Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Вычислительная техника

УТВЕРЖДАЮ
Заведующий кафедрой

_____ _____
подпись инициалы, фамилия
«____» _____ 2021 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

Звуковой синтезатор с MIDI-клавиатурой

Руководитель _____ ст. преп. каф. ВТ И. В. Матковский
подпись, дата

Выпускник _____ Р. С. Янмурзин
подпись, дата

Нормоконтролер _____ И. В. Матковский
подпись, дата

Красноярск 2021

РЕФЕРАТ

Выпускная квалификационная работа по теме «Звуковой синтезатор с MIDI-клавиатурой» содержит 32 страницы текстового документа, 24 изображения, 6 формул, 5 использованных источников.

СИНТЕЗАТОР, ГЕНЕРАТОР, MIDI-КЛАВИАТУРА, C++, ADSR-ОГИБАЮЩАЯ, ЧАСТОТА, НОТА, ЗВУКОВАЯ ВОЛНА, ЗВУКОВАЯ ТАБЛИЦА.

Цель – создание музыкального приложения для генерации звукового сигнала.

Задачи:

– изучение предметной области;

– рассмотрение аналогов;

– выбор средств разработки;

– рассмотрение методов реализации;

– разработка приложения.

Были рассмотрены аналоги, а также выделены основные особенности современных синтезаторов. В результате выполнения работы был получен полифонический синтезатор звуковой волны. Высота звука определяется нотой, нажатой на MIDI-клавиатуре. Амплитуда звука каждой ноты изменяется во времени в зависимости от параметров ADSR-огибающей.

Данная работа предоставляет быстрый доступ к синтезу простых звуковых волн, не является ресурсоемким. Описание данной работы демонстрирует возможности работы со звуком, что может помочь разработчикам, желающим разрабатывать музыкальные приложения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Синтезатор – это электронный музыкальный инструмент, который генерирует звуковой сигнал. В настоящее время существует множество различных синтезаторов в виде аналоговых приборов, а также в виде программного обеспечения, что позволяет синтезировать звук сидя прямо за компьютером.

Весь звук внутри компьютера является цифровым сигналом – сигналом, который можно представить в виде последовательности цифровых значений. Благодаря этому звук можно создавать, либо видоизменять при помощи программного обеспечения. Процесс создания звука на программном обеспечении называется звуковым синтезом.

В современном мире синтез звука используется в самых различных отраслях. Почти любая музыка имеет синтезированные звуки, каждый фильм, видеоигра, реклама, даже техника – всё это имеет звук, благодаря чему различные бренды приобретают уникальность. Звуковой синтезатор – это неотъемлемая часть любого звукового дизайнера и музыканта.

Данная работа посвящена разработке звукового синтезатора. На основании этого были поставлены следующие задачи выпускной квалификационной работы:

– изучение предметной области;

– рассмотрение аналогов и выделение важных особенностей;

– выбор средств разработки;

– рассмотрение методов реализации;

– разработка приложения;

– подведение итогов.

1 Аналитическая часть

1.1 Описание предметной области

MIDI, расшифровывается как Musical Instrument Digital Interface, что в переводе на русский язык означает цифровой интерфейс музыкального инструмента – это технический стандарт, описывающий протокол связи, цифровой интерфейс и электрические разъемы, которые соединяют широкий спектр электронных музыкальных инструментов, компьютеров и связанных аудиоустройств для воспроизведения, редактирования и записи музыки [1].

MIDI-клавиатура – это музыкальная клавиатура, которая посылает MIDI-сигналы. Эти сигналы могут хранить различную информацию касательно проигрываемых нот, например, высота, громкость, длительность, характер и так далее.

VST (Virtual Studio Technology) – формат плагинов реального времени, которые подключаются к звуковым редакторам, секвенсорам, цифровым звуковым рабочим станциям. Формат был разработан совместно с Propellerhead и Steinberg, впоследствии Propellerhead отказался от дальнейших работ над VST, и дальнейшая разработка осуществлялась исключительно Steinberg.

Когда говорят о VST-плагинах, чаще всего имеют в виду программные инструменты, подгружаемые в программы для работы со звуком. Основное достоинство VST-плагинов – простота подключения и хранения, удобство в работе.

Секвенсор – аппаратное устройство или прикладная программа для записи, редактирования и воспроизведения «последовательности MIDI-данных» [2].

Звук внутри цифрового синтезатора представляет собой последовательность семплов – численных значений, представляющих амплитуду звуковой волны в определенный момент времени.

1.2 Аналоги

Первый коммерческий синтезатор появился в 1964 году. Он был разработан американским инженером Робертом Мугом и был назван в честь в честь автора – Moog synthesizer [3]. Синтезатор состоит из отдельных модулей, таких как генераторы, управляемые напряжением, усилители и фильтры, генераторы огибающей, генераторы шума, кольцевые модуляторы, триггеры и микшеры, которые создают и формируют звуки. В него можно играть с помощью контроллеров, включая музыкальные клавиатуры, джойстики, педали, или управлять им с помощью секвенсоров. Его осцилляторы могут генерировать волны разных тембров, которые можно модулировать и фильтровать для формирования звука. Это очень большой, громоздкий синтезатор, что является существенным недостатком.

Yamaha DX7 – цифровой синтезатор, выпущенный фирмой Yamaha в 1983 году [4]. Это профессиональный инструмент, который основан на FM-синтезе. Синтезатор имеет 6 генераторов, которые могут быть соединены между собой 32 способами, сам он довольно компактный и включает в себя игровую клавиатуру. Недостатком данного синтезатора можно назвать его сравнительно высокую цену. В настоящее время существует компьютерный аналог данного синтезатора – FM8.

Serum – синтезатор, разработанный компанией Xfer Records в виде vst-плагина для секвенсоров. Данный синтезатор имеет два осциллятора, которые имеют множество волновых таблиц, в том числе возможность создать и загрузить в него свои. Осцилляторы поддерживают множество модуляций, в том числе могу модулировать друг друга тремя видами синтеза. Он также содержит множество фильтров и эффектов, что позволяет создавать самые качественные звуки. Из недостатков можно отметить высокую цену, немалое требование к ресурсам, а также невозможность использовать его в качестве самостоятельного приложения.

Massive – vst-синтезатор, принадлежащий компании Native Instruments. Синтезатор имеет три осциллятора, содержащих большое количество разнообразных волновых таблиц, несколько модуляций, а также фильтров и эффектов. Отсутствие встроенной клавиатуры является недостатком таким же, как и цена, а также невозможность использовать его как самостоятельное приложение.

