**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное автономное образовательное   
учреждение высшего профессионального образования**

**«Южный федеральный университет»**

**Факультет математики, механики и компьютерных наук**

**Кафедра информатики и вычислительного эксперимента**

**Чубарьян Артем Суренович**

**ИНКРЕМЕНТНАЯ ГЕНЕРАЦИЯ МУЗЫКАЛЬНЫХ ПРОИЗВЕДЕНИЙ НА ОСНОВЕ ДИНАМИЧЕСКИХ ПАТТЕРНОВ**

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ**

**по направлению 010400— Прикладная математика и информатика**

**Научный руководитель –**

**ст. преп. Брагилевский Виталий Николаевич**

**Рецензент –**

**доцент. к.ф.-м.н. Абрамян Михаил Эдуардович**

**Ростов-на-Дону – 2013**

Оглавление

[Введение 3](#_Toc357338158)

[1. Алгоритмическая музыка 3](#_Toc357338159)

[1.1. Краткая история алгоритмической музыки 3](#_Toc357338160)

[1.2. Обзор существующих библиотек для программирования музыки 6](#_Toc357338161)

[1.3. Постановка задачи 6](#_Toc357338162)

[2. Математическая модель музыкальной формы 7](#_Toc357338163)

[2.1. Основы музыкальной формы: фраза, тема, контр тема, разработка 7](#_Toc357338165)

[2.2. Упрощенная модель музыкального произведения 10](#_Toc357338166)

[2.3. Конструирование мелодий с помощью паттернов 11](#_Toc357338167)

[2.3.1. Формализация отношений между двумя соседними нотами 11](#_Toc357338171)

[2.3.2. Построение мотива с помощью паттернов 18](#_Toc357338178)

[2.4. Алгоритм гармонизации мелодии 25](#_Toc357338179)

[2.5. Генерация второстепенных голосов и аккомпанемента 30](#_Toc357338180)

[3. Реализация генератора музыки 34](#_Toc357338181)

[3.1. Архитектура генератора музыки 34](#_Toc357338185)

[3.2. Основные программные элементы 36](#_Toc357338186)

[3.3. Программные модули 40](#_Toc357338187)

[3.3.1. Основной управляющий модуль 40](#_Toc357338191)

[3.3.2. Модуль генерации 40](#_Toc357338199)

[3.3.2.1. Модуль генерации главной мелодии 44](#_Toc357338200)

[3.3.2.2. Модуль гармонизации 46](#_Toc357338201)

[3.3.2.3. Модуль генерации второстепенных голосов и аккомпанемента 48](#_Toc357338202)

[3.3.3. Модуль воспроизведения 49](#_Toc357338203)

[3.3.4. Модуль взаимодействия с пользователем 52](#_Toc357338204)

[Заключение 53](#_Toc357338205)

[Литература 54](#_Toc357338206)

# Введение

Идея алгоритмизации процесса написания музыки зародилась еще в средневековье в тот момент, когда в музыке начали сформировываться правила композиции и гармонии. Первые серьезные попытки заняться алгоритмической музыкой относятся ко времени возникновения компьютеров (1950-е годы), и принадлежат они, в первую очередь, композиторам-авангардистам. Благодаря их вкладу, а также исследованиям в области теории музыки, композиции и гармонии, на сегодняшний день мы имеем целые исследовательские группы, работающие над алгоритмизацией музыки. Существуют экспериментальные разработки, генерирующие музыку в стиле известных композиторов, джазовые вариации, электронную музыку и многое другое.

В рамках данной работы будет рассмотрен процесс создания итеративного генератора музыки, работающего в реальном времени, начиная с построения математической модели музыкального произведения, и заканчивая непосредственно реализацией программы.

# Алгоритмическая музыка

## Краткая история алгоритмической музыки

Алгоритмической музыкой называют музыку, с помощью математических моделей, правил и алгоритмов.

Алгоритмы в музыке появились еще в античности. Так, Пифагор, формулируя акустические законы музыки, руководствовался строгими математическими правилами. В средние века итальянский теоретик музыки Гвидо д’Ареццо разработал метод привязки текста к нотам, и таким образом мог по тексту составлять мелодии.

Наиболее известным примером алгоритмической музыки является игра в «музыкальные кости», разработанная Моцартом и его современниками. Она заключалась в объединении уже написанных музыкальных фрагментов, выбираемых путем бросания костей (Рис. 1).

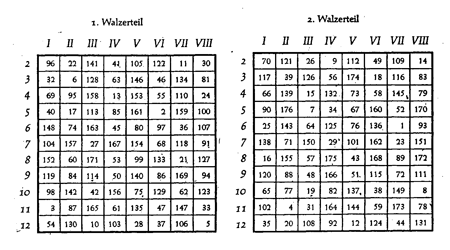


Рис. 1 Таблица тактов для игры в "Музыкальные кости".

Настоящее развитие алгоритмическая музыка получила только с появлением первых компьютеров. Главной персоной в этой области был греческий архитектор и композитор Янис Ксенакис, который ввел в технику музыкальной композиции методы теории вероятностей. Он использовал стохастику, теорию игр и марковские цепи в качестве формальных процессов композиции (Рис. 2). Случайные процессы и математические принципы построения произведения начали широко применяться музыкантами и композиторами той эпохи, среди которых были Л. Хиллер, Р. Бейкер, Д. Тенни, Р. Зарипов и многие другие.

На сегодняшний день масса разнообразных программ и исследовательских проектов по алгоритмической музыке. Среди них хочется выделить работу профессора Дэвида Коупа EMI (Experiments in Musical Intelligence), способную не только генерировать произведения в определенном жанре, но и с высокой точностью подражать стилю конкретного композитора.

Существуют также и предназначенные для обычных пользователей программы, позволяющие быстро и легко генерировать музыку, такие как Band-in-a-Box и Impo-Visor

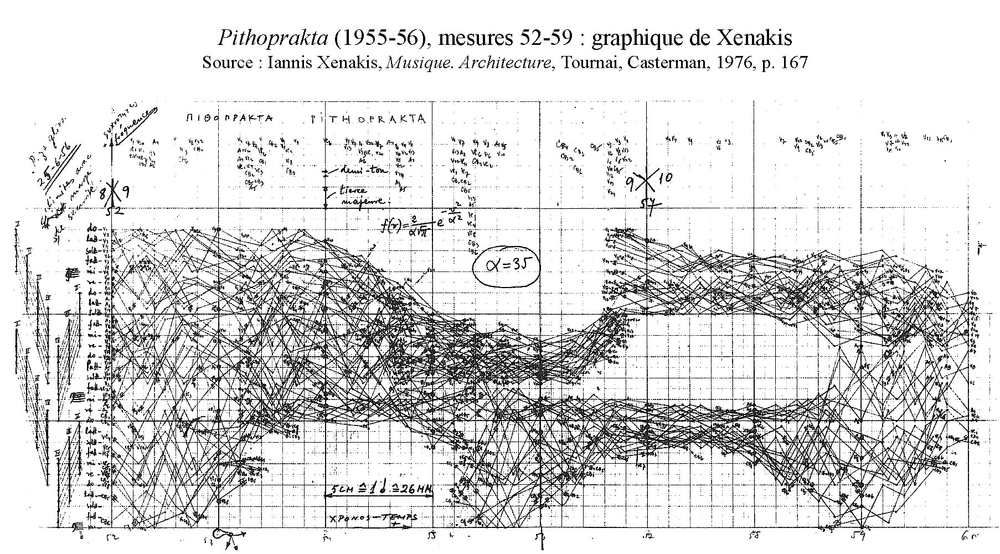


Рис. 2 График произведения "Pithoprakta" Я. Ксенакиса.

## Обзор существующих библиотек для программирования музыки

Число существующих библиотек, предназначенных для программирования музыки, сравнительно невелико. Среди них стоит выделить два наиболее удачных проекта: CFugue и jMusic. Первый из них является библиотекой для C/C++, а второй – библиотекой для Java. Основная их цель – построить мост между классическими понятиями о нотах, тактах, ритмах и мелодиях и некими абстракциями, которым удобно оперировать программистам. Благодаря использованию таких библиотек становится возможным выполнение более сложных задач: от гармонизации мелодий, поиска закономерностей в наборе музыкальных произведений и формализации существующих методик композиции до генерации музыки с применением возможностей искусственного интеллекта.

Существуют также платформы, предназначенные скорее для музыкантов, чем для разработчиков, например SoundCipher или Impromptu, которые предоставляют удобный доступ к простым музыкальным объектам, и позволяют быстро описывать простые музыкальные алгоритмы, но однако неприменимы для задач более широкого спектра.

## Постановка задачи

В рамках данной работы будут рассмотрены следующие задачи:

* Формализация структуры музыкального произведения;
* Построение алгоритма генерации мелодий, второстепенных голосов и аккомпанемента;
* Построение алгоритма гармонизации мелодии;
* Реализация вышеперечисленных алгоритмов в виде программы-генератора музыки.

