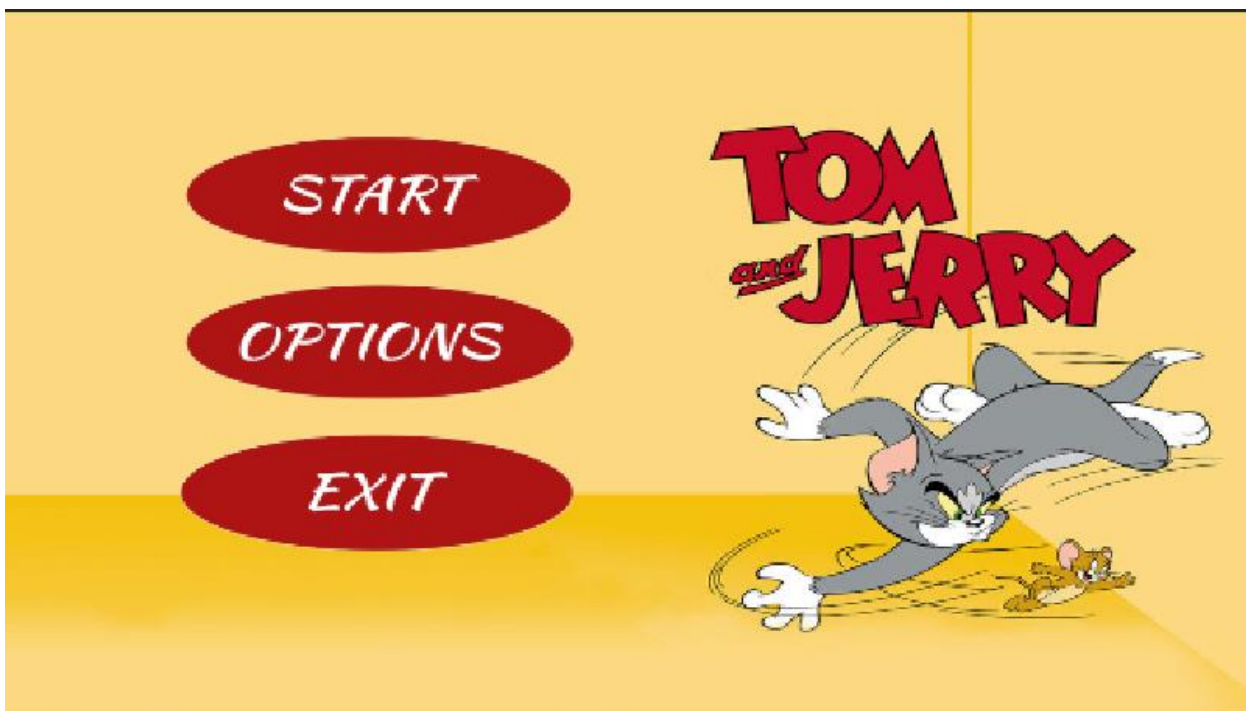


Рисунок 6 - Окно меню игры

При нажатии на кнопку «Start», запускается и начинается игра, которая показана на рисунке 13. В этом окне показаны игрок, который будет ловить падающие объекты, количество жизней игрока, счет очков, кнопка паузы, к

которому можно перейти, нажав на правую кнопку мыши или на Esc. Также показан слайдер для контроля звука.



Рисунок 7 – Начальная сцена игры

При нажатии на кнопку паузы, открывается панель, где есть возможность продолжить игру, либо перейти в меню, результат показана на рисунке 14.



Рисунок 8 - Окно паузы

Панель выигрыша, при соприкосновении игрока с хорошими объектами, в результате достижения 50 очков. Результат показан ниже.



Рисунок 9 - Сцена выигрыша

А теперь, можем видеть сцену проигрыша, при окончании жизней у игрока. Игрок может проиграть, если 5 раз столкнется с плохим объектом. Результат показан на рисунке 16.



Рисунок 10 - Сцена проигрыша

Также, в окне меню можно перейти к настройкам, где можно управлять разрешением экрана. Можно изменять качество, режим экрана и сохранять выбранные функции. Результат показан на рисунке 17.