1.3 Сравнение аналогов

Для выполнения работы был проведен анализ и сравнение рассмотренных аналогов. Каждый синтезатор обладает своими преимуществами и недостатками. Отличия заключаются в различных модуляциях для видоизменения и модификации звуковых волн, в наличии фильтров для видоизменения спектра частот проигрываемого сигнала. Среди рассмотренных синтезаторов были выделены ключевые особенности каждого синтезатора:

– имеет как минимум один генератор, который может формировать звуковые волны простых форм;

– работает с MIDI-сигналом, то есть принимает на вход ноты, которые в дальнейшем преобразуются в звуковой сигнал;

– имеет ADSR-огибающую, которая определяет изменение сигнала в течении проигрываемого времени.

Решено, что данные особенности должен содержать проектируемый в данной работе синтезатор.

1.4 Представление звуковой информации в цифровом виде

Звук внутри компьютера представляет собой звуковую волну, которую описывает множество маленьких фрагментов, называемых семплами. Они содержат текущие значения амплитуды звуковой волны. То есть, если

представить звуковую волну в виде точек на плоскости, эти точки будут определять семплы.

Частота дискретизации – это частота, с которой происходит оцифровка сигнала. Именно она определяет количество семплов, обрабатываемых в секунду. Согласно теореме Котельникова, частота дискретизации должна быть вдвое выше самой верхней частоты, которую имеет сигнал.

В таком случае для человека, слух которого обычно распознает звук с частотой до 20 килогерц, достаточно, чтобы частота дискретизации была около 40 килогерц или выше. Это гарантирует, что каждый звук будет услышан без искажений. При этом там, где не требуется особо четкое распознавание высоких частот, например для низкочастотных звуков, радио, телефонного разговора – частота дискретизации может быть гораздо ниже.

Существуют разные форматы, хранящие звук внутри компьютера. Самые популярные из них:

– WAV, расшифровывается как Wave Audio File Format – стандартный формат для хранения звука на компьютере, также является основным внутри Windows. Этот формат не имеет никакого сжатия сигнала, что обеспечивает максимальное качество звука, но при этом имеет очень большой размер файла. Практически каждый звуковой редактор работает с этим форматом;

– MP3 является распространенным форматом кодирования звуковой информации. Этот формат использует сжатие с потерями оригинального качества, но при этом незаметными для слуха обычного пользователя, особенно на мультимедийных системах воспроизведения звука;

– MIDI формат обычно используется для обмена данными между электронными музыкальными инструментами. Этот формат не содержит звуковую волну, вместо этого он хранит данные о музыкальных партиях, их нотах и многой другой информацией, связанной с ними.

2 Подходы к реализации

2.1 Генерация формы волны

Для проектируемого синтезатора потребуется генератор, который может воспроизводить четыре простые формы волны. Для этого нужно представить волны при помощи математических формул. В дальнейшем это понадобится для того, чтобы написать функции, которые будут изображать формы волн в цифровом виде. На рисунке 1 изображены формы необходимых волн – синусоидная, треугольная, пилообразная и квадратная.

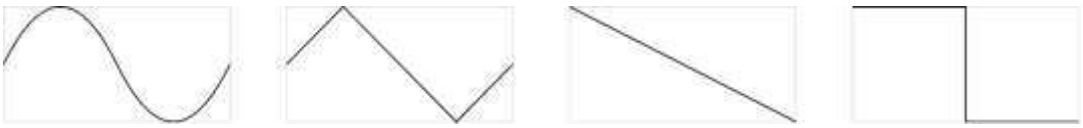

Рисунок 1 – Синусоидная, треугольная, пилообразная и квадратная формы волн

Данные волны были выбраны благодаря тому, что являются основными и довольно часто используются в синтезе звуков. Спектр частот данных волн (кроме синусоидной) имеет ровное распределение гармоник – призвуков, входящих в звук, частоты которых находятся в прямой математической зависимости друг от друга. Эти волны имеют следующее распределение гармоник:

– синусоида не имеет гармоник, ее составляет лишь одна частота, и именно из нее складываются все остальные звуки;

– треугольная волна представляет собой только нечетные гармоники, которые плавно спадают по мере увеличения частоты в соотношении 12 децибел на октаву;

– пилообразная волна имеет в себе и четные, и нечетные гармоники, амплитуда которых падает в соотношении 6 децибел на октаву;

– прямоугольная волна также как и треугольная имеет только нечетные гармоники, но амплитуда которых падает в соотношении 6 децибел на октаву.

Так как после каждого прохода волна будет повторяться, достаточно будет описать лишь один период. Представим, что длительность одного периода равна единице, при этом значение колеблется от -1 до +1. Тогда получим следующие формулы.

Формула для синусоидной формы волны:

$$x(t) = \sin(2\pi t), \tag{1}$$

где t – положение периода, принимает значения от 0 до 1.

Формула для треугольной формы волны:

$$x(t) = 2(|1 - 2t| - 0{,}5), \tag{2}$$

где t – то же, что и в формуле (1).

Формула для пилообразной формы волны:

$$x(t) = 1 - 2t, \tag{3}$$

где t – то же, что и в формуле (1).

Формулы для квадратной формы волны:

$$x(t) = -1, \tag{4}$$

где t – положение периода, при $0 \leq t < 0{,}5$;

$$x(t) = 1, \tag{5}$$

где t – положение периода, при $0{,}5 \leq t \leq 1$.

2.2 Преобразование ноты в частоту

Необходимо чтобы синтезатор умел воспроизводить звуки с разной частотой. В этом поможет реализация MIDI-технологии.

Внутри MIDI каждой ноте присвоен свой номер. Самая первая нота имеет номер 0 – это нота «до», которая располагается ниже субконтроктавы. Самая последняя – это нота соль шестой октавы, имеет номер 127.

Для преобразования MIDI-ноты в частоту используется формула:

$$f = 440 \cdot 2^{\left(\frac{n-69}{12}\right)}, \tag{6}$$

где f – частота проигрываемой ноты; 440 – частота камертона, которая является международным стандартом настройки звука и соответствует ноте ля первой октавы; n – номер проигрываемой ноты; 69 – номер ноты ля первой октавы внутри MIDI.

2.3 ADSR-огибающая

Амплитуда звука должна меняться при проигрывании. В качестве реализации этого будет использоваться ADSR-огибающая.