# Математическая модель музыкальной формы



## Основы музыкальной формы: фраза, тема, контр тема, разработка

Прежде чем приступать к описанию процесса генерации музыки, целесообразно будет сформулировать основные понятия и термины классической музыки, следуя [1, 2, 3, 4], которые будут использоваться в дальнейшем.

Музыкальной формой называется строение музыкального произведения. Форма каждого музыкального произведения всегда индивидуальна, присуща именно ему и неповторима. Но в то же время число законов и правил образования формы – относительно ограничено, и, благодаря этому, множество сочинений имеет общие признаки в строении. Это дает возможность вывести типы форм или общие композиционные схемы, получившие значительное распространение вследствие своей гибкости и целесообразности, при большем или меньшем соответствии их общеэстетическому закону единства в разнообразии.

Основное значение для образования музыкальной формы имеют мелодия, гармония и ритм в их взаимодействии.

В современной профессиональной музыке организация звуковой речи совершается путем соединения первичных мотивов во фразы, предложения, периоды и целые разделы. Мотив – наименьшая конструктивная и выразительная единица формы; это звук или группа звуков, объединенная вокруг сильной доли.

Последовательность из двух (в особых случаях трех) мотивов, объединенных на основе метрической закономерности, называется мотивной группой или фразой. В свою очередь, фразы объединяются в предложения - наименьшую форму изложения темы (или ее части), которая в метрическом отношении исходит из четырех или восьми тактов. В тематическом отношении они представляют собой сцепление мотивов, число которых определяется числом тактов, а в гармоническом – являются взаимодействием тонических и нетонических элементов (при доминировании первых), ограниченным каденцией. Предложение понимается как форма относительно законченной музыкальной мысли, в гармоническом отношении завершающейся кадансом, который знаменует либо вопрос, либо утверждение.

Совокупность нескольких предложений образует период – вполне законченную музыкальную мысль, выраженную путем логического сочетания двух, трех, а иногда и более предложений – вопросительных или утвердительных.

Под разделом уже имеется в виду любое по величине построение, отличающееся определенной функцией, в частности их можно подразделять на экспозиционные и подсобные. В экспозиционных построениях излагается основное содержание, основные сюжетные образы; подсобные построения играют роль вступлений, заключений, разработок, связок между основными. Только из экспозиционно-тематических разделов строятся так называемые простые формы. Сопоставление двух периодов приводит к возникновению простой двухчастной формы.

Простая трехчастная форма составляется путем сопоставления трех периодов, из которых первый представляет собой повторение первого периода (реприза).

Сложные формы образуются: 1) путем сопоставления простых форм (включая период); 2) путем использования в них подсобных разделов – связок, вступлений, заключений, разработок. Сюда относится сложная трехчастная форма, форма рондо, сонатная форма, смешанные формы.

На Рис. 3 изображена схема музыкального произведения, с использованием введенных ранее понятий.

Гармонией называется объединение звуков в созвучия и связная последовательность таких созвучий. Под созвучиями будем понимать одновременное сочетание нескольких звуков. Созвучие из трех, четырех или пяти звуков, различных по высоте и по названиям, представляют собой аккорд, если они расположены по терциям (с точностью до октавных перемещений).

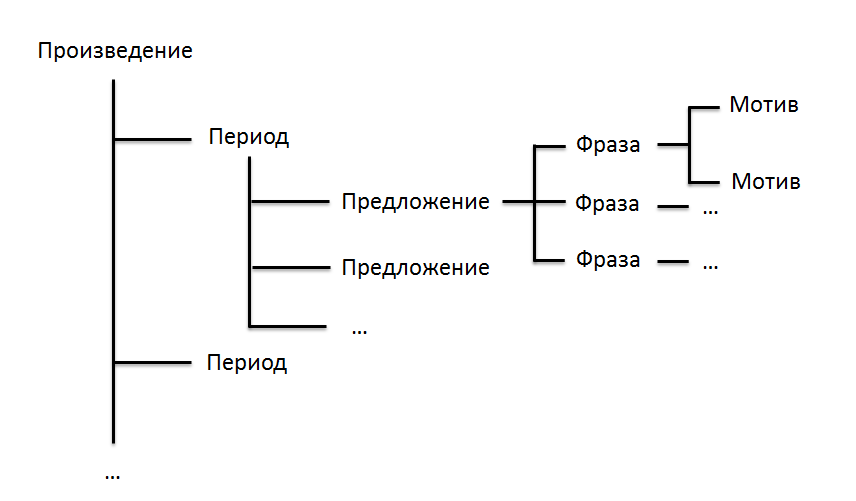


Рис. 3 Структура музыкального произведения.

Из различных свойств гармонии основное формообразующее значение имеет функциональность. Главная (тоническая) тональность играет роль тоники в широком смысле слова, то есть она устойчива. В главной тональности произведение большей частью начинается и почти всегда завершается. Поэтому она служит централизующим, подчиняющим началом.

Всякая другая тональность в произведении неустойчива, в широком смысле этого слова, и будет называться подчиненной. Введение подчиненной тональности – стимул для дальнейшего развития.

Под понятием «ритм», в тесном смысле, подразумевается организованная последовательность звуковых длительностей. Ритм скрепляет воедино различные звуки и аккорды и создает из них группы, имеющие осмысленность.

Основные ритмические “измерения”, применяемые в музыке — целая нота, половинная, четверть, восьмая, шестнадцатая, тридцать вторая и т.д. Все это величины относительные, каждая из них вдвое меньше предшествующей и вдвое больше последующей; абсолютная их длительность приблизительно устанавливается обозначением темпа, т. е. скорости исполнения.

## Упрощенная модель музыкального произведения

Сочинение музыки относится к классу задач, связанному с имитацией и моделированием творческой деятельности человека. Условия и методы их решения, как правило, трудно формализуемы, а алгоритмы решения носят эвристический характер. [5]

Ключом к решению эвристической задачи подобного класса является правильное составление абстрактной модели рассматриваемой задачи и принятие ряда упрощений и допущений, отбрасывающих наиболее неэффективные с точки зрения реализации элементы. Введем ряд ограничений и допущений, позволяющих сформулировать такую модель музыкального произведения, которая будет реализуема за реальное время.

Во-первых, упростим структуру произведения, сделав ее следующей:

* Произведение будет состоять из трех периодов (изложение темы, разработка, реприза);
* Период будет включать в себя два предложения;
* Предложение будет состоять из двух фраз;
* Фраза будет включать один или два мотива;
* Мотив – минимальная единица мелодии, длиной в полтакта или такт.

Во-вторых, ритмическая модель будет включать только целые ноты, половинные, четвертные, восьмые и шестнадцатые. Таким образом, мы исключаем все «нестандартные» длительности, вроде триолей и квинтолей и убираем продление длительностей через границу такта. Однако при этом допускается использование точки при каждой из длительностей (кроме шестнадцатой), т.е. половинная с точкой и четверть будут образовывать завершенный такт.

В-третьих, будут использоваться только два вида ладов – натуральный минор и натуральный мажор. Другие лады также часто применяются в музыке, однако реализация измененных ступеней звукоряда влечет за собой ряд других изменений, начиная от типа аккордов, строящихся на ступенях звукоряда, и заканчивая правилами гармонизации и звуковедения в целом. Поэтому пожертвуем всем разнообразием ладов в пользу двух наиболее часто употребляемых и обеспечим себя, таким образом, твердой основой для гармонизации.

В-четвертых, упростим правила гармонизации мелодии до небольшого набора правил, касающихся гармонизации лишь аккордами основных ступеней. Наиболее сложно формализуемые правила гармонии связаны, в основном, с септаккордами (аккордами из четырех нот), которыми можно пожертвовать без значительного ущерба для музыкальной модели в целом.

## Конструирование мелодий с помощью паттернов



### Формализация отношений между двумя соседними нотами

Главным в генерации музыки является процесс генерации мелодии. Она может пониматься широко – как отдельный мотив и как набор из нескольких фраз, и даже предложений. Для удобства, будем называть мелодией один или нескольких, связанных чем-то общим, мотивов.

Сочинение мелодии композитором – процесс чрезвычайно изменчивый и практически не формализуемый. Невозможно точно сказать, по каким правилам строится тот или иной мотив, почему идущие в одном порядке ноты услаждают слух, в то время как в ином порядке они превращаются в жуткую какофонию.

Поиск этих правил построения мелодии – задача, не имеющая универсального решения. В общем она сводится к выведению таких правил, которыми можно описать структуру абсолютно любой мелодии, написанной людьми на протяжении всей истории. Количество таких правил прямо пропорционально количеству музыкальных произведений, написанных в различных музыкальных стилях – а оно исчисляется сотнями (если не тысячами) миллионов.

Основываясь на вышесказанном, сформулируем принципы сочинения мотива, которые позволят описать простые мелодии, взятые из классической музыки.

Для начала рассмотрим, как могут быть связаны две подряд идущие ноты в одном голосе:

* Они могут быть соседними элементами в составе одного мотива/мелодии
* Первая может быть концом одного мотива, а вторая – началом другого

Теперь рассмотрим возможные варианты положения нот в одном и том же мотиве. Строить будем в тональности ля-минор от ноты ля (Рис. 4, Рис. 5).



