**FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE**

Zadání bakalářské práce

Název:	Generování a vizualizace hudby
Student:	Maksym Botsuliak
Vedoucí:	Ing. Radek Richtr, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je vytvořit generátor notového zápisu hudby, která bude převedena na zvuky a zároveň vizualizována. Soustřeďte se především na kompozici a estetickou hodnotu hudby.

1) Proveďte rešerši adaptivního generování hudby.
2) Proveďte rešerši způsobů vizualizace a zvuku.
3) Analyzujte současná softwarová řešení pro generování a vizualizaci hudby využívající netriviální způsob vytváření dat.
3) Navrhněte způsob efektivního provázání generátoru a vizualizace.
4) Implementujte prototyp realizující toto řešení esteticky a netriviálně.
5) Výsledný prototyp vhodně otestujte.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 4. února 2021 v Praze.

Bachelor's thesis

Music visualization

Maksym Botsuliak

Department of software engineering
Supervisor: Ing. Radek Richtr, Ph.D.

May 20, 2021

Acknowledgements

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 20, 2021   . . . .

Citation of this thesis

Abstrakt

Tato práce se zabývá vývojem prostředí pro generování a vizualizaci hudebních skladeb. Zaměřuje se zejména na výběr kombinací algoritmů tvorby hudby z využitím softwarového syntetizátoru (pro syntéz hudby). Proces vývoje algoritmu pro generování hudby je zaměřen na analýzu stávajících řešení a nalezení jejich optimální kombinace. Algoritmus spočívá v analýze existujících skladeb, které uživatel nahraje, ale existuje také možnost generování bez vstupních dat. Metoda implementace tohoto algoritmu (také metoda vizualizace) spočívá v analýze stávajících vizuálních řešení v kombinaci s požadavky na agilní vývojové prostředí. Součástí praktické práce (kromě samotného vývoje softwaru) je testování a analýza výsledků. Výsledný program je optimalizován pro provádění změn v algoritmu generování hudby a také přizpůsoben pro rychlou vizuální analýzu generované hudby.

Klíčová slova   Generování hudby, hudební vizualizace, Unity 3D, hudební analýza, formát MIDI, kontextová gramatika, Synthesia

Abstract

This work deals with the development of an environment for generating and visualizing musical compositions. Mainly, this work focuses on the selection of

combination of music creation algorithms using a software synthesizer to synthesize music. The process of developing an algorithm for generating music is focused on analyzing existing solutions and finding their optimal combination. The algorithm consists in analyzing existing compositions uploaded by user, but there is also an opportunity of generating without input data. The method for implementing this algorithm (also the visualization method) is selected by analyzing existing visual solutions, in combination with the requirements for an agile development environment. Part of the practical work (in addition to developing the software itself) is testing and analyzing the results. The resulting program is optimized for making changes to the music generation algorithm and is also adapted for quick visual analysis of the generated music.

Keywords   Music generation, music visualization, Unity 3D, music analysis, MIDI format, CSG, Synthesia

Contents

List of Figures

Introduction

The topic of the work is the problem of fully automatic real-time generation of music. The result of this work will be an application that implements the algorithm derived from the analysis of sources. Such an application is a rather valuable information product for the reason that there is still no effective way to automate the production of musical compositions. With such a tool, many publishers who use music to create content will be able to significantly reduce the cost of creating music of certain genres. Since this problem has been voiced many times (including many years ago), there are many published methods of solving this problem among open sources. Therefore, the purpose of this work is to analyze, compare and find the most optimal way to generate and visualize music.

Since, within the limits of the compatibility of modern technologies and the perception of the human brain, it is impossible to simulate an absolutely ideal automatic melody generator, the goal of this project is to create the basis of the algorithm as accurately as possible to make it more universal before it's subsequent infinite expansion.

Briefly about music

This chapter will reveal the topic of generating music in more detail. Firstly, the historical basis for shaping the modern form of this small area of programs will be presented. It will also provide the basic necessary terms so that a person who encounters this area for the first time could better navigate in it.

1.1   The history of generated music

The first attempts to automate the process of writing music began when the music was just beginning to look like something systematic. In ancient times, when the first rules for harmonizing music and building compositions were born, musicians of those times began to look for ways to automate this process, at least partially. For example, Pythagoras, while trying to create some musical rules, introduced a number of strict acoustic laws according to which music could be written.

More famous example in history where the composer tried to automate the writing of music is Wolfgang Mozart. He invented the famous musical dice game: rolling a dice predetermined the order of fragments of music (not the sounds themselves, as many believe).

However, the real development in music generation was happening in the later period, when humanity gained access to computing power. One of the most significant figures in this area is the man named Iannis Xenakis. This man was the first to introduce the methods of probability theory into the technique of musical composition. To create a prediction of subsequent notes, he used game theory, stochastics, and Markov chains to create a melody. After he published his works in this area, many famous composers also began to use a variety of mathematical processes to construct compositions.

Nowadays, there are many different programs for generating works of varying quality. Some of them will be described in more detail below.

1.2 Introduction to Music Theory

This section will sequentially describe the basic musical concepts the understanding of which is necessary for the analysis below. It is important to familiarize yourself with the following basic terms at least: phrase, subject, counterpoint, musical development. These concepts form the foundations of the musical form.

A musical form is a characteristic assigned to a particular piece of music. So the form of each composition is absolutely individual. Form is not a collection of all the characteristics that can be assigned to a work, but it completely defines them. Despite this, the number of rules by which this form is formed is very limited. Because of these limitations, many music compositions have some kind of similarity in their structure. This, in turn, allows you to derive a set of rules by which it becomes possible to generate music. The basic variables for the form creation are melody, harmony and rhythm of their interaction.

Like human speech, music has larger and smaller structural units into which it can be divided. The smallest of these units is the motive. A motive is one or (most often) several sounds combined around a strong beat.

A sequence of two (rarely three) motives, combined on the basis of a metric pattern, are called a phrase (in separate sources "motive group"). Musical phrases, in turn, are combined into sentences. A sentence is the minimum logical unit for a form of presentation such as music. For this reason, the pattern of repetitions of a small musical section will be built around the musical sentences. The sentence is four or eight bars long. Harmoniously, the sentence ends with a cadence, which means either a question or a statement.

A combination of several sentences forms a period. The period is a complete musical thought and consists of several, logically connected sentences (interrogative or affirmative), which are logically related to each other.

There are also divisions of the composition according to the purpose of each part. For example, a section is a part of a work that performs a function. That means, that there are exposition (main) and subsidiary (secondary) sections. While the expositions are used to guide the main narrative, the subsidiaries are necessary for communication, transitions and developments.

Harmony is the unification of sounds into consonances and the subsequent coherent combination of such consonances. Consonance is the simultaneous sounding of several sounds. A consonance of three, four or five sounds, different in pitch and name, is a chord if sounds are arranged in thirds (with an accuracy of octave displacements).

Of the various properties of harmony, functionality has the main formative value. The main (tonic) tonality plays the role of tonic in the broad sense of the word, that is, it is stable. In the main key, the piece for the most part begins and almost always ends with tonic. Therefore, it serves as a centralizing, subordinating principle.

Any other tonality in the composition is unstable in the broad sense of the word and will be called subordinate. Appearance of a subordinate tonality is an incentive for further development.

The concept of "rhythm", in a narrow sense, means an organized sequence of sound duration. Rhythm holds together different sounds and chords and creates meaningful groups out of them.

The following basic rhythmic "measurements" are used in music: whole note, half, quarter, eighth, sixteenth, thirty-second, etc. All these values are relative, each of them is half the previous one and twice the next; their absolute duration is approximately set by the designation of the tempo, which is the speed of performance.

