МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

# ЦИФРОВАЯ МУЗЫКАЛЬНАЯ СТУДИЯ С УКЛОНОМ НА БЫСТРУЮ РАБОТУ ПОЛЬЗОВАТЕЛЯ

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-б-о-191(1)
Фамилия Имя Отчество

направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель _____
(оценка)
старший преподаватель кафедры
компьютерной инженерии и моделирования _____ Чабанов В.В.
(подпись, дата)

Симферополь, 2020

# РЕФЕРАТ

Цифровая музыкальная студия с уклоном на быструю работу пользователя. – Симферополь: ФТИ КФУ им. В. И. Вернадского, 2020. – 26 с., 16 ил., 9 ист.

Объект исследования данной работы – цифровая музыкальная студия, нацеленная на ускорение работы пользователя над музыкой.

Предмет исследования – позволение пользователю сфокусироваться на своих музыкальных идеях путём подготовки среды к продуктивной работе со старта.

Методы исследования – тестирование пользователями, попытка создать базовые идеи.

Музыкальная среда «Kiwi» была создана, данная комбинация является достаточно новой для рынка.

Данная среда действительно может помочь музыканту быстро создать идею и поделиться ей с другими.

Kiwi может использоваться начинающими музыкантами, которым важно быстро протестировать новые музыкальные идеи или записать небольшую мелодию, пока она не ушла из головы. Рекомендуется использовать как первую ступень в создании музыки, в случае если проект оказывается удачным, переносить идею в более «тяжелую» музыкальную среду.

МУЗЫКАЛЬНАЯ СРЕДА, СКОРОСТЬ, УДОБСТВО, C++, SFML, СОКЕТЫ, ЗВУК

# ОГЛАВЛЕНИЕ

# СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

| | |
|---|---|
| DAW | Digital Audio Workstation |
| ЦМС | Цифровая Музыкальная Студия |
| ADSR | Attack Delay Sustain Release |

# ВВЕДЕНИЕ

На текущий день мы можем наблюдать образ жизни, который в разы «быстрее», чем буквально 50-100 лет назад. Такое явление неизбежно вызвало ускорение всех привычных нам занятий – от повседневных развлечений до обучения новым вещам. Благодаря наличию огромного количества онлайн курсов на совершенно различные тематики, сейчас, как никогда до этого, стало возможным становиться мастером какой-либо сферы, обучаясь полностью самостоятельно. Что касается создания музыки, появление профессиональных DAW – цифровых сред для работы со звуком – значительно понизило порог вхождения новых музыкантов и композиторов на рынок: достаточно лишь иметь какой-либо персональный компьютер или даже смартфон с установленным на нём специализированным программным обеспечением. Наличие бесплатных ЦМС позволяет заниматься композиторством людям без нужды вкладывать деньги в настоящие инструменты и дорогое звукозаписывающее оборудование.

Процесс обучения композиторству так же значительно сократился во времени – получить знания о базовой музыкальной теории можно буквально за 30-60 минут [4][5] используя различные онлайн источники (к примеру, YouTube). Это, в свою очередь, неизбежно приводит к тому, что сам творческий процесс сокращается во времени: в то время, как полноценное создание песни всё ещё занимает большое количество времени, процесс генерации идей[1] стал быстрее. Цифровое пространство музыкальной среды позволяет быстро добавлять и удалять элементы, перемещать их местами, копировать их из других участков песни или слегка менять без нужды делать полную перезапись участка, как бы это пришлось делать на магнитной ленте. Именно поэтому решения, направленные на продуктивность, являются актуальными.

[1] Музыкальная идея – короткий (по сравнению со всем музыкальным произведением) участок, содержащий в себе какую-то обособленную мелодию, ритмическую фигуру или прочее.

# ГЛАВА 1
# ПОСТАНОВКА ЗАДАЧИ

## 1.1 Цель проекта

По итогу разработки планируется создать среду, которая поможет музыканту быстро создавать и развивать свою идею, а также при желании сразу делится ей с другими при помощи архитектуры сервер-клиент.

## 1.2 Существующие аналоги

Google Experiments (https://musiclab.chromeexperiments.com/Song-Maker/) – небольшая и легкая онлайн среда, в которой можно поэкспериментировать и создать музыку без наличия знаний о музыкальной теории.

Bandlab (https://www.bandlab.com/mix-editor) – полноценный онлайн редактор музыкальных треков, позволяющий работать над треком совместно.

Soundtrap (https://www.soundtrap.com/) – позволяет совместно с другими людьми производить коллаборацию через отправку друг другу файлов, текста, нот и прочего через специальное общее хранилище.

## 1.3 Основные отличия от аналогов

Такие ЦМС как Bandlab и Soundtrap являются достаточно «тяжёлыми» – в плане скорости первичной настройки проекта. Bandlab предоставляет возможность работать из браузера, однако это может быть не самым лучшим решением, если устройство является недостаточно мощным. Ближайшим решением к моему является Эксперименты от Google – «легкая» онлайн среда, где можно попробовать сделать что-то музыкальное. В отличие от неё, моя версия добавляет названия нот и позволяет применять знания музыкальной теории. Моя версия также добавляет базовую возможность к коллаборации – пользователи могут отправлять файлы на сервер и делиться ими друг с другом.

### 1.4 Техническое задание

Наименование продукта – «Kiwi».

Краткая характеристика: Программа предназначена для композиторов с начальным знанием музыкальной теории. «Kiwi» должна помогать в создании простых музыкальных идей и «лупов» (loops).