Рис. 4 Построение от ноты ля вверх.

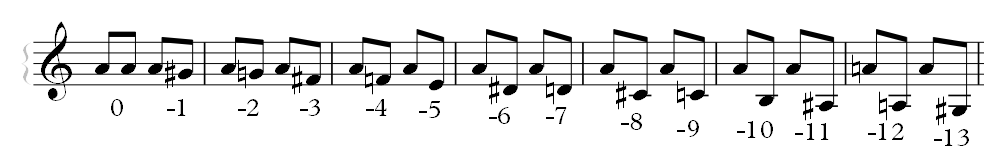


Рис. 5 Построения от ноты ля вниз.

На Рис. 4 изображены все интервалы от 0 (прима) до 13 (малая нона), построенные от ноты ля вверх (интервалы могут и превосходить малую нону – на общей формулировке правил это не скажется). Все эти интервалы можно охарактеризовать общим правилом **«вторая нота отстоит от первой на n нот (где n ∈** \mathbb N∪{0}**)»**. Аналогично, для движения вниз – будем использовать расстояние в нотах с отрицательным знаком. Таким образом, правила можно объединить: **«вторая нота отстоит от первой на n нот (где n ∈** \mathbb Z**)».**

Такое правило позволяет описать все отношения между двумя соседними нотами, однако формулировать правила композиции с помощью одного лишь такого условия – чрезвычайно неудобно. Поэтому, сгруппируем некоторые из интервалов в отдельные правила.

Свяжем переход от одной ноты к другой с текущей тональностью произведения. Так как мы ограничили используемые виды ладов только до двух – натурального минора и натурального мажора – то будут рассматриваться только они. Заметим, что для минора и мажора расстояния между нотами будут различаться (Рис. 6, Рис. 7).

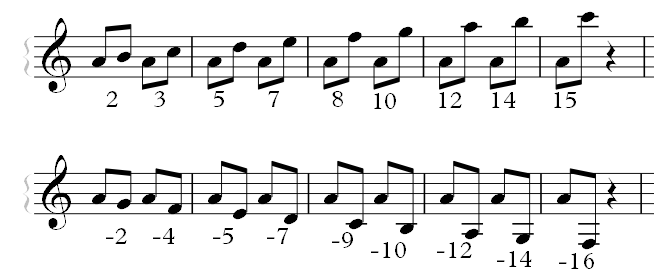


Рис. 6 Построение ступеней вверх и вниз в натуральном миноре

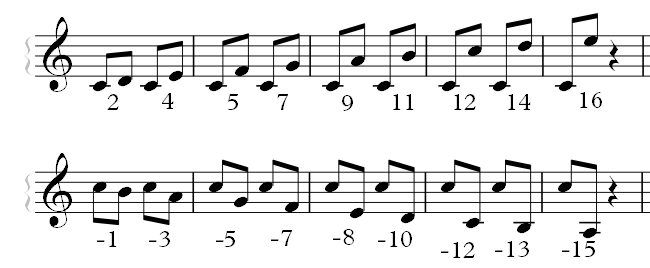
от ноты ля.

Рис. 7 Построение ступеней вверх и вниз в натуральном мажоре

от ноты до.

На Рис. 6 и Рис. 7 числовые значения под группами нот означают расстояние между нотами в абсолютных единицах. Например, если нота До четвертой октавы имеет высоту, равную 60, а Ре - 62, то 2 это Ре - До = 62-60 = 2.

Очевидно, что при построении от других нот соотношения внутри минора и мажора останутся неизменными, т.е. для минора это будет:  
**{0, 2, 3, 5, 7, 8, 10}, {0, 2, 3, 5, 7, 8, 10} + 12, {0, 2, 3, 5, 7, 8, 10} + 24, ...**

а для мажора, соответственно:  
**{0, 2, 4, 5, 7, 9, 11}, {0, 2, 4, 5, 7, 9, 11} + 12, {0, 2, 4, 5, 7, 9, 11} + 24, ...**

Числовая последовательность при нисходящем движении (D) определяется по следующему правилу:  
**T = {0, 2, 3, 5, 7, 8, 10, 12}; I = invert(T) = {12, 10, 8, 7, 5, 3, 2, 0};   
Dk = 12 - {Ik}; D = {0, -2, -4, -5, -7, -9, -10, -12}**

И, аналогично, для мажора:  
**T = {0, 2, 4, 5, 7, 9, 11, 12}; I = invert(T) = {12, 11, 9, 7, 5, 4, 2, 0};   
Dk = 12 - {Ik}; D = {0, -1, -3, -5, -7, -8, -10, -12}**

Таким образом, сформулируем следующее правило: **«вторая нота отстоит от первой на n ступеней тональности T (где n ∈** \mathbb Z**)»**. В частном случае: **«вторая нота является следующей (предыдущей) ступенью тональности, по отношению к первой».**

После введения связи с тональностями логично будет переформулировать первоначальное правило: **«вторая нота отстоит от первой на n ступеней хроматической гаммы тональности T   
(где n ∈** \mathbb Z**)».**

Это правило будет использоваться нами очень редко, однако, следующий его частный случай мы будем применять в мелодическом движении: **«вторая нота отстоит от первой на полтона вверх (вниз)».**

Теперь, когда у нас уже есть правила, описывающие прыжки на произвольные интервалы, а также движения по ступеням тональности и по хроматической гамме, сформулируем правила, связанные с движением по ступеням определенного аккорда. В классической музыке очень часто используются приемы движения по какому-либо аккорду, поэтому игнорировать их было бы упущением. Рассмотрим движения по нескольким аккордам (Рис. 8).

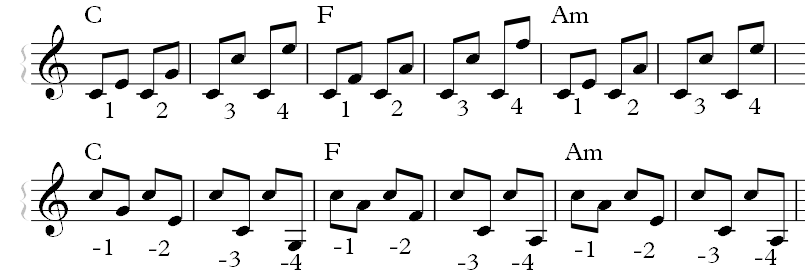


Рис. 8 Движение по аккордам до-мажор (C), фа-мажор (F) и ля-минор (Am) вверх и вниз.

В данном случае движение происходит только по нотам, входящим в состав того или иного аккорда (с учетом октавных сдвигов), то есть, для до-мажорного трезвучия это до-ре-ми, для фа-мажорного – фа-ля-до, а для ля-минорного – ля-до-ми. Числа под нотами обозначают, на сколько ступеней аккорда делается шаг. Так, например, 1 – следующая ступень, т.е. определяем какой ступенью была первая нота (например, в до-мажорном аккорде, нота соль – третья ступень аккорда), и берем следующую аккорда с учетом сдвигов на октаву (т.е. после ноты соль в до-мажорном аккорде вновь появится нота до). Число же 3 будет означать, что нужно перескочить через две последующие ступени и взять третью ступень (в случае с трезвучиями такой шаг эквивалентен прыжку на октаву). Аналогично и для движения вниз.

В общем виде правило звучит так: **«вторая нота отстоит от первой на n ступеней аккорда A(где n ∈** \mathbb Z**)»**.

Подобные движения для аккордов чрезвычайно удобны для описания мотивов следующего вида (Рис. 9). Как видно из примера – все мотивы имеют одинаковую структуру, которая описывается набором переходов по аккордам («0» означает, что берется такая же ступень, как и у предыдущей ноты), и лишь изменяя аккорд, по которому ведется движение, получаются различные мотивы.

Также, такое правило позволит описывать стандартные приемы аккомпанемента, широко используемые в классической музыке (Рис. 10).



Cm 1 1 -1 -1 1 -1 0 0

Cm 1 1 -1 -1 1 -1 0 0

Fm 1 1 -1 -1 1 -1 0 0

Fm 1 1 -1 -1 1 -1 0 0

Hdim 1 1 -1 -1 1 -1 0 0

Hdim 1 1 -1 -1 1 -1 0 0

Cm 1 1 -1 -1 1 -1 0 0

Рис. 9 И.С. Бах, Маленькая прелюдия в до-миноре.



**1 1 -1 2 -1 1 1 -1 1 3 2 -2**

Рис. 10 Пример аккомпанемента, построенного по аккордовым правилам.

Во всех вышеуказанных правилах случай, когда вторая нота совпадает с первой, задается по-разному: это либо движение по гамме на 0 ступеней, либо прыжок на 0 нот в любом направлении и т.д. Суть же одна – **«вторая нота является повтором первой»**. Введем такое правило как отдельное для удобства применения при записи мотивов.

