

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Кубанский государственный технологический университет»
(ФГБОУ ВО «КубГТУ»)

Институт компьютерных систем и информационной безопасности

Кафедра информационных систем и программирования

Направление подготовки: 09.03.04 Программная инженерия
(код и наименование направления)

Профиль: Проектирование и разработка программного обеспечения
(наименование профиля)


КУРСОВОЙ ПРОЕКТ

по дисциплине Технологии разработки программного обеспечения
(наименование дисциплины)



на тему: «Кодовый замок»
(тема курсовой работы)

Выполнила студентка 2 курса группы 19-КБ-ПР2
Наталочка Анастасия Денисовна
(фамилия, имя, отчество)

Допущен к защите 15.12.2020г.
(дата)

Руководитель (нормоконтролер) работы  к.т.н., доц. Попова О.Б.
(должность, подпись, дата)

Защищен 15.12.2020г. Оценка 
(дата)

Члены комиссии:   ст. преп. Кушнир Н.В.
к.т.н., доц. Тотухов К.Е.
(должность, подпись, дата, расшифровка подписи)

Краснодар
2020г.

ФГБОУ ВО «Кубанский государственный технологический университет»
(ФГБОУ ВО КубГТУ)

Институт компьютерных систем и информационной безопасности
Кафедра информационных систем и программирования
Направление подготовки 09.03.04 Программная инженерия
(код и наименование направления)
Профиль Проектирование и разработка программного обеспечения
(наименование профиля)

УТВЕРЖДАЮ

Зав. кафедрой М.В. Янаева
«22» сентября 2020 г.

ЗАДАНИЕ

на курсовой проект

Студентке Наталочка Анастасии Денисовне группы 19-КБ-ПР2 курса 2
(Ф.И.О.)

Тема проекта: Кодовый замок

(утверждена указанием директора института № от 2020г.)

План работы:

1. Исследование предметной области
2. Проектирование приложения
3. Реализация программы

Объем работы:

а) пояснительная записка 36 с.

Рекомендуемая литература

1. Зараминский Е.М. Управление жизненным циклом информационных систем
2. Васильев К. Применение ИИ и машинного проектирования
3. Пирот Г. Основы функционального анализа и проектирования систем

Срок выполнения: с «1» сентября по «30» декабря 2020г.

Срок защиты: «21» декабря 2020г.

Дата выдачи задания: «1» сентября 2020г.

Дата сдачи работы на кафедру: «21» декабря 2020г.

Руководитель работы доцент. [подпись] к.т.н., доц. Попова О.Б.
(должность, подпись)

Задание приняла студентка [подпись] Наталочка А.Д.

Реферат

Курсовая работа: 30 страниц, 8 рисунков, 2 таблицы, 9 источников, 2 приложения.

MIDI ИНТЕРФЕЙС, МЕТОДЫ СИНТЕЗА ЗВУКА, ЧАСТОТНО-МОДУЛЯЦИОННЫЙ СИНТЕЗ, ТАБЛИЧНО-ВОЛНОВОЙ СИНТЕЗ, СЭМПЛИРОВАНИЕ, ФАЙЛОВЫЙ СТАНДАРТ, ПРОИГРЫВАТЕЛЬ MIDI-МЕЛОДИЙ, СЕКВЕНСОР

В данной курсовой работе подробно рассмотрен стандарт цифровой звукозаписи MIDI и синтез цифрового звука. Благодаря сравнительно небольшому размеру файлов, формат MIDI и на сегодняшний день остаётся одним из самых распространенных.

Целью настоящей работы является исследование алгоритмов синтеза звука и его воспроизведения, а также изучение работы и разработка приложения для проигрывания и создания мелодий в данном формате.

Основные полученные результаты:

- проведён анализ существующего программного и аппаратного обеспечения для работы с MIDI-композициями;
- изучены различные методы синтеза звука;
- спроектирована и разработана прикладная программа, позволяющая синтезировать и прослушивать MIDI-композиции.

Данный программный продукт можно использовать как пример общей работы проигрывателя и секвенсора и применять для прослушивания композиций и генерации простых мелодий.

Содержание

Введение	5
1 Описание MIDI-интерфейса	6
1.1 Особенности MIDI-коммутации	6
2 Описание файлового стандарта	8
3 Методы синтеза звука	11
3.1 Аддитивный синтез звука	11
3.2 Субтрактивный синтез звука	12
3.3 Частотно-модуляционный синтез звука	13
3.4 Сэмплерный синтез звука	14
3.5 Таблично-волновой синтез звука	14
4 Анализ MIDI аппаратуры и программного обеспечения	15
5 Технические требования	18
5.1 Назначение разработки	19
5.2 Требования к программе или программному изделию	19
5.2.1 Требования к функциональным характеристикам	19
5.2.2 Требования к составу и параметрам технических средств	20
6 Проектирование программы	20
7 Реализация программного продукта	22
7.1 Выбор инструментальной среды разработки	22
7.2 Реализация пользовательского интерфейса	22
7.2.1 Описание пользовательского интерфейса	23
7.3 Реализация функционала проигрывателя	26
7.4 Реализация синтеза MIDI-композиций	26
Заключение	29
Список использованных источников	30
Приложение А Листинг программы.....	31
Приложение Б Проверка на плагиат.....	39

Введение

Появившаяся в начале восьмидесятых годов MIDI-технология вскоре получила новый импульс в связи с широким распространением персональных компьютеров. MIDI – стандарт цифровой звукозаписи, описывающий протоколы передачи данных, цифровой интерфейс и электрические разъёмы, соединяющие широкий спектр электронных музыкальных инструментов, компьютеров и связанных с ними аудиоустройств для воспроизведения, редактирования и записи музыкальных композиций.

Суть MIDI-технологии можно изложить так: компьютер не просто проигрывает нужную мелодию, а синтезирует ее с помощью звуковой карты. MIDI-мелодии - это всего лишь системы команд, управляющие звуковой картой, коды нот, которые она должна воспроизвести. Эта технология идеальна для компьютерных композиторов, поскольку позволяет с легкостью изменять любые параметры созданной на компьютере мелодии - заменять инструменты, добавлять или удалять их, изменять темп и тональность. MIDI-технология остаётся ведущей в компьютерной и аппаратно-студийной области. Она совершенствуется, учитывает новые требования и новые технические возможности, а также вовлекает в сферу своего влияния все новые и новые области, для которых она и не предназначалась, — управление магнитофонами, устройствами звуковой обработки, микшерскими пультами (не говоря уже о мультимедийных продуктах и компьютерных играх).

Данная курсовая работа посвящена разработке прикладной программы, позволяющей воспроизводить и синтезировать MIDI-композиции, секвенсора.

Целью данной курсовой работы является закрепление и углубление теоретических знаний и практических умений в области информационных технологий, приобретение требуемых компетенций в работе с цифровым звуком, укрепление навыков исследовательской работы и проектирования.

1 Описание MIDI-интерфейса

В конце 1970-х решением проблемы подключения единой клавиатуры к множеству различных звуковых генераторов стал цифровой интерфейс музыкальных инструментов (Musical Instrument Digital Interface). Стандарт описывает аппаратный интерфейс, который позволяет соединять электронные музыкальные инструменты и компьютеры различных производителей, описывает протоколы связи для передачи данных от одного устройства к другому. MIDI-устройства могут взаимодействовать с программными приложениями, используя коммуникационный протокол MIDI.

Каждое нажатие клавиши на клавиатуре, усиление или ослабление давления на клавишу, переключение тумблеров, нажатие на педаль и т.д. инструмент преобразует в соответствующее сообщение, в закодированном виде переходящее от инструмента к инструменту – событие MIDI (MIDI event). Эти события распределяются по шестнадцати каналам (каждое сообщение может идти только по одному из них), а реагировать на них будут только те устройства, которые запрограммированы на ответ именно по этому каналу.

Альтернативным вариантом может быть одиночный синтезатор, который сам может управлять всеми каналами. Сообщения генерируются и отправляются достаточно быстро – 1000-1500 в секунду, поэтому они весьма точно описывают не только сами действия исполнителя, но и его индивидуальную манеру игры. По MIDI можно соединить практически любое количество инструментов, и все они могут обмениваться сообщениями друг с другом. [1, с. 277-278]

1.1 Особенности MIDI-коммутации

Устройства соединяются кабелями. Выход данных ведущего устройства соединяется со входом ведомого устройства. Данные по кабелю

передаются только в одном направлении, от ведущего устройства к ведомому (например, от музыкальной клавиатуры к синтезатору). Для двусторонней передачи данных требуется второй кабель.