Research

This chapter will describe the existing solutions (the principles and algorithms used for them) that are required to implement music generation. Also, this section will properly describe the existing music visualizers and principles that guided their creators.

2.1 Harmonization algorithm according to musical theory

This method is the most common among implemented automatic music generators. This way of implementing the algorithm is to transfer as much of the restrictive musical theory as possible into the program and then randomly generate a melody within the given rules. This principle is used for the most of all works.

The author with nickname Armaxis [1] invented and implemented an algorithm for generating music, which perfectly reflects this principle. Brief description of the theory used:

At first, the author has split the problem into three parts:

- Generating the main melody.

- Harmonization of the main melody.

- Construction of secondary melodies of theme according to the main melody.

The first part contains the implementation of some basic rules of musical theory. The first rule used is that the entire composition is generated entirely in one scale (since in this implementation the user is able to change all settings of the composition during playing, the scale can change, but only with user intervention). The principle of generating euphonious sequences of sounds here is based on a concept in music that defines frequently used patterns.

A pattern in this context is the difference (in semitones) between every two consecutive sounds in a melody. The implementation consists in defining some standard patterns and randomly choosing some of them from this variety. This is followed by the process of breaking some patterns into shorter sequences of other notes, which as a result give the same transition between the heights of the sounds, only including new notes between the extreme ones.

This whole motive is defined for four measures, and then either repeats or changes (according to the certain structure of the current musical sentence).

The second step of this algorithm is melody harmonization. Harmonization of the resulting melody consists in the correct construction of chords. For this, many musical assumptions have been made, including:

- At the end of the fourth measure, chords always go to the dominant.

- At the end of the eighth measure, chords always go to the tonic.

- Only triads are used.

Also, rules were drawn up for transitions from one chord to another. That is, a chord grid was defined, according to which the harmony module determined the chord sequence.

The final stage generates secondary melodies. This part uses an algorithm for generating a sequence of sounds similar to the first paragraph - using the simplest patterns and from the chord as a basis, a harmonious sound of minor voices is achieved.

Many other researchers have implemented similar algorithms as well. Each implementation is distinguished by a unique choice of implemented rules and a scheme to which the final melody must correspond. Some authors, like Sadler [2] have chosen a melody generation method based initially on generating a chord grid. And only then the melody itself can be generated.

2.2 Generating a composition using context-sensitive grammar

An online service for generating music according to a given genre is available on the Wolfram website [3]. This algorithm is based on the idea described in the book "A New Kind of Science" by Stephen Wolfram [4].

This method is based on setting a limited number of rules, which form a final 2D figure from an initial random set of bits (in the example, only one bit is indicated). After turning the resulting pattern on its side and cutting out only one middle within a given height, we get a finished MIDI pattern. After some processing (recalculation of the pitch of the notes within the musical scale, removal of a large number of adjacent sounds, and so on), a complete melody is obtained, within which the user can catch some regular repetitions.

Genre defines only customizable (by user) characteristics of the algorithm (instruments, scale, tempo, etc.).

A similar algorithm for generating music is used by other authors. For example, a user SkywalkerY [5] published an algorithm that fully generates a song according to certain rules, which are also designed as a context-sensitive grammar. Unlike the previous version of the algorithm, this case does not use random generation of the initial line with the subsequent generation of rules. The algorithm looks a little simpler here:

- A ready-made string is taken from which the rules for generating a particular letter are gradually obtained. Since ultimately this process will lead to the rules according to which we will get the only generated melody - the same one with which we started, the author has implemented an algorithm that does not extract the rules for determining the next character of the sequence one-by-one, but only reaches some stage this process, when most of the random meaningless connections are excluded.

- The first sound value is selected and then by means of a random choice between the rules defined in the first paragraph, the subsequent sounds are also determined.

- The same is calculated separately for the duration of sounds.

Some combination of the previous two algorithms is the idea described by user junshern [6] called Fractal Music Principle. The algorithm builds a pyramid similar to the one described in Wolfram's work. Only if in that case the rules were clearly defined for each given set of pixels and the number of notes in the next row was equal to the number of notes in the previous one, then here is a somewhat simplified version of these set rules, which sounds like "When playing a note, the sequence, defined by this note, is added to the end of the composition". Thus, the result is an endlessly expanding sequence of notes in real time, which at the same time is also playing.

User junshern [6] also wrote about another similar generation method, which is called "Markov Music". This is a simplified version of the algorithm from the previous one, which has been already mentioned. Its essence lies in the process of constructing a graph of the possibilities of transitions between two pitches according to the defined part of melody. But unlike the previous author, here the depth of checking the rules is always equal to 1. Thus, the algorithm determines a finite number of possible transitions between states, randomly choosing between them (there is also an increasing chance of choosing the order, that is the most often found in the composition, by defining multiple arrows between these states).

2.3 Genetic algorithms

Most authors also mention the possibility of implementing a genetic algorithm. For example, user junshern [6] describes an example of such an algorithm. The way of implementation is quite standard - there are determined parameters of the assessment and all possibilities for mutation. Then, after a certain number of generations, created pieces are able to produce some pleasant sequence of sounds.

2.4 Other used techniques

This section will describe other generation algorithms and examples of implementations of this algorithm that are worth mentioning, but have not been categorized and therefore not available in other sections.

2.4.1 Key Phrase Animation

The principle that simplifies some of the stages of music generation is described by Peter S. Langston [7].

The algorithm is that it is not necessary to generate all notes to create a melody. It is enough just to write the key sounds that are at some interval from each other. Then you need to fill the interval between these notes with other notes with shorter duration. The author also uses some predetermined differences in pitch between adjacent notes. A predetermined tempo was also used to determine how long the added notes should be.

2.4.2 Auto Composer Muzz

This version of the music generator has both strengths, which will be partly used in this work, and weaknesses, which will be analyzed as necessary.

The algorithm is a combination of the previously mentioned methods of making music. The main supporting mechanism for generating here is the programming of the musical structure and the subsequent randomization of music within this structure.

The method includes those necessary structural components that cannot be dispensed with when creating music in this way - melody, accompaniment and drum component. The last part will not be used in the developed version of the algorithm due to inconsistency with the current musical instruments.

This algorithm implements a huge variety of all kinds of relationships: tables of paired chords, a table of scales that convey moods, priorities and functions of various chords, transitions, developments, and so on. In this regard, the author has done a tremendous job and was able to correctly apply musical theory to establish the rules for generating music.

However, the algorithm also has a rather serious drawback: within the environment of the given rules, the music is determined completely randomly. The author tried to reduce the influence of this flaw on the quality of the music being created as follows - more dependencies were added between the selected parameters. For example, a relationship was introduced between mood (which is almost entirely determined by the scale) and priority chord progressions. Thus, when trying to introduce more and more rules for communication, the author each time increases the number of dimensions of parameters by 1, which very quickly complicates the algorithm.

This problem is one of the main controversial points in this theory of music generation. Here it is necessary to find a balance between setting a certain number of rules and between the analysis of finished works to determine the bias of the randomness of choice when randomizing the prepared structural units.

2.5   Music visualization techniques

A person perceives much more information visually than by ear. Therefore, various applications that can play animations to the beat of the playing music are very popular. There are endless ways to visualize music. For example, there are many templates on the [8] site.

The visualization of music happens for different purposes and using different methods. Therefore, to begin with, we should consider the various reasons for using visualization algorithms. Since each of the algorithms fully describes its scope and satisfies the requirements of certain groups of users, it will be sufficient to describe a list of different functional implementations of this problem.