Итак, у нас имеется четыре вида правил, описывающих движения от одной ноты к другой:

* **«вторая нота отстоит от первой на n ступеней тональности T (где n ∈** \mathbb Z**)»;**
* **«вторая нота отстоит от первой на n ступеней хроматической гаммы тональности T (где n ∈** \mathbb Z**)»;**
* **«вторая нота отстоит от первой на n ступеней аккорда A(где n ∈** \mathbb Z**)»;**
* **«вторая нота является повтором первой»**.

Теперь сформулируем правила построения мотивов в целом. Для этого введем понятие «музыкального паттерна».



### Построение мотива с помощью паттернов

**Паттерн** - закономерная регулярность, повторяющийся шаблон, образец.

**Музыкальный паттерн** – последовательность из двух или более нот (или более мелких паттернов), имеющих относительную длительность, не привязанная к какой-либо тональности или аккорду.

**Простейшими паттернами** являются две ноты, связанные любым из четырех вышеуказанных правил для связи двух нот. Выпишем наиболее часто используемые простейшие паттерны (Рис. 11).

Рис. 11 Виды простейших паттернов.

Простейшие паттерны уже являются достаточными кирпичиками для построения мелодии, однако было бы логично скомбинировать наиболее часто встречающиеся последовательности паттернов и сформировать **составные паттерны**.

**Составными паттернами** будем называть комбинацию из двух и более паттернов (как простейших, так и составных). На Рис. 12 и Рис. 13 представлены варианты различных составных паттернов.

Рис. 12 Примеры составных паттернов.



Рис. 13 И.С. Бах, ХТК (Том I), Прелюдия и фуга №2 до-минор.

Из Рис. 13 видно, что с помощью составных паттернов уже можно описать целые мотивы, и представить их в виде структуры, удобной для транспонирования по гамме и/или аккордам.

Этап 1

Этап 2

Этап 3

Определение длительности мотива

Выбор первой ноты мотива

Построение каркаса из паттернов

Замена простых паттернов на составные аналоги

Добавление мелизмов к паттернам

Выбор ритмической структуры

Диаграмма 1 Процесс построения мотива.

Итак, мы подошли к тому, что **любой мотив – есть набор простейших и составных паттернов**. А процесс сочинения нового мотива, в конечном счете, сводится к комбинации различных паттернов друг с другом.

Но в этом случае возникает вопрос: как именно производить такую комбинацию? Если брать произвольные паттерны и произвольные ноты и пытаться их присоединить друг к другу, то мы, в конце концов, получим мотив, звучащий бессвязно и отрывисто. А нашей первостепенной задачей является построение лаконичной, благозвучной мелодии с аккуратной структурой. Поэтому, выведем алгоритм конструирования, позволяющий получать приемлемые мотивы.

Процесс построения мотива будет происходить в три этапа.

Этап первый:

* Определяем общую длительность мотива (в метрических единицах);
* Выбираем ритмическую структуру мотива (длительности опорных нот в метрических единицах)
* Выбираем первую ноту, с которой будет начинаться мотив;
* Начинаем пошагово накладывать простейшие паттерны (уже в нужных метрических единицах), выбирая каждый последующий паттерн произвольным образом, но с чередованием движения вверх и вниз, чтобы в конечном счете не получалось только восходящих или только нисходящих движений (однако, в некоторых случаях такие «односторонние» мотивы могут быть созданы умышленно). Используются все типы простейших паттернов, кроме аккордовых.
* В итоге получим мотив из нот равных длительностей (Рис. 14).



Рис. 14 Результат первого этапа конструирования мотива.

Этап второй:

* Начинаем второй проход по мотиву: на этом проходе некоторые из нот заменяются простейшими и составными паттернами, выбираемыми на основе расстояния между соседними нотами.
* В итоге получается мотив, состояний из нот разных длительностей и уже претендующий на звание завершенного мотива (Рис. 15).

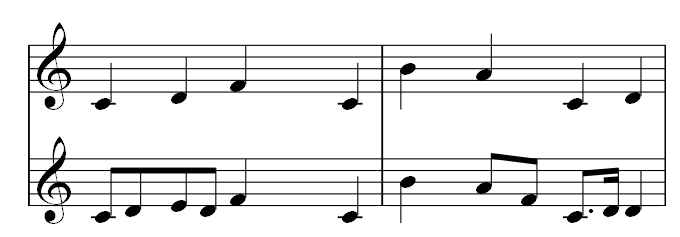


Рис. 15 Результат второго этапа конструирования мотива.

Этап третий:

Основная задача третьего этапа – «отполировать» полученный на втором этапе мотив.

* Начинаем третий проход по мотиву: анализируем связи между нотами и производим замену некоторых простейших паттернов на эквивалентные составные; добавляем мелизмы (форшлаги и трели) к некоторым нотам, удовлетворяющим условиям мелизмов.
* В итоге получается завершенный мотив, который может служить основой для дальнейшего построения мелодии и музыкального произведения в целом (Рис. 16).



Рис. 16 Результат последнего этапа конструирования мотива.

Теперь, когда есть готовый мотив можно двигаться дальше и составлять музыкальную фразу.

Как уже было сказано в разделе 2.2 «Упрощенная модель музыкального произведения», в состав одного предложения будет входить две фразы, а фраза будет состоять либо из одного, либо из двух мотивов. Слегка уточним это правило, отметив, что предложение должно состоять либо из двух, либо из четырех мотивов (т.е. опустим ситуации с длиной предложения в три мотива). Это обуславливается тем фактом, что музыке по своей природе присуща квадратность – метрическая организация, при которой число тактов в построениях равно числу два, возведенному в какую-либо степень [2]. Случаи с иной метрической организацией также встречаются в классической музыке, однако в рамках данной модели мы не будем рассматривать их.

При построении второго мотива мы должны учитывать данные, вводимые пользователем за время воспроизведения первого мотива. Пользователь может задать направление движения мелодии, и новое опорное значение будет учитываться при построении второго и последующих мотивов.

Итак, для начала рассмотрим более простой случай – предложение, состоящее из двух мотивов. Возможны два варианта комбинации мотивов в таком случае:

* Второй мотив является видоизменением первого (A A’);
* Второй мотив построен независимо от первого (A B);

Алгоритм построения мотива в первом случае следующий:

* Берем первый мотив в виде, полученном после второго этапа генерации (мы храним информацию обо всех трех этапах генерации мотива);
* В зависимости от положения опорой ноты (введенной пользователем) и стартовой ноты мотива вносим соответствующие коррективы: либо сдвигаем мотив целиком в нужном направлении, либо правим некоторые из паттернов, чтобы достичь движения до нужной отметки;
* Повторяем третий этап генерации мотива и получаем таким образом видоизмененный мотив ведущий в нужном направлении.

Алгоритм построения мотива во втором случае:

* В зависимости от положения опорой ноты (введенной пользователем) и стартовой ноты выполняем полный трехэтапный алгоритм генерации мотива. При этом возможны ситуации и восходящих\нисходящих мотивов.

Второй мотив является видоизменением первого?

Третий этап генерации мотива

Конец

Генерация нового мотива с учетом опорной ноты

Сдвиг мотива/изменение некоторых паттернов

Да

Нет

Диаграмма 2 Генерация второго мотива.

Для предложений, состоящих из четырех мотивов, комбинаций будет значительно больше. Оставим из них только те, что имеют применение в классической музыке:

AA’A’’A’’’ AA’A’’B AA’BB’ ABA’B’ AA’BC ABCB’ ABCD

Каждый новый мотив строится в зависимости от вводимых пользователем данных.

После того, как сгенерировано первое предложение, начинается генерация второго предложения. Второе предложение является модификацией первого, то есть AA’ – каждый из мотивов модифицируется по отдельности и из них складывается предложение.

Произведение строится из периодов (каждый из них включает два предложения) по схеме A B A’, так что второй период строится независимо от первого, а третий является модификацией первого.

Отдельно стоит отметить случай многоголосой генерации мелодии. Здесь мотив, прозвучав в одном из голосов, может быть повторен без изменений в другом, так что принцип построения будет немного отличаться от одноголосого. Подробнее см. Раздел 2.5. «Генерация второстепенных голосов и аккомпанемента».

## Алгоритм гармонизации мелодии

В классической музыке под термином «гармонизация мелодии» понимается поиск аккордовой сетки и построение голосов до полного четырехголосия. В рамках нашей модели под «гармонизацией мелодии» будем понимать только **подбор аккордового сопровождения**.

Как уже было отмечено в пункте об упрощенной музыкальной модели, при гармонизации будут использоваться не все средства современной гармонии (включающие в себя гармонизацию септаккордами, кадансовый оборот, модуляции в смежные тональности и т.п.), а только базовые.

Подбор осуществляется путем поиска среди всех аккордов ступеней текущей тональности тех, которые включают в себя текущий звук мелодии. Так, нота «ми» в тональности до-мажор входит в состав аккордов I, III и VI ступеней (до-мажорное, ми-минорное и ля-минорное трезвучия). Другие примеры вхождения нот в состав аккордов показаны на Рис. 17.