Большая часть MIDI-устройств не копируют сообщения с входного на выходной разъём. Существует третий тип разъёма, на котором дублируется поток данных со входа. Этот тип разъёма позволяет соединить в цепочку произвольное количество синтезаторов. Однако такой тип разъёма есть не у всех синтезаторов.

В устройствах со стандартным MIDI-интерфейсом эти разъёмы обозначаются как MIDI IN, MIDI OUT и MIDI THRU и описаны в таблице 1.

Таблица 1 – Виды MIDI-разъёмов

MIDI IN	Входной разъем, через который поступает MIDI-информация с других устройств
MIDI OUT	Выходной разъем, через который устройство передает информацию о производимых на нем действиях
MIDI THRU	Выходной разъем, через который устройство в неизменном виде пересылает информацию, полученную через MIDI IN.

Поток MIDI-информации передается побайтно. Для контроля за состоянием линии в начале каждого байта передается стартовый бит (1), а в конце — стоповый (0). Каждый байт состоит из 8 значащих битов.

Любой передаваемый байт является байтом либо статуса, либо значения. Статусный байт всегда первый в MIDI-сообщении, он определяет его тип и номер MIDI-канала. В каждом MIDI-сообщении содержится только один статусный байт. Байты значения содержат параметры, необходимые для данного типа MIDI-сообщения.

Каждое сообщение содержит информацию о двух параметрах: номере нажатой клавиши и силе удара по ней.

Большинство параметров MIDI могут принимать значения от 0 до 127 (1 байт). Поэтому размер полной MIDI-клавиатуры составляет 128 клавиш. Так как музыкантам привычнее оперировать нотами и номерами октав, в MIDI описано соответствие между номером клавиши и его интуитивным значением. Однако номера октав здесь отличаются от принятых на традиционных акустических инструментах. [2]

Зависимость воспроизводимых нот от ID можно увидеть на рисунке 1.

Octave	Октава	Note Numbers											
		C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	—	0	1	2	3	4	5	6	7	8	9	10	11
0	Субконтроктава	12	13	14	15	16	17	18	19	20	21	22	23
1	Контроктава	24	25	26	27	28	29	30	31	32	33	34	35
2	Большая	36	37	38	39	40	41	42	43	44	45	46	47
3	Малая	48	49	50	51	52	53	54	55	56	57	58	59
4	Первая	60	61	62	63	64	65	66	67	68	69	70	71
5	Вторая	72	73	74	75	76	77	78	79	80	81	82	83
6	Третья	84	85	86	87	88	89	90	91	92	93	94	95
7	Четвертая	96	97	98	99	100	101	102	103	104	105	106	107
8	Пятая	108	109	110	111	112	113	114	115	116	117	118	119
9	—	120	121	122	123	124	125	126	127				

Рисунок 1 – Номера нот в MIDI-стандарте

2 Описание файлового стандарта

В отличие от других форматов это не оцифрованный звук, а наборы команд (проигрываемые ноты, ссылки на проигрываемые инструменты, значения изменяемых параметров звука). Файлы MIDI не определяют точное звучание инструмента. MIDI первоначально был связующим протоколом, звук зависел от того, какие устройства подключались к каналу.

Стандартный MIDI-файл (SMF – Standard MIDI File) – это специально разработанный формат файлов, предназначенный для хранения данных, записываемых и/или исполняемых секвенсором, который может быть, как программой для компьютера, так и аппаратно-выполненным модулем.

Главным преимуществом MIDI заключается в том, что файлы, как правило, имеют на несколько порядков меньший размер, чем оцифрованный звук сравнимого качества – 1 минута MIDI-звука занимает в среднем 10 Кбайт, поскольку это не детальная запись звука, а фактически некоторый расширенный электронный эквивалент традиционной нотной записи.[3] Но это же свойство одновременно является и недостатком: поскольку звук не детализирован, то разное оборудование будет воспроизводить его по-разному, что в принципе может даже заметно исказить авторский музыкальный замысел, кроме того низкая скорость передачи информации, узкий диапазон изменения параметров и ограниченная сфера применения. [1, с. 278]

Но использование файлов MIDI для обмена музыкой делает совместимость более важным фактором. Если один и тот же файл звучит как два различных инструмента на разных синтезаторах, то польза от обмена файлами MIDI значительно уменьшается. В результате были приняты также следующие стандарты:

- General MIDI (GM) – первая разработка фирмы Roland, унифицирующая набор MIDI-инструментов, предложенная в 1991 году. Он объединил изделия многих производителей под именем General MIDI (System) Level 1. Сейчас его поддерживает абсолютное большинство моделей электронных синтезаторов, звуковых карт и клавиатур, но, так как требования этого стандарта уже давно морально устарели, обычно, современные электронные инструменты поддерживают его лишь для базовой совместимости;

- General Standard (GS) – общий стандарт фирмы Roland, определяющий набор тембров. Начал развиваться в 1991 году сразу за GM и в последующем неоднократно расширялся в связи с выпуском новых продвинутых моделей. Помимо элементов стандарта General MIDI включает в себя дополнительные наборы мелодических и ударных инструментов, а также различные эффекты (скрип двери, звук мотора, крики и т.д.);

– Extended General (XG) – новый стандарт фирмы Yamaha, включающий несколько сотен мелодических и ударных инструментов, ставший альтернативой формату GS;

– General MIDI 2 (GM2) – в 1999 году MMA (MIDI Manufacturers Association) выпустила расширение GM, получившее наименование General MIDI Level 2 (GM2). В новом стандарте расширили полифонию и палитру доступных инструментов до 256, добавили ряд новых контроллеров. В GM2 прослеживается влияние стандартов Роланда и Ямахи. Несмотря на соглашение между этими фирмами пока не получил широкого распространения. [4]

Ниже в таблице 2 сведены глобальные характеристики данных стандартов.

Таблица 2 – Общие характеристики MIDI-стандартов

Параметр сравнения	GM	GS	XG	GM2
Полифония (количество одновременно звучащих голосов)	24	24 и выше	32 и выше	32
Количество MIDI-каналов	16	16, 32	16, 32	16
Номера каналов для набора перкуссии (ударных)	10	Любые	Любые	10-11
Количество мелодических инструментов	128	226 и выше	480 и выше	256
Количество наборов ударных	1	9 и более	9 и более	9
Реакция на давления индивидуальных клавиш (Polyphonic Aftertouch)	нет	да	да	нет

Существует три типа файлов MIDI, отличающихся организацией дорожек. Файлы нулевого типа содержат всего одну дорожку. Очевидно, что

их легче всего проигрывать. Именно таким файлам следует отдавать предпочтение, если необходимо легко воспроизвести файлы MIDI где-либо ещё. Файлы первого типа содержат несколько дорожек, проигрывающихся одновременно. Программы для проигрывания файлов первого типа должны перед проигрыванием преобразовать и выровнять данные в единый поток событий. Наконец, файлы второго типа содержат несколько дорожек, но не предполагают никакой связи между дорожками, используются относительно редко.

3 Методы синтеза звука

Создание (синтез) звука в основном преследует следующие цели: имитация различных естественных звуков (шум ветра и дождя, звук шагов, стук сердца и т.п.), а также акустических музыкальных инструментов (имитационный синтез), и получение принципиально новых звуков, не встречающихся в природе (чистый синтез). Обработка звука обычно направлена на получение новых звуков из уже существующих (например, искажение голоса), либо придание им дополнительных качеств или устранение существующих (добавление эффекта хора, удаление помех или щелчков). Каждый из методов синтеза и обработки имеет свою математическую и алгоритмическую модель, что позволяет их компьютерную реализацию; однако, многие методы, будучи реализованы точно, требуют слишком большого объема вычислений, отчего их обычно реализуют с какой-либо степенью допущения.[6]

3.1 Аддитивный синтез звука

Аддитивный (additive) синтез основан на утверждении Фурье о том, что любое периодическое колебание можно представить в виде суммы чистых тонов (синусоидальных колебаний с различными частотами и амплитудами).

