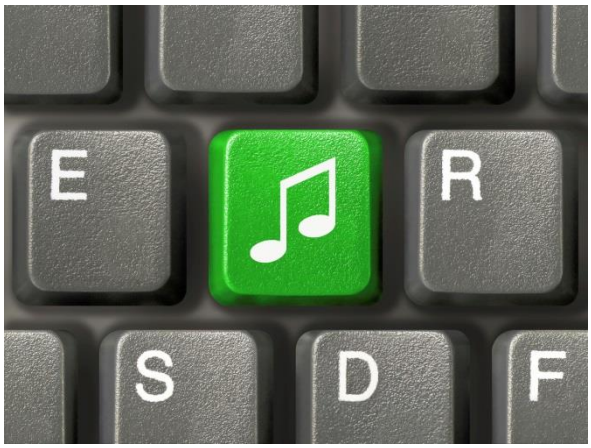
judgemysound



From bytes to music

Minor thesis: Generating music through algorithm composition.

Copyright © J.P. van der Boog
All right reserved.

Author:	J.P. van der Boog
Student number:	4215443
Education:	Mechanical Engineering
University:	TU Delft
Date:	1st of February 2017
Supervisor:	Dr. C.C.S. Liem
Faculty:	Computer science

Preface

This report has been written to fulfil the graduation requirements of the free minor in computer science at the TU Delft. I have worked on the JudgeMySound project for the last one and a half year, of which the last half year was during the minor. I am pleased with the end result of this report and what the application and algorithm have come to so far.

I would like to thank my supervisor, Cynthia Liem, for giving me the possibility to combine my study with a project I believe in. Writing the report was not always easy and much had to be rewritten. However, Cynthia Liem has guided me through this process and gave me insight on how to write a more professional report.

I want to thank Lieke Michielsen and my parents in particular, for supporting me during the whole project. If I ever lost interest or needed someone to talk to, they helped me in any way they could.

I hope you enjoy this report.

Johan van der Boog

Table of contents

Introduction 4
Chapter 1: Project JudgeMySound 5
Chapter 2: The old generating algorithm 7
General parameters 7
Scale 8
Chord progression 8
Rhythm 10
Bassline 11
Chord line 11
Melody 12
Chapter 3: Application 13
Import page 13
Create page 14
General 15
Design 15
Chapter 4: New Algorithm 17
Fixed parameters 18
Chapter 5: Chord progression 19
Analysis of dance chord progressions 19
Generating new chord progressions 22
Bassline 24
Chord line 24
Chapter 6: Melody 25
Analysis of rhythm 25
Analysis of pitch 26
Generating new melodies 29
Chapter 7: Evaluation plan 32
Evaluation method 32
Storage of opinions of audience 33
Different creation methods 34
References 35

Introduction

This report is a detailed description of the JudgeMySound project. JudgeMySound started in September 2015. The original concept was to make an algorithm that could generate decent song fragments. The decentness of a song is based on the appreciation of the public. To do so, there were three main parts in the process, namely the generating, rating and learning part. First, the algorithm would generate an amount of song fragments based on probability. Secondly, these fragments were rated online on a website which was accessible to everyone. Thirdly, a learning algorithm would analyse these ratings to rate the decentness of each newly generated song fragment. Before this process started, the question was, and still is, if it is even possible to create a song completely composed by a computer algorithm. Scientists have already proven that this is possible, with programs based on genetic algorithms like GenJam (Biles, 1994), Darwin Tunes (MacCallum, Mauch, Burt, & Leroi, 2012) or through algorithmic composition (Jacob, 1996) with programs like Kalitta (Quick, 2014). However, JudgeMySound did not function as what was hoped for, like the other mentioned programs. Therefore a new algorithm has been coded. The new algorithm has some modifications relative to its older version. The first main modification is that the algorithm uses existing songs as data to create new song fragments. The second modification is that the algorithm will be implemented in an application in which musicians can interact with the algorithm to create new song fragments. Therefore, the main research question is:

How can existing songs be used to generate the same type of music through algorithmic composition?

The algorithm should be able to use the given data to create a new song fragment. It should deviate from its input to explore the solution space, while still be creating the song fragment in the same style. The algorithm will be divided into two sections, the analysis and the generation. The analysing algorithm will retrieve musical information from existing songs and translate them into formats that the generative algorithm can use.

The research question can be decomposed in the following questions:

What characteristics of music are taken into account?
How are these components integrated in the algorithm?
How can results of the generating algorithm be evaluated?

To answer these questions, the report is divided into 5 parts. Firstly, the JudgeMySound project is explained further in more detail. Secondly, the old algorithm is explained. Thirdly, the designs of the application are shown and described. Fourthly, the new algorithm is explained including why certain functionalities have been modified relative to the old algorithm. Finally, a plan on how to evaluate the results of the generative algorithm is described.

**The definition of musical terms can be found at http://www.potsdam.edu/academics/Crane/MusicTheory/Musical-Terms-and-Concepts*

Chapter 1: Project JudgeMySound

As described briefly in the introduction, JudgeMySound is a project with as main goal to generate a decent song fragment algorithmically through probability without human interference. The first attempt to reach this goal led to some complications that were too hard to solve. Therefore a new approach arose that avoided these complications.
In this chapter, the different components of the first attempt are being described. The complications that arose and the modifications for the new approach relative to the first attempt will be discussed as well. The multiple components of the first attempt are listed below and are shown in Figure 1:

- The algorithm that composes the song fragment through probability (generation).
- The website that collects ratings of these song fragments (rating).
- The neural network that evaluates these ratings (learning).
- The newly generated song fragments that get filtered on their decentness (generation with knowledge).

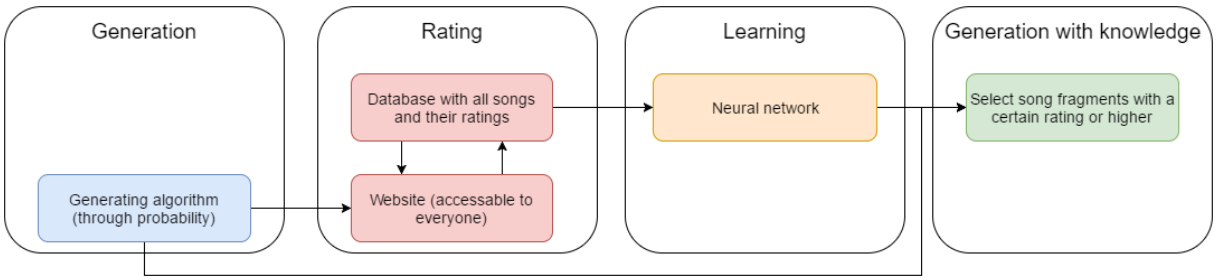


Figure 1. Original structure of the JudgeMySound project.

The generating algorithm (first block of Figure 1) is based on music theory and certain parameters that are set. These parameters will be discussed further in the next chapter. The generating algorithm has created 10.000 song fragments which are stored in a database and can be listened to on the website www.judgemysound.com (second block of Figure 1). Everyone that visits the website can rate song fragments with an integer between 1 and 5. The average rating of each song is calculated and is analysed by a multilayer classification neural network with backpropagation (Nielsen, 2017) (third block of Figure 1). The neural network will be able to rate newly generated song fragment based on the knowledge of the ratings people gave on the website. When new song fragments are generated, the algorithm filters them on the rating they have been given by the neural network (fourth block of Figure 1). This way only the decent songs remain.
The neural network, however, could not find any relations between its input and its ratings. This is probably because of the following:

- Each individual person rates differently, due to emotions or music preference.
- There is not enough data.
- The neural network structure dimensions are not fit for the given problem.

It is not precisely clear what caused the neural network to dysfunction. A neural network functions like a black box, so it is not possible yet to know exactly how a neural network learns (Castelvecchi, 2016). It is therefore also hard to configure the input properly. The output on the other hand, would

be used to filter only the decent songs with a certain rating or higher. Is it necessary though to use a neural network to find out decentness of a song fragment? No, it is not. There are already so many songs in the world, so it is already known what song fragments are successful. It is not explicitly known what makes them great (Engelhardt & Pasquier, 2010), but programming an algorithm with certain aspects of these songs, like tonics and structures, will most likely result in a well-functioning algorithm with decent output. For that reason, the focus of the JudgeMySound project is to improve and rewrite the generative algorithm (first block of Figure 1) and neglect the rating, learning and generation with knowledge parts (last 3 blocks of Figure 1). This new and improved generative algorithm will purely base its probabilities on data of existing songs.

Although the goal is to create song fragments without human interference, it does not mean it replaces humans. Since music is subjective, not all song fragments will be good enough for everyone. Besides, a song fragment created with an algorithm will most likely not be directly perfect at all. Therefore, there needs to be a human opinion (read: musician) on whether a song fragment is good enough to be successful. An application will make an interaction between the algorithm and musicians possible. With an application, a musician can generate song fragments with the algorithm and can edit them to their preference after. While editing the song fragment, the musician uses the algorithm again to regenerate it partially or completely. With this application, the crowdsourced rating of song fragments is totally replaced. Instead of rating a complete song fragment and use these ratings to create better ones, musicians can now interactively change a song fragment partially, until the song fragment meets their preference. The application will be discussed in detail in the third chapter.

To conclude, these are the modifications that the new algorithm will contain:

- The algorithm will base its probabilities on existing songs.
- An application will be an interaction between the algorithm and musicians.

Chapter 2: The old generating algorithm

This chapter will describe the old algorithm in detail. The old algorithm is the first generative algorithm that has been coded for this project (described in the previous chapter). The song fragments on the JudgeMySound website are generated with this algorithm. There is much resemblance between the old and new algorithm. They both generate multiple musical layers that are played on top of each other and they share some functionality on how they generate the song fragments. In the old algorithm the layers that are generated are the chord progression, bassline, chord position and melody. Each layer contains notes with a certain length and pitch. The bassline, chord position and melody also have a rhythmic component. The rhythm of these layers is determined before the pitch of the notes is determined. In this chapter, the set of parameters for each musical layer is illustrated. These parameters are shown in a table per musical layer; left the name of the parameter, right a short explanation. They can be determined by a human and will be explained chronologically as how the algorithm generates them (Figure 2).

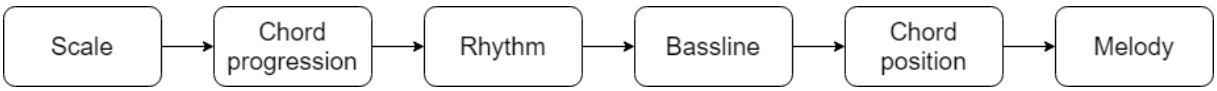


Figure 2. Chronological order in which the algorithm generates a song fragment.

General parameters

There is one general parameter which is set manually, that is not specifically linked to any musical layer, namely beats per minute (BPM). BPM defines how fast a song fragment is played.

BPM	Beats per minute

The length and structure of the song fragment is determined beforehand as well. Although, these are parameters set by the algorithm. The algorithm handles the song fragment as quadruple time and divides the song fragment in 64 equal blocks. It divides the song fragment in 4 bars, which means that each bar contains 16 blocks.

The position and length of notes are defined in these blocks. For example, if a note starts at the fifth block of the song fragment and has a length of 4 blocks, it means that the second quarter note of the song fragment is played.

The division into blocks also means that the maximum amount of notes played in 1 bar is 16. Figure 3 shows the structure of one bar of the song fragment.

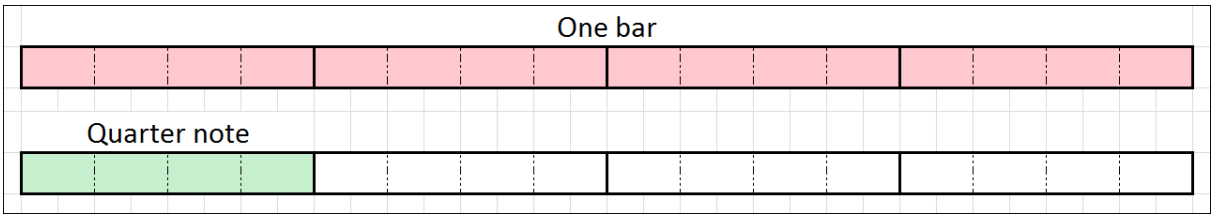


Figure 3. Structure of quadruple time bar.