ADSR-огибающая – это функция, описывающая изменения какого-либо параметра во времени, которая используется в синтезаторах звука. Она представляет собой четыре параметра Attack, Decay, Sustain и Release, которые характеризуют поведение звука. Определяются они так:

– Attack – атака, длительность начального нарастания громкости сигнала. При нажатии клавиши на музыкальной клавиатуре сигнал должен плавно нарастать за время, определенное этим параметром;

– Decay – спад, длительность ослабления сигнала после начального нарастания. При удержании клавиши этот параметр будет снижать амплитуду сигнала после того, как он прошел время нарастания;

– Sustain – удержание, уровень постоянной силы сигнала. Этот параметр представляет собой амплитуду сигнала, которая будет постоянной до тех пор, пока не будет отпущена клавиша на клавиатуре;

– Release – затухание, длительность окончательного затухания сигнала. После отпускания клавиши амплитуда звука должна плавно опускаться до полной тишины.

Алгоритм работы огибающей таков: при удержании ноты, уровень сигнала начинает подниматься от уровня близкого к нулю до максимального пика, затем происходит спад уровня до уровня параметра затухания и не падает до тех пор, пока клавиша MIDI-клавиатуры не будет отпущена; после того как клавиша отпущена, происходит спад до уровня близкого к нулю, при чем спад происходит всегда, после отпускания клавиши, вне зависимости от того, на каком этапе находится звук.

2.4 Графический интерфейс

Графический интерфейс приложения позволит пользователю взаимодействовать с синтезатором. Интерфейс должен включать в себя ручки, которые дадут возможность выбирать форму волны и работать с ADSR-огибающей. На рисунке 2 изображен концепт графического интерфейса к синтезатору, где OSC – это ручка выбора формы волны, остальные ручки описывают ADSR.

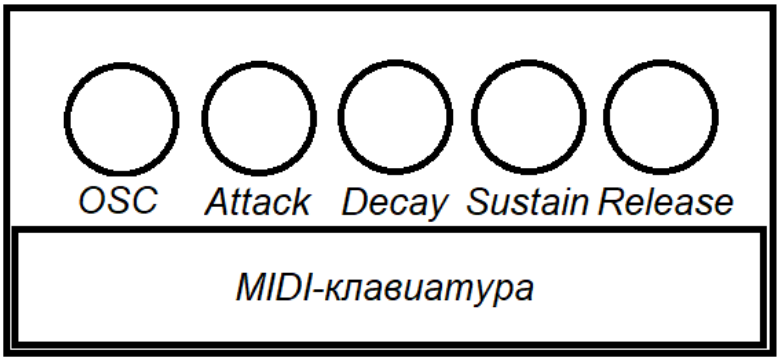

Рисунок 2 – Концепт графического интерфейса

2.5 Библиотека WDL-OL

Для выполнения данной работы была выбрана библиотека WDL-OL, которая включает в себя IPlug – это простой в использовании C++ фреймворк для разработки кроссплатформенных аудиоплагинов, который нацелен на несколько API плагинов с одним и тем же кодом. Первоначально разработанный компанией Schwa/Cockos, IPlug был усовершенствован различными разработчиками. IPlug зависит от WDL, и именно поэтому этот проект называется WDL-OL, хотя большинство отличий от WDL компании Cockos заключается в том, что он работает с IPlug.

Эта версия IPlug нацелена на API VST2, VST3, AudioUnit RTAS и AAX. Она также может создавать автономные аудио/миди приложения для Windows/OSX и приложения для IOS-устройств Apple [5].

2.6 Вывод по главе

В этой главе были определены главные свойства проектируемого синтезатора, а также средства разработки. Было выделено, что синтезатор должен:

– иметь генератор, умеющий создавать волны синусоидной, треугольной, пилообразной и квадратной форм;

– иметь ADSR-огибающую, позволяющую амплитуде звука изменяться во время проигрывания ноты;

– графический интерфейс, позволяющий пользователю взаимодействовать с музыкальной клавиатурой, а также параметрами ADSR-огибающей.

В качестве средств разработки был выбран язык C++ и библиотека WDL-OL.

3 Разработка синтезатора

3.1 Диаграмма классов

Для реализации работы потребуются три класса: класс приложения, в котором будет проводиться инициализация, класс ADSR-огибающей и класс генератора для преобразования волновой таблицы. Диаграмма классов представлена на рисунке 3.