Для этого требуется несколько синусоидальных генераторов, управляемых независимо друг от друга. Их выходные сигналы суммируются для получения результирующего сигнала. На этом методе основан принцип создания звука в духовом органе.[7]

Достоинства метода заключается в том, что с его помощью возможно синтезировать любой периодический звук, и процесс создания звука предсказуем (изменение настройки одного из генераторов не влияет на остальную часть спектра звука). Основной же недостаток – для звуков сложной структуры может потребоваться большое количество генераторов, что значительно повышает сложность и стоимость реализации. [6]

3.2 Субтрактивный синтез звука

Субтрактивный (subtractive), разностный синтез идеологически противоположен аддитивному. В основу положена генерация звукового сигнала с богатым спектром (множеством частотных составляющих) с его последующей фильтрацией (выделением одних составляющих и ослаблением других) – по такому же принципу устроен речевой аппарат человека. В качестве исходных сигналов обычно используются меандр, с переменной скважностью (отношением всего периода к положительному полупериоду), пилообразный – прямой и обратный, и треугольный, а также различные виды шумов (случайных непериодических колебаний). Основным органом синтеза в этом методе служат управляемые фильтры: резонансный (полосовой) – с изменяемым положением и шириной полосы пропускания и фильтр нижних частот (ФНЧ) с изменяемой частотой среза. Для каждого фильтра также регулируется добротность (Q) – крутизна подъема или спада на резонансной частоте.

Достоинства субтрактивного метода – относительно простая реализация и довольно широкий диапазон синтезируемых звуков. На этом методе построено множество студийных и концертных синтезаторов. Главный

недостаток в том, что для синтеза звуков со сложным спектром требуется большое количество сложных и дорогих управляемых фильтров. [7]

3.3 Частотно-модуляционный синтез звука

В основу частотно-модуляционного (frequency modulation - FM) синтеза положена взаимная модуляция по частоте между несколькими синусоидальными генераторами. Каждый из таких генераторов, имеющий собственный формирователь амплитудной огибающей, амплитудным и частотным вибратором, называется оператором. Различные способы соединения нескольких операторов, когда сигналы с выходов одних управляют работой других, называются алгоритмами синтеза. Алгоритм может включать как один, так и несколько операторов, соединенных последовательно, параллельно, последовательно-параллельно, с обратными связями и в иных сочетаниях – все это дает практически бесконечное множество возможных звуков.[6]

Благодаря простоте цифровой реализации, метод получил широкое распространение в студийной и концертной практике (типичный представитель класса синтезаторов - Yamaha DX). Однако практическое использование этого метода достаточно сложно из-за того, что большая часть звуков, получаемых с его помощью, представляет собой шумоподобные колебания, и даже небольшое изменение в настройке одного из генераторов превращает чистый тембр в шум. Однако метод дает широкий спектр возможностей по синтезу разного рода перкуSSIONных и ударных звуков, а также звуковых эффектов, недостижимых в других методах разумной сложности. [7]

3.4 Сэмплерный синтез звука

В сэмплерном (sample - выборка) методе записывается реальное звучание (сэмпл), которое затем в нужный момент воспроизводится. Для получения звуков разной высоты воспроизведение ускоряется или замедляется; при неизменной скорости выборки применяется расчет промежуточных значений отсчетов (интерполяция). Чтобы тембр звука при сдвиге высоты не менялся слишком резко, используется несколько записей звучания через, как правило, одну-две октавы. В ранних сэмплерных синтезаторах звуки записывались на магнитофон, в современных применяется цифровая запись звука.

Метод позволяет получить сколь угодно точное подобие звучания реального инструмента, однако для этого требуются достаточно большие объемы памяти. К тому же, запись звучит естественно только при тех же значениях параметров, при которых она была сделана – при попытке, например, придать ей другую амплитудную огибающую естественность значительно понижается.

Для уменьшения требуемого объема памяти применяется зацикливание сэмпла (looping). В этом случае записывается только короткое время звучания инструмента, затем в нем выделяется средняя фаза с установившимся звуком, которая при воспроизведении повторяется до тех пор, пока включена нота (нажата клавиша), а после отпускания воспроизводится концевая фаза.

На самом деле этот метод нельзя с полным правом называть синтезом, скорее методом записи и воспроизведения, однако на нём основан один из важнейших, наряду с FM-синтезом, таблично-волновой синтез. [6]

3.5 Таблично-волновой синтез звука

Таблично-волновой (wave table, WT) – разновидность сэмплерного метода, при котором записывается (оцифровывается) не все звучание

полностью, а его частные фазы - атака, начальное затухание, средняя фаза и конечное затухание, что позволяет значительно снизить объем памяти, требуемый для хранения сэмплов. Эти фазы записываются на различных частотах и при различных условиях (мягкий или резкий удар по клавише рояля, различное положение губ и языка при игре на саксофоне и т.п.), в результате чего образуется множество звучаний одного инструмента. При воспроизведении эти фазы установленным образом объединяются, что дает возможность при относительно небольшом объеме сэмплов получить достаточно широкий спектр различных звучаний инструмента, а главное – заметно усилить выразительность звучания.

Основная проблема этого метода – в сложности сопряжения различных фаз друг с другом, чтобы переходы не воспринимались на слух и звучание было непрерывным и цельным. Поэтому синтезаторы этого класса достаточно редки и дороги.

Этот же метод используется в синтезаторах звуковых карт персональных компьютеров, однако его возможности там сильно урезаны. [6]

4 Анализ MIDI аппаратуры и программного обеспечения

Хотя и существует множество различных аппаратных устройств для ввода-вывода и работы со звуком, в случае персонального компьютера следует подробнее рассмотреть звуковые карты. Их принято делить на звуковые, музыкальные и звукомзыкальные. По конструкции же все звуковые платы можно классифицировать следующим образом: основные (устанавливаемые на материнской плате компьютера и обеспечивающие ввод и вывод аудио данных) и дочерние, внешние (имеют принципиальное конструктивное отличие от основных плат - они чаще всего подключаются к специальному разъему, расположенному на основной плате). Именно последние служат чаще всего для обеспечения или расширения возможностей MIDI-синтезатора (рисунок 2).

