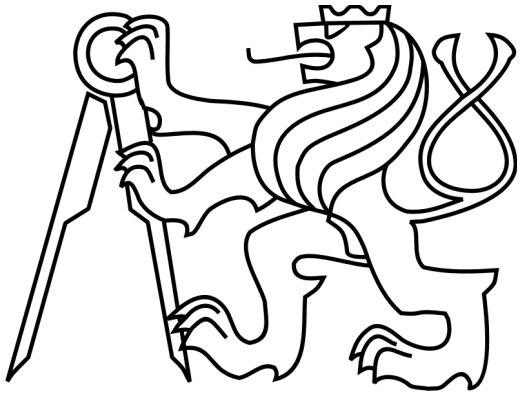
ZADANI

Czech Technical University in Prague
Faculty of Electronic Engineering
Department of Computer Graphics
and Interaction



Bachelor's thesis

GENERATIVE MUSIC TECHNIQUES

Supervisor: Ing. Adam Sporka, Ph.D.

Study Programme: Software Engineering and Management
Field of Study: Web and Multimedia

Prague, 2014 Jiří Troup

Declalaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used. I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb.(copyright law), and with the rights connected with the copyright act including the changes in the act.

Aknowledgements

I would like to thank my supervisor Adam Sporka for all help with my thesis and also for his patience with me. And I would like to thanks all my friends which keep supporting me while I was doing this work.

Abstrakt

Generování hudby se zabývá nejen technikami vytváření hudby , ale také analýzou hudby jako takové a jejím popisem.

Algoritmy pro analýzu hudby jsou dneska už poměrně rozvinuté. Příkladem jsou internetové rádia jako Pandora, nebo Spotify. Tyto stránky dovolují uživateli vytvořit si vlastní rádio, které hraje písničky na základě uživatelovo preferencí. Toto rádio se dokonce učí, a postupně si vytváří profil uživatele.
Toto předpokládá analýzu existujících skladem. Vyhodnocení, které skladby jsou podobné a tudíž by se mohli uživateli líbit.

Další krok hned po analýze hudby, a to přímo generování hudby na základě daných preferencí, se dostává právě do popředí. Objevují se první pokusy o generování hudby na pozadí např. pro počítačových hry. Ale tyto techniky rozhodně nejsou běžné. Proto jsem se rozhodl se tímto tématem zabývat.

Přesněji zajímají mě techniky, které mi umožní generovat hudbu na základě daného vstupu.
V této práci se zabývám možnými postupy jak tohoto dosáhnout, a pak detailním popisem vybráno přístupu a jeho zhodnocení.

Abstract

Generative music are techniques exists for quite a long time. But by the time this thesis is written, generative music is coming more and more to the fore. Mainly because generating music is a much more complex topic than only rules for music generation. It is also techniques for analysing music and music structure description.

Algorithms for music analysis are quite developed. E.g. internet radio Pandora or Spotify benefits of music analysis. These web pages allow the users to create their own personalized radio. This radio play music based on user preferences. It is able to learn adjust itself for personal taste of its listener.
To be able to do that, understanding of which songs are similar is needed. From my personal experience the speed and accuracy of guessing is amazing.

These algorithms are quite developed, but what is the situation of real generating of music ? Right now, there are the first attempts to use generative music e.g. for background music in computer games. But generative music is still in phase of becoming a common part of our lives, it is literally trying to find it is own purpose.

This is the reason I chose this topic. I'm interested in the techniques of generation of music, especially how from given input get structure, which can be considered as music. In this thesis I'm going to probe common ways of music generation and I'm going to describe my own approach to generative music.

Content

1. Introduction

In this chapter is described what is considered as generative music. Why to be interested in generative music at all and what are different approaches beside approach in this thesis.

Our goal is to put down requirement for our application based on research of different solution.

1.1. Generative music

Generative music (abbr. GM) is a term created and popularized by Brian Eno nearly 20 years ago [0] to describe music, which is composed by a system. The borderline what can be called generative music and what not is unclear. In this thesis generative music is considered every music composed by a man, precisely music generated by computer software.

Generative music can by dived (according to Wooller, R. et al., 2005) into further categories by input.

a) Procedural
b) Behavioural

And according to output

c) Deterministic
d) Non-deterministic

Procedural GM creates music by sets of rules defined by music composer. It is direct composing music using computer.

Behavioural creates uses chosen system to map output of this system to music. Structured data can be from binary file to parameters of defined dynamic physical system. E.g. imagine interpreting colour as tone. Such a program could take a photo as as input, and play its colour . This was example of behavioural generating.

Deterministic mean, that it is possible to foresee output of the application. Non-deterministic is based on randomness and probability. For two same inputs significantly different answer can be returned.

Procedural GM is more deterministic than non-deterministic. Behavioural is exact opposite. The behavioural are non-deterministic.

In general terms Behavioural system can generate music easily, but the music is very random. It generates listenable music without very much effort, but it is really hard to improve this system to yield good music. The factor of randomness is too high. On the other hand it has balanced output quality. It is not expected that Behavioural system start playing bad suddenly. The constant quality of output is expected through the time. Procedural algorithms generate music via given rules. Rules can be specified as language grammar, or lots of if-else conditions or state machine etc. Different approach is using neural networks or other techniques involving adaptive learning. Output of Procedural GM is much more similar to music than Behavioural GM, but the quality of output is not balanced. The system can perform both perfect and terrible. In Behavioural GM there is literally nothing to spoil, in the Procedural GM is a very susceptible system. One error can disrupt the whole system because the rules are complex and needs to be chained together. So error in one rule is distributed through the whole application.

But the motivation to overcome these issues is fact that Procedural GM is able to perform very pleasant music and can perform much better than Behavioural systems.

This thesis is focused on procedural composig. The main reason is further development posibilities. From the point of output deterministic approach to a certain extent with non-deterministic features is preffered.

The final goal is to have a deterministic procedural music generator which on same input values is returning similar outputs.

1.2. Present state

Generative music is right now mostly used in the artists performances. It is used for an improvisation or as estranging feature. The reason why going to concerts of composing programs is not common is clear. The success rate of generating music is still to low. It is possible to generate music, but someone have listen to it and correct the program's mistakes. The use-case - using generating music as background music for e.g. a restaurant without any other control mechanism is not considered as a good idea.

It is required to postprocess generated music either by external audio editor or simply after listening to output to judge which output preserve, and which discard. Discarding is not problem, because one of main the advantages of generative music is simplicity of generating huge amount musical data. That is the reason why discarding is not problem.

The need of processing the generated data is why the generating music is not spread remarkably so far.

There are more approaches how to generate music. According to my research there are three main different categories of approaches right now.

a) Neural networks

b) Sequencers

c) Algorithmic composition

Neural network approach suffer from typical problem of AI. It need hight amount of learning cycles to be able to play something listenable. This issue much more complicated, because to be able to learn, result need to be able figured out if is right or bad. But how to evaluate result of generating music ? The most simple way is by listening to its composition. In case of neural network, that means listening to hours/day/months of (bad) music. It is clear that music output evaluation by score and fetching it back to neural network is needed. But evaluating music is task with equal difficulty as music generating (see Temperley et al. [1])

-

Sequencers is object, which mix music on pattern like structure of predefined short tracks. This is most common approach today, because the smallest unit can composed by human before. If patches are combined without connecting glitches the result is good. Next advantage is easy and comprehensible design. The main disadvantage is the need of having readymade patches and another disadvantage is the limited number of outputs. This approach have the best ration of the performance to difficulty of the implementation.

The most complex of approach is using Algorithmic composition. In the fact by writing Algorithmic music you are learning computer to compose music by itself. It consist of task as transferring musical theory rules into computer readable data, and task creating algorithms that generate music while respecting musical rules and even writing some basic sequencer to compose overall structure of composition. The main advantage is ability to create music unrecognisable from human-composed music. The biggest obstacle are the rules itself. Because music is art, so the rules are not such a strict as expected. So ideal algorithm will follow rules, but at same time it have to know when it can break them.

After consultation with my supervisor we agreed that neural networks networks are out of our scope. I preferred Sequencer-like design because of great ration of difficulty of implementation to quality of output ration, but supervisor advocated Algorithmic solution because of greater possibilities and more musicians approach.

Algorithmic solution was chosen as the most interesting approach for implementation, thus application will be generating the music directly by algorithms.

1.3. Motivation

Why generate music at all ? Nowadays there is a huge trend of sandbox games. Sandbox refers to endless possibilities, no constrains. The classic computer games used to had clearly defined goal (e.g. kill the dragon). In sandbox games the goal of the game is playing the game itself. To be able to provide players with such world, it is not possible to have static content. Generation of the content on the fly and even responding to players actions is need on fly.