Требования к функциональным характеристикам: пользователю предоставляется временная линия (или рабочая область), по умолчанию состоящая из восьми пустых музыкальных тактов. Каждый бит (удар, счёт) отделяется тонкой вертикальной линией. Каждый четвертый бит выделяется толстой вертикальной линией, помечая, что это конец такта. Итого на экране приложения должно быть видно 5-5,5 тактов.

Слева от временной линии расположены подписи, обозначающие ноты – (C, C#, D … ) с числом, обозначающим их октаву. Должен быть диапазон от 0-й до 9-й октавы. На экране за раз должно быть видно 2 или немного больше октавы. Все белые клавиши должны представлять из себя белый прямоугольник с чёрной подписью, а всё чёрные – серо-чёрный с белой подписью. От каждой ноты должны тянуться горизонтальные тонкие чёрные полосы в правую часть экрана, таким образом, в соединении с вертикальными, образуя сетку, на которую можно будет расставлять ноты. Нажатие на подпись должно проиграть превью соответствующей ноты.

Справа и снизу экрана должны находиться «скроллы» – полосы прокрутки. Они должны отображать корректное положение видимой рабочей области. При помощи колёсика мыши рабочую область можно прокручивать вверх и вниз, если при этом зажать Ctrl, то будет происходить прокрутка влево и вправо.

При вертикальном скролле меняются какие «линии» – соответствующие подписи и их горизонтальные разделители – видны на экране. Одна позиция колёсика мыши должна соответствовать одной прокрученной ноте вверх или вниз, зависимо от направления движения колёсика. При достижении нижней или верхней границ скролл должен прекращаться в соответствующем направлении.

При горизонтальном скролле меняются какие вертикальные разделители видны на экране. Одна позиция колёсика мыши должна соответствовать одной прокрученной полоске влево или вправо, зависимо от направления движения колёсика. Визуально это выглядит, как перемещение толстых вертикальных линий по позициям на сетке. Движение колёсиком вниз должно расцениваться как движение влево. При достижении левой или правой границ скролл должен прекращаться в соответствующем направлении.

При нажатии на любую пустую клетку сетки на этом месте должна появиться нота. Визуально, она будет выглядеть как закрашенный зелёным цветом прямоугольник с подписью соответствующей нотой. При этом должен проиграть звук как при нажатии на саму подпись в левой части экрана. Если нажать на уже поставленную ноту, то она уберётся (без проигрывания звука превью).

Нажатие Ctrl + R должно полностью очищать сетку от нот.

В программе должно присутствовать визуальное отображение «маркера проигрывания» – вертикальная белая черта, которая располагается по центру счёта, воспроизведение которого сейчас происходит/будет происходить по нажатию на пробел. Нажатие на пробел заставляет маркер перемещаться слева направо, параллельно с этим проигрывая все установленные ноты на этом счёте. Скорость перемещения должно диктоваться установленным темпом. Воспроизведение можно в любой момент приостановить нажатием на пробел, после чего маркер остановится в том месте, где он был. Нажатие на пробел продолжит воспроизведение с этого же места. Нажатие на Ctrl + Space, независимо от того, в каком состоянии сейчас маркер воспроизведения, поставит паузу и переместит маркер на первую позицию.

Нажатие на Ctrl + S должно создать два файла: save*DATETIME*.kiwi и last_save.kiwi. *DATETIME* здесь – это UNIX время (пример: «1589464004»). Внутри этих файлов нужно сохранять информацию о том, как расставлены ноты.

Нажатие Ctrl + O должно автоматически подгрузить last_save.kiwi.

Приложение должно уметь принимать аргументы из командной строки при запуске. Список аргументов:

- **-f** *filename* – Открывает kiwi файл *filename* и загружает его.
- **-i** *instrument* – Подгружает новый звук инструмента из .wav файла на замену дефолтного. Если будет найден файл **instrument*.default*, то информацию о том, какая нота проигрывается в этом файле инструмента будет взята оттуда (аналогично аргументу -n), иначе предполагается, что C4.
- **-n** *note* – Меняет дефолтную ноту выбранного инструмента на *note*. Формат ноты – «C4», «A#6» и т.д.
- **-t** *tempo* – Меняет темп на *tempo* BPM.

Помимо этого, программа должна поддерживать два других режима работы, которые так же запускаются при помощи аргументов. При нахождении одного из них остальные аргументы игнорируются, а сама программа запускается в режиме без визуальной части (основная программа никогда не запускается).

**«-u»** или **«--uploadingMode» –** режим выгрузки. Клиент подключается по вшитому в программу IP и порту к KiwiServer. Если подключение не удаётся, его нужно повторять каждую секунду, пока подключение не удастся или программа не будет терминирована. При успешном подключении пользователю будет предоставлен выбор: выгрузить инструмент или сохранение. При выборе одного из пунктов пользователю печатаются все локальные .wav или .kiwi файлы соответственно, а затем предлагают вписать путь файла. Файлы, показанные на экране, не должны быть единственными доступными для выгрузки, программа должна принимать любой абсолютный путь в том числе. По нажатию на Enter, если такой файл будет существовать, этот файл будет передан на сервер (где сохранится в соответствующей папке), а клиенту вернётся уникальный ID, который будет принадлежать этому файлу. Его можно будет использовать при загрузке. Режим выгрузки «сбросится» – снова запросит пользователя выбрать, что он хочет выгружать.