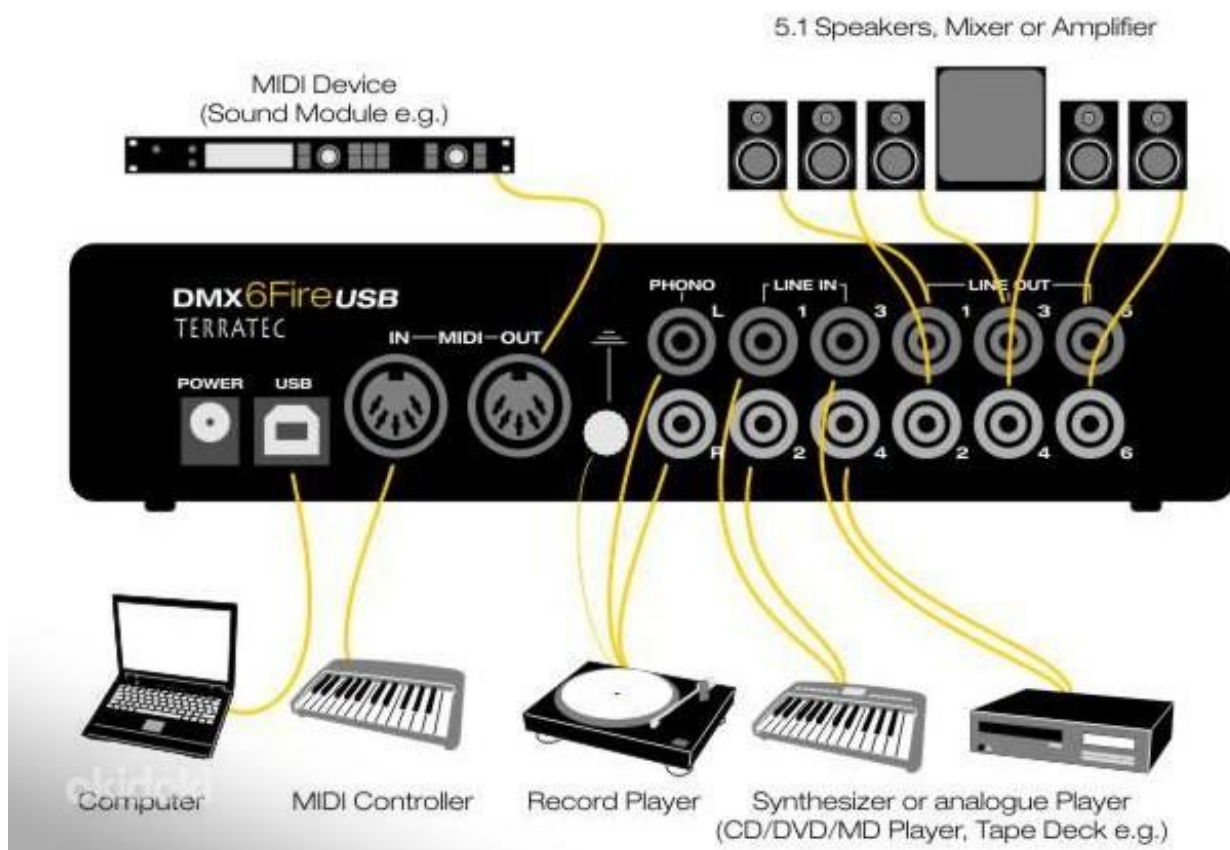


Рисунок 2 – Схема подключения устройств к профессиональной звуковой карте

Для работы с MIDI-форматом используют так называемые программы-секвенсоры (sequencer), аналогичные аппаратным MIDI-секвенсорам. Название происходит от термина sequence – последовательность, поскольку первые секвенсоры (тогда еще некомпьютерные) предназначались для записи последовательности MIDI-событий и последующего ее воспроизведения в неизменном виде, и лишь затем к этому добавились функции монтажа и редактирования.

В их функции входит запись, редактирование и воспроизведение MIDI-партитур, отображение их в различных форматах, осуществление типовых музыкальных операций как над нотами (транспонирование, квантование, сдвиг фрагмента и т.д.), так и над управляющими событиями – смены

инструментов, генерации серий значений контроллеров, имитирующих движение регуляторов и т.п.

Всегда многодорожечные — допускают формирование произведения из множества независимых партий. Большинство современных секвенсоров имеет поддержку аудиотехнологии, позволяя размещать на отдельных дорожках акустические или голосовые партии; окончательное смешивание сигналов при этом выполняется внешними аппаратными (звуковой адаптер, микшерный пульт) или программными (виртуальный синтезатор, многоканальный рекордер) средствами. [8]

Обычно профессиональные секвенсоры поддерживают три основных формата отображения:

- нотный (staff). Воссоздает классический нотный стан, принятый в музыкальной практике. Однако в связи с тем, что MIDI-формат описывает события, а не нотную запись, многие принятые в музыке обозначения недопустимы (в первую очередь это относится к лигам — некоторые секвенсоры расставляют их автоматически);
- временно-высотный (piano roll). Изображается временной график включения/выключения нот (нажатий/отпусканий);
- событийный (events). Изображается список всех MIDI-событий с указанием времени появления каждого из них.

Специализированные секвенсоры позволяют также присоединять к партитуре WAV-файлы, которые будут воспроизводиться вместе с ней в нужные моменты времени.

Наиболее известны секвенсоры Voyetra Plus Gold - для DOS и Recording Session, Cakewalk, Cubase и Logic - для Windows. Первый и два последних относятся к профессиональным, хотя Cakewalk по некоторым своим возможностям уступает Voyetra и Cubase. Cakewalk и Cubase выпускаются в нескольких версиях: Cakewalk - Apprentice, Pro и Pro Audio, Cubase - Lite, Score и Studio.

Cakewalk Pro Audio (разработчик – Twelve Tone Systems) – наиболее массовый и популярный MIDI-секвенсор с поддержкой аудиодорожек. Имеет удобный и интуитивно понятный интерфейс, множество необходимых функций редактирования и обработки. Работает с различными видами MIDI-и аудиооборудования, поддерживает частоты дискретизации до 96 кГц и разрядность оцифровки до 24 бит.

Мощный и весьма сложный в освоении секвенсор Cubase VST/24 от Steinberg имеет специфический и далеко не всегда понятный интерфейс, за что ценится в основном профессионалами с большим опытом работы над серьезными музыкальными проектами. Аббревиатура VST происходит от Virtual Studio Technology — технология виртуальной студии. Cubase VST предоставляет все функции, необходимые для организации настольной студии звукозаписи. Поддерживается до 96 звуковых и неограниченное количество MIDI-дорожек. Поддерживаются звуковые форматы с частотой дискретизации до 96 кГц и разрядностью до 24 бит. [8]

Audio Compositor - секвенсор, объединенный с эмулятором синтезатора. Позволяет загрузить сэмплы инструментов в различных форматах и составить из них композицию, записываемую потом в WAV-файл.

MIDIMon, MIDI-OX, HUBI's Loopback и HUBI's MIDI Tools - средства для организации виртуальных MIDI-кабелей под Windows, отслеживания приходящих MIDI-сообщений, приема/передачи команд. [7]

5 Технические требования

Настоящие технические требования распространяются на разработку проигрывателя и синтезатора MIDI-композиций.

Программа должна обеспечивать возможность синтеза мелодий посредством отправки MIDI-сообщений и воспроизведения уже существующих композиций и контроля над ним.

Кроме того, требуется наличие способа формирования плейлиста и обзора файлов компьютера.

5.1 Назначение разработки

Программный продукт предназначен для проигрывания медиа контента в формате MIDI. Программа представляет собой графический интерфейс пользователя (GUI) для прослушивания аудиофайлов и управления плейлистом, а также генерации мелодий в соответствии с предъявляемыми требованиями.

5.2 Требования к программе или программному изделию

5.2.1 Требования к функциональным характеристикам

Программа должна объединять в себе две следующие части: управляемый проигрыватель композиций формата *.mid и эмуляцию MIDI-клавиатуры, позволяющую воссоздавать звучание тех или иных нот.

Проигрыватель должен обеспечивать выполнение следующих функций:

- выбор одного или нескольких файлов формата *.mid из библиотеки;
- добавление выбранных композиций в плейлист;
- удаление композиций из плейлиста;
- автоматическое начало проигрывания композиции при выборе её в сформированном плейлисте;
- пауза/продолжение/полная остановка воспроизведения текущей композиции;
- отображение длительности композиции и её перемотка;

Клавиатура же должна реагировать как на клик мыши на экранном отображении, так и на нажатие кнопки, привязанной к каждой клавише и воссоздавать звучание конкретной ноты фиксированной длительности.

5.2.2 Требования к составу и параметрам технических средств

Программа должна работать на IBM-PC-совместимых персональных компьютерах.

Минимальная конфигурация:

- процессор: 1 ГГц;
- свободное место на жёстком диске: 120 Мбайт;
- оперативная память:
 - для 32-разрядной операционной системы – 1 ГБ;
 - для 64-разрядной операционной системы – 2 ГБ.

6 Проектирование программы

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.

Диаграмма классов UML – это граф, узлами которого являются элементы статической структуры проекта (классы, интерфейсы), а дугами – отношения между узлами (ассоциации, наследование, зависимости).

На рисунке 3 представлена диаграмма классов для проигрывателя и синтезатора MIDI-композиций. Класс Audio содержит основную информацию о каждой композиции, методы класса Проигрыватель управляют воспроизведением. Класс формы обеспечивает взаимодействие программы с пользователем, а также позволяет синтезировать мелодии.