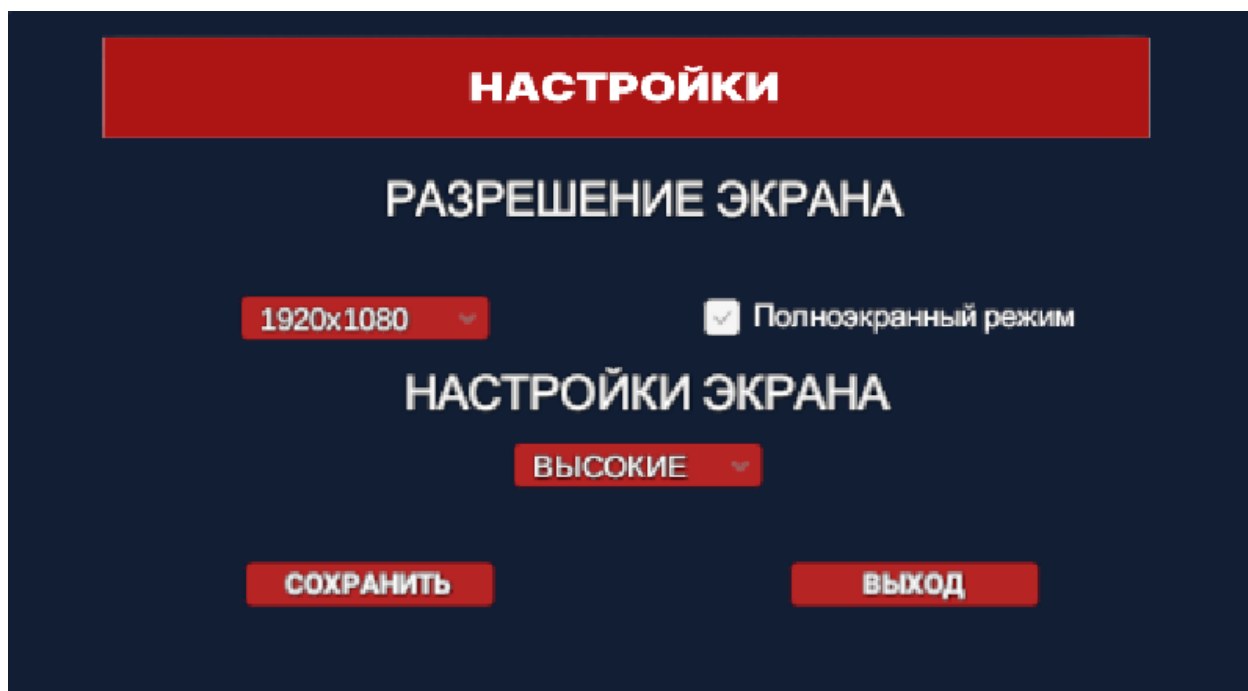
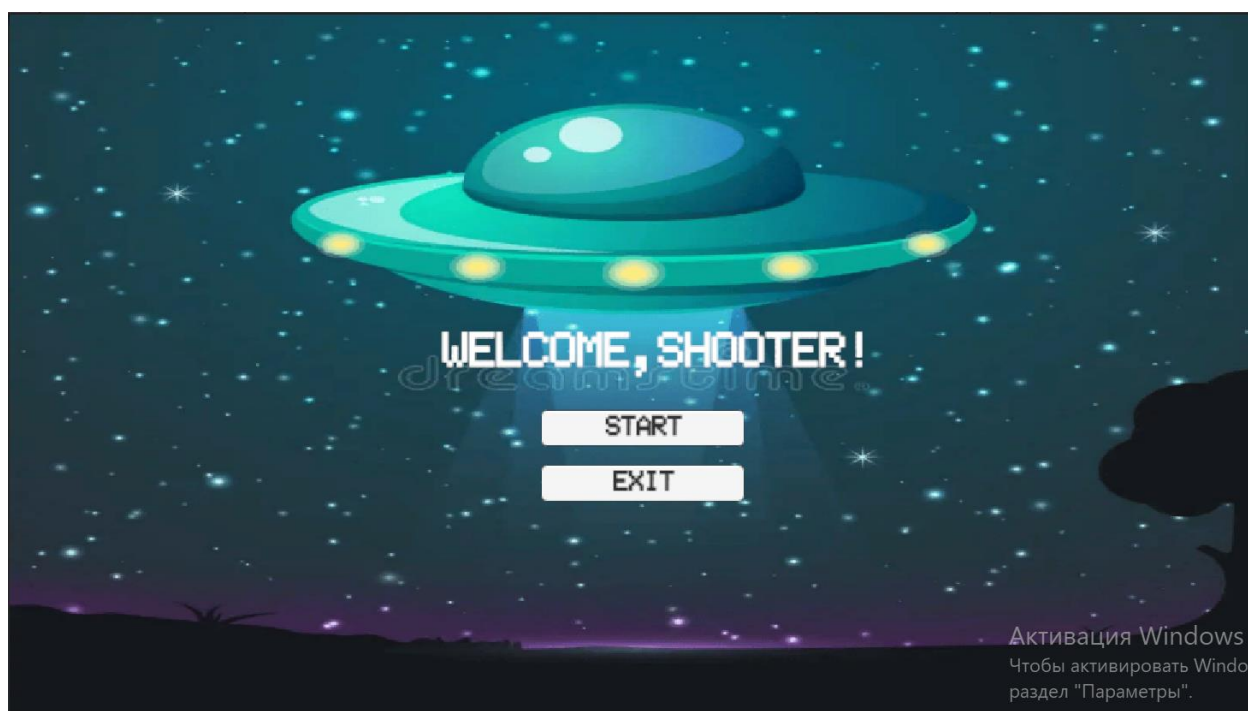


Рисунок 11 - Настройка разрешения экрана

После запуска игры Шутер, на сцене, мы видим игрока, слайдер звука, контроллер его жизни, в виде слайдера и счет очков, которая показана на рисунке 18.



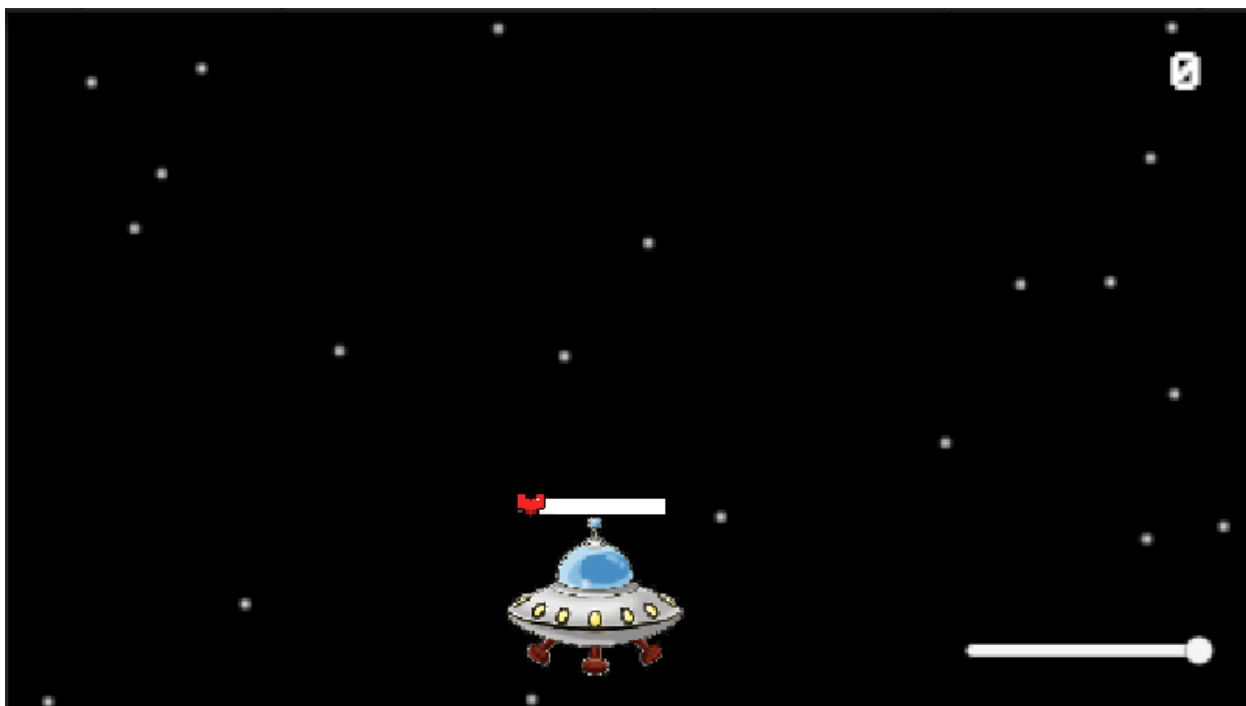


Рисунок 12 - Запуск игры Shooter

После того, как начинается игра, появляются астероиды, враги, рисунок 19.