Scale

The scale and tonic pitch limits the amount of notes that can be chosen from an octave. The scales that can be chosen by the user are always heptatonic, so the maximum amount of notes that can be chosen from is 7 instead of 12. All heptatonic scales are shown in the table below.

Scales using notes A-B-C-D-E-F-G	
Starting note	Scale name
A	Aeolian (Minor)
B	Myxolydian
C	Lydian (Major)
D	Phrygian
E	Dorian
F	Syntolydian
G	Ionian

Figure 4. All heptatonic scales based on starting note.

The tonic pitch is determined with an uniform distribution. When the scale is chosen as well, based on its probability of occurrence, the algorithm creates an array with all diatonic triads based on the notes of the selected scale. The triads that are neither major or minor are deleted from the array.

Tonic pitch	The beginning and ending note of the diatonic scale
Chance scale	Probability array with chances for any heptatonic scale to occur

In the example below the major scale is chosen with C as tonic pitch.

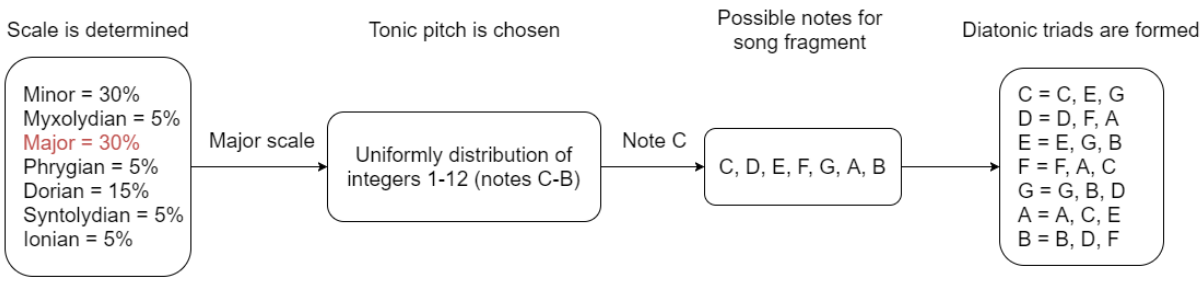


Figure 5. Process of forming diatonic triads.

Chord progression

The chord progression is the base of the song fragment. Both the bassline, chord position and melody are derived from the chord progression. The chord progression is divided into two parts, namely what chords are played and the chord change positions.

The user assigns three parameters that will influence the algorithm its choice of what chords to select; the number of occurrences of chords, the number of unique chords and the chances of occurrence for the diatonic triads that can be chosen from. The number of chords is limited by the maximum amount of notes possible in a scale and must be equal or higher than the amount of

unique chords. The amount of unique chords is limited to the maximum amount of diatonic triads that can be chosen, which is 7 in a heptatonic scale.

Unique chords	The amount of unique diatonic triads in the song fragment (< 7)
Chord number	The amount of diatonic triads in the song fragment (>= Unique chords)
Chord occurrence	The chances for each diatonic triad to occur in the chord progression

Based on the above information, the algorithm goes through the following steps (Figure 6):

1. It chooses what unique diatonic triads will be in the chord progression based on the amount of unique chords.
2. If the amount of chords is larger than the amount of unique chords, it chooses what diatonic triads occur once more. This is repeated until the amount of occurrences of chords is reached.
3. It divides the chords in a sequence with a uniform distribution.

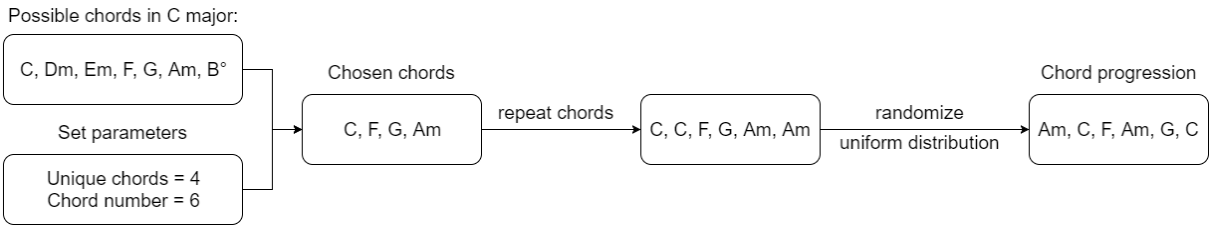


Figure 6. Process of determining chords in chord progression.

After the chord progression is determined, the chord change position is calculated. The chord change position is the position at which a chord transition occurs. Most songs in top hit lists have a repetitive pattern in their chord change position[1] (Mauch, Noland, & Dixon, 2009). The chords are mostly equally divided over the first and last 2 bars and the chords also have the same duration most the time (see Figure 7). Figure 7 is based on chords chosen in Figure 6.

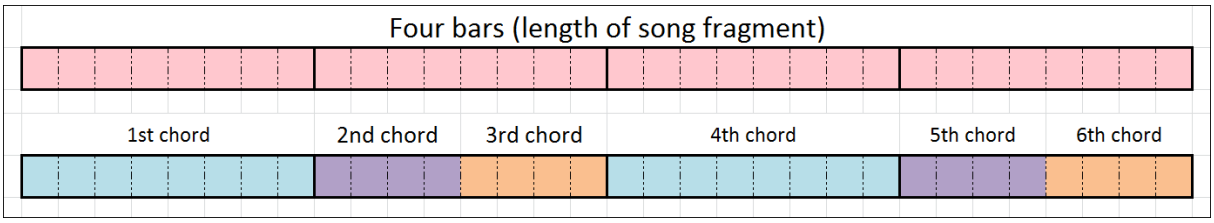


Figure 7. The chord change position of the first two bars is the same as the last bars

Therefore, the user sets a percentage as parameter to let the algorithm decide whether the chord change position is the same in the first and last 2 bars of the song fragment. If this is true, the algorithm also decides whether the chords played will be the same as well. If that is the case, the chord progression of the last 2 bars that is calculated before, is overwritten. However, to be able to execute these steps, the total amount of occurrences of chords must meet the condition that it is an even number, otherwise the chords cannot be equally divided. With the parameter set, the algorithm calculates all possible combinations that meet the criteria stated above. These combinations are

[1] This assumption is also based on an own observation of many hit songs on www.hooktheory.com.

ordered by the length of the chords. If all chords have roughly the same length, they will be on top of the list. After the list is sorted, one of these combinations is chosen through a Gaussian distribution with a sigma given by the user and the first row of the list as mean.

Chord length sigma	Sigma for choosing the chord change position in the chord progression
Chance identical struct	Chance that chord change position is identical in bars 1-2 and 3-4
Chance identical chord	Chance that also the chord progression is identical in bar 1-2 and 3-4

Rhythm

Humans tend to like things that they have consciously or unconsciously encountered before (Margulis, 2014). Repetition is therefore an important musical component (Bradly, 1971). The algorithm implements rhythm for the bassline, chord position and the melody. The bassline has the same rhythm as the chord progression. This means that the notes of the bassline start at the same position and have the same length as the chords in the chord progression. On the other hand, the rhythm of the chord position and melody are calculated differently. The algorithm uses the same functionality on both musical layers. It uses two functions; one determines the repetition of the rhythm in bars and the other calculates the position and length of the notes within these bars.
To determine repetition, the algorithm first determines if the rhythm of the chord position will be the same as the rhythm of the melody.

Melody Rhythm Chord	If true, rhythm of chord position is equal to the rhythm of the melody

After that, the algorithm creates 1 or 2 (dependent on the previous decision) array(s) of length 4 with uniformly distributed integers between 1 and 4. Every index in each array represents 1 of the 4 bars in the song fragment. If the values of any indices match, the bars, that these indices represent, will be rhythmic the same (see Figure 8).

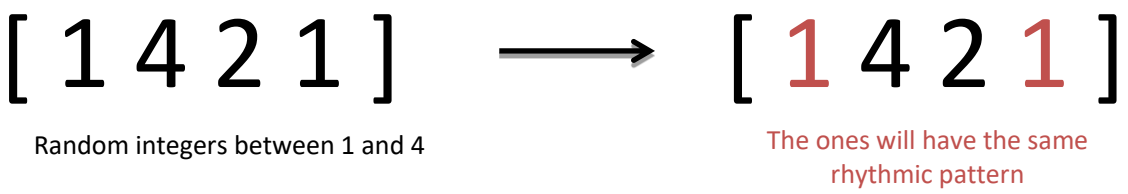


Figure 8. Determine repetition of the bars in the song fragment

For the rhythm, the user sets an array of length 4 for the position and a matrix of 4 by 4 for the length. Each index in the position array represents a quarter of a quarter note, in other words one sixteenth of a bar. The value of each index is a percentage that is used independently to calculate if a note is starting on a particular position (table 1). This means that the algorithm determines if a note starts for every possible position in the song fragment (64 possibilities). After it knows where all notes start, it calculates the length of these notes based on the length matrix accordingly (table 2). The length matrix is a matrix of 4 by 4. Each row represents a quarter note in a bar and contains percentages that sum up to 1. Each column represents the length of a note in blocks.

[75% 35% 50% 25%]

Table 1. Position array (each index represents independently the chance that a note is played on that position).

	Length (in blocks)				
	1	2	3	4	
1st quarter note	Percentages				100%
2nd quarter note					100%
3rd quarter note					100%
4th quarter note					100%

Table 2. Length chance matrix.

The process of how the note starting positions and lengths are selected is shown below (Figure 9).

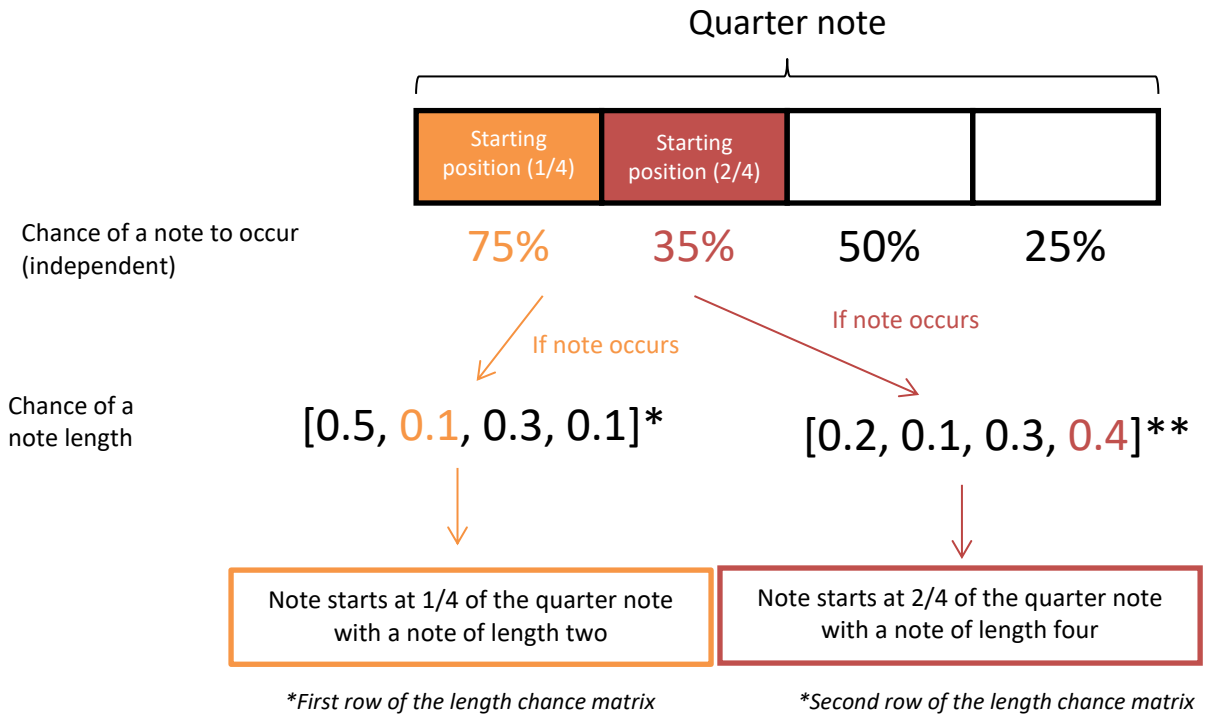


Figure 9. Process to determine where a note starts with what length (64 possible notes in total).

