

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**   
**«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ»**

Факультет Физико-технический

Кафедра Компьютерных технологий (КТ)

|  |  |  |
| --- | --- | --- |
|  | | |
| Зав. кафедрой | | КТ |
| \_\_\_\_\_\_\_\_\_\_\_\_ | | Г.В. Аверин |
| (подпись) | |  |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. | |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе бакалавра 2 курса

на тему:

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ОБРАБОТКИ ТЕКСТА

В ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЯХ

|  |  |  |  |
| --- | --- | --- | --- |
| Автор работы |  |  | Карамян Э.А. |

подпись

|  |  |  |
| --- | --- | --- |
| Направление | 09.03.01 | Информатика и вычислительная техника |

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ассистент Маруга.М.М

подпись

Консультанты по разделам:

Техническое задание \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент Т.В. Шарий

подпись

Нормоконтроль \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. лаборант В.Г. Медведева

подпись

Курсовая работа защищена \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата итоговая оценка комиссия

Подписи членов комиссии: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Донецк  
2021

ГОУ ВПО «Донецкий национальный университет»

кафедра Компьютерных технологий

Утверждаю

Зав. кафедрой КТ

\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

**ЗАДАНИЕ**

на курсовую работу студента 2 курса Карамян Э.А.

*Тема курсовой работы*: Приложение для обработки и улучшения гитарного звука

*Краткая постановка задачи*: 1. 1. Изучить и проанализировать предметную область по созданию приложений для работы со звуком 2. Ознакомиться с программными продуктами и инструментами для создания приложений для прослушивания и проигрывания звука; 3. Разработать техническое задание на создание приложения. 4. Разработать проект программного обеспечения системы. 5. Разработать средствами языка C# приложение, которое позволяет взаимодействовать со звуком в приложении. 6. Протестировать программное обеспечение. 7. Оформить отчёт.

Исходные данные: 1. Документация библиотеки CSCORE. 2. Документация по языку программирования C#. 3. Набор графических элементов для оформления приложения.

*Календарный план работы:*

|  |  |  |
| --- | --- | --- |
| Даты консультаций | Этапы выполнения работы | Отметки о выполнении |
| 30.01.2021 | Постановка задачи и обсуждение литературы | выполнено |
| 07.02.2021 | Предварительное утверждение содержания отчёта | выполнено |
| 15.02.2021 | Утверждение проекта, алгоритмов, методов, технологий | выполнено |
| 10.03.2021 | Ход реализации проекта | выполнено |
| 25.03.2021 | Обсуждения организации тестирования программы | выполнено |
| 14.04.2021 | Демонстрация программного продукта руководителю | выполнено |
| 15.04.2021 | Оформление отчёта | выполнено |
| 18.04.2021 | Предоставление отчёта руководителю | выполнено |

Дата выдачи задания 30.01.2021 года

Студент Э.А.Карамян

Руководитель М.М.Маруга

**АННОТАЦИЯ**

Отчет о курсовой работе: 100 с., 48 рис., 14 табл., 2 приложения, 12 источников.

Объект исследования – Библиотека для работы со звуком CSCORE.

Предмет исследования – Приложение для обработки и улучшения гитарного звука.

Цель работы – разработка приложения, которое позволяло бы записывать, прослушивать и обрабатывать звук электрогитары.

Метод исследования – анализ возможностей языка С#, предметов с аналогичными функциями из реального мира и библиотеки CSCORE для работы со звуком.

В работе были использованы возможности .NET Framework, библиотека CSCORE

В результате решения задачи было разработано приложение для компьютера на основе операционной системы Windows. Приложение позволяет проигрывать звук с входного сигнала в реальном времнеи, накладывать эффекты и записывать звук с входного канала.

Главный модуль приложения работает с библиотекой CSCORE. Главной операцией приложения является проигрывание звука в реальном времени.

Приложение Live\_play может использоваться музыкантами для самостоятельных занятий и репетиций.

С#, ПРОГРАММНЫЙ ПРОДУКТ, ПРИЛОЖЕНИЕ ПОД WINDOWS, МУЗЫКА,CSCORE.

**СОДЕРЖАНИЕ**

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc494639103)

[1.1 Основные понятия компьютерной игры 6](#_Toc494639104)

[1.2 Актуальность и цель работы 6](#_Toc494639105)

[2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 8](#_Toc494639106)

[2.1 Описание области применения и исходных данных приложения 8](#_Toc494639107)

[2.2 Требования к пользовательским интерфейсам 8](#_Toc494639108)

[2.3 Требования к аппаратным, программным и коммуникационным интерфейсам 8](#_Toc494639109)

[2.4 Требования к пользователям продукта 9](#_Toc494639110)

[2.5 Требования к адаптации на месте 9](#_Toc494639111)

[2.6 Функции продукта 10](#_Toc494639112)

[2.7 Ограничения 11](#_Toc494639113)

[3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ 12](#_Toc494639114)

[4 РАЗРАБОТКА ПРИЛОЖЕНИЯ (ТЕМА КУРСОВОЙ РАБОТЫ) 13](#_Toc494639115)

[4.1 Входные и выходные данные приложения 13](#_Toc494639116)

[4.2 Проектирование структуры приложения 13](#_Toc494639117)

[4.3 Описание алгоритмов работы скриптов системы или (Описание объектов и их взаимодействия для ООП) 15](#_Toc494639118)

[5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 16](#_Toc494639119)

[5.1 Аппаратные и программные средства создания и эксплуатации интернет-приложения 16](#_Toc494639120)

[5.2 Руководство пользователя 16](#_Toc494639121)

[5.3 Описание контрольных примеров 17](#_Toc494639122)

[ЗАКЛЮЧЕНИЕ 18](#_Toc494639123)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc494639124)

[ПРИЛОЖЕНИЕ А Экранные формы 20](#_Toc494639125)

[ПРИЛОЖЕНИЕ Б Фрагменты листинга 21](#_Toc494639126)

**ВВЕДЕНИЕ**

…………………………………………………………………………

Данное приложение основано на предмете из реального мира – гитарном усилителе, так что для понимания работы приложения стоит уточнить что это такое.

Гитарный усилитель  — это [электронный усилитель](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D1%83%D1%81%D0%B8%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C), предназначенный для использования совместно с электрическими и электронными музыкальными инструментами, в частности, [электрогитарами](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%B3%D0%B8%D1%82%D0%B0%D1%80%D0%B0).