Рисунок 13 - Появление врагов

Далее показана, нанесение урона во врага. Если игрок попадет лазером 5 раз, то астероид умирает. Результат показан на рисунке 20.

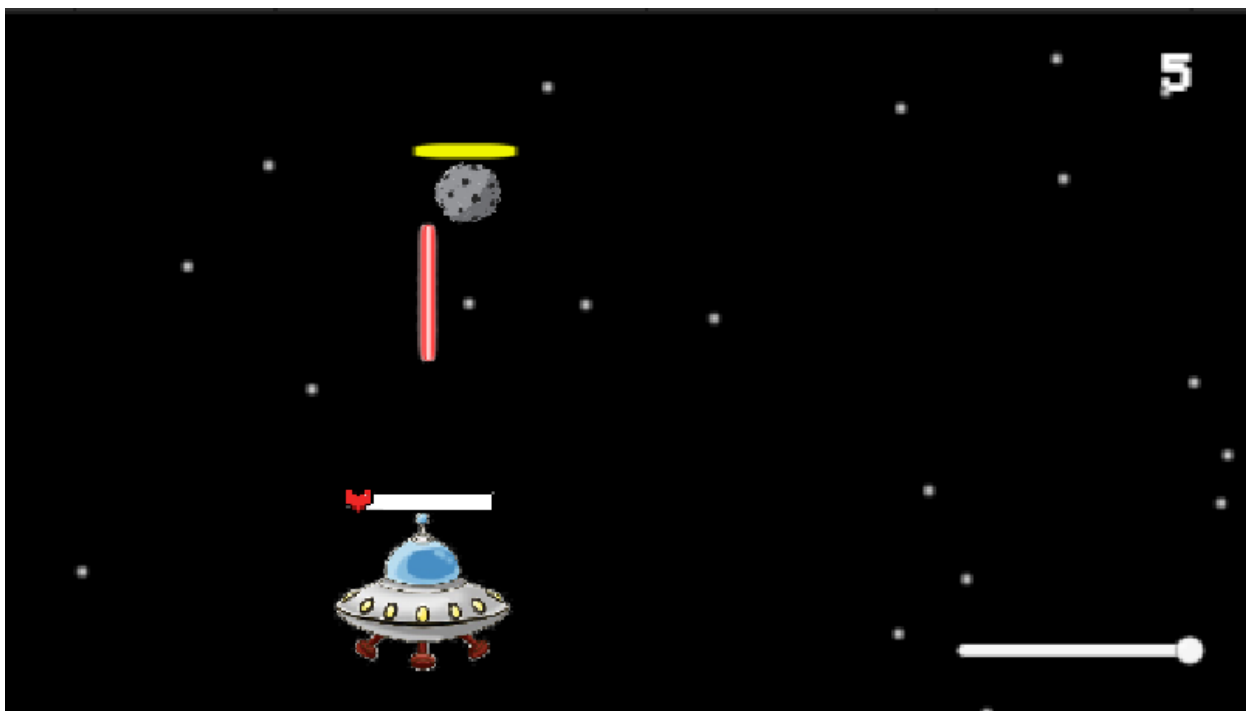


Рисунок 14 - Уничтожение врага

Нанесение урона на игрока. Если враг попадет 5 раз на игрока, то игрок умирает. Результат показан на рисунке 21.



Рисунок 15 - Нанесение урона на игрока

После уничтожения врага, появляется сцена проигрыша, в которой есть кнопка рестарт. Нажав на кнопку рестарт, можно снова начать игру. Результат показан на рисунке 22.

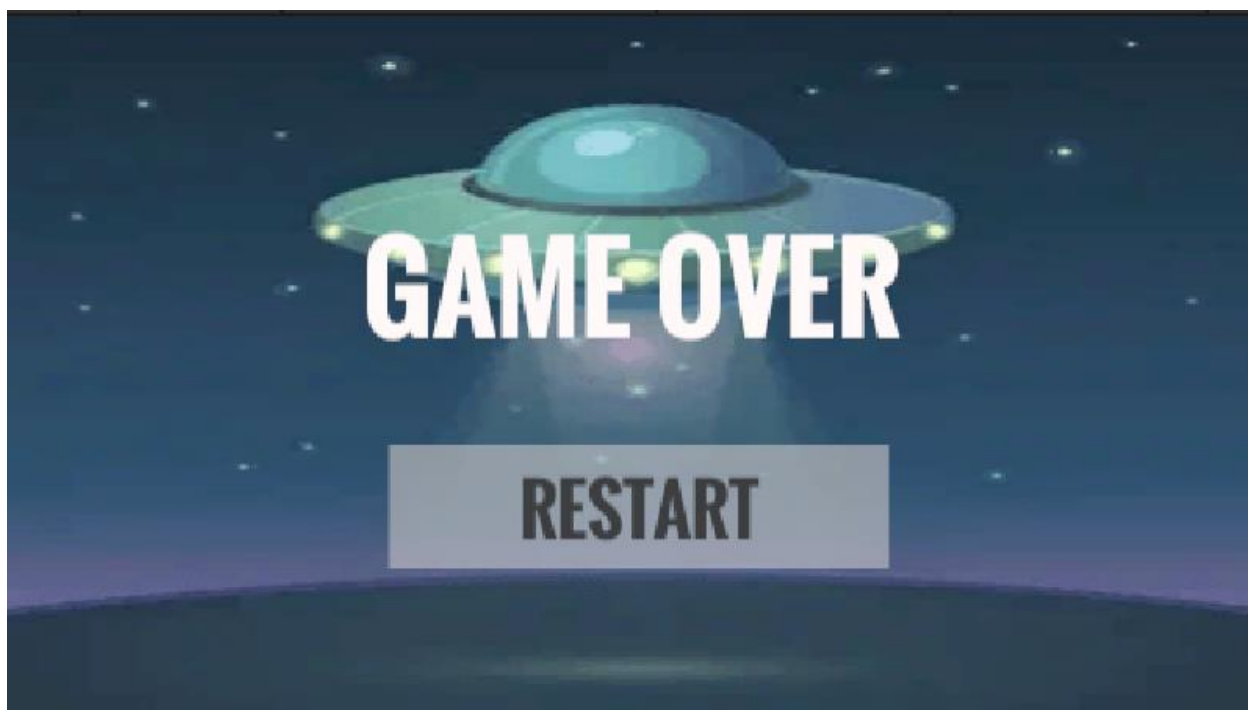


Рисунок 16 - Сцена проигрыша

Если игрок, уничтожив врага и заработав 50 очков, то он выиграет и появится окно выигрыша, где также есть кнопка рестарт, нажав на нее, можно перезагрузить игру заново. Результат показан на рисунке 23.