The first group of applications uses the most common rendering method. The essence of these programs is to generate animations in real time. The generation takes place depending on the parameters that are obtained from playing the music in real time. Removing these parameters from the input data stream is the main task in drawing up visualization algorithms. In the example above, it is possible to familiarize yourself with the implementation options for such a solution. The details of such solutions and methods for extracting parameter changes in musical works will not be described here for the reason that the current work will not operate with real works, only with their midi versions.

First, let's get acquainted with the methods of extracting the values of some parameters of the finished product. For the sake of completeness, we will consider a raw music file - for example, wav format. Let's take a look at the most important parameters in this process, according to user *gheljenor* [9].

Volume. Volume is the most obvious parameter to extract when analyzing sound. From a physical point of view, it characterizes the energy transmitted by the sound wave, and, therefore, determines the power of the sound flow at the current moment. On the graph of the sound flow, the power is determined by the amplitude of the graph. The simplest implementation of the algorithm for extracting this parameter will be to return the average value of the modulus of the deviation of this graph from the axis for a certain period of time. Any similar algorithm that works with local maxima will also work.
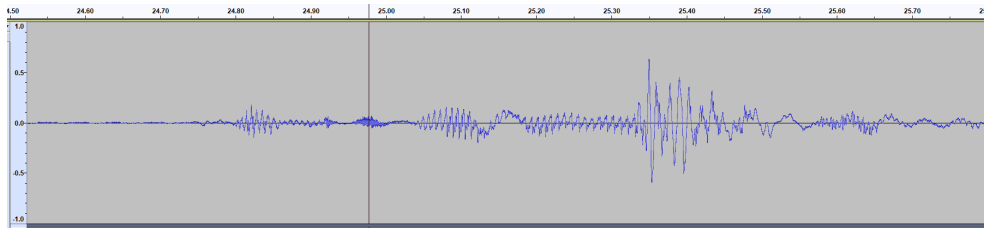

Figure 2.1: Example of recording in Audacity. You can evaluate the loudness of various parts of the composition by the amplitude of the graph.

Frequency. This parameter is not so obvious anymore, but it is the most commonly used one. Frequency is the main indicator when visualizing a composition for the reason that almost all dependencies and rules in music are related to the pitch - it is the same frequency. The pitch on the graph is determined by the distance between adjacent peaks. The degree of compression of the sinusoid, the number of full cycles of the microphone membrane per unit of time - all this is the pitch. It is important to note that every note in music has a well-defined frequency. For example, for A4 note this is 440 Hz. Real instruments, of course, have a different timbre of tone than a pure sound, but this is achieved by adding overtones and frequencies close in value (for example, the A4 key triggers three strings at once - two adjacent ones differ in frequency by about 1 %). Some music analyzers take this into account, but, in general, to get the pitch, it is enough to count the intersections of the axis and the graph.

Timbre. This is the characteristic that was already discussed in the previous paragraph. In fact, this characteristic completely decomposes into the previous two components in the digital world. However, this characteristic should definitely be mentioned due to the fact that it fully describes the difference between the tools. The same note sounds completely different on different instruments for this reason.
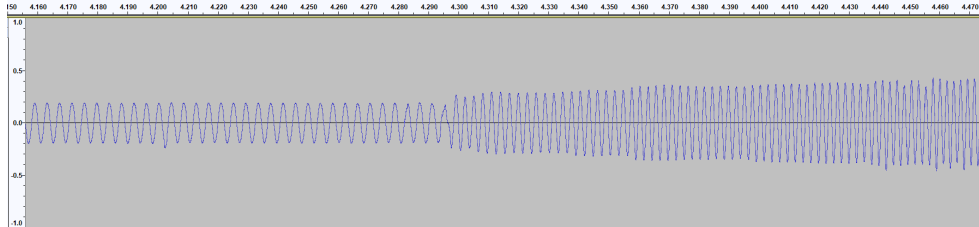
Figure 2.2: An example of sound recording in Audacity. In the center of the graph, you can clearly see the transition from low frequencies to higher ones. In this case, increasing the volume at the same time as the frequency is just a coincidence.
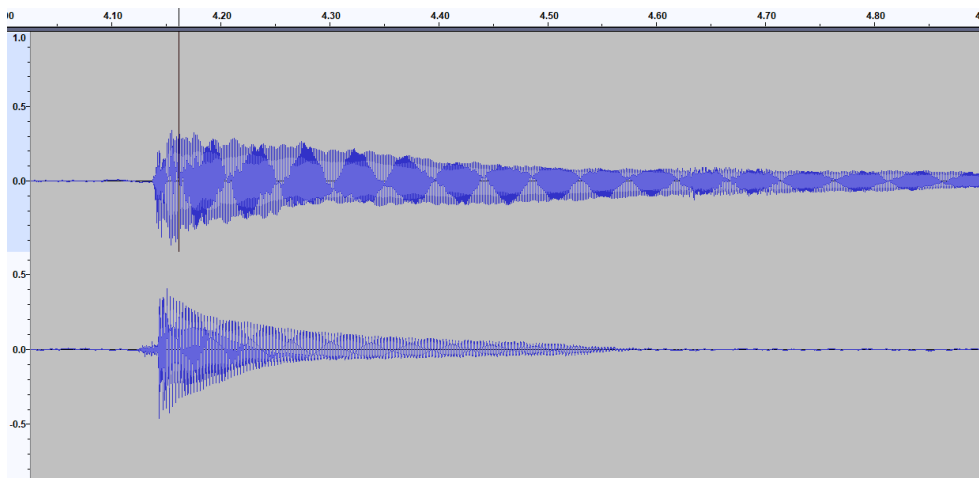

Figure 2.3: An example of sound recording in Audacity. Here you can clearly see the difference between the piano tone and the banjo tone. This screenshot is especially successful, because at this scale there is a phenomenon of interference and we can see the beats that occur due to close frequencies that sound simultaneously.

13

However, to extract more information, it is necessary to transform the current composition into harmonic components. For example, the example shows what a single key press looks like on a piano. But the problem is that we already receive the sum of all signals as input. That is, one key was pressed on the piano, which struck three strings at once and produced three decaying harmonic graphs. But we got only the final sound, which is only the sum of the three harmonic values. To do this, you need to do some processing on the input file and get the original harmonic functions.

This is solved using the **Fourier transform**. According to the post on Habr [10], we work with this transform as follows. We select a short frame (interval) of the composition, consisting of discrete samples, which we conventionally consider periodic and apply the Fourier transform to it. As a result of the transform, we obtain an array of complex numbers containing information about the amplitude and phase spectra of the analyzed frame. Moreover, the spectra are also discrete with a step equal to (sampling frequency) / (number of samples). That is, the more samples we take, the more accurate frequency resolution we get. However, at a constant sampling rate, increasing the number of samples, we increase the analyzed time interval, and since in real musical works the notes have different sounding duration and can quickly replace each other, they overlap, so the amplitude of long notes "overshadows" the amplitude of short ones. On the other hand, for guitar tuners, this method of increasing the frequency resolution works well, since the note usually sounds long and alone.
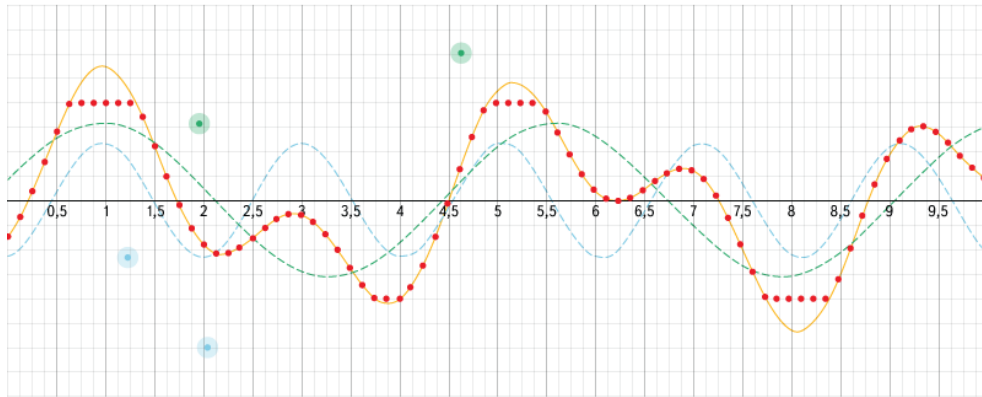

Figure 2.4: An example of a simple Fourier transform, the data is interactively generated using Desmos service. [11] The signal recorded by the microphone is marked in red. Yellow - the summed signal, which is visible to us in Audacity. The blue and green dashed lines are the original functions obtained by us using the Fourier transform.