Первые музыкальные усилители были изобретены для усиления звука гитары. Самые ранние примеры подобных устройств относятся к началу [1930-х годов](https://ru.wikipedia.org/wiki/1930-%D0%B5), когда появление [электролитических конденсаторов](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BB%D0%B8%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BD%D0%B4%D0%B5%D0%BD%D1%81%D0%B0%D1%82%D0%BE%D1%80) и выпрямительных ламп позволило построить экономичный встроенный блок питания с питанием от сети переменного тока. В результате переносные усилители на [электронных лампах](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BB%D0%B0%D0%BC%D0%BF%D0%B0) перестали зависеть от тяжёлых и неудобных наборов [батарей](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D0%BA%D1%83%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B0%D1%82%D0%B0%D1%80%D0%B5%D1%8F) питания. Изначально гитарные усилители применялись для усиления звука [акустических гитар](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%83%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B3%D0%B8%D1%82%D0%B0%D1%80%D0%B0). Электрическое усиление гитары получило распространение в [1930-х](https://ru.wikipedia.org/wiki/1930-%D0%B5) и [1940-х](https://ru.wikipedia.org/wiki/1940-%D0%B5) годах на волне популярности [гавайской](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D0%B2%D0%B0%D0%B9%D0%B8) музыки, в которой широко применялась усиленная [гавайская гитара](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D0%B2%D0%B0%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D0%B3%D0%B8%D1%82%D0%B0%D1%80%D0%B0).

Управление тембром в ранних гитарных усилителях было очень простым и в основном заключалось в усилении высокочастотной компоненты звука. Однако ранняя схемотехника, несовершенные громкоговорители и низкая мощность (менее 15 ватт вплоть до середины [1950-х годов](https://ru.wikipedia.org/wiki/1950-%D0%B5)) первых усилителей не позволяла качественно воспроизводить верхнюю и нижнюю часть звукового спектра. Лучшие модели тех времён также содержали эффекты, например пружинный [ревербератор](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B2%D0%B5%D1%80%D0%B1%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80) или электронное [тремоло](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D0%BC%D0%BE%D0%BB%D0%BE). В ранних усилителях фирмы [Fender](https://ru.wikipedia.org/wiki/Fender) тремоло обозначалось как «[вибрато](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B1%D1%80%D0%B0%D1%82%D0%BE)», а рычаг вибрации на гитаре [Fender Stratocaster](https://ru.wikipedia.org/wiki/Fender_Stratocaster) назывался (и называется до сих пор) «рычаг тремоло».

В [1960-х годах](https://ru.wikipedia.org/wiki/1960-%D0%B5) [гитаристы](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D1%82%D0%B0%D1%80%D0%B8%D1%81%D1%82) экспериментировали с искажениями звука, преднамеренно [перегружая](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D0%BE%D1%80%D1%88%D0%BD) свои усилители. Гитарист группы [The Kinks](https://ru.wikipedia.org/wiki/The_Kinks) [Дэйв Дэвис](https://ru.wikipedia.org/wiki/%D0%94%D1%8D%D0%B9%D0%B2_%D0%94%D1%8D%D0%B2%D0%B8%D1%81) получал эффект перегруза, подключая выход одного усилителя ко входу другого — способ использования, который разработчики даже не могли себе представить. Позже большинство гитарных усилителей стали комплектоваться специальным перегружаемым предусилителем, а также были придуманы [педали эффектов](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D0%B4%D0%B0%D0%BB%D1%8C_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82%D0%BE%D0%B2) и прочее оборудование для безопасного и надёжного получения таких звуков. Сегодня искажённое звучание стало неотъемлемой частью многих стилей игры на электрогитаре.

С появлением [микропроцессоров](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) и цифровой обработки сигналов в конце [1990-х годов](https://ru.wikipedia.org/wiki/1990-%D0%B5) были разработаны [«моделирующие усилители»](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B3%D0%B8%D1%82%D0%B0%D1%80%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80), которые могут симулировать звук многих известных ламповых усилителей без использования ламп в схеме. Новые усилители со специальными [процессорами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) и [программным обеспечением](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) могут [эмулировать](https://ru.wikipedia.org/wiki/%D0%AD%D0%BC%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) ламповый звук классических усилителей. Главный недостаток цифровой обработки сигнала состоит в том, что теряются некоторые оттенки, нюансы звукоизвлечения и звучания различных гитар, в то время, как для опытного гитариста, каждый инструмент «звучит» по-своему. При этом само звучание остаётся довольно хорошим.

Так же можно провести параллель между данным приложением и программным гитарным симулятором.

Программный гитарный симулятор или виртуальный гитарный процессор — это программа-эмулятор (симулятор) процессов, происходящих в гитарном тракте различных устройств: эффект-педалей, рэковых устройств, микрофонов, [гитарных усилителей](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D1%82%D0%B0%D1%80%D0%BD%D1%8B%D0%B9_%D1%83%D1%81%D0%B8%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C) и т. д.. при помощи специальных программных алгоритмов моделирования работы этих (физических) устройств электронной обработки сигнала с электрогитары, тем самым декларируя замену дорогой электронной аппаратуры. Однако из-за резкого падения цен на эффект-педали и аппаратные гитарные процессоры в конце 20 и начала 21 века не всегда декларации совпадают с реальностью. Цена на программные продукты такого сорта порой превышает цену аналогичных аппаратных решений и даже самих компьютеров на которых они могут работать (данные могут быть проверены любым читателем этой статьи просмотром каталога цен любого интернет-магазина типа MusciansFriend).

Первые варианты виртуальных гитарных процессоров появились в конце 90-х годов прошлого века. Связано это было с тем, что общедоступны стали персональные компьютеры на процессорах i486 и Pentium, обладающих достаточной мощностью для вычислений в реальном времени необходимых для обработки гитарных сигналов. Простейшим примером может служить вышедшая в 1997г программа виртуального процессора [GuitarFX](http://guitarfx.net/). Эта разработка в реальном времени эмулировала дисторшн, ревербератор, нойс гейт, многополосный эквалайзер, хорус, дилэй. Пример звука.

Различные виртуальные гитарные процессоры (симуляторы) обычно имеют форматы DX (Direct-X), [VST](https://ru.wikipedia.org/wiki/Virtual_Studio_Technology) плагина, так и RTAS (игра в реальном времени). Удобны для быстрого развёртывания. Имеют достаточно убедительное звучание, хотя, конечно, абсолютно все нюансы в обработке аналоговой (реальной) гитарной электронной аппаратуры, возможно, пока учесть сложно, но это не мешает с успехом использовать виртуальные гитарные процессоры в записи и в живом исполнении, как молодыми исполнителями, так и опытными.

Основной недостаток этих «устройств» — это временная (естественная) жертва — хоть и небольшая, и практически не ощутимая, но задержка по времени в обработке, чем обычно пренебрегают, (используя бесплатный драйвер ASIO4All) устанавливая величину этой задержки минимальной, при которой компьютер ещё справляется с обработкой аудиопотока отдавая на выход звуковой карты его без «тресков и бульканий» — красивым.