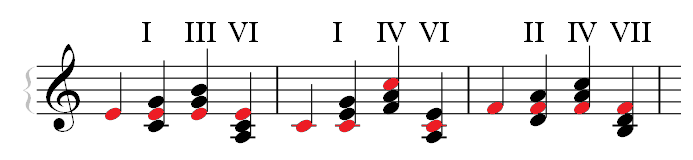


Рис. 17 Примеры вхождения нот в состав аккордов в тональности до-мажор.

Каждая нота в гамме входит в состав трех различных трезвучий (как первая, вторая и третья ноты трезвучия, соответственно). Отсюда следует, что каждую ноту теоретически можно гармонизировать тремя разными способами. Однако, принципы музыкальной гармонии таковы, что не все последовательности аккордов допустимы. Так, одним из основных принципов является принцип ~~D-S~~ – запрет на использование аккорда IV после V.

Для формализации подобных запретов гармонии (а также правил, связанных с тяготением одних аккордов к другим) удобнее всего задать для каждого из аккордов список из «последователей» - аккордов, которые могут быть применены после текущего. В общем виде, список будет выглядеть следующим образом:

I → II, III, IV, V, VI, VII V → I, III, VI, VII

II → I, III, V VI → I, II, III, IV

III → I, IV, V, VI, VII VII → I

IV → I, II, V

Основываясь за списке допустимых аккордовых последований, составим алгоритм гармонизации мелодии:

1. Мелодия разбивается на такты. Работа с каждым тактом ведется отдельно.
2. Внутри каждого берется нота мелодии. По ней выбирается список аккордов, в состав которых она входит, а затем этот список сверяется со списком «последователей» предыдущего аккорда (если таковой есть). Если пересечение списков не пусто – то мелодия гармонизируется аккордом из пересечения. Если пересечение списков пусто – то происходит «заход в тупик» - доступных вариантов гармонизации больше нет. В такой ситуации процесс гармонизации возвращается на один шаг назад и предыдущая нота гармонизируется уже другим аккордом, чтобы предотвратить заход в тупик.
3. Процесс повторяется для каждой ноты. Как только последняя нота последнего такта гармонизирована – процесс гармонизации останавливается.

Алгоритм при этом имеет ряд дополнительных условий:

1. При гармонизации самого первого такта мелодии для первой ноты всегда берется тоническое трезвучие – I – это делается для демонстрации тональности и закрепления тоники.
2. При гармонизации четвертого такта мелодии (конец первого предложения) для первой ноты берется доминанта – V – так как в классическом периоде первое предложение обычно заканчивается на доминанте. Это неявно демонстрирует тональность доминанты, в которую в дальнейшем происходит переход.
3. При гармонизации восьмого такта мелодии (конец второго предложения) для первой ноты берется тоника – I – это показывает завершение мелодии и придает ей завершенность.
4. При гармонизации третьего и седьмого тактов избегается использование аккордов V и I соответственно – чтобы избежать повтора гармонии с последующими тактами, для которых гармония задается жестко.
5. При гармонизации такта максимальное количество аккордов в нем – 2. Такое ограничение позволяет избежать перегрузки такта различными аккордами, поскольку слишком частая их смена тяжела для восприятия слушателем. В случае если в такте их уже два, все дальнейшие ноты гармонизируются последним аккордом такта.
6. Сохранение текущего аккорда имеет больший приоритет над сменой: например, если подряд идут ноты «до» и «ми», и первая была гармонизирована аккордом до-мажор (до-ми-соль), то нота «ми» также будет гармонизирована до-мажором. Исключение – если ноты принадлежат разным тактам, тогда необходима перегармонизация (поиск другой допустимой аккордовой сетки).

4

Гармонизация

Нет

Есть не гармонизированные ноты?

Успех

Другой

В такте уже есть два разных аккорда?

Да

Гармонизация через последний аккорд такта

Успех

Нет

Выбор списка аккордов, гармонизирующих ноту

Удаление аккордов, не гармонирующих с предыдущим аккордом

Нет

Номер текущего такта

3 или 7

Удаление из списка аккордов V или I

Список оставшихся аккордов пуст?

Да

Нет

Неудача

Гармонизация (остаток такта) успешна?

Нет

Удаление из списка использованного аккорда

Да

Успех

Да

Номер текущего такта

1, 8

Гармонизация через I

Гармонизация через V

Диаграмма 3 Алгоритм гармонизации.

## Генерация второстепенных голосов и аккомпанемента

В любом музыкальном произведении основную роль играет мелодия – именно она представляет собой стержень композиции, его сюжетную линию. Однако, произведения, состоящие только из одной мелодии, звучат слишком бедно и «скучно», и поэтому чаще всего они дополняются второстепенными голосами и аккомпанементом.

В общем случае, под **аккомпанементом** подразумевается инструментальное сопровождение одного солирующего голоса. В рамках данной работы удобнее будет так называть **любой инструментальный голос, построенный нотами мелкой длительности с использованием аккомпанементных паттернов.**  Под мелкой длительностью будем подразумевать шестнадцатые и восьмые ноты.

Также введем понятие второстепенного голоса. **Второстепенный голос - любой инструментальный голос, построенный нотами крупной длительности с использованием аккомпанементных паттернов.** Ноты крупной длительности – четвертные, половинные и целые ноты.

Аккомпанемент лучше всего подходит для генерации «левой руки фортепиано», гитары, арфы и других подобных инструментов-аккомпаниаторов. Второстепенные голоса – для флейт, скрипок, медных и хора.

В классической музыке встречается очень много различных видов аккомпанемента, однако из них можно выделить наиболее часто встречающиеся паттерны. Например, в музыке барокко это – альбертиевы басы, в музыке XIX века – гармонические фигурации (Рис. 18).



Рис. 18 Примеры аккомпанемента.

Генерация аккомпанемента происходит по следующему алгоритму:

* По основной мелодии путем гармонизации определяется набор аккордов для текущего такта.
* Из списка паттернов произвольным образом выбирается аккомпанирующий паттерн.
* Для выбранного паттерна случайным образом определяется длительность нот (восьмые или шестнадцатые) и такт наполняется полученными паттернами, с учетом текущих аккордов.
* В качестве опорной ноты используется произвольная ступень аккорда, по которому ведется построение, взятая из определенного диапазона – таким образом, аккомпанемент не будет выходить за пределы двух октав и, тем самым, не будет конфликтовать с мелодией.

Гармонизация мелодии, создание аккордовой сетки

Выбор ритмической структуры

Выбор опорной ноты аккомпанемента

Наполнение такта паттернами

Выбор паттерна аккомпанемента

Диаграмма 4 Алгоритм построения аккомпанемента.

Второстепенные голоса по своей структуре очень схожи с результатом первого этапа генерации основной мелодии. Однако, в отличие от мелодии, при генерации второстепенных голосов разрешено применение аккордовых паттернов (более того, они являются приоритетными) и упор делается на использование половинных нот, половинных с точкой и целых. На Рис. 19 изображены примеры второстепенных голосов по аккорду до-мажор.



Рис. 19 Примеры второстепенных голосов по аккорду до-мажор.

Алгоритм построения второстепенных голосов имеет следующий вид:

* По основной мелодии путем гармонизации определяется набор аккордов для текущего такта.
* Определяем первую ноту, с которой будет начинаться мотив (аналогично аккомпанементу);
* Начинаем пошагово накладывать простейшие паттерны, выбирая каждый последующий паттерн произвольным образом. При этом избегаем больших прыжков (более чем на две ступени аккорда).

Гармонизация мелодии, создание аккордовой сетки

Выбор первой ноты

Пошаговое построение голоса из простейших паттернов

Выбор ритмической структуры

Диаграмма 5 Алгоритм построения второстепенных голосов.

Таким образом, путем генерации дополнительных второстепенных голосов и аккомпанемента основная мелодия становится гармонически дополненной и благозвучной.

# Реализация генератора музыки



## Архитектура генератора музыки

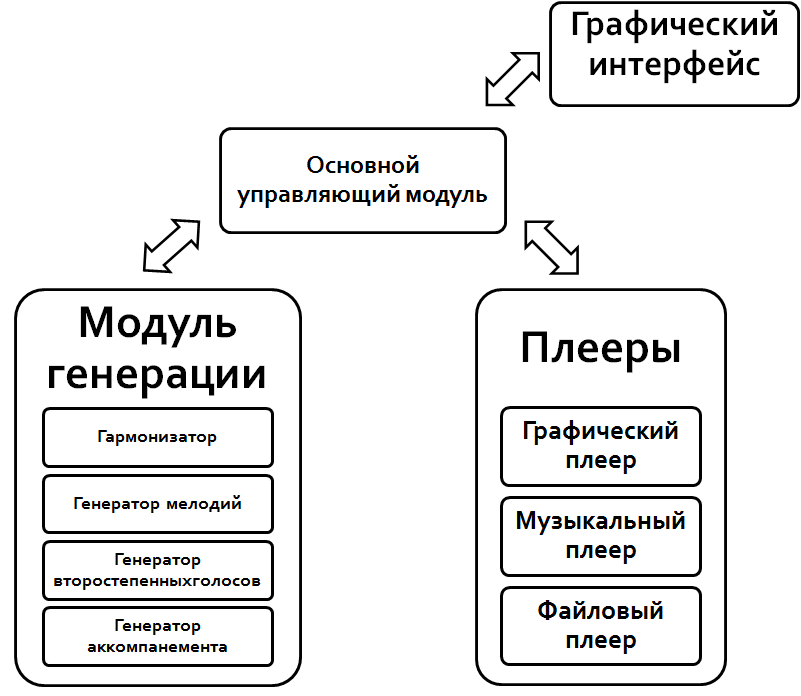


Рис. 20 Схематическое изображение архитектуры генератора музыки.