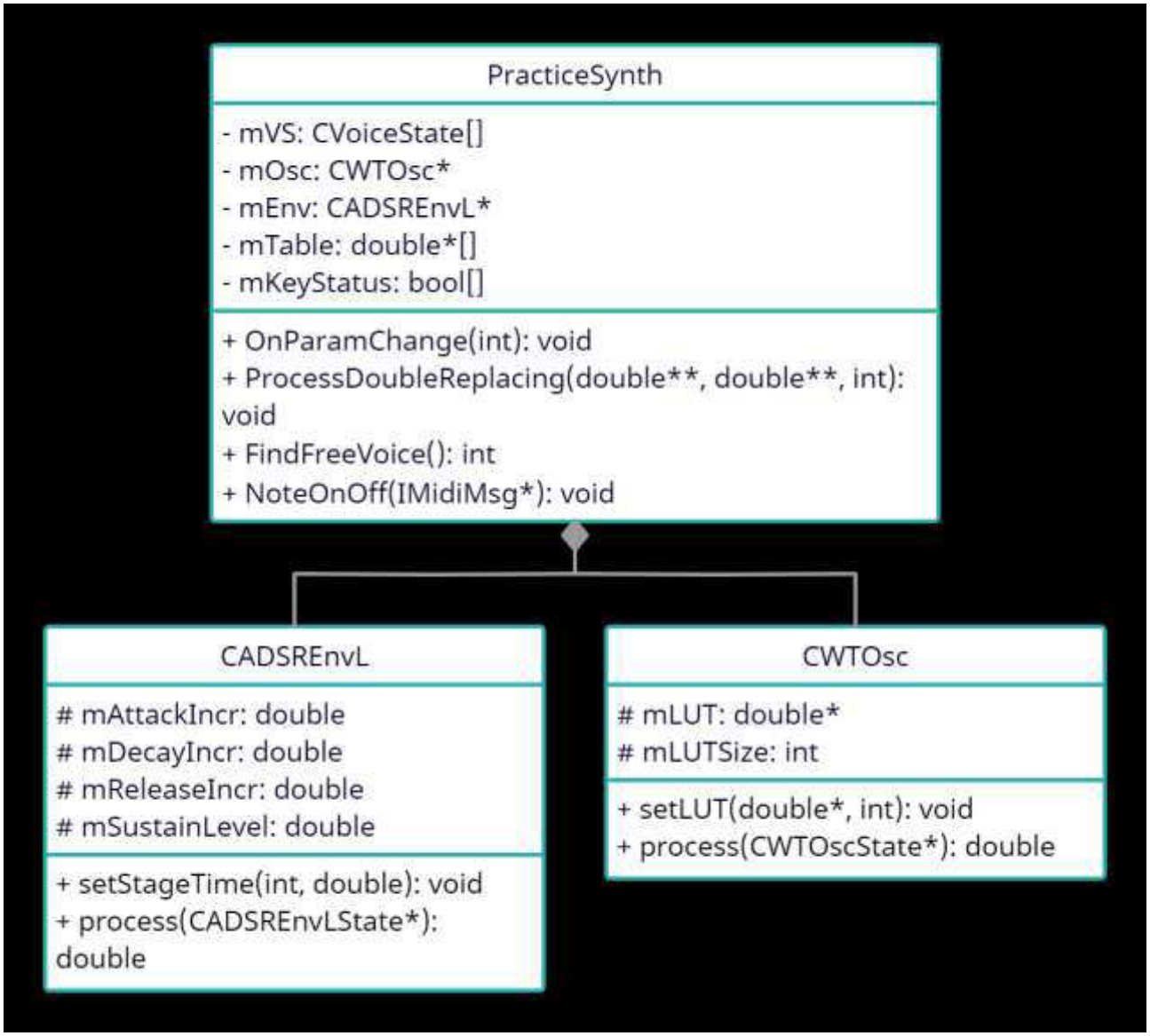

Рисунок 3 – UML-диаграмма классов

3.2 Представление класса PracticeSynth

Класс PracticeSynth является главным классом приложения, в котором происходит инициализация параметров, связывание и вызов функций других классов.

Основные методы и свойства класса:

– PracticeSynth(IPlugInstanceInfo instanceInfo) – конструктор, инициализирует волновые таблицы, ручки, MIDI-клавиатуру и параметры;

– CWTOsc* mOsc – указатель на объект класса генератора;

– CADSREnvL* mEnv – указатель на объект класса огибающей;

– double* mTable[4] – указатель на массивы волновых таблиц;

– bool mKeyStatus[128] – массив, который хранит состояние, нажата ли нота или нет;

– enum EParams – описывает параметры, которые должны изменяться ручками в интерфейсе приложения;

– void OnParamChange(int paramIdx) – вызывается, когда происходит поворот какой-либо ручки на интерфейсе приложения, и устанавливает значения в другие классы;

– void ProcessDoubleReplacing(double** inputs, double** outputs, int nFrames) – описывает процесс работы со звуком, в том числе вызывает метод генерации MIDI-сообщений;

– int FindFreeVoice() – ищет свободные голоса для воспроизведения MIDI-сообщений;

– void NoteOnOff(IMidiMsg* pMsg) – генерирует MIDI-сообщения о проигрывании или глушении ноты.

– CVoiceState mVS[16] – массив структур голосов, в данном случае одновременно может звучать не более 16 нот.

Структура CVoiceState хранит состояние голоса. Голос – это нота, состояние которой определяется, как свободная для звучания либо занятая.

3.3 Представление класса CADSREnvL

Класс CADSREnvL описывает работу ADSR-огибающей.

Основные методы и свойства класса:

– double mAttackIncr, mDecayIncr, mReleaseIncr – инкременты изменяющихся состояний;

– enum EADSREnvStage – перечисление всех состояний огибающей;

– void setStageTime – устанавливает инкременты состояний атаки, спада и затухания огибающей;

– void setSustainLevel – устанавливает уровень состояния удержания огибающей;

– double process – описывает процесс работы огибающей, переходы между состояниями и возвращает уровень сигнала.

Структура CADSREnvLState хранит:

– double mEnvValue – текущее значение огибающей;

– int mStage – текущее состояние огибающей;

– double mLevel – глубина огибающей, которая определяет максимальную громкость ноты.

3.4 Представление класса CWTOsc

Класс CWTOsc представляет собой генератор и отвечает за преобразование волновой таблицы в звуковую волну.

Основные методы и свойства класса:

– void setLUT – устанавливает внутри класса массив волновой таблицы и ее размер;

– double process – описывает процесс работы класса, вычисляет значения буфера из волновой таблицы и частоты дискретизации с учетом текущей фазы.

Структура CWTOscState хранит:

– double mPhase – текущее значение приращения фазы;

– double mPhaseIncr – инкремент изменения фазы.

3.5 Реализация графического интерфейса

Графический интерфейс приложения включает в себя несколько ручек, а также виртуальную MIDI-клавиатуру. С помощью MIDI-клавиатуры реализуется воспроизведение генерируемого звука. В зависимости от того, какая клавиша была нажата, определяется высота этого звука.

Ручки на графическом интерфейса определяют свойства генерируемого сигнала. В зависимости от их положения меняется форма волны, амплитуда, а также ее поведение. Данное приложение включает в себя ручку выбора формы волны, а также ручки Attack, Decay, Sustain, Release, представляющие собой ADSR-огибающую. Ручка выбора формы волны (OSC) позволяет переключаться между четырьмя генерируемыми волнами: синусоида, треугольная, пилообразная и квадратная (меандр). На рисунке 4 изображен графический интерфейс приложения.

Рисунок 4 – Графический интерфейс приложения

3.6 Волновые таблицы

Генерация волн реализуется в виде воспроизведения таблиц, которые представляют собой массивы размерности TABLE_SIZE и хранящие значения

от минус 1 до плюс 1. Массивы заполняются в момент выполнения программы при помощи функций, которые возвращают значения в зависимости от поданного итератора. Функции для таблиц с синусоидной, пилообразной, квадратной и треугольной формами волн изображены на рисунке 5.

```
double TABLE_SINE(int i, double table_size)
{
    return sin(i/table_size * 2. * M_PI);
}

double TABLE_SAW(int i, double table_size)
{
    return 1. - 2. * i/table_size;
}

double TABLE_SQUARE(int i, double table_size)
{
    return (i < table_size / 2) ? -1. : 1.;
}

double TABLE_TRIANGLE(int i, double table_size)
{
    double value = 1. - (2. * i/table_size);
    return 2. * (fabs(value) - 0.5);
}
```

Рисунок 5 – Функции для синусоидной, пилообразной, квадратной и треугольной форм волн