When all notes have a starting position and length, the notes that overlap are shortened. The starting position of all notes does not change, only the length is adapted. In the example of Figure 9, the first and second note have an overlap of 1 block. The first note will therefore be shortened to 1 block and the second note will not be altered, unless another note starts somewhere in the 4 blocks after the starting position of the second note. This process is repeated independently for all notes in the chord position and melody.

Bassline

As already described in the previous section, the bassline has the same rhythm as the chord progression. The pitch of the notes of the bassline are the same as all root notes of all the chords in the chord progression.

Chord line

The notes that are played in the chord line are randomly, based on a uniform distribution, selected notes from the diatonic triad that is played in the chord progression at that time. However, the user

sets a parameter that determines how many notes of these triads are selected and played simultaneously. The amount of notes can differ from a minimum of 1 and a maximum of 3 notes.

Chord notes played	The amount of notes played simultaneously in the chord layer

Melody

The algorithm goes through several steps to determine the pitches of all notes:

1. The key shift is determined with a Gaussian distribution with a sigma given by the user and a mean of 0 that represents the tonic note.
2. The pitch range is determined. In other words, the lowest and highest possible pitches are set, so the pitch in the melody will not be too high or too low.
3. The first pitch is chosen, based on a uniform distribution, from all pitches in the pitch range.

Steps 4 to 8 are repeated as many times as the number of notes that are in the melody (determined at rhythm section), minus the first note (which is already set at step 3):

4. The algorithm checks if the pitch of the current note is the same as the pitch of the previous note, based on the parameter 'chance repetition' set by the user. If the pitch is the same, steps 5-8 are skipped and the algorithm continues with the next note.
5. The algorithm checks if the note will be a member of the chord that is played at that particular position.
6. First all possible pitches that can be chosen from are filtered on their presence in the scale. Only pitches that are part of the scale can be chosen for the melody. Secondly the pitches are filtered on the outcome of step 5.
7. One of the pitches that remains after step 6 is selected through a Gaussian distribution with a sigma given by the user and the previous pitch as mean.
8. If the note is not member of the chord that is played, the algorithm can overwrite the pitch and change it to a member of the chord. This only happens if the note has a certain length or when the amount of blocks between the previous note and the current note is too high.

Range Melody	The range of the notes (lowest and highest possible pitch)
Key shift sigma	Sigma for choosing the lowest note in scale range
Note pitch sigma	Sigma for distribution for choosing the pitch of the next note
Chance repetition	The chance that the note is repeated
Chance member chord	Chance that the note is a member of the chord played
Length Note	If the note is larger than length note, the pitch of the note is changed to one that matches with one of the chord that is currently played
Gap notes	If the amount of blocks between two notes is larger than gap notes, the pitch of the note is changed to one that matches with one of the chord that is currently played

When all calculations are finished, the song fragment is converted to MIDI format.

Chapter 3: Application

The application contains the new algorithm. It is designed to make an interaction between the new algorithm and a musician possible. Musicians can use this application to get inspired to make a new song. The algorithm generates song fragments that are based on existing songs which are selected beforehand by the musician. The application gives them the possibility to (re)generate each musical layer of the song fragment partially or completely.
The application has an interface in which the algorithm is implemented. This interface has 2 main pages, the import page and the create page. The import page is where existing songs can be implemented and can be uploaded into the song database. The create page is the page where the musician can generate a song fragment, based on the songs in the database from which the user can choose any. The functionality of both pages are independent. They only share the database. The structure of the application is visualized in Figure 10. Each green box represents a song fragment from the database. These songs have thus been imported on the import page. The rest of Figure 10 is part of the create page.

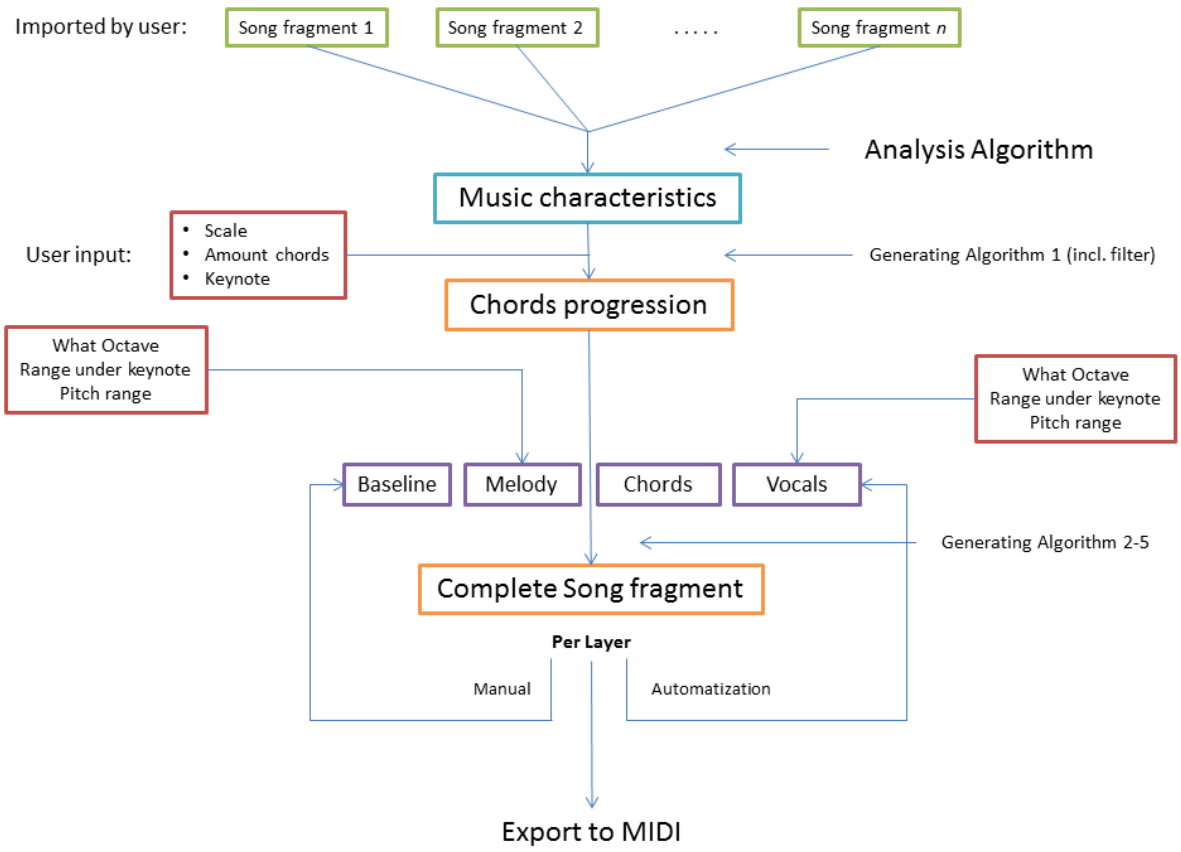


Figure 10. Structure of the application.

Import page

To import an existing song into the database, the user takes the following steps (see Figure 11):

1. Type the name of the artist and the song.
2. Modify general and musical options; the user fills in the right BPM, scale and tonic.

3. Add notes to the stave of each layer of the song fragment the user wants to import.
4. Store the song in the database.

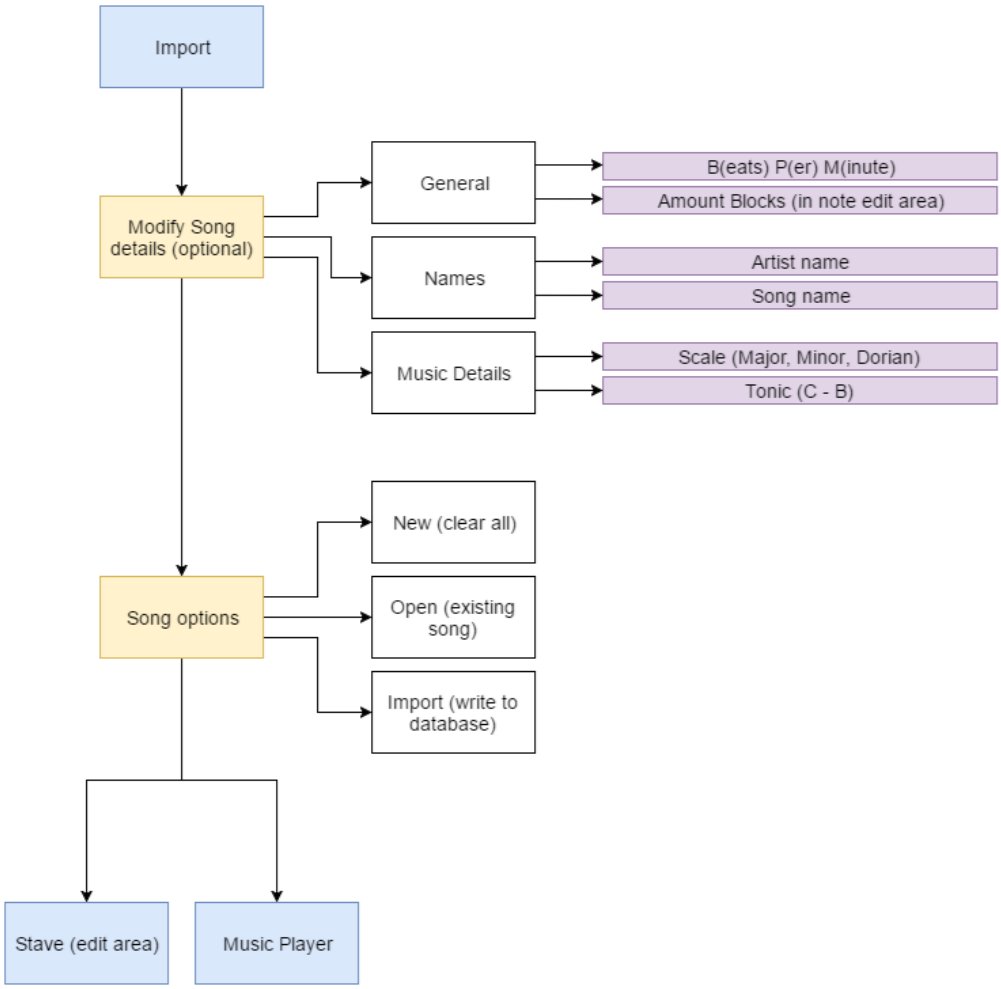


Figure 11. Visualization of import page functionality.

Create page

The musician goes through several steps before they have a generated song fragment. These steps are also visualized in Figure 12 and bottom images of Figure 14:

1. The user selects existing songs from a list of songs that are imported into the database of the application beforehand. The algorithm will retrieve music characteristics from the selected songs and converts them into a certain format. This format can be used by the algorithm to generate each musical layer in the song fragment.
2. The user sets three parameters; scale, amount of occurrences of chords and tonic note. The algorithm creates a chord progression within the conditions of these parameters.
3. If the user is satisfied with the chord progression, the user can generate each musical layer optionally, based on specific parameters that are different for each musical layer.
4. If the user is not satisfied with a musical layer (partially or completely), the user can change the notes manually or let the algorithm regenerate the song fragment partially or completely.