Одними из самых популярных гитарных симуляторов являются программы [Guitar Rig](https://ru.wikipedia.org/wiki/Guitar_Rig) (фирмы [Native Instruments](https://ru.wikipedia.org/wiki/Native_Instruments)), [TH2](https://ru.wikipedia.org/w/index.php?title=TH2&action=edit&redlink=1) (Overloud) и [AmpliTube](https://ru.wikipedia.org/w/index.php?title=AmpliTube&action=edit&redlink=1) от IK media. Имеют форматы [VST](https://ru.wikipedia.org/wiki/Virtual_Studio_Technology) (плагин-обработчик) и Standalone (процессор, для игры в реальном времени)

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

**1.1. Основные понятия компьютерной игры**

Разработка приложений для работы с гитарным звуком - это процесс создания приложений для взаимодействия со звуком приходящим с входного канала. Разработкой данных приложений занимается программист, который может быть представлен как одним человеком, так и фирмой.

Обычно крупномасштабные коммерческие приложения разрабатываются командами программистами в пределах компании, специализирующейся на приложениях для персонального компьютера или смартфона.

Как правило, разработку финансирует другая, более крупная компания, которая по окончанию разработки занимается продвижением приложения и связанными с ним тратами.

Коммерческие приложения зачастую имеют большую функциональность, нежели open-source или условно-бесплатные, например большее количество эффектов, возможность параллельной записи и прослушивания, огромное количество pre-amp и amp моделей.

1.2 Актуальность и цель работы

Музыка всегда была популярна и важна для массового потребителя, гитаристов сегодня намного больше, чем 40 лет назад, когда компьютерные технологии были хуже развиты и пользуясь техническим прогрессом можно развивать и сферу искусства создавая подобные приложения. Так же присутствует финансовый фактор гитарные усилители для обычного пользователя стоят много, но все же желание создавать музыку есть, но так как в каждом доме есть компьютер можно пользоваться приложениями подобными данному.

Актуальность создания приложения для работы с гитарным звуком неоспоримо, так как конкуренция мала, так как на рынке в данной сфере есть несколько монополистов и их позиции должны быть ослаблены более простыми в использовании и недорогими, а возможно и бесплатными приложениями.

Музыка появилась еще 50 тысяч лет назад, но это не столь важно, ведь мы рассматриваем обработку гитарного звука, электро-гитары же появились не так давно в 1936 году изобретенная Жержем Бошамом и Адольфом Рикенбекром, но отличием от сегодняшних инструментов являлся металлический корпус. В начале 50-х годов [Лес Пол](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D1%81_%D0%9F%D0%BE%D0%BB) изобретает электрическую гитару со сплошным деревянным корпусом, но позже отдаёт идею Лео Фендеру, так как идея цельнокорпусной гитары не вызвала интереса компании Gibson, где работал Лес Пол. Конструкция электрогитары сохранилась без изменений до настоящего времени.

Так как сама по себе электрогитара очень тихая и в ней присутствует звукосниматель исходит логический вывод, что ее нужно куда-то подключать, а кроме компьютера обычному пользователю ее подключать некуда и для использования всевозможных эффектов и записи нужен софт.

**2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

2.1 Описание области применения и исходных данных приложения

Основным средством разработки данного проекта является язык программирования C# с библиотекой CSCORE. CSCore является бесплатноц .NET аудио библиотекой, полностью написанной на C#. Этот проект предлагает множество функций, таких как воспроизведение или захват звука, кодирование или декодирование множества различных кодеков, эффектов и многое другое. Дает возможность создавать музыкальные плееры, голосовые чаты, диктофоны и так далее. Созданные с помощью CSCore приложения работают под операционной системой Windows.

Приложения, созданные с помощью CSCore, поддерживают ASIO.

**Поддерживаемые функции CSCore:**

-Обработка звука в реальном времени

-Обработка аудиоданных в реальном времени

-Применяйте любые процессоры в любом порядке в реальном времени

-Создавайте собственные процессоры (например, эффекты, анализ, декодеры, и т.д.)

**Кодеки:**

-MP3

-WAVE (PCM, IeeeFloat, GSM, ADPCM, и т.д.)

-FLAC

-AAC

-AC3

-WMA

-Raw data

-OGG-Vorbis (через NVorbis)

-FFmpeg (множество дополнительных форматов, см. CSCore.Ffmpeg)

**Поддержка FFmpeg:**

Поддерживается через CSCore.Ffmpeg)

**Выход на динамик:**

-WaveOut

-DirectSoundOut (потоковая передача в реальном времени)

WASAPI (обратный вызов цикла и события + доступен эксклюзивный режим)

-XAudio2

**Запись:**

-WaveIn

-WASAPI (циклический обратный вызов и обратный вызов событий + доступен эксклюзивный режим)

-WASAPI loopback capture (захват вывода со звуковой карты)

**Алгоритмы DSP:**

-Быстрое преобразование Фурье (БПФ)

-Эффекты (эхо, компрессор, реверберация, хорус, полоскание, флэнджер, ...)

-Ресамплер

-Микширование каналов с использованием настраиваемых матриц каналов

-Общий эквалайзер

**поддержка XAudio2:**

-Поддержка XAudio2.7 и XAudio2.8

-Поддержка 3D Audio (см. Образец X3DAudio)

-Реализация потоковой передачи исходного голоса, позволяющая клиенту передавать любой кодек, настраиваемый эффект ... в XAudio2

-Оптимизирован для игр

**-Кодирование и декодирование Mediafoundation**

**-Оболочка объектов мультимедиа DirectX**

**-Оболочка CoreAudioAPI**

-WASAPI

-Мультимедийные устройства Windows

-Аудиосеанс Windows

-Объем конечной точки

**-Многоканальная поддержка**

**-Гибкий**

-Присутствует возможность настраивать под себя любые части CSCore

-Можно использовать низкие значения задержки для производительности в реальном времени, высокие значения задержки для стабильности

-Отрегулируйте качество звука

-Настроить матрицы клиентских каналов

-Создавайте собственные эффекты

-Или просто: сделайте CSCore отвечающим вашим потребностям!

-Теги (ID3v1, ID3v2, FLAC)

-Оптимизированная производительность за счет использования инструкций CLI, предоставляемых настраиваемым компилятором сообщений.

В качестве целевой платформы разработки выбраны операционные системы семейства MS Windows (Vista/7/8/10).

Основным языком программирования является C# с платформой.NET

2.2 Требования к пользовательским интерфейсам

Интерфейс разрабатываемого приложения должен быть простым в использовании, интуитивно понятным, функциональным.

Цветовая гамма приложения должна быть не особо насыщенной, лаконичной, чтобы не рассеивать внимание пользователя.