3.7 Инициализация ручек

Инициализация ручек начинается с добавления параметров выбора формы волны и ADSR-огибающей в перечисление EParams (рисунок 6).

```
enum EParams
{
  kWaveform = 0,
  kAttack,
  kDecay,
  kSustain,
  kRelease,
  kNumParams
};
```

Рисунок 6 – Перечисление EParams

Далее происходит инициализация этих самых параметров (рисунок 7). При инициализации в аргументах указывается имя параметра, значение по умолчанию, минимальное значение, максимальное значение и, в случае с InitDouble, шаг.

```
GetParam(kWaveform)->InitInt("Oscillator", 0, 0, 3);
GetParam(kAttack)->InitDouble("Attack", ATTACK_DEFAULT, TIME_MIN, TIME_MAX, 0.001);
GetParam(kDecay)->InitDouble("Decay", DECAY_DEFAULT, TIME_MIN, TIME_MAX, 0.001);
GetParam(kSustain)->InitDouble("Sustain", 1., 0., 1., 0.001);
GetParam(kRelease)->InitDouble("Release", RELEASE_DEFAULT, TIME_MIN, TIME_MAX, 0.001);
```

Рисунок 7 – Инициализация новых параметров

Затем к интерфейсу добавляются ручки и связываются с ними параметры. Анимация поворота ручек реализована за счет изображения, в котором располагается несколько фреймов с разными положениями ручки. При загрузке изображения указывается идентификатор, путь к изображению, а также количество фреймов, содержащихся в нем.

Для связывания параметров с ручкой используется метод AttachControl, который принимает в качестве аргумента объект класса IKnobMultiControl, тот в свою очередь принимает указатель на текущий экземпляр класса, координаты x и y, по которым должна располагаться ручка, идентификатор инициализированного ранее параметра, которым должна управлять ручка, а также указатель на объект IBitmap загруженного изображения (рисунок 8).

```
IBitmap knob4 = pGraphics->LoadIBitmap(KNOB4_ID, KNOB4_FN, 4);
IBitmap knob = pGraphics->LoadIBitmap(KNOB_ID, KNOB_FN, kKnobFrames);
pGraphics->AttachControl(new IKnobMultiControl(this, 1 * (kWidth - 48) / 5. - 48, 34, kWaveform, &knob4));
pGraphics->AttachControl(new IKnobMultiControl(this, 2 * (kWidth - 48) / 5. - 48, 34, kAttack, &knob));
pGraphics->AttachControl(new IKnobMultiControl(this, 3 * (kWidth - 48) / 5. - 48, 34, kDecay, &knob));
pGraphics->AttachControl(new IKnobMultiControl(this, 4 * (kWidth - 48) / 5. - 48, 34, kSustain, &knob));
pGraphics->AttachControl(new IKnobMultiControl(this, 5 * (kWidth - 48) / 5. - 48, 34, kRelease, &knob));
```

Рисунок 8 – Создание ручек и привязка параметров

При повороте какой-либо ручки вызывается метод OnParamChange, который принимает в качестве аргумента идентификатор этой ручки (рисунок 9). Значения повернутых ручек считываются и вносятся в переменные классов.

```
void PracticeSynth::OnParamChange(int paramIdx)
{
  IMutexLock lock(this);

  switch (paramIdx)
  {
    case kWaveform:
      mOsc = new CWTOsc(mTable[(int)GetParam(kWaveform)->Value()], TABLE_SIZE);
      break;
    case kAttack:
      mEnv->setStageTime(kStageAttack, GetParam(kAttack)->Value());
      break;
    case kDecay:
      mEnv->setStageTime(kStageDecay, GetParam(kDecay)->Value());
      break;
    case kSustain:
      mEnv->setSustainLevel( GetParam(kSustain)->Value() );
      break;
    case kRelease:
      mEnv->setStageTime(kStageRelease, GetParam(kRelease)->Value());
      break;
    default:
      break;
  }
}
```

Рисунок 9 – Метод OnParamChange

3.8 Реализация MIDI-клавиатуры

Инициализация клавиатуры также, как и инициализация ручек, начинается с загрузки изображений с помощью IBitmap. Первое изображение – это вся музыкальная клавиатура в состоянии покоя, которая просто лежит на интерфейсе приложения. Затем необходимо сделать так, чтобы нажатия клавиш были видны визуально. Для этого дополнительно используются изображения нажатых белых и черных клавиш, которые накладываются поверх существующей клавиатуры по нажатию соответствующей клавиши.

В этом случае метод AttachControl принимает в качестве аргумента объект класса IKeyboardControl, который принимает указатель на текущий экземпляр класса, координаты x и y, номер самой низкой ноты, количество

октав, ссылки на изображения нажатых белых и черных клавиш и массив coords, который хранит координаты нот в пределах одной октавы от ноты «до» до ноты «си». Это нужно, чтобы обозначить смещение каждой клавиши относительно левой границы (рисунок 10).

```
IBitmap keysBitmap = pGraphics->LoadIBitmap(KEYBOARD_ID, KEYBOARD_FN);
pGraphics->AttachControl(new IBitmapControl(this, 0, kHeight - 66 - 1, &keysBitmap));
IBitmap regular = pGraphics->LoadIBitmap(WHITE_KEY_ID, WHITE_KEY_FN, 6);
IBitmap sharp   = pGraphics->LoadIBitmap(BLACK_KEY_ID, BLACK_KEY_FN);
//                    C#     D#          F#      G#      A#
int coords[12] = { 0, 7, 12, 20, 24, 36, 43, 48, 56, 60, 69, 72 };
mKeyboard = new IKeyboardControl(this, kKeybX, kKeybY, 48, 5, &regular, &sharp, coords);
pGraphics->AttachControl(mKeyboard);
```

Рисунок 10 – Инициализация MIDI-клавиатуры

3.9 Обработка MIDI-сообщений

За процесс работы со звуком отвечает метод ProcessDoubleReplacing. В нем происходит генерация MIDI-сообщений и их обработка. После того, как была нажата какая-либо клавиша, создается сообщение, в котором хранится номер нажатой клавиши. Это сообщение уведомляет о том, что должна быть проиграна нота. При чем, отпускание клавиш тоже является сообщением, которое говорит о том, что нота должна прекратить звучать. Затем оно отправляется в очередь MIDI-сообщений.