**«-d»** или **«--downloadingMode» –** режим загрузки. Клиент подключается по вшитому в программу IP и порту к KiwiServer. Если подключение не удаётся, его нужно повторять каждую секунду, пока подключение не удастся или программа не будет терминирована. При успешном подключении пользователю будет предоставлен выбор: загрузить инструмент или сохранение. При выборе одного из пунктов, пользователю предлагается ввести ID. Если ID не будет найден, сервер вернёт false и пользователю будет предложено ввести ID ещё раз. В ином случае, файл будет успешно загружен (положен в папку Downloaded) и будет иметь название *ID*.

# ГЛАВА 2
# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

## 2.1 Анализ инструментальных средств

Самым большим вложением является SFML – Simple and Fast Multimedia Library. SFML предоставляет удобный способ создавать и контролировать окна, предлагает упрощенную обёртку над OpenGL для отрисовки своей графики, а так же имеет встроенные модули для того, чтобы воспроизводить звуки и производить интернет подключения к TCP серверам и UDP машинам (также позволяет создавать такие сервера/прослушивающие машины). Всё это было крайне важно при создании музыкальной студии, где нужно графически принимать ввод от пользователя, затем создавать из этого звук и производить соединение с сервером. SFML так же позволяет подключать ровно столько, сколько необходимо. К примеру, если необходимо использовать только сетевую часть кода, не проблема, в финальный .exe файл будет вшит только тот модуль, который отвечает за сетевой код. Из аналогов был также рассмотрен фреймворк Qt, так как он предоставлял возможность создавать привычный интерфейс, используя Windows Forms. Помимо прочего, в Qt есть модули работы со звуком и сетью. В конечном итоге я избрал именно SFML, так как имел куда больший опыт в создании интерфейсов на такой библиотеке, как p5.js, которая позволяла рисовать примитивными фигурами, текстом, изображениями и прочим прямо в браузере на элементе canvas. SFML предоставляет крайне схожий образ работы.

У этого подхода есть и минусы, например, невозможно так же просто и элегантно создать, скажем, текстовое поле или кнопку – обработчик этого придётся писать самостоятельно или обращаться к использованию сторонних библиотек. Такая библиотека существует [6], однако для облегчения задачи отладки и следования срокам решено было использовать простую консоль для того, чтобы получать ввод от пользователя в случае необходимости. Остальное было вынесено на сочетания клавиш и сам графический интерфейс.

Что касается сетевого кода, SFML предоставляет свой вспомогательный класс для отправки пакетов, который невероятно облегчает написание логики сетевого кода. Как и в обычных win сокетах, сокеты SFML поддерживают отправку буфера – массива char-ов – который определяется размером. Это классический подход, но он обладает несколькими минусами. Прежде всего, протокол Tcp делит пакеты на кусочки, а это означает, что, если кто-то из них задержался, на принимающей стороне может сработать timeout и приём данных оборвётся куда раньше, чем эти данные могут быть отправлены. Это означает, что конец всех данных необходимо вручную помечать как конец, а на принимающей стороне плодить кучу циклов, которые будут считывать данные в нужный момент до того, пока не будет получена зарезервированная пользователем последовательность байт, означающая конец передачи. Второй проблемой является отсутствие автоматической конвертации данных – между сервером и клиентом бегают байты, которые по умолчанию интерпретируются как обычная строка. Это означает, что нужно самостоятельно делать конвертацию тем или иным способом – переводить в строку на выходе и обратно в данные на входе или отправлять сырые данные. В случае с переводом в строку всё еще остаётся огромная проблема выделения полезных данных из всей строки – действительно, как за один раз отправить несколько данных, как не в строке через разделитель? А это означает, что на принимающей стороне придётся плодить код, который только и занимается тем, что ищет следующую подстроку и пытается её сконвертировать в нужные данные. В случае с отправкой «сырых» байт (например, отправлять вместо массива char-ов по указателю структуру, int, прочее) очень легко потерять кроссплатформенность: а что, если на принимающей стороне кодировка идёт по Little Endian вместо Big Endian, а не наоборот?

Все эти проблемы решаются при помощи класса Packet, реализованного в SFML. Прежде всего, созданный Packet решает проблему с разделением данных друг от труда, так как работа с ним совершается точно так же, как и с любым другим потоком – будь то std::stringstream, std::cout/std::cin, std::ofstream/std::ifstream и другие. Данные в пакет можно закидывать при помощи

перегруженного оператора “<<” и выгружать при помощи “>>”. Уже этот факт заставляет меня переходить на SFML, даже если с его помощью будет реализован только сетевой код. Данные корректно достаются и автоматически конвертируются в нужные типы, кроме этого, была решена проблема, которая мешала кроссплатформенности – автоматически учитывается, как именно интерпретируются данные в конкретной архитектуре. К тому же, сокеты обрели перегрузку встроенного метода send и receive, который теперь может работать с Packet, а не только с массивом char-ов. Принимающая сторона будет автоматически знать, когда весь пакет был отправлен.

Нельзя не упомянуть и звуковую часть моего проекта, за которую тоже отвечает SFML. SFML поддерживает возможность загружать звуки в оперативную память, чтобы они воспроизводились с минимальным откликом. Не стоит и говорить, что отзывчивость для любой музыкальной среды является крайне важной составляющей. Помимо всего прочего, SFML поддерживает изменение скорости воспроизведения. Это позволило существенно упростить как распространение инструментов между музыкантами (через клиент-сервер), так и облегчило добавление новых собственных инструментов пользователю. Теперь инструмент – это единственный .wav файл, к которому может быть прикреплён текстовый файл с совпадающим названием, говорящий о том, какая нота проигрывается в этом wav файле. Именно возможность проигрывать звук быстрее или медленнее позволяет мне точно высчитывать, во сколько раз проигрываемая нота выше или ниже оригинальной (при сравнивании их частот), и генерировать необходимый звук. Подробнее об этом алгоритме в пункте 2.2.