There is also a rather simple trick for increasing the frequency resolution - you need to fill the original discrete signal with zeros between samples. However, as a result of such filling, the phase spectrum is strongly distorted, but the amplitude resolution increases. It is also possible to use Farrow filters and artificially increase the sampling rate, however, it also introduces distortions in the spectra.

The duration of the frame in question is usually approximately 30 ms to 1 s. The shorter it is, the better resolution we get in time, but the worse in frequency, the longer the sample, the better in frequency, but worse in time.

2.6 Fractal audio visualizer

When analyzing possible solutions for the visualization of music, mention should be made of the work "Fractal audio visualizer" by Radka Hošková [12]. This entire section will be devoted to a succinct description of her work with a mention of important details that will be useful for analysis for the current work.

The project is an audio file visualizer that uses Spotify API to get more loyal data from raw files. Most of the implementation lies in the processing of data obtained using this API. To visualize changes in music parameters, the author uses fractal visualization algorithms.

Now a little more detail. In the theoretical part, the author analyzes all the basic terms that are necessary for the subsequent analysis of musical information. This first part of the theory can be divided into four categories:

- Description of digitizing sound. It describes terms and processes that are important to understanding the modern way of simulating natural sound sources in a digital environment. The author talks about how audio sampling occurs and describes the necessary sampling frequency for minimum loss of information about the audio frequency.

- Ways to record sound. The theoretical part is described here, which is also necessary for this work. Here we are talking about some of the ways to record and store sound in digital form. Here, among others, a method is described for storing ordinary audio files in the form we are accustomed to. They consist of a part that describes the content of the file and a part that is the metadata of the audio file itself. Also described is the MIDI standard - a music file format that does not in any way determine the quality of a sounding file. In fact, this format is a digital analogue of musical notation - it contains information about which note was played on the instrument at a certain point in time. Also, there are many other characteristics that are fully capable of describing the music being played.
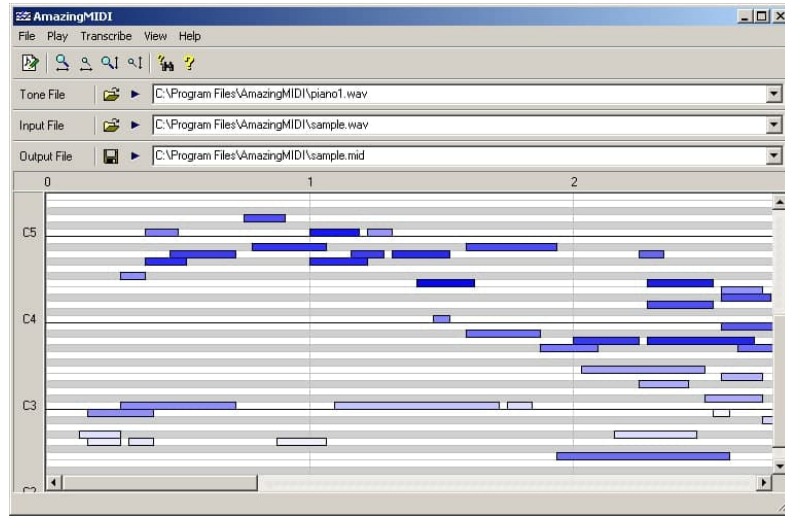
15

Figure 2.5: An example of displaying data stored in a MIDI file. In this case, we are talking about transferring to MIDI format from WAV files - a program by [13].

As the last way of storing musical information, the author described Stem - a format that stores information about several main channels at once. The information contained in the file with this extension is more than redundant for visualizing the composition separately for its components - vocals, base, synth, beats.

- Musical theory which has been already mentioned in this work. For example, a description of the characteristics of sound, Fourier transform.

- The last part is devoted directly to the visualization of music. The area that is directly involved in the extraction of musical information is described in detail. As an introduction, the principles of MIR are described here. Several examples of projects that relate to the MIR topic are described below. For example, visualizing structure in music, or trying to visualize semantic structures in classical music. Then the author briefly describes the current state of this branch of research and algorithms and methods of obtaining information that are used in the present. The section ends with a short summary of organizations and developments that relate to this area of researches.

The second section of this work contains analysis tools that can be used to analyze music files. Since this work addresses the issue of analyzing music, this section contains the bulk of the work under consideration. However, we will consider only an abbreviated description, since the data from this part will not be used in our work. The following three programs are described.
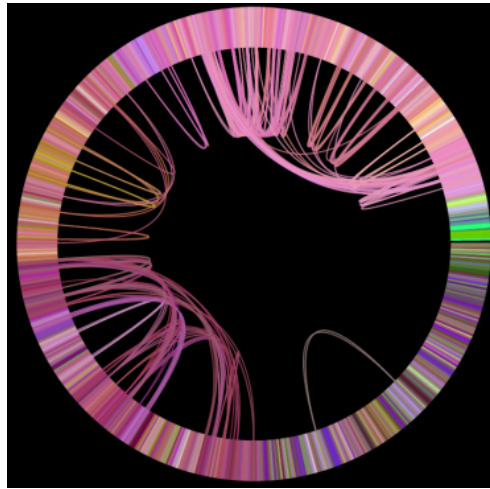
Figure 2.6: An example of work in the field of MIR: visualization of the structure of the Led Zeppelin song using the Infinite Jukebox by [14].

- **Sonic visualiser**. A standard program that contains several useful tools for visualizing music in various areas: frequency spectrum, amplitude of frequencies in a given interval, the width of the choice of melodies, a simple graph of the signal over time.

- **Spleeter**. A useful program for dividing a music file into 2, 4 or 5 separate streams, which are then easier to work with.

- **WaoN**. Program for converting WAV files to MIDI files. It also has tools for changing the duration of the music without changing the frequency, as well as GUI for the instruments mentioned above.

- **Spotify API**. API, which the author eventually uses in his project. The functionality of this API allows you to deeply analyze music, providing information about the music genre, acoustics, tempo, energy and many other variables in a short time.

The second major part of the work analyzes algorithms for generating fractals and various ways of their implementation. This section contains many theories on how to implement the main idea of this work - visualization of music using fractal generation. At the beginning of the section, information on programs for generating fractals is briefly presented, and then 6 sections are described, each of which is devoted to a separate algorithm for constructing fractals:

- **IFS**. The most common way of constructing fractals is a set of self-copies of a figure, in which one of the transformation functions is applied to each of the copies.

17

- **Kaleidoscopic IFS**. Building a fractal using iterative space displacement and mirroring.

- **L-systems**. These are fractals generated by repeatedly applying a specific grammar to an axiom that is a visual definition.

- **Chaos and dynamics**. It describes algorithms for developing a fractal system using a mathematical model called a dynamical system. For example, a Mandelbrot set is described here.