Генератор музыки состоит из нескольких обособленных модулей, которые, взаимодействуя друг с другом, позволяют в реальном времени генерировать потактово музыкальное произведение.

Всего есть четыре основных программных модуля:

* Основной управляющий модуль;
* Модуль генерации музыки;
* Модуль воспроизведения (плеер);
* Модуль взаимодействия с пользователем (графический интерфейс).

Каждый модуль функционирует независимо от других, и взаимодействие происходит только через управляющий модуль. Подобная схема позволяет менять и отключать любой из модулей, без необходимости внесения изменений в остальные модули. Например, можно оставить только один из подмодулей воспроизведения (или отключить его целиком) – и при этом генерация мелодии будет по-прежнему происходить без изменений.

Для реализации генератора музыки был выбран язык Java, который, как и другие высокоуровневые языки, предоставляет удобные средства для работы со звуковыми устройствами операционной системы и со звуком в целом. При разработке использовался комплект разработчика приложений Java Development Kit 7, включающий в себя расширенный пакет работы с музыкальными синтезаторами, и открытая библиотека jMusic 1.6.4, предоставляющая доступ к музыкальным примитивам, вроде ноты и фразы, а также средства взаимодействия с MIDI.

Структура проекта представляет собой набор взаимодействующих друг с другом модулей, которые, согласно принятым в Java стандартам, разнесены по различным пакетам.

* **core –** пакет, содержащий основной управляющий модуль генератора;
* **core.data.\*** - пакеты, включающие в себя основные элементы генератора: ноты, паттерны, аккорды, тональности, константы и все сопутствующие подклассы;
* **core.generators.\*** - пакеты, содержащие модули генерации музыки (включая гармонизацию, построение основного и второстепенных голосов, аккомпанемента);
* **gui** – пакет, включающий в себя графический интерфейс генератора музыки;
* **jm.\*** - пакеты библиотеки jMusic;
* **main –** базовый пакет, содержащий класс, необходимый для запуска приложения;
* **player –** пакет, содержащий в себе все модули воспроизведения (музыкальный, графический и файловый);
* **tests –** пакет, включающий в себя набор jUnit-тестов для приложения;
* **util –** пакет со вспомогательными классами.

## Основные программные элементы

В состав библиотеки jMusic уже входят необходимые музыкальные примитивы: ноты, паузы, фразы, партии, партитуры и инструмента, которые позволяют программировать музыку на приемлемом уровне абстракции. Однако для формализации музыкальных мотивов необходимо ввести ряд дополнительных классов, являющихся надстройкой над базовыми классами, что позволит оперировать такими понятиями как паттерн и мотив. Рассмотрим такие элементы по-отдельности.

* + 1. Нота паттерна (PNote) – наименьший музыкальный элемент, являющийся расширением базового класса ноты (Note), но вносящий в нее ряд дополнительных параметров, которые позволяют описывать ноты не в абсолютных звуковых высотах, а в относительных единицах.

-pitch:int

-absolute\_length:double

-length:double

-transitionType:int

-transitionValue:int

-volumeChange:int

-rhythmValue:double

-melisma:Melisma

PNote

* + 1. Паттерн (Pattern) – объект, хранящий в себе список нот, отражающих его структуру. Также есть два расширения паттернов – это «паттерн одной ноты» (SNPattern) и «составной паттерн» (CPattern). Первый паттерн может хранить только одну ноту и используется для задания каркаса мотива, а второй позволяет строить сложные структуры, состоящие из более простых паттернов.

#notes:List<PNote>

-name:String

Pattern

* + 1. Мотив (Motive) – аналог базового класса фразы (Phrase), позволяющий описывать отрывок мелодии в виде последовательности паттернов (в основном, имеет длительность равную одному такту). Хранит в себе результаты всех трех этапов генерации мелодии.

+firstRunPatterns:LinkedList<Pattern>

+secondRunPatterns:LinkedList<Pattern>

+thirdRunPatterns:LinkedList<Pattern>

Motive

* + 1. Голос (Voice) – класс, представляющий отдельный голос (инструмент) в произведении. Отвечает за генерацию мелодии, в зависимости от ее типа и гармонизацию мотива в голосе, а также хранит все параметры инструмента (высота, громкость, тип инструмента, номер звукового канала и т.д.)

+role:int

+name:String

+channel:int

+instrument:int

+octaveSummand:int

-voiceGenerator:BaseVoiceGenerator

-motive:Motive

-lastNote:int

-muted:boolean

Voice

* + 1. Мелизм (Melisma) – используется для описания мелизмов, добавляемых к нотам. Включает в себя только два поля: тип мелизма (type) и расстояние до ноты (pitchValue). Из этих двух параметров в конечном итоге составляются ноты-украшения, которые могут быть либо форшлагами, либо трелями.

-type:int

-pitchValue:int

Melisma

Для хранения набора применяемых в программе паттернов было принято решение использовать в качестве базы данных текстовые файлы. Такой подход не требует долгой реализации и позволяет быстро и легко вносить изменения в данные.

Все паттерны хранятся в специальном каталоге database в файлах с расширением .txt. Ниже приведены примеры задания паттерна в текстовом виде:

Name~C1T-1T1

2~1~1

1~-1~1

1~1~1

End

Как видно из структуры паттерна, описание начинается с кодового слова «Name», за которым после символа тильды следует имя паттерна (может быть пусто). Дальше построчно идет задание нот, входящих в паттерн, тройками из чисел, которые отвечают на определенные поля класса PNote : первая число – transitionType, второе – transitionValue, а третье – length. Также? могут быть указаны дополнительно еще три поля (rhythmValue, pitch и absolute\_length). В конце описания паттерна стоит кодовое слово End.

За работу с базой паттернов отвечает класс DataStotage, который инициализируется при запуске программы и считывает в оперативную память все входные файлы.

## Программные модули



### Основной управляющий модуль

Основной управляющий модуль представляет собой класс, инициализирующий остальные модули и контролирующий весь процесс работы генератора музыки. В его задачи входят:

* инициализация всех модулей;
* обеспечение взаимодействия между различными модулями;
* передача входных и выходных данных;
* контроль над процессом генерации в целом.

При запуске программы сначала происходит полная инициализация всех программных модулей, а также запуск плееров в отдельных потоках исполнения. Затем управление передается основному модулю, запускающему основной цикл, в котором происходит периодическое обращение к генератору музыки, и результат (сгенерированная мелодия) перенаправляется в плееры. Генерация следующего такта происходит во время воспроизведения текущего такта, то есть, таким образом, генерируется непрерывная бесшовная мелодия.



### Модуль генерации

Модульная структура программы обеспечивает возможность объединения нескольких подмодулей, работающих вместе. Поэтому логично сгруппировать все элементы, отвечающие за генерацию музыки – при таком подходе можно целиком обособить генератор от внешней среды и протестировать его работоспособность независимо от плееров или основного управляющего модуля.

В состав модуля генерации входят следующие подмодули:

* модуль генерации главной мелодии;
* модуль гармонизации;
* модуль генерации второстепенных голосов;
* модуль генерации аккомпанемента.

Слаженность работы этих подмодулей обеспечивается классом генератора (MultiVoicesGenerator)

initVoices()

getVoicesHarmony()

generateMainMotive()

getScore()

addVoice(v:Voice)

MultiVoicesGenerator

-harmonizer:Harmonizer

-voices:LinkedList<Voice> Voice

В обязанности этого класса входит:

* инициализация подмодулей;
* обеспечение доступа к настройкам каждого голоса через пользовательский интерфейс;
* потактовая генерация музыкального произведения по запросу от управляющего модуля.

Сам процесс генерации происходит по следующему алгоритму:

1. Управляющий модуль запрашивает у модуля генерации следующий такт;
2. Модуль основной мелодии генерирует основную мелодию;
3. Модуль гармонизации гармонизирует основную мелодию и передает аккордовую сетку модулю генерации;
4. Модули второстепенных голосов и аккомпанемента генерируют свои голоса на базе аккордовой сетки;
5. Модуль генерации преобразует мелодии всех голосов из типа мотив (Motive) в тип фразы (Phrase). Затем все фразы складываются в объект типа партитура (Score) – примитив библиотеки jMusic – и партитура возвращается основному управляющему модулю;
6. Управляющий модуль передает партитуру на воспроизведение всем плеерам и запрашивает у модуля генерации следующий такт.