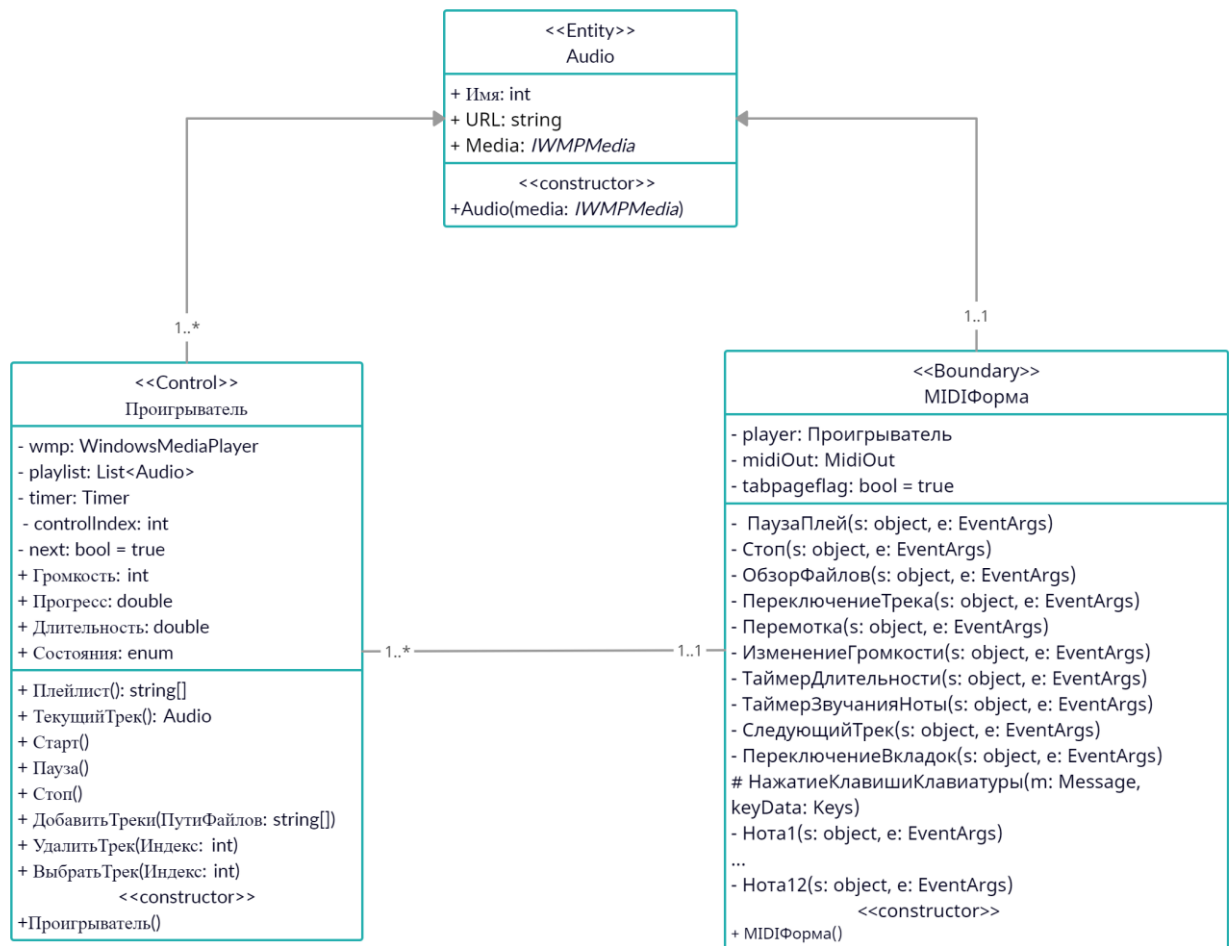


Рисунок 3 – Диаграмма классов

Для того, чтобы реализация программы стала возможной требуется подключить две библиотеки:

- WMPLib для проигрывателя
- NAudio для синтеза мелодий

Для `wmplib.dll` достаточно добавить ссылку на Windows Media Player в проект и подключить WMPLib с помощью директивы `using` к классу, использующему эту библиотеку.

NAudio же – сторонняя библиотека для работы со звуком на платформе .NET, является «оберткой» для системных API, упрощающей разработку. Для её подключения стоит воспользоваться консолью диспетчера пакетов (рисунок 4) NuGet, механизма совместного использования кода, поддерживаемого Майкрософт.

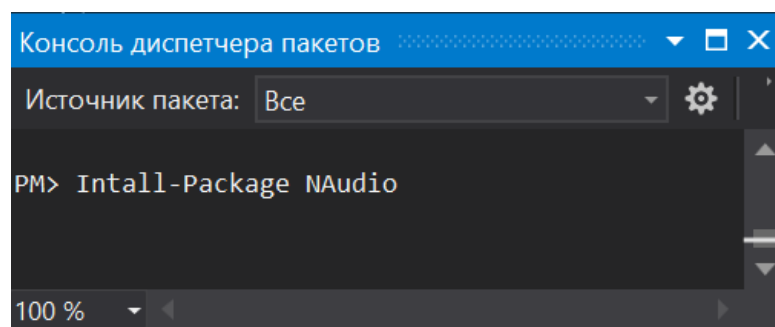


Рисунок 4 – Команда для подключения библиотеки NAudio в консоли диспетчера пакетов

7 Реализация программного продукта

7.1 Выбор инструментальной среды разработки

Для создания приложения выбрана интегрированной среде разработки MS Visual Studio.NET 2019 с применением объектно-ориентированного императивного языка программирования C#.

7.2 Реализация пользовательского интерфейса

Для реализации GUI создан проект Windows Forms Application. Функции пользовательского интерфейса реализованы в одном из основных классов программы MIDIForm. Он предусматривает наличие двух вкладок элемента управления TabControl. На первой из них размещены все элементы плеера, вторая отвечает за представление эмулятора MIDI-клавиатуры.

Интерфейс плеера позволяет составлять и изменять плейлист, контролировать воспроизведение, перематывать и переключать треки,

увеличивать и уменьшать громкость с помощью соответствующих кнопок и элементов `ListBox` и `TrackBar`.

Интерфейс клавиатуры реализован с помощью двенадцати кнопок, каждая из которых воссоздает при нажатии звучание нот третьей (малой) октавы.

7.2.1 Описание пользовательского интерфейса

При запуске программы загружается стартовая форма, изначально отображается вкладка проигрывателя, что можно увидеть на рисунке 5.

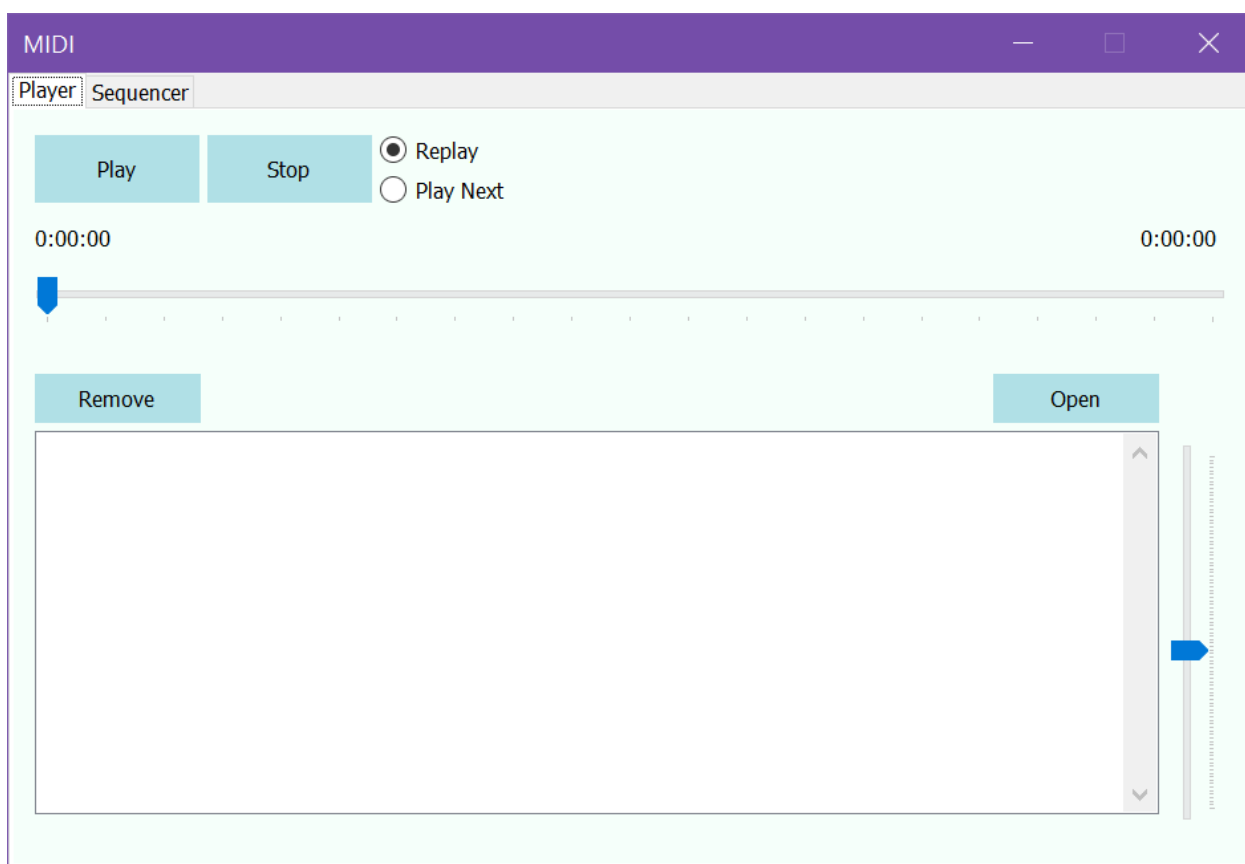


Рисунок 5 – Начальное состояние приложения

Громкость устанавливается на значение 50 и в дальнейшем может изменяться от 0 до 100.