- **Random fractals**. Fractals generated using randomness. Most often they are based on self-copying with some random parameter, which partly determines one of the future characteristics of the copy.

- **Cohomology fractals**. Here we are talking about fractals, which are created using various 3D manifolds.

The following is a number of existing implementations of programs that solve the problem of visualizing music in different ways. Most of these methods are similar to the topic of a diploma and parameterize algorithms to generate fractals based on variables derived from music.

In the implementation, the author uses Spotify API. Using the obtained parameters, one of the selected types of fractals is created.

2.7 Programmable tools for visualization.

Further, in the course of researching existing solutions, we needed to analyze existing tools for visualizing music, which at least partially have a wide range of tools for interacting with the user. Ideally, a program in which you can directly program the generation method and redirect the output to a custom visualizer. There are countless instruments in which it is possible to manually set the structure of the composition and describe any piece. However, here will be disassembled those that potentially fulfill at least the condition of having the necessary API for interaction.

2.7.1 PreSonus Notion 6.

Notion 6 for notation or tablature is the environment which implements music composition and music performance, and allows to compose, play back and edit music [15]. This PreSonus product has deep integration with Studio One.

Notion 6 has interactive tools:

- **Interactive keyboard** - is convenient for entering notes and chords or auditioning a melody.

- **Drum pad** - for entering the desired drum or pitch while in step-time entry mode.

- **Fretboard** - allows to audition a melody or enter notes and chords.

- **Mixer** - function for opening the audio mixer console for adjusting individual instrument audio, editing effects, routing output to external equipment.

It allows to capture drum parts, pitches, and guitar fingerings. You can customize the number of guitar strings on the fretboard and the order of entry for each voice for the drum pad. PreSonus program also allows to choose chords from a chord library and create new chords and has cross-platform handwriting recognition, powered by MyScript for written entry.
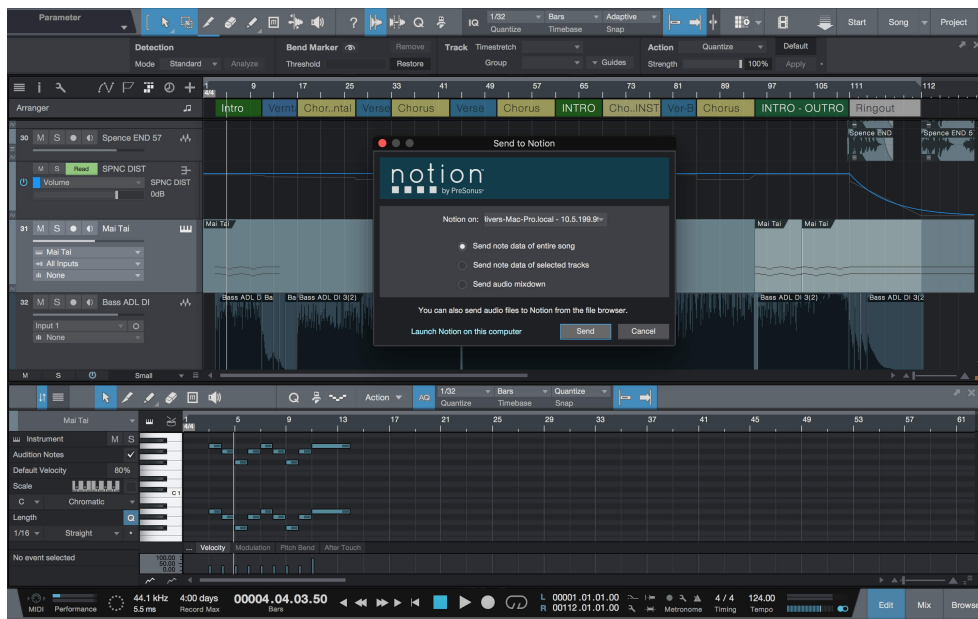
Figure 2.7:   The example of PreSonus Studio One integration with Notion 6 [15]

The opportunity to see notation and MIDI data on the same staff is realized using the sequencer overlay.

Notion 6 has additional custom rules editor for instrument specific articulations and besides that it is possible to increase sonic palette with VST Instruments and libraries or additional expansion sounds from PreSonus.

Other VST plug-ins also include such sound processing modules as PreSonus Native Effects Limiter, Compressor and Pro EQ.

19

From other features of PreSonus Notion 6 can be mentioned the possibility to mix with a DAW-style console with up to 8 individual buses, ReWire support with real-time audio and MIDI transfers to a DAW.

2.7.2 MuseScore.

MuseScore is considered to be an alternative software for music notation[16]. It features the WYSIWYG editor with audio score playback. This scorewriting software allows to create unlimited number of staves per system and has no limit for score length. Among other properties it is necessary to mention possibility of four independent voices per staff, custom text markings, import of graphics, user-defined score styles and mixer for instrument levels and effects.
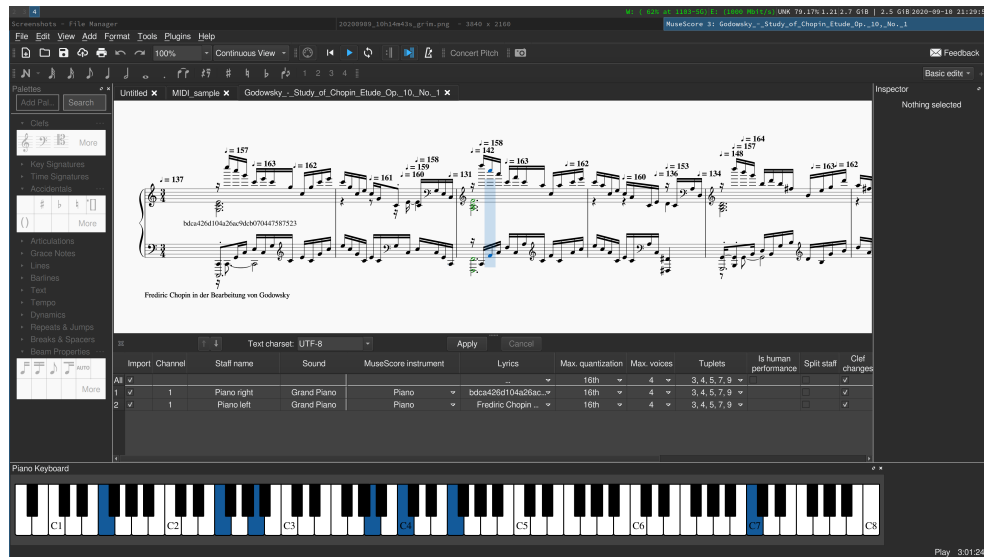
Figure 2.8: The interface of the scorewriter "MuseScore" [16]

Among the others MuseoScore's features are:

- Score creation wizard and templates.

- Automatic part extraction and transposition.

- Repeats, including segnos, codas, and measure repeats.

- Dynamics, articulations, and other expression markings, with playback support for most.

- Swing and shuffle playback.

Another advantage of this software is that the capabilities of MuseScore can be extended via plugins, and the extending repository on musescore.org contains many plugins which are regularly submitted by users.

MuseScore is based on a set of common instruments sounds (as defined by General MIDI). However whether it is necessary for realism either for wider variety of sounds, it is possible to substitute any SoundFont.

MuseScore can import and export MIDI and MusicXML files, and it can also integrate with several similar programs. Regarding formats of exported files, MuseScore can export to PDF, PNG, and other graphic formats, to WAV and other audio formats, or to Lilypond for an alternative layout and print option.