В остальном, моя программа использовала решения, доступные внутри языка C++. Например, я использовал библиотеку thread, чтобы позволить себе переместить цикл отрисовки на другой поток. Таким образом, даже если основной код программы будет по той или иной причине тормозить (например, при подгрузке файлов или обработке системных событий), то визуально программа всё ещё будет показывать своё корректное состояние. И наоборот, отрисовка не будет мешать окну отвечать на системные вызовы и совершать прочие действия.

Кроме того, управление звуком происходит именно на потоках. Именно это решение позволяет мне получить полифонию – возможность проигрывать несколько нот одновременно. Каждый звук запускается на отдельном потоке. В этом же потоке прослеживается, чтобы все звуки (по возможности) играли определённое количество времени. Повышая и понижая ноты, мы, как уже было указано выше, изменяем длительность воспроизведения. При понижении длительность файла увеличивается, при понижении, наоборот, уменьшается. Так как ноты расположены на сетке, недопустимо, чтобы какие-то из них были в несколько раз длиннее других. Именно этим потоки и занимаются: каждая нота запускается в отдельном потоке, который ждёт определённое время, а затем плавно отключает звук – сначала линейно снижает громкость до нуля, а затем и вовсе отключает звук. Благодаря умно реализованной логике потоков в C++, моя программа может без проблем запустить десятки таких звуковых потоков за раз, без заметного расхождения во времени начала звука.

### 2.2 Описание алгоритмов

Во время написания музыкальной среды «Kiwi» возникало множество мелких задач, которые предстояло решить. К примеру, информацию о нотах удобно хранить, как одномерный массив, где ноты идут от нуля от самого низа. Однако, такой подход будет неудобен пользователю. Куда проще сказать название ноты и её октаву. Данное решение было решено через создание специальных массивов, которые хранят названия всех 12 нот в октаве.

```
75  std::string notes_sharps[NoteNumber] = {
76      "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"
77  };
78  std::string notes_flats[NoteNumber] = {
79      "C", "Db", "D", "Eb", "E", "F", "Gb", "G", "Ab", "A", "Bb", "B"
80  };
```

Рис. 2.2.1. Массив, хранящий названия нот.

Далее названия этих нот используются при расшифровке введённых данных пользователя. Например, при анализе введенного аргумента «-n» или при

прочтении .default файла. На рисунке 2.2.2 приведена функция, которая занимается этим.

```
299 int noteToPitch(std::string notename) {
300     std::string noteLetter;
301     if (notename.size() == 2) noteLetter = notename.substr(0, 1);
302     else noteLetter = notename.substr(0, 2);
303 
304     int noteIndex = -1;
305     for (int i = 0; i < NoteNumber; i++) {
306         if (notes_sharps[i] == noteLetter) {
307             noteIndex = i;
308             break;
309         }
310     }
311 
312     if (noteIndex == -1) return -1;
313     char noteNumber = notename[notename.size() - 1];
314     if (noteNumber < '0' && noteNumber > '9') return -1;
315     std::cout << noteIndex << " " << noteNumber << " " << (noteNumber - '0') << '\n';
316 
317     return 12 * (noteNumber - '0') + noteIndex;
318 }
```

Рис. 2.2.2. Функция, преобразующая название ноты в её индекс.

Данный алгоритм берёт строку с нотой и разделяет её на две части – название ноты и её октава. Затем название находится в массиве из нот и к этому прибавляется номер октавы, умноженный на 12. Таким образом, мы получаем индекс именно той ноты, которая необходима. Схожий алгоритм используется, когда обрабатывается нажатие на левую кнопку мыши, чтобы определить на то, какая нота нажимается.

Далее, из индекса ноты можно вычислить то, на сколько нужно ускорить или замедлить оригинальный звук, чтобы получить именно эту ноту. Это изображено на рисунке 2.2.3.

```
double speed = std::pow(twelvesRootOf2, noteAblsoluteIndex - defaultInstrumentPitch);
createSound(instrumentBuffer, speed);
```

Рис. 2.2.3. Алгоритм вычисления скорости воспроизведения ноты.

Здесь используется следующее соотношение [7][8]:

## Mathematical properties [edit]

In twelve-tone equal temperament, which divides the octave into 12 equal parts, the width of a semitone, i.e. the frequency ratio of the interval between two adjacent notes, is the twelfth root of two:

$$\sqrt[12]{2} = 2^{\frac{1}{12}} \approx 1.059463$$

Рис. 2.2.4. Математическое соотношение частот полутона.

```
47  const double twelvesRootOf2 = std::pow(2, 1.0 / 12);
```

Рис. 2.2.5. Корень двенадцатой степени из двух, вынесенный в константу.

Соотношение на рисунке 2.2.4 говорит нам, что в нашей музыкальной системе частоты двух нот, стоящих рядом на клавиатуре фортепиано (то есть образующих интервал в полутон), соотносятся, как корень двенадцатой степени из двух. Отсюда нетрудно понять, что если возвести корень двенадцатой степени из двух в степень, равной количеству полутонов от данной ноты, то мы получим соотношение этих нот [9].