Try to imagine a task, compose music for such a game. In classic game different short track can be scripted track, but in sandbox games there is nothing to rely on.

This create the need of some generative techniques. De facto there is not much more solutions how to do this (e.g. from player generated content). Techniques of generating content are quite advantage so far, but techniques of generating music are little bit left behind.

The goal is to able to create any number of music based on given definition. Like sorrow-burial music or joyful-parade music. Music have to be generated by given length, tempo, mood, and it needs to be synchronized with different player actions.

This concept can be adjusted even to real life as music in café or tearoom with music adjusting to current atmosphere automatically. Or having wearable mp3 player-like device for sportsmen which compose music by their current running tempo.

Generative music is still looking for itself but future seems to be bright.

1.4. Requirements

Ideal music generating application is able to compose music on the fly. On same result is returning very similar outputs. Variable level of control over program is required. Sometime the task is to compose music with very specific properties, sometimes program can be given more freedom, and specify e.g. only mood of the song. This is reason why output cannot be total random. This eliminates behavioural approaches.

The output must be music. It must be recognized as music by people, and it must be distinguishable compare to random sequence of tones.

The quality of music resulting from our algorithm need to fulfill several requirements - multi-voice melody, respecting voice-leading and counterpoint.

This is ideal application, requirements for such a programs are

1. System should be deterministic, with small factor or randomness.
    - outputs should not differ very much for same input
    - its possible to “force” system to create chosen input
2. System should generate output, which is evoking common music
    - output have to follow harmony rules
    - output have to create listenable melody, distinguishable against random melody
    - output have to have rhythm
3. Output is polyphonic
    - it consist of more than one voice
    - output have to follow voice leading rules
    - output have to follow counterpoint rules
4. Generated melody is played as MIDI
    - sending midi signals, so it is possible to postprocess output data
5. Application is generating music on the fly
    - streaming music, or generating music with enough speed

Such requirements are beyond extent of bachelor thesis. Implement must be reduced only to the core of the application. Some part of application must by simplified or omitted.

Not implemented part of application.

- Not generating data on the fly
- Advanced method of inputs
- GUI
- Forcing application to exact output
- Simplified music rules handling

Generating music on the fly is not recommended. This would create only difficulties with implementation.

And in application is not implemented any of the advanced method of inputting music structure data. So generated song is defined by classic musical terms as rhythm melody and harmony and not by terms as “happy”, “sorrow”, ”fast” end so on.

And the application will be console based. There is no need of GUI so far.

The forcing application to exact output is omitted for now as function which so far is not providing any additional value to the application and on the same time is difficult to implement. This functionality will be implemented after completing the whole project.

2. Background

In this chapter is introduces basics of the music theory rules, mainly for representation of music data.

If you understand music theory you can skip this chapter and jump right into chapter 2.3. There are written details about implemented musical rules.

2.1. Brief introduction into representation of music

Basic unit of the music is note and mark (silence). The length is represented by the note value (marks are using same system as notes).

Notes are placed on the staff (Fig. 1). Staff consist of 5 lines. Staff begin with a clef. Until the clef is defined, staff represents only relative pitch. There is more types of clefs, but two the most common are tremble clef - G4 and bass clef F3 (see Fig. 2 and Fig. 4).

When note is placed out of staff it is placed on the ledger line (Fig. 3). After clefs are chosen pitch are defined unambiguously.

1 = whole 2 = half 4 = quarter 8 = eighth

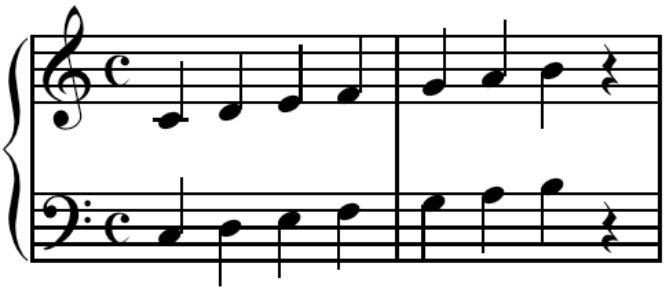


Fig. 1: Example of the staff with notes

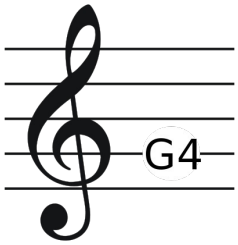


Fig. 2: G4 Tremble clef

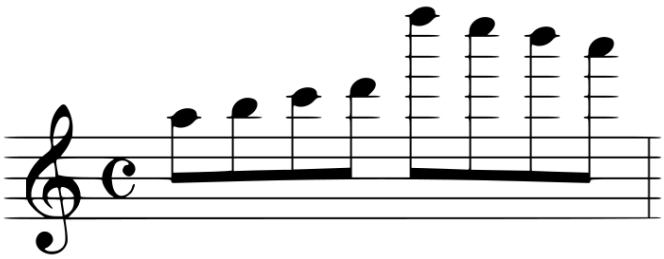


Fig. 3: Notes on the ledger lines

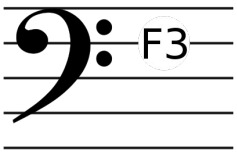


Fig. 4: F3 Bass clef

All used pictures are distributed under public domain

Note pitch is defined as letter from set [C, D, E, F, G, A, B]. In Czech Republic dominates German system, where instead of B is used H like [C, D, E, F, G, A, H]. English speaking countries is used system with B. In this thesis English system is preferred.

Note pitch repeat itself [… G, A, B, *C, D, E, F, G, A, B,* C, D, E …] tones are grouped into octave. Octave number represent which repetition the note it is. G3 is 3rd repetition of basic scale. So result look like this [… G1, A1, B1, **C2, D2, E2, F2, G2, A2, B2,** C3, D3, E3 …].

Octave number can be from 1 to 9. This naming system is called *numbered system*. Aside from this note can be represented as number (MIDI system) or as frequency. A4 corresponds to 440 Hz.

The numbered system is not very good for computer processing so MIDI system is used instead. In MIDI is note pitch is represent as number from 0 to 127. C4 is mapped to pitch number 60.

But another issue appears. Note pitch can be even augmented or distinguished. In MIDI notation it is easy, pitch number + 1 for augmentation and pitch - 1 for diminishing.

Augment pitches uses sharp notation " ♯ " (don't mistake it with "#"). Sharp notation increase original (called natural) note pitch by half step, diminished uses flat notation " ♭ " (don't mistake it with "b"). Flat notation decrease note pitch by one half step. In the end 12 note notation which repeats every octave is used.

0	1	2	3	4	5	6	7	8	9	10	11
C	C♯/D♭	D	D♯/E♭	E	F	F♯/G♭	G	G♯/A♭	A	A♯/B♭	B

Doubled notation exist too."𝄪" for double sharp, and "𝄫" for double flat. In the rest of the text I will be using "#" for single sharp (or "##" for double sharp) and "b" for flat ("bb" for double flat).

Double notation enable us to augment or diminished note by two halfstep. C## is equivalent of the D, Dbb is equivalent of the C.

We can return note to its natural pitch by using natural symbol "♮".

All symbol last until end of the bar or until natural sign is used. Symbol is

Similar advanced rules are even for note length. Note can be "dotted". Dotted note (note with dote above) have increased length by its length/2. Dotted 8th note is 1/8+1/4 time length. This is because sometimes note of the length different than power of 2 is needed.

Flat and sharp notation was defined as moving halfstep. When moving halfstep down from C resulting note is B. But when moving up halftstep up from C result is C#. Maybe so far it is not clear how halfsteps works but, after looking at Fig. 5 it will definitely be clear. Figure is showing movement by halfsteps in C Major scale. Movement in others scale is discussed in following chapter.

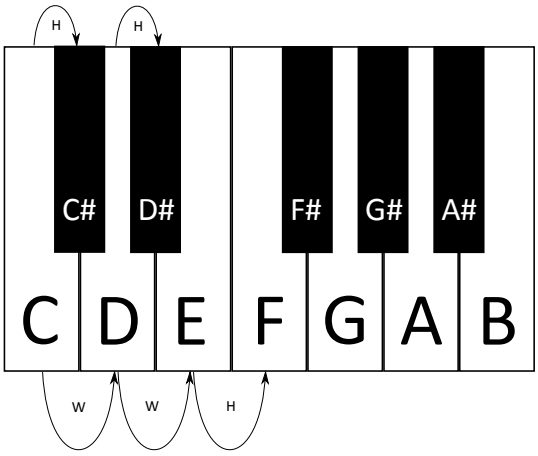


Fig. 5: Halfsteps on piano keyboard

Half steps are notated as H, whole steps (2 halfsteps) are W. Basic 7 notes are distributed in manner W W H W W W H (C Major) on piano keyboard. Half step is in fact only moving to next note (up or down). Movement by diminished or augmented pitch is simple, with exception of E - F and B - C where movement is directly next tone letter.