Наличие широкого набора иерархических меню, подсказок и т.п., облегчающих процесс взаимодействия пользователя с приложением;

2.3 Требования к аппаратным, программным и коммуникационным интерфейсам

Для установки и работы приложения необходимо иметь вычислительную систему следующей минимальной аппаратной конфигурации:

Минимальные системные требования:

– ОС (операционная система): Windows Vista/7/8/10;

– Процессор: Intel Core 2 Duo @ 3.0 Ghz / AMD Athlon 64 X2 6000+;

– Оперативная память: 1 Gb;

– Жесткий диск: 10 Gb свободно (для записей);

– Клавиатура, Мышь, Наушники или динамики, линейный/микрофонный вход на материнской плате.

Рекомендуемые системные требования:

 ОС (операционная система): Windows Vista/7/8/10;

 Процессор: Intel Core i5 @ 3.2 GHz / AMD Phenom II X4 @ 3.6 GHz;

 Оперативная память: 4 Gb;

 Жесткий диск: 20 Gb свободно(для записей);

 Клавиатура, Мышь, Наушники или динамики, линейный/микрофонный вход на материнской плате.

2.5. Требования к информационной и программной совместимости

Программа должна функционировать под управлением ОС семейства Windows следующих версий: Windows Vista, 7, 8, 10.

В приложении используются библиотеки платформы.NET Framework.

Необходимо обеспечить программное взаимодействие системы с:

1. MS dotNet Framework 4.5+;
2. Драйвер для аудио устройства

2.4 Требования к пользователям продукта

Программа является продуктом сферы приложений для работы с аудио.

Программный продукт создан для обычных пользователей желающих прикоснуться к музыкальной сфере а виде игры на электрогитаре.

Приложение не имеет возрастных или системных ограничений и может быть запущена практически на любом компьютере.

2.5 Функции продукта

Подробный перечень функций.

1. Запись входящего аудиосигнала
2. Проигрывание выбранного файла
3. Проигрывание входящего сигнала в реальном времени
4. Наложение эффектов на аудио в реальном времени

5) Повышение/понижение громкости при проигрывании записи

6) Перемотка записи

7) Выбор устройства вывода при проигрывании записи

**2.6 Общая функциональная схема программы**

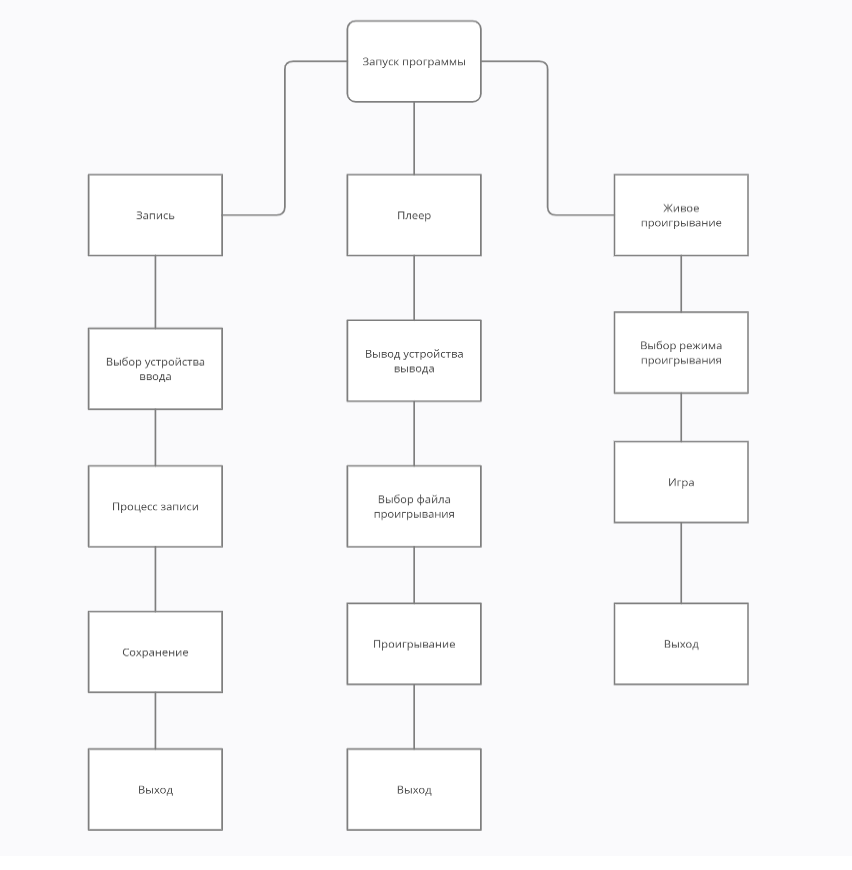
****

Рисунок 1 - Общая функциональная схема

ОБЯЗАТЕЛЬНО ДОБАВИТЬ

Рисунок 2 - Диаграмма классов

# 3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ



# Рисунок 3.1. Логотип CSCore.

CSCore — Гибкая библиотека для работы со звуком с огромной перспективой на развитие проекта в будущем.

CSCore позволяет настраивать звук и взаимодействовать с ним в огромном количестве форматов.

Выпуск CSCore состоялся в 2015 году и с того времени идёт постоянное развитие.

Основными преимуществами CSCORE являются:

-Высокооптимизированная ПРОИЗВОДИТЕЛЬНОСТЬ за счет использования инструкций интерфейса командной строки

-Создан для новичков и профессионалов

-Тонны функций

-Быстрая поддержка на github и stackoverflow

-Высокое покрытие кода с помощью модульных тестов

-Лицензировано в соответствии с MS-PL (не включая проект CSCore.Ffmpeg, который находится под лицензией LGPL).

Особенности для обоснования выбора:

Использование данной библиотеки значительно упростило взаимодейстиве с аудио и позволило реализовать функцию проигрывания в реальном времени с применением эффектов. Так же в CSCore упрощено взаимодействие с эффектами, то есть все эффекты уже реализованы в библиотеке, их нужно только объявить и дать им соответствующие параметры. Простая работа с входными и выходными каналами звука. Так же к плюсам выбора данной библиотеки можно отнести высокую производительность и простую оптимизацию кода для последующего усовершенствования проекта в будущем.