2.7.3 Unity.

Unity is an environment of cross-platform computer game development. Unity allows to create applications that run on more than 25 different platforms, including personal computers, game consoles, mobile devices and Internet applications. Unity brings together many software tools for creating software - a text editor, compiler, debugger, and so on. At the same time, the process of creating programs in Unity is extremely understandable and does not require additional knowledge, and the multiplatform nature of the engine allows you to cover as many game platforms and operating systems as possible.
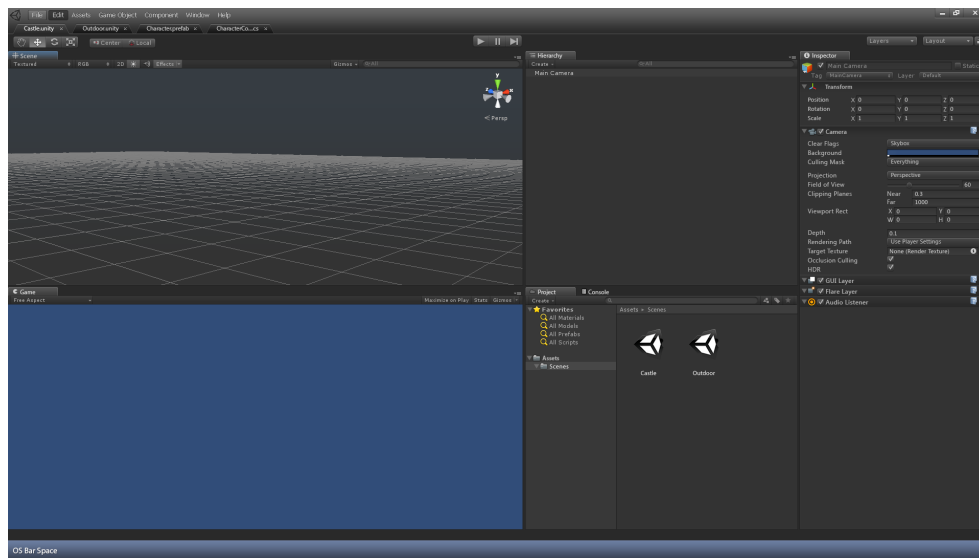

Figure 2.9: The interface of Unity

The Unity3D engine makes it possible to develop games without requiring any special knowledge. It uses a component-based approach, in which the developer creates and adds various components to them. It has a convenient Drag & Drop interface and a functional graphics editor that allow you to place objects in real time and immediately test the result.

21

Other advantages of Unity include a huge library of assets and plugins that can significantly speed up the program development process. They can be imported and exported, ready-made blanks can be added to the program. It is also possible to create your own templates.

In addition to all this, Unity 3D supports a huge number of platforms, technologies, APIs. Games created on the engine can be easily ported between Windows, Linux, OS X, Android, iOS, PlayStation, Xbox, Nintendo, VR and AR devices. Unity supports DirectX and OpenGL, works with all modern rendering effects, including the latest real-time ray tracing technology.

Audio Helm for Unity is asset that considered to be a live audio **synthesizer**, sequencer and sampler for Unity. Tools of this asset are proper for creating generative music and different musical sound effects.

Figure 2.10: Audio Helm Unity Asset

As the synthesizer Audio Helm generates dynamic audio live, for which no samples or recordings are required. It runs as a native plugin to ensure low latency, high performance, mobile ready audio.

As the sequencer Audio Helm is a tool for creating musical patterns and rhythms by playing synthesizer or sampler notes over time. It is also possible to create individual patterns inside Unity's inspector or to create them live from code to generate procedural music. Dynamic tempo is supported by the sequencer so it is also available to increase or decrease the bpm.

As the sampler Audio Helm can be used for playing an audio sample or recording back at different speeds to create musical pitches. Full spectrum piano sampler is also available via using different keyzones. Moreover Audio Helm comes with 4 drum machines each with a separate sample bank.

2.7.4 Synthesia.

Synthesia is a method of interactive visualization of a melody, in which user via using a MIDI keyboard or computer keyboard needs to press the same notes that are displayed on the screen. Synthesia's interface consists of piano keys and the melody itself, which is displayed by notes in the form of moving tiles.
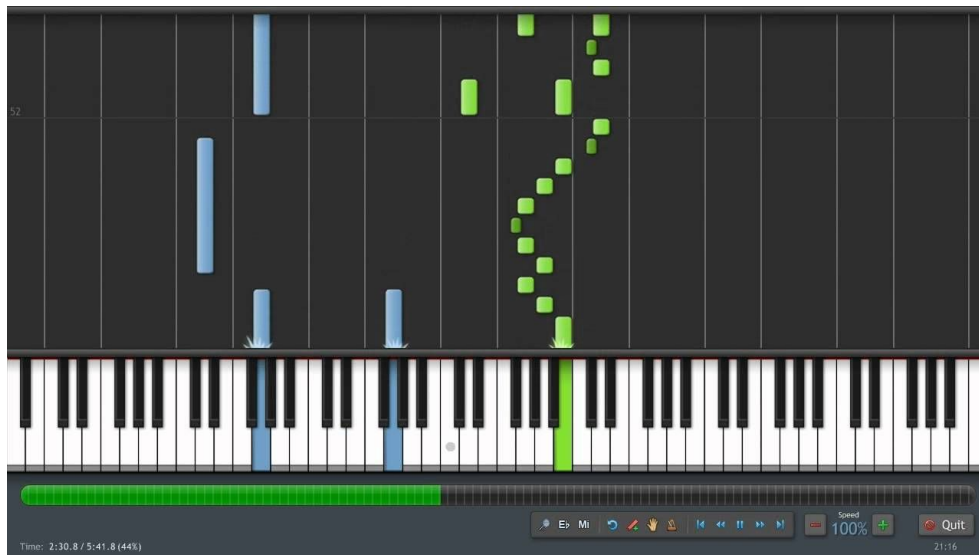

Figure 2.11: Synthesia

From a music visualization point of view, Synthesia's display form is very convenient as it is dynamic and is displayed in real time. At the same time, it is simple and straightforward, ideal for visualizing piano melodies.

Analysis

This chapter describes the analysis of all the solutions that were described earlier. The goal is to compare and analyze all the possible pros and cons of certain solutions. Be it in terms of generating or visualizing music. The purpose of this part of the work is to highlight all the successful solutions voiced in the research section that can be used when creating a design.

3.1   Musical constraints from music theory

Since one of the goals of our project is to generate harmonious melodies, this approach definitely makes sense. The rules of music theory are one of the most obvious ways to program harmony. The only problem is that it is necessary to find a middle ground between the purity of the generated composition and the number of possible generated compositions. Harmony theory determines exactly the best ways of interaction of various musical rules, because there are no rules that would be followed during all cases.

For example, one of the most common constraints in such situations is the choice of a single scale for an entire composition. However, if you look at, for example, classical music, many works are written in parallel scales that change over the course of a piece [17]. Therefore, there are no absolute rules.

Overuse of strict rules in the program will lead to the result that all the resulting melodies will be very similar to each other. The opposite situation, when we do not use any of the rules from musical theory, we may end up with an absolutely random set of dissonant sounds.

For this reason that it makes sense to set only the most basic and strict musical rules that are fulfilled in most cases, as well as the widest possible combination of using some other musical principles.

3.2 Using grammars to generate a melody

This is one of the most powerful ways to program melody generation rules. However, there are two different approaches to this method. One of them is to build rules based on existing works. The second is to create your own set of rules and choose how to switch between them.