Кнопка Play управляет воспроизведением и при проигрывании трека заменяется на Pause. Кнопка Stop полностью останавливает текущий трек.

Кнопки Replay/Play Next позволяет выбрать какой трек из плейлиста включиться при завершении текущего. Replay перезапускает ту же самую композицию, при выборе Play Next будет воспроизведена следующая за текущей или, если текущий трек был последним, первый трек из плейлиста.

Перематывать трек можно с помощью горизонтального ползунка, над ним же демонстрируются длительность трека (справа) и текущий прогресс (слева).

Кнопка Open открывает обозреватель файлов (рисунок 6) и добавляет файлы в плейлист, отображаемый ниже (при реализации устанавливается фильтр на формат *.mid).

Кнопка Remove удаляет из плейлиста выбранный трек.

На рисунке 7 можно видеть плеер в активном состоянии, с композициями, добавленными в плейлист, измененной громкостью

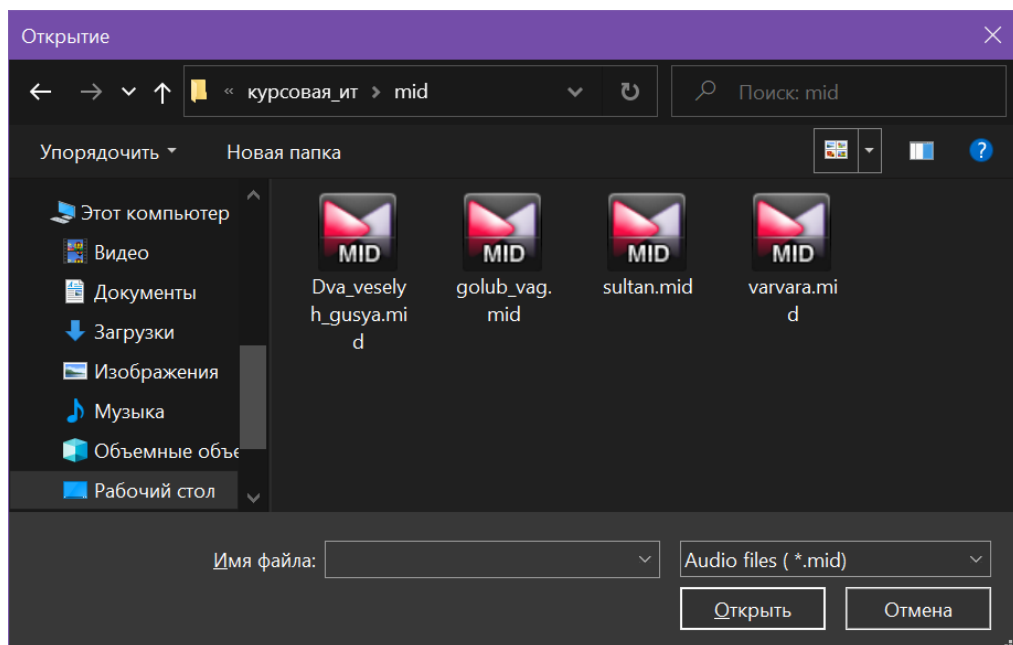


Рисунок 6 – Обозреватель файлов

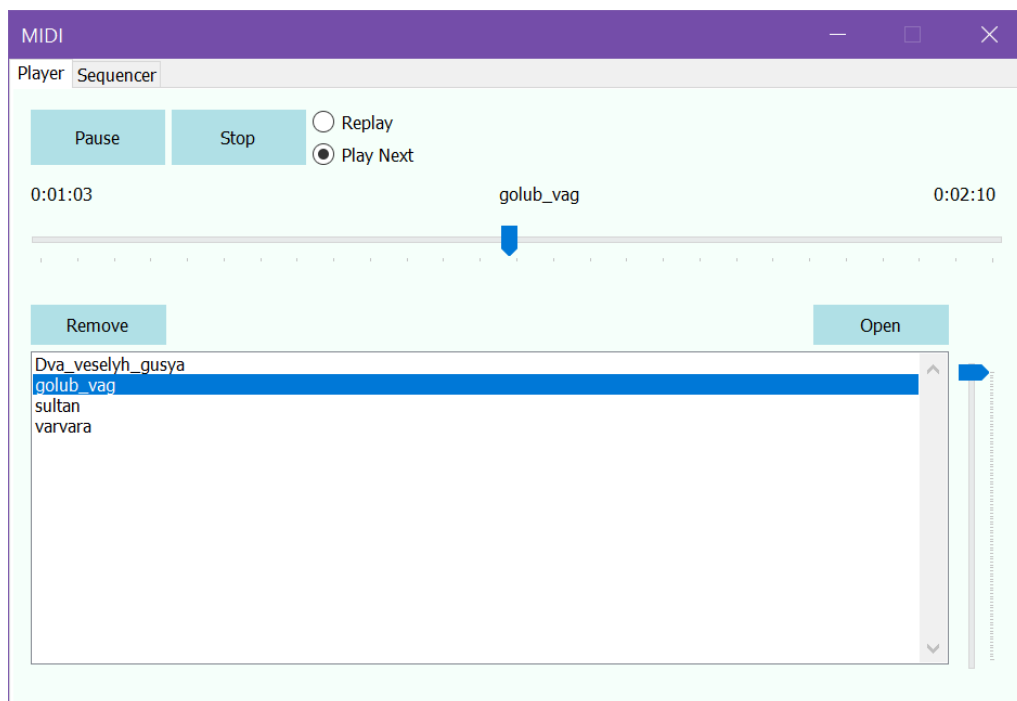


Рисунок 7 – Работающий плеер

При переключении на вторую вкладку пользователю становится доступна MIDI-клавиатура. Каждая из 12-ти кнопок помечена в соответствии с нотой, которую она воспроизводит (в третьей октаве, включая альтерацию) и может быть вызвана как мышью, так и с помощью соответствующих кнопок клавиатуры устройства.

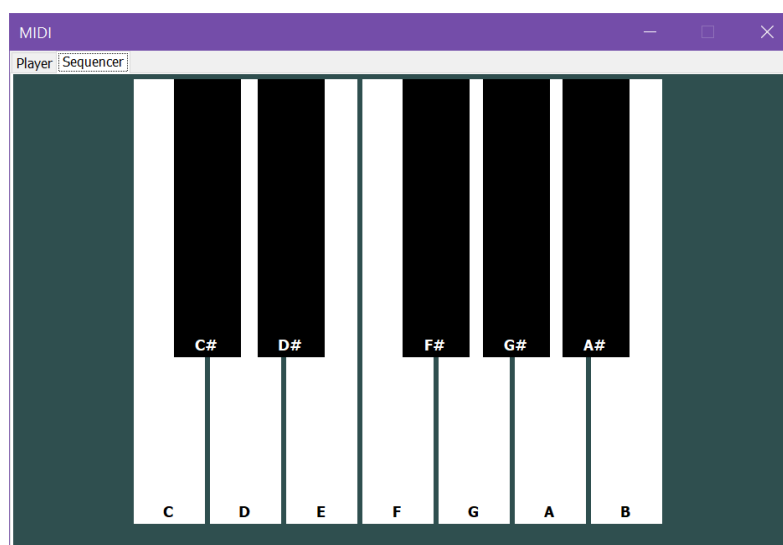


Рисунок 8 – MIDI-клавиатура

7.3 Реализация функционала проигрывателя

Представляющий трек класс `Audio` содержит в себе доступные для чтения и закрытые для чтения свойства, такие как имя файла, путь к нему, сама же композиция содержится в свойстве `Media` (интерфейс `IWMPMedia`).

Плеер реализованный в классе `Player`, основан на экземпляре класса `WindowsMediaPlayer`, является дополненной его версией.