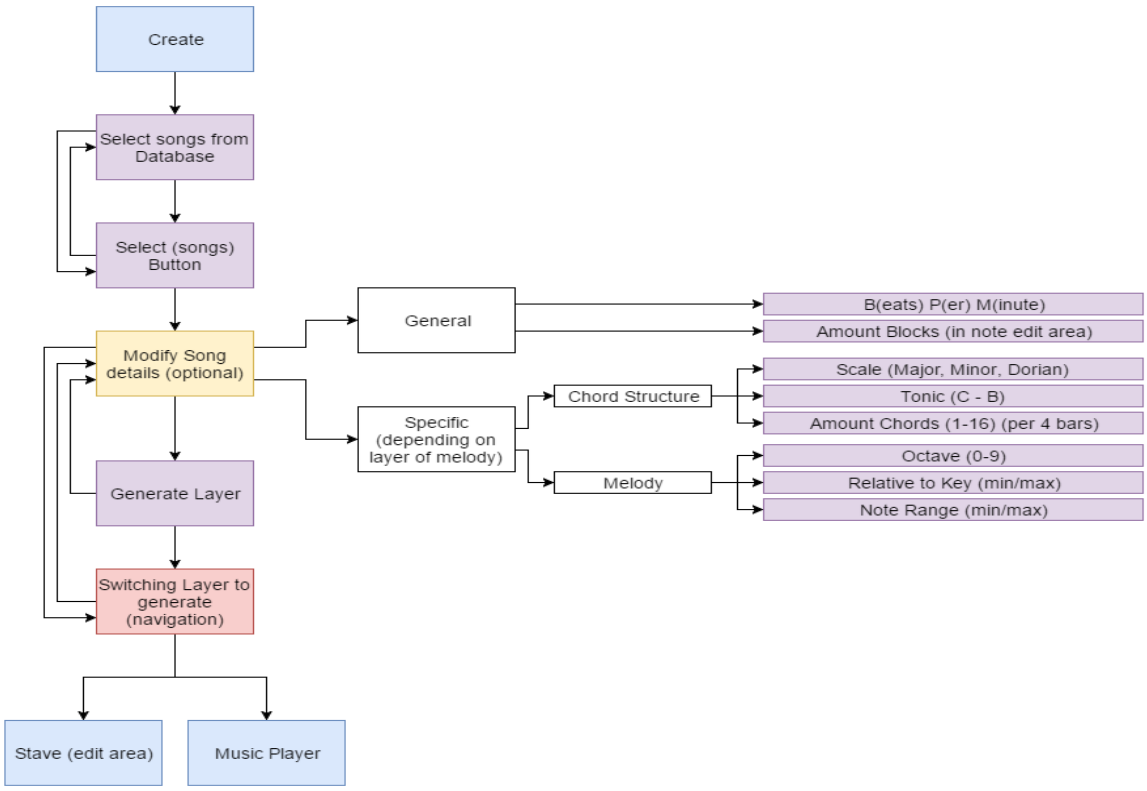


Figure 12. Visualization of create page functionality.

General

There are some general functionalities that the create page and import page share, namely the stave and the music player (both seen on Figure 11 and 12). The stave is the area where notes are displayed and can be modified. The functionality of both the stave and music player are shown in Figure 13.

Figure 13. Functionality stave and music player.

Design

The create page and import page have the same layout. The design of the create page is shown below. The original designs are made by Jill van Remundt (architecture student at TU Delft) and are focussed on the create page.

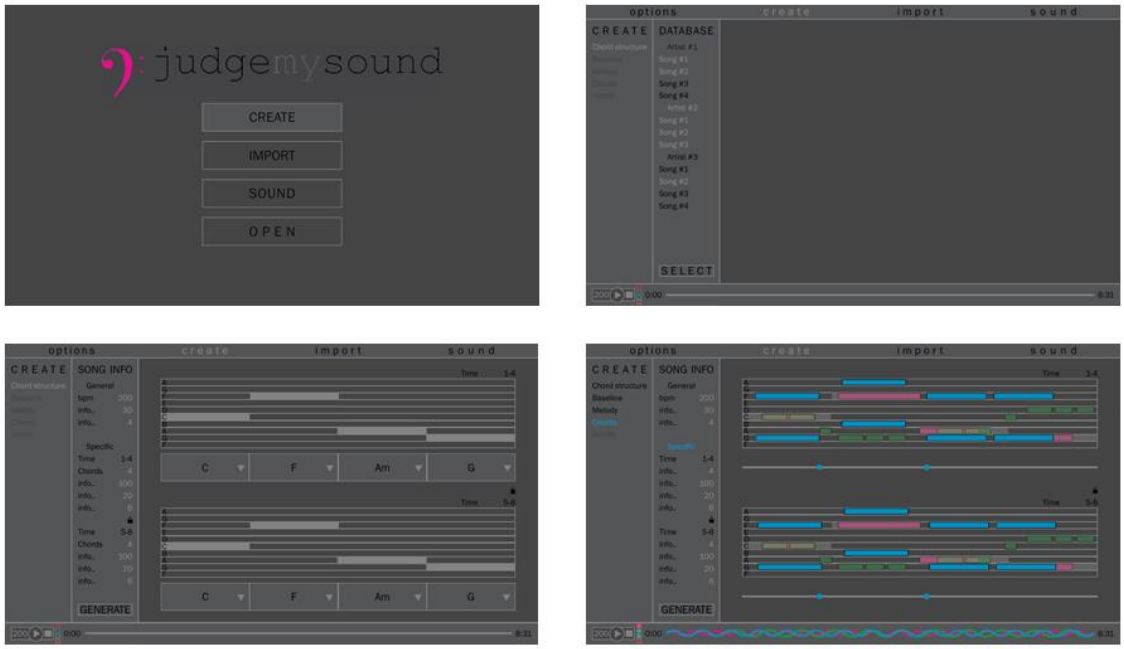


Figure 14. Design of the application.

The upper left image of Figure 14 is the page the user sees when he starts the application. On this page, the user has four choices; the user can either go to the import or create page, change the sound settings or open an existing project (which will open in the create page).

The upper right image of Figure 14 is the selection of songs from the database. On this page, the user selects the songs from the database on which the algorithm will base its probabilities.

The bottom left image of Figure 14 is the generation page for the chord progression. On this page, the user generates the chord progression that will be used as base for the other musical layers. This page includes a chord recognizer, that will give direct feedback on what chords are played.

The bottom right image of Figure 14 is the generation page for the other musical layers. Each layer has a colour which represents that specific layer. In all layers, except for the chord progression, the notes from the other musical layers can be seen with their transparent colour.

Chapter 4: New Algorithm

In chapters 4, 5 and 6, all processes of the new algorithm are explained. The new algorithm is the algorithm the application is built on. As stated in chapter 1, the main modification of the new algorithm relative to the old algorithm is that it will base its probabilities on existing songs. Therefore, the new algorithm can be split in 2 parts, namely the analysis and generative algorithm. The analysis algorithm uses a database of existing songs. It retrieves information from these songs for the chord progression, bassline, chord position and melody. It converts this information into a format the generative algorithm can use (see Figure 15).

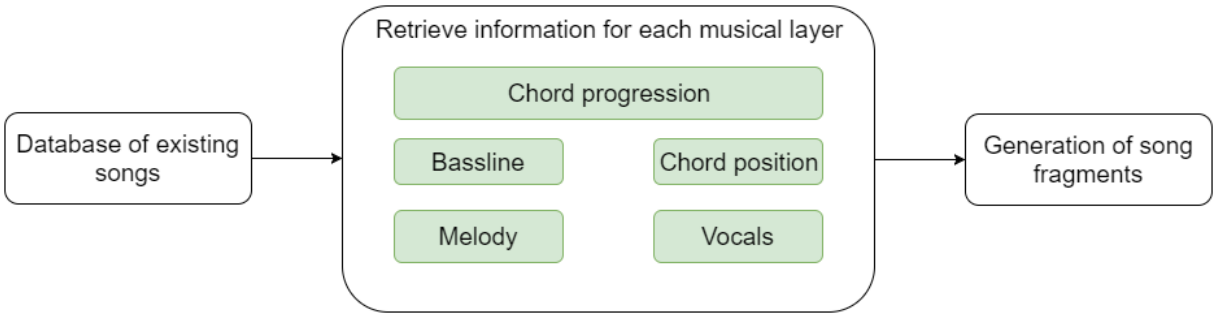


Figure 15. Data analysis structure.

Each song fragment has 4 layers; the chord progression, bassline, chord position and melody. In the old algorithm the generating process of the different layers happened simultaneously; as one process. One click on the button resulted in the generation of a complete song fragment. In the new algorithm on the other hand, each layer is generated separately. However, the old and new algorithm both generate all layers (except the chord progression) based on the chord progression. Without the chord progression, no other layers can be generated. For that reason, just like in the old algorithm, the chord progression is the first layer that is created. All layers are generated on top of the chord progression. They only use the information from the chord progression and existing songs, not from the other layers (see Figure 16).

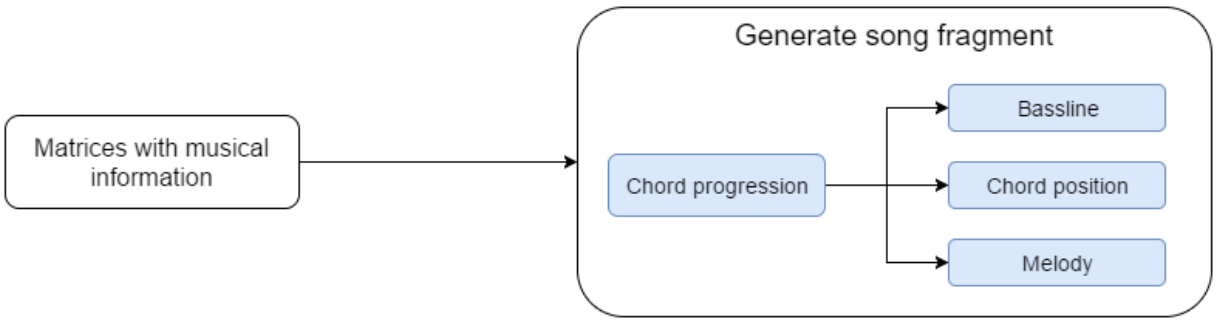


Figure 16. Generative algorithm structure.

In chapters 5 and 6, each layer is explained through its analysis and generative part. The functionality of the analysis part is described first and a possible format is given as example on how to store the data. After that, the generative algorithm for that particular layer is explained, including how the data from the analysis is used.

Fixed parameters

The algorithm uses, just like in the old algorithm, certain parameters and structures that do not change throughout the whole code. Some of these parameters can be considered as limitation for the variety of the outcome. However, these limitations make programming easier, because they narrow the variety (read: randomness) in certain extent and consequently increase the chance of generating better song fragments. The parameters that are set are for the rhythm and possible scales. These were set in the old algorithm as well. The new algorithm is also only based on heptatonic scales. The only difference relative to the old algorithm is that with respect to the rhythm, the amount of blocks per bar is doubled to 32 instead of 16 (see Figure 17).

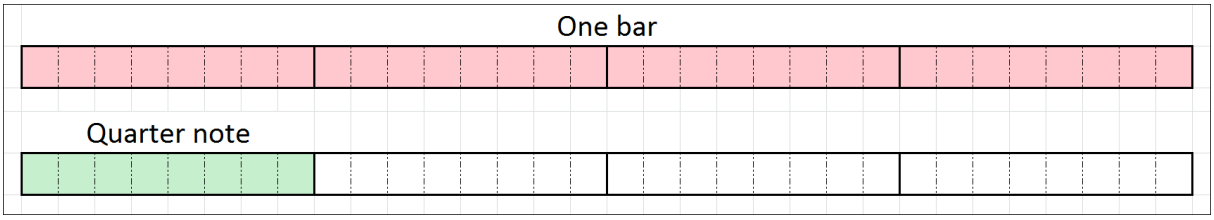


Figure 17. Blocks of one bar in new algorithm.

Chapter 5: Chord progression

As explained earlier, a chord progression is a succession of musical chords. Whether a chord progression is good, depends on the relation between the chords in the succession (Ewer, 2010). To express this relation, the analysis algorithm retrieves information for the starting chords, starting progressions, occurrences of chords and the occurrences of transitions of progressions. The reason why this information is retrieved is explained first using a computational analysis of 172 dance songs, whereafter a possible format for each type of information is described. However, these songs are not part of the database. The analysis of these dance songs is purely used to argue the choice of information that is retrieved from the songs in the actual database. After all the retrieved information is described, the functionality of the generative algorithm is illustrated. Since the bassline and chord position are closely related to the chord progression (Gore, 2009), these are discussed in this chapter as well.

Analysis of dance chord progressions

An analysis of 172 dance songs[2] of the major and minor scale points out musical characteristics for a chord progression in the dance genre (table 3), since the algorithm is mainly based on dance music. The upper tables show the amount of times a chord transition occurs. The numbers one to seven in the rows and columns represent the diatonic triad in its scale (example in Figure 6). The lower tables show the (unique) occurrence of each chord, the starting chord and the last chord of the progression. There are a couple of things that can be concluded from these tables:

- Some chords are more prominently present than others (which chords differ per scale category).
- Some transitions of 2 chords do never/hardly occur.
- The majority of song fragments start mostly from a selection of 2 or 3 chords.

The analysis algorithm compresses the information listed above into 4 different types of information. It uses the data from the starting chord, starting progression, occurrences of chords and the occurrences of transitions of progressions. These formats are all used in the generation of new chord progressions.

Major scale		TO						
Songs:	68	1	2	3	4	5	6	7
FROM	1	0	6	7	25	27	3	0
	2	4	0	2	6	3	7	0
	3	0	3	0	4	4	7	0
	4	28	6	4	0	25	10	0
	5	20	7	2	17	0	32	0
	6	19	1	3	18	18	0	0
	7	0	0	0	0	0	0	0

		1	2	3	4	5	6	7
	Unique	59	21	14	63	61	56	0
	Occurence	77	24	19	81	86	62	0
	Starts with	17	5	1	26	6	13	0
	Ends with	12	2	5	11	30	8	0
	Start and End	0	0	0	2	1	1	0
	How often	87%	31%	21%	93%	90%	82%	0%

Minor scale		TO						
Songs:	104	1	2	3	4	5	6	7
FROM	1	0	1	14	17	21	36	27
	2	2	0	0	1	0	0	0
	3	5	0	0	9	6	9	18
	4	18	1	5	0	13	11	7
	5	30	0	2	7	0	20	4
	6	17	1	16	10	15	0	28
	7	42	0	9	13	10	12	0

		1	2	3	4	5	6	7
	Unique	104	3	43	52	55	80	76
	Occurence	120	3	49	57	67	94	90
	Starts with	70	0	0	6	2	25	1
	Ends with	34	1	4	8	25	12	20
	Start and End	18	0	0	0	0	1	0
	How often	100%	3%	41%	50%	53%	77%	73%

Table 3. Analysis of 172 songs (which are not part of the actual database)

[2] Songs are manually analysed from www.hooktheory.com

Starting chord, starting progression and occurrences of chords

The starting chord, starting progression and occurrences of chords can directly be derived from the song database. An example of how they can be stored are shown in Figure 18 and 19. Each row in this format represents a certain scale. The columns represent the occurrence of each chord or the occurrence of a progression (dependent on what information should be stored). For example for the starting chord format, if the song fragment starts with the root chord (first chord in the scale) of a major scale, the first column of the third row is incremented by 1.

		Chord						
		1	2	3	4	5	6	7
Scales	Minor	Number of occurrences first chords **OR** Unique occurrences chords						
	Myxolydian							
	Major							
	Phrygian							
	Dorian							
	Syntolydian							
	Ionian							

Figure 18. Matrix for occurrences starting chord and unique occurrences of chords.

		Progression						
		0	1	2	3	4	5	6
Scales	Minor	Number of occurrences first progressions						
	Myxolydian							
	Major							
	Phrygian							
	Dorian							
	Syntolydian							
	Ionian							

Figure 19. Matrix for occurrences starting progression.

Occurrences of transitions of progressions

To calculate the occurrences of transitions of progressions, all progressions in each chord progression are retrieved from the song database. A progression is specifically defined here as the relation between only 2 chords. To do so, the chords are first categorized as a diatonic triad from the scale. Figure 20 shows an example of a C^7 chord in a major C scale that is categorized as a C diatonic triad.

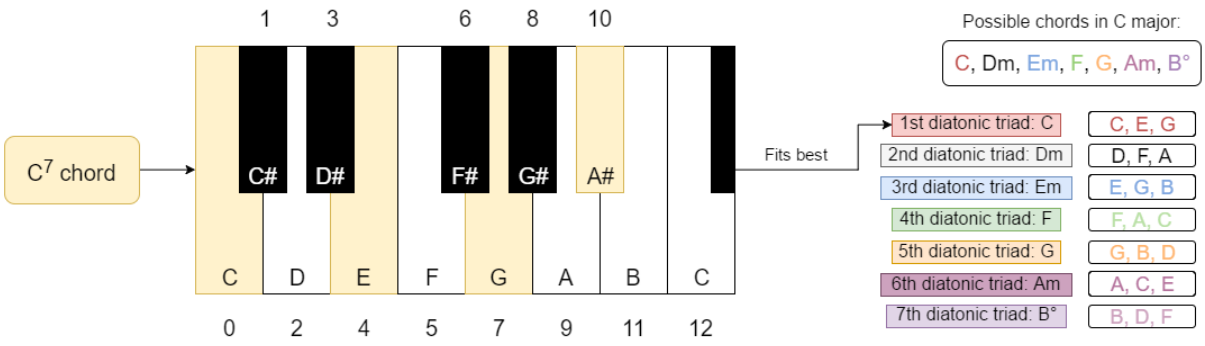


Figure 20. Categorizing chords into diatonic triads from the scale.

When all the chords are categorized, the algorithm determines the relation for each chord combination that is retrieved from the song database. This relation is expressed as an integer between 0 and 6 (Figure 21). This integer represents two things; the amount of notes that are the same in both categorized chords and how the notes that are different change through the progression. If only diatonic triads are taken into account (which is true), there are 4 possible progressions (Tymoczko, 2012), namely:

- None; all notes are the same (the first and second chord are the same)
- Weak; only 1 note is different
- Medium; 2 notes are different
- Strong; all notes are different

Integer	progression	description	chord progressions
0	None	all notes stay the same	1st - 1st
5	weak	the root and third from starting chord are the third and fifth of the second chord	1st - 6th - 4th - 2nd - 7th - 5th - 3rd - 1st
3	medium	fifth note from starting chord is the root note for the next chord	1st - 4th - 7th - 3rd - 6th - 2nd - 5th - 1st
1	strong	all notes from the chord go one step up in the scale	1st - 2nd - 3rd - 4th - 5th - 6th - 7th - 1st
2	weak	the third and fifth from starting chord are the root and third of the second chord	1st - 3rd - 5th - 7th - 2nd - 4th - 6th - 1st
4	medium	the root note from starting chord is the fifth note of next chord	1st - 5th - 2nd - 6th - 3rd - 7th - 4th - 1st
6	strong	all notes from the chord go one step down in the scale	1st - 7th - 6th - 5th - 4th - 3rd - 2nd - 1st

Figure 21. Information for each progression between two chords.

Weak, medium and strong are terms that indicate how many notes change (Joutsenvirta & Perkiömäki, 2010). The more notes differ between the diatonic triads, the stronger the progression is. However, there are two possibilities for each progression on how the notes change. This depends on what the notes, that differ between the chords, change to. If, for example, in a C major chord one note changes, it can either form an A minor chord (by changing the G to A) or an E minor chord (by changing the C to B). All the possible progressions are shown in Figure 21. Figure 21 also shows the integer and its expression, progression name, description and the related progressions. The green boxes are defined as upwards progressions and the red boxes are defined as downwards progressions. The downwards progressions contain the opposite progressions of the upwards progressions.

For example, if in the major scale the first chord progresses to the second chord of the scale, the notes of the first chord all go up by one (Figure 22). The fact that they all go up and do not share any notes, means that the progression is strong upwards. If the progression would be reversed, namely from the second to the first chord, it would be a strong downwards progression.

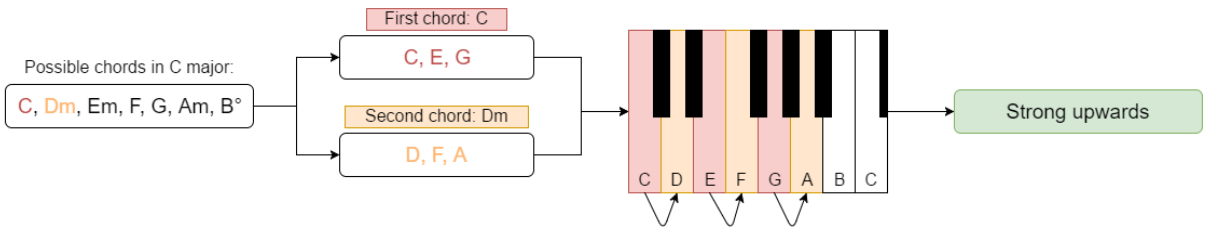


Figure 22. Progression of the first to the second chord in the major scale.

Now that all the progressions are calculated from the information retrieved from the song database, the algorithm looks at the transitions between these progressions (seen in Figure 23). The hexagons

represent the transitions of progressions. The reason why the transitions of progressions are stored, instead of only the progressions, is that the transitions of progressions contain more information. The transitions connect every possible successive combination of three chords, whereas the progressions only link combinations of two chords. All this information is stored per heptatonic scale.

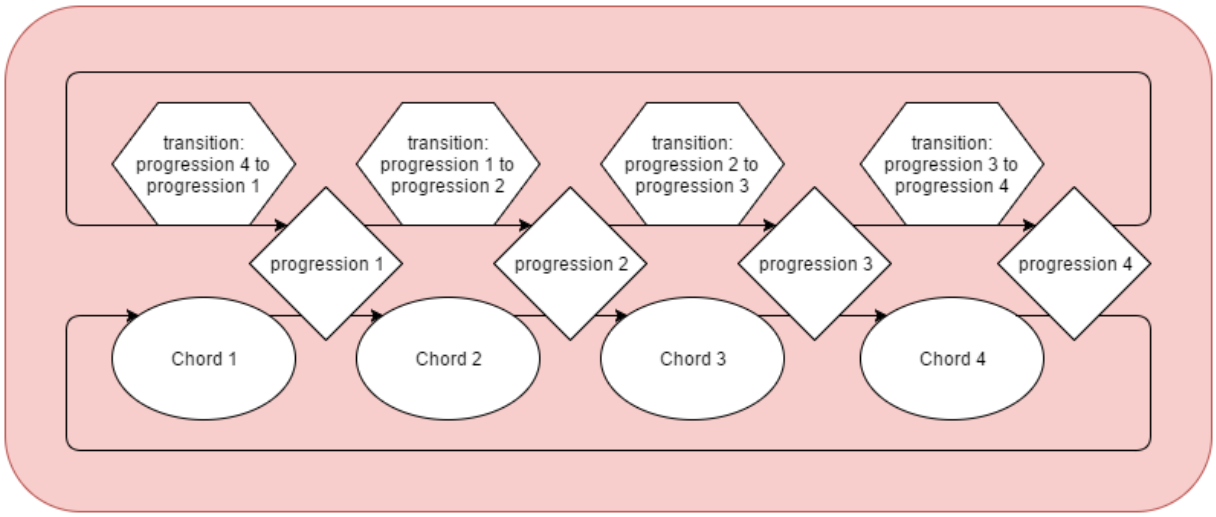


Figure 23. Visual overview of transitions between progressions.

A possible format in which these transitions can be stored is visualized in Figure 24. In this format, each row represents the progression it comes from and each column represents the progression it transitions to. Each layer in the third dimension represents one of the heptatonic scales.

3D: Heptatonic scales		TO (progression)						
		0	1	2	3	4	5	6
FROM (progression)	0	Number of occurrences progression transitions						
	1							
	2							
	3							
	4							
	5							
	6							

Figure 24. Matrix for occurrences progression transitions

Generating new chord progressions

Now that the algorithm has the information about the starting chords, starting progressions, occurrences of chords and the occurrences of transitions of progressions, it only needs the number of chords occurrence in the chord progression. This parameter is given by the user. The process the algorithm goes through, after the amount of chord occurrences is set, is listed below (also seen in Figure 25 with green boxes and numbers indicating each step):

1. The algorithm selects the first chord from the starting chord matrix (Figure 18).
2. With the starting chord selected, the algorithm multiplies the chances of occurrences of chords (Figure 18) and the starting progression data (Figure 19) to get the chances for all possible second chords. Based on these chances, it chooses a second chord.

3. With the first progression and first 2 chords selected, the algorithm uses a Markov chain (Eigenfeldt, 2010) to add new chords based on the occurrences of chords and the transitions of progressions. When there are still 2 chords left to determine to reach the total amount of chord occurrences in the chord progression, the algorithm goes to step 4.
4. To get all the possible solutions to fit the last two spots in the chord progression, the Cartesian product for each possible chord in the scale with itself is calculated. The last chord that is already chosen is put at the beginning of each result and the first chord of the already chosen chords is put at the end of each result. This way, each result of 2 chords is linked to the chords that are already selected in the chord progression.
5. For each result from the Cartesian product, the chance it may occur is calculated based on the occurrences of chords (Figure 18) and occurrences of transitions of progressions (Figure 24). Step 4 ensured that the chances can be linked to the already chosen chords in the chord progression.

If the root chord of the scale is already in the chord progression, continue to step 7.

6. If the root chord (of the scale) is not already in the chord progression, the algorithm filters all the possibilities of the 2 chords from the Cartesian product on whether it contains the root chord.
7. A possibility of 2 chords is selected based on the calculated chances. If, after step 4 to 6, the chance for each result of the Cartesian product is 0, a random possibility of 2 chords is selected (with root chord if step 6 is done).
8. The selected 2 chords are added as the last 2 chords of the chord progression.

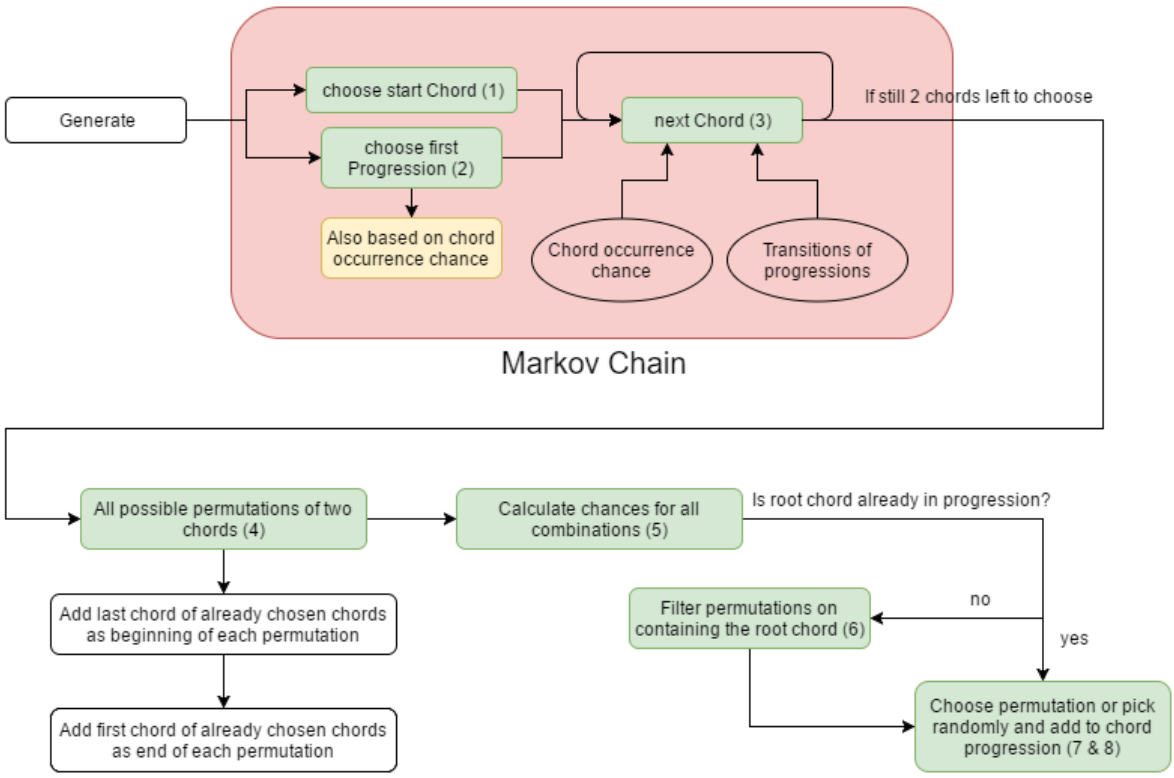


Figure 25. Process of choosing the chords for the chord progression.

Bassline

The bassline is the same as the bassline in the old algorithm. The notes of the bassline have the same length and position as the chords in the chord progression. The notes are the root notes of these chords.

Chord line

The chord line is not yet defined in the new algorithm.

Chapter 6: Melody

A melody is described in the dictionary[3] as "the aspect of musical composition concerned with the arrangement of single notes to form a satisfying sequence". The melody is therefore a sequence in which only one note can be played at once. Firstly, the analysis algorithm of the melody will be explained in 2 parts, namely the rhythm (read: starting block and length) and the pitch. Secondly, the generative part is explained with a block scheme.

Analysis of rhythm

In this work, the rhythm is defined as a combination of notes with different starting blocks and lengths. The analysis algorithm stores 3 different sources of information:

- Length of notes and rests of silence (explained below).
- The chance a note occurs based on starting block.
- The occurrences of different lengths based on starting block.

A rest of silence is defined as a silence in the melody. The length of such a rest of silence is defined as an amount of blocks no note is played. Figure 26 shows a melody with 2 rests of silence and shows all the starting blocks (numbers in blue). In this melody, the second highlighted rest of silence has a starting block of 60 and a length of 16 blocks, whereas the highlighted note has a starting block of 108 and a length of 8 blocks.

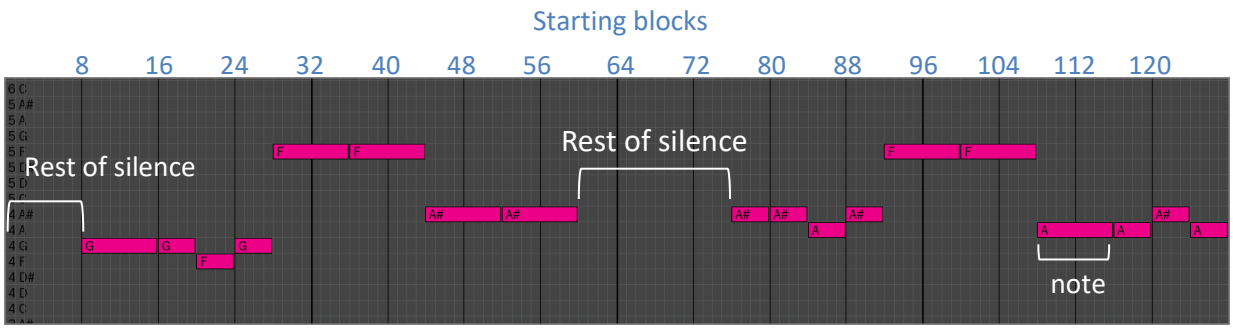


Figure 26. Starting positions and defined rest of silence and note.

Length of notes and rests of silence

The length of rests of silence and notes is one dimensional. It is a collection of the unique lengths that occur in the song database. The format can, for example, be an array that contains all the unique lengths that occur in all the selected songs from the database. Figure 27 shows such arrays based on Figure 26. In Figure 26, all notes are either of length 4 or 8 blocks, as a result that the array only contains the values 4 and 8. For the rests of silence, there are 2 rests of silence with lengths 8 and 16, which results in an array with these values. The data on lengths of rests of silence and notes can be used to store the data on occurrences of different lengths based on starting blocks more efficiently.

[3] https://en.wikipedia.org/wiki/Melody

Note sizes: [4, 8]	Empty sizes: [8, 16]

Figure 27. A possible format for note and rest of silence sizes based on Figure 24.

Chance note occurrence based on starting position

The chance of note occurrence is used to determine whether a note or rest of silence is played at a certain starting block. If the chance of note occurrence is 40 percent, there is automatically a 60 percent chance of rest of silence occurrence. The choice is only between those 2 options. A possible format to store this data can be an array of length 128, in which each index represents a starting block and its value is the chance of note occurrence. To calculate this chance, the algorithm stores all starting positions of notes and rests of silence separately. After that, the algorithm calculates the ratio between the note and rest of silence occurrence for every starting block. This ratio represents the chance a note occurs at a particular starting block (see Equation 1 below).

$$chance\ note\ occurrence = \frac{\#\ notes}{\#\ note\ and\ rests\ of\ silence}$$

Equation 1. Chance a note occurs which is calculated for every starting block.

Occurrences of different lengths based on starting position

The occurrences of lengths are used to determine the length of a note or rest of silence at a certain starting block. These occurrences can, for example, be stored in a matrix with *n* rows and 128 columns. In this format, the rows represent note or rest of silence sizes and the columns represent every starting block. The value of *n* (amount of rows) depends on the size of the array which is determined earlier in Figure 27. Figure 28 shows a representation of such a format.

	Starting block						
	0	1	2	...	125	126	127
Note or rest of silence Lengths (based on lengths that occur in song database)	Number of occurrences of notes or rests of silence						

Figure 28. A format of occurrences of different lengths at every starting block.

Analysis of pitch

The pitch is defined as a number between 0 and $12 \cdot m$ in which *m* is the total amount of octaves. The pitch is also only related to the notes, not the rests of silence. The pitch of each note is determined using 5 sources of information listed below, namely:

- Occurrences of note based on note sizes.
- Occurrences of note based on starting position.
- Chance for note to be member of the current chord played.
- Chance for repetition of pitch.
- Standard deviation for the distribution that determines the next pitch with the previous pitch as mean.

The first two of the above list have a third dimension which represents all unique chords that occur in the song database. A unique chord is defined as a unique combination expressed in the distance between the root note and the other notes in the chord. Figure 29 shows the unique combination of a major and minor chord (which are most common).

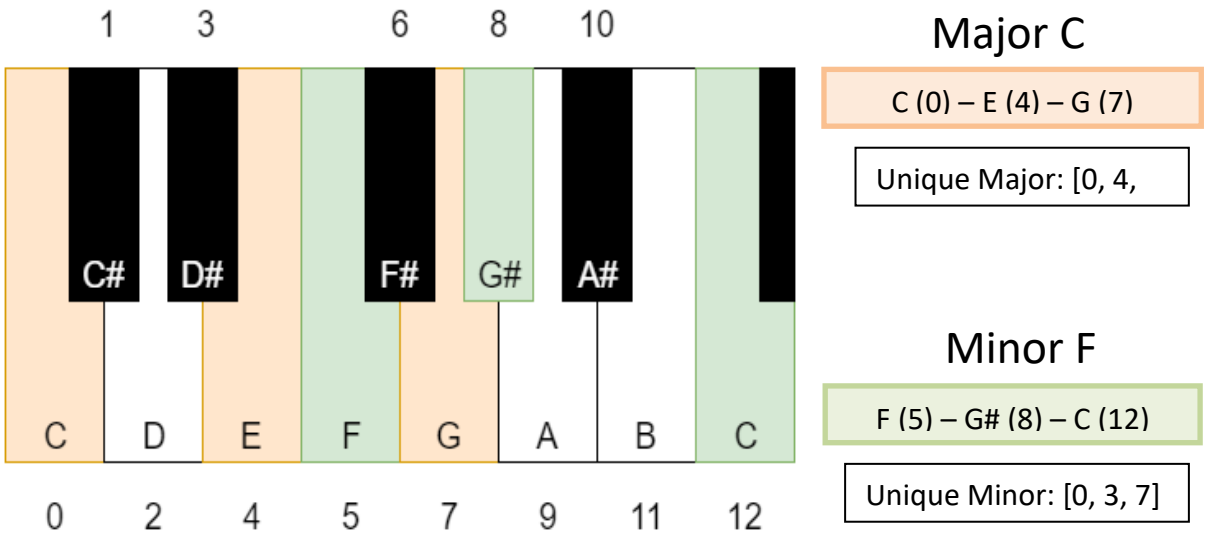


Figure 29. Explanation unique chord notation.

Occurrences of note based on note length or starting block

The occurrences of notes are stored for 2 musical characteristics, namely the note length and starting block. The information about the occurrences of notes is three dimensional. The third variable the data depends on is the unique chords (Figure 29). The format in which this data can be stored is, for example, a 3 dimensional matrix. In this matrix, the rows represent the note length or starting block, the columns represent the note relative to the root note of the chord that is played at that particular moment and the third dimension represents each unique chord in the song database. Figure 30 visualizes such a format. If, for example, a note G and a length of 2 blocks is played, while the Minor F chord is played as well; the value of the row with length 2, the third column and third dimension of the unique minor chord increments.

3D: Unique chords				Note relative to root note of the chord that is played						
				0	1	2	...	9	10	11
Note length; dependent on information from rhythm (figure 27)	OR	Starting block	0	Number of occurrences pitches						
			1							
			2							
			...							
			125							
			126							
			127							

Figure 30. A format to store occurrences of notes based on note size or starting block.

Chance for note to be part of current chord played

The algorithm also stores the chances if a note is the same as one of the notes in the chord that is currently played. These chances are based on the length and starting block of a note. It can, for example, be stored as a two dimensional matrix with rows that represent the possible lengths of

notes and the columns represent the starting block. The values of this matrix are calculated through Equation 2. Figure 31 shows such a format in which these chances can be stored.

$$chance\ if\ pitch\ is\ member\ of\ chord = \frac{\#\ pitch\ is\ member}{\#\ pitch\ is\ either\ member\ or\ not}$$

Equation 2. Chance the pitch of a note is member of the current chord.

	Starting block						
	0	1	2	...	125	126	127
Note length; dependent on information from rhythm (figure 27)	Chances that the note is part of the chord played						

Figure 31. A format to store the chances for a pitch to be member of the pitches in the current chord.

Repetition of pitch

The repetition of pitches is one dimensional and purely based on starting blocks. It can, for example, be stored as an array in which the columns represent the starting blocks. Each value in this array is a chance that a pitch is the same as the previous pitch at a certain starting block. Figure 32 shows such a format in which it can be stored.

Starting block						
0	1	2	...	125	126	127
Chances that pitch is the same as the previous pitch						

Figure 32. A format for pitch repetition.

Standard deviation for distribution to determine the next pitch

If a pitch is not repeated, the next pitch is determined through a Gaussian distribution with the previous pitch as mean. The standard deviation of this distribution depends on the height differences of the pitches that follow one another. Figure 33 shows the definition of height difference. The F of the fifth octave is defined as number 65 and the A# of the fourth octave is defined as 58. The height difference is therefore seven.

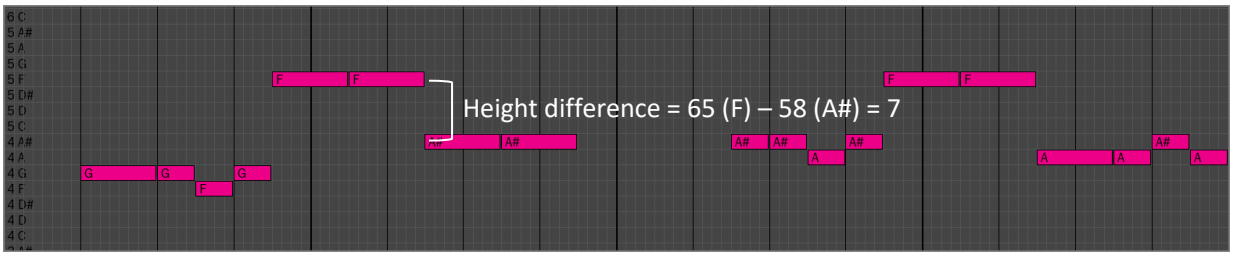


Figure 33. Definition of height difference.

When all height differences are calculated for the blocks at which notes start, the standard deviation is calculated through Equation 3. After that, each standard deviation is stored based on its starting block. A possible format is shown in Figure 34.

$$Standard\ deviation = \sqrt{\frac{\sum\ (all\ height\ differences\ for\ particular\ starting\ block)^2}{amount\ of\ height\ differences\ for\ particular\ starting\ block}}$$

Equation 3. Formula to calculate the standard deviation for the Gaussian distribution to determine the next pitch.

Starting block						
0	1	2	...	125	126	127
Sigma for height difference between pitches						

Figure 34. A format for the standard deviation for pitch distribution.

Generating new melodies

With all the data stored, the generative algorithm still needs certain parameters that are set by the user, namely the lowest and highest pitch possible. The pitches between these values form the pitch range. The pitch range contains all possible pitches the algorithm can choose from. The process which the algorithm goes through while creating a melody is illustrated in Figure 35. Each number in the Figure that is at the right of a box, indicates one of the steps that are described below. Steps 1 to 4 set the rhythm, the other steps determine the pitch.

1. The algorithm starts with the starting block set on 0. The starting block variable is used to retrieve the right chances from the analysed data that are based on starting blocks.
2. The algorithm chooses whether a note or rest of silence occurs next in the melody. It uses the chances from the chances a note occurs (Equation 1) and depends on the starting block the algorithm is at.
3. The length of the note or rest of silence (dependent on step two) is determined through the occurrences of different lengths (Figure 28) and it is also dependent on starting block.
4. The next starting block is calculated with the addition of the current starting block and the length of the note or rest or silence.

If step 2 resulted in a rest of silence, the algorithm goes back to step 2 in which a note must be chosen. It is not possible for 2 rests of silence to follow up on each other.

5. The chord that is currently played is retrieved from the chord progression.
6. The algorithm determines whether the new note is member of the chord that is retrieved in step 5. This decision is based on the calculated chances for a note to be part of the current chord (Figure 31).
7. Then the algorithm filters the pitch range (set by the user) based on the outcome of step 6. If a note is, for example, part of the chord, the algorithm is only able to choose the pitches from the notes of the current chord. All the other pitches cannot be chosen from.
8. The algorithm retrieves information from the occurrences of notes based on note length and starting block using the information determined in steps 2 to 4. For each possible pitch (filtered in step 7), the algorithm multiplies the chances of occurrence based on note length and starting block (both in Figure 30).

If the pitch that is to be determined is the first pitch in the melody, the algorithm selects a pitch purely based on the chances calculated in step eight and goes back to step 2. If not, the algorithm continues with step 9.

9. The algorithm retrieves the previous pitch in the melody.
10. It is determined whether the next pitch is the same as the previous one. This decision is based on the repetition of pitch (Figure 32) which is part of the analysed data.

If the pitch is to be repeated according to step 10, the previous pitch is assigned to the current note as well, whereafter the algorithm goes back to step 2 for the next note or rest of silence. If not, the algorithm continues to step 11.

11. The algorithm calculates a Gaussian distribution. The mean of this distribution is the pitch of the previous note. The standard deviation is retrieved from the data of the analysis algorithm based on the starting block (Figure 34).
12. The algorithm multiplies the chances of step 8 with the calculated distribution from step 11. Because of this distribution, the pitches that are closest to the previous pitch have a higher chance of occurrence. As a result, the distribution decreases the possibility that the melody will go up and down in pitch all the time. The last scenario is only possible if it happens in the melodies from the song database.
13. A pitch is randomly chosen from the chances calculated in step 12. The pitch is added to the melody with the length determined in step 3.

When the next note is determined and the starting block is not equal to the last block of the melody, the algorithm goes back to step 2 to repeat the whole process.

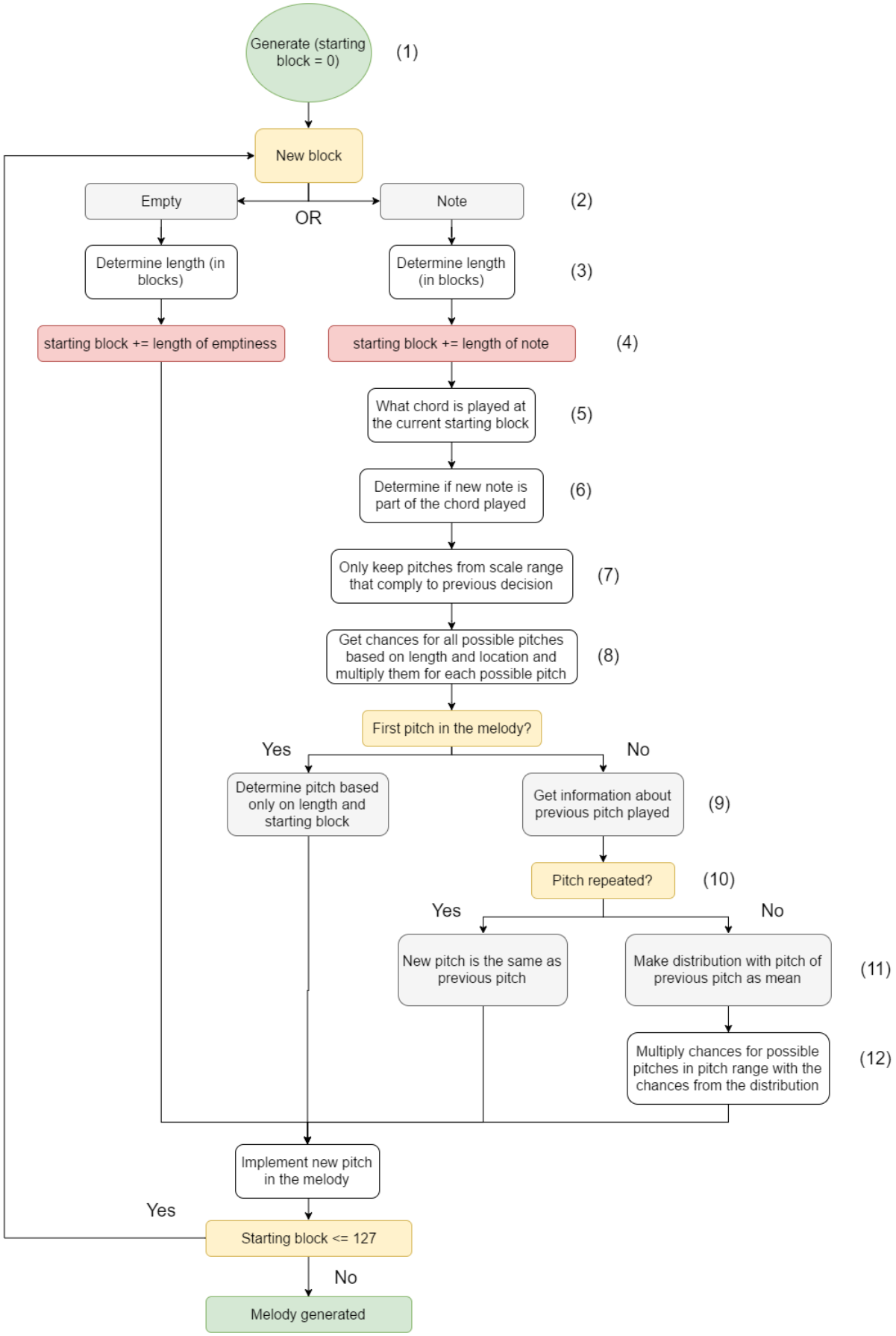


Figure 35. Visualization of the generative algorithm for the melody.

Chapter 7: Evaluation plan

JudgeMySound is a project with as main goal to generate a decent song fragment algorithmically through probability. The algorithmic part of creating a song fragment is already explained in the previous chapters. After this creation, it is still not guaranteed that the generated song fragment is decent. As already stated in the introduction, decentness of a song is based on the appreciation of the public and is therefore subjective. Each step in the creation process has influence on what notes are being played. It has therefore a direct influence on whether the song fragment will be appreciated or not. For that reason, each step has to be evaluated on its added value to the end result based on the opinion of the public. This chapter will discuss an evaluation method based on the opinion of an audience, what is stored from those opinions and how the song fragments can differ from one another. Within this evaluation method, there are 2 different approaches to vary chord progressions and notes in the song fragments, namely:

- Leaving out certain steps in the generative algorithm.
- Using different song databases from various genres.

Evaluation method

For the evaluation method, an audience of dozens of people is needed. This audience does not necessarily need to contain musicians. Everyone in the world is somehow connected to music and is able to give their opinion on it. The evaluation method itself is quite straight forward. Every person from the audience gets to hear two song fragments and needs to decide which one is better. This decision is purely subjective. However, if more people give their opinion, more data will be available and the evaluation will be more reliable (Wu, 2013). The two song fragments they hear, differ either in creation method or they are generated based on different songs. These are two different categories and cannot be compared with each other. It is for that reason only possible to compare two different creation methods with each other or two song fragments with a different song database. The evaluation method can for example be implemented in a website or application. Figure 36 shows a visualisation of the evaluation method.

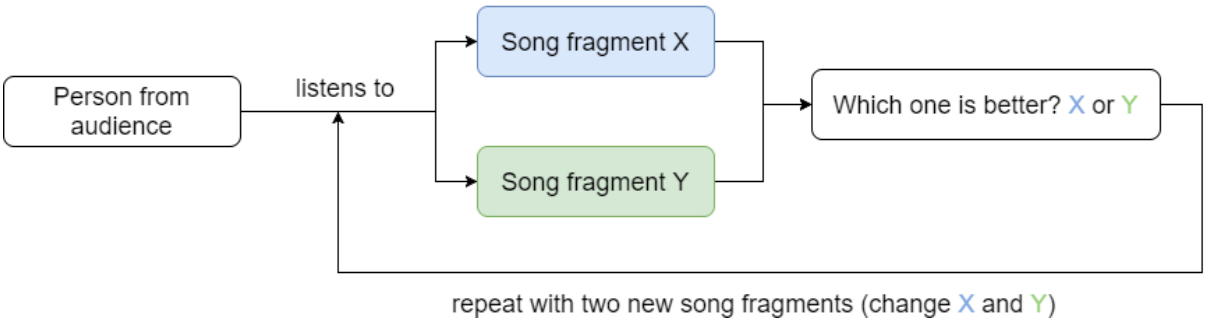


Figure 36. Visualisation of evaluation method from the audience its perspective.

The two categories test different aspects of the algorithm. The different song databases are used to test the base of the generative process. This is done by replacing the song database partially or completely with different songs while generating. If most of the song fragments that are generated with this method are rated as not decent, the base of the generative process is at fault and has to be modified. It is important that the base of the generative process is considered as decent, because the

different creation methods are an addition on the base of the generative process. For that reason, the base of the generative algorithm is tested first. Only if the results of these tests are plausible, the different creation methods can be tested as well.
The different creation methods are tested separately for the chord progression and melody and are therefore split in two new categories. Since the melody is based on the chord progression, the creation methods of the melody are tested on existing chord progressions retrieved from the song database.
Each category has multiple batches with 10 song fragments per batch. Each batch represents a different song database or an unique combination of steps that are left out in the creation method. Song fragments within a batch cannot be compared with each other. Figure 37 shows a visualization of the categories (blue boxes) and batches (green boxes). The amount of batches for the song databases is self-determined, whereas the amount of batches for the chord progression and melody depend on the amount of combinations of steps that can be left out of their generative process.

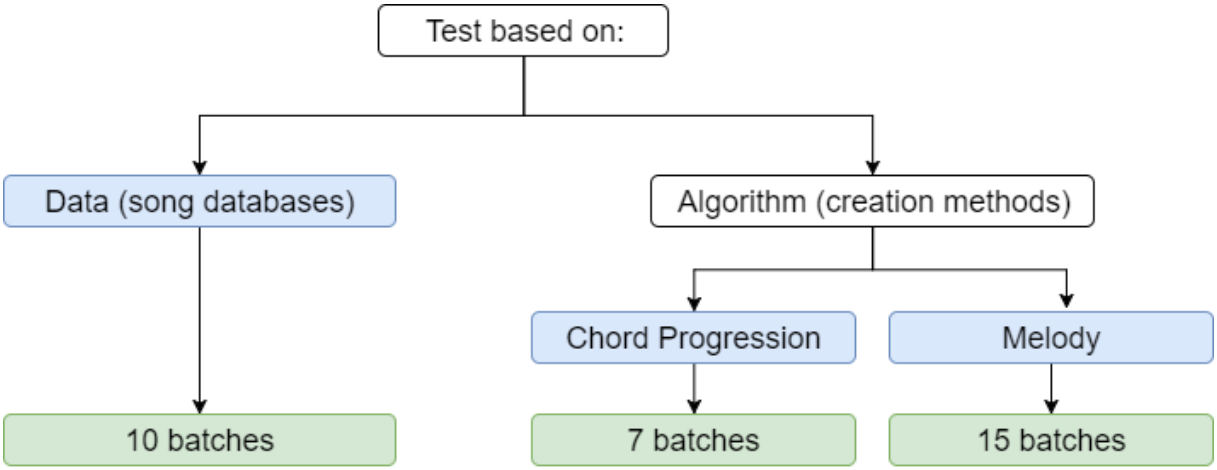


Figure 37. Categories and their batches.

During testing, each batch within a category is compared to every other batch in that same category at least once. This means that the song database test needs at least 45 comparisons, the chord progression test needs 21 and the melody test needs 105. These tests can be done with the minimum amount of comparisons, although the more song fragments are compared, the better the results will be (Wu, 2013). If every person in the audience would compare 2 song fragments 10 times, the audience should contain at least 18 people.

Storage of opinions of audience

Each song fragment has four variables that are stored, namely:

- A unique identification code.
- The creation method or song database that is used to create the song fragment.
- The amount of times it has been graded in total.
- The amount of times it has been graded as better than another song fragment (with the evaluation method shown in Figure 36).

With the last two variables in the list above, an evaluation algorithm uses Equation 4 to calculate the percentage of being better relative to the amount of times it has been graded in total. These

calculated percentages indicate which song fragments are best. The closer the value is to 1.0, the better the song fragment is.

$$Percentage\ better = \frac{\#\ graded\ as\ better}{\#\ graded\ in\ total}$$

Equation 4. Formula to calculate the percentage of being better than other song fragments.

If each batch has at least 1 song compared with another, the average percentage of each batch can indicate which batch is best. Since every batch represents a method or different database, they can now indicate which methods or databases are best, relative to the other batches.

Different creation methods

The chord progressions are evaluated on their own and the melodies are evaluated based on chord progressions that occur in the song database. To evaluate the chord progressions and melodies, some steps of the generative process are left out. For each song fragment, 0, 1 or multiple steps are left out. The steps that can be left out for the chord progression are listed below. Each bullet point also indicates which step of Figure 25 is influenced or changed:

- The chord occurrence chance can be left out, while choosing the progressions (steps 2 and 3).
- The Markov chain functions for the whole chord progression. The last 2 chords are not chosen separately (step 4 is left out).
- It is not mandatory that the root chord is part of the chord progression (step 6 is left out).

Without these steps, the generative algorithm can still function. The other steps form the base of the chord progression. These other steps will therefore only be tested by choosing different song databases and not through different creation methods.
The melody can be divided into 2 parts, namely the rhythm and the pitches. The only other creation method that is used for the rhythm is randomized selection. All the lengths are chosen randomly starting at the block the previous note ended. The pitches on the other hand, have steps that can be left out, which are listed below. Each bullet point indicates which step of Figure 35 is influenced or changed:

- It is not determined if a note is part of the chord played (steps 6 and 7 are left out).
- The chances based on length and location are replaced by random values (step 8).
- The pitch repeat chance is left out (step 10).
- The distribution is left out (step 11).

For the chord progression, as well as for the melody, every combination of steps that can be left out is generated multiple times. Within these combinations, the song fragments will still differ from each other, because the generative algorithm is based on chances. That is the reason why there are multiple song fragments per combination, so that a bad generated song fragment has less influence on the end result of the evaluation process.
Whenever all song fragments are generated, the audience can start comparing the song fragments in the evaluation method.

References

Biles, J. A. (1994). GenJam: A genetic algoritm for generating jazz solos. *ICMC*(94), 131-137.

Bradly, I. L. (1971). Repetition as a factor in the development of musical preferences. *Journal of Research in Music Education, 19*(3), 295-298.

Castelvecchi, D. (2016, October 5). Can we open the black box of AI? *Nature, 538*(7623), 20-23.

Eigenfeldt, A. (2010). Realtime generation of harmonic progressions using controlled markov selection. *Proceedings of ICCC-X-Computational Creativity Conference*, 16-25.

Engelhardt, T., & Pasquier, P. (2010, December). *Visualizing a hit - InfoVis Final Project*. Retrieved January 31, 2017, from https://sites.google.com/site/visualizingahit/home

Ewer, G. (2010, May 5). *How to Make a Chord Progression Work Every Time*. Retrieved January 16, 2017, from Secrets of songwriting: http://www.secretsofsongwriting.com/2010/05/05/how-to-make-a-chord-progression-work-every-time/

Gore, A. (2009, August 24). *How to arrange a great bass line for any chord progression*. Retrieved January 16, 2017, from Sydney bass lessons: http://sydneybasslessons.com/lesson1.html

Jacob, B. L. (1996). Algorithmic composition as a model of creativity. *Organised Sound, 1*(03), 157-165.

Joutsenvirta, A., & Perkiömäki, J. (2010, December 1). *Chord relationships*. Retrieved November 20, 2016, from Music Theory 1: http://www2.siba.fi/muste1/index.php?id=92&la=en

MacCallum, B., Mauch, M., Burt, A., & Leroi, A. (2012). Evolution of music by public choice. *Proceedings of the National Academy of Sciences, 109*(30), 12081-12086.

Margulis, E. H. (2014, July). *On Repeat: How music plays the mind.* New York: Oxford University Press.

Mauch, M., Noland, K., & Dixon, S. (2009, October). Using musical structure to enhance automatic chord transcription. *ISMIR*, 231-236.

Nielsen, M. (2017, January). *Neural Networks and Deep Learning.* Retrieved January 5, 2017, from http://neuralnetworksanddeeplearning.com/chap2.html

Quick, D. (2014, 01 22). *Kulitta: A framework for automated music composition.* (Doctoral dissertation, Yale University).

Tymoczko, D. (2012). The Generalized Tonnetz. *Journal of Music Theory, 56*(1), 1-52.

Wu, G. (2013, August 7). *Why more data and simple algorithms beat complex analytics models*. Retrieved January 31, 2017, from http://data-informed.com/why-more-data-and-simple-algorithms-beat-complex-analytics-models/