На рисунке 2.2.3 видно, как при возведении корня двенадцатой степени из двух в степень, равной разности индекса выбранной ноты и индекса ноты файла, мы получаем нужную скорость воспроизведения. Она будет полностью совпадать с соотношением частот. Отсюда так же видно, что если выбранная нота находится ниже ноты файла, то возведение произойдёт в отрицательную степень и файл будет проигрываться медленнее.

```
 99  void createSound(sf::SoundBuffer& buffer, double pitch = 1) {
100      std::thread soundThread([buffer, pitch]() {
101          sf::Sound sound(buffer);
102          sound.setPitch(pitch);
103
104          sound.play();
105          std::this_thread::sleep_for(200ms);
106
107          if (sound.getStatus() == sf::Sound::Playing) {
108              for (int i = 100; i >= 0; i--) {
109                  sound.setVolume(i);
110                  std::this_thread::sleep_for(5ms);
111              }
112              sound.stop();
113          }
114      });
115
116      soundThread.detach();
117  }
```

Рис. 2.2.6. Алгоритм воспроизведения ноты.

На рисунке 2.2.6 показывается, как именно воспроизводятся звуки. Грубо говоря, эта функция очень ограничено реализовывает такое понятие, как ADSR.

ADSR означает «Attack Decay Sustain Release», или четыре параметра, которые контролируют, как громкость звука ведёт себя во времени. На рисунке 2.2.7 можно увидеть, что именно из себя представляет ADSR-огибающая.

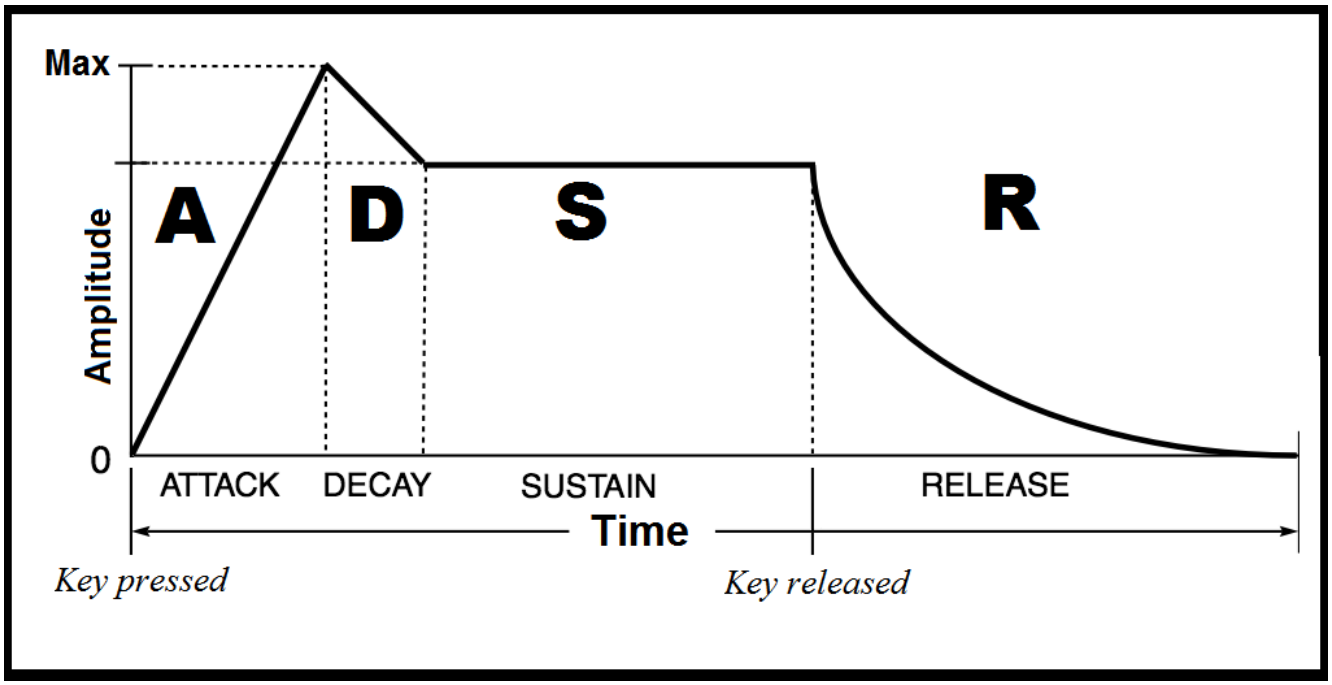


Рис. 2.2.7. ADSR-огибающая на графике зависимости амплитуды от времени.

В моём же случае я не трогал Attack и Decay – время нарастания и время плавного перехода звука к «сустейну». Оба они равны нулю для упрощения, это значит, что при старте звука он тут же начинает звучать с постоянной громкостью.

Время Sustain равно 200 миллисекунд, а время Release 500 миллисекунд. В моем случае громкость убывает в цикле for от 100 до 0 за равные промежутки времени. Мой график ADSR для звука выглядит подобным образом (Рисунок 2.2.8).

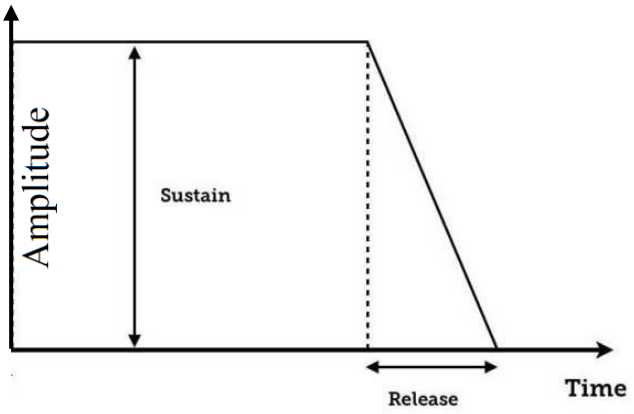


Рис. 2.2.8. Примерный график ADSR-огибающей.