Перечисление состояний `States`, реакция на изменение которых прописана в конструкторе класса, позволяет точно контролировать воспроизведение и изменять требуемым образом элементы управления в пользовательском интерфейсе. Этой же цели служат события `AudioSelected` и `PlayStateChange`. Также конструктор содержит в себе объявление закрытого плейлиста – списка типа класса `Audio` – и обработчик тиков таймера (интервал 50 мс).

Метод `Playlist` формирует массив названий треков для использования его в коде формы, а метод `ControlAudio` – возвращает текущий трек с помощью.

Методы `Play`, `Stop`, `Pause` являются оболочкой для аналогичных методов `WindowsMediaPlayer` и одновременно с этим осуществляют управления запуском и остановкой таймера.

Функции `AddAudio` и `RemoveAudio` отвечают за добавление композиций в плейлист и удаление их из него соответственно.

Важнейший из методов класса, `SelectAudio` вызывается при выборе трека пользователем и автоматически начинает его воспроизведение, а также фиксирует его индекс в плейлисте и запускает таймер.

7.4 Реализация синтеза MIDI-композиций

Данная возможность реализована в классе самой формы `MIDIForm`.

Для управления отдельными нотами требуется подключить к классу пространство `NAudio.Midi` и создать экземпляр класса `MidiOut`, который и будет направлять данные на вывод.

Воспроизведение нот реализовано в обработчиках нажатия кнопки клавиатуры на пользовательском интерфейсе. Метод класса `MidiOut.Send` отправляет сообщение `MidiMessage.StartNote`, содержащее в себе ID ноты, силу её нажатия, а также один из шестнадцати каналов по которому она передаётся. Но так как нота сама по себе будет звучать достаточно долго, постепенно затухая, требуется также предусмотреть её остановку. Для этого в обработчике тиков таймера с интервалом 150 мс `MidiOut.Send` направляет уже сообщение об остановке. Пример такого воспроизведения одной из нот показан в листинге 1.

Листинг 1 – код ноты Си третьей октавы

```
        midiOut.Send(MidiMessage.StartNote(59, 127,
1) .RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(59, 0, 1) .RawData);
        timer2.Stop();
```

Для удобства взаимодействия с кнопками они также привязаны к кнопкам клавиатуры ПК (листинг 2). Чтобы интерфейс реагировал на нажатие, следует установить при инициализации формы значение свойства `KeyPreview` (указывает, получит ли форма события клавиш перед передачей событий элементу управления, на который установлен фокус) как `true`.

Листинг 2 – код метода `ProcessCmdKey`

```
        protected override bool ProcessCmdKey(ref Message msg,
Keys keyData)
        {
            timer2.Start();
```

```

switch (keyData)
{
    case Keys.X:
        button1.PerformClick();
        break;
    case Keys.C: button2.PerformClick();
        break;
    case Keys.V: button3.PerformClick();
        break;
    case Keys.B: button4.PerformClick();
        break;
    case Keys.N: button5.PerformClick();
        break;
    case Keys.M: button6.PerformClick();
        break;
    case Keys.Oemcomma:button7.PerformClick();
        break;
    case Keys.D: button8.PerformClick();
        break;
    case Keys.F: button9.PerformClick();
        break;
    case Keys.H: button10.PerformClick();
        break;
    case Keys.J: button11.PerformClick();
        break;
    case Keys.K: button12.PerformClick();
        break;
    default: break;
}
return base.ProcessCmdKey(ref msg, keyData);
}

```

Заключение

Итогом данной курсовой работы является реализованное на языке программирования C# приложение, позволяющее как воспроизводить существующие композиции в MIDI-формате, так и синтезировать новые. Был проведён анализ существующих программ и аппаратных устройств, служащих тем же целям, изучены особенности протокола MIDI.

Приложение может использоваться на персональных компьютерах для прослушивания и создания мелодий.

В качестве перспектив усовершенствования можно отметить расширение диапазона звучания клавиатуры, возможность смены инструментов, а также добавление новых функций проигрывателя, таких как, например, эквалайзер или формирование различных плейлистов.

Список использованных источников

1. Кинтцель Т. Руководство программиста по работе со звуком = A Programmer's Guide to Sound: Пер. с англ. – М.: ДМК Пресс. – 432 с.
2. Подробнее о формате MIDI [Электронный ресурс] – Url: <https://audacity.ru/p9aa1.html> (Дата обращения 07.12.2020)
3. Интерфейс MIDI. Преимущества и недостатки технологии MIDI [Электронный ресурс] – Url: https://vuzlit.ru/2231768/interfeys_midi_preimuschestva_nedostatki_tehnologii_midi (Дата обращения 09.12.2020)
4. Цифровой звук и MIDI [Электронный ресурс] – Url: https://studopedia.su/10_66773_tsifrovoy-zvuk-i-MIDI.html (Дата обращения 09.12.2020)
5. Стандарты MIDI: GM, GS, XG, GM2 [Электронный ресурс] – Url: <https://www.ixbt.com/proaudio/midi-standards.shtml> (Дата обращения 09.12.2020)
6. Романова В.А. Цифровые синтезаторы и основные модели синтеза звука// Вестник Московского государственного университета печати. – 2015. – с.12-16
7. Музыченко Е.В. Часто задаваемые вопросы по электронному созданию и обработке звука [Электронный ресурс] – Url: <http://websound.ru/articles/theory/soundfaq.htm#14> (Дата обращения 10.12.2020)
8. Обзор программ для работы со звуком и музыкой [Электронный ресурс] – Url: <https://compress.ru/article.aspx?id=12202&part=index11ext1> (дата обращения 10.12.2020)

Приложение А

Листинг программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using WMPLib;
using System.IO;
using System.Windows.Forms;
using NAudio.Midi;

namespace MIDI
{
    public class Audio
    {
        public string Name { get; private set; }
        public string URL { get; private set; }
        public IWMPMedia Media { get; private set; }
        public Audio(IWMPMedia media)
        {
            Media = media;
            Name =
Path.GetFileNameWithoutExtension(media.sourceURL);
            URL = media.sourceURL;
        }
    }
    public class Player
    {
        private WindowsMediaPlayer wmp;
        private List<Audio> playlist;
        private Timer timer;
        private int controlIndex;
        private bool next = false;

        public int Volume
        {
            get { return wmp.settings.volume; }
            set { wmp.settings.volume = value; }
        }
        public double Progress
        {
            get { return wmp.controls.currentPosition; }
            set { wmp.controls.currentPosition = value; }
        }
        public double Duration
        {
            get { return wmp.currentMedia.duration; }
        }
        public bool Next { get { return next; } set { next =
value; } }
    }
}
```

```

public enum States
{
    Playing,
    Paused,
    Stopped,
    Ended,
    Undefined
}
public States PlayState
{
    get
    {
        switch (wmp.playState)
        {
            case WMPPlayState.wmppsPlaying:
                return States.Playing;
            case WMPPlayState.wmppsPaused:
                return States.Paused;
            case WMPPlayState.wmppsStopped:
                return States.Stopped;
            case WMPPlayState.wmppsMediaEnded:
                return States.Ended;
            default: return States.Undefined;
        }
    }
}
public Player()
{
    wmp = new WindowsMediaPlayer();
    playlist = new List<Audio>();

    wmp.PlayStateChange += (e) =>
    {
        if (PlayState == States.Undefined)
            return;
        PlayStateChanged?.Invoke(this, PlayState);
    };
    timer = new Timer() { Interval = 50 };
    timer.Tick += (s, e) =>
    {
        if (PlayState == States.Stopped || PlayState ==
States.Ended)
            ((Timer)s).Stop();
        if ((PlayState == States.Stopped || PlayState ==
States.Ended) && (next == false))
            SelectAudio(controlIndex);
        else
            if ((PlayState == States.Stopped || PlayState ==
States.Ended) && (next == true))
            {
                if (controlIndex >= playlist.Count - 1)
SelectAudio(0);
                else

```



```

        SelectAudio(controlIndex + 1);
    }
};