-

Staff is vertically divided into section called bars (or measures) of same time length.

Bar is segment of time defined by given number of beats. You have to define length of beat, usually length is equal quarter note.

Defining number of beats for staff with beats length is called the signature. It is represent as “fraction” like 2/4, 4/4, 3/4. Such description is read as “number of beats” to “beats note value”. E.g. 3/4 means 3 beats of note value 4. The forward slash is only visual feature, there is no fraction dividing.

Specifying number of beats and their length is important because of beat structure of the music. What is beat can be described in simple way. Play your favourite song, and try to tap into the rhythm. Tapping corresponds to beat structure of you favourite music.

E.g. 4 beats bar have usually structure like X - 0 - X - 0 where X means strong beat and 0 means weak beat.

2.2. Brief introduction into music theory

For purpose of our program it is required to understand the concept of scales, degrees, pitches, pitch classes and relationship between them.

At first repetition of classic notation of pitches for one octave from previous chapter.

```
0     1      2     3      4   5     6      7     8      9    10     11
C | C#/Db | D | D#/Eb | E | F | F#/Gb | G | G#/Ab | A | A#/Bb | B
```

Pitch class is similar representation to [C, D, E, F, G, A, B]. It is pitch modulo 12. Consequently pitch class can be from 0 to 11 and it represent 12 possible tones. Octave number can be neglected. So C4 and C6 have same pitch class = 0. G2 and G3 have same pitch class = 7.

This is important information, because relationship between notes repeats itself every 12 notes, it is possible to simplify our calculation by using pitch class.

Next concept scale. Scale is 7 ascending tones in manner of 2+2+1+2+2+2+1 half steps for major scale.
Scale can by build easily. Choose root of the scale. Root is first - basic - tone of the scale. Let's choose C. Than add 2 halfsteps. C → C# → D. Result is have D. Add 2 halfsteps. D → D# → E. Results is E. Add 1 halfstep. E → F etc.

Minor scales can be build with only small tweak in numbers, for natural 2+1+2+2+1+2+2 or for harmonic as 2+1+2+2+1+3+1.

Scale	Notes	Scale	Notes
C	C D E F G A B	C#	C# D# F F# G# A# C
D	D E F# G A B C#	D#	D# F G G# A# C D
E	E F# G# A B C# D#	F#	F# G# A# B C# D# F
F	F G A A# C D E	G#	G# A# C C# D# F G
G	G A B C D E F#	A#	A# C D D# F G A
A	A B C# D E F# G#		
B	B C# D# E F# G# A#		

Tab. 1 Example of major scales

Degree of Note (pitch) is index in scale + 1. Thus Degree can be 1 - 7. Degree is usually notated as Rome number I - ii - iii - IV - V - vi - vii (or uppercase). Notes outside of the scale are called chromatic.

If note is chromatic can still degree be chosen. Sharp and flat notation can be used for degrees too. For example ii# of scale C is D# (or pitch 63). Degree have names - see table Tab. 2. Subtonic is sometimes called Leading tone.

I	ii	iii	IV	V	vi	vii
tonic	supertonic	mediant	subdominant	dominant	submediant	subtonic

Tab. 2 Names of degrees

The purpose of degrees is simple, it is giving us more information about note. Information that not possible to calculate from the pitch in trivial way (see Temperley et al. [1]).

Example - tone G4 → pitch 67 → pitch class 7. If it is G in G major, its degree I (one) . From this information can be read that G in G major is basic tone and its degree is I (one). If it is in C major, its degree is V (five).

2.3. Music theory rules in application

These advanced advanced techniques are not considered for implementation so far.

- Dotted notes
- Double sharp and flat notation
- Note value only 1, 2, 4, 8, 16

For purpose of algorithm it is assumed that note values can be only from 1 to 16. This set covers most of common music [4]. Tones bigger length than 1 or smaller length than 16 are infrequent.

Dotted notes and double sharps are omitted to simplify the algorithm and because algorithm work with enharmonic substitution.

Enharmonic substitution means neglecting differences between C# and Db (augment first note and diminished second tone), or simple said using MIDI note pitch system - pitch is number from 0 - 127 with middle C pitch = 60.

These techniques are omitted to simplify the implementation. Priority of adding these rules into implementation is in order note values, dotted notes, double sharp notation, enharmonic subs.

The rest is described in next chapter design.

3. Design

In this chapter are designed structures for work with music structures. Afterwards there is proposal of main - music generating - algorithm. In the end of the chapter there is list of the excepted issues and final summary of chosen design.

Right design of the Procedural Algorithmic generative music program is able to follow music theory rules, and it is be able to handle work with data from proposed music data structures.

Design have to take this requirement into consideration. Only with correct work with music structure it is possible to implements the core of the application, the algorithm for composing music.

As programming language is preferred Java. After consulting with my supervisor Windows was chosen as target platform.

3.1. Design of musical structures representation

Basic unit of music is the Note. Bar can't be leaky. Mark of defined length is putted in bar in place where should be silence. Thus next basic building element is mark. Note length is saved in two form. Note value and real note length. Note pitch uses pitch representation same as MIDI.

The third element is ligature. Ligature is two notes of same pitch linked together and the second note is not played.

Bar are defined by signature. Number of beats - signature beats, and beats length - signatures length. See Fig. 6, black dot is start of the note and white note is end of the note. Bars are linked together into bigger structure - called Phrase.

Phrase is overlay of bars, so it is possible to add as many tones as like without caring about note overflowing. This will be allowed by holding information, which index of Bar (and which bar) is filling right now. After adding note into bar index have to shift in such way, that next note will not collide with previous. Aside from adding note and shifting adding index by note length (real) it should be able to add note to specific index to support chords.

Phrase and Bars have to be able to return organized content of. Organized the way our MIDI player will be able to read (and play).

Phrase have stable signature. To change this parameter, another Phrase need to created. The highest structure called Track link Phrases together. It allow using phrases with different signature.

Diatonic Scale is defined as array of 7 elements. Computing simple chromatic values from given notes is simple.

Program have to be able to convert between pitch <-> pitch class <-> degree in scale [I-II-III-IV-V-VI-VII] <-> index of degree [I I# II II# III IV IV# V V# VI VI# VII]. To be able to do the tasks like “return pitch of the next degree of the note with pitch class X”.

Last thing needed is output structure. Result need to be organized. There are more possibilities. It is possible to represent result as list of tones sorted by start time, or preserve more structural data when representing result as several list (again sorted by time) but different voices will be in different list. The fact having different voices stored separately, it is possible to play different voices with different instruments (but such a functionality isn't planned yet) .

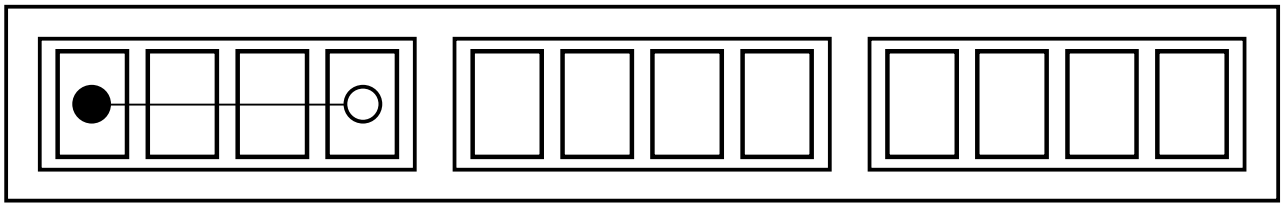


Fig. 6: Structure of 3/4 Bar with one Note of real length 4

3.2. Matrices of probabilities

Main idea of generating melody is matrix of probability. Matrix of probability is matrix with size 12x12. It correspond to 12 possible degrees of scale. Matrix can be viewed as 2D position system. Columns correspond to X-axis and rows to Y-axis. Matrices are save in list, that can be viewed as third dimension - Z. Z corresponds to time.

Whole matrix can be viewed in image Fig. 7. In picture it is used indexing from 0. So first degree is on the index 0 and last one on the index 11.

In every cell of the matrix is stored number. This number represent probability. Probability that degree X, is moving to degree Y. Z represent time in bar. That means after creating new tone by cycling through matrices, the one corresponding to actual time have to be found.