При обращении к модулю генерации управляющий модуль вызывает метод getScore(), в котором и реализованы этапы 2-5 алгоритма генерации.

public Score getScore() {

Score s = new Score();

generateMainMotive();

HashMap<Double, Chord> harmony = getVoicesHarmony();

for (Voice v : voices){

v.generateMotive(harmony, CURRENT\_TONALITY);

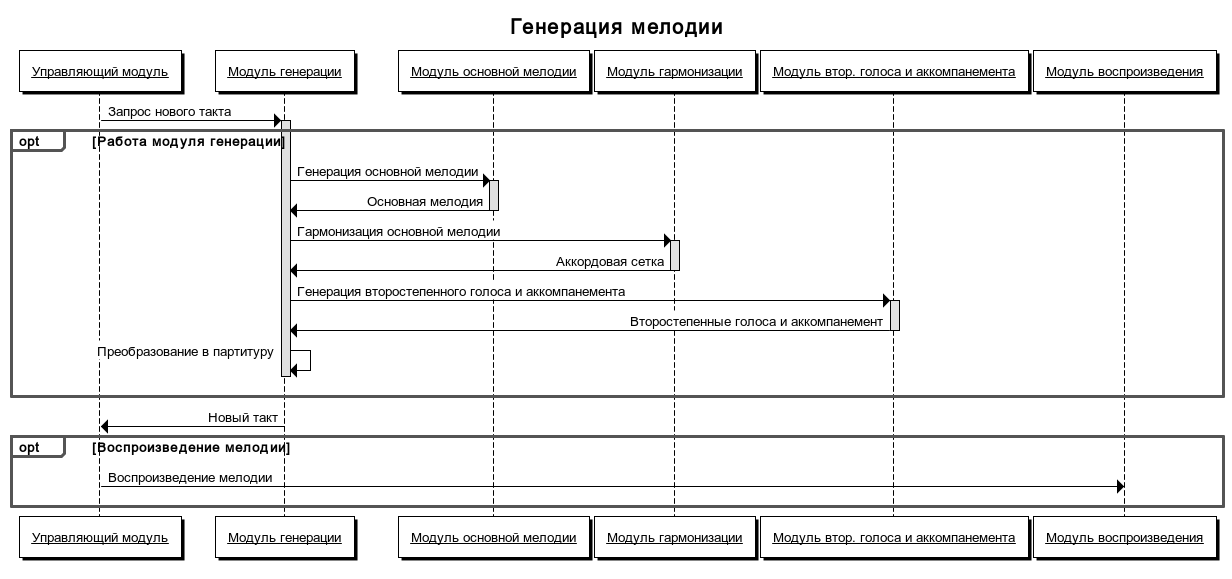
Phrase phr = v.getPhrase(harmony, CURRENT\_TONALITY);

s.add(new Part(phr, v.name, v.instrument, v.channel));

}

return s;

}



### Модуль генерации главной мелодии

Модуль генерации главной мелодии, как уже было сказано, потактово генерирует основную мелодию произведения. Процесс генерации происходит в три этапа, каждый из которых вынесен в отдельный метод. В качестве входных параметров генератор принимает опорную ноту для нового мотива, тональность и предыдущий мотив (если необходимо), на базе которого нужно строить новый мотив.

Класс генератора основной мелодии (MotivesGenerator) наследует базовый класс генератора мелодий (BaseVoiceGenerator) – последний является общей основой и для генераторов аккомпанемента и второстепенных голосов.

generateMotive(arguments:Object)

BaseVoiceGenerator

#currentTonality:Tonality

#currentMotive:Motive

#rhythmLengths:LinkedList<List<Double>>

generateMotive(...)

MotivesGenerator

-previousMotive:Motive

-firstNotePitch:int

-lastBarRhythmLengths:LinkedList<List<Double>>

Процессы генерации немного различаются для разных случаев: для генерации нового мотива используется следующий алгоритм:

1. Инициализация переменных, необходимых для генерации (опорная нота, тональность и т.д.);
2. Генерация каркаса мотива (первый этап);
3. Наложение на каркас составных паттернов (второй этап);
4. Добавление украшений и мелизмов (третий этап).

public Motive generateMotive(int pitch, Tonality cT)

{

currentMotive = new Motive();

firstNotePitch = pitch;

currentTonality = cT;

generateFirstRun(rhythmLengths);

generateSecondRun();

generateThirdRun();

return currentMotive;

}

Генерация мотива на базе уже существующего мотива отличается тем, что первый и второй этапы генерации опускаются и вместо них берутся последовательности паттернов из существующего мотива, а затем во второй этап вносятся изменения. Третий этап генерации сохраняется. Алгоритм при этом можно сформулировать так:

1. Инициализация переменных, необходимых для генерации (опорная нота, тональность и т.д.);
2. Копирование результатов первых двух этапов генерации в новый мотив;
3. Внесение изменений в результаты второго этапа генерации;
4. Добавление украшений и мелизмов (третий этап).

public Motive generateSimilarMotive(Motive prevMotive, int pitch, Tonality cT)

{

currentMotive = new Motive();

previousMotive = prevMotive;

firstNotePitch = pitch;

currentTonality = cT;

duplicatePatternsList(previousMotive.firstRunPattern,

currentMotive.firstRunPatterns);

duplicateCPatternsList(previousMotive.secondRunPatterns,

currentMotive.secondRunPatterns);

randomizeSecondRun();

generateThirdRun();

return currentMotive;

}

Третий вариант выбора основного мотива подразумевает генерацию заключительного такта предложения – то есть четвертого или восьмого по счету тактов. В данном случае алгоритм совпадает с алгоритмом для генерации нового мотива, и отличается лишь ритмическим рисунком – для заключительных тактов приоритет отдается половинным и целым нотам.

### Модуль гармонизации

Модуль гармонизации – второй по важности для всего процесса генерации после модуля главной мелодии. В его задачи входит корректное наложение сетки аккордов на отрывок мелодии с учетом правил гармонии и положения такта с отрывком в музыкальном предложении.

Весь модуль вынесен в отдельный класс Harmonizer, который хранит в себе все необходимые данные для гармонизации отрывка мелодии. При инициализации гармонизатора происходит составление списка возможных последовательностей аккордов.

private void initConsequents() {

chordConsequents = new HashMap<String,LinkedList<String>>();

chordConsequents.put("I", new LinkedList<String>( Arrays.asList("I", "IV", "V", "II", "III", "VI")));

chordConsequents.put("II", new LinkedList<String>( Arrays.asList("II", "V", "I", "III")));

chordConsequents.put("III", new LinkedList<String>( Arrays.asList("III", "IV", "I", "V", "VI")));

chordConsequents.put("IV", new LinkedList<String>( Arrays.asList("IV", "V", "I", "II")));

chordConsequents.put("V", new LinkedList<String>( Arrays.asList("V", "I", "VI")));

chordConsequents.put("VI", new LinkedList<String>( Arrays.asList("VI", "IV", "III", "I", "II")));

chordConsequents.put("VII", new LinkedList<String>( Arrays.asList("VII", "I")));

}

При каждом обращении к гармонизатору вызывается функция getHarmonyForMotive, которая в качестве параметров принимает отрывок мелодии (Motive) и текущую тональность (Tonality). Сначала происходит определение, можно ли гармонизировать отрывок как конец предложения – то есть является ли переданный в качестве параметра такт четвертым или восьмым. Если так – то отрывок гармонизируется определенными аккордами, в противном случае запускается рекурсивный процесс гармонизации нот. Он будет прерван в следующих случаях:

* Гармонизированы все ноты мелодии (процесс гармонизации завершен успешно);
* При гармонизации возникла ошибка и одну из нот невозможно гармонизировать;
* Если в такте уже использовано два разных аккорда – рекурсивный вызов прекращается, и все дальнейшие ноты циклически гармонизируются последним аккордом.

Стоит отметить, что на практике ошибок не возникает – правила гармонизации позволяют подобрать аккордовую сетку под любую из мелодий, сгенерированных модулем основной мелодии.

Результатом работы гармонизатора является отображение из множества позиций в такте во множество аккордов (HashMap <Double, Chord>). Таким образом задается положение аккорда внутри такта и определяется его длительность. В таком виде результат возвращается в модуль генерации, который на базе него строит второстепенные голоса и аккомпанемент.

### Модуль генерации второстепенных голосов и аккомпанемента

Модули генерации второстепенных голосов и аккомпанементов схожи по своим задачам – оба генерируют мелодию на базе имеющейся аккордовой сетки и тональности – что позволяет условно объединить их в одну группу. При этом алгоритмы генерации у модулей несколько различаются.

Модуль аккомпанемента реализован в виде класса AccompanimentGenerator, наследующего BaseVoiceGenerator. За генерацию также отвечает метод generateMotive(), но, в отличие от генерации главной мелодии, первый и третий этапы пропускаются.

public Motive generateMotive(HashMap<Double, Chord> harmony, Tonality currentTonality) {

currentMotive = new Motive();

this.currentTonality = currentTonality;

this.harmony = harmony;

generateSecondRun();

return currentMotive;

}

При этом метод generateSecondRun() работает следующим образом:

1. Из списка аккомпанементных паттернов выбирается произвольный паттерн;
2. Циклически такт наполняется этим паттерном в соответствии с тональностью и аккордовой сеткой;
3. Когда весь такт наполнен аккомпанементом, генерация прекращается.