}
public string[] Playlist() { return playlist.Select(n =>
n.Name).ToArray(); }
public Audio ControlAudio => playlist[controlIndex];
public void Play()
{
    timer.Start();
    wmp.controls.play();
}
public void Pause()
{
    timer.Stop();
    wmp.controls.pause();
}
public void Stop()
{
    timer.Stop();
    wmp.controls.stop();
}
public void AddAudio(string[] paths)
{
    foreach (var p in paths) playlist.Add(new
Audio(wmp.newMedia(p)));
}
public void RemoveAudio(int index)
{
    if (controlIndex == playlist.Count - 1)
        controlIndex = 0;
    playlist.RemoveAt(index);
}
public void SelectAudio(int index)
{
    controlIndex = index;
    wmp.currentMedia = ControlAudio.Media;
    wmp.controls.play();
    AudioSelected?.Invoke(this, ControlAudio);
    timer.Start();
}
public event Action<object, Audio> AudioSelected;
public event Action<object, States> PlayStateChanged;
}

public partial class MIDIForm : Form
{
    Player player;
    MidiOut midiOut;
    public MIDIForm()
    {
        InitializeComponent();
        player = new Player();
    }
}

```

```

        player.PlayStateChanged += (s, e) =>
playpauseButton.Text = e == Player.States.Playing ? "Pause" :
"Play";
        player.AudioSelected += (s, e) =>
        {
            listBox1.SelectedItem = e.Name;
            label1.Text = e.Name;
        };
        KeyPreview = true;
    }
    private void playpauseButton_Click(object sender,
EventArgs e)
    {
        if (Player.States.Playing == player.PlayState)
player.Pause();
        else player.Play();
    }
    private void stopButton_Click(object sender, EventArgs
e)
    {
        player.Stop();
    }
    private void openButton_Click(object sender, EventArgs
e)
    {
        var open = new OpenFileDialog() { Multiselect =
true, Filter = "Audio files| *.mid" };
        if (open.ShowDialog() == DialogResult.OK)
        {
            player.AddAudio(open.FileNames);
            listBox1.Items.Clear();
            listBox1.Items.AddRange(player.Playlist());
        }
    }
    private void listBox1_SelectedIndexChanged(object
sender, EventArgs e)
    {
        if (((ListBox)sender).SelectedItem == null) return;
        player.SelectAudio(((ListBox)sender).SelectedIndex);
        timer1.Enabled = true;
    }
    private void timeBar_Scroll(object sender, EventArgs e)
    {
        player.Progress = ((TrackBar)sender).Value;
    }
    private void volumeBar_Scroll(object sender, EventArgs
e)
    {
        player.Volume = ((TrackBar)sender).Value;
    }

    private void timer1_Tick(object sender, EventArgs e)
    {

```

```

timeBar.Maximum = Convert.ToInt32(player.Duration);
timeBar.Value = Convert.ToInt32(player.Progress);
if (player.ControlAudio != null)
{
    int s = (int)player.Duration;
    int h = s / 3600;
    int m = (s - (h * 3600)) / 60;
    s = s - (h * 3600 + m * 60);
    label3.Text =
String.Format("{0:D}:{1:D2}:{2:D2}", h, m, s);

    s = (int)player.Progress;
    h = s / 3600;
    m = (s - (h * 3600)) / 60;
    s = s - (h * 3600 + m * 60);
    label2.Text =
String.Format("{0:D}:{1:D2}:{2:D2}", h, m, s);
}
else
{
    label3.Text = "0:00:00";
    label2.Text = "0:00:00";
}
}
private void radioButton1_CheckedChanged(object sender,
EventArgs e)
{
    if (radioButton1.Checked) player.Next = true;
    else player.Next = false;
}
private void removeButton_Click(object sender, EventArgs
e)
{
    player.Stop();
    player.RemoveAudio(listBox1.SelectedIndex);
    listBox1.Items.RemoveAt(listBox1.SelectedIndex);
}
protected override bool ProcessCmdKey(ref Message msg,
Keys keyData)
{
    timer2.Start();
    switch (keyData)
    {
        case Keys.X:
            button1.PerformClick();
            break;
        case Keys.C: button2.PerformClick();
            break;
        case Keys.V: button3.PerformClick();
            break;
        case Keys.B: button4.PerformClick();
            break;
        case Keys.N: button5.PerformClick();

```

```

        break;
    case Keys.M: button6.PerformClick();
        break;
    case Keys.Oemcomma: button7.PerformClick();
        break;
    case Keys.D: button8.PerformClick();
        break;
    case Keys.F: button9.PerformClick();
        break;
    case Keys.H: button10.PerformClick();
        break;
    case Keys.J: button11.PerformClick();
        break;
    case Keys.K: button12.PerformClick();
        break;
    default: break;
}
return base.ProcessCmdKey(ref msg, keyData);
}

#region Notes

private void button1_Click(object sender, EventArgs e)
{
    midiOut.Send(MidiMessage.StartNote(48, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
    timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(48, 0, 1).RawData);
    timer2.Stop();
}

private void button2_Click(object sender, EventArgs e)
{
    midiOut.Send(MidiMessage.StartNote(50, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
    timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(50, 0, 1).RawData);
    timer2.Stop();
}

private void button3_Click(object sender, EventArgs e)
{
    midiOut.Send(MidiMessage.StartNote(52, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
    timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(52, 0, 1).RawData);
    timer2.Stop();
}

private void button4_Click(object sender, EventArgs e)
{
    midiOut.Send(MidiMessage.StartNote(53, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;

```

```

        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(53, 0, 1).RawData);
        timer2.Stop();
    }

    private void button5_Click(object sender, EventArgs e)
    {
        midiOut.Send(MidiMessage.StartNote(55, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(55, 0, 1).RawData);
        timer2.Stop();
    }

    private void button6_Click(object sender, EventArgs e)
    {
        midiOut.Send(MidiMessage.StartNote(57, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(57, 0, 1).RawData);
        timer2.Stop();
    }

    private void button7_Click(object sender, EventArgs e)
    {
        midiOut.Send(MidiMessage.StartNote(59, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(59, 0, 1).RawData);
        timer2.Stop();
    }

    private void button8_Click(object sender, EventArgs e)
    {
        midiOut.Send(MidiMessage.StartNote(49, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(49, 0, 1).RawData);
        timer2.Stop();
    }

    private void button9_Click(object sender, EventArgs e)
    {
        midiOut.Send(MidiMessage.StartNote(51, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(51, 0, 1).RawData);
        timer2.Stop();
    }

    private void button10_Click(object sender, EventArgs e)
    {

```

```

        midiOut.Send(MidiMessage.StartNote(54, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(54, 0, 1).RawData);
        timer2.Stop();
    }
    private void button11_Click(object sender, EventArgs e)
    {
        midiOut.Send(MidiMessage.StartNote(56, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(56, 0, 1).RawData);
        timer2.Stop();
    }
    private void button12_Click(object sender, EventArgs e)
    {
        midiOut.Send(MidiMessage.StartNote(58, 127,
1).RawData); //id звука, сила нажатия (0-127), номер канала;
        timer2.Tick += (s, ev) =>
midiOut.Send(MidiMessage.StopNote(58, 0, 1).RawData);
        timer2.Stop();
    }
    #endregion Note

    private bool tabpageflag = true;
    private void tabControl1_Selected(object sender,
TabControlEventArgs e)
    {
        if (tabpageflag)
        {
            midiOut = new MidiOut(0);
            tabpageflag = !tabpageflag;
        }
        else
        {
            midiOut.Dispose();
            tabpageflag = !tabpageflag;
        }
    }
}

```

Приложение Б

Проверка на плагиат

Отчет о проверке на заимствования №1



Автор: Наталочка Анастасия Денисовна asnatalochka@gmail.com / ID: 7155147
Проверяющий: Наталочка Анастасия Денисовна (asnatalochka@gmail.com / ID: 7155147)
Отчет предоставлен сервисом «Антиплагиат»- <http://users.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 33
Начало загрузки: 11.12.2020 21:06:18
Длительность загрузки: 00:00:01
Имя исходного файла: ИТ_КР.pdf
Название документа: ИТ_КР
Размер текста: 46 кБ
Символов в тексте: 46933
Слов в тексте: 5615
Число предложений: 444

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 11.12.2020 21:06:20
Длительность проверки: 00:00:07
Комментарии: не указано
Модули поиска: Модуль поиска Интернет



ЗАИМСТВОВАНИЯ
23,44%

САМОЦИТИРОВАНИЯ
0%

ЦИТИРОВАНИЯ
0%

ОРИГИНАЛЬНОСТЬ
76,56%