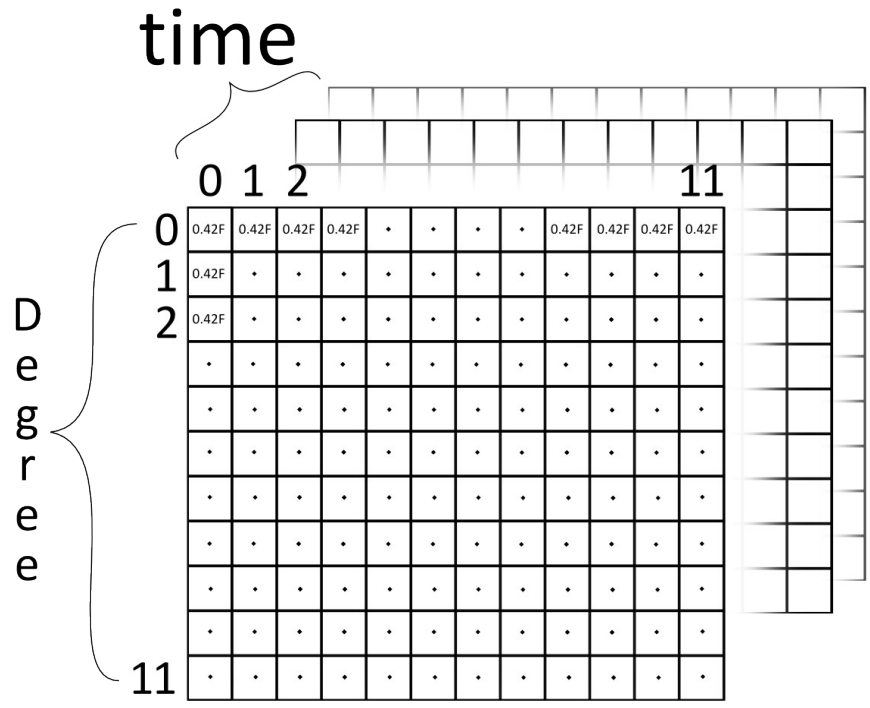


Fig. 7: Matrices of probabilities

Such a matrix can vastly simplify generating process. The next tone is determined as in simple way as Matrix[X][Y] where X is current note degree, and Y is next degree. Afterwards the next task is finding maximum value for Matrix[X][1 … 11]. After maximum is found degree have to be converted to pitch.

The purpose of having multiple matrices is because existence of something called beats structure. Every song have invisible structure, representing rhythm. Easiest way of finding beat structure of chosen composition is to play it and a parson is given task to tap into rhythm. Such a tapping represent one level of beats. The higher is beat level the more general and regular structure it is. And there is longer period between beats [1]. The biggest, the longest beats are called strong beat. They represent overall structure like beginning of bar. Weak beats are more specific, and they follow rhythm of melody.

The algorithm have to respect such a structure by having different reaction of algorithm. Algorithm have to check, if current position in bar is on strong beat of weak beat. In general term significant changes are recommended on strong beats (such as changing harmony) [1].

The algorithm is trying to handle strong / weak beats by preferring some of the degree. On the strong beats are preferred degrees I, III, V. Rest are less common II, IV, VI, VII. Chromatic degrees like II# are the worst [8]. Chromatic degrees are evaded on strong beats.

Disadvantage of using matrices of probabilities is the need to get data for matrices. Getting data for whole composition will probably be problem. But finding data about moving from degree to degree in different part of bar will be much more complicated. I don't expect such a data will be available in computer readable format.

This leads, to need to generate such a matrix from existing songs. But again, to get piece of music in computer readable format with specified harmonies is very hard. MIDI represent only pitch. And analysing harmony and degree from set of pitch data are task many time harder than is whole proposed generating of the music.

3.3. Sound output

As output of application is used MIDI. MIDI is able to play prepared track. After generate music, conversion to format readable by MIDI is needed. Thus it is preferable to play notes sorted by start time. Testing connection with external devices is not planed. This is considerable in later phase of development.

For purpose of testing project the MIDI messages must be received and played. For this purpose is used external program creating virtual MIDI port - LoopMIDI by Tobias Erichsen.[1]

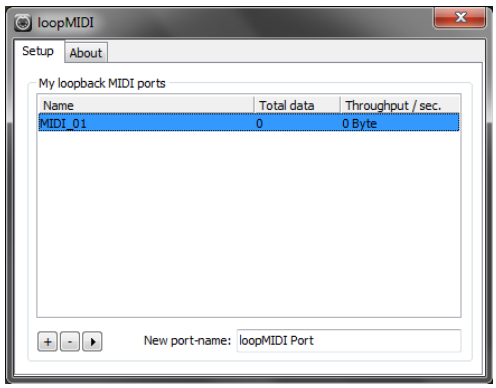


Fig. 8: LoopMIDI minimalistic interface

1 Available online at: http://www.tobias-erichsen.de/software/loopmidi.html

3.4. Voice-leading and counterpoint

Voice-leading and counterpoint rules defining creating a handling multi-voice harmony. Matrix of probabilities can be used to generate second voice too, but such generating cannot be done separately. Pitch and degree of top voice have to be taken into account.

Algorithm have to evaluate every note from point of voice leading and counterpoint, and chose best suiting note from both view of probabilities matrix and voice leading.

Double voice composition is assumed. At first top voice is created. Than bass line - bottom voice will try to fit itself into composition following the most rules it can.

3.5. Expected issues

First problem will be converting note value to the real note length. Note value can be represent as number from set [1, 2, 4, 8, 16]. But note length differs based on bar's signature length. With signature length 4 note value is 1, 2, 4, 8, 16 length 16, 8, 4, 2, 1. But with signature length 8 th length is 32, 16, 8, 4, 2.

And when comparing, note values 2 > 4, 4 > 8, 1>2 etc. Real length can be easily represented as fraction but after consulting with my supervisor, this is not recommended. Note length should be represent with integer number.

Bar have defined length too. It defines the biggest sum of real note lengths, which can fit into bar. Overflow is caused when note that is longer than remaining space is putted in bar.

When overflow occurs, note have to be divided into two notes. The first fit into rest of the bar (it is length is setted to be same as remaining space) and the seconds length is setted as difference between original note length minus remaining space. The second note must not be played. It is only placeholder. Mark (silence) can be used as second note, but from view of music theory, pair of notes is correct solution. This pair of note is called Ligature.

Another problem will be adding more tones into same time to create chord. Note have to be putted at exact time in the song (not trivial problem), and collision have to be treated.

Next question is how to handle moving storing index - pivot. Pivot cannot move when adding individual notes of chord, but after last note is added pivot should move by length. That is next problem, when adding chord with notes with different length which length pick.

These question have to be solved during implementation.

The last question is how to handle matrix of probabilities. If there will be no appropriate data some kind of workaround will be needed.

4. Implementation

This chapter is giving information about final implementation of defined task. At first there is description of component (aiming at music structure implementation). Than both voice-leading and melody (harmony matrix) are discussed and described. In the and, there is overview list showing music generation walk-through.

4.1. Chosen platform and programming language

As target platform was chosen Windows 7. Java is used version

IDE is used NetBeans 7.4. Java JDK used 1.7 version. For better portability of the application to other platform I decided to not use any library, and used clear Java.

4.2. Component description

Application is made of four big parts. The first is musical structures. These structures are used for storing data and information needed for data processing. These structures are mainly Note, Bar and Phrase, which represents many of bar lined together.

Another part is MIDI player. This part is used only for testing output. There is no ambition to create complex MIDI player. Only requirement is to be able to play created data.

Next part is evaluators. Evaluators are basic manipulators with music. Evaluators usually create vector of values, which represents probability of moving tone from degree to other degree. Some of them are directly editing structures (e.g. when generating second voice).

Last part is system for linking and generated music by evaluators. Basic element is called Unit. Unit are processed independently. These Unit create chain of Phrases. Output of this chain is used as input in the MIDI player.

Rest of application are various utilities tools, for better work with music data.

4.3. MIDI player

Data entering into MIDI player are in form similar to ArrayList<Note> (read as array list full of objects Note). MIDI player simple cycle through this list a play individual notes.

It stores notes into MIDI own structure called track, and after all notes are putted in, the track is played.

There is second variant of MIDI player, which can play more complex structure - Two voice input. Program should be able to choose correct MIDI player according to input.

4.4. Music data

Are stored in various way. It was mentioned above. From top level Track to Phrase to Bar to Note.

Note store data about its note value. It holds information about in which scale it belongs. After inserting note into Bar it is possible to calculate note real length and start time towards bar beginning. Note is represented as start time and length.

There is solution where note is represented as start time and end time. This is quite significant difference. In variation with note defining as start time and end time. Note is stored through whole its length.

There is possible to know which notes are playing right at specific time. But for simplicity of the program, Notes are holding information only about start times and note length.