Генерация второстепенных голосов похожа на первый этап генерации главной мелодии, но с привязкой к ритмической сетке. При инициализации генератора (SecondVoiceGenerator) происходит наполнение массива возможных ритмических рисунков такта:

protected void initRhythmLengths()

{

rhythmLengths = new LinkedList<List<Double>>();

rhythmLengths.add(Arrays.asList(JMC.WHOLE\_NOTE)); rhythmLengths.add(Arrays.asList(JMC.HALF\_NOTE, JMC.HALF\_NOTE)); rhythmLengths.add(Arrays.asList(JMC.HALF\_NOTE, JMC.QUARTER\_NOTE, JMC.QUARTER\_NOTE));

...

}

После того как выбран ритмический рисунок второстепенного голоса, происходит циклическое наложение простейших паттернов на ритм, начиная с опорной ноты первого аккорда такта.

### Модуль воспроизведения

Наиболее удобный способом ознакомления с полученным в процессе генерации музыкальным отрывком – прослушивание его или представление в классической нотной нотации. Для этой цели предназначены модули воспроизведения, каждый из которых по-своему обращается с полученными от генератора данными.

Всего модулей воспроизведения (или, говоря проще, плееров) три:

* Графический плеер – выводит мелодию в виде нот на экран.
* Музыкальный плеер – воспроизводит мелодию в реальном времени.
* Файловый плеер – сохраняет мелодию в MIDI-файл.

Каждый из плееров работает независимо от других и подключается отдельно при инициализации генератора. Все плееры наследуют базовый класс Player и переопределяют в нем три основных метода:

init()

playScore(score:Score)

run()

Player

+isReady:boolean

Добавление плееров к генератору происходит следующим образом:

Player midiPlayer = **new** RealtimeMidiPlayer();

Player filePlayer = **new** FileOutputPlayer();

midiPlayer.init();

filePlayer.init();

generatorManager.addPlayer(midiPlayer);

generatorManager.addPlayer(filePlayer);

Рассмотрим каждый из плееров по отдельности. Самый простой из всех плееров – файловый (FileOutputPlayer). В его обязанности входит конкатенация полученных от генератора отрывков мелодии и сохранение всей партитуры в MIDI файл по требованию. Для этого в методе playScore(Score score) был реализован функционал слияния партитура, а в дополнительном методе - saveMidiFile(String filename) – сохранение текущей партитуры в файл. В дальнейшем такой файл может быть открыт с помощью любого нотного редактора.

Несмотря на тот факт, что сохраненный MIDI файл может быть проигран любым музыкальным плеером, прослушивание уже целиком сгенерированного произведения менее удобно, чем возможность ознакомления с результатом генерации непосредственно в процессе генерации. Именно этой цели служит музыкальный плеер (RealtimePlayer). Принимая на входе партитуру (Score), он конвертирует ее в MIDI-последовательности (Sequence) и направляет на стандартный MIDI-секвенсор, который, используя внешние звуковые библиотеки, воспроизводит мелодию.

В качестве синтезатора используется идущий в комплекте с JDK 1.7 синтезатор Gervill, который позволяет подключать внешние банки звуков в формате .sf2 – они позволяют достичь реалистичного звучания при минимальной нагрузке на процессор и незначительном объеме затраченной оперативной памяти.

Помимо прослушивания, еще одним удобным способом ознакомления с музыкальным материалом является визуальное отображение его в классической нотной нотации. Правильное, с точки зрения классической записи, отображение нот на нотном стане – задача очень сложная, имеющая много нюансов. Мощные современные нотные редакторы умеют правильно конвертировать MIDI-файлы в нотную запись, однако реализация подобного алгоритма в рамках данной работы потребовала бы больше времени, чем реализация самого генератора музыки. Поэтому было принято решение воспользоваться уже имеющимися в пакете jMusic методами для отображения нот в классической записи. Они были модифицированы для поддержки динамического обновления партитуры в процессе генерации. На Рис. 21 представлено окно графического плеера с партиями двух инструментов.

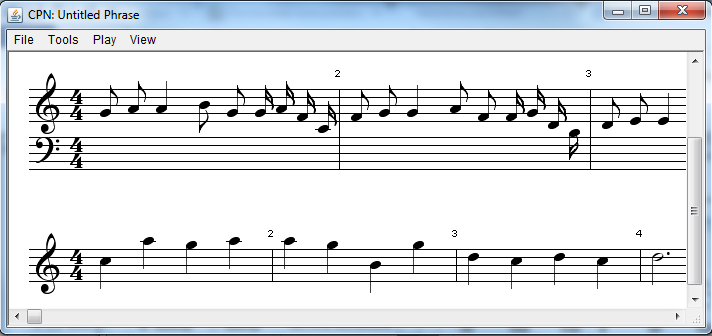


Рис. 21 Окно графического плеера.

### Модуль взаимодействия с пользователем

Генерация музыки в реальном времени подразумевает возможность изменения в любой момент параметров генерации, таких как: темп, тональность, набор инструментов, их громкость и прочее. Для этой цели был разработан пользовательский интерфейс, который целиком умещается в модуле взаимодействия с пользователем и позволяет целиком контролировать весь процесс генерации музыки.

Интерфейс программы представлен на Рис. 22**Ошибка! Источник ссылки не найден.**. В верхней части расположены общие настройки для всей программы: тон и тип тональности, наличие мелизмов и их частота появления, общий темп произведения и громкость. Также здесь есть доступ к меню, где можно сохранить сгенерированное произведение в MIDI-файл и кнопки добавления и выключения инструментов.

Каждый добавленный инструмент имеет собственную панель настроек, где можно задать сам инструмент, тип голоса (мелодия\второстепенный голос\аккомпанемент), громкость инструмента, высоту мелодии (позволяет направлять движение мелодии) и октаву, в которой должен играть инструмент. Также есть опция выключения инструмента и иконка, отображающая выбранный инструмент.

В обязанности интерфейса входит не только передача введенных пользователем параметров генератору, но и предварительная проверка корректности этих параметров. Например, интерфейс не позволяет назначить нескольких инструментам роль основной мелодии, равно как и оставить произведение совсем без основной мелодии.

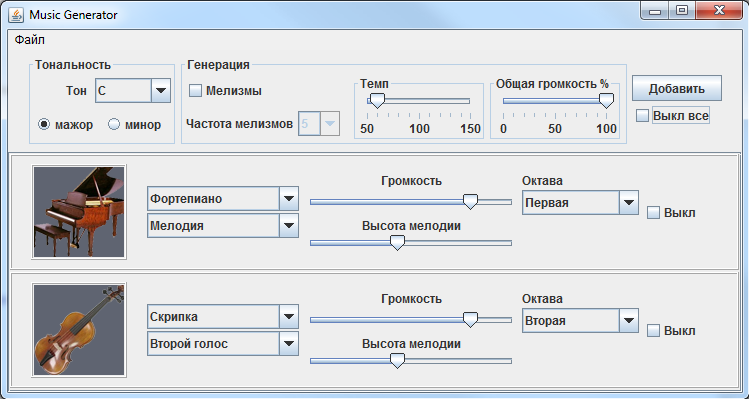


Рис. 22 Пользовательский интерфейс генератора музыки.

Максимальное количество инструментов теоретически не ограничено, на практике не должно превышать 16-20 голосов. При большем числе голосов синтезатор не успевает обрабатывать такое количество звуковых дорожек и корректно воспроизводить их.

# Заключение

В результате проделанной работы был написан полноценный генератор музыки, работающий в реальном времени. В процессе разработки была построена упрощенная модель музыкального произведения и описаны алгоритмы построения мелодий и гармонизации. На базе сформулированных алгоритмов реализован инкрементный генератор музыкальных произведений, работающий с динамически формируемыми паттернами.

Созданные с помощью генератора мелодии имеют простую и приятную на слух структуру. Благодаря модулю гармонизации даже при большом количестве инструментов не возникает диссонансов и конфликтов. Добавление к мелодии 3-4 второстепенных голосов и аккомпанемента значительно обогащает общее звучание, и, в итоге, полученное произведение можно с уверенностью назвать приятным на слух.

# Литература

1. Месснер Е. Основы композиции. — М.: МУЗЫКА, 1968
2. Кюрегян Т.С. Форма в музыке XVII-XX веков. — М.: ТЦ «Сфера», 1998
3. Способин И.В. Музыкальная форма. — М.: МУЗЫКА, 1984
4. Способин И.В. Учебник гармонии. — М.: МУЗЫКА, 1965
5. Тимофеев А.В. Информатика и компьютерный интеллект. — М.: Педагогика, 1991
6. David Cope. Computers and Musical Style. — Madison, WI: A-R Editions, 1991