# Рисунок 3.2. Пример Кода

# What's New in Visual Studio 2019 version 16.5 Preview 2 for C++, Xamarin, and Azure Tooling Experiences / Блог компании Microsoft / Хабр

# Рисунок 3.3. Логотип Visual Studio

**Visual Studio** — **линейка продуктов компании**[**Microsoft**](https://ru.wikipedia.org/wiki/Microsoft)**, включающих**[**интегрированную среду разработки**](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)**программного обеспечения и ряд других инструментальных инструментов. Данные продукты позволяют разрабатывать как**[**консольные**](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F)[**приложения**](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)**, так и игры и приложения с**[**графическим интерфейсом**](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F)**, в том числе с поддержкой технологии**[**Windows Forms**](https://ru.wikipedia.org/wiki/Windows_Forms)**, а также**[**веб-сайты**](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82)**,**[**веб-приложения**](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)**,**[**веб-службы**](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0)**как в**[**родном**](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)**, так и в**[**управляемом**](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)**кодах для всех платформ, поддерживаемых**[**Windows**](https://ru.wikipedia.org/wiki/Windows)**,**[**Windows Mobile**](https://ru.wikipedia.org/wiki/Windows_Mobile)**,**[**Windows CE**](https://ru.wikipedia.org/wiki/Windows_CE)**,**[**.NET Framework**](https://ru.wikipedia.org/wiki/.NET_Framework)**,**[**Xbox**](https://ru.wikipedia.org/wiki/Xbox)**,**[**Windows Phone**](https://ru.wikipedia.org/wiki/Windows_Phone)[**.NET Compact Framework**](https://ru.wikipedia.org/wiki/.NET_Compact_Framework)**и**[**Silverlight**](https://ru.wikipedia.org/wiki/Silverlight)**.**

**Visual Studio включает в себя**[**редактор исходного кода**](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0)**с поддержкой технологии**[**IntelliSense**](https://ru.wikipedia.org/wiki/IntelliSense)**и возможностью простейшего**[**рефакторинга кода**](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3)**. Встроенный**[**отладчик**](https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger)**может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер**[**классов**](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))**и дизайнер**[**схемы базы данных**](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)**. Visual Studio позволяет создавать и подключать сторонние дополнения (**[**плагины**](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD)**) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем**[**контроля версий исходного кода**](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8)**(как, например,**[**Subversion**](https://ru.wikipedia.org/wiki/Subversion)**и**[**Visual SourceSafe**](https://ru.wikipedia.org/wiki/Microsoft_Visual_SourceSafe)**), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на**[**предметно-ориентированных языках программирования**](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)**) или инструментов для прочих аспектов**[**процесса разработки программного обеспечения**](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F)**(например, клиент Team Explorer для работы с**[**Team Foundation Server**](https://ru.wikipedia.org/wiki/Team_Foundation_Server)**).**

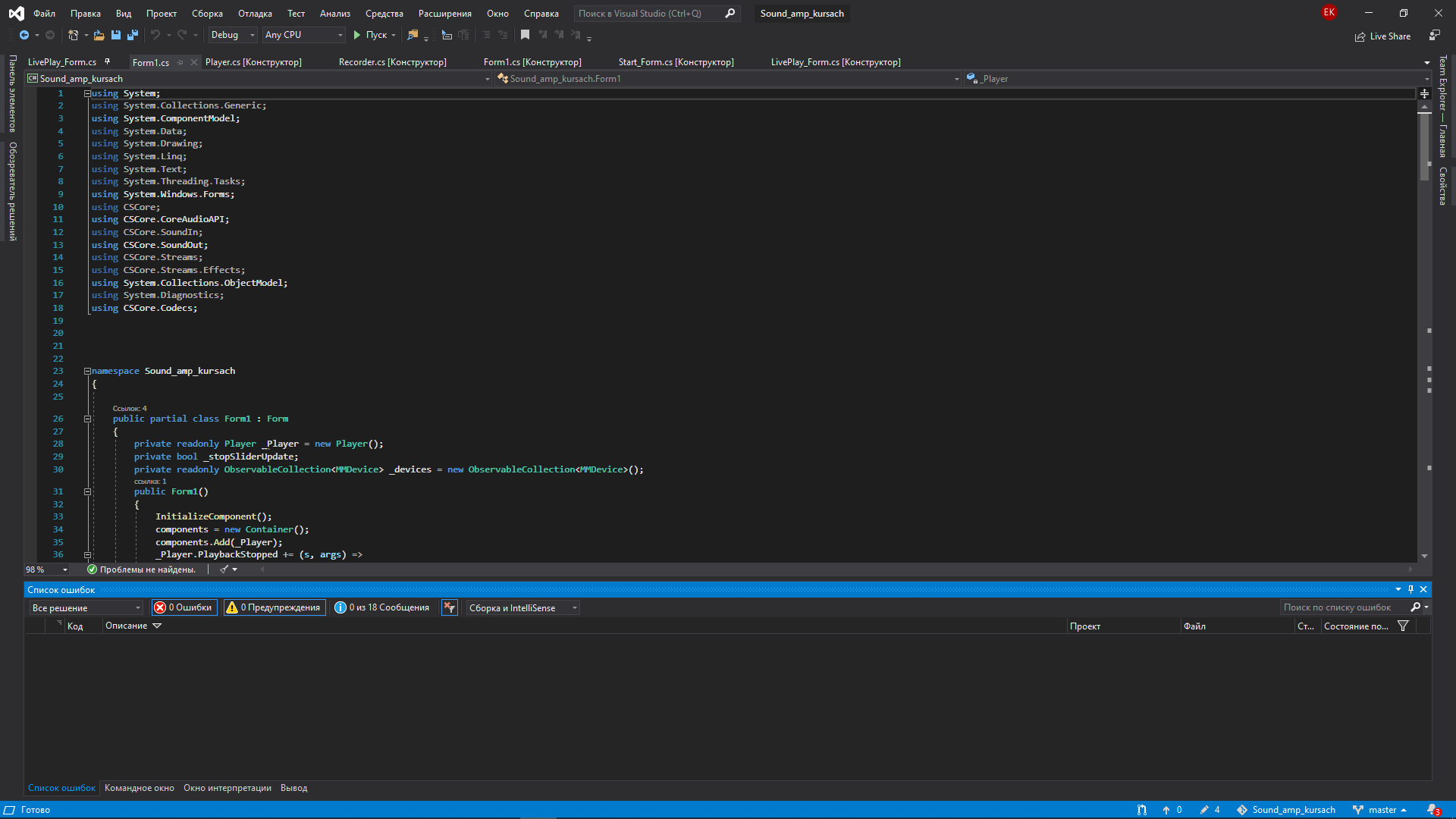


Рисунок 3.4. Графический интерфейс Visual Studio

Язык программирования C#

C# (произносится как "си шарп") — простой, современный объектно-ориентированный и типобезопасный язык программирования. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript.

C# является объектно-ориентированным языком, но поддерживает также и ***компонентно-ориентированное*** программирование. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности.

Важная особенность таких компонентов — это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также встроенные элементы документации.

C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# отлично подходит для создания и применения программных компонентов.