Кроме того, в моей программе всегда резервируется отдельный поток для обработки, вероятно, самого важного алгоритма всей DAW – проигрывание с определённым темпом (Рисунок 2.2.9).

```
620     // Выделяем поток для обработки задержки между проигрыванием
621     std::thread playThread([&instrumentBuffer]() {
622         while (!programIsShuttingDown) {
623             while (!isPlaying);
624 
625             // Сыграть все звуки в данной колонке
626             for (int i = 0; i < NoteNumber * 10; i++) {
627                 if (isNoteSet[i][currentPlayPosition])
628                     createSound(instrumentBuffer, std::pow(twelvesRootOf2, i - defaultInstrumentPitch));
629             }
630 
631             std::this_thread::sleep_for(playDelay);
632 
633             if (!isPlaying) continue;
634 
635             currentPlayPosition++;
636             if (currentPlayPosition >= 32) currentPlayPosition = 0;
637             D(std::cout << "currentPlayPosition: " << currentPlayPosition << '\n');
638         }
639     });
640 
```

Рис. 2.2.9. Алгоритм воспроизведения музыки.

В нём, если выставлен флаг «isPlaying», проверяется по очереди все колонки (т.е. по всем нотам всех 10-ти доступных октав) с нотами – если нота на текущей позиции была выставлена, её нужно проиграть. После этого алгоритм ждёт количество миллисекунд, которое соответствует выставленному темпу (Рисунок 2.2.10) и перемещает указатель текущей позиции на один пункт дальше. Если мы дошли до конца, то указатель сбрасывается в начало, чтобы включить зацикливание.

```
119 std::chrono::milliseconds tempoToMs(int tempo) {
120     return std::chrono::milliseconds(60000 / (tempo));
121 }
```

Рис. 2.2.10. Алгоритм перевода темпа (в ударах в минуту) в миллисекунды.

### 2.3 Описание структур данных

Самой главной структурой данных у меня выступает двумерный массив атомических булевых переменных, хранящий в себе сетку (Рисунок 2.3.1).

```
82  // Поставлена ли нота в этом месте сетки?
83  // 10 октав
84  std::atomic<bool> isNoteSet[NoteNumber * 10][32];
```

Рис. 2.3.1. Главная структура данных

Булевы переменные являются атомическими, так как доступ к ним происходит из нескольких потоков одновременно. Это сделано, чтобы убрать неопределённое поведение при одновременном доступе. Итого данный массив имеет размеры 120 на 32, что означает, что в нём хранится 3840 элементов.

Во время сохранения в файл данный массив записывается бинарно, чтобы уменьшить финальный размер. Отсюда следует, что сохранённый файл будет весить 3840 байт и, как видно на рисунке 2.3.2, это действительно так.

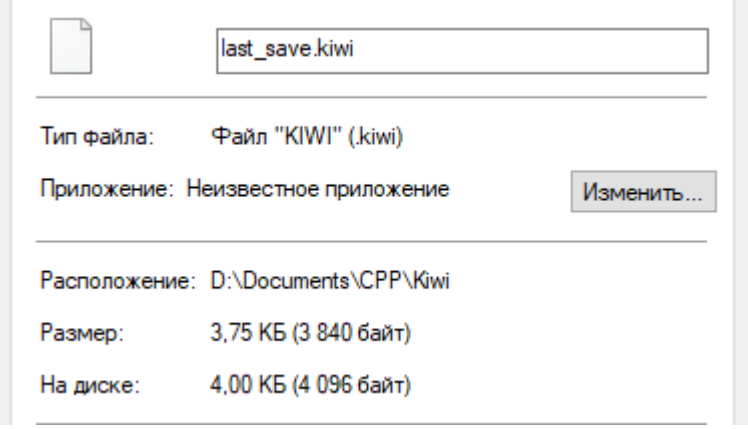


Рисунок 2.3.2. Сохранённый файл на диске.

Я решил хранить ноты в двумерном массиве, так как это облегчает саму работу с ним во время написания программы, а также 4 килобайта является совершенно мизерным размером как для оперативной памяти, так и для современных жестких дисков. Данный файл так же не составит труда передать по сети, так что нет смысла добавлять дополнительную упаковку данных.

### 2.4 Описание основных модулей

Программа делится на три основных модуля: основной поток, обрабатывающий события окна, поток, занимающийся отрисовкой графики и поток, занимающийся воспроизведением звука.

Основной поток располагается в функции main и занимается обработкой всех системных событий, чтобы окно не «зависло». Кроме того, это позволяет извлекать информацию о нажатиях на мышь, перемещения курсора, нажатия на клавиши клавиатуры и прочее.

Отрисовка графики располагается в функции draw и занимается циклической отрисовкой графики на экран 60 раз в секунду. Затирание окна и послойная отрисовка элементов.

Модуль воспроизведения звука был рассмотрен во втором разделе второй главы (Рисунок 2.2.9).

# ГЛАВА 3
# ТЕСТИРОВАНИЕ ПРОГРАММЫ

## 3.1 Тестирование исходного кода

Мой исходный код не был протестирован.

## 3.2 Тестирование интерфейса пользователя и юзабилити

Мой проект прошёл тестирование на юзабилити, все тесты были успешно пройдены. Ниже предоставлены скриншоты тест-кейсов и результаты.

**Тест 1/Test Name 1**

| | |
|---|---|
| **Среда тестирования /Environment** | Windows. |
| **Предварительные действия /Pre Requisites** | Нет. |
| **Комментарии /Comments** | Вертикальный скролл внутри среды выполняется при помощи колёсика, горизонтальный – при помощи колёсика с зажатым Ctrl. |

| Шаг /Step No. | Действие (операция) /Process (Actions) | Ожидаемый результат /Expected Results | Результат /Actual Results | Пройден /не пройден/ не доступен* | Комментарии /Notes |
|---|---|---|---|---|---|
| | | | | | |
| 1 | Перейти в папку Client и открыть Kiwi.exe | Успешно откроется окно среды вместе с консолью. | Соответствует ожиданию. | Пройден. | |
| 2 | Нажать на «превью» нот в левой части экрана – подписи «C», «D» и прочее | Проиграется звук фортепиано. Чем выше нота, тем, соответственно, выше звук | Соответствует ожиданию. | Пройден. | |
| 3 | Поставить несколько нот в произвольные места на сетке. Сохраниться при помощи Ctrl + S. | Ноты успешно встанут на место нажатия, проиграется звук превью. В консоли должно появится уведомление о сохранении | Соответствует ожиданию, но при первом сохранении, в консоль выводится сообщения о невозможности найти файл last_save.kiwi. | Пройден. | |
| 4 | Закрыть и открыть среду заново. Нажать Ctrl + O. | Все ноты, что были поставлены на предыдущем шаге, должны снова появиться на экране. | Соответствует ожиданию. | Пройден. | |

Рис. 3.2.1. Первый тест-кейс.

**Тест 2/Test Name 2**

| | |
|---|---|
| **Среда тестирования /Environment** | Windows. |
| **Предварительные действия /Pre Requisites** | Открыть powershell в папке Client |
| **Комментарии /Comments** | |

| Шаг /Step No. | Действие (операция) /Process (Actions) | Ожидаемый результат /Expected Results | Результат /Actual Results | Пройден /не пройден/ не доступен* | Комментарии /Notes |
|---|---|---|---|---|---|
| | | | | | |
| **1** | Запустить Kiwi.exe (в папке Client) из консоли со следующей командой: .\Kiwi.exe -f .\Shrek.kiwi -t 160 | Запустится окно приложения, в котором уже будут загружены ноты. | Соответствует ожиданию. | Пройден. | |
| **2** | Запустить воспроизведение, нажав на пробел. | По сетке «побежит» курсор – вертикальная белая палочка. Начнёт воспроизводиться “All Star”. | Соответствует ожиданию. | Пройден. | |
| **3** | Нажать на пробел, чтобы остановить воспроизведение, поперемещать курсор при помощи стрелочек (вверх и вниз – такт, влево и вправо – один счет). После перемещения начать воспроизведение нажатием на пробел. Повторить несколько раз | Курсор без проблем перемещается по экрану, не выходит за пределы, музыка воспроизводится с новых мест. | Соответствует ожиданию. | Пройден. | |
| **4** | Закрыть окно. Терминировать процесс из консоли при помощи Ctrl + C, если тот не остановился самостоятельно. Запустить приложение следующей командой: .\Kiwi.exe -f .\Shrek.kiwi -i .\banjo.wav -t 160 Запустить воспроизведение через пробел | Откроется тот же самый файл “All Star”, но на этот раз инструмент будет банджо вместо фортепиано | Соответствует ожиданию. | Пройден. | Немного не удобно то, что процесс приложения в консоли не остановился самостоятельно при закрытии окна приложения. |

Рис. 3.2.2. Второй тест-кейс.

**Тест 3/Test Name 3**

| | |
|---|---|
| **Среда тестирования /Environment** | Windows. |
| **Предварительные действия /Pre Requisites** | Открыть powershell в папке Client. Закрыть приложения, работающие на порту 8888, если такие имеются. |
| **Комментарии /Comments** | |

| Шаг /Step No. | Действие (операция) /Process (Actions) | Ожидаемый результат /Expected Results | Результат /Actual Results | Пройден /не пройден/ не доступен* | Комментарии /Notes |
|---|---|---|---|---|---|
| | | | | | |
| 1 | Запустить сервер (KiwiServer.exe в папке Server) | Появится консольное окно, которое сообщит, что сервер начал прослушку на 8888 порту | Соответствует ожиданию. | Пройден. | |
| 2 | Запустить Kiwi.exe (в папке Client) из консоли со следующей командой:<br>.\Kiwi.exe -u | В консоли клиента выведется сообщение «Welcome to uploading mode», после чего клиент подключится к серверу. Клиент предложит выбрать один из двух вариантов. | Соответствует ожиданию. | Пройден. | |
| 3 | Написать в консоли клиента 1 и нажать Enter | Клиент очистит экран и выведет все .wav файлы локальной директории. | Соответствует ожиданию. | Пройден. | |
| 4 | Написать в консоли<br>banjo.wav<br>И нажать Enter | На клиенте отобразится «Success», в папке Server/Sounds появится .wav файл, который можно будет прослушать и услышать оригинальный инструмент. | Соответствует ожиданию. | Пройден. | При записи в консоль сервера выводится крайне много "Writing". После успешной записи первого звука я попробовал записать еще один, и из за этого сломался клиент. |
| 5 | Написать в консоли клиента 2 и нажать Enter | Клиент очистит экран и выведет все .kiwi файлы локальной директории. | Соответствует ожиданию. | Пройден. | |
| 6 | Написать в консоли<br>Save1.kiwi<br>И нажать Enter<br>Скопировать полученный ID и прервать выполнение программы через Ctrl + C | На клиенте должно появиться «Upload complete, here is the ID of your file:». После этого на строчке должен быть ID. | Соответствует ожиданию. | Пройден. | ID: 26783 |

Рис. 3.2.3. Третий тест-кейс.

**Тест 4/Test Name 4**

| | |
|---|---|
| **Среда тестирования /Environment** | Windows. |
| **Предварительные действия /Pre Requisites** | Открыть powershell в папке Client. Выполнить тест 3. Если сервер открыт с прошлого раза, его можно оставить, иначе открыть Server аналогично первому шагу теста 3. |
| **Комментарии /Comments** | |

| Шаг /Step No. | Действие (операция) /Process (Actions) | Ожидаемый результат /Expected Results | Результат /Actual Results | Пройден /не пройден/ не доступен* | Комментарии /Notes |
|---|---|---|---|---|---|
| | | | | | |
| 1 | Запустить Kiwi.exe (в папке Client) из консоли со следующей командой:<br>.\Kiwi.exe -d | В консоли клиента выведется сообщение «Welcome to downloading mode», после чего клиент подключится к серверу. Клиент предложит выбрать один из двух вариантов. | Соответствует ожиданию. | Пройден. | |
| 2 | Написать в консоли клиента 2 и нажать Enter | Клиент попросит ввести ID проекта | Соответствует ожиданию. | Пройден. | |
| 3 | Вставить ID, который был получен с теста 3 | Будет написано «Downloading...», а затем опять предложено выбрать из двух вариантов. | Соответствует ожиданию. | Пройден. | |
| 4 | Терминировать процесс из консоли при помощи Ctrl + C. Запустить Kiwi.exe (в папке Client) из консоли со следующей командой:<br>.\Kiwi.exe -f .\Downloaded*ID*<br><br>Вставить вместо *ID* скопированный ID с теста 3 | Должен запуститься проект, который должен быть идентичен тому, который получается при запуске с командой .\Kiwi.exe -f .\Save1.kiwi | Соответствует ожиданию. | Пройден. | |

Рис. 3.2.4. Четвертый тест-кейс.

Данное тестирование помогло мне отловить несколько важных багов: программа не всегда завершалась и не всегда корректно передавала файлы.

# ГЛАВА 4
# ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

## 4.1 Перспективы технического развития

За время создания проекта, я смог реализовать большое количество фич, которые смогли превратить эту DAW в ту, которой действительно можно пользоваться. К сожалению, несколько запланированных вещей не было реализовано из-за нехватки времени. К примеру, изначально должна была поддерживаться не только полифония одного инструмента, но и полифония из нескольких инструментов. Реализация этой возможности позволило бы создавать куда более полноценные треки, чем при использовании только одного инструмента.

Изначально планировалось, что размер сетки можно будет редактировать – добавлять или убирать такты. Эта функция была вырезана, чтобы облегчить функционал сохранения и загрузки файлов, а также более легкой отладки остальных частей проекта. Добавление этой функции позволило бы куда более гибко преследовать оригинальную цель проекта – иметь среду, в которой можно быстро опробовать какую-то новую музыкальную идею. К сожалению, сейчас среда обязует иметь размер идеи именно в восемь тактов, иначе зациклить её нормально не получится.

Кроме того, выгрузка инструментов на данный момент была не доработана. Был написан только код клиентской части, так как на сервере я столкнулся с проблемой просматривания содержимого папки, которое я не знал, как реализовать. Данная функция позволила бы делиться с другими не только сохранениями, но и инструментами, для которых они были написаны.

## 4.2 Перспективы монетизации

Данный проект можно развивать дальше. Например, перевести режим работы на единый центральный сервер, а затем позволить каждому, выкладывающему своё сохранение или файл инструмента, выставлять цену за

скачивание и использование. Таким образом музыканты смогут получать деньги за их созданные идеи или записанные звуки, а я смогу забирать процент-комиссию.

# ЗАКЛЮЧЕНИЕ

В результате создания данного проекта я получил много опыта, а самое главное, я могу сказать, что я смог выполнить свою цель. Моя работа принесла мне инструмент быстрого создания экспериментов с музыкой, которое потом, при желании, возможно развить во что-то полноценное в других DAW. Данный проект оказался очень полезным для меня. Можно подвести следующие выводы:

- Был получен опыт работы с SFML: Графика, звук, сокеты;
- Изучены математические выражения, стоящие за получением корректных частот нот;
- Получен опыт написания документации и отчётов по проекту;
- Были приобретены навыки в анализе и тестировании чужих проектов.

# ЛИТЕРАТУРА

1. ГОСТ 19.002-80 Схемы алгоритмов и программ. Правила выполнения [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

2. ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначение условные графические [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

3. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2017. – 31 с.

4. Learn music theory in half an hour / Andrew Huang [Электронный ресурс] – URL: https://youtu.be/rgaTLrZGlk0, 2019.

5. Understanding Music Theory in One Hour - Animated Music Lesson / Ross the Music and Guitar Teacher [Электронный ресурс] URL: https://youtu.be/kvGYl8SQBJ0, 2017.

6. Texus' Graphical User Interface / Bruno Van de Velde [Электронный ресурс] URL: https://tgui.eu, 2013-2020

7. 12-ти тоновая равномерно темперированная система / Wikipedia [Электронный ресурс] URL: https://en.wikipedia.org/wiki/12_equal_temperament#Mathematical_properties – Раздел «Mathematical properties».

8. Полутон / Wikipedia [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Полутон

9. 12-ти тоновая равномерно темперированная система / Wikipedia [Электронный ресурс] URL: https://en.wikipedia.org/wiki/12_equal_temperament#Calculating_absolute_frequencies – Раздел «Calculating absolute frequencies»