3.2.1 Defining rules from already existing compositions.

This method ensures that rules are used that are already in unambiguous harmony. This is the strength of this approach. The user can simply choose a composition that is generally accepted. However, there are also disadvantages to this approach. First, a connection that sounded great in one context can sound very dissonant in another. Therefore, it is also necessary on top of this method to define the rules for combining these rules and the rest of the composition elements. Second, the problem of uniqueness. If you clearly define the euphony of a given context and contentious rules, the output will be exactly the same melody that was received at the beginning. This problem can be avoided by defining several melodies at the beginning, but then there is an obvious problem of dissonance between the rules taken from different compositions. Therefore, it is necessary to determine the correct combination of selection of rules and their application.

3.2.2 Hard coding the definition of a set of rules

This approach requires sufficient knowledge of the structure of music production. However, the approach of debugging the created rules in a practical way is not excluded.

According to Stephen Wolfram's work, the generation of such a structure using rule 30 will result in a complete structure for harmonious music. However, the same rule is used in the pseudo-random number generator, which makes one think that the harmony in the works obtained in this way is achieved rather due to the harmonization rules from music theory, which are applied after the main algorithm and which, accordingly, will give approximately the same result, even put an absolutely random set of sounds at the input.

However, the structure of the figure that is generated using the aforementioned rule 30 is somewhat similar to a fractal pattern. Which, in turn, suggests that the generation algorithm, called "Fractal Music" by junshern [6], may have a positive result in practice. However, the described algorithm is only way of applying rules, not creating them. Similar rules cannot be obtained from other compositions (due to the specifics of generation). Therefore, it is possible to simply create a random set of rules within our harmony rules (for example, using one scale) and this will give a similar result. The advan-

tages of this approach are obvious - the output will be an absolutely unique melody that already contains some harmony.

3.3 Using genetic algorithms

The genetic algorithm is unlikely to be used to generate music due to the difficulty of finding the necessary parameters that defines the algorithm. And also due to the fact that the result does not differ much in quality, even for correctly selected parameters [6, 18].

On the other hand, this algorithm has quite strong advantages. For example, the fact that it is possible to set a specific assessment of music and improve the algorithm in this direction provides space for creating the necessary conditions for serious development in quality. However, this strong point of the project is practically canceled out by the fact that such training and setting the necessary estimation algorithms will take a huge time before the output will have a decent result.

3.4 Fractal audio visualizer

The work "Fractal Audio Visualizer" [12] is a full-fledged program for visualizing a music using fractals. When working with standard audio files, this approach is undoubtedly optimal. The implementation, in which almost all of the above-mentioned fractal construction techniques were used, fully covers the purpose of creating the program. However, the use of Spotify API slightly limited the capabilities of the implemented visualizer: not being able to analyze different streams for each instrument separately, the author was forced to implement an algorithm that visualizes only the parameters extracted from the main stream of a musical work.

Also, a noticeable disadvantage of this implementation is that the analysis of music does not include the analysis of the subsequent behavior of the music. This would greatly mitigate the unpleasant experience of unexpected changes in the music data in real time. Harmony implies the correctness of the music and the pleasant experience due to the ability to predict it. And when harmony theory is poorly observed in a musical composition, this effect can be mitigated by warning the user about the next visual change.

The differences between the problems solved in the work of this author and the current one lies in the fact that in our case we do not need to convert the music file into parameters, since the music itself is also generated in our program and, therefore, we already have ready-made parameters for visualization. Also, in the author's work, the MIDI file format is often mentioned. This format is very convenient for analysis if an algorithm is chosen for implementation in our case, which analyzes already existing works.
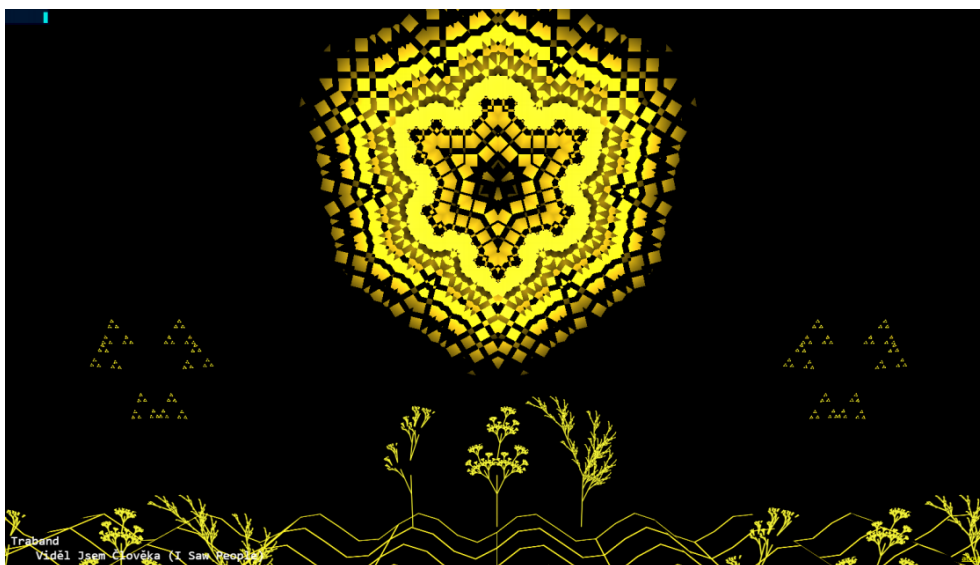
Figure 3.1: Screenshot from Fractal Audio Visualizer. Song "I saw people" [12].

3.5 Analysis of visualization methods

Before starting the analysis of solutions, it is necessary to describe the initial conditions for which methods will be selected. So, here is a list of limitations that cut off the ability to use most of the described visualization methods.

Figure 3.2: Piano playing visualization by Mr. Toms Mucenieks [19].

Music will be generated directly in our program. On the one hand, this is a big plus for visualization - there is no need to analyze the work, all the necessary parameters have already been determined and obtained. On the other hand, this greatly limits us in choosing a program for visualizing music - most programs are designed for manual composing and do not have the necessary API for full-fledged interaction.

This means that you will need to choose one of two options - create a visualizer using one of the mentioned tools, or find a program with a well-developed API so that you can focus on the music generation algorithm. Since the second option was not found during the research, this means that of all the visualizers described in the research, we need to use the most programmable tools.

In this case, the set of programs available for high-quality visualization narrows down to a list of development tools, in which we also need to find a suitable synthesizer that will play the generated music we need. The combination of AudioHelm + Unity is most suitable for these purposes. Unity allows you to create any type of visualization, since it does not limit the user in the programmable content. Helm, on the other hand, will provide real-time music synthesis, which is necessary in an environment where the composition is generated while the program is running.

The choice of the visualization option, however, still remains with us, since we have so far chosen only the development environment. The most informative from the point of view of the played notes is the MIDI format. For this reason, we need to choose one of the options for rendering MIDI files. In the research, one of such classic methods was mentioned - the Synthesia program. The interface of this program fully describes the notes that are played in a musical composition, and this will greatly help to visually analyze the generated music. Also, an undoubted advantage is that the user can predict the music that will be played next: in the program, the user sees notes that will sound in the next few seconds. This means that one of the main rules of harmony will be fulfilled in an unusual way - the listener will be able to predict the next sound.

Design

This chapter will describe the design of the upcoming implementation. This part is a logical continuation of the analysis and will refer to its results to justify the choice of implemented algorithms.

An equally important part is the definition of a clear framework for the functionality of the program, as well as the designation of the requirements for the program that must be met after the completion of the implementation. The requirements described here and the description of the functionality also determine the type of testing that will be carried out after the completion of the implementation.

4.1 Selected algorithm for music generation implementation

In this part of the work, the process of choosing an algorithm for composing music will be described. Since each of the above methods has pros and cons, the goal is to find the right mix of the pros for most implementations.