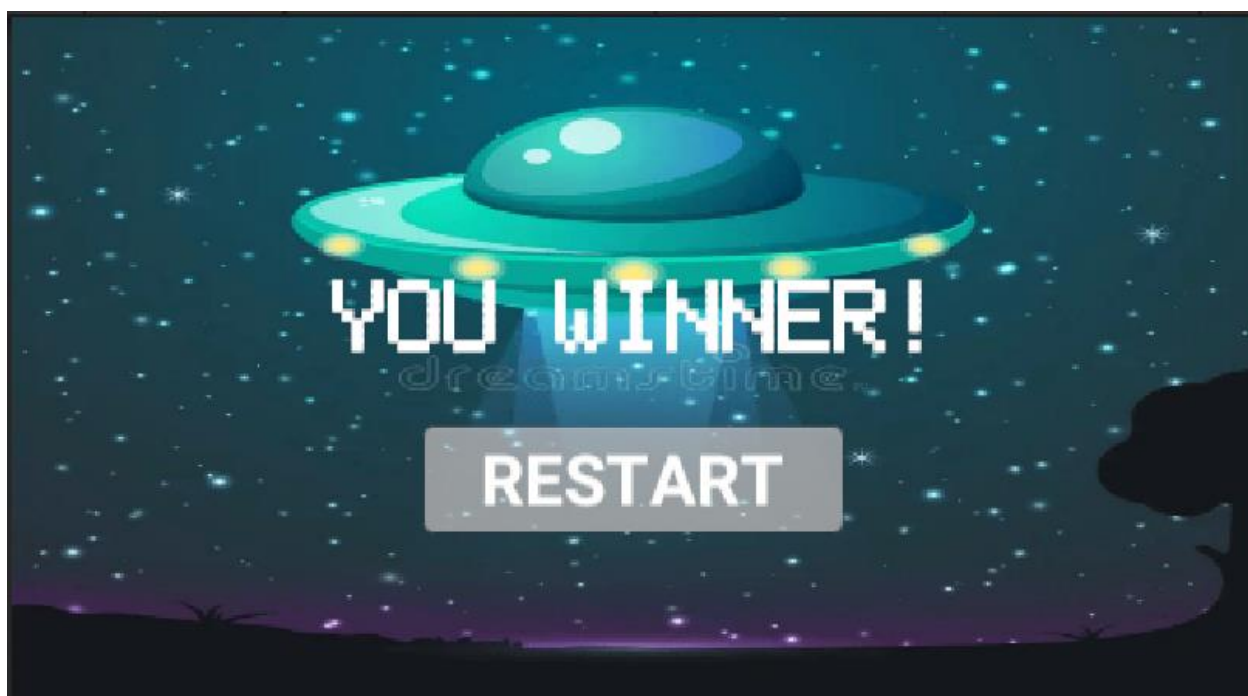


Рисунок 17 - Сцена выигрыша

Заключение

В курсовой работе исследовали процесс разработки трех 2D игр разных жанров: Платформера, Кетчер и Шутер, каждая из которых обладает уникальными особенностями. Все игры были разработаны в среде Unity и включают в себя графический интерфейс, меню, спрайты, анимации, а также условия выигрыша и проигрыша, звуковые эффекты и фоновую музыку для погружения игрока в процесс. При выполнении данной курсовой работы были получены навыки программирования и разработки игры на языке с#

Анализ и разработка этих игр позволили углубить понимание технологий, применяемых в разработке программного обеспечения, и отработать практические навыки, необходимые для создания игровых проектов.

Список использованных источников

Фон для Шутера

https://avatars.mds.yandex.net/i?id=49fbc6d1c11342cdabc77ecd92662595_1-5279455-images-thumbs&n=13

Фон для Кетчера

<https://i.ytimg.com/vi/k7Ynz3qJ0Y8/maxresdefault.jpg>

Фон для Платформера

<https://i.pinimg.com/736x/08/07/0d/08070da0dfb1bab37e91be48ee0980ba.jpg>

Жизнь для игрока

https://avatars.mds.yandex.net/i?id=4a9fa0a7464a9b9bf6460acfca659686_1-5260221-images-thumbs&n=13

Методические указания

Для добавления звука –

<https://www.youtube.com/watch?v=g3Y6aHUPhWs>

Для добавления меню паузы –

<https://www.youtube.com/watch?v=LE44TT2mQ1I>

Для управления разрешением экрана –

<https://www.youtube.com/watch?v=HoqEeSjlfWI>

Для добавлений кат сцен –

<https://www.youtube.com/watch?v=QfVC0D7vxhU>
https://www.youtube.com/watch?v=uOm0gU_NiL4

Враг для игры Кетчер

<https://i.pinimg.com/736x/f1/f8/5e/f1f85ea1cab54cd3d8d934c35a4531d3.jpg>

Игрок для Кетчера

<https://avatars.mds.yandex.net/i?id=3cea9b6cacao68fe8af82abf49da5b2bfc8ff1b3-10814598-images-thumbs&ref=rimg&n=33&w=480&h=270>

Корабль для игры Шутер

<https://udoba.org/sites/default/files/h5p/content/75974/images/match-642c4cab17439.jpg>

Враг для игры Шутер

https://avatars.mds.yandex.net/i?id=207cf557a47f8ac9f36ab74f02f34e52_1-5257302-images-thumbs&n=13

Лазер для Шутера

<https://avatars.mds.yandex.net/i?id=420c12c903428d50bfbb1af59d7ab1169c0e3e84-9066439-images-thumbs&ref=rin&n=33&w=450&h=300>

Приложение

Файл Hero.cs

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Hero : Entity
{
    [SerializeField] private float speed = 10f; //скорость
    [SerializeField] private int lives = 3; //количество жизни
    [SerializeField] private float jumpForce = 15f; //сила прыжка

    private bool isGrounded = false;

    private Rigidbody2D rb;
    private Animator anim;
    private SpriteRenderer sprite;
    //public LayerMask whatIsGround;
    //public Transform groundCheck;

    public static Hero Instance { get; set; }

    private States State
    {
        get { return (States)anim.GetInteger("state"); }
        set { anim.SetInteger("state",(int)value); }
    }

    private void Awake()
    {
        sprite = GetComponentInChildren<SpriteRenderer>(); //Обращение к
спрайту
        anim=GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
        Instance = this;
    }

    private void Run()
    {

```

```

        if(isGrounded) State = States.Run;
        Vector3 dir = transform.right * Input.GetAxis("Horizontal");
        transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, speed * Time.deltaTime);
        sprite.flipX = dir.x < 0.0f; //поворот глаз
    }

    private void Update()
    {
        if (isGrounded) State = States.Idle;
        if (Input.GetButton("Horizontal"))
            Run();
        if (isGrounded && Input.GetButtonDown("Jump"))
            Jump();
        if (isGrounded && Input.GetKeyDown(KeyCode.R))
        {
            dopanim();
        }
        if (isGrounded && Input.GetKeyDown(KeyCode.C))
        {
            dop2();
        }
    }
    private void dopanim()
    {
        if (Input.GetKeyDown(KeyCode.R)) State = States.dopanim;
    }
    private void dop2()
    {
        if (Input.GetKeyDown(KeyCode.C)) State = States.dop2;
    }
    private void FixedUpdate()
    {
        GroundCheck();
    }
    private void Jump()
    {
        rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
    }
    private void GroundCheck()
    {
        Collider2D[] collider = Physics2D.OverlapCircleAll(transform.position, 0.3f);

```

```

        isGrounded = collider.Length > 1;
        if(isGrounded) State = States.Jump;;
    }
    //public override void GetDamage()
    //{
    //    // lives--;
    //    //Debug.Log(lives);
    //}
    /* private void OnCollisionEnter2D(Collision2D collision)
    {
        /*lives--;
        Debug.Log("Количество жизни у персонажа: " + lives);
        if(lives==0)
        {
            Destroy(gameObject);
            SceneManager.LoadScene(2);
        }
        if (collision.gameObject.CompareTag("Worm2"))
        {
            Destroy(gameObject);
            SceneManager.LoadScene(1);
        }
    }
    */
    public enum States
    {
        Idle,
        Run,
        Jump,
        dopanim,
        dop2
    }
}

```

Файл Monster.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Scripting.APIUpdating;

public class Monster : Entity
{
    private float speed = 10f;
    private Vector3 dir;
    private int collisionCount = 0;
}

```

```

private SpriteRenderer sprite;

private void Start()
{
    dir = transform.right;
}

protected void Update()
{
    Move();
}

protected void Awake()
{
    sprite = GetComponentInChildren<SpriteRenderer>();
}

private void Move()
{
    Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position +
transform.up * 0.1F + transform.right * dir.x * 0.7F, 0.01F);

    if (colliders.Length > 0) dir *= -1F;

    transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, Time.deltaTime);
}

private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject == Hero.Instance.gameObject)
    {
        collisionCount++;

        if (collisionCount >= 3)
        {
            Hero.Instance.Die();
        }
    }
}
}

```

Файл RandomTree.cs

```

using System.Collections;

```

```

using System.Collections.Generic;
using UnityEngine;

public class RandomTree : MonoBehaviour
{
    public GameObject [] objects;

    void Start()
    {
        int rand=Random.Range(0,objects.Length);
        Instantiate(objects[rand], transform.position, Quaternion.identity);
    }
}

```

Файл Wood.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Wood : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject==Hero.Instance.gameObject)
        {
            Hero.Instance.GetDamage();
        }
    }
}

```

Файл Wood2.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Scripting.APIUpdating;

public class Wood2 : Entity
{
    private float speed = 10f;
    private Vector3 dir;
    private SpriteRenderer sprite;
}

```

```

private void Start()
{
    dir = transform.right;
}

protected void Update()
{
    Move();
}

protected void Awake()
{
    sprite = GetComponentInChildren<SpriteRenderer>();
}

private void Move()
{
    Vector3 targetPosition = Hero.Instance.transform.position;
    dir = (targetPosition - transform.position).normalized;
    Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position +
transform.up * 0.1F + transform.right * dir.x * 0.7F, 0.01F);

    if (colliders.Length > 0) dir *= -1F;

    transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, Time.deltaTime);
}

private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject == Hero.Instance.gameObject)
    {
        Hero.Instance.GetDamage();
    }
}
}

```

Файл Worm.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Worm : Entity

```

```

{
    [SerializeField] private int lives = 3;

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject==Hero.Instance.gameObject)
        {
            Hero.Instance.GetDamage();
            lives--;
            Debug.Log("червяк" + lives);
        }
        if (lives < 1) Die();
    }
}

```

Файл Worm2.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Worm2 : MonoBehaviour
{
    private Vector3 initialPosition;
    private float jumpHeight = 1f; // Высота прыжка
    private float jumpSpeed = 2f; // Скорость прыжка
    private float timeToJumpApex = 0.5f; // Время до достижения максимальной
    // высоты прыжка

    private float gravity;
    private float jumpVelocity;

    void Start()
    {
        initialPosition = transform.position;

        gravity = -(2 * jumpHeight) / Mathf.Pow(timeToJumpApex, 2);
        jumpVelocity = Mathf.Abs(gravity) * timeToJumpApex;

        StartCoroutine(Jump());
    }
}

```

```

IEnumerator Jump()
{
    while (true)
    {
        float timeInAir = 0f;
        while (timeInAir < timeToJumpApex)
        {
            float jumpDistance = jumpVelocity * Time.deltaTime;
            transform.position += new Vector3(0, jumpDistance, 0);
            timeInAir += Time.deltaTime;
            yield return null;
        }

        yield return new WaitForSeconds(0.1f); // Пауза перед следующим прыжком

        float timeDescending = 0f;
        while (timeDescending < timeToJumpApex)
        {
            float descendDistance = jumpVelocity * Time.deltaTime;
            transform.position -= new Vector3(0, descendDistance, 0);
            timeDescending += Time.deltaTime;
            yield return null;
        }

        yield return new WaitForSeconds(0.1f); // Пауза перед следующим прыжком
    }
}

```

Файл CanvasButton.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class CanvasButton : MonoBehaviour
{
    public void RestartGame()
    {
        SceneManager.LoadScene(0); //перезагрузка текущей сцены
    }
}

```

```

    }
    public void Menu()
    {
        Application.OpenURL("https://pnzgu.ru/");
    }
    public void Exit()
    {
        Application.Quit();
    }
}

```

Файл Entity.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Entity : MonoBehaviour
{
    public virtual void GetDamage()
    {

    }

    public virtual void Die()
    {
        Destroy(this.gameObject);
    }
}