Данный язык был выбран в качестве основного, так как обладает нужными качествами для реализации

# 4 4 РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ Работы со звуком

4.1 Входные и выходные данные приложения

Входными данными в приложении для работы с гитарным звуком является то, что сыграл пользователь и то, как эти данные принял компьютер. Основные используемые типы данных приведены на рисунке 4.1

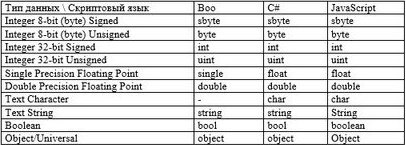


Рисунок 4.1. Основные типы данных

4.2 Проектирование структуры приложения

Структурно приложение состоит из следующих cs-файлов:

1. класс-персонажа – файлы RigidBodyController.cs;
2. Движение персонажа – файл AnimContr.cs

3) cs-файл для возможности управление персонажем от 3 лица –RotatePlayer.cs

4) cs-файл для графического отображения правой руки – IK.cs;

5) cs-файл класс врага – TDA\_Enemy.cs;

6) cs-файл магического уменя – Skill1.cs;

7) cs-файл удар –PlayerShot.cs

8) cs-файл—Смерть врага –DamegeEmery.cs

4.3 Описание алгоритмов работы скриптов системы или (Описание объектов и их взаимодействия для ООП)

RigidBodyController.cs

|  |  |
| --- | --- |
| public Image UIHP | Здоровье |
| public float HP | Количество здоровья |
| public GameObject Ragdoll | Смерть |
| public GameObject Skill1 | Магическое умение |
| void Start() | Метод вызывающийся при ходьбе |
| void Update() | Метод вызывающийся при использовании(Маг.Умений,ходьбы,здоровья) |
| void OnTriggerStay | Метод триггера |
| void OnTriggerEnter | Метод через который проходит урон при атаке на персонажа |

IK.CS

|  |  |
| --- | --- |
| public float lookIKweight | Отвечает за вес поворота частей тела в позицию пустого обьекта |
| public float bodyweight | Отвечает за вес поворота частей тела в позицию пустого обьекта |
| public float headweight | Отвечает за вес поворота частей тела в позицию пустого обьекта |
| public float eyesweight | Отвечает за вес поворота частей тела в позицию пустого обьекта |
| Public  float clampweight | Отвечает за вес поворота частей тела в позицию пустого обьекта |
| public Transform targetRotate | Переменная Transform для пустых обьектов(У них будем брать позицию) |
| public Transform targetHand | Переменная Transform для пустых обьектов(У них будем брать позицию) |
| void Start() | В старте считываем аниматор персонажа |
| void OnAnimatorIK() | В методе IK установим вес поворта частей тела |

# **RotatePlayer.cs**

|  |  |
| --- | --- |
| public Transform PosTarget | Публичная переменная Transform(С нее считываются координаты созданого обьекта) |
| public float turnSpeed | Скорость пороворота персонажа |
| void update() | В методе хранится вектор направление от пустого обьекта камеры до нашего персонажа |
| Ray Ray | Создаем райкаст и направляем с камеры,вперед |

# **AnimContr.cs**

|  |  |
| --- | --- |
| Animator animator | Переменная аниматор(В нее запишем считанный компонент аниматор персонажа) |
| float vertical | Переменная(в которой будет записываться значение нажатых клавиш) |
| float horizontal | Переменная(в которой будет записываться значение нажатых клавиш) |
| void Start() | В метод старт записывается компоненент в переменную |
| void Update() | Метод для записи значений нажатых клавиш |

# **TDA\_Enemy.cs**

|  |  |
| --- | --- |
| public GameObject player | Обьект до которого будет рассчитываться дистанция |
| public float dist | Дистанция |
| void Update() | Метод где прописываться дистанция до player,navmesh |
| NavMeshAgent nav | переменная |
| void Start() | Прописываем Меш гент |
| void Update() | Метод вызывающийся при использовании(Маг.Умений,ходьбы,здоровья) |
| public float HP | Здоровье врага |
| public GameObject Ragdoll | Добовляем процедурную анимацию |

# **Skill1.cs**

|  |  |
| --- | --- |
| public float timer | Время работы магического умения |
| void Update() | Метод где прописываться таймер(Удаление) |
| void Start() | Прописываем Меш гент |
| void Update() | Метод вызывающийся при использовании(Маг.Умений,ходьбы,здоровья) |

# **DamegeEmery.cs**

|  |  |
| --- | --- |
| public GameObject Emery | Обьект Emery |
| public GameObject Ragdoll | Обьект Ragdoll |

# **PlayerShor.cs**

|  |  |
| --- | --- |
| public GameObject Player | Обьект Player |
| public GameObject Ragdoll | Обьект Ragdoll |
| void Update() | Записываем нажатие мышки |
| Void start() |  |

# 5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

5.1 Аппаратные и программные средства создания и эксплуатации интернет-приложения

Тестирование программного обеспечения - процесс исследования, испытания программного продукта, имеющий две различные цели: – убедиться, что программа соответствует требованиям; – выявить ситуации, в которых поведение программы является неправильным, нежелательным.

Существует несколько уровней тестирования программного обеспечения: – модульное тестирование (юнит-тестирование) - тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция; – интеграционное тестирование - тестируются интерфейсы между компонентами, подсистемами или системами; – системное тестирование - тестируется интегрированная система на её соответствие требованиям;

– альфа-тестирование - имитация реальной работы с системой штатными разработчиками, либо реальная работа с системой потенциальными пользователями/заказчиком;

– Бета-тестирование - в некоторых случаях выполняется распространение предварительной версии для некоторой большей группы лиц с тем, чтобы убедиться, что продукт содержит достаточно мало ошибок.

Проект прошел все стадии тестирования на различных версиях операционных систем семейства Windows (XP, 7, 8) с различными конфигурациями оборудования (тактовая частота процессора, количество ядер, разрядность, объем оперативной и видео памяти и т.д.).

Системные требования к программе соответствуют заявленным в техническом задании.

Все найденные ошибки исправлены, большинство алгоритмов было оптимизировано, что позволило увеличить скорость работы приложения и его стабильность.

Аппаратные требования для работы приложения:

Требования к составу и параметрам технических средств Минимальные системные требования:

– ОС (операционная система): Windows Vista/7/8/10;

– Процессор: Intel Core 2 Duo @ 3.0 Ghz / AMD Athlon 64 X2 6000+;

– Оперативная память: 1 Gb;

– Жесткий диск: 10 Gb свободно;

–– Клавиатура, Мышь.

Рекомендуемые системные требования:

– ОС (операционная система): Windows XP/Vista/7/8;

– Процессор: Intel Core i5 @ 3.2 GHz / AMD Phenom II X4 @ 3.6 GHz;

– Оперативная память: 2 Gb;

– Жесткий диск: 10 Gb свободно;

–– Клавиатура, Мышь.