4.1.1 Major limitations from music theory

The basis of our implementation is the introduction of restrictions accepted in music. Due to the fact that the rules and recommended methods vary greatly in different styles of music, it is necessary to choose one musical style in the beginning.

When implementing the following steps, the algorithm should be implemented so as not to violate the basic rules and the transition grid defined in this way:

- At the beginning of the program, one scale is selected, in which the entire melody will then be generated.

- Used transitions in pitch / length of notes will be determined initially.

- Chords must match one of the commonly used standard chord progressions.

- The repetition grid should be initially rigidly defined. This can then be expanded to multiple valid grids.

4.1.2 A way to custom define rules for generation

The rules for the next two points can be rewritten by parsing already existing MIDI files. The analyzer will read all used MIDI files to determine some of rules. The program will define an initial list of rules for generating a melody. There will be two such sets of rules - for the pitch and for the duration of a particular note played.

The measure of implementing the algorithm for changing the generation rules for a specific loaded music will not be determined. The aim of this part is to implement the minimum possible adaptation of music composition depending on the loaded melody in order to expand and refine this influence in the future. Therefore, during implementation, a small mechanism will be developed to influence the final result by the user. The purpose of this is to demonstrate the functionality of loading MIDI files, as well as the operation of the analyzer in conjunction with the generator architecture.

Figure 4.1 shows the theoretical architecture of a software music composer. This scheme is the starting point for theoretical implementation, since it implements all the stages of generation in the most simplified form. This framework provides for a friendly implementation architecture that will meet the requirement for future extensibility.

As a result, this program will be guided by three main principles (in terms of music composition):

1. Create the necessary architecture to be able to program various **musical patterns** and levels of randomness in them, then set the basic structure.

2. Adapt the created structure for **parameterisation** using data from other songs.

3. An analyzer that will **extract** these **parameters** from a user-specified input.

The visualization module should be connected in a minimal way to the implemented generation algorithms.
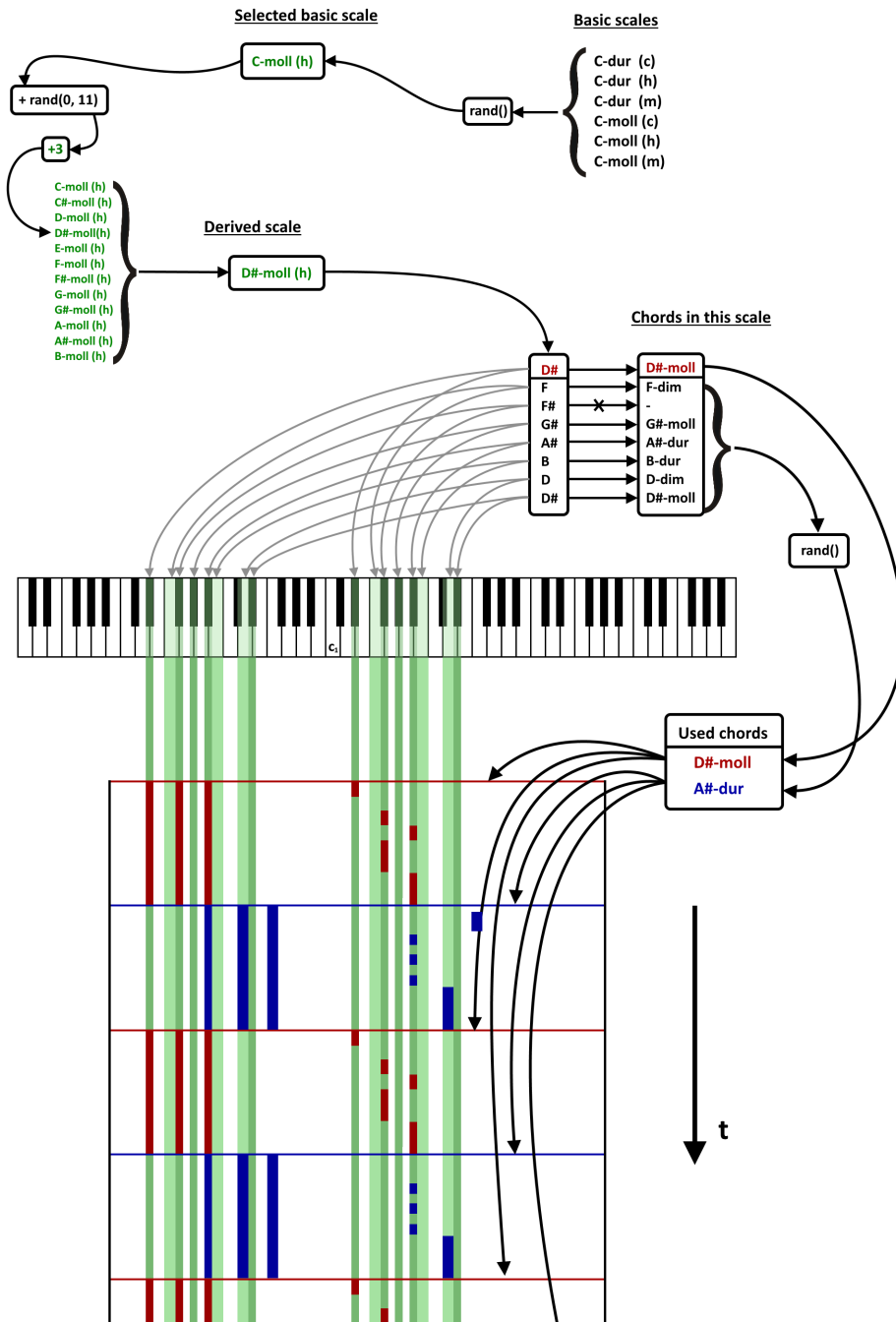
Figure 4.1: Abbreviated scheme of the general algorithm

4.2 Use cases and requirements

To begin with, it is worth describing in general terms the idea of the application that is to be implemented. This is a project that can be used to create music even without raw data - the user can simply generate music without having to rely on data. Another option assumes that the user will upload the MIDI file to the application and will be able to generate a new composition, which has adjusted the algorithms for creating music in favor of the file that was uploaded.

Now for the details. The application, first of all, should do the generation and rendering of music. But in order to demonstrate the full capabilities of the implemented algorithm, it is also necessary to implement a flexible scheme of interaction with the user, where he can control the process of playing the composition, and can observe the result in a form convenient for him. For this, the following list of requirements for the application functions from the user's side was formed.

- Ability to load a file, data from which will be extracted, loaded, processed and added to the current modifiers of the algorithm. The file can be uploaded regardless of the number of already uploaded files.

- Ability to reset music generation algorithm modifiers.

- Generation of a new melody. This function will delete the previously generated melody and overwrite it with a new one.

- Start playing the current melody.

- Pause. Stops playback at the current moment.

- Sound volume control.

- Adjusting the playback speed. It gives the user the ability to control the speed of the melody playing within a certain scale.

Several other important details and requirements are also worth mentioning. Despite its similarity to editors, where the user can change or create a playable melody, this application will not offer such functions. That is, the user will not be allowed to change the generated melody in order to be able to objectively evaluate the created composition, which was guaranteed to be created without the participation of third parties.

Also, the user will not be able to use the program as a tool for playing music - the piano keys will not interact with the user this way. They are only needed to display the music being played, not to be able to play something else.

From a performance point of view, problems can also arise - a large number of objects will be drawn on the scene at the same time. Therefore, it is necessary to formulate the following technical requirement: the application will not experience delays or freezes even when generating the maximum possible number of piano tiles and with the maximum music playback speed turned on.

4.3   Visual design of application

Visually, the application will resemble the Synthesia piano learning tool mentioned in the analysis. However, it will not allