Министерство образования и науки Российской Федерации

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

«Утверждаю»
Зав. кафедрой «МО и ПЭВМ»
_____ Макарычев П.П.
« » _____ 2018

Пояснительная записка

к курсовому проектированию по дисциплине
«Объектно-ориентированное программирование»

на тему: «Разработка программы сложной структуры методом нисходящего программирования»

Автор работы:	Угроватов Д.В.
Направление бакалавриата	09.03.04 («Программная инженерия»)
Обозначение курсовой работы	ПГУ 09.03.04 - 04КП161.24 ПЗ
Группа	16ВП1
Руководитель работы	Афонин А.Ю., к.т.н.
Работа защищена «__» ____ 2018 г.	Оценка _____

Пенза 2018 г.

Объем работы по проекту

1. Расчетная часть

1. Анализ требований к разработке программного обеспечения
2. Проектирование программы
3. Разработка программного обеспечения
4. Тестирование программного обеспечения

2. Графическая часть

1. Диаграмма вариантов использования
2. Диаграмма классов
3. Диаграмма компонентов

3. Экспериментальная часть

1. Подготовка набора тестовых данных
2. Отладка и тестирование программного обеспечения

Срок выполнения проекта по разделам

1. Анализ требований к разработке ПО	к	15.03.18
2. Проектирование программы	к	01.04.18
3. Разработка программного обеспечения	к	01.05.18
4. Тестирование программного обеспечения	к	15.05.18
5. Оформление пояснительной записки	к	28.05.18
6. Защита курсовой работы	к	24.05.18
	к	

Дата выдачи задания « 10» февраля 2018

Дата защиты проекта « »

Руководитель

Задание получил «10» февраля 2018 г.

Студент

Оглавление

Введение

В рамках данной курсовой работы было решено создать игру в жанре 2D (от англ. Dimension - измерение) файтинг (от англ. Fighting – бой, драка). Это жанр компьютерных игр, имитирующих рукопашный бой малого числа персонажей (как правило двух) в пределах ограниченного пространства, называемого ареной. Важной особенностью файтингов является их нацеленность на соревнование, а не на сотрудничество игроков, что делает игры этого жанра подходящими для киберспортивных чемпионатов.

Файтинги, являясь аркадным жанром, получили распространение и пользовались популярностью преимущественно в тех странах, где присутствовала развитая сеть игровых залов с аркадными автоматами, в первую очередь в США и Японии.

Цель игры – довести уровень здоровья противника до нуля. Персонажи могут передвигаться по вертикали и горизонтали, наносить удары противнику и защищаться.

Управлять персонажами может как пользователь, так и компьютер. Данная особенность так же будет присутствовать в данном проекте.

1. Постановка задачи и анализ предметной области

1.1. Анализ требований

1.1.1. Требования к интерфейсу пользователя

Приложение должно предоставлять пользователю удобный интерфейс, позволяющий:

- выбирать режим игры (игрок против игрока; игрок против компьютера; компьютер против компьютера);
- выбирать фон «игровой площадки»;
- изменять разрешение приложения;
- изменять уровень громкости приложения;
- ознакомиться с управлением персонажами;

1.1.2. Требования к программным средствам

Анализ задания на разработку позволяет выделить следующие варианты использования (рисунок 1).

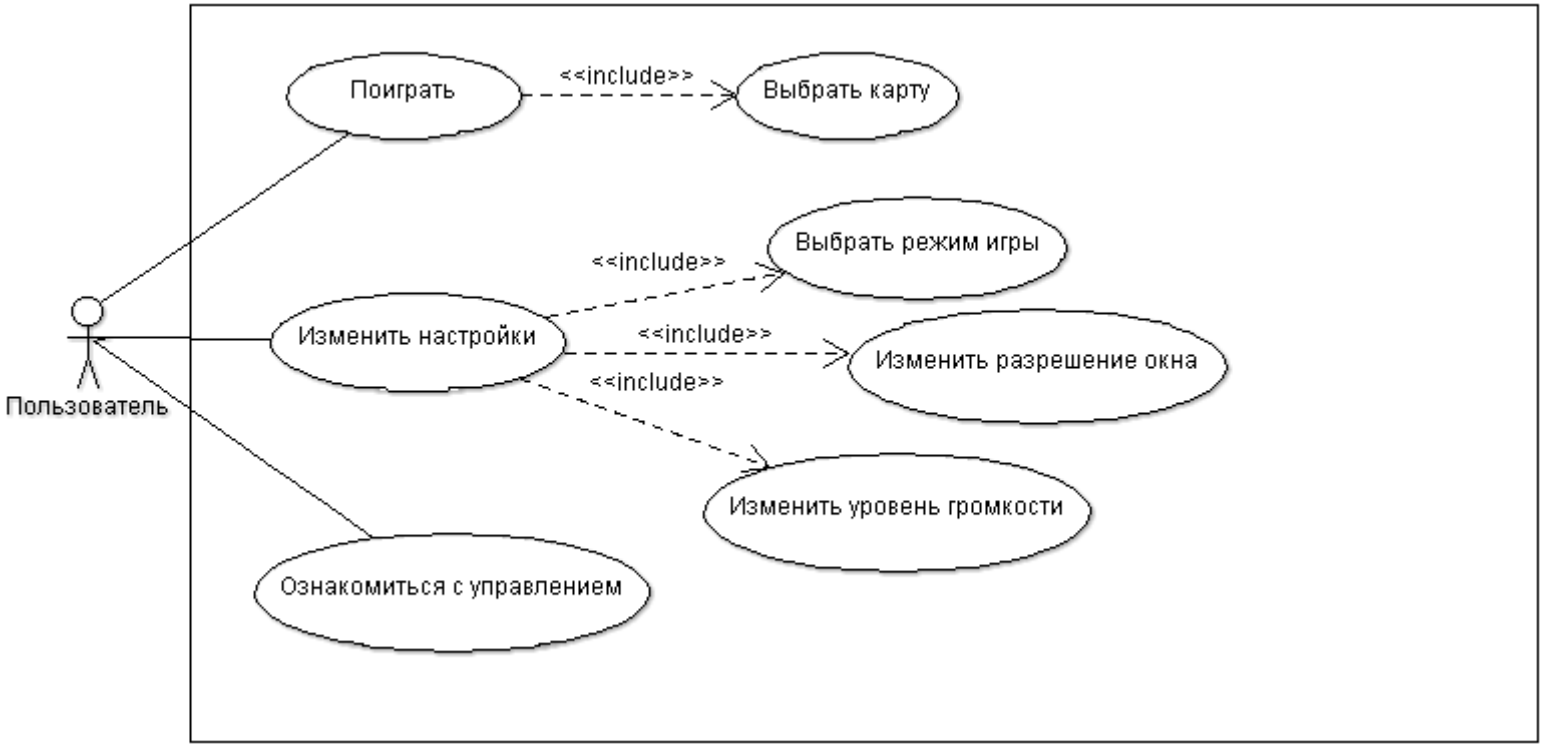

Рисунок 1 - Диаграмма вариантов использования

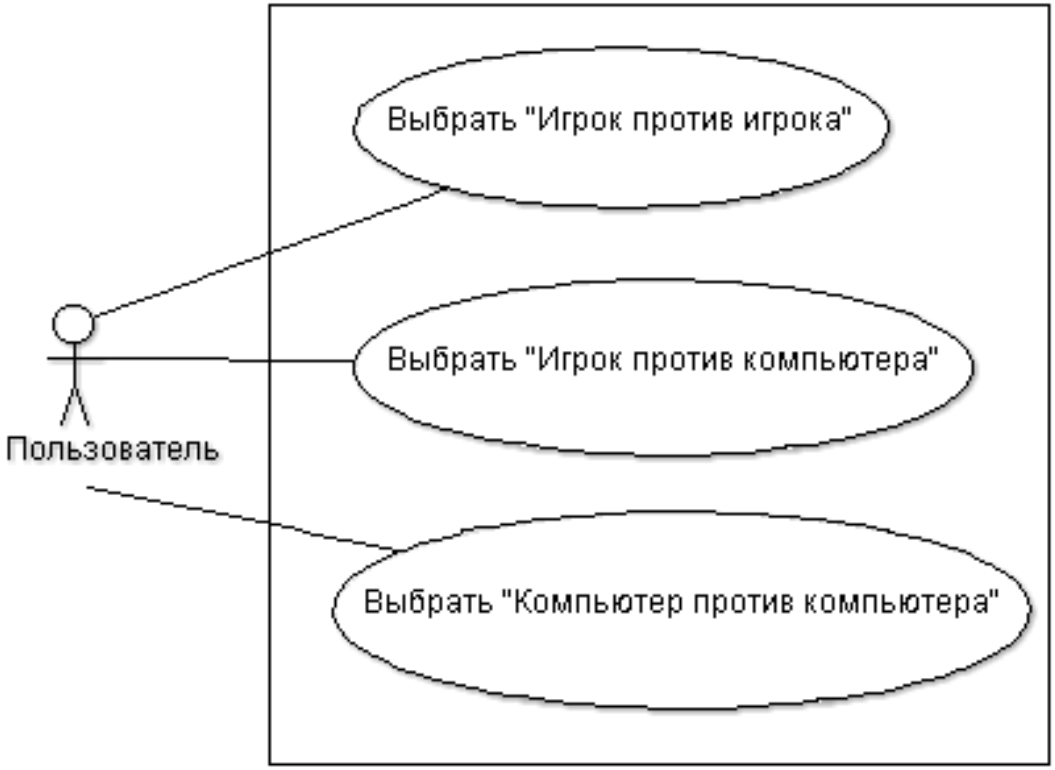

Рисунок 2 - уточнение варианта «Выбрать режим игры»

Опишем спецификацию нескольких прецедентов. Сначала рассмотрим спецификацию прецедента «Поиграть» без прецедента «Изменить настройки» (Таблица 1):

Таблица 1 – спецификация прецедента «Поиграть»

Прецедент: Поиграть
ID: 1
Краткое описание: Запуск игровой сессии.
Действующие лица: Пользователь
Предусловие: Пользователь вошел в систему
Основной поток: 1. Пользователь выбирает «Начать игру» 1.1. Пользователь выбирает режим игры «Игрок против компьютера» 1.2. Пользователь выбирает карту №1 2. Начинается игровой процесс 3. Если побеждает пользователь 3.1. Система отображает надпись о победе Пользователя 4. Иначе 4.1. Система отображает надпись о победе компьютера
Постусловие: Пользователь перемещен на экран главного меню

Теперь опишем спецификацию прецедента «Поиграть» с прецедентом «Изменить настройки» (Таблица 2):

Таблица 2 – спецификация прецедента «Поиграть» с прецедентом «Изменить настройки»

Прецедент: Поиграть
ID: 2
Краткое описание: Запуск игровой сессии и изменение настроек.
Действующие лица: Пользователь
Предусловие: Пользователь вошел в систему
Основной поток: 1. Пользователь выбирает «Настройки»

1.1. Система перемещает пользователя в меню настроек 1.2. Пользователь изменяет разрешение экрана 1.3. Пользователь изменяет уровень громкости 1.4. Пользователь выбирает «Назад» 1.5. Система возвращает пользователя в главное меню 2. Пользователь выбирает «Начать игру» 2.1. Пользователь выбирает режим игры «Игрок против компьютера» 2.2. Пользователь выбирает карту №1 3. Начинается игровой процесс 4. Если побеждает пользователь 4.1. Система отображает надпись о победе Пользователя 5. Иначе 5.1. Система отображает надпись о победе компьютера
Постусловие: Пользователь перемещен на экран главного меню

1.2. Технология разработки программного обеспечения

Для разработки ПО использованы следующие технологии разработки:

1) RAD – для разработки интерфейса

2) Каскадная методология

Каскадная модель (англ. waterfall model) – модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.

В каскадной модели этапы проектирования постепенно проходят в следующем порядке:

1) Проектирования

2) Дизайн

3) Кодирование

4) Тестирование

5) Поддержка

Визуальное программирование – способ создания программы для ЭВМ путём манипулирования графическими объектами вместо написания её текста. Наглядным примером средства визуальной разработки служит «Microsoft Visual Studio».

Визуальное программирование в основном используется для создания программ с графическим интерфейсом для ОС с графическим интерфейсом пользователя.

2. Проектирование программы

2.1. Модель интерфейса

Опираясь на ранее составленную диаграмму использования (рис. 1), был спроектирован следующий дизайн интерфейса (рис. 3-9):

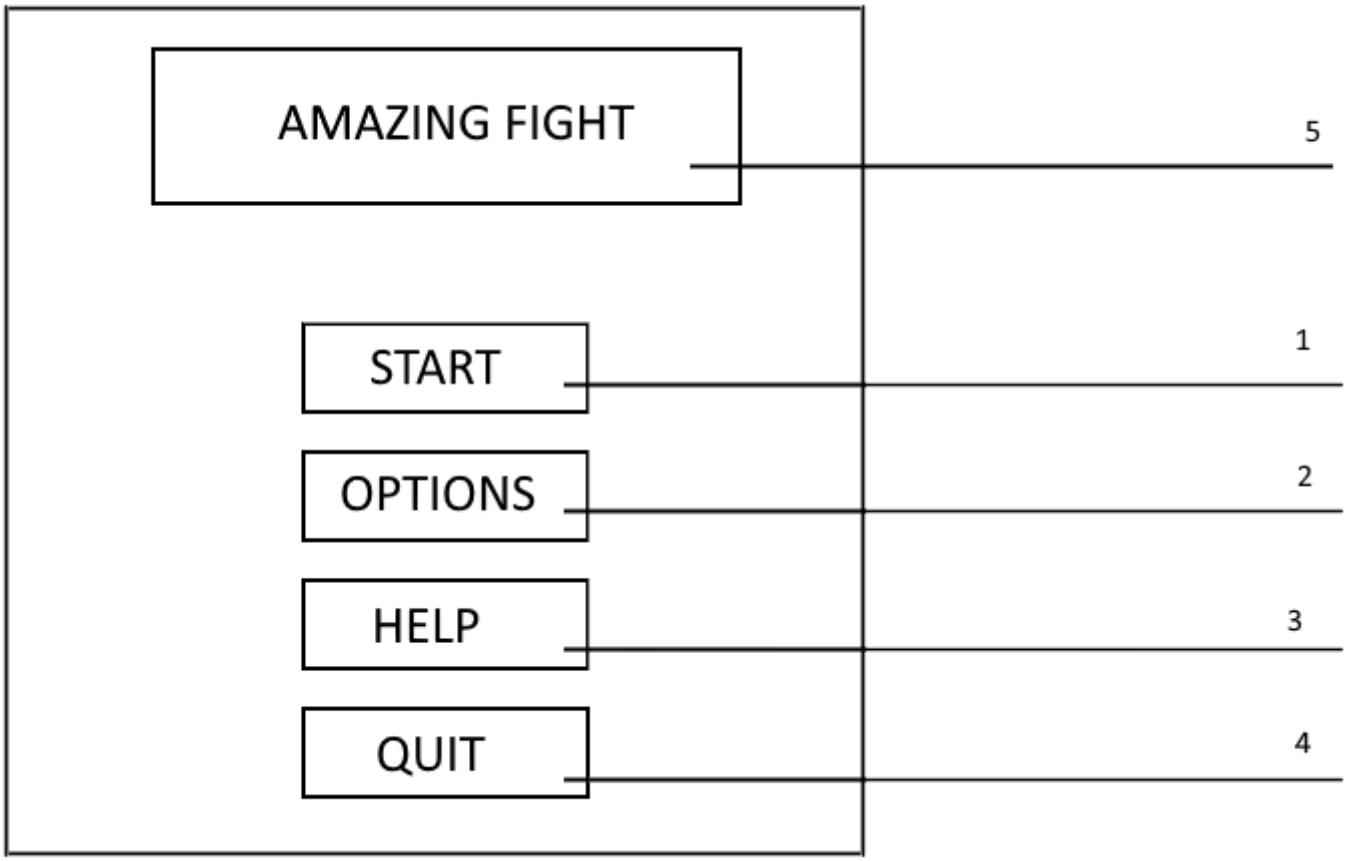

Рисунок 3 – дизайн главного меню

1) Кнопка «Начать игру» (переход в меню выбора карты)

2) Кнопка «Настройки» (переход в меню настроек)

3) Кнопка «Управление» (переход в меню управлению)

4) Кнопка «Выйти из игры» (завершение работы программы)

5) Название игры

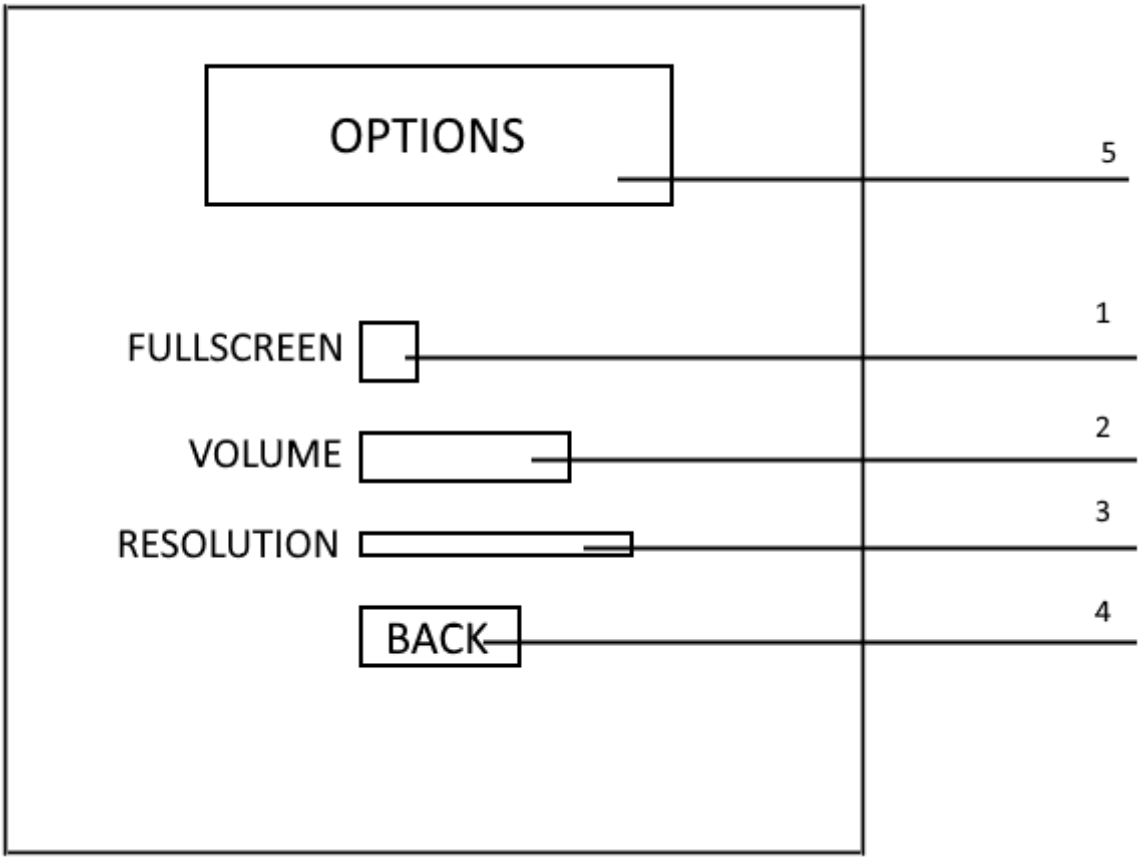

Рисунок 4 – дизайн меню настроек

1) Переключатель «Изменение полноэкранного режима» (устанавливает полноэкранный режим)

2) Слайдер «Изменение громкости» (изменяет уровень громкости звуков программы)

3) Выпадающий список «Разрешение» (изменяет разрешение окна программы)

4) Кнопка «Назад» (возвращает в главное меню)

5) Название меню

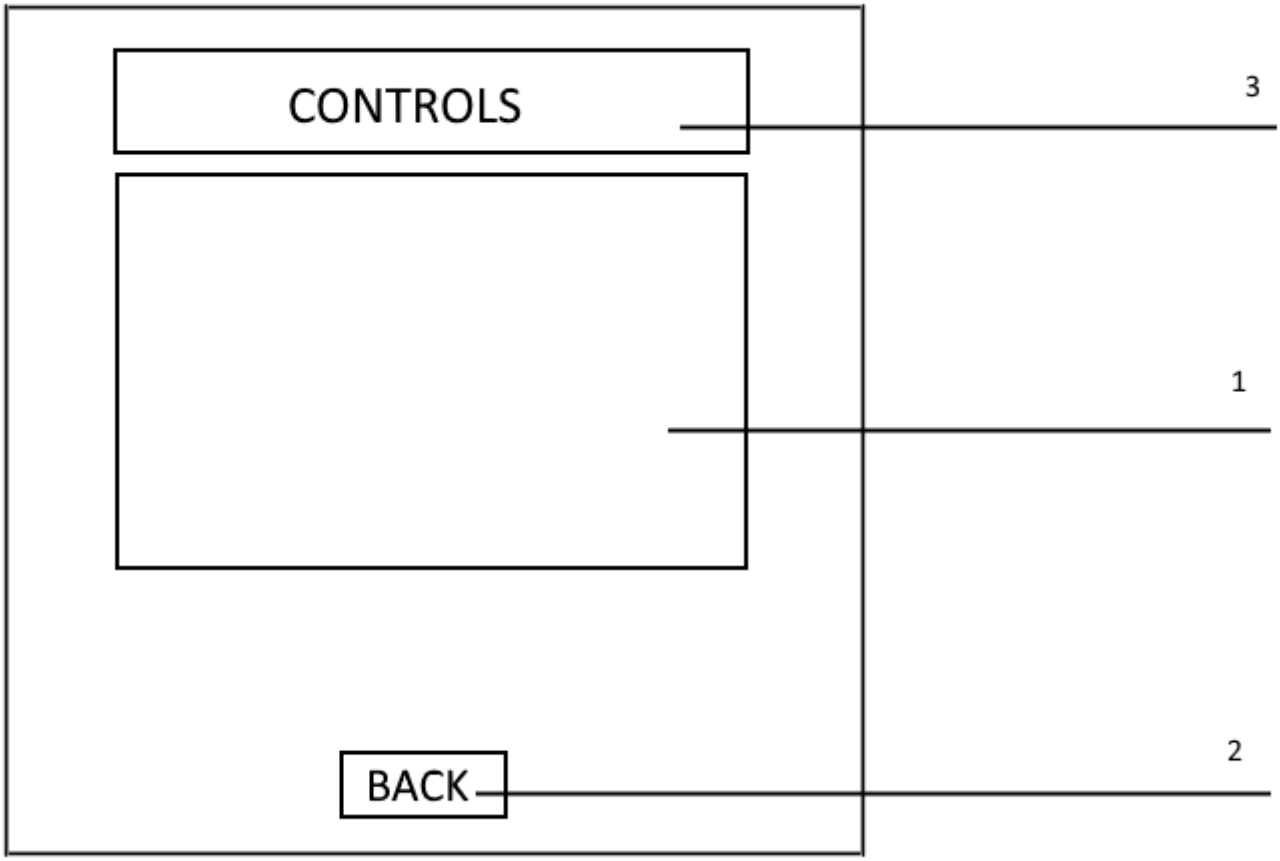

Рисунок 5 – дизайн меню управления

1) Область с описанием управления

2) Кнопка «Назад» (возвращает в главное меню)

3) Название меню

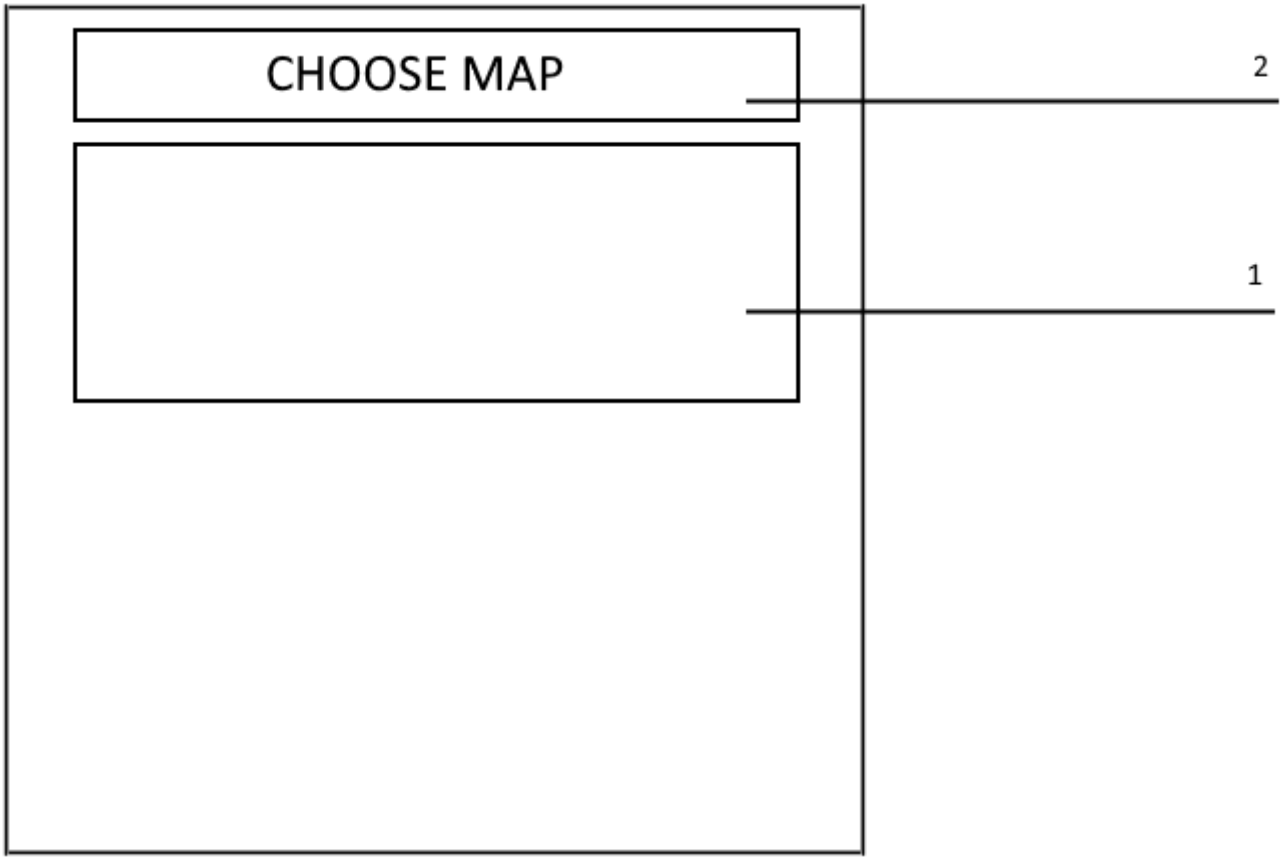

Рисунок 6 – дизайн меню выбора карты

1) Область с выбором карты (после выбора переносит в меню выбора режима)

2) Название меню

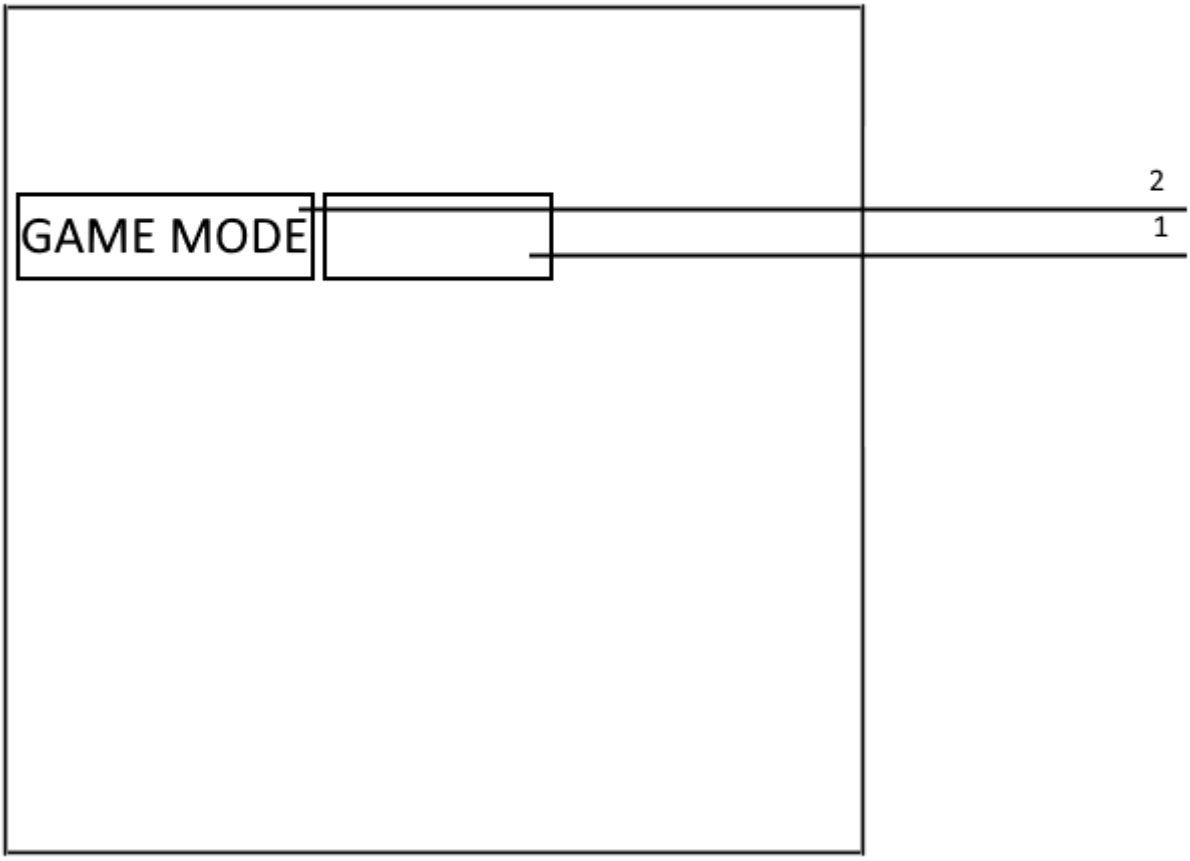

Рисунок 7 – дизайн меню выбора режима игры

1) Выпадающий список с режимами игры (после выбора переносит на игровое поле)

2) Название меню

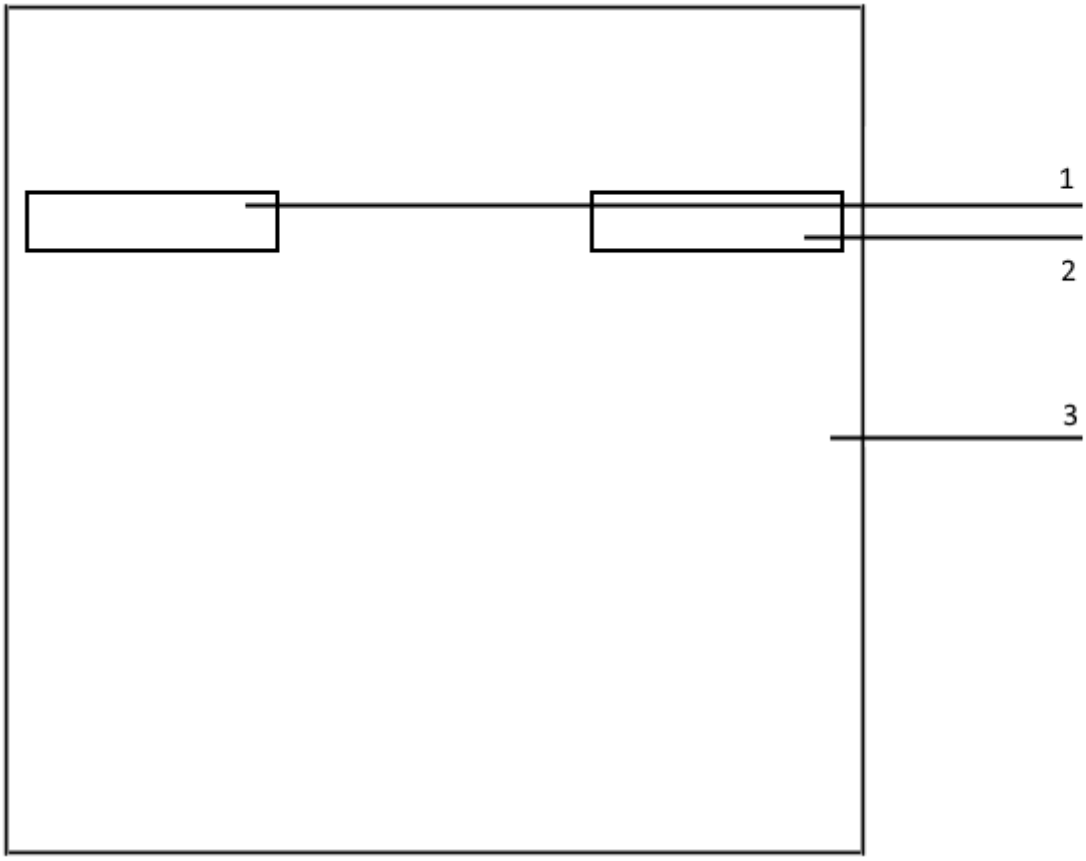

Рисунок 8 – дизайн игрового поля

1) Индикатор здоровья первого игрока

2) Индикатор здоровья второго игрока

3) Игровое поле

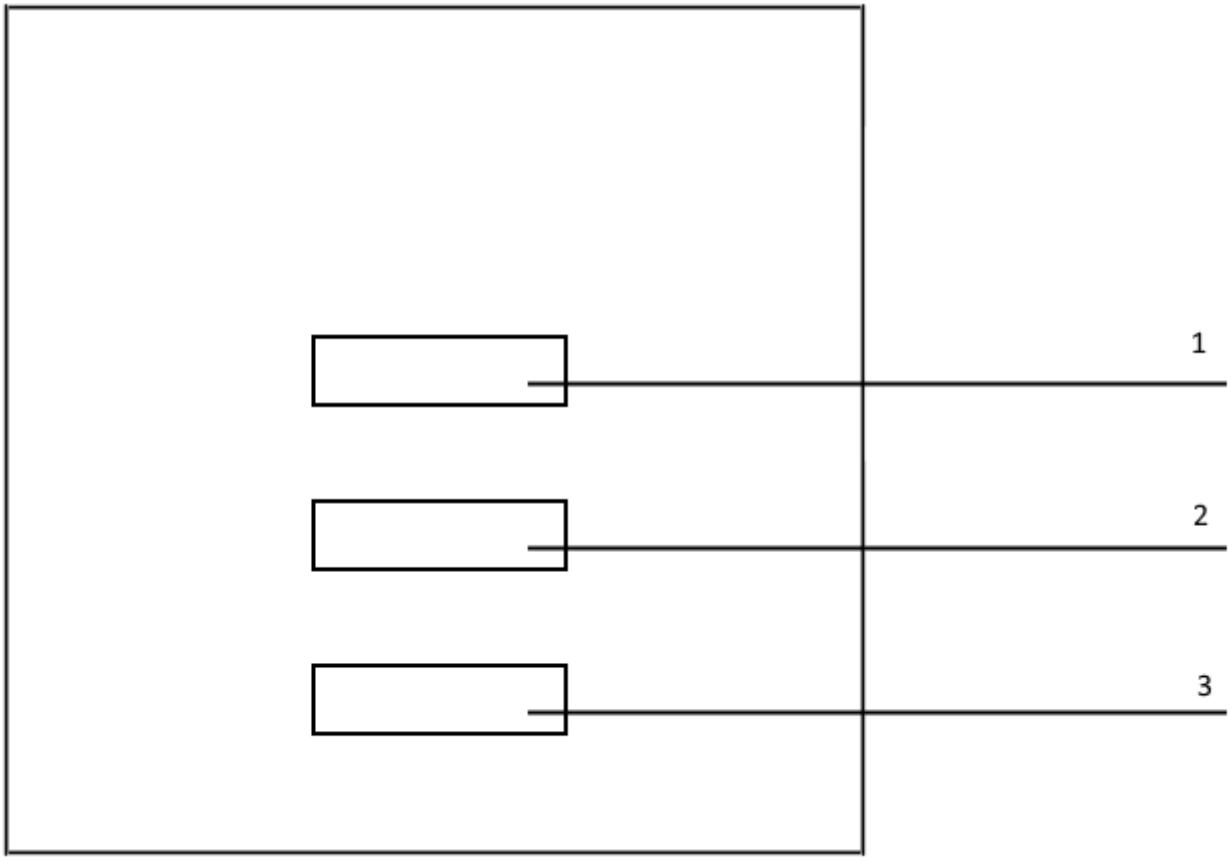

Рисунок 9 – дизайн меню паузы

1) Кнопка «Продолжить» (возобновляет игру)

2) Кнопка «Главное меню» (переносит в главное меню, прерывает игру)

3) Кнопка «Выйти из игры» (завершение работы программы)

2.2. Структура программного обеспечения

На рисунке 10 изображена диаграмма классов:

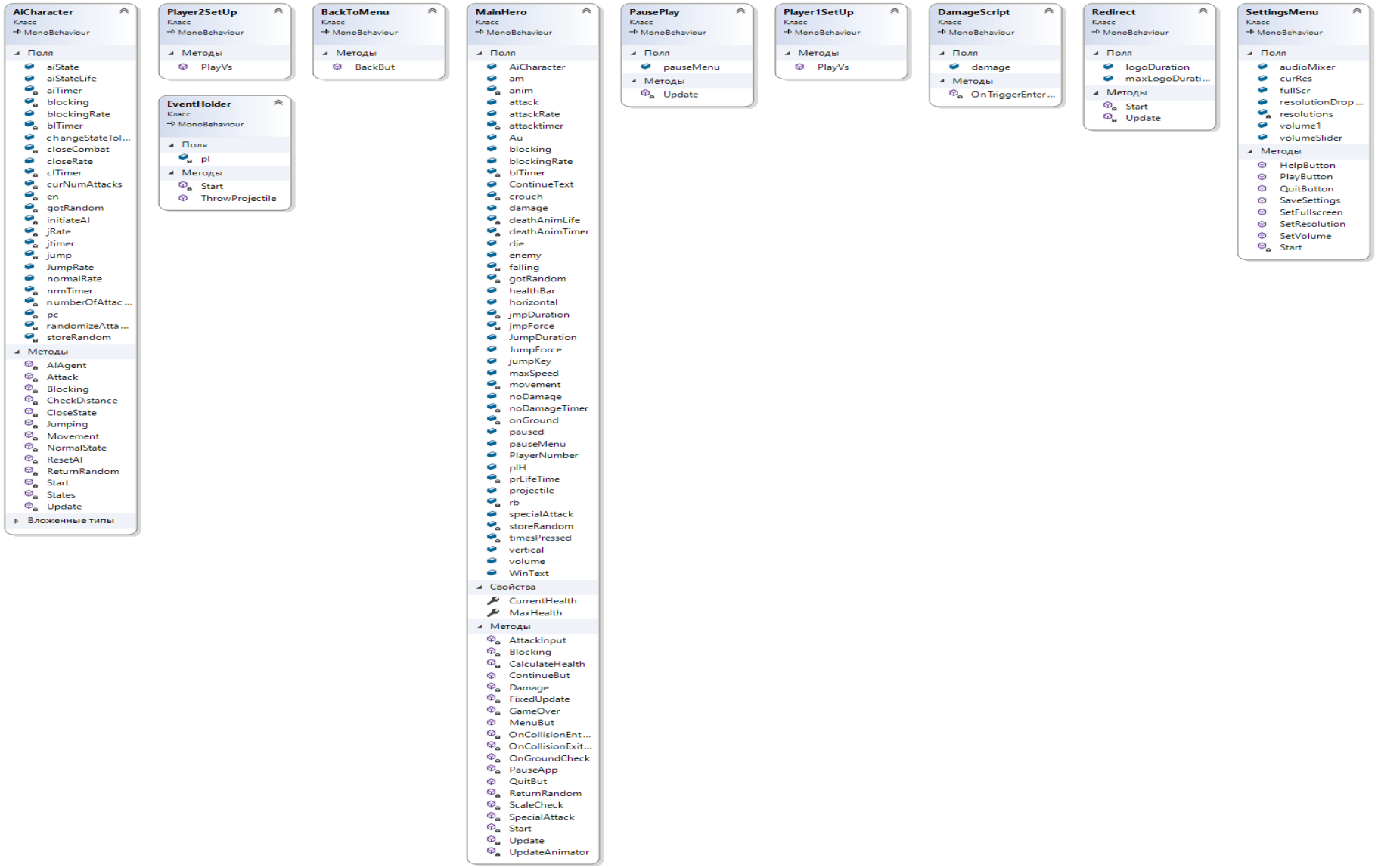

Рисунок 10 – диаграмма классов

Продемонстрируем спецификацию класса на примере AiCharacter:

Сущность: система приложения

Назначение: реализует управление персонажем искусственным интеллектом

Поля:

+ aiState – Структура состояний: малая дистанция, большая дистанция, перезагрузка состояния : *структура*

+ aiStateLife – Максимальная длительность состояния : *с плав. т.*

- aiTimer – Длительность состояния : *с плав. т.*

- blocking – Сработал ли блок : *булев.*

+ blockingRate – Максимальная длительность блока : *с плав. т.*

- blTimer – Длительность блока : *с плав. т.*

+ changeStateTolerance – Дистанция изменения состояния : *с плав. т.*

- closeCombat – На малой ли дистанции к противнику : *булев.*

+ closeRate – Максимальное время нахождения на малой дистанции : *с плав. т.*

- clTimer – Время нахождения на малой дистанции : *с плав. т.*

- curNumAttacks – Количество произведенных атак : *целочисл.*

- en – Противник

- gotRandom – Получено ли случайное число : *булев.*

- initiateAI – Спровоцирован ли : *булев.*

- jRate – Шанс прыжка : *с плав. т.*

- jtimer – Время без прыжка : *с плав. т.*

- jump – Прыгнул ли : *булев.*

+ JumpRate – Частота прыжка : *с плав. т.*

+ normalRate – Максимальное время нахождения на большой дистанции : *с плав. т.*

- nrmTimer – Время нахождения на большой дистанции : *с плав. т.*

- numberOfAttacks – Количество сгенерированных атак : *целочисл.*

- pc – Управляемый персонаж

- randomizeAttacks – Сгенерирована ли атака : *булев.*

- storeRandom – Случайное число : *с плав. т.*

Методы:

- AIAgent – Изменить состояния искусственного интеллекта (далее –ИИ)

- Attack – Атаковать противника

- Blocking – Заблокировать атаку

- CheckDistance – Проверить дистанцию до противника

- CloseState – Засечь время нахождения на малой дистанции от противника

- Jumping - Прыгнуть

- Movement - Передвинуться

- NormalState – Засечь время нахождения на большой дистанции от противника

- ResetAI – Обновить состояние ИИ

- ReturnRandom – Сгенерировать случайное число : *целочисл.*

- Start – Системная функция, срабатывает при запуске программы

- States – Устанавливает состояния

- Update – Системная функция, срабатывает каждый тик

Подробнее разберем работу данного класса:

```
public class AiCharacter : MonoBehaviour
{
    MainHero pc;                                    //управляемый персонаж
    MainHero en;                                    //персонаж противника
    public float changeStateTolerance = 3;      // Дистанция изменения состояния
    public float normalRate = 1;         // Максимальное время нахождения на большой дистанции
    float nrmTimer;                                 // nrmTimer
    public float closeRate = 0.5f;       // Максимальное время нахождения на малой дистанции
    float clTimer;                                  // Время нахождения на малой дистанции
    public float blockingRate = 1.5f;               // Максимальная длительность блока
    float blTimer;                                  // Длительность блока
    public float aiStateLife = 1;                   // Максимальная длительность состояния
    float aiTimer;                                  // Длительность состояния
    bool initiateAI;                                // Спровоцирован ли
    bool closeCombat;                               // На малой ли дистанции к противнику
    bool gotRandom;                                 // Получено ли случайное число
    float storeRandom;                              // Случайное число
    bool blocking;                                  // Сработал ли блок
    bool randomizeAttacks;                          // Сгенерирована ли атака
    int numberOfAttacks;                            // Количество сгенерированных атак
    int curNumAttacks;                              // Количество произведенных атак
    public float JumpRate = 1;                      // Частота прыжка
    float jRate;                                    // Шанс прыжка
```

```
bool jump;                                  // Прыгнул ли
float jtimer;                               // Время без прыжка
public enum AIState
{
    closeState,
    normalState,
    resetAI
}
public AIState aiState;
```

Выше были объявлены все необходимые переменные. Далее идет функция `Start()`, в которой определяется противник.

```
void Start()
{
    pc = GetComponent<MainHero>();
}
```

В следующей функции происходит постоянное обновление и применение основых методов данного класса:

```
void Update()
{
    if (!en)
    {
        en = pc.enemy.GetComponent<MainHero>();
    }

    CheckDistance();
    States();
    AIAgent();
}
```

Функция `States()` устанавливает одно из состояний ИИ, а так же позволяет прыгать и блокировать удар.

```
void States()
{
    switch (aiState)
    {
        case AIState.closeState:
            CloseState();
            break;
        case AIState.normalState:
            NormalState();
            break;
        case AIState.resetAI:
            ResetAI();
            break;
    }
    Blocking();
    Jumping();
}
```

Функция `AIAgent()`отвечаеть за передвижение и атаку. Заметим, что если персонаж некоторое время находится на расстоянии от противника, то шанс изменить позицию увеличивается.

```
void AIAgent()
{
    if (initiateAI)
```

```
    {
        aiState = AIState.resetAI;
        float multiplier = 0;

        if (!gotRandom)
        {
            storeRandom = ReturnRandom();
            gotRandom = true;
        }

        if (!closeCombat)
        {
            multiplier += 30;
        }
        else
        {
            multiplier -= 30;
        }

        if (storeRandom + multiplier < 50)
        {
            Attack();
        }
        else
        {
            Movement();
        }
    }
}
```

В функции `Attack()` генерируются атаки. До с вероятностью 75% будут сгенерированы обычные атаки и с 25% специальная.

```
void Attack()
{
    if (!gotRandom)
    {
        storeRandom = ReturnRandom();
        gotRandom = true;
    }
    if (storeRandom < 75)
    {
        if (!randomizeAttacks)
        {
            numberOfAttacks = (int)Random.Range(1, 4);
            randomizeAttacks = true;
        }
        if (curNumAttacks < numberOfAttacks)
        {
            int attackNumber = Random.Range(0, pc.attack.Length);
            pc.attack[attackNumber] = true;
            curNumAttacks++;
        }
    }
    else
    {
        if (curNumAttacks < 1)
        {
            pc.specialAttack = true;
            curNumAttacks++;
        }
    }
}
```

Функция `Movement()` отвечает за передвижение. С вероятностью 90% персонаж будет идти к противнику, и, соответственно, с 10% вероятностью от него.

```
void Movement()
{
    if (!gotRandom)
    {
        storeRandom = ReturnRandom();
        gotRandom = true;
    }
    if (storeRandom < 90)
    {
        if (pc.enemy.position.x < transform.position.x)
            pc.horizontal = -1;
        else
            pc.horizontal = 1;
    }
    else
    {
        if (pc.enemy.position.x < transform.position.x)
            pc.horizontal = 1;
        else
            pc.horizontal = -1;
    }
}
```

В функции `ResetAI()` происходит изменение состояния. С вероятностью 50% оно изменится на маленькую дистанцию или большую.

```
void ResetAI()
{
    aiTimer += Time.deltaTime;
    if (aiTimer > aiStateLife)
    {
        initiateAI = false;
        pc.horizontal = 0;
        pc.vertical = 0;
        aiTimer = 0;
        gotRandom = false;
        storeRandom = ReturnRandom();
        if (storeRandom < 50)
            aiState = AIState.normalState;
        else
            aiState = AIState.closeState;
        curNumAttacks = 1;
        randomizeAttacks = false;
    }
}

void CheckDistance()
{
    float distance = Vector3.Distance(transform.position, pc.enemy.position);
    if (distance < changeStateTolerance)
    {
        if (aiState != AIState.resetAI)
            aiState = AIState.closeState;
        closeCombat = true;
    }
    else
    {
        if (aiState != AIState.resetAI)
```

```
            aiState = AIState.normalState;
        if (closeCombat)
        {
            if (!gotRandom)
            {
                storeRandom = ReturnRandom();
                gotRandom = true;
            }
            if (storeRandom < 60)
            {
                Movement();
            }
        }
        closeCombat = false;
    }
}
```

Функция `Blocking()` отвечает за блокирование поступающего удара. Блок сработает с вероятностью 50%.

```
void Blocking()
{
    if (pc.damage)
    {
        if (!gotRandom)
        {
            storeRandom = ReturnRandom();
            gotRandom = true;
        }
        if (storeRandom < 50)
        {
            blocking = true;
            pc.damage = false;
            pc.blocking = true;
        }
    }
    if (blocking)
    {
        blTimer += Time.deltaTime;
        if (blTimer > blockingRate)
        {
            pc.blocking = false;
            blTimer = 0;
        }
    }
}
```

В следующих функциях `CloseState()` и `NormalState()` засекается время жизни состояний малой и большой дистанции соответственно.

```
void NormalState()
{
    nrmTimer += Time.deltaTime;
    if (nrmTimer > normalRate)
    {
        initiateAI = true;
        nrmTimer = 0;
    }
}
void CloseState()    {
    clTimer += Time.deltaTime;
    if (clTimer > closeRate)
    {
        clTimer = 0;
        initiateAI = true;
    }
}
```

Функция `Jumping()` отвечает за прыжок. Персонаж будет прыгать либо при прыжке противника, либо с вероятностью 50%.

```
    void Jumping()
    {
        if (en.jumpKey || jump)
        {
            pc.vertical = 1;
            jump = false;
        }
        else
        {
            pc.vertical = 0;
        }
        jtimer += Time.deltaTime;
        if (jtimer > JumpRate * 10)
        {
            jRate = ReturnRandom();
            if (jRate < 50)
            {
                jump = true;
            }
            else
            {
                jump = false;
            }
            jtimer = 0;
        }
    }
```

В функции `ReturnRandom()` генерируется случайное число для методов, описанных выше.

```
    float ReturnRandom()
    {
        float retVal = Random.Range(0, 101);
        return retVal;
    }
}
```

3. Реализация программы

Для решения поставленной задачи было написано приложение в среде Visual Studio 2017 с использованием языка C#, а весь интерфейс реализован в среде Unity 3D. Исходный код приведен в приложении А. Скриншоты программы представлены в руководстве пользователя.

Таблица 3 демонстрирует реализованные модули и реализованные в них классы.

Таблица 3 – модули приложения

Модуль	**Реализуемый класс**
AiCharacter	AiCharacter
DamageScript	DamageScript

EventHolder	EventHolder
MainHero	MainHero
PausePlay	PausePlay
Player1SetUp	Player1SetUp
Player2SetUp	Player2SetUp
Redirect	Redirect
SettingsMenu	SettingsMenu
BackToMenu	BackToMenu

Ниже представлена диаграмма компонентов, которая показывает, каким образом собирается исполняемый файл (рисунок 11).

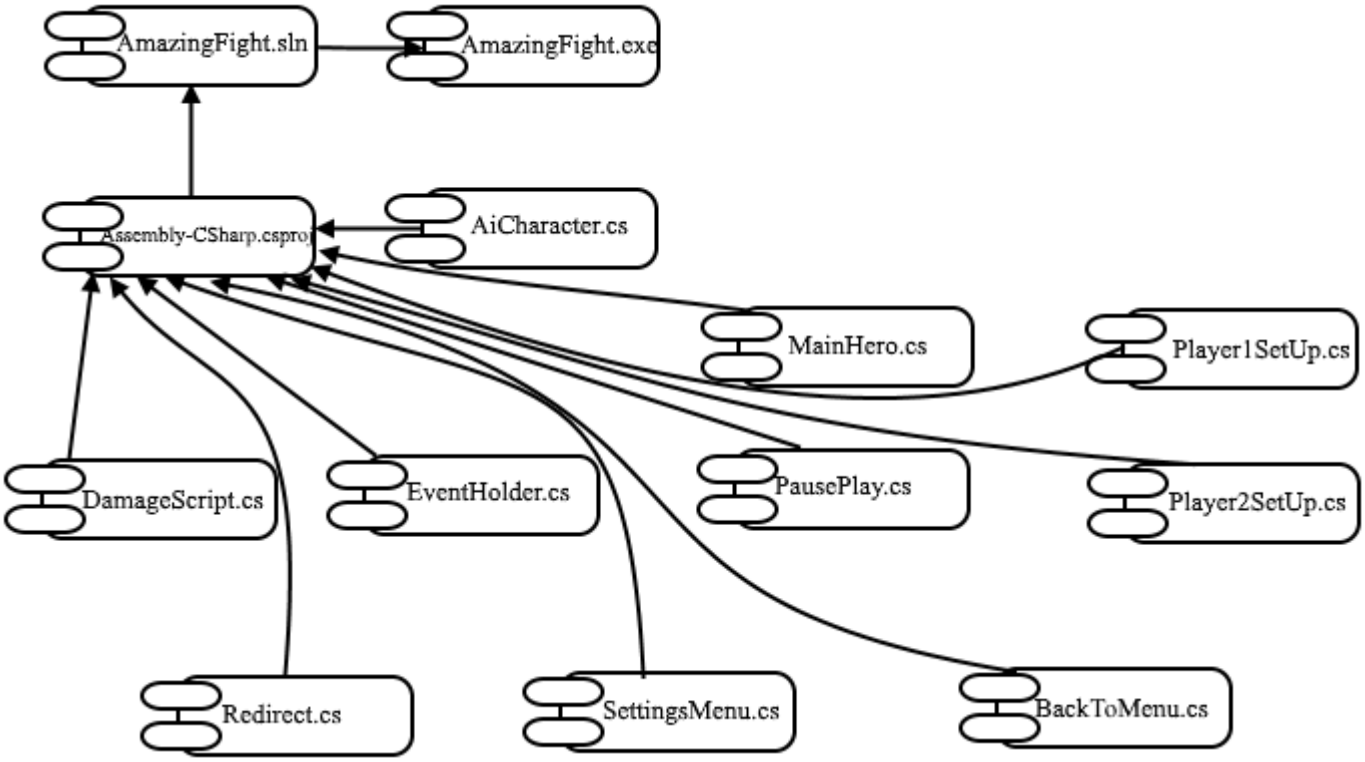

Рисунок 11 – диаграмма компонентов

Таблица 4 описывает назначение компонентов.

Таблица 4 - компоненты

Компонент	**Назначение**
AmazingFight.exe	Исполняемый файл приложения

AmazingFight.sln	Файл решения
Assembly-CSharp.csproj	Файл со свойствами решения
*.cs	Исходный код модуля с соответствующим названием

4. Тестирование программы

В таблице 5 представлены функциональные тесты

Таблица 5 – функциональные тесты

Вариант использования	**Параметры теста**	**Результат**
Изменить разрешение окна	600x800	Пройден (рис.12)
Изменить уровень громкости	>0.5	Пройден (рис.12)
Ознакомиться с управлением	---	Пройден (рис. 13)
Сыграть в игру	Первая карта	Пройден (рис.14, 16)
Выбрать режим игры	Игрок против компьютера	Пройден (рис.15)

Изменить разрешение окна и уровень громкости

Цель теста: определить, позволяет ли приложение изменять разрешение окна и уровень громкости.

Ход выполнения:

1. Перешли в меню настроек
2. Выбрали разрешение окна 600x800
3. Перетащили ползунок громкости звука ближе к правой границе

Результат: у приложения изменилось разрешение экрана и уровень громкости на заданные значения.

Вывод: тест пройден.

Ознакомиться с управлением

Цель теста: определить, позволяет ли приложение ознакомиться с управлением персонажей.

Ход выполнения:

1. Перешли в меню помощи

2. Увидели, какие клавиши отвечают за управление персонажей

Результат: мы ознакомились с управлением.

Вывод: тест пройден.

Сыграть в игру и выбрать режим

Цель теста: определить, позволяет ли приложение изменять режим игры и сыграть с выбранным режимом.

Ход выполнения:

1. Перешли в меню выбора карты и первую выбрали карту

2. Перешли в меню выбора режима игры и выбрали режим «Игрок против компьютера»

3. Перешли на игровое поле и сыграли против компьютера.

Результат: у приложения изменился режим игры на заданный, задний фон соответствовал выбранному и мы сыграли против компьютера.

Вывод: тест пройден.

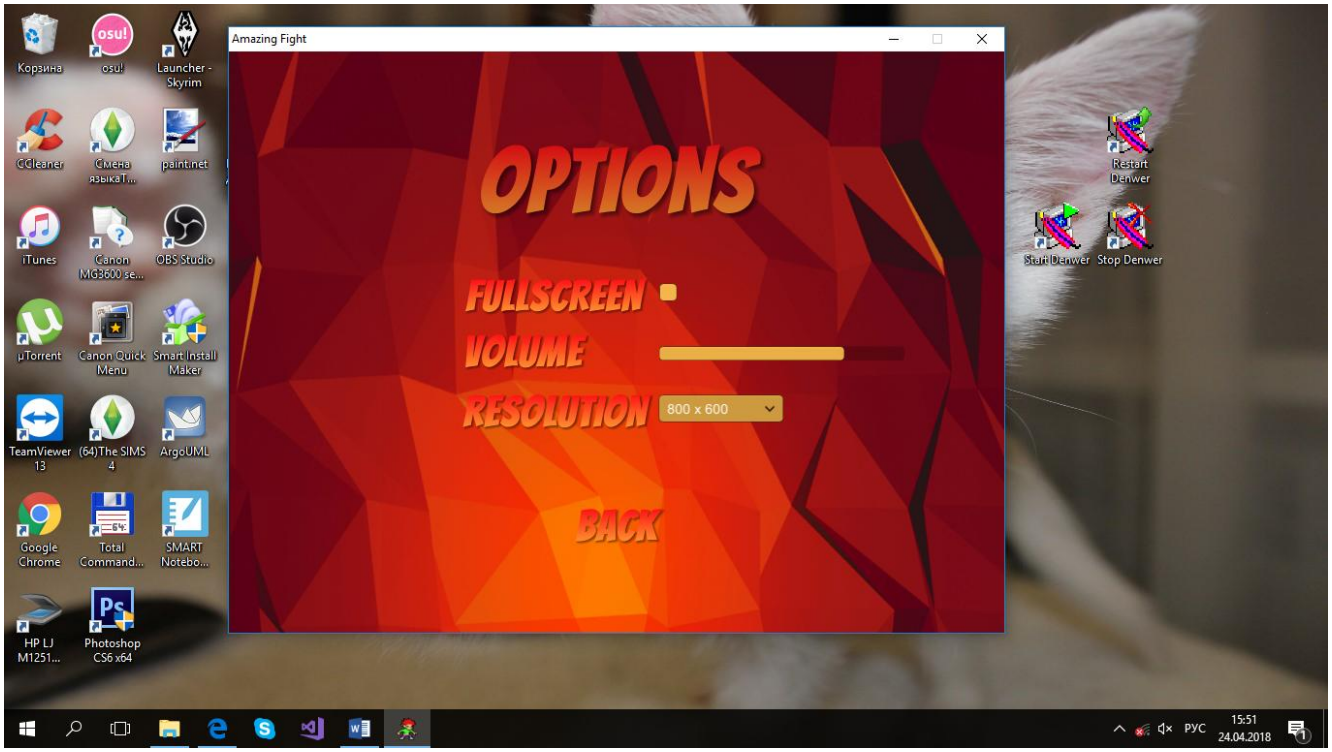

Рисунок 12 – Тест изменения уровня громкости и разрешения

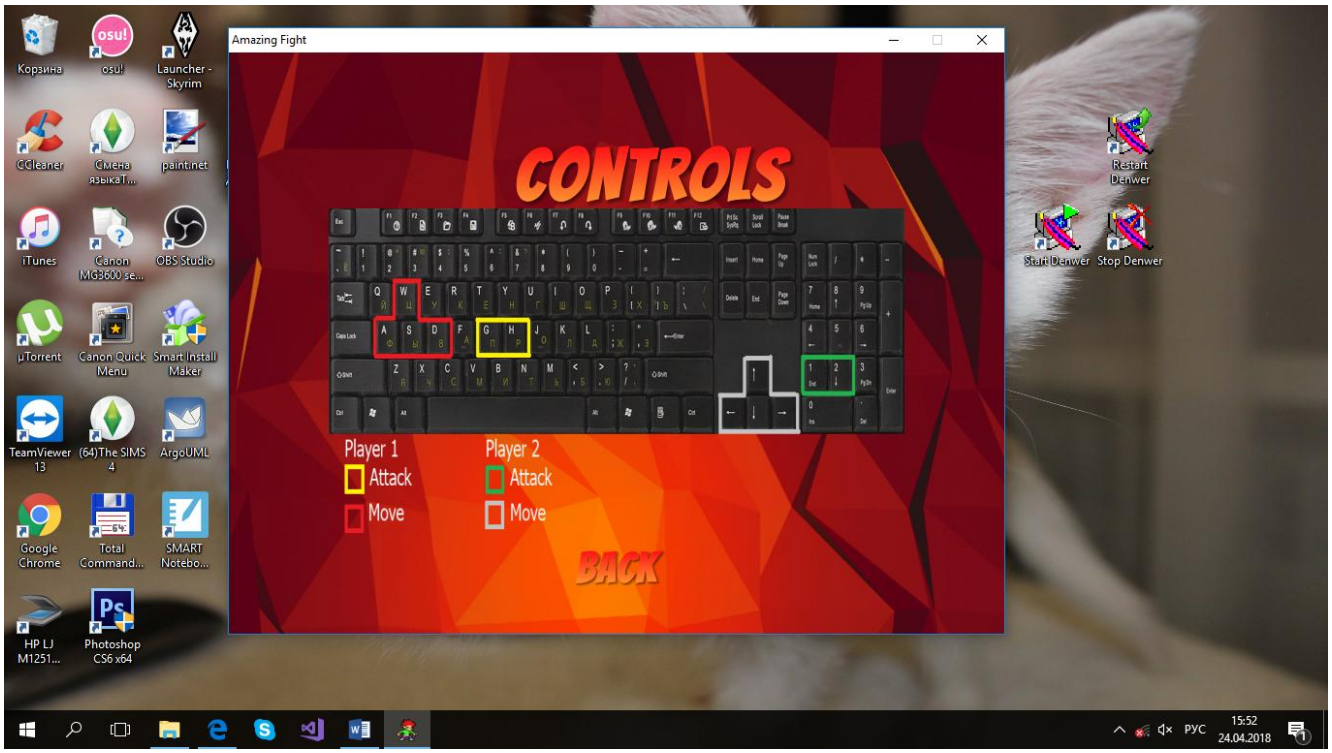

Рисунок 13 – Тест ознакомления с управлением

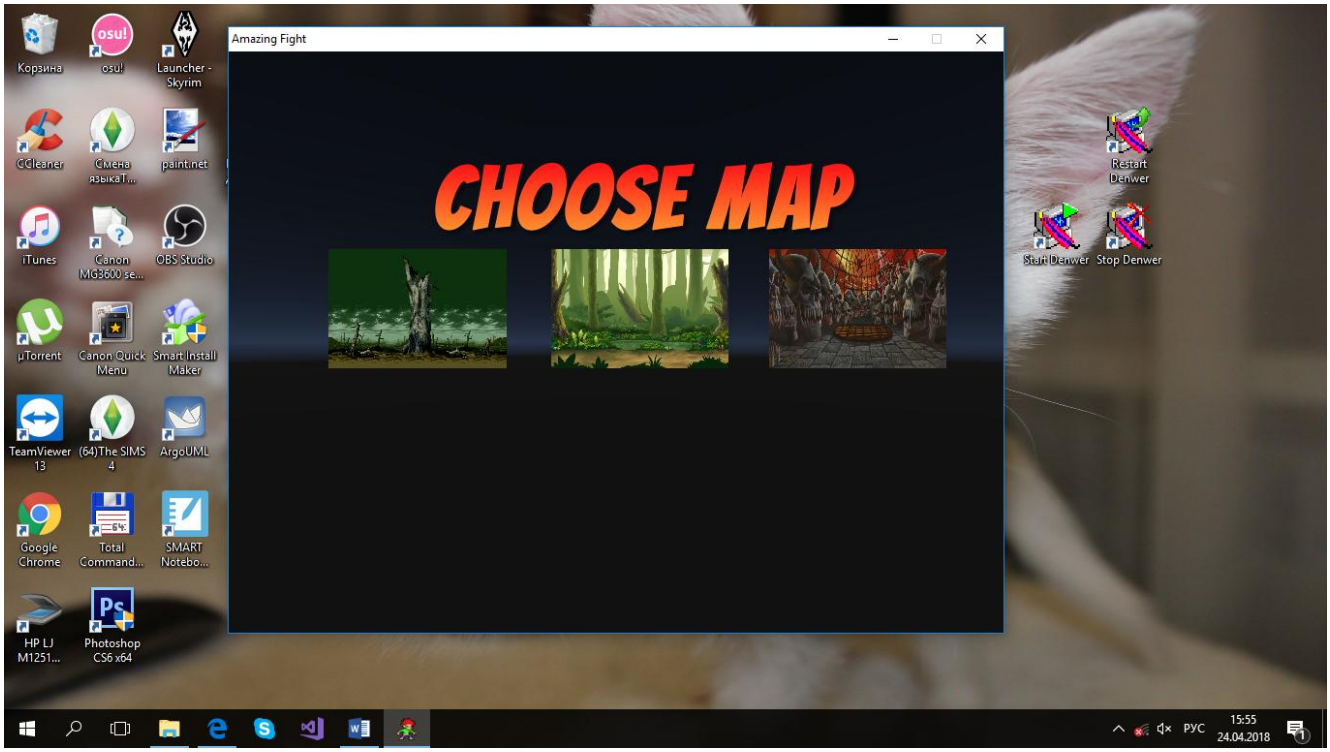

Рисунок 14 – Тест выбора карты

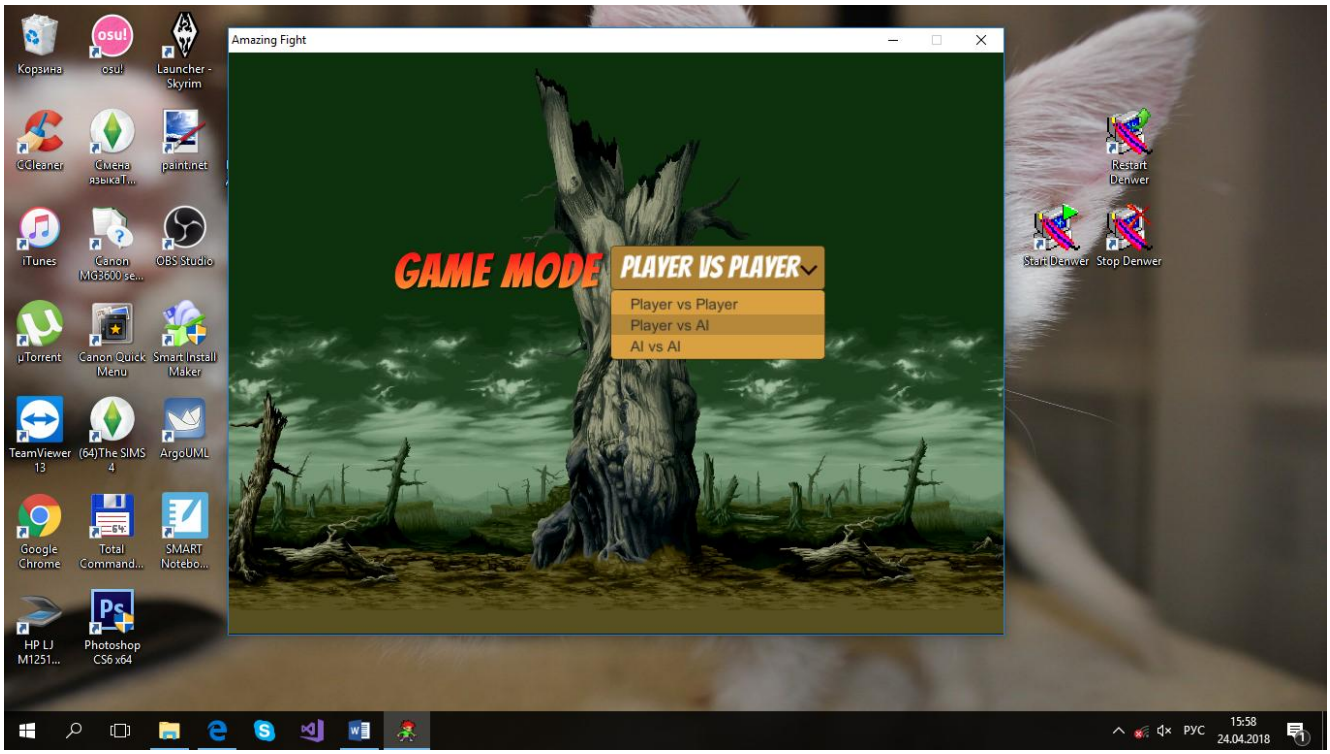

Рисунок 15 – Тест выбора режима игры

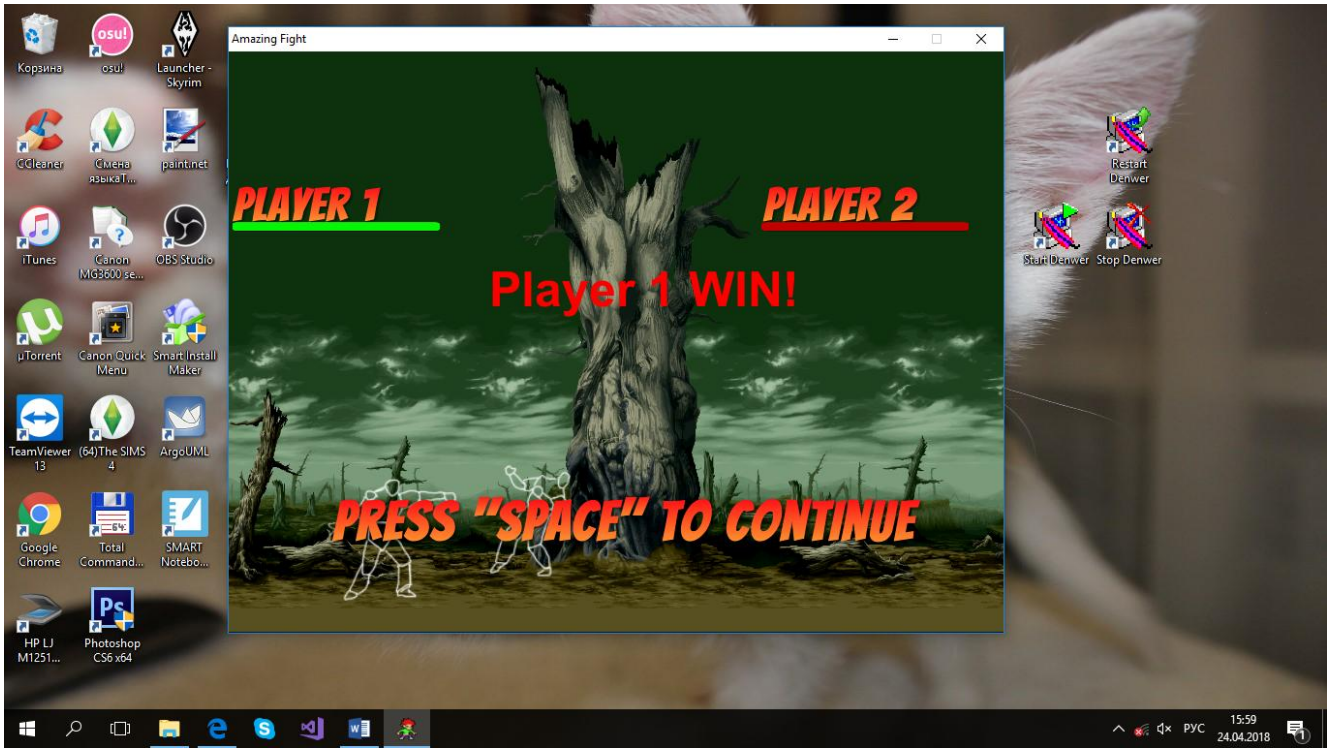

Рисунок 16 – Тест игры

5. Подготовка установочного файла

Для создания установочного файла будем использовать программу «Smart Instal Maker», так как она проста в использовании и предоставляет весь необходимый функционал.

На рисунке 17 изображено окно основной информации проекта по создаваемому установщику:

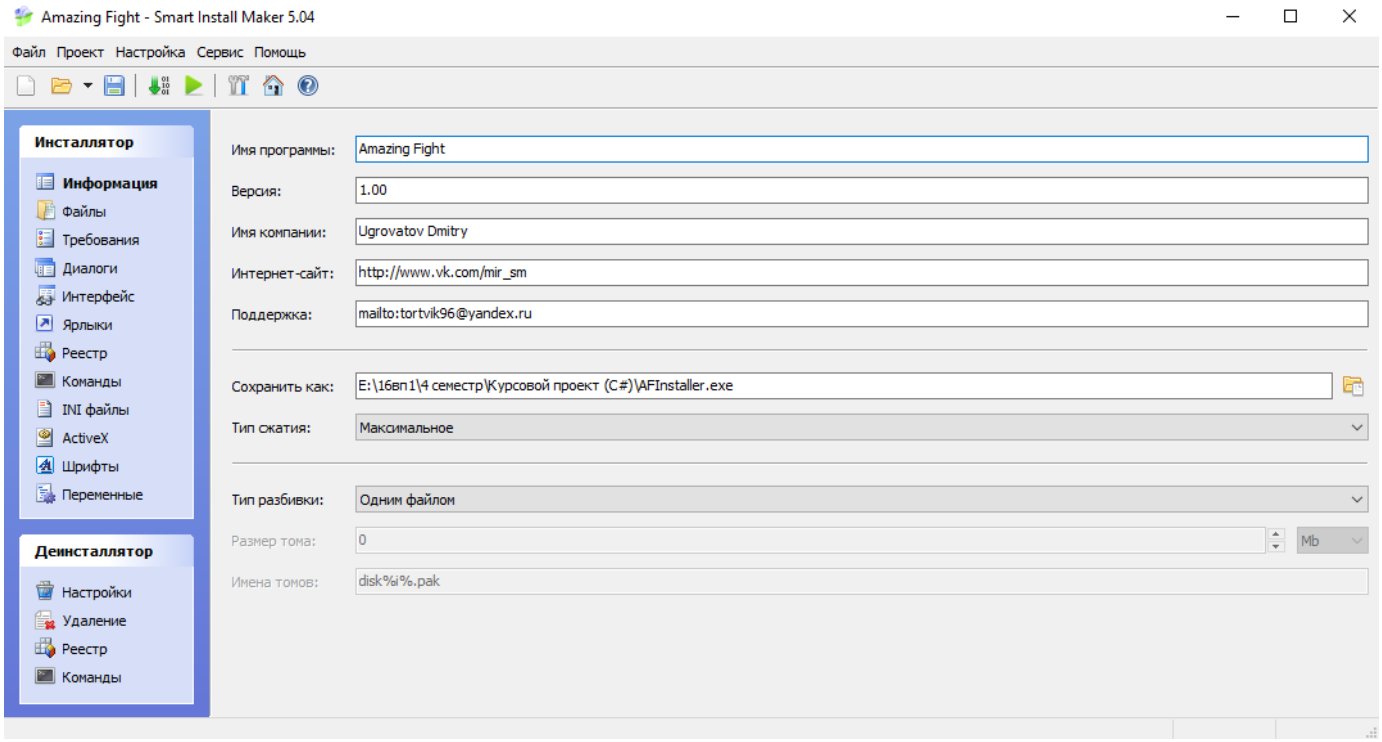

Рисунок 17 - Окно основной информации по создаваемому установщику

Далее выбираем, какие файлы должны устанавливаться (рисунок 18):

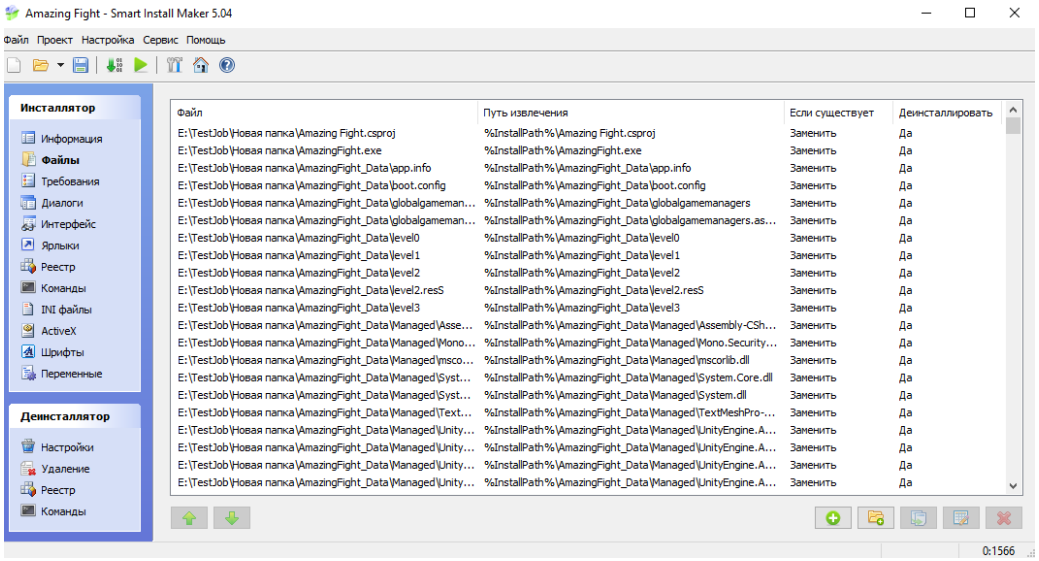

Рисунок 18 – Выбор устанавливаемых файлов

Устанавливаем требования для установки (рисунок 19):

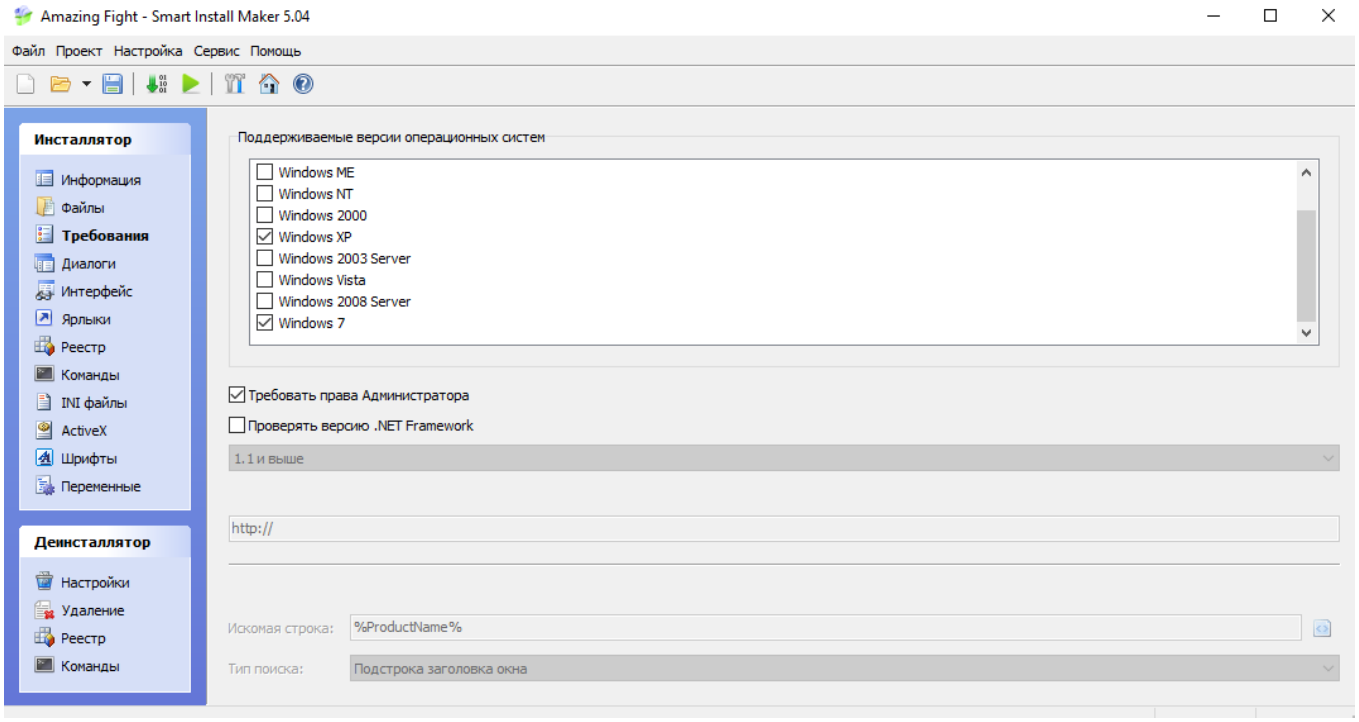

Рисунок 19 – Выбор требований для установки

Выбираем диалоги и устанавливаемый путь (рисунок 20):

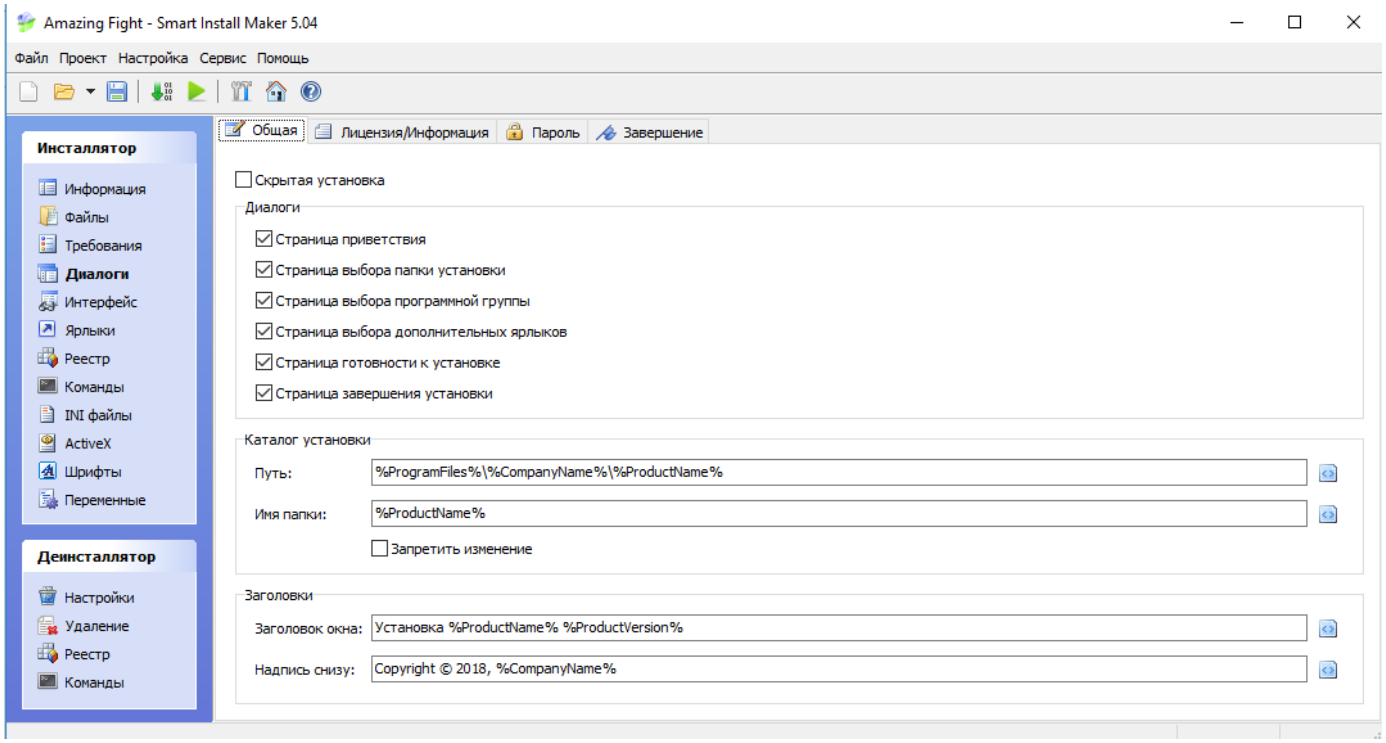

Рисунок 20 – Выбор диалогов и устанавливаемого пути

После чего компилируем проект и получаем установочный файл.

6. Руководство пользователя

6.1. Установка приложения

Для установки игры Amazing Fight (далее AF) откроем файл AFInstaller.exe (для открытия требуются права администратора) и выберем язык установки (рисунок 21).

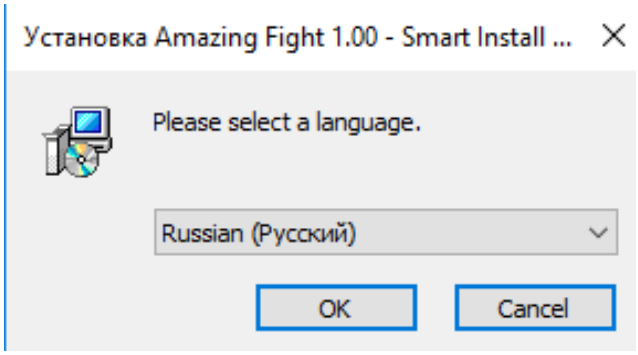

Рисунок 21 – Окно выбора языка установки

Далее нам откроется окно приветствия (рисунок 22).

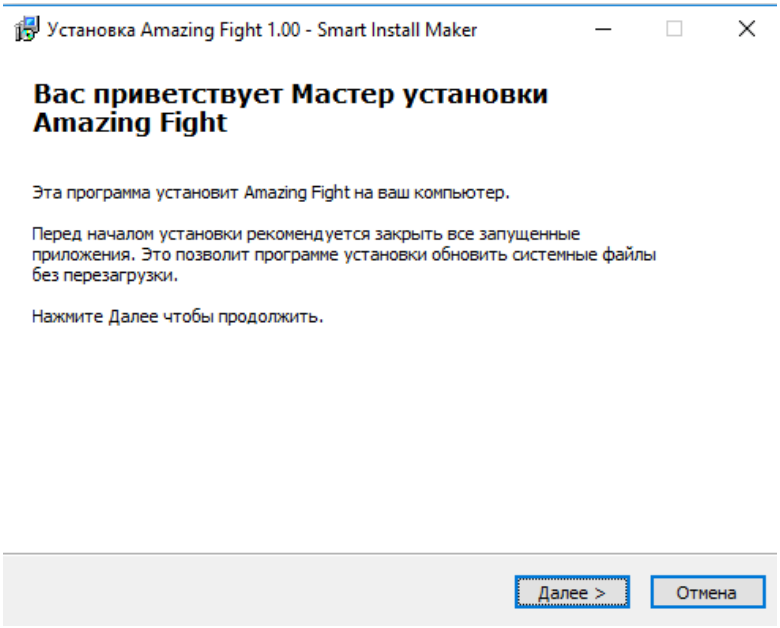

Рисунок 22 – Окно приветствия

Приложение предложит выбрать каталог установки AF (рисунок 23).

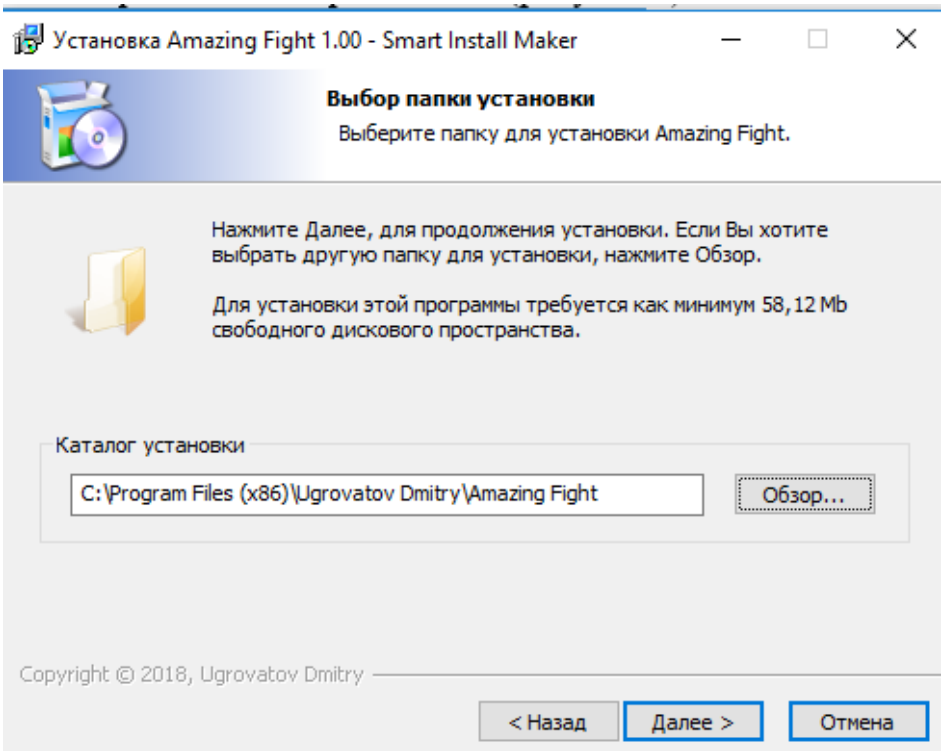

Рисунок 23 – Выбор каталога установки

Далее приложение оповестит Вас, что все готово для установки (рисунок 24).

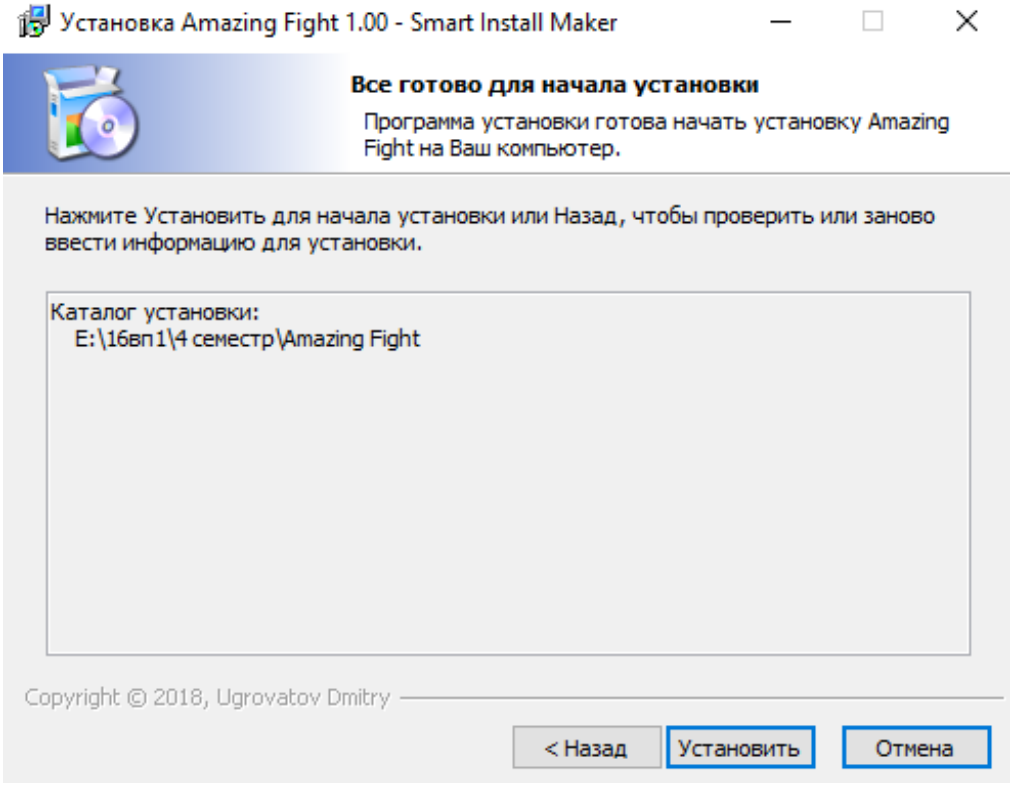

Рисунок 24 – Окно с информацией об установке

После нажатия кнопки «Установить» программа начнет копировать файлы игры в выбранную ранее директорию и после успешной установки откроется окно о завершении установки (рисунок 25).

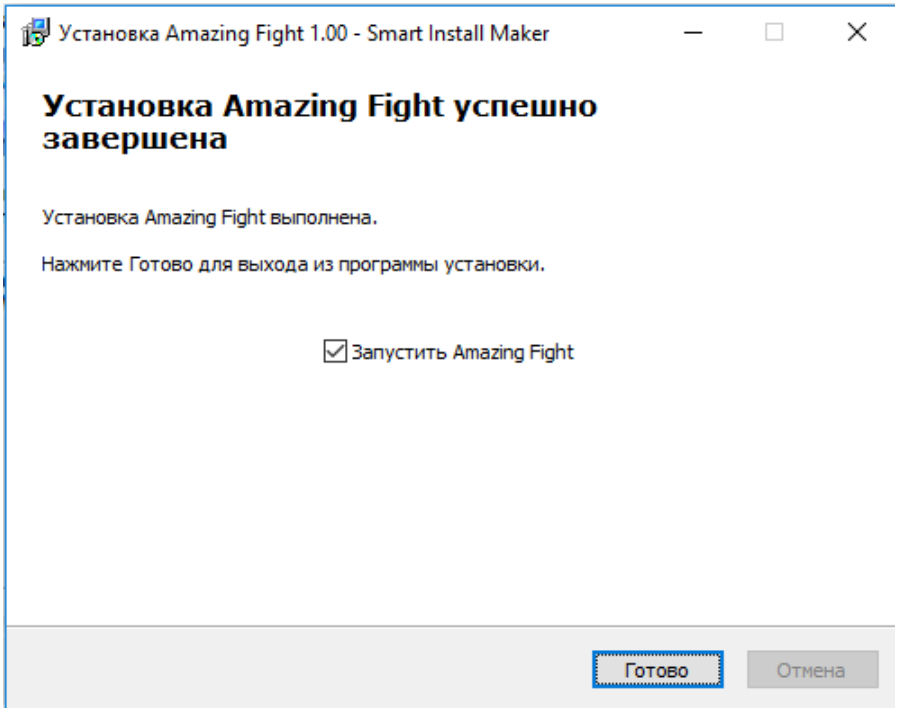

Рисунок 25 – Окно завершения установки

Приложение установлено!

6.2. Запуск приложения

После запуска приложения появится информация об авторе и потом откроется главное меню (рисунки 26-33):

Рисунок 26 – Скриншот приложения (информация об авторе)

Рисунок 27 – скриншот приложения (главное меню)

Рисунок 28 – скриншот приложения (меню настроек)

Рисунок 29 – скриншот приложения (меню управления)

Рисунок 30 – скриншот приложения (меню выбора карты)

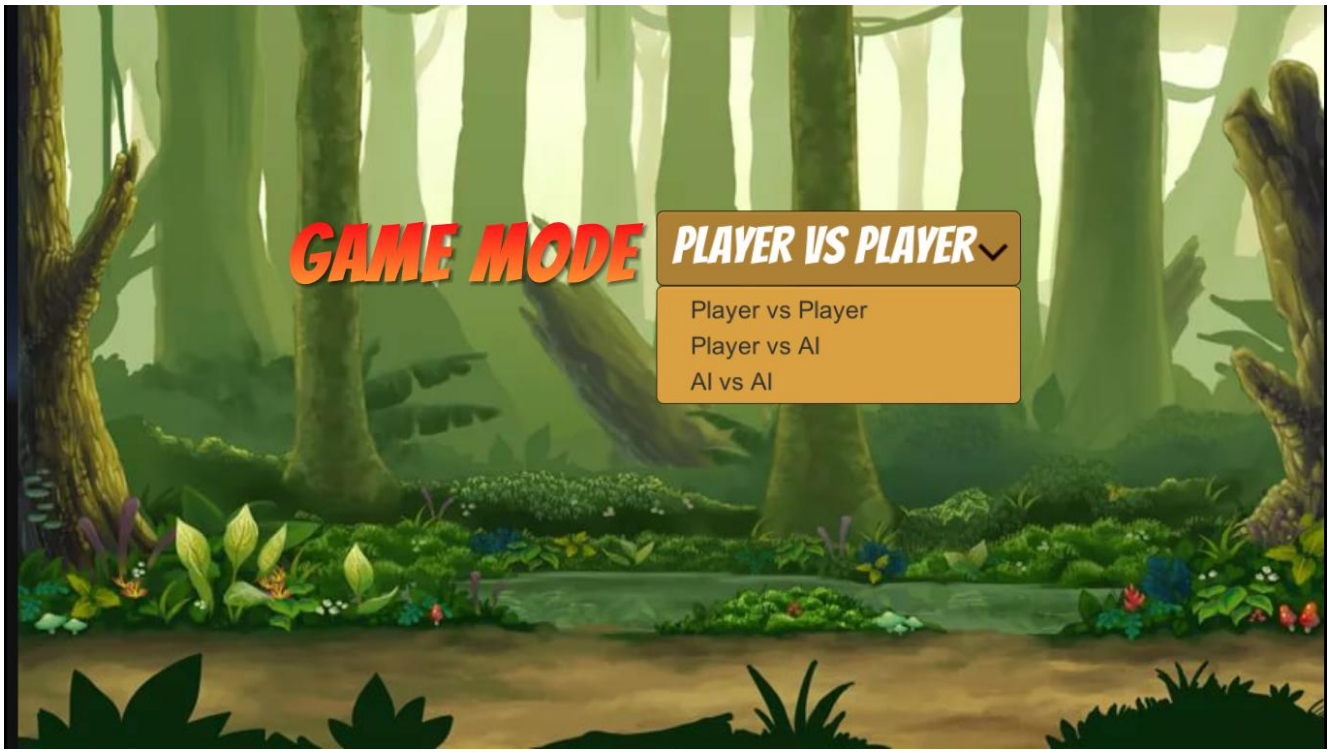

Рисунок 31 – скриншот приложения (меню выбора режима игры)

Рисунок 32 – скриншот приложения (игровое поле)

Рисунок 33 – скриншот приложения (меню паузы)

6.3. Управление персонажем

Теперь подробнее об управлении персонажем.

Объяснение будет идти для первого персонажа (левого), так как для правого все аналогично.

Начнем с кнопок передвижения. Для передвижения используются стандартные клавиши W, A, S, D:

- A, D – Перемещение влево/вправо;
- W – Прыжок;
- S – Приседание.

Второй персонаж (правый) аналогично управляется с помощью стрелок (Вверх, влево, вниз, вправо).

Теперь об атаках. Для осуществление атаки есть две кнопки:

- G – Атака 1;
- H – Атака 2.

Для второго персонажа соответственно это клавиши num1, num2 на дополнительной клавиатуре

Специальный прием. При нажатии комбинации клавиш G+H+S, будет выпущена энергетическая сфера, которая оттолкнет противника, если заденет его.

Для второго персонажа эта комбинация аналогична (num1+num2+↓).

ЗАКЛЮЧЕНИЕ

Результатом проекта является приложение «Amazing Fight», написанное на языке C# в среде Visual Studio 2017 и Unity3D. Данное приложение удовлетворяет всем изложенным требованиям.

В дальнейшем планируется поддерживать этот проект. В первую очередь добавить возможность выбора персонажей.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1) Разработка вашей первой игры с помощью Unity и C# [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/magazine/dn802605.aspx (Дата обращения: 10.03.2018)

2) Руководство Unity [Электронный ресурс]. – Режим доступа: www.docs.unity3d.com (Дата обращения: 10.03.2018)

3) Албахари Д., Албахари Б. / C# 7.0 Карманный справочник : Пер. с англ. – Ю. Артеменко, из-во «Вильямс», 2017. – 224с.

4) Вагнер Б. / Наиболее эффективное программирование на C#. 50 способов улучшения кода : Пер. с англ. – Ю. Артеменко, из-во «Вильямс», 2017. – 240с.

5) Скит Д. / C# для профессионалов. Тонкости программирования : Пер. с англ. – Ю. Артеменко, из-во «Вильямс», 2017. – 608с.

6) Торн. А. / Искусство создания сценариев в Unity : Пер. с англ. – Р. Рагимов, из-во «ДМК», 2016. – 360с.

ПРИЛОЖЕНИЕ А – ЛИСТИНГ ИСХОДНОГО КОДА

Листинг модуля «AiCharacter.cs»

```
using UnityEngine;
/// <summary>
/// Ai character
/// </summary>
public class AiCharacter : MonoBehaviour
{
    #region Variables

    /// <summary>
    /// Управляемый персонаж
    /// </summary>
    MainHero pc;

    /// <summary>
    /// Противник
    /// </summary>
    MainHero en;

    /// <summary>
    /// Дистанция изменения состояния
    /// </summary>
    public float changeStateTolerance = 3;

    /// <summary>
    /// Максимальное время нахождения на большой дистанции
    /// </summary>
    public float normalRate = 1;

    /// <summary>
    /// Время нахождения на большой дистанции
    /// </summary>
    float nrmTimer;

    /// <summary>
    /// Максимально время нахождения на маленькой дистанции
    /// </summary>
    public float closeRate = 0.5f;

    /// <summary>
    /// Время нахождения на маленькой дистанции
    /// </summary>
    float clTimer;

    /// <summary>
    /// Максимальная длительность блока
    /// </summary>
    public float blockingRate = 1.5f;

    /// <summary>
    /// Длительность блока
    /// </summary>
    float blTimer;

    /// <summary>
    /// Максимальная длительность состояния
    /// </summary>
    public float aiStateLife = 1;

    /// <summary>
```

```
/// Длительность состояния
/// </summary>
float aiTimer;

/// <summary>
/// Спровоцирован ли
/// </summary>
bool initiateAI;

/// <summary>
/// Близко ли к противнику
/// </summary>
bool closeCombat;

/// <summary>
/// Получено ли случайное число
/// </summary>
bool gotRandom;

/// <summary>
/// Случайное число
/// </summary>
float storeRandom;

/// <summary>
/// Сработал ли блок
/// </summary>
bool blocking;

/// <summary>
/// Сгенерирована ли атака
/// </summary>
bool randomizeAttacks;

/// <summary>
/// Количество сгенерированных атак
/// </summary>
int numberOfAttacks;

/// <summary>
/// Количество произведенных атак
/// </summary>
int curNumAttacks;

/// <summary>
/// Частота прыжка
/// </summary>
public float JumpRate = 1;

/// <summary>
/// Шанс прыжка
/// </summary>
float jRate;

/// <summary>
/// Прыгнул ли
/// </summary>
bool jump;

/// <summary>
/// Время без прыжка
/// </summary>
float jtimer;

/// <summary>
```

```
/// Структура состояний
/// </summary>
public enum AIState
{
    closeState,
    normalState,
    resetAI
}

/// <summary>
/// Состояния
/// </summary>
public AIState aiState;

#endregion

void Start()
{
    pc = GetComponent<MainHero>();
}

void Update()
{
    if (!en)
    {
        en = pc.enemy.GetComponent<MainHero>();
    }

    CheckDistance();
    States();
    AIAgent();
}

#region Functions

/// <summary>
/// Установка состояния
/// </summary>
void States()
{
    switch (aiState)
    {
        case AIState.closeState:
            CloseState();
            break;
        case AIState.normalState:
            NormalState();
            break;
        case AIState.resetAI:
            ResetAI();
            break;
    }
    Blocking();
    Jumping();
}

/// <summary>
/// Действия АИ
/// </summary>
void AIAgent()
{
    if (initiateAI)
    {
        aiState = AIState.resetAI;
        float multiplier = 0;
```

```
        if (!gotRandom)
        {
            storeRandom = ReturnRandom();
            gotRandom = true;
        }

        if (!closeCombat)
        {
            multiplier += 30;
        }
        else
        {
            multiplier -= 30;
        }

        if (storeRandom + multiplier < 50)
        {
            Attack();
        }
        else
        {
            Movement();
        }

    }
}

/// <summary>
/// Атака
/// </summary>
void Attack()
{
    if (!gotRandom)
    {
        storeRandom = ReturnRandom();
        gotRandom = true;
    }

    if (storeRandom < 75)
    {
        if (!randomizeAttacks)
        {
            numberOfAttacks = (int)Random.Range(1, 4);
            randomizeAttacks = true;
        }

        if (curNumAttacks < numberOfAttacks)
        {
            int attackNumber = Random.Range(0, pc.attack.Length);

            pc.attack[attackNumber] = true;

            curNumAttacks++;
        }
    }
    else
    {
        if (curNumAttacks < 1)
        {
            pc.specialAttack = true;
            curNumAttacks++;
        }
    }
}
```

```
/// <summary>
/// Передвижение
/// </summary>
void Movement()
{
    if (!gotRandom)
    {
        storeRandom = ReturnRandom();
        gotRandom = true;
    }

    if (storeRandom < 90)
    {
        if (pc.enemy.position.x < transform.position.x)
            pc.horizontal = -1;
        else
            pc.horizontal = 1;
    }
    else
    {
        if (pc.enemy.position.x < transform.position.x)
            pc.horizontal = 1;
        else
            pc.horizontal = -1;
    }
}

/// <summary>
/// Обновление действий
/// </summary>
void ResetAI()
{
    aiTimer += Time.deltaTime;

    if (aiTimer > aiStateLife)
    {
        initiateAI = false;
        pc.horizontal = 0;
        pc.vertical = 0;
        aiTimer = 0;

        gotRandom = false;

        storeRandom = ReturnRandom();

        if (storeRandom < 50)
            aiState = AIState.normalState;
        else
            aiState = AIState.closeState;

        curNumAttacks = 1;
        randomizeAttacks = false;
    }
}

/// <summary>
/// Проверка дистанции с противником
/// </summary>
void CheckDistance()
{
    float distance = Vector3.Distance(transform.position, pc.enemy.position);

    if (distance < changeStateTolerance)
    {
```

```
            if (aiState != AIState.resetAI)
                aiState = AIState.closeState;
            closeCombat = true;
        }
        else
        {
            if (aiState != AIState.resetAI)
                aiState = AIState.normalState;

            if (closeCombat)
            {
                if (!gotRandom)
                {
                    storeRandom = ReturnRandom();
                    gotRandom = true;
                }
                if (storeRandom < 60)
                {
                    Movement();
                }
            }
            closeCombat = false;
        }
    }

    /// <summary>
    /// Блокирование
    /// </summary>
    void Blocking()
    {
        if (pc.damage)
        {
            if (!gotRandom)
            {
                storeRandom = ReturnRandom();
                gotRandom = true;
            }

            if (storeRandom < 50)
            {
                blocking = true;
                pc.damage = false;
                pc.blocking = true;
            }
        }
        if (blocking)
        {
            blTimer += Time.deltaTime;

            if (blTimer > blockingRate)
            {
                pc.blocking = false;
                blTimer = 0;
            }
        }
    }

    /// <summary>
    /// Большая дистанция
    /// </summary>
    void NormalState()
    {
        nrmTimer += Time.deltaTime;

        if (nrmTimer > normalRate)
```

```
    {
        initiateAI = true;
        nrmTimer = 0;
    }
}

/// <summary>
/// Маленькая дистанция
/// </summary>
void CloseState()
{
    clTimer += Time.deltaTime;

    if (clTimer > closeRate)
    {
        clTimer = 0;
        initiateAI = true;
    }
}

/// <summary>
/// Прыжок
/// </summary>
void Jumping()
{
    if (en.jumpKey || jump)
    {
        pc.vertical = 1;
        jump = false;
    }
    else
    {
        pc.vertical = 0;
    }

    jtimer += Time.deltaTime;

    if (jtimer > JumpRate * 10)
    {
        jRate = ReturnRandom();

        if (jRate < 50)
        {
            jump = true;
        }
        else
        {
            jump = false;
        }

        jtimer = 0;
    }
}

/// <summary>
/// Возвращает случайное число
/// </summary>
/// <returns>Случайное число</returns>
float ReturnRandom()
{
    float retVal = Random.Range(0, 101);
    return retVal;
}

#endregion
```

```
}
```

Листинг модуля «DamageScript.cs»

```
using UnityEngine;

/// <summary>
/// Damage script
/// </summary>
public class DamageScript : MonoBehaviour
{
    /// <summary>
    /// Наносимый урон
    /// </summary>
    public float damage = 10f;

    /// <summary>
    /// Соприкосновение коллайдера кулака с объектами
    /// </summary>
    /// <param name="other">Коллайдер, с которым соприкасается кулак</param>
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.transform.root != transform.root && other.tag != "Ground" &&
!other.isTrigger && other.tag != "Wall")
        {
            if(other.tag!="Projectile")
            if (!other.transform.GetComponent<MainHero>().blocking &&
!other.transform.GetComponent<MainHero>().damage)
            {

                other.transform.GetComponent<MainHero>().damage = true;

                //Триггер анимации
                other.transform.root.GetComponentInChildren<Animator>().SetTrigger("Damage");

                //Звук удара
                other.transform.GetComponent<MainHero>().Au.Play();

                //Отнимание здоровья у противника
                other.transform.GetComponent<MainHero>().CurrentHealth -= damage;
                other.transform.GetComponent<MainHero>().healthBar.value =
other.transform.GetComponent<MainHero>().CurrentHealth /
other.transform.GetComponent<MainHero>().MaxHealth;

                //Вывод надписи Победы
                if (other.transform.GetComponent<MainHero>().CurrentHealth <= 0f)
                {

other.transform.GetComponent<MainHero>().GetComponents<CircleCollider2D>()[1].enabled=false;

                    other.transform.GetComponent<MainHero>().WinText.text =
other.transform.GetComponent<MainHero>().enemy.name + " WIN!" ;
                    other.transform.GetComponent<MainHero>().WinText.enabled = true;
                    other.transform.GetComponent<MainHero>().ContinueText.SetActive(true);
                }
            }
        }
    }
}
```

Лисинг модуля «EventHolder.cs»

```
using UnityEngine;

/// <summary>
/// Event holder
/// </summary>
public class EventHolder : MonoBehaviour {

    /// <summary>
    /// Персонаж
    /// </summary>
    MainHero pl;

       void Start ()
    {
        pl = transform.root.GetComponent<MainHero>();
       }

    /// <summary>
    /// Специальная атака
    /// </summary>
    public void ThrowProjectile()
    {
        pl.specialAttack = true;
    }
}
```

Листинг модуля «MainHero.cs»

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

/// <summary>
/// Main hero
/// </summary>
public class MainHero : MonoBehaviour
{
    #region Variables

    /// <summary>
    /// Индекс персонажа
    /// </summary>
    public int PlayerNumber = 1;

    /// <summary>
    /// Управляется ли персонаж с помощью АИ
    /// </summary>
    public bool AiCharacter;

    /// <summary>
    /// Противник
    /// </summary>
    public Transform enemy;

    /// <summary>
    /// Ригидбоди объекта
    /// </summary>
    private Rigidbody2D rb;

    /// <summary>
    /// Аниматор объекта
    /// </summary>
    Animator anim;

    /// <summary>
    /// Перемещение по оси X
    /// </summary>
    public float horizontal;

    /// <summary>
    /// Перемещение по оси Y
    /// </summary>
    public float vertical;

    /// <summary>
    /// Максимальная скорость передвежения персонажа
    /// </summary>
    public float maxSpeed = 25f;

    /// <summary>
    /// Вектор передвежения
    /// </summary>
    Vector3 movement;

    /// <summary>
    /// Сидит ли персонаж
    /// </summary>
    bool crouch;

    /// <summary>
```

```
/// Сила прыжка (устанавливается из дебаг окна)
/// </summary>
public float JumpForce = 20;

/// <summary>
/// Сила прыжка
/// </summary>
float jmpForce;

/// <summary>
/// Максимальная продолжительность прыжка
/// </summary>
public float JumpDuration = 0.1f;

/// <summary>
/// Продолжительность прыжка
/// </summary>
float jmpDuration;

/// <summary>
/// Прыгает ли
/// </summary>
public bool jumpKey;

/// <summary>
/// Падает ли
/// </summary>
bool falling;

/// <summary>
/// Стоит ли на земле
/// </summary>
bool onGround;

/// <summary>
/// Скорость атаки
/// </summary>
public float attackRate = 0.3f;

/// <summary>
/// Массив с вариантами атаки
/// </summary>
public bool[] attack = new bool[2];

/// <summary>
/// Продолжительность нажатий клавиш атаки
/// </summary>
float[] attacktimer = new float[2];

/// <summary>
/// Количество нажатий клавиш атаки
/// </summary>
int[] timesPressed = new int[2];

/// <summary>
/// Нанесен ли урон
/// </summary>
public bool damage = false;

/// <summary>
/// Максимальная продолжительность неуязвимости после получения урона
/// </summary>
public float noDamage = 1;

/// <summary>
```

```
/// Продолжительность неуязвимости после получения урона
/// </summary>
float noDamageTimer;

/// <summary>
/// Сработал ли блок
/// </summary>
public bool blocking = false;

/// <summary>
/// Максимальная продолжительность блока
/// </summary>
public float blockingRate = 1.5f;

/// <summary>
/// Продолжительность блока
/// </summary>
float blTimer;

/// <summary>
/// Нажата ли клавиша специальной атаки
/// </summary>
public bool specialAttack;

/// <summary>
/// Время жизни прожектайла специальной атаки
/// </summary>
float prLifeTime = 0.3f;

/// <summary>
/// Получено ли случайное число
/// </summary>
bool gotRandom;

/// <summary>
/// Случайное число
/// </summary>
float storeRandom;

/// <summary>
/// Прожектайл
/// </summary>
public GameObject projectile;

/// <summary>
/// Аудиомикшер
/// </summary>
public AudioMixer am;

/// <summary>
/// Источник звука удара
/// </summary>
public AudioSource Au;

/// <summary>
/// Упало ли здоровье ниже 0
/// </summary>
public bool die = false;

/// <summary>
/// Максимальное время работы игры после смерти персонажа
/// </summary>
float deathAnimLife = 0.2f;

/// <summary>
```

```
/// Время работы игры после смерти персонажа
/// </summary>
float deathAnimTimer;

/// <summary>
/// Слайдер со здоровьем персонажа
/// </summary>
public Slider healthBar;

/// <summary>
/// Текущее здоровье
/// </summary>
public float CurrentHealth { get; set; }

/// <summary>
/// Максимальное здоровье
/// </summary>
public float MaxHealth { get; set; }

/// <summary>
/// Текст "Continue"
/// </summary>
public GameObject ContinueText;

/// <summary>
/// Текст "...WIN"
/// </summary>
public Text WinText;

/// <summary>
/// Объект со слайдером здоровья (для паузы)
/// </summary>
public GameObject plH;

/// <summary>
/// Нажата ли пауза
/// </summary>
public bool paused = false;

/// <summary>
/// Уровень громкости
/// </summary>
public float volume;

/// <summary>
/// Объект меню паузы
/// </summary>
public GameObject pauseMenu;

#endregion

void Start()
{
    Cursor.visible = false;

    am.GetFloat("volume", out volume);

    Time.timeScale = 1;

    GetComponents<CircleCollider2D>()[1].enabled = false;

    rb = GetComponent<Rigidbody2D>();
    anim = GetComponentInChildren<Animator>();

    am.GetFloat("volume", out volume);
```

```
    WinText.gameObject.SetActive(true);
    WinText.enabled = false;

    ContinueText.SetActive(false);
    MaxHealth = 100f;
    CurrentHealth = MaxHealth;
    healthBar.value = CalculateHealth();

    healthBar.enabled = false;
    jmpForce = JumpForce;

    GameObject[] players = GameObject.FindGameObjectsWithTag("Player");

    foreach (GameObject pl in players)
    {
        if (pl.transform != this.transform)
        {
            enemy = pl.transform;
        }
    }


    Au = GetComponent<AudioSource>();
}

void Update()
{
    PauseApp();
    AttackInput();
    ScaleCheck();
    OnGroundCheck();
    Blocking();
    Damage();
    SpecialAttack();
    UpdateAnimator();
    GameOver();
}

void FixedUpdate()
{
    if (AiCharacter)
    {

        if (!GetComponent<AiCharacter>())
        {
            gameObject.AddComponent<AiCharacter>();
        }
        GetComponent<AiCharacter>().enabled = true;
    }
    else
    {
        GetComponent<AiCharacter>().enabled = false;
    }

    if (!AiCharacter)
    {
        horizontal = Input.GetAxis("Horizontal" + PlayerNumber.ToString());
        vertical = Input.GetAxis("Vertical" + PlayerNumber.ToString());
    }
    Vector3 movement = new Vector3(horizontal, 0, 0);

    crouch = (vertical < -0.1f);

    CircleCollider2D[] cr;
    cr = GetComponents<CircleCollider2D>();
```

```
    if (vertical > 0.1f)
    {
        if (!jumpKey)
        {
            jmpDuration += Time.deltaTime;
            jmpForce += Time.deltaTime;

            if (jmpDuration < JumpDuration)
            {
                rb.velocity = new Vector2(rb.velocity.x, jmpForce);
                jumpKey = false;

            }
            else
            {
                jumpKey = true;
                jmpDuration = 0;
            }
        }
        else
            if (onGround) jumpKey = false;
    }

    if (!onGround && vertical < 0.1f)
    {
        falling = true;
    }

    if (attack[0] && !jumpKey || attack[1] && !jumpKey)
    {
        movement = Vector3.zero;
    }

    if (!crouch && !damage && !blocking)
    {
        cr[1].enabled = true;
        rb.AddForce(movement * maxSpeed);
    }
    else
    {
        if (!jumpKey)
        {
            rb.velocity = Vector3.zero;
            cr[1].enabled = false;
        }
    }

}

#region Functions

/// <summary>
/// Изменение гравитации в зависимости от положения
/// </summary>
void OnGroundCheck()
{
    if (!onGround)
    {
        rb.gravityScale = 5;
    }
    else
    {
        rb.gravityScale = 1;
    }
```

```
}

/// <summary>
/// Нажатие клавиш атаки
/// </summary>
void AttackInput()
{
    if (Input.GetButtonDown("Attack1" + PlayerNumber.ToString()))
    {
        attack[0] = true;
        attacktimer[0] = 0;
        timesPressed[0]++;

        CircleCollider2D[] enemyCC = enemy.GetComponents<CircleCollider2D>();
        enemyCC[1].enabled = true;
    }

    if (attack[0])
    {
        attacktimer[0] += Time.deltaTime;
        if (attacktimer[0] > attackRate || timesPressed[0] >= 4)
        {
            attacktimer[0] = 0;
            attack[0] = false;
            timesPressed[0] = 0;

        }
    }

    if (Input.GetButtonDown("Attack2" + PlayerNumber.ToString()))
    {
        attack[1] = true;
        attacktimer[1] = 0;
        timesPressed[1]++;

        CircleCollider2D[] enemyCC = enemy.GetComponents<CircleCollider2D>();
        enemyCC[1].enabled = true;
    }

    if (attack[1])
    {
        attacktimer[1] += Time.deltaTime;
        if (attacktimer[1] > attackRate || timesPressed[0] >= 4)
        {
            attacktimer[1] = 0;
            attack[1] = false;
            timesPressed[1] = 0;

        }
    }
}

/// <summary>
/// Получение урона
/// </summary>
void Damage()
{
    if (damage)
    {
        noDamageTimer += Time.deltaTime;

        if (noDamageTimer > noDamage)
        {
            damage = false;
            noDamageTimer = 0;
```

```
            }
            if (CurrentHealth <= 0f)
            {
                die = true;
                damage = false;
            }
        }
    }

    /// <summary>
    /// Специальная атака
    /// </summary>
    void SpecialAttack()
    {
        if (specialAttack)
        {

            GameObject pr = Instantiate(projectile, transform.position, Quaternion.identity)
as GameObject;
            DestroyObject(pr, prLifeTime);
            Vector3 nrDir = new Vector3(enemy.position.x, transform.position.y, 0);
            Vector3 dir = nrDir - transform.position;
            pr.GetComponent<Rigidbody2D>().AddForce(dir * 10, ForceMode2D.Impulse);

            specialAttack = false;

        }
    }

    /// <summary>
    /// Обновление анимации
    /// </summary>
    void UpdateAnimator()
    {

        anim.SetBool("Die", die);
        anim.SetBool("Attack1", attack[0]);
        anim.SetBool("Attack2", attack[1]);
        anim.SetBool("Crouch", crouch);
        anim.SetBool("OnGround", this.onGround);
        anim.SetBool("Falling", this.falling);
        anim.SetFloat("Movement", Mathf.Abs(horizontal));
        anim.SetBool("Blocking", blocking);
    }

    /// <summary>
    /// Проверка на соприкосновение с полом
    /// </summary>
    /// <param name="col">Коллайдер, с которым соприкасается персонаж</param>
    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.collider.tag == "Ground")
        {
            onGround = true;

            jumpKey = false;
            jmpDuration = 0;
            jmpForce = JumpForce;
            falling = false;
        }

    }

    /// <summary>
    /// Проверка на отталкивание от пола
```

```
/// </summary>
/// <param name="col">Коллайдер, от которого отталкивается персонаж</param>
void OnCollisionExit2D(Collision2D col)
{
    if (col.collider.tag == "Ground")
    {
        onGround = false;
    }

}

/// <summary>
/// Разворот в сторону противника
/// </summary>
void ScaleCheck()
{
    if (transform.position.x < enemy.position.x)
        transform.localScale = new Vector3(-1, 1, 1);
    else
        transform.localScale = Vector3.one;
}

/// <summary>
/// Окончание игры
/// </summary>
void GameOver()
{
    if (CurrentHealth <= 0f)
    {
        deathAnimTimer += Time.deltaTime;
        if (deathAnimTimer > deathAnimLife)
        {
            Time.timeScale = 0;
            if (Input.GetKeyDown(KeyCode.Space))
            {
                Cursor.visible = true;
                SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
            }
        }
    }
}

/// <summary>
/// Возвращает текущее здоровье
/// </summary>
/// <returns>Текущее здоровье</returns>
float CalculateHealth()
{
    return CurrentHealth / MaxHealth;
}

/// <summary>
/// Блокирование
/// </summary>
void Blocking()
{
    if (damage)
    {
        if (!gotRandom)
        {
            storeRandom = ReturnRandom();
            gotRandom = true;
        }

        if (storeRandom < 50)
```

```
        {
            blocking = true;
            damage = false;
            blocking = true;
        }
    }

    if (blocking)
    {
        blTimer += Time.deltaTime;

        if (blTimer > blockingRate)
        {
            blocking = false;
            blTimer = 0;
        }
    }
}

/// <summary>
/// Возвращает случайное число
/// </summary>
/// <returns>Случайное число</returns>
float ReturnRandom()
{
    float retVal = Random.Range(0, 101);
    return retVal;
}

/// <summary>
/// Ставит игру на паузу
/// </summary>
void PauseApp()
{

    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (!paused)
        {
            Cursor.visible = true;
            Time.timeScale = 0;
            paused = true;
            //показать меню паузы
            pauseMenu.SetActive(true);
            plH.SetActive(false);
        }

    }
}

/// <summary>
/// Продолжает игру (Кнопка "Continue")
/// </summary>
public void ContinueBut()
{
    plH.SetActive(true);
    //healthBar.enabled = true;
    Time.timeScale = 1;
    paused = false;
    pauseMenu.SetActive(false);
    Cursor.visible = false;
}

/// <summary>
/// Переход на сцену с главным меню (Кнопка "Main menu")
```

```
    /// </summary>
    public void MenuBut()
    {
        Time.timeScale = 1;
        paused = false;
        pauseMenu.SetActive(false);
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
    }

    /// <summary>
    /// Выход из игры (Кнопка "Quit")
    /// </summary>
    public void QuitBut()
    {
        Application.Quit();
    }
    #endregion
}
```

Листинг модуля «PausePlay.cs»

```
using UnityEngine;
/// <summary>
/// Pause play
/// </summary>
public class PausePlay : MonoBehaviour {

	/// <summary>
	/// Объект меню паузы
	/// </summary>
	public GameObject pauseMenu;
	//public AudioSource musik;
	// Use this for initialization


	// Update is called once per frame
	void Update () {
		if (pauseMenu.activeInHierarchy)
			GetComponent<AudioSource> ().Pause ();
	}


}
```

Листинг модуля «Player1SetUp.cs»

```
using UnityEngine;
/// <summary>
/// Player1 set up
/// </summary>
public class Player1SetUp : MonoBehaviour {

    /// <summary>
    /// Устанавливает тип управления персонажем 1
    /// </summary>
    /// <param name="dropDownIndex">Индекс из выпадающего списка</param>
    public void PlayVs(int dropDownIndex)
    {
        switch (dropDownIndex)
        {
            case 2:
                GetComponent<MainHero>().AiCharacter = true;
                break;
            default:
                GetComponent<MainHero>().AiCharacter = false;
                break;
        }

    }
}
```

Листинг модуля «Player2.SetUp.cs»

```
using UnityEngine;
/// <summary>
/// Player2 set up
/// </summary>
public class Player2SetUp : MonoBehaviour {

    /// <summary>
    /// Устанавливает тип управления персонажем 2
    /// </summary>
    /// <param name="dropDownIndex">>Индекс из выпадающего списка</param>
    public void PlayVs(int dropDownIndex)
    {
        switch (dropDownIndex)
        {
            case 0:
                GetComponent<MainHero>().AiCharacter = false;
                break;
            case 1:
                GetComponent<MainHero>().AiCharacter = true;
                break;
            case 2:
                GetComponent<MainHero>().AiCharacter = true;
                break;
        }

    }
}
```

Листинг модуля «Redirect.cs»

```
using System.IO;
using UnityEngine;
using UnityEngine.SceneManagement;
/// <summary>
/// Redirect.
/// </summary>
public class Redirect : MonoBehaviour {

       /// <summary>
       /// Продолжительность показа лого
       /// </summary>
    public float logoDuration = 0;

       /// <summary>
       /// Максимальная продолжительность показа лого
       /// </summary>
       public float maxLogoDuration = 1f;

    // Use this for initialization
    void Start () {
        using (BinaryReader br = new BinaryReader(new FileStream("Resoludion.dat",
FileMode.Open)))
        {
            Screen.fullScreen= br.ReadBoolean();

            int currentResolutionIndex = br.ReadInt32();
            br.Close();
            //resolutionDropDown.value = currentResolutionIndex;
            //resolutionDropDown.RefreshShownValue();
        }
    }
    private void Update()
    {
        logoDuration += Time.deltaTime;
        if (logoDuration > maxLogoDuration)
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        }

    }

}
```

Листинг модуля «SettingMenu.cs»

```
using System.IO;
using UnityEngine;
using UnityEngine.Audio;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
/// <summary>
/// Settings menu
/// </summary>
public class SettingsMenu : MonoBehaviour
{

    #region Variables

    /// <summary>
    /// Аудиомикшер
    /// </summary>
    public AudioMixer audioMixer;

    /// <summary>
    /// Слайдер громкости
    /// </summary>
    public Slider volumeSlider;

    /// <summary>
    /// Уровень громкости
    /// </summary>
    public float volume1;
    /// <summary>
    /// Тагл "Fullscreen"
    /// </summary>
    public Toggle fullScr;

    /// <summary>
    /// Выпадающий список с разрешениями экрана
    /// </summary>
    public Dropdown resolutionDropDown;

    /// <summary>
    /// Массив разрешений экрана
    /// </summary>
    Resolution[] resolutions;

    /// <summary>
    /// Индекс разрешения экрана на данный момент
    /// </summary>
    public int curRes;

    #endregion

    private void Start()
    {
        Cursor.visible = true;

        resolutions = new Resolution[5];

        resolutions[0].width = 800;
        resolutions[0].height = 600;
        resolutions[1].width = 1024;
        resolutions[1].height = 768;
        resolutions[2].width = 1280;
        resolutions[2].height = 720;
        resolutions[3].width = 1360;
        resolutions[3].height = 768;
```