```

Файл PlayerCatcherMove.cs

```

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

public class PlayerCatcherMove : MonoBehaviour
{
    [SerializeField] private Image[] lives;
    public float speed;
    private Rigidbody2D rb;
}

```

```

private float x;
private Vector2 move;
private int currentLives;

void Start()
{
    rb = GetComponent<Rigidbody2D>();
    currentLives = lives.Length; // Инициализация текущих жизней
}

void LoseLife()
{
    if (currentLives > 0)
    {
        currentLives--;
        lives[currentLives].gameObject.SetActive(false); // Делаем жизнь
неактивной
    }
}

private IEnumerator GameOverCoroutine()
{
    yield return new WaitForSeconds(1f); // Задержка перед перезагрузкой
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}

void FixedUpdate()
{
    x = Input.GetAxis("Horizontal");
    move = new Vector2(x * speed, rb.velocity.y);
    rb.velocity = move;
}

// Метод, который должен вызываться, когда игрок теряет жизнь
public void TakeDamage()
{
    LoseLife();
}
}

```

Файл Settings.cs

```

using System.Collections;
using System.Collections.Generic;

```

```

using Unity.VisualStudio.Antlr3.Runtime;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Settings : MonoBehaviour
{
    public Dropdown resolutionDropdown;
    public Dropdown qualityDropdown;

    Resolution[] resolutions;
    void Start()
    {
        resolutionDropdown.ClearOptions();
        List<string> options = new List<string>();
        resolutions = Screen.resolutions;
        int currentResolutionIndex = 0;

        for (int i = 0; i < resolutions.Length; i++)
        {
            string option = resolutions[i].width + "x" + resolutions[i].height + " " +
resolutions[i].refreshRateRatio + "Hz";
            options.Add(option);
            if (resolutions[i].width == Screen.currentResolution.width &&
resolutions[i].height == Screen.currentResolution.height)
                currentResolutionIndex = i;
        }

        resolutionDropdown.AddOptions(options);
        resolutionDropdown.RefreshShownValue();
        LoadSettings(currentResolutionIndex);
    }

    public void SetFullscreen(bool isFullscreen)
    {
        Screen.fullScreen = isFullscreen;
    }

    public void SetResolution(int resolutionIndex)
    {
        Resolution resolution = resolutions[resolutionIndex];
        Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
    }
}

```

```

public void SetQuality(int qualityIndex)
{
    QualitySettings.SetQualityLevel(qualityIndex);
}

public void ExitSettings()
{
    SceneManager.LoadScene("MenuCatcher");
}

public void SaveSattings()
{
    PlayerPrefs.SetInt("QualitySettingPreference", qualityDropdown.value);
    PlayerPrefs.SetInt("ResolutionPreference", resolutionDropdown.value);
    PlayerPrefs.SetInt("FullscreenPreference",
System.Convert.ToInt32(Screen.fullScreen));
}

public void LoadSettings(int currentResolutionIndex)
{
    if (PlayerPrefs.HasKey("QualitySettingPreference"))
        qualityDropdown.value = PlayerPrefs.GetInt("QualitySettingPreference");
    else
        qualityDropdown.value = 3;

    if (PlayerPrefs.HasKey("ResolutionReference"))
        resolutionDropdown.value = PlayerPrefs.GetInt("ResolutionReference");
    else
        resolutionDropdown.value = currentResolutionIndex;

    if (PlayerPrefs.HasKey("fullscreenPreference"))
        Screen.fullScreen =
System.Convert.ToBoolean(PlayerPrefs.GetFloat("fullscreenPreference"));
    else
        Screen.fullScreen = true;
}
}

```

Файл **timeDestroyer.cs**

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

```

```

public class timeDestroyer : MonoBehaviour
{
    //время жизни объекта
    public float aliveTimer;

    //исполняется 1 раз при запуске скрипта
    void Start()
    {
        //уничтожит текущий объект через заданное время
        Destroy(gameObject, aliveTimer);
    }
}

```

Файл Volume.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Volume : MonoBehaviour
{
    private AudioSource audioSrc;
    private float musicVolume = 1f;

    // Start is called before the first frame update
    void Start()
    {
        audioSrc = GetComponent<AudioSource>();
    }

    // Update is called once per frame
    void Update()
    {
        audioSrc.volume = musicVolume;
    }
    public void SetVolume(float vol)
    {
        musicVolume = vol;
    }
    public void StopMusic() // Метод для остановки музыки
    {
        audioSrc.Stop();
    }
}

```

```
}
```

Файл ItemChaker.cs

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

public class ItemChacker : MonoBehaviour
{
    [SerializeField] private UnityEngine.UI.Image[] lives;
    public AudioClip OkSound, BoomSound;
    public int score;
    public GameObject ScoreObject;
    private Text scoreText;
    public GameObject PanelWin; // Переменная для панели Выигрыша
    public GameObject GameOver; // Переменная для панели Game Over
    private int currentLives;

    void Start()
    {
        currentLives = lives.Length; // Инициализация текущего количества
        жизней
        for (int i = 5; i < currentLives; i++)
        {
            lives[i].gameObject.SetActive(false);
        }
        scoreText = ScoreObject.GetComponent<Text>();
        GameOver.SetActive(false); // Скрыть панель Game Over в начале
        PanelWin.SetActive(false); // Скрыть панель Выигрыша в начале
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Good")) // Проверяем тег "Good"
        {
            score += 10; // Увеличиваем счет
            scoreText.text = score.ToString(); // Обновляем текст счета
        }
    }
}
```

```

        AudioSource.PlayClipAtPoint(OkSound, other.transform.position); //
Пройгрываем звук
        Destroy(other.gameObject); // Удаляем объект
    }

    if (other.gameObject.CompareTag("Goodlife")) // Проверяем тег "Goodlife"
    {
        score += 10; // Увеличиваем счет
        scoreText.text = score.ToString(); // Обновляем текст счета
        AudioSource.PlayClipAtPoint(OkSound, other.transform.position); //
Пройгрываем звук
        GiveLife(); // Получаем жизнь
        Destroy(other.gameObject); // Удаляем объект
    }

    if (other.gameObject.CompareTag("Bad")) // Проверяем тег "Bad"
    {
        score -= 10; // Уменьшаем счет
        scoreText.text = score.ToString(); // Обновляем текст счета
        AudioSource.PlayClipAtPoint(BoomSound, other.transform.position); //
Пройгрываем звук
        Destroy(other.gameObject); // Удаляем объект
        LoseLife(); // Уменьшаем жизнь
    }
    // Проверяем условие выигрыша
    if (score >= 50)
    {
        ShowPanelWin();
    }
}

void LoseLife()
{
    if (currentLives > 0)
    {
        currentLives--;
        lives[currentLives].gameObject.SetActive(false); // Делаем жизнь
неактивной
    }

    if (currentLives <= 0)
    {
        GameOver.SetActive(true); // Активируем панель Game Over
        Debug.Log("Game Over");
    }
}

```

```

        Time.timeScale = 0; // Останавливаем игру
    }
}

void GiveLife()
{
    if (currentLives < lives.Length)
    {
        lives[currentLives].gameObject.SetActive(true); // Активируем
следующую жизнь
        currentLives++;
    }
}

void ShowPanelWin()
{
    PanelWin.SetActive(true); // Активируем панель Выигрыша
    Debug.Log("You Winner!");
    Time.timeScale = 0; // Останавливаем игру
}

public void RestartGame()
{
    Time.timeScale = 1; // Снимаем паузу
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex); //
Перезагружаем текущую сцену
}

// Метод для загрузки главного меню
public void LoadMainMenu()
{
    Time.timeScale = 1; // Снимаем паузу
    SceneManager.LoadScene("MenuCatcher"); // Замените на имя сцены
главного меню
}
}

```

Файл ItemSpawn.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ItemSpawn : MonoBehaviour
{

```

```

public GameObject RightSide;
public GameObject[] items;
public float startDelay, repeatRate;
private Animator anim;
private Rigidbody2D rb;
private SpriteRenderer sprite;

public static ItemSpawn Instance { get; set; }
private bool isGrounded = false;
private void Awake()
{
    sprite = GetComponentInChildren<SpriteRenderer>(); //Обращение к
спрайту
    anim = GetComponent<Animator>();
    rb = GetComponent<Rigidbody2D>();
    Instance = this;
}
private States State
{
    get { return (States)anim.GetInteger("state"); }
    set { anim.SetInteger("state", (int)value); }
}
private void myanim()
{
    if (isGrounded) State = States.myanim;
}
void Start()
{
    InvokeRepeating("Spawn", startDelay, repeatRate);
}
void Spawn()
{
    Vector3 pos = new Vector3(Random.Range(gameObject.transform.position.x,
RightSide.transform.position.x), gameObject.transform.position.y, 0);
    Instantiate(items[Random.Range(0, items.Length)], pos,
gameObject.transform.rotation);
}
public enum States
{
    myanim
}

```

```
}
```

Файл MainMenu.cs

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

public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
    public void OptionGame()
    {
        SceneManager.LoadScene("Settings");
    }

    public void ExitGame()
    {
        Debug.Log("Игра закрылась");
        Application.Quit();
    }
}
```

Файл PauseMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.SceneManagement;
public class PauseMenu : MonoBehaviour
{
    public bool Pausegame;
    public GameObject pauseGameMenu;
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (Pausegame)

```

```

        {
            Resume();
        }
        else
        {
            Pause();
        }
    }
}
public void Resume()
{
    pauseGameMenu.SetActive(false);
    Time.timeScale = 1f;
    Pausegame = false;
}
public void OptionMenuButton()
{
    SceneManager.LoadScene("Settings");
}
public void Pause()
{
    pauseGameMenu.SetActive(true);
    Time.timeScale = 0f;
    Pausegame = true;
}
public void LoadMenu()
{
    Time.timeScale = 1f;
    SceneManager.LoadScene("MenuCatcher");
}
}

```

Файл AsteriodHP.cs

```

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

public class AsteroidHP : MonoBehaviour
{
    public GameObject HealthBar;
    Image img;
    public int hp;
}

```

```

float maxHp;

private ScoreManager scoreManager;
void Start()
{
    img = HealthBar.GetComponent<Image>();
    maxHp = hp;
    img.fillAmount = hp / maxHp;
    scoreManager=FindObjectOfType<ScoreManager>();
}

void MakeDamage(int damage)
{
    hp = hp - damage;

    if (hp <= 0)
    {
        scoreManager.IncreaseScore(5);
        Destroy(gameObject);
    }
    img.fillAmount = hp / maxHp;
}
}

```

Файл AsteroidMove.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class AsteroidMove : MonoBehaviour
{
    public float speed;
    Rigidbody2D rb;
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        Vector3 move = new Vector3(0, -1, 0);
        rb.velocity = move * speed;
    }

    void OnBecameInvisible()
    {

```

```

        Destroy(gameObject);
    }
    // Update is called once per frame
    void OnCollisionEnter2D(Collision2D something)
    {
        if (something.gameObject.tag == "Player")
        {
            Destroy (gameObject);
            SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        }
    }
}

```

Файл EnemyBullet.cs

```

using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class EnemyBullet : MonoBehaviour
{
    GameObject player;
    Rigidbody2D rb;
    public float force;
    public int damage;
    public AudioClip OkSound, BoomSound;
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        player = GameObject.FindWithTag("Player");
        if (player != null)
        {
            Vector3 dir = player.transform.position - transform.position;
            float angle = Mathf.Atan2(dir.x, dir.y) * Mathf.Rad2Deg;
            transform.Rotate(0, 0, angle);
            rb.AddForce(dir.normalized * force, ForceMode2D.Impulse);
        }
        else
            Destroy(gameObject);
    }

    // Update is called once per frame

```

```

void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.tag == "Player")
    {
        AudioSource.PlayClipAtPoint(BoomSound, other.transform.position); //
        Проигрываем звук
        other.gameObject.SendMessage("MakeDamage", damage,
        SendMessageOptions.DontRequireReceiver);
        Destroy(gameObject);
    }
}

void OnBecameInvisible()
{
    Destroy(gameObject);
}
}

```

Файл EnemyShoot.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using Unity.Mathematics;
using UnityEngine;

public class EnemyShoot : MonoBehaviour
{
    public GameObject bullet;
    public float fireDelay;
    GameObject player;
    bool canShoot = true;
    private ScoreManager scoreManager;
    public int hp;
    float maxHp;
    void Start()
    {
        player = GameObject.FindWithTag("Player");
        scoreManager = FindObjectOfType<ScoreManager>();
    }

    // Update is called once per frame

```

```

void Update()
{
    if (canShoot && player != null)
    {
        canShoot= false;
        Instantiate(bullet, transform.position, Quaternion.identity);
        StartCoroutine(firePause());
    }
}
IEnumerator firePause()
{
    yield return new WaitForSeconds(fireDelay);
    canShoot = true;
}

internal bool IsDead()
{
    throw new NotImplementedException();
}
void MakeDamage(int damage)
{
    hp = hp - damage;

    if (hp <= 0)
    {
        scoreManager.IncreaseScore(5);
        Destroy(gameObject);
    }
    //img.fillAmount = hp / maxHp;
}
}

```

Файл HPController.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HpController : MonoBehaviour
{
    public int hp;
    void MakeDamage(int damage)
    {
        hp = hp - damage;
    }
}

```

```

        if (hp <= 0)
        {
            Destroy(gameObject);
        }
    }

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

    }
}

```

Файл LaserShot.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class LaserShot : MonoBehaviour
{
    Rigidbody2D rb;
    public int damage;
    public float force;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        Vector3 direction = new Vector3(0, force, 0);
        rb.AddForce(direction, ForceMode2D.Impulse);
    }

    void OnBecameInvisible()
    {
        Destroy(gameObject);
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.tag == "Enemy")
        {
            other.gameObject.SendMessage("MakeDamage", damage,
            SendMessageOptions.DontRequireReceiver);
            Destroy(gameObject); // Уничтожаем лазер
        }
    }
}

```

```

    }
}

private int hitCount = 0;

}

```

Файл **Object.Spawner.cs**

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ObjectSpawner : MonoBehaviour
{
    public Transform RightPosition;
    public float spawnDelay;
    public GameObject[] Item;
    // Start is called before the first frame update
    void Start()
    {
        InvokeRepeating("Spawn", spawnDelay, spawnDelay);
    }

    // Update is called once per frame
    void Spawn()
    {
        Vector3 spawnPos = new Vector3(Random.Range(transform.position.x,
            RightPosition.position.x), transform.position.y, 0);
        int i = Random.Range(0, Item.Length);
        Instantiate(Item[i], spawnPos, transform.rotation);
    }
}

```

Файл **OverShooter.cs**

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class OverShooter : MonoBehaviour

```

```

{
    public void GameOver()
    {
        SceneManager.LoadScene(4); //перезагрузка текущей сцены
    }
    public void RestartGame()
    {
        SceneManager.LoadScene(3); // Загружаем сцену 3
    }
}

```

Файл PlayerHP.cs

```

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

public class PlayerHP : MonoBehaviour
{
    public GameObject HealthBar;
    Image img;
    public int hp;
    float maxHp;

    void Start()
    {
        img = HealthBar.GetComponent<Image>();
        maxHp = hp;
        img.fillAmount = hp / maxHp;
    }
    public void GameOver()
    {
        SceneManager.LoadScene(4); // Перезагрузка сцены 3
    }

    // Метод для получения урона

    void MakeDamage(int damage)
    {
        hp = hp - damage;

        if (hp <= 0)

```

```

    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
    if (hp <= 0)
    {
        GameOver(); // Вызов метода GameOver при смерти игрока
    }
    img.fillAmount = hp / maxHp;
}
}

```

Файл PlayerMove.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMove : MonoBehaviour
{
    Vector3 clickPos;
    Vector3 move;
    private Animator anim;
    private SpriteRenderer sprite;

    private States State
    {
        get { return (States)anim.GetInteger("state"); }
        set { anim.SetInteger("state", (int)value); }
    }

    public float speed = 1;
    Rigidbody2D rb;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>(); // Не забудьте получить Animator
        clickPos = transform.position;
    }

    void Update()
    {
        if (Input.GetMouseButton(0))
        {

```

```

        clickPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    }

    move = clickPos - transform.position;

    // Вызов анимации при нажатии клавиши C
    if (Input.GetKeyDown(KeyCode.C))
    {
        State = States.Shooteranim;
    }
    else if (State == States.Shooteranim)
    {
        State = States.idleShooter; // Возвращаем в состояние idleShooter после
        // выполнения
    }
}

void FixedUpdate()
{
    rb.velocity = new Vector2(move.x, move.y) * speed;
}

public enum States
{
    Shooteranim,
    idleShooter
}
}

```

Файл PlayerShoot.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerShoot : MonoBehaviour
{
    public GameObject laser;
    public float delayTime;
    bool canShoot = true;

    void Update()
    {
        if (canShoot && Input.GetMouseButton(1))

```

```

    {
        canShoot = false;
        Instantiate(laser, transform.position, transform.rotation);
        StartCoroutine(NoFire());
    }
}

IEnumerator NoFire()
{
    yield return new WaitForSeconds(delayTime);
    canShoot = true;
}
}

```

Файл PlayerSpawner.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerSpawner : MonoBehaviour
{
    public Transform RightPosition;
    public GameObject Item;
    void Start()
    {
        Vector3 spawnPos = new Vector3(Random.Range(transform.position.x,
            RightPosition.position.x), transform.position.y, 0);
        Instantiate(Item, spawnPos, transform.rotation);
    }
}

```

Файл ScoreManager.cs

```

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

public class ScoreManager : MonoBehaviour

```

```

{
    public static int score = 0;
    public Text scoreText;

    void Update()
    {
        scoreText.text = " " + score;

        if (score >= 50)
        {
            SceneManager.LoadScene(5);
        }
    }

    public void IncreaseScore(int amount)
    {
        score += amount;
    }
    public void RestartButton()
    {
        SceneManager.LoadScene(3); // Загружаем сцену 3
    }
}

```

Файл WinShooter.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class WinShooter : MonoBehaviour
{
    public void RestartButton()
    {
        SceneManager.LoadScene(3); // Загружаем сцену 3
    }
    public void WinnerButton()
    {
        SceneManager.LoadScene(5); //перезагрузка текущей сцены
    }
}

```

Файл MenuShooter.cs

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

public class MenuShooter : MonoBehaviour
{
    public void RestartGame()
    {
        SceneManager.LoadScene("Shooter");
    }
    public void Exit()
    {
        SceneManager.LoadScene("FinishGame");
    }
}
```

Файл Finish.cs

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

public class Finish : MonoBehaviour
{
    public void Platformer()
    {
        SceneManager.LoadScene("Menu");
    }
    public void Catcher()
    {
        SceneManager.LoadScene("MenuCatcher");
    }
    public void Shooter()
    {
        SceneManager.LoadScene("MenuShooter");
    }
}
```