So task what notes are playing at specific time means iterating back in time and checking previous notes. But so far such a task is not needed (Program can check for start notes).

Notes can return if there are on white key or black keys. Not chromaticity, but real white or black keys. This is valuable information for rendering data.

The algorithm is this.

```
return (pitch - Math.round(
2.3975F * Math.round((pitch) / 2.403996266F + 0.5829F) - 1.5586F)) == 0;
```

At first glance it looks queer. But position of such notes are not evenly spaced.
So graph of these positions was created and interspaced by linear regression function.
After some while and bruteforce of trial and error testing this formula was made.

Comparing with other array could be used. But solution without cycling thought the array was preferred.

Bar stores note. Bar are defined by signature length and signature beats. Signature beats represent number of segments. Signature length represent segment length. But bar must be divided into more segments than just signature beats, to fit fast notes. Thus total number of time segments is calculated as signature beats * signature length. Remember that length is represented as note value. So this calculation is not correct with using note value 2 and 1. (Whole note and Half note).

When handling such signature multiplying number of segment by chosen constant is needed. If not, there is not enough segment to fit in 1/8 and 1/16 notes.

When adding chord to bar, pivot (current time representation) is moved by length of the first note.

-

Phrase stores bars. Its main purpose is handling note overflowing. When note overflow original note is putted into bar (this is because MIDI, for MIDI overflowing note is irrelevant. Overflowing is handled only because right pivot setting) and into next bar is added ligature, which behaves as note of same pitch and has length defined as original note length minus remaining space in previous bar. Ligature is marked as non playable event. To be precise first note of ligature - original note is played and second note which is type of Ligature is not played. Thus when MIDI encounter ligature, it is skipped (but original note is still played).

Phrase is also setting real length of notes. Because notes should be added into bar through Phrase. Phrase have to set correct note length according to bar's signature.

```
int noteLength = (sign_length * sign_length)/ n.getNoteValue();
```

Algorithm for creating ligature is

- get real note length
- if length > than remaining space in bar
  - if remaining space is 0 - create new bar and add note
  - else
    - into last bar add note
    - create new bar
    - add ligature into new bar (length of ligature = note length - free space)
- else
  - if bar is empty create bar and add note
  - else add note to bar

After adding note to bar from free space is deducted real note length.

Common task for Phrase is getting all notes which starts at given time. Corresponding bar is inspected and list of notes at given time is returned. (All notes which starts at given time)

Last part of our music data structures is structure called harmony. It is used for computing note degree. In fact it is simple scale stored in array, with methods for manipulation.

We need to compute with 12 different degrees but how to represent chromatic degrees ? Implemented solution represent chromatic degrees using negative flat notation.

```
degreeArray = new int[]{1,  -2,  2,   -3,   3,   4,  -5, 5,  -6,  6,  -7,   7};
                        I  bII  II  bIII  III   IV  bIV  V  bVI  VI bVII  VII
```

This enable going into next degree as degree +/- 1. And going from chromatic degree into natural degree as simple as *(-1) for halfstep up and *(-1) -1 for halfstep down.

4.5. Units

Units are high structure of music. Each unit stores one phrase. Each unit is identified by its name. Name is simple string, usually one letter.

Units are stored in UnitFactory. Unit factory stores units, and keep records about inputted harmonies, melodies, and rhythms. When new unit is added into Unit factory, copy of the unit is stored in UnitAlphabet.

UnitFactore takes as input String called sentence. Sentence is sequence of letters, where each letter represent one unit. Example “AABAABACABAC”. When analysing sentence UnitFactory cycle through sentence, a look into UnitAlphabet for corresponding unit. If none unit is found, UnitFactory creates one. This is place, where music is generated from the scratch. UnitFactory also build unit based on given information like Harmony, Melody, Rhythm. Interesting feature is ability of UnitFactory to recycle generated content. With some low probability instead of creating new unit, UnitFactory can build unit from available content.

Processing of sentence:

- `for sentence length get letter on index i`
  - `if alphabet contains letter`
    - `build unit accordion to available data`
  - `else (alphabet don't contains letter)`
    - `pick random harmony from available list`
    - `pick random beats from available list`
    - `compose unit`

When build unit the Harmony Melody and Rhythm is needed to be given. When composing whole new unit, melody is composed based on chosen rhythm and harmony. Rhythm is expressed as array with number from 0 to 2. Where 2 is strong beat, 0 is weak beat and 1 can be described as “less strong”.

Detailed description of composing algorithm is in next chapter.

Unit stored in UnitFactory are two kinds. Data Unit which is Unit filled with data (no generating). Nothing else. Template unit, which stores information as beat structure, melody (sequence of degrees), tempo etc. When invoked this Unit build itself and fetch data in Unit Factory. Or unitd generated from UnitFactory. The result of generation is Data unit.

4.6. Evaluators

The first evaluator is Harmony evaluator. Its purpose is evaluating transition between different degrees. It is corresponding to Matrix of probabilities from chapter 3.2.. Harmony evaluator have stored matrix 12x12 with values representing transferring from degree to degree [3].

```
                    0       1       2       3       4       5       6       7       8       9       10      11
      //            I       bII     II      bIII    III     IV      #IV     V       bVI     VI      bVII    VII
      /*0 -  I   */ {0.000F, 0.027F, 0.121F, 0.004F, 0.016F, 0.176F, 0.008F, 0.453F, 0.043F, 0.066F, 0.012F, 0.074F,},
      /*1 - bII  */ {0.200F, 0.000F, 0.533F, 0.000F, 0.000F, 0.000F, 0.067F, 0.133F, 0.000F, 0.000F, 0.000F, 0.067F,},
      /*2 -  II  */ {0.222F, 0.030F, 0.000F, 0.010F, 0.040F, 0.010F, 0.071F, 0.455F, 0.020F, 0.081F, 0.000F, 0.061F,},
      /*3 - bIII */ {0.100F, 0.100F, 0.000F, 0.000F, 0.000F, 0.000F, 0.000F, 0.400F, 0.400F, 0.000F, 0.000F, 0.000F,},
      /*4 -  III */ {0.053F, 0.000F, 0.105F, 0.000F, 0.000F, 0.368F, 0.000F, 0.053F, 0.000F, 0.368F, 0.000F, 0.053F,},
      /*5 -  IV  */ {0.471F, 0.029F, 0.147F, 0.000F, 0.059F, 0.000F, 0.044F, 0.162F, 0.000F, 0.015F, 0.015F, 0.059F,},
      /*6 - bV   */ {0.438F, 0.000F, 0.000F, 0.000F, 0.000F, 0.000F, 0.000F, 0.562F, 0.000F, 0.000F, 0.000F, 0.000F,},
      /*7 -  V   */ {0.848F, 0.000F, 0.041F, 0.005F, 0.010F, 0.020F, 0.000F, 0.000F, 0.036F, 0.030F, 0.000F, 0.010F,},
      /*8 - bVI  */ {0.192F, 0.077F, 0.308F, 0.000F, 0.038F, 0.115F, 0.000F, 0.077F, 0.000F, 0.115F, 0.077F, 0.000F,},
      /*9 -  VI  */ {0.093F, 0.047F, 0.651F, 0.000F, 0.023F, 0.093F, 0.047F, 0.023F, 0.000F, 0.000F, 0.000F, 0.023F,},
      /*10- bVII */ {0.000F, 0.000F, 0.000F, 0.833F, 0.000F, 0.000F, 0.000F, 0.167F, 0.000F, 0.000F, 0.000F, 0.000F,},
      /*11-  VII */ {0.818F, 0.000F, 0.000F, 0.000F, 0.091F, 0.000F, 0.030F, 0.030F, 0.030F, 0.000F, 0.000F, 0.000F,},};
```

The only problem is often converting from degree to degree index. Because for example degree V is at index 7. Probability for e.g. going from degree III to V is

```
int orig_index = Harmony.getIndexByDegree(3);
int other_index = Harmony.getIndexByDegree(5);
return this.harmonyProb[orig_index][other_index];
```

Another quite common task is finding best degree X, when degree start - X - degree final will be maximal value. Or by word finding best transition from degree start to degree final through certain degree. The best value can be returned or whole array storing transferring values as

```
float [] res = new float[this.harmonyMatrix.length];
for (int i = 0; i < res.length; i++) {
      res[i] += this.harmonyProb[orig_index] [i];
      res[i] += this.harmonyProb[i][other_index];
}
```

Evaluators are usually using array of float of length 12 for exchanging data. Such array represent probability of moving from current degree into 12 different degrees. For array of float will be used term vector (from JAVA structure).