Программа должна функционировать под управлением ОС семейства Windows следующих версий: Windows Vista, 7, 8, 10. В приложении используются библиотеки платформы.NET Framework.

5.2 Руководство пользователя

Программа является продуктом сферы компьютерных развлечений. Игра относится к жанру ролевых компьютерных игр - одному из самых популярных жанров.

Игра не имеет возрастных ограничений и может служить отличным способом времяпрепровождения. .

Установка, запуск, минимальные требования и состав программы Для функционирования ПП требуется установить необходимое программное обеспечение. Все необходимое является бесплатным (кроме самой ОС) и легкодоступным. Для функционирования программы требуется установить следующие компоненты:

– MS dotNet Framework 3.5 или выше;

– драйвер для видео устройства.

В состав программы входят:

– Rise of Abbys Demo.exe, исполняемый файл программы;

– Rise of Abbys \_Data, каталог ресурсов программы.

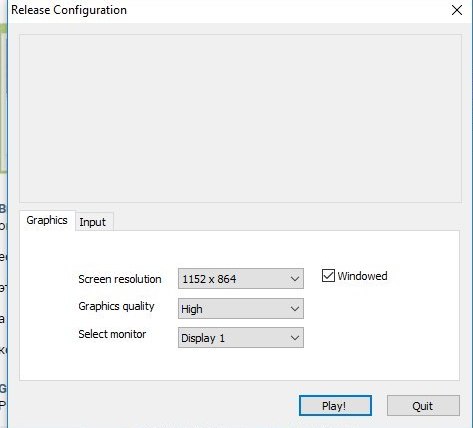
Для запуска программы « Rise of Abbys » требуется запустить исполняемый файл «Rise of Abbys Demo.exe».

На рисунке 5.1 изображено окно запуска приложения, в котором пользователь может настроить следующие параметры:

– разрешение экрана;

– оконный или полноэкранный режим;

– качество графики.



5.3 Описание контрольных примеров

# **После запуска приложения , пользователь попадает в главное меню.**

# **После нажатия пользователем кнопки произойдет загрузка игрового мира следующего характера:** ЗАКЛЮЧЕНИЕ

Результатом работы является программный продукт «Техническая демонстрационная системы боя в трехмерной компьютерной игры “Rise of abbys”».

Реализованы системы передвижение персонажей, системы боя и взаимодействия внутриигровых объектов.

Все подсистемы отлажены, оптимизированы, протестированы и интегрированы в единую программную систему.

Данный программный продукт является уникальным, т.к. все системы были реализованы без использования готовых решений.

.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Андреев Г.И. Практикум по оценке интеллектуальной собственности / Г.И. Андреев, В.В. Витчинка, С.А. Смирнов. - Москва: Финансы и статистика, 2002. - 176 с. .
2. Jonathan Tweet. Player's Handbook. Dungeons & Dragons Core Rulebook I / Jonathan Tweet, Monte Cook, Skip Williams. - Wizards of the Coast, Belgium, 2003. - 320 с. .
3. Jonathan Tweet. Dungeon Master's Guide. Dungeons & Dragons Core Rulebook II / Jonathan Tweet, Monte Cook, Skip Williams. - Wizards of the Coast, Belgium, 2003. - 320 с. .
4. Jonathan Tweet. Monster Manual. Dungeons & Dragons Core Rulebook III / Jonathan Tweet, Monte Cook, Skip Williams. - Wizards of the Coast, Belgium, 2003. - 320 с.
5. . Unity - Scripting [Электронный ресурс]. - Режим доступа: https://unity3d.com/learn/tutorials/modules/beginner/scripting. - Загл. с экрана. .
6. Объектно-ориентированное программирование [Электронный ресурс.]. - Режим доступа: http://msdn.microsoft.com/ru-ru/library/dd460654.aspx. - Загл. с экрана.

# ПРИЛОЖЕНИЕ А Экранные формы



Рисунок A.1 – Главный персонаж

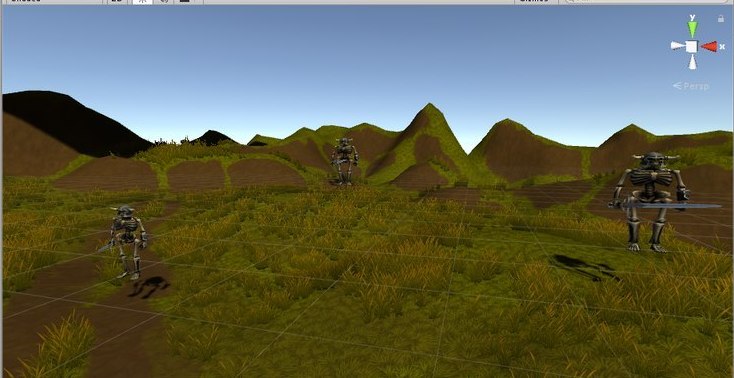


Рисунок A.2 – Враги

# 

# 

Рисунок A.3 – Демонстрация IK анимации

# 

Рисунок A.3 – Демонстрация боя и здоровья

# 

# Рисунок A.4 – Демонстрация бега

# 

# Рисунок A.5– Демонстрация магического уменя

# ПРИЛОЖЕНИЕ Б Фрагменты листинга

Листинг Б1 – Файл «RigidBodyController»

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class RigidBodyController : MonoBehaviour {  
    public Image UIHP;  
    public float HP = 1f;  
    float speed = 1.0f;  
    Rigidbody rigidBody;  
    float horizontal;  
    float vertical;  
    public GameObject Ragdoll;  
    public GameObject Skill1;  
    Animator animator;  
    public Transform rHand;  
    private HandItem item;  
  
    // Use this for initialization  
    void Start () {  
        rigidBody = GetComponent<Rigidbody> ();  
        animator = GetComponent<Animator> ();  
    }  
      
    // Update is called once per frame  
    void Update () {  
        if(Input.GetKeyDown(KeyCode.Alpha1)){  
            Instantiate (Skill1, transform.position, transform.rotation);  
        }  
        UIHP.fillAmount = HP;  
        horizontal = Input.GetAxis ("Horizontal");  
        vertical = Input.GetAxis ("Vertical");  
        rigidBody.AddForce (((transform.right \* horizontal) + (transform.forward \* vertical)) \* speed / Time.deltaTime);  
        if (HP < 0) {  
            gameObject.SetActive (false);  
            Instantiate (Ragdoll, transform.position, transform.rotation);  
        }  
    }  
    void OnTriggerStay (Collider other){  
        if (other.tag == "Dead") {  
            HP -= Time.deltaTime/10f;  
        }  
    }  
    void OnTriggerEnter(Collider other){  
        if (other.tag == "Sword2") {  
            HP = HP - 0.1f;  
        }  
    }  
  
    public void addHand(HandItem it)  
    {  
        if (item != null)   
        {  
            item.transform.SetParent(null);  
            item.gameObject.AddComponent<Rigidbody>();  
        }  
        it.transform.SetParent (rHand);  
        it.transform.localPosition = it.position;  
        it.transform.localRotation = Quaternion.Euler (it.rotation);  
        Destroy(it.GetComponent<Rigidbody>());  
        item = it;  
    }  
  
    public void addHealth(float count)  
    {  
        HP += count;  
        if (HP > 1f)   
        {  
            HP = 1f;  
        }  
    }  
  