Далее сообщение считывается из очереди, и, если это сообщение о том, что должна проиграться либо заглушиться нота, вызывается метод NoteOnOff, в который передается текущее сообщение (рисунок 11).

```
if (status == IMidiMsg::kNoteOn && velocity) // Note on
{
  v = FindFreeVoice(); // or free the quietest
  mVS[v].mKey = note;
  mVS[v].mOsc_ctx.mPhaseIncr = (1. / mSampleRate) * midi2CPS(note);
  mVS[v].mEnv_ctx.mLevel = (double) velocity / 127.;
  mVS[v].mEnv_ctx.mStage = kStageAttack;
  mActiveVoices++;
}
else  // Note off
{
  for (v = 0; v < MAX_VOICES; v++)
  {
    if (mVS[v].mKey == note)
    {
      if (mVS[v].GetBusy())
      {
        mVS[v].mKey = -1;
        mVS[v].mEnv_ctx.mStage = kStageRelease;
        mVS[v].mEnv_ctx.mReleaseLevel = mVS[v].mEnv_ctx.mPrev;

        return;
```

Рисунок 11 – Фрагмент метода NoteOnOff

Если это сообщение о проигрывании ноты, ищется свободный для проигрывания голос. Голоса представляют собой массив структур CVoiceState, которые хранят объекты классов CWTOscState и CADSREnvLState, а также номера нот, которые необходимо проиграть (рисунок 12).

```
struct CVoiceState
{
  CWTOscState mOsc_ctx;
  CADSREnvLState mEnv_ctx;
  int mKey;

  CVoiceState()
  {
    mKey = -1;
  }
```

Рисунок 12 – Фрагмент структуры CVoiceState

В объекты найденного свободного голоса устанавливаются нота, инкремент фазы, который вычисляется из частоты ноты деленной на частоту дискретизации, уровень громкости ноты и стадия атаки.

Если это сообщение о глушении ноты, в массиве голосов ищется нужная играющая нота и мгновенно устанавливается в стадию затухания.

После предыдущих манипуляций начинается пробег по всем голосам. Если какой-то из голосов находится в стадии, отличной от холостой, на выход поступает сигнал в виде произведения работы класса обработки волновой таблицы CWTOsc и работы класса обработки ADSR-огибающей CADSREnvL (рисунок 13).

```
for(int v = 0; v < MAX_VOICES; v++) // for each vs
{
  vs = &mVS[v];

  if (vs->GetBusy())
  {
    output += mOsc->process(&vs->mOsc_ctx) * mEnv->process(&vs->mEnv_ctx);
  }
}
```

Рисунок 13 – Цикл пробега по голосам

3.10 Реализация ADSR-огибающей

Фрагмент реализации, включающий структуру, перечисление и класс огибающей, приведен на рисунке 14.

Для ADSR-огибающей используется перечислимый тип EADSREnvStage, в котором присутствуют все стадии, а также структура CADSREnvLState. Кроме состояний атаки, спада, удержания и затухания, также существует стадия kIdle, что означает холостое состояние.

```
struct CADSREnvLState
{
  double mEnvValue;
  int mStage;
  double mLevel;
  double mPrev;
  double mReleaseLevel;

  CADSREnvLState()
  {
    mEnvValue = 0.;
    mStage = kIdle;
    mLevel = 0.;
    mReleaseLevel = 0.;
    mPrev = 0.;
  }

} WDL_FIXALIGN;
```

```
class CADSREnvL
{
protected:
  double mAttackIncr, mDecayIncr, mReleaseIncr;
  double mSustainLevel, mReleaseLevel;
  double mPrev;
  double mSampleRate;

public:
  CADSREnvL()
  {
    mSustainLevel = 1.;
    mReleaseLevel = 0.;
    mPrev = 0.;
    mSampleRate = 44100.;

    setStageTime(kStageAttack, 1.);
    setStageTime(kStageDecay, 100.);
    setStageTime(kStageRelease, 20.);
  }
```

```
enum EADSREnvStage
{
  kIdle = 0,
  kStageAttack,
  kStageDecay,
  kStageSustain,
  kStageRelease,
};
```

Рисунок 14 – Фрагмент реализации ADSR-огибающей

Класс CADSREnvL отвечает за реализацию огибающей. Он хранит свое значение огибающей и значения инкрементов каждой стадии, которые вычисляет в зависимости от положения ручек. Значение огибающей будет меняться от нуля до единицы. Вычисленные инкременты изменяют по ходу времени значение огибающей от нуля до единицы и обратно, что и реализовывает весь механизм.

Допустим число, очень близкое к нулю, – условный ноль, а число, очень близкое к единице, – условная единица. Механизм начинается со стадии атаки. Длится стадия атаки до тех пор, пока инкремент атаки, начиная с нуля, не поднимет значение огибающей выше условной единицы. После этого значение устанавливается в единицу, и начинается стадия спада. Длится она до тех пор, пока значение при помощи инкремента спада не упадет ниже условного нуля. Затем значение устанавливается в единицу, и следует стадия удержания, но в отличии от предыдущих стадий она не переходит самостоятельно в следующую стадию, а продолжает удерживаться до тех пор, пока нажата клавиша. Последняя стадия – стадия затухания – длится до тех пор, пока значение огибающей с помощью инкремента затухания не упадет ниже условного нуля. Тогда значение устанавливается в ноль и состояние переходит в холостую

стадию, что говорит о том, что звук перестал звучать. Механизм описанный выше проходит в методе process, в этом же методе вычисляется и возвращается амплитуда сигнала (от нуля до единицы), соответствующий текущему положению в огибающей.

3.11 Реализация генератора

На рисунке 15 изображен фрагмент реализации, включающий структуру CWTOscState, а также часть класса CWTOsc.

```
struct CWTOscState
{
  double mPhase;
  double mPhaseIncr;

  CWTOscState()
  {
    mPhase = 0.;
    mPhaseIncr = 0.;
  }

} WDL_FIXALIGN;
```

```
class CWTOsc
{
protected:
  const double* mLUT;
  unsigned long int mLUTSize;
  unsigned long int mLUTSizeM;
  double mLUTSizeF;

public:

  CWTOsc(const double* LUT, unsigned long int LUTSize)
  {
    setLUT(LUT, LUTSize);
  }
```

Рисунок 15 – Фрагмент реализации преобразования волновой таблицы

Класс CWTOsc принимает указатель на волновую таблицу и ее размер. Значение фазы, хранящееся в структуре, постоянно меняется, так как оно содержит в себе информацию о том, в каком моменте цикла генерации формы волны осциллятор находится. Метод process этого класса возвращает значения буфера, преобразуя из волновой таблицы с учетом частоты дискретизации (рисунок 16). Это реализуется за счет того, что ранее было вычислено приращение фазы.

```
inline double process(CWTOscState* pState)
{
  pState->mPhase = wrap(pState->mPhase, 0., 1.);
  const double output = lerp(pState->mPhase * mLUTSizeF, mLUT, mLUTSizeM);
  pState->mPhase += pState->mPhaseIncr;

  return output;
}
```

Рисунок 16 – Метод обработки класса

3.12 Вывод по главе

В этой главе были описаны этапы разработки синтезатора. Программа представляет собой три класса: класс генератора, класс огибающей и класс приложения, объединяющий все модули программы.