-

Rhythm evaluator is choosing right note length for current time. The key factor for this task is position in beats structure. Rhythm evaluator is evaluating degrees into three bad, good and perfect. Strong [I - III - V], medium [II - IV - VI - VII] and weak are all chromatic degrees [8]. Rhythm evaluator is returning longer notes for strong and medium harmony on strong beat. Normal length for strong or medium harmonies for the rest of the beat levels. When weak harmony is encountered on strong beat (program will try to evade this situation) at least its length is shortened, so the impact factor is decreased.

Melody evaluator is trying to prevent cycles in decision chain. Because from point of Harmony evaluator the best transfer from I is V or I. And the best transfer from V is V or I. This implies that algorithm have to remember previous degree position and evade cycling forever in few most probably degrees.

Dividing into three quality of degrees is used [8]. See Fig. 9. There is simple sketch of moving from different state of melody. S stands for Strong, M for medium, and W for weak. Discontinuous line indicates that there is lesser probability of moving into weak state. On the other hand, weak state prefer moving into strong state, with less probable move into medium state. This way is ensured cycling through different degrees of harmony.

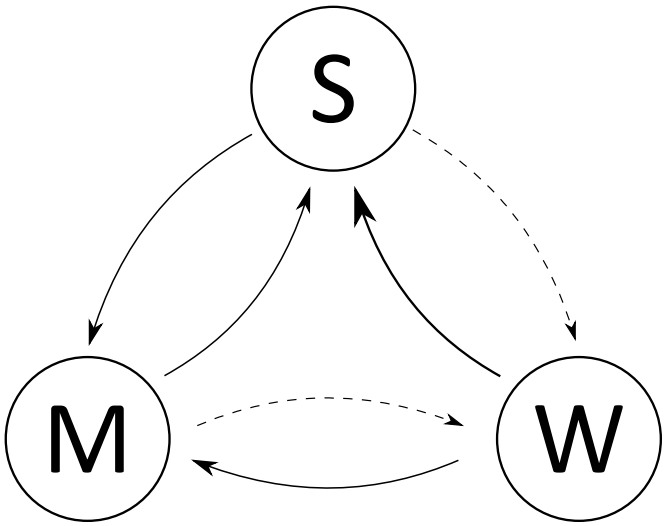


Fig. 9: Degrees cycling state machine

The input is array of float. The result is same array of float but modified by Melody evaluator. Degrees which are preferred to move are left untouched. The rest is divided, to decrease of chance cycling.

-

Voice leading evaluator is used to determine direction of voice movement. The probability of ascending melody and descending melody from chosen tone is not equal [4]. Unfortunately only data for degree I to VII was obtained. So for chromatic values is used 50:50 ascending to descending ratio.

This evaluator is using unusual array of floats. It represent probability of moving to pitch up and down. See Tab. 3. In second column you can see how is output of evaluators represent. In both cases in array of float is stored probability. But indexes are shifted. By default is index = 5 taken as movement to same degree. Index 0 - 4 is descending movement, index 6 - 11 is ascending movement. To sum this vector with output of harmony evaluator, these array have to be joined correctly. At first index is current degree is computed. For example assume index 7 (degree V). Thus it is needed to shift whole vector +2 index to match output of harmony evaluator.

The input is harmony vector, the result is modified vector. After decision is made opossite direction indexes are divided by constant, and chosen direction id multiplied. First few tones are multiplied strongly but the rest is inhibit too.

Because Voice leading prefer moving in close steps (mostly into next non-chromatic degree) .

I	Pitch - 5	**I**	**Pitch + 5**
bII	Pitch - 4	**bII**	**Pitch + 6**
III	Pitch - 3	III	Pitch - 5
bIII	Pitch - 2	bIII	Pitch - 4
III	Pitch - 1	III	Pitch - 3
IV	**Pitch + 0 ← constant movement**	IV	Pitch - 2
bV	Pitch + 1	bV	Pitch - 1
V	Pitch + 2	**V**	**Pitch + 0 ← constant movement**
bVI	Pitch + 3	bVI	Pitch + 1
VI	Pitch + 4	VI	Pitch + 2
bVII	Pitch + 5	bVII	Pitch + 3
VII	Pitch + 6	VII	Pitch + 4

Tab. 3 Voice leading vs. Harmony output arrays of float connection

This is reason why tones are inhibit even in chosen direction is big distance in pitch against original tone. From point of voice-leading small steps are preferred. When constant move is chosen both direction are inhibited and only constant movement is multiplied.

-

Second voice evaluator is creating second (bass) voice. Evaluator is cycling through top voice and following data are needed. Previous top note, actual top note, previous bass note, harmony and actual beat power. Three tones are needed to evaluate direction of voice movement. This will allow to manage contrary motion of voices. See Fig. 10 *for example of contrary motion.

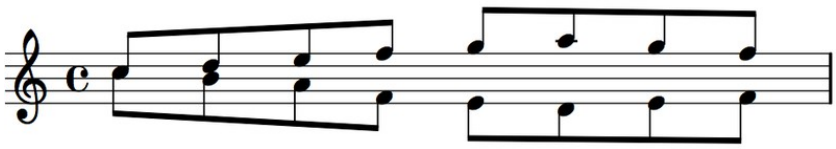


Fig. 10: Example of two voices in contrary motion.

Top and bottom voice are moving in opposite direction. This leads to highlighting of each voices.

Another set of information is length of top note, with beat level. Counterpoint rules have to be followed. Because of that algorithm analyse current beat level.

* cc Memoryboy (9 June 2008) http://en.wikipedia.org/wiki/File:ContraryMotion.png

Second voice is adding notes only on strong beats time (2 or 1 beats on 1 bar). In weak beats, nothing is added. In middle beat (when using notation that 2 is strong and 0 is weak middle is 1) is added note with some probability. Lower level of beats are less regular [1]. This will emulate such behaviour. Algorithm is looking for places where counterpoint rules can be applied, when top voice is playing fast notes bottom voice is balancing this by playing notes only on strong beats. When top voice is playing long notes algorithm switch and starts adding fast notes into bass.

4.7. Final algorithm generating music walkthrough

- Get starting note (randomly generated from harmony)
- get Harmony evaluator output
- get Voice leading output
- join both outputs together, weight each one by chosen coefficient
- normalize
- get Melody evaluator output
- normalize resulting vector to values from 0 to 1
- search for every value bigger than constant (0.5f)
- store indexes of this values
- pick one of the chosen values by random
- save chosen degree into variable

The code of this algorithm is available on attached CD/DVD, or see chapter 7.3. on page 35.

More detailed:

- input data: random Scale, random beat structure, name, length

Beats are represented as array of integers. The bigger the integer, the stronger the beat. Length represent length of generated unit.

- with chosen probability recycle generated melody

The chance is now setted to 20% (for purpose of testing). When using algorithm higher value is preferred. Music repeats itself quite often, so program will have to emulate such behaviour.

- get random starting degree (based of given scale) [5]

Starting degree is non-chromatic degree of given scale.

- Create storing structure Phrase from initial data
- while cycle until pivot is bigger than length

Pivot is position in time.

- evaluate rhythm → set note length

We prefer longer notes with matching strong beat good harmony. When chromatic note on weak beat note is made shorter.

- evaluate harmony and voice leading movement → result are two vectors
- join vector while using given weighting coefficients
- normalize result
- into melody evaluator insert result → evaluate possible move from point of melody [8]

Check from which degree (good, normal, bad) are origin of movement.

- normalize result
- every value of result bigger than 0.5F add into best match list

Only reasonable result are taken in account.

- pick one result from list by random number

To provide variability even on same input best result is chosen randomly.

- repeat

5. Conclusion

In this chapter is conclusion of whole project. At first evaluation of application, than found issues. Afterwards is short chapter about testing the project.

In the end is are proposed changes to design of application.

To sum whole development realization was much more complicated that excepted. Every steps of developing was heavily paid by time in doing research and by time studding music theory.

The implementation itself wasn't such difficult. Only unpleasant surprise was time consumed in building structures for work with music. Design generating algorithm was expected to be developed very quickly (from very beginning), but implementation of Notes, Bars and different evaluating matrix took longer time to develop. With combination with time spend on research, the whole project was very time consuming.

Right now, project is in phase of testing. According to WordCount plugin in Netbeans application has 5k lines of code. And With Unit test the border 8k lines of code is attacked. After testing and removing unneeded code the resulting number will be around 6k - 3k lines of code.

5.1. Real application behaviour

Application handle generating two voice output quite well. It is doing too much skip compare to steps, but this is matter enough testing and setting variables. Quite well is functioning rhythm evaluator. Even such simple algorithm create surprising nice rhythmical structure.

Harmony evaluating work as expected. Most of time it can “hit” right degree, but there are still bad sound from time to time. Mainly created by bad handling of chromatic notes.

Voice leading works similar way as Harmony evaluator. It handle it is job, but it need more tweaking to cut down bad pitch movement.

Melody work better than expected. It stopped application from infinite looping between some degrees. More complicated state machine would definitely help application to get more depth in harmony cycling.

Second voice is function quite well. This part can be considered most difficult. It follows specified rules, but decision algorithms are still simple so they need further development.

MIDI output work correctly. Same for musical structural.

Some flaws of design and implementation was found, and these are mentioned in chapter 5.2. Found issues.

5.2. Found issues

At first Design level issues. The performance of musical structures is very good. They can handle generating music as expected. The weak part of the chain is work with multiple voice.

From beginning was Phrase designed to contain chords. But while generating and evaluating music, Phrase with only one note at one time came as much better solution. Again multiple voices creates problem with exporting music to MIDI player. Thus joining two Phrases together is definitely desirable which could simplify the MIDI player.

Another possible issues is dividing bar into time segments. Formula sig. beats * sig. length is not working well for signatures with signature length 2 and 1. There is need to divide such case into more segment. But this would brake many of time based calculation. I was working mainly with 3/4 and 4/4 signatures so I have not encountered this problem. For further development support of all possible signatures is needed.

Another design flaw is not constraining average pitch of generated text. Generated melody sometimes ascend (or descend) more than 2 octaves. Present solution is relying on statistic to keep melody in reasonable pitch values, but this isn't enough. Something like gravity to certain pitch is needed to implement. But these cases are very rare, so this functionality do not have high priority.

The last design flaw is inconsistency of evaluators. Abstract interfaces for such classes have to be created to be able to maintain easy extendibility.

From general point of view, in application is right now in phase of testing, so founded errors are being removed. Because of lack of time, application do not have enough protection against wrong input. Imputing correct values is assumed. Correct handling of bad inputs are needed, especially if project will be developed by more than one people.

5.3. Accomplishment of the requirements

The requirement was accomplished. Application have stronger and weaker links but overall performance is promising.

5.4. Approach conclusion

Using matrices and procedural generating music is viable way of creating generative music. Main disadvantage is, without balancing probability matrices there is no clearly preferable solution.

This lead to interpreting resulting vector as recommendation like “It is recommended to move into this degree”. This system clearly need some supervisor. Something which made final decision. The performance of the application is quite good. Based experiences with application I believe, that on fly music generating is possible.

5.5. Proposed change and future development

My proposed change is to create State machine to create melodies. This state machine will use probability matrices to evaluate best movement, but it will fulfill role of its supervisor to make final decision with longer-term plan.

Probability system tend to be good in short playing windows, but in longer track it is clear that song is lacking enough structure. Just linking good melodies together does not yield good music as output. Proposed state machine could hold information about longer-term relationship, so this could improve phrasing and overall inner structure of composition.

Another adjust recommended by me is implementation of some learning (e.g. genetic algorithm) principles. State machine can solve most of problems, but application as is lacking of memory as it now. Learning principles could improve generating new melodies and rhythm costomized to application present needs.

6. Bibliography

[1] TEMPERLEY, David. The cognition of basic musical structures. Cambridge, Mass.: MIT Press, 2001, xvi, 404 p. ISBN 02-622-0134-8.

[2] CHEN, Chun.Chi J. a Risto MIKKULAINEN. Creating melodies with evolving recurrent neural networks.

[3] TEMPERLEY, David. A Statistical Analysis of Tonal Harmony. A Statistical Analysis of Tonal Harmony [online]. [cit. 2014-05-23]. Dostupné z: http://theory.esm.rochester.edu/temperley/kp-stats/index.html

[4] ELOWSSON, Anders. Statistical Analysis of Vocal Folk Music

[5] MARVIN, ELIZABETH WEST a DAVID TEMPERLEY. PITCH-CLASS DISTRIBUTION AND THE IDENTIFICATION OF KEY.

[6] CHAFFIN, Lon W. Music theory notes. [online]. [cit. 2014-05-23]. Dostupné z: http://www.lcsproductions.net/MusicTheory/MusThry/TheoryNotebook/pages/1.html

[7] COMMON-PRACTICE TONALITY: A Handbook for Composition and Analysis. [online]. [cit. 2014-05-23]. Dostupné z: http://www.dangutwein.net/courses/theorytxt/text/4-part.htm#top

[8] JOUTSENVIRTA, Aarre a Jari PERKIÖMÄKI. Tonal degrees and degree tendencies. [online]. [cit. 2014-05-23]. Dostupné z: http://www2.siba.fi/muste1/index.php?id=63&la=en

[9] HURON, David. A Derivation of the Rules of Voice-leading from Perceptual Principles. [online]. [cit. 2014-05-23]. Dostupné z: http://csml.som.ohio-state.edu/Huron/Publications/huron.voice.leading.html#Rules

[0] COLLINS, Nick a Andrew R. BROWN. Generative Music special edition, editorial.

7. Appendix

7.1. Content of the CD

On the CD is source code of whole project in format of Netbeans project in folder “Source”. In the root folder aside from folder “Source” and “Samples” is PDF with this thesis and editable file in open document format (.odt).

In folder “Samples” are generated samples of music from program. The samples are generated with different parameters.

01-03 Standard	Defualt setting of the system
04 Low finesse	The minimal limit for considering tone as valid move risen from 0.5F to 0.2F. Limit can by from 0.0F (every tone) to 1.0F (only one the best tone).
05 Hight finesse	Limit setted to 0.8F. Program is choosing from less options.
06 - 07 Inc second voice	Coefficient for adding second voice is risen from 0.5F to 0.8F. This coeff. Increase number of contemporary tones in top and bottom voice by increasing number of bottom voice notes.
08 desc SKIP	Prefer moving to next non-chromatic degree. This leads to worse harmony, but better voice-leading.
09 desc STEP	Prefer good transition from degree to degree at the expanse of bigger leaps in melody
10 - 11 Dec second voice	Coefficient decreased to 0.2F. Bass is playing less tones than top voice.

In folder “Samples” is file UserTesting. There are stored data from testing generated samples on different people. The order of samples was mixed by random sequence. Users was tasked to sort heard melodies from the best to the worst.

7.2. User manual

Program is stored in Jar file. To run Jar you need to have latest Java installed. To run the project open command line, and open directory where is Jar stored. Then type “java -jar composer.jar” to run project.

Program input is very simplified only to enable setting of following variables or to generate composition with default settings.

First available variable is Second voice regularity. In creating second voice, program generate second voice only when on strong beats. On weak beats program leave silence. On middle beats - beats which are not strong and not weak there is no strictly defined what to do. So if random number (from 0 to 1) is less than Second voice regularity, second voice note is generated. If random number is bigger, again nothing is done.

This creates irregular structure of beats while maintain regular structure on strong beats. By setting this parameter to 1 algorithm always add second voice (bass) note when program is on middle beat. By setting it to 0 program is adding notes only on strong beats.

Next is Skip / Step factor. When devising where to move there are two opposite opinions. Harmony (transition from degree to degree) prefer jumps by pitch 3 (remember good harmony I-III-V).

But from point of voice-leading movement by small steps is preffered. (I → II, II → 3 etc) . Different opinion can be weighted by value from 0 to 1. When choosing Skip 1 harmony decision vector is multiplied by 0.1F. When choosing 10 vector is multiplied by 1.F (default value). The result is that application should start to move by smaller steps than before. By choosing Steps 1 application proffer good degree transition at the expanse of bigger leaps.

7.3. User testing

Application enable setting different variables. Such a setting was experimentally tested on three (mostly) volunteers. This is too small sample to provide relevant statistic data, but samples with decrease skips and the first increased second voice regularity achieve best score.

About methodology of testing, users given task to order heard samples from the best to the worst. Order of samples was changed randomly.

After collecting more data, it is possible to prove is setting above mentioned parameters can really achieve better feeling of generated music.