```
        resolutions[4].width = 1366;
        resolutions[4].height = 768;

        using (BinaryReader br = new BinaryReader(new FileStream("Resoludion.dat",
FileMode.Open)))
        {
            fullScr.isOn = br.ReadBoolean();

            int currentResolutionIndex = br.ReadInt32();
            resolutionDropDown.value = currentResolutionIndex;
            resolutionDropDown.RefreshShownValue();
            volumeSlider.value =br.ReadSingle();
            br.Close();
        }
        audioMixer.GetFloat("volume", out volume1);
        volumeSlider.value = volume1 * 3;
    }

    #region Functions

    /// <summary>
    /// Установка разрешения экрана
    /// </summary>
    /// <param name="resolutionIndex">Индекс элемента из выпадающего списка</param>
    public void SetResolution(int resolutionIndex)
    {
        curRes = resolutionIndex;
        Resolution resolution = resolutions[resolutionIndex];
        Screen.SetResolution(resolution.width, resolution.height, fullScr.isOn);

    }

    /// <summary>
    /// Сохранение настроек
    /// </summary>
    public void SaveSettings()
    {
        using (BinaryWriter bw = new BinaryWriter(new FileStream("Resoludion.dat",
FileMode.Create, FileAccess.Write)))
        {
            bw.Write(fullScr.isOn);
            Debug.Log(fullScr.isOn);
            bw.Write(resolutionDropDown.value);
            bw.Write(volumeSlider.value);
            bw.Close();
        }
    }

    /// <summary>
    /// Изменение громкости
    /// </summary>
    /// <param name="volume">Уровень громкости</param>
    public void SetVolume(float volume)
    {
        audioMixer.SetFloat("volume", volume / 3);
    }

    /// <summary>
    /// Установко полноэкранного режимы
    /// </summary>
    /// <param name="isFullscreen">Тагл "Fullscreen"</param>
    public void SetFullscreen(bool isFullscreen)
    {
        Screen.fullScreen = isFullscreen;
    }
```

```
    /// <summary>
    /// Переход на игровую сцену
    /// </summary>
    public void PlayButton()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    /// <summary>
    /// Переход на сцену Main Menu
    /// </summary>
    public void HelpButton()
    {
        SceneManager.LoadScene("Help");
    }

    /// <summary>
    /// Выход из игры
    /// </summary>
    public void QuitButton()
    {
        Application.Quit();
    }
    #endregion
}
```

Листинг модуля «BackToMenu.cs»

```
using UnityEngine;
using UnityEngine.SceneManagement;
/// <summary>
/// Start play
/// </summary>
public class BackToMenu : MonoBehaviour {

    /// <summary>
    /// Возвращает на сцену Main Menu
    /// </summary>
    public void BackBut()
    {
        SceneManager.LoadScene("MainMenu");
    }
}
```

ПРИЛОЖЕНИЕ Б – XML представление модулей

```
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>Assembly-CSharp</name>
    </assembly>
    <members>
        <member name="T:AiCharacter">
            <summary>
            Ai character
            </summary>
        </member>
        <member name="F:AiCharacter.pc">
            <summary>
            Управляемый персонаж
            </summary>
        </member>
        <member name="F:AiCharacter.en">
            <summary>
            Противник
            </summary>
        </member>
        <member name="F:AiCharacter.changeStateTolerance">
            <summary>
            Дистанция изменения состояния
            </summary>
        </member>
        <member name="F:AiCharacter.normalRate">
            <summary>
            Максимальное время нахождения на большой дистанции
            </summary>
        </member>
        <member name="F:AiCharacter.nrmTimer">
            <summary>
            Время нахождения на большой дистанции
            </summary>
        </member>
        <member name="F:AiCharacter.closeRate">
            <summary>
            Максимально время нахождения на маленькой дистанции
            </summary>
        </member>
        <member name="F:AiCharacter.clTimer">
            <summary>
            Время нахождения на маленькой дистанции
            </summary>
        </member>
        <member name="F:AiCharacter.blockingRate">
            <summary>
            Максимальная длительность блока
            </summary>
        </member>
        <member name="F:AiCharacter.blTimer">
            <summary>
            Длительность блока
            </summary>
        </member>
        <member name="F:AiCharacter.aiStateLife">
            <summary>
            Максимальная длительность состояния
            </summary>
        </member>
        <member name="F:AiCharacter.aiTimer">
            <summary>
```

```
    Длительность состояния
    </summary>
</member>
<member name="F:AiCharacter.initiateAI">
    <summary>
    Спровоцирован ли
    </summary>
</member>
<member name="F:AiCharacter.closeCombat">
    <summary>
    Близко ли к противнику
    </summary>
</member>
<member name="F:AiCharacter.gotRandom">
    <summary>
    Получено ли случайное число
    </summary>
</member>
<member name="F:AiCharacter.storeRandom">
    <summary>
    Случайное число
    </summary>
</member>
<member name="F:AiCharacter.blocking">
    <summary>
    Сработал ли блок
    </summary>
</member>
<member name="F:AiCharacter.randomizeAttacks">
    <summary>
    Сгенерирована ли атака
    </summary>
</member>
<member name="F:AiCharacter.numberOfAttacks">
    <summary>
    Количество сгенерированных атак
    </summary>
</member>
<member name="F:AiCharacter.curNumAttacks">
    <summary>
    Количество произведенных атак
    </summary>
</member>
<member name="F:AiCharacter.JumpRate">
    <summary>
    Частота прыжка
    </summary>
</member>
<member name="F:AiCharacter.jRate">
    <summary>
    Шанс прыжка
    </summary>
</member>
<member name="F:AiCharacter.jump">
    <summary>
    Прыгнул ли
    </summary>
</member>
<member name="F:AiCharacter.jtimer">
    <summary>
    Время без прыжка
    </summary>
</member>
<member name="T:AiCharacter.AIState">
    <summary>
```

```
    Структура состояний
    </summary>
</member>
<member name="F:AiCharacter.aiState">
    <summary>
    Состояния
    </summary>
</member>
<member name="M:AiCharacter.States">
    <summary>
    Установка состояния
    </summary>
</member>
<member name="M:AiCharacter.AIAgent">
    <summary>
    Действия АИ
    </summary>
</member>
<member name="M:AiCharacter.Attack">
    <summary>
    Атака
    </summary>
</member>
<member name="M:AiCharacter.Movement">
    <summary>
    Передвижение
    </summary>
</member>
<member name="M:AiCharacter.ResetAI">
    <summary>
    Обновление действий
    </summary>
</member>
<member name="M:AiCharacter.CheckDistance">
    <summary>
    Проверка дистанции с противником
    </summary>
</member>
<member name="M:AiCharacter.Blocking">
    <summary>
    Блокирование
    </summary>
</member>
<member name="M:AiCharacter.NormalState">
    <summary>
    Большая дистанция
    </summary>
</member>
<member name="M:AiCharacter.CloseState">
    <summary>
    Маленькая дистанция
    </summary>
</member>
<member name="M:AiCharacter.Jumping">
    <summary>
    Прыжок
    </summary>
</member>
<member name="M:AiCharacter.ReturnRandom">
    <summary>
    Возвращает случайное число
    </summary>
    <returns>Случайное число</returns>
</member>
<member name="T:DamageScript">
```

```
    <summary>
    Damage script
    </summary>
</member>
<member name="F:DamageScript.damage">
    <summary>
    Наносимый урон
    </summary>
</member>
<member name="M:DamageScript.OnTriggerEnter2D(UnityEngine.Collider2D)">
    <summary>
    Соприкосновение коллайдера кулака с объектами
    </summary>
    <param name="other">Коллайдер, с которым соприкасается кулак</param>
</member>
<member name="T:EventHolder">
    <summary>
    Event holder
    </summary>
</member>
<member name="F:EventHolder.pl">
    <summary>
    Персонаж
    </summary>
</member>
<member name="M:EventHolder.ThrowProjectile">
    <summary>
    Специальная атака
    </summary>
</member>
<member name="T:MainHero">
    <summary>
    Main hero
    </summary>
</member>
<member name="F:MainHero.PlayerNumber">
    <summary>
    Индекс персонажа
    </summary>
</member>
<member name="F:MainHero.AiCharacter">
    <summary>
    Управляется ли персонаж с помощью АИ
    </summary>
</member>
<member name="F:MainHero.enemy">
    <summary>
    Противник
    </summary>
</member>
<member name="F:MainHero.rb">
    <summary>
    Ригидбоди объекта
    </summary>
</member>
<member name="F:MainHero.anim">
    <summary>
    Аниматор объекта
    </summary>
</member>
<member name="F:MainHero.horizontal">
    <summary>
    Перемещение по оси X
    </summary>
</member>
```

```
<member name="F:MainHero.vertical">
    <summary>
    Перемещение по оси Y
    </summary>
</member>
<member name="F:MainHero.maxSpeed">
    <summary>
    Максимальная скорость передвежения персонажа
    </summary>
</member>
<member name="F:MainHero.movement">
    <summary>
    Вектор передвежения
    </summary>
</member>
<member name="F:MainHero.crouch">
    <summary>
    Сидит ли персонаж
    </summary>
</member>
<member name="F:MainHero.JumpForce">
    <summary>
    Сила прыжка (устанавливается из дебаг окна)
    </summary>
</member>
<member name="F:MainHero.jmpForce">
    <summary>
    Сила прыжка
    </summary>
</member>
<member name="F:MainHero.JumpDuration">
    <summary>
    Максимальная продолжительность прыжка
    </summary>
</member>
<member name="F:MainHero.jmpDuration">
    <summary>
    Продолжительность прыжка
    </summary>
</member>
<member name="F:MainHero.jumpKey">
    <summary>
    Прыгает ли
    </summary>
</member>
<member name="F:MainHero.falling">
    <summary>
    Падает ли
    </summary>
</member>
<member name="F:MainHero.onGround">
    <summary>
    Стоит ли на земле
    </summary>
</member>
<member name="F:MainHero.attackRate">
    <summary>
    Скорость атаки
    </summary>
</member>
<member name="F:MainHero.attack">
    <summary>
    Массив с вариантами атаки
    </summary>
</member>
```

```
<member name="F:MainHero.attacktimer">
    <summary>
    Продолжительность нажатий клавиш атаки
    </summary>
</member>
<member name="F:MainHero.timesPressed">
    <summary>
    Количество нажатий клавиш атаки
    </summary>
</member>
<member name="F:MainHero.damage">
    <summary>
    Нанесен ли урон
    </summary>
</member>
<member name="F:MainHero.noDamage">
    <summary>
    Максимальная продолжительность неуязвимости после получения урона
    </summary>
</member>
<member name="F:MainHero.noDamageTimer">
    <summary>
    Продолжительность неуязвимости после получения урона
    </summary>
</member>
<member name="F:MainHero.blocking">
    <summary>
    Сработал ли блок
    </summary>
</member>
<member name="F:MainHero.blockingRate">
    <summary>
    Максимальная продолжительность блока
    </summary>
</member>
<member name="F:MainHero.blTimer">
    <summary>
    Продолжительность блока
    </summary>
</member>
<member name="F:MainHero.specialAttack">
    <summary>
    Нажата ли клавиша специальной атаки
    </summary>
</member>
<member name="F:MainHero.prLifeTime">
    <summary>
    Время жизни прожектайла специальной атаки
    </summary>
</member>
<member name="F:MainHero.gotRandom">
    <summary>
    Получено ли случайное число
    </summary>
</member>
<member name="F:MainHero.storeRandom">
    <summary>
    Случайное число
    </summary>
</member>
<member name="F:MainHero.projectile">
    <summary>
    Прожектайл
    </summary>
</member>
```

```
<member name="F:MainHero.am">
    <summary>
    Аудиомикшер
    </summary>
</member>
<member name="F:MainHero.Au">
    <summary>
    Источник звука удара
    </summary>
</member>
<member name="F:MainHero.die">
    <summary>
    Упало ли здоровье ниже 0
    </summary>
</member>
<member name="F:MainHero.deathAnimLife">
    <summary>
    Максимальное время работы игры после смерти персонажа
    </summary>
</member>
<member name="F:MainHero.deathAnimTimer">
    <summary>
    Время работы игры после смерти персонажа
    </summary>
</member>
<member name="F:MainHero.healthBar">
    <summary>
    Слайдер со здоровьем персонажа
    </summary>
</member>
<member name="P:MainHero.CurrentHealth">
    <summary>
    Текущее здоровье
    </summary>
</member>
<member name="P:MainHero.MaxHealth">
    <summary>
    Максимальное здоровье
    </summary>
</member>
<member name="F:MainHero.ContinueText">
    <summary>
    Текст "Continue"
    </summary>
</member>
<member name="F:MainHero.WinText">
    <summary>
    Текст "...WIN"
    </summary>
</member>
<member name="F:MainHero.plH">
    <summary>
    Объект со слайдером здоровья (для паузы)
    </summary>
</member>
<member name="F:MainHero.paused">
    <summary>
    Нажата ли пауза
    </summary>
</member>
<member name="F:MainHero.volume">
    <summary>
    Уровень громкости
    </summary>
</member>
```

```
<member name="F:MainHero.pauseMenu">
    <summary>
    Объект меню паузы
    </summary>
</member>
<member name="M:MainHero.OnGroundCheck">
    <summary>
    Изменение гравитации в зависимости от положения
    </summary>
</member>
<member name="M:MainHero.AttackInput">
    <summary>
    Нажатие клавиш атаки
    </summary>
</member>
<member name="M:MainHero.Damage">
    <summary>
    Получение урона
    </summary>
</member>
<member name="M:MainHero.SpecialAttack">
    <summary>
    Специальная атака
    </summary>
</member>
<member name="M:MainHero.UpdateAnimator">
    <summary>
    Обновление анимации
    </summary>
</member>
<member name="M:MainHero.OnCollisionEnter2D(UnityEngine.Collision2D)">
    <summary>
    Проверка на соприкосновение с полом
    </summary>
    <param name="col">Коллайдер, с которым соприкасается персонаж</param>
</member>
<member name="M:MainHero.OnCollisionExit2D(UnityEngine.Collision2D)">
    <summary>
    Проверка на отталкивание от пола
    </summary>
    <param name="col">Коллайдер, от которого отталкивается персонаж</param>
</member>
<member name="M:MainHero.ScaleCheck">
    <summary>
    Разворот в сторону противника
    </summary>
</member>
<member name="M:MainHero.GameOver">
    <summary>
    Окончание игры
    </summary>
</member>
<member name="M:MainHero.CalculateHealth">
    <summary>
    Возвращает текущее здоровье
    </summary>
    <returns>Текущее здоровье</returns>
</member>
<member name="M:MainHero.Blocking">
    <summary>
    Блокирование
    </summary>
</member>
<member name="M:MainHero.ReturnRandom">
    <summary>
```

```
    Возвращает случайное число
    </summary>
    <returns>Случайное число</returns>
</member>
<member name="M:MainHero.PauseApp">
    <summary>
    Ставит игру на паузу
    </summary>
</member>
<member name="M:MainHero.ContinueBut">
    <summary>
    Продолжает игру (Кнопка "Continue")
    </summary>
</member>
<member name="M:MainHero.MenuBut">
    <summary>
    Переход на сцену с главным меню (Кнопка "Main menu")
    </summary>
</member>
<member name="M:MainHero.QuitBut">
    <summary>
    Выход из игры (Кнопка "Quit")
    </summary>
</member>
<member name="T:PausePlay">
    <summary>
    Pause play
    </summary>
</member>
<member name="F:PausePlay.pauseMenu">
    <summary>
    Объект меню паузы
    </summary>
</member>
<member name="T:Player1SetUp">
    <summary>
    Player1 set up
    </summary>
</member>
<member name="M:Player1SetUp.PlayVs(System.Int32)">
    <summary>
    Устанавливает тип управления персонажем 1
    </summary>
    <param name="dropDownIndex">Индекс из выпадающего списка</param>
</member>
<member name="T:Player2SetUp">
    <summary>
    Player2 set up
    </summary>
</member>
<member name="M:Player2SetUp.PlayVs(System.Int32)">
    <summary>
    Устанавливает тип управления персонажем 2
    </summary>
    <param name="dropDownIndex">>Индекс из выпадающего списка</param>
</member>
<member name="T:Redirect">
    <summary>
    Redirect.
    </summary>
</member>
<member name="F:Redirect.logoDuration">
    <summary>
    Продолжительность показа лого
    </summary>
```

```
</member>
<member name="F:Redirect.maxLogoDuration">
    <summary>
    Максимальная продолжительность показа лого
    </summary>
</member>
<member name="T:SettingsMenu">
    <summary>
    Settings menu
    </summary>
</member>
<member name="F:SettingsMenu.audioMixer">
    <summary>
    Аудиомикшер
    </summary>
</member>
<member name="F:SettingsMenu.volumeSlider">
    <summary>
    Слайдер громкости
    </summary>
</member>
<member name="F:SettingsMenu.volume1">
    <summary>
    Уровень громкости
    </summary>
</member>
<member name="F:SettingsMenu.fullScr">
    <summary>
    Тагл "Fullscreen"
    </summary>
</member>
<member name="F:SettingsMenu.resolutionDropDown">
    <summary>
    Выпадающий список с разрешениями экрана
    </summary>
</member>
<member name="F:SettingsMenu.resolutions">
    <summary>
    Массив разрешений экрана
    </summary>
</member>
<member name="F:SettingsMenu.curRes">
    <summary>
    Индекс разрешения экрана на данный момент
    </summary>
</member>
<member name="M:SettingsMenu.SetResolution(System.Int32)">
    <summary>
    Установка разрешения экрана
    </summary>
    <param name="resolutionIndex">Индекс элемента из выпадающего списка</param>
</member>
<member name="M:SettingsMenu.SaveSettings">
    <summary>
    Сохранение настроек
    </summary>
</member>
<member name="M:SettingsMenu.SetVolume(System.Single)">
    <summary>
    Изменение громкости
    </summary>
    <param name="volume">Уровень громкости</param>
</member>
<member name="M:SettingsMenu.SetFullscreen(System.Boolean)">
    <summary>
```

```
            Установко полноэкранного режимы
            </summary>
            <param name="isFullscreen">Тагл "Fullscreen"</param>
        </member>
        <member name="M:SettingsMenu.PlayButton">
            <summary>
            Переход на игровую сцену
            </summary>
        </member>
        <member name="M:SettingsMenu.HelpButton">
            <summary>
            Переход на сцену Main Menu
            </summary>
        </member>
        <member name="M:SettingsMenu.QuitButton">
            <summary>
            Выход из игры
            </summary>
        </member>
        <member name="T:BackToMenu">
            <summary>
            Start play
            </summary>
        </member>
        <member name="M:BackToMenu.BackBut">
            <summary>
            Возвращает на сцену Main Menu
            </summary>
        </member>
    </members>
</doc>
```