Класс генератора преобразует волновую таблицу в звуковой волну, за счет инкрементирования и приращения фазы звука.

Класс огибающей описывает стадии звука и процесс перехода между ними.

Класс приложения работает с пользовательским интерфейсом программы, создает MIDI-сообщения и формирует звук за счет остальных классов.

По нажатии клавиши на музыкальной клавиатуре программа создает MIDI-сообщение, которое хранит номер нажатой ноты, вычисляется частота звука этой ноты. В этот момент начинает двигаться фаза звука и приводится в действие огибающая, переходя в стадию атаки.

При отпускании клавиши на музыкальной клавиатуре программа создает MIDI-сообщение, которое хранит номер отпущенной ноты. Затем эта нота мгновенно переходит в стадию затухания – амплитуда звука начинает опускаться до полной тишины.

4 Демонстрация работы программы

4.1 Интерфейс пользователя

Пользовательский интерфейс программы представляет собой пять ручек, предназначенных для настройки синтезатора.

Ручка OSC позволяет выбрать форму волны для синтезируемого звука и может принимать четыре положения (рисунок 17):

– положение ручки 1 устанавливает синусоидную форму волны;

– положение ручки 2 устанавливает треугольную форму волны;

– положение ручки 3 устанавливает пилообразную форму волны;

– положение ручки 4 устанавливает квадратную форму волны.

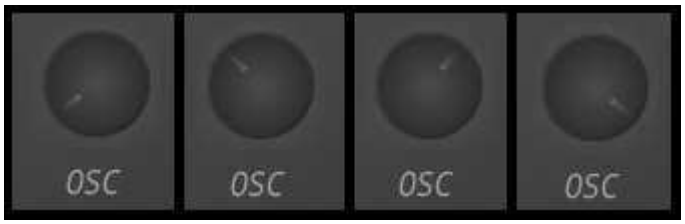

Рисунок 17 – Положения ручки OSC 1,2,3 и 4 соответственно

Ручки Attack, Decay, Sustain, Release позволяют настроить параметры ADSR-огибающей и в отличии от ручки OSC могут принимать множество значений (рисунок 18):

– Ручка Attack устанавливает время нарастания звука, то есть время, за которое амплитуда звука после нажатия клавиши достигнет максимального значения начиная от нуля. Поворот ручки по часовой стрелке увеличивает время нарастания;

– Ручка Decay устанавливает время спада звука, то есть время, за которое амплитуда звука опустится от максимального значения до значения ручки Sustain. Поворот ручки по часовой стрелке увеличивает время спада;

– Ручка Sustain устанавливает амплитуду звука в состоянии удержания. Поворот ручки против часовой стрелки уменьшает амплитуду звука вплоть до нуля;

– Ручка Release устанавливает время затухания звука, то есть время, за которое амплитуда звука упадет до нуля после отпускания клавиши. Поворот ручки по часовой стрелке увеличивает время затухания.

Рисунок 18 – Ручки настройки ADSR-огибающей

Музыкальная клавиатура позволяет воспроизводить синтезируемый звук (рисунок 19). Клавиатура имеет пять октав, начиная с ноты до малой октавы и заканчивая нотой до пятой октавы.

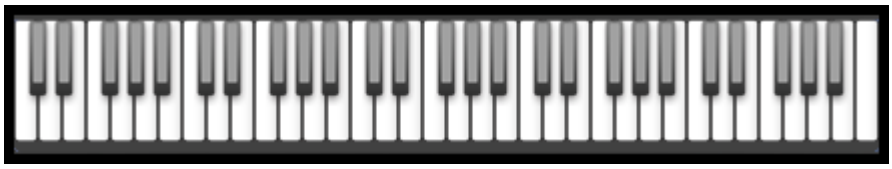

Рисунок 19 – Музыкальная клавиатура синтезатора

4.2 Примеры настройки синтезатора

Здесь рассматривается несколько примеров настройки синтезатора.

Пример 1. Форма волны – синусоидная, минимальное время атаки, минимальное время спада, максимальная амплитуда удержания, короткое время затухания (рисунок 20). Данный пример описывает звук, который появляется мгновенно и удерживается на максимальной амплитуде. После отпускания клавиши звук очень быстро затухает.

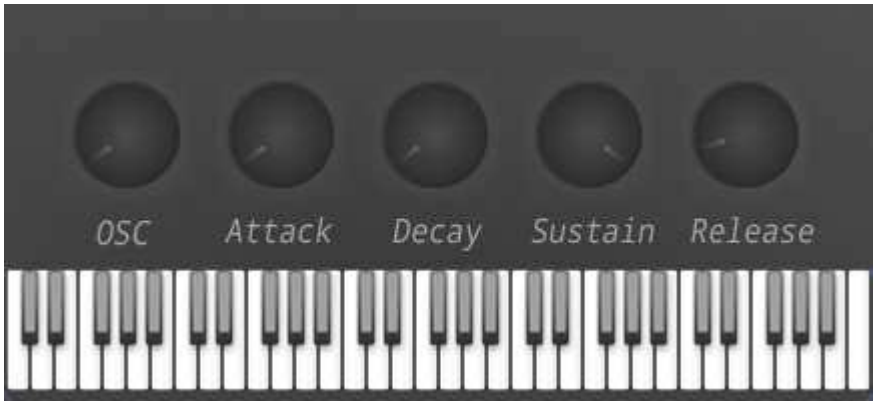

Рисунок 20 – Настройка синтезатора. Пример 1

Пример 2. Форма волны – треугольная, минимальное время атаки, среднее время спада, нулевая амплитуда удержания, минимальное время затухания (рисунок 21). В данном случае звук мгновенно вступает и начинает затихать по удержании клавиши. При отпускании – звук затухает мгновенно.