7.4. Complete decision algorithm

```
public Unit_Data composeUnit(Harmony h, int[] beats, int recommendetLength, String name) {
        if (Math.random() < 0.2F && melody_list.size() > 1) {
              Unit_Template tm = new Unit_Template("name", h, this.pickRandomMelody().getMelody(),
               beats);
            return this.buildUnit(tm);
        }
        RhythmEvaluator rh = new RhythmEvaluator();
        HarmonyEvaluator he = new HarmonyEvaluator();
        OneVoiceLeadingEvaluator vl = new OneVoiceLeadingEvaluator();
        MelodyEvaluator me = new MelodyEvaluator();
        ArrayList<Integer> melody = new ArrayList<>();
        int pivot = 0;
        int degree = Harmony.getRanProb_Strong();
        melody.add(degree);
        Phrase p = new Phrase(beats.length, 4);
        p.setBeatRhythm(beats);
        Note actualNote = new Note(60, 4);
        while (pivot < recommendetLength * p.getSignatureBeats() * p.getSignatureLenght()) {
            LG.fine("pivot " + pivot);
            //ADD NOTE
            int length = rh.evaluate(pivot, beats, 4, degree);
            //
            int pomPitch = Harmony.getClosesPitch_ByGivenPitchClass(actualNote.getPitch(),
            h.getNotePitch_Class_ByDegree(degree));
            actualNote = new Note(pomPitch, length, h);
            p.addNote(actualNote);
            pivot += actualNote.getRealLength();
            //
            //Evaluate next Note
            PenaltyVector harmony = he.evaluate(actualNote);
            PenaltyVector voiceLead = vl.evaluate(actualNote, actualNote, harmony);
            //find max
            int offset = vl.MIDDLE_POINT - Harmony.getIndexByDegree(degree);
            int vl_index = 0;
            float[] res = new float[harmony.vector.length];
            float HARMONY_COE = ((this.SKIP*10)/100.F); //1.0F
            float VOICELEAD_COE = (this.STEP*10)/100.F;
            this.normalize(harmony);
            this.normalize(voiceLead);
            ArrayList<Integer> list = new ArrayList<>();

            for (int i = 0; i < harmony.vector.length; i++) {
                //vector from one voice is centered around middle point
                //so rows dont match
                vl_index = (i + offset) % 12;
                if (vl_index < 0) {
                    vl_index = 12 + vl_index;
                }
                        res[i] = harmony.vector[i] * HARMONY_COE + voiceLead.vector[vl_index] *
VOICELEAD_COE;
           }
            this.normalize(res);
            me.evaluate(res, actualNote);
            res[Harmony.getIndexByDegree(actualNote.getDegreeInHarmony())]
                    = 0.2F;
            normalize(res);
            int actualBeat = RhythmEvaluator.getActualBeatLevel(pivot, beats, 4);
```

```
            for (int i = 0; i < res.length; i++) {
                //vector from one voice is centered around middle point
                //so rows dont match
                if (res[i] > 0.5F) {
                    list.add(i);
                }
            }
            Random rng = new Random();
            int resultPick = rng.nextInt(list.size());
            degree = Harmony.getDegreeByIndex(list.get(resultPick));
            LG.fine("final degree " + degree);
            melody.add(degree);
        }
        int[] temp = new int[melody.size()];
        //
        for (int i = 0; i < melody.size(); i++) {
            temp[i] = melody.get(i);
        }
        Melody newMel = new Melody(temp);
        this.melody_list.add(newMel);
        return new Unit_Data(name, p);
    }
```

7.5. Application output example

Phrase 0

TOP: 55 ON 0 OFF 8 D 1 L8

TOP: 50 ON 8 OFF 20 D 5 L4

TOP: 55 ON 12 OFF 32 D 1 L8

TOP: 57 ON 20 OFF 28 D 2 L4

TOP: 62 ON 24 OFF 36 D 5 L4

TOP: 67 ON 28 OFF 48 D 1 L8

D = degree, L = length of note, TOP = top voice, BASS = bottom voice

Phrase 1

TOP: 59 ON 36 OFF 44 D 3 L8

TOP: 62 ON 44 OFF 56 D 5 L4

TOP: 64 ON 48 OFF 64 D 6 L4

TOP: 62 ON 52 OFF 60 D 5 L8

TOP: 64 ON 60 OFF 72 D 6 L4

TOP: 67 ON 64 OFF 84 D 1 L8

Phrase 2

TOP: 62 ON 72 OFF 80 D 5 L8

TOP: 67 ON 80 OFF 92 D 1 L4

TOP: 69 ON 84 OFF 100 D 2 L4

TOP: 72 ON 88 OFF 92 D 4 L4

TOP: 74 ON 92 OFF 104 D 5 L8

TOP: 72 ON 100 OFF 114 D 4 L2

TOP: 74 ON 102 OFF 124 D 5 L8

BASS: 66 ON 102 OFF 120 D 7 L 4

Phrase 3

TOP: 64 ON 110 OFF 118 D 3 L8

TOP: 60 ON 118 OFF 130 D 1 L4

TOP: 59 ON 122 OFF 138 D 7 L4

TOP: 62 ON 126 OFF 130 D 2 L4

TOP: 67 ON 130 OFF 142 D 5 L8

TOP: 69 ON 138 OFF 154 D 6 L4

Phrase 4

TOP: 55 ON 142 OFF 150 D 5 L8

TOP: 60 ON 150 OFF 162 D 1 L4

TOP: 62 ON 154 OFF 170 D 2 L4

TOP: 65 ON 158 OFF 160 D 4 L2

TOP: 67 ON 160 OFF 170 D 5 L8

BASS: 59 ON 160 OFF 166 D 7 L 4

TOP: 72 ON 168 OFF 182 D 1 L4

BASS: 64 ON 168 OFF 182 D 3 L 4

TOP: 74 ON 172 OFF 190 D 2 L4

BASS: 65 ON 172 OFF 190 D 4 L 4

Phrase 5

TOP: 60 ON 176 OFF 184 D 1 L8

TOP: 55 ON 184 OFF 196 D 5 L4

TOP: 57 ON 188 OFF 204 D 6 L4

TOP: 62 ON 192 OFF 196 D 2 L4

TOP: 60 ON 196 OFF 208 D 1 L8

TOP: 55 ON 204 OFF 224 D 5 L

Output 2

```
TOP:  59 ON 0 OFF 8 D 3 L 8
BASS: 55 ON 0 OFF 4 D 1 L 4

TOP:  62 ON 8 OFF 20 D 5 L 4
BASS: 43 ON 8 OFF 20 D 1 L 4

TOP:  64 ON 12 OFF 28 D 6 L 4
TOP:  62 ON 16 OFF 24 D 5 L 8
BASS: 55 ON 16 OFF 20 D 1 L 4

TOP:  60 ON 24 OFF 36 D 4 L 4
TOP:  55 ON 28 OFF 48 D 1 L 8
BASS: 43 ON 28 OFF 44 D 1 L 4

TOP:  67 ON 36 OFF 44 D 1 L 8
BASS: 55 ON 36 OFF 40 D 1 L 4

BASS: 57 ON 40 OFF 48 D 2 L 4

TOP:  67 ON 44 OFF 60 D 1 L 8
BASS: 55 ON 44 OFF 56 D 1 L 4

BASS: 54 ON 48 OFF 64 D 7 L 4

TOP:  74 ON 52 OFF 60 D 5 L 8
BASS: 55 ON 52 OFF 56 D 1 L 4

BASS: 69 ON 56 OFF 64 D 2 L 4

TOP:  71 ON 60 OFF 76 D 3 L 8
BASS: 67 ON 60 OFF 72 D 1 L 4
```

```
TOP:  72 ON 68 OFF 72 D 4 L 4
TOP:  71 ON 72 OFF 84 D 3 L 8
BASS: 67 ON 72 OFF 80 D 1 L 4

TOP:  62 ON 80 OFF 88 D 5 L 8
BASS: 55 ON 80 OFF 84 D 1 L 4

TOP:  67 ON 88 OFF 100 D 1 L 4
BASS: 55 ON 88 OFF 100 D 1 L 4

TOP:  69 ON 92 OFF 108 D 2 L 4
BASS: 55 ON 92 OFF 108 D 1 L 4

TOP:  69 ON 96 OFF 98 D 2 L 2
BASS: 67 ON 96 OFF 100 D 1 L 4

TOP:  67 ON 98 OFF 108 D 1 L 8
TOP:  62 ON 106 OFF 120 D 5 L 4
TOP:  60 ON 110 OFF 128 D 4 L 4

TOP:  55 ON 114 OFF 122 D 5 L 8
BASS: 48 ON 114 OFF 118 D 1 L 4

TOP:  60 ON 122 OFF 134 D 1 L 4
BASS: 48 ON 122 OFF 134 D 1 L 4

TOP:  62 ON 126 OFF 142 D 2 L 4
BASS: 48 ON 126 OFF 142 D 1 L 4

TOP:  62 ON 130 OFF 134 D 2 L 4
TOP:  60 ON 134 OFF 146 D 1 L 8
BASS: 48 ON 134 OFF 142 D 1 L 4

BASS: 50 ON 138 OFF 150 D 2 L 4

TOP:  55 ON 142 OFF 162 D 5 L 8
BASS: 48 ON 142 OFF 158 D 1 L 4
```