  
}

Листинг Б2 – Файл «TDA\_enemy»

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.AI;  
  
public class TDA\_Enemy : MonoBehaviour {  
    public GameObject player;  
    public float dist;  
    NavMeshAgent nav;  
    public float Radius = 15;  
    public float HP = 1;  
    public GameObject Ragdoll;  
    // Use this for initialization  
    void Start () {  
        nav = GetComponent<NavMeshAgent> ();  
    }  
      
    // Update is called once per frame  
    void Update () {  
        if (HP < 0) {  
            Instantiate (Ragdoll, transform.position, transform.rotation);  
            Destroy (gameObject);  
        }  
        dist = Vector3.Distance (player.transform.position, transform.position);  
        if (dist > Radius) {  
            nav.enabled = false;  
            gameObject.GetComponent<Animator> ().SetTrigger ("Idle");  
  
        }  
        if (dist < Radius & dist>1.5f) {  
            nav.enabled = true;  
            nav.SetDestination (player.transform.position);  
            gameObject.GetComponent<Animator> ().SetTrigger ("Run");  
        }  
        if (dist < 1.5f) {  
            nav.enabled = false;  
            gameObject.GetComponent<Animator> ().SetTrigger ("Attack");  
        }  
    }  
    void OnTriggerEnter(Collider other){  
        if (other.tag == "Sword1p") {  
            HP = HP - 30;  
        }  
        if (other.tag == "Skill1") {  
            HP = HP - 50;  
        }  
}  
}

Листинг Б3 – Файл «IK»

using UnityEngine;  
using System.Collections;  
  
public class IK : MonoBehaviour {  
    public float lookIKweight;  
    public float bodyweight;  
    public float headweight;  
    public float eyesweight;  
    public float clampweight;  
    public Transform targetRotate;  
    public Transform targetHand;  
    Animator animator;  
    // Use this for initialization  
    void Start () {  
        animator = GetComponent<Animator>();  
    }  
      
    // Update is called once per frame  
    void OnAnimatorIK(){  
        animator.SetLookAtWeight (lookIKweight, bodyweight, headweight, eyesweight, clampweight);  
        animator.SetLookAtPosition (targetRotate.position);  
        if(Input.GetKey(KeyCode.Z)){  
            animator.SetIKPositionWeight (AvatarIKGoal.RightHand, 1.0f);  
            animator.SetIKPosition (AvatarIKGoal.RightHand, targetHand.position);  
        }  
    }  
}

Листинг Б4 – Файл «AnimContr»

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class AnimContr : MonoBehaviour {  
  
    Animator animator;  
  
    float vertical;  
    float horizontal;  
  
    // Use this for initialization  
    void Start () {  
        animator = GetComponent<Animator> ();  
    }  
      
    // Update is called once per frame  
    void Update () {  
        vertical = Input.GetAxis("Vertical");  
        horizontal = Input.GetAxis("Horizontal");  
  
        if (vertical == 0) {  
            animator.SetBool ("Run", false);  
            animator.SetBool ("RunBack", false);  
        }  
        if (horizontal == 0) {  
            animator.SetBool ("StrafeL", false);  
            animator.SetBool ("StrafeR", false);  
        }  
  
        if (vertical >= 0.1f) {  
            animator.SetBool ("Run", true);  
        }  
  
        if (vertical <= -0.1f) {  
            animator.SetBool ("RunBack", true);  
        }  
        if (horizontal <= -0.1f) {  
            animator.SetBool ("StrafeL", true);  
        }  
        if (horizontal >= 0.1f) {  
            animator.SetBool ("StrafeR", true);  
        }  
  
    }  
}

Листинг Б5 – Файл «RotatePlayer»

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class RotatePlayer : MonoBehaviour {  
  
    public Transform PosTarget;  
    public float turnSpeed;  
  
    // Update is called once per frame  
    void Update () {  
          
        Vector3 dir = PosTarget.position - transform.position;  
        dir.y = 0;  
        transform.rotation = Quaternion.Slerp (transform.rotation, Quaternion.LookRotation (dir), turnSpeed \* Time.deltaTime);  
        Ray ray = new Ray (Camera.main.transform.position, Camera.main.transform.forward);  
        PosTarget.position = ray.GetPoint (15);  
    }  
}

Листинг Б6 – Файл «Skill1»

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class RotatePlayer : MonoBehaviour {  
  
    public Transform PosTarget;  
    public float turnSpeed;  
  
    // Update is called once per frame  
    void Update () {  
          
        Vector3 dir = PosTarget.position - transform.position;  
        dir.y = 0;  
        transform.rotation = Quaternion.Slerp (transform.rotation, Quaternion.LookRotation (dir), turnSpeed \* Time.deltaTime);  
        Ray ray = new Ray (Camera.main.transform.position, Camera.main.transform.forward);  
        PosTarget.position = ray.GetPoint (15);  
    }  
}

Листинг Б7 – Файл «DamegeEmery»

using UnityEngine;  
using System.Collections;  
  
public class DamegeEmery : MonoBehaviour {  
    public GameObject Emery;  
    public GameObject Ragdoll;  
      
    void OnTriggerEnter(Collider other){  
        if(other.tag=="ShotShot"){  
            Emery.SetActive (false);  
            Ragdoll.SetActive (true);  
        Instantiate(Ragdoll, transform.position, transform.rotation);  
        }  
    }  
      
  
}

Листинг Б8 – Файл «PlayerShot »

using UnityEngine;  
using System.Collections;  
  
public class PlayerShot : MonoBehaviour {  
    public GameObject Player;  
    public GameObject Shot;  
      
    void Start(){  
        Shot.SetActive (false);  
    }  
      
    void Update(){  
        if(Input.GetMouseButtonDown(0)){  
            Player.GetComponent<Animator>().SetTrigger ("Shot");  
            Shot.SetActive (true);  
        }  
        if(Input.GetMouseButtonUp(0)){  
            Player.GetComponent<Animator>().SetTrigger ("Idle");  
            Shot.SetActive (false);  
        }  
    }  
}