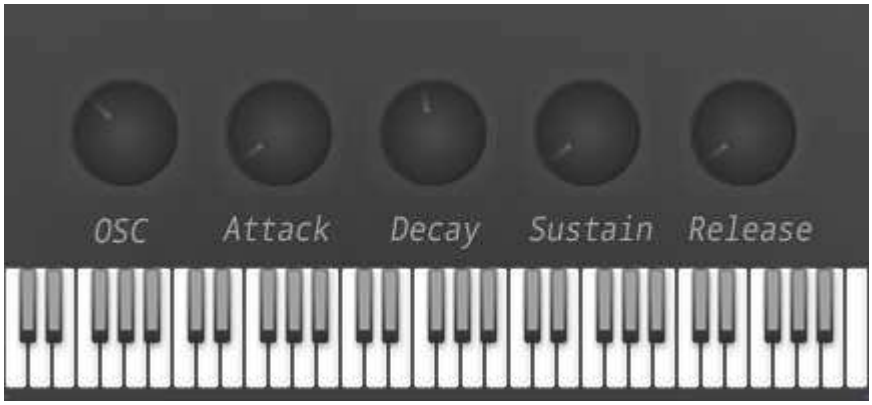

Рисунок 21 – Настройка синтезатора. Пример 2

Пример 3. Форма волны – пилообразная, минимальное время атаки, минимальное время спада, средняя амплитуда удержания, среднее время затухания (рисунок 22). В этом случае звук появляется мгновенно, по удержанию клавиши принимает среднее значение амплитуды, при отпускании – плавно затухает. Данный случай позволяет показать полифонию – когда ноты, плавно затухая, продолжают звучать, но можно проиграть другие, не прекращая звучания предыдущих.

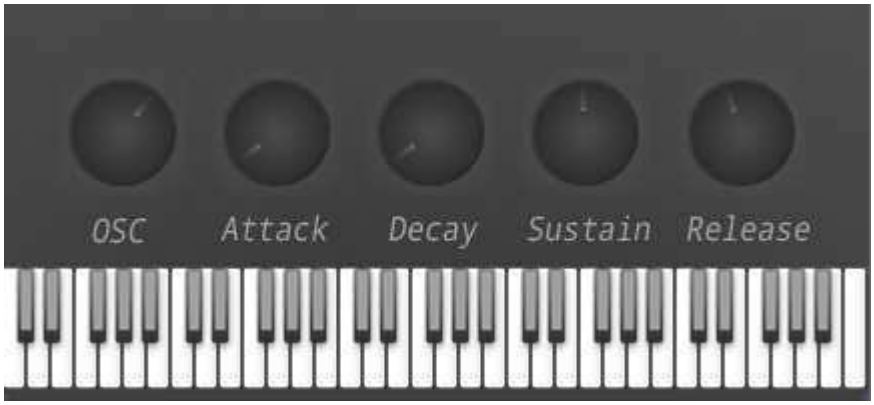

Рисунок 22 – Настройка синтезатора. Пример 3

Пример 4. Форма волны – квадратная, небольшое время атаки, минимальное время спада, максимальная амплитуда удержания, среднее время затухания (рисунок 23). В данном примере, по удержании клавиши звук плавно нарастает, а по отпускании – начинает затухать. Этот пример также, как и предыдущий, позволяет продемонстрировать полифонию.

Рисунок 23 – Настройка синтезатора. Пример 4

4.3 Вывод по главе

В этой главе был продемонстрирован интерфейс программы, а также примеры настройки полученного синтезатора. Интерфейс имеет ручку выбора формы волны, четыре ручки ADSR-огибающей, а также MIDI-клавиатуру.

ЗАКЛЮЧЕНИЕ

В заключение был получен синтезатор с ручками выбора формы волны и ADSR-огибающей. Синтезатор предоставляет быстрый доступ к синтезу простых звуковых волн, требуя лишь небольшое количество ресурсов процессора. Также для синтезатора был разработан незамысловатый внешний вид, добавлены подписи, а также задний фон (рисунок 24).

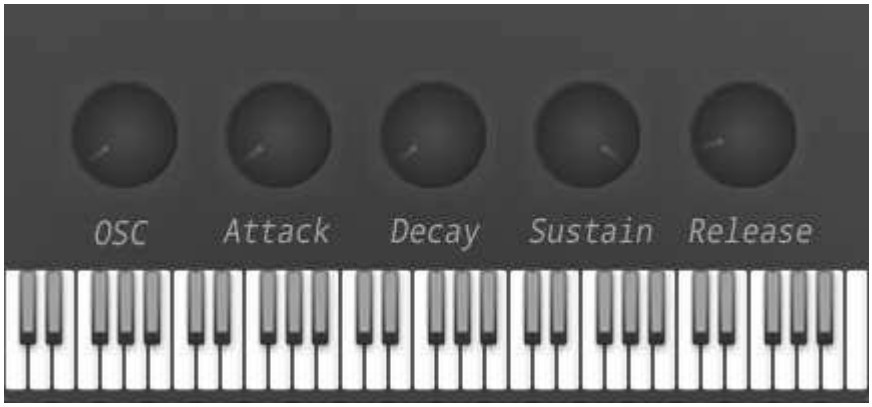

Рисунок 24 – Синтезатор с MIDI-клавиатурой

Подведя итоги, можно выделить ряд недостатков:

– отсутствует защита от алиасинга, таким образом проигрывая пилообразную и квадратную волны на высоких нотах, можно услышать сильные искажения звука;

– отсутствует возможность использовать приложение с компьютерной клавиатурой, так что невозможно (при помощи мыши) проиграть несколько нот одновременно;

– не все процессы перехода состояний ADSR-огибающей происходят плавно;

– в приложении отсутствует ручка выходной громкости, что в некоторых случаях приносит некоторые неудобства.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. A brief Introduction to MIDI // Wayback Machine URL: https://web.archive.org/web/20120830211425/http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/ (дата обращения: 20.04.21).

2. Шилов В. Л. Практический словарь иностранных музыкальных терминов // М.: Композитор. – 2003.

3. Chronology 1953-1993 // Moog archives URL: http://moogarchives.com/chrono.htm (дата обращения: 21.04.21).

4. Vintage synth explorer URL: http://www.vintagesynth.com (дата обращения: 20.04.21).

5. olilarkin/wdl-ol // GitHub URL: https://github.com/olilarkin/wdl-ol (дата обращения: 21.04.21).

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись инициалы, фамилия
« 10 » 06 2021 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

Звуковой синтезатор с MIDI-клавиатурой

Руководитель	8.06.21 _____ подпись, дата	ст. преп. каф. ВТ	И. В. Матковский
Выпускник	8.06.21 _____ подпись, дата		Р. С. Янмурзин
Нормоконтролер	8.06.21 _____ подпись, дата		И. В. Матковский

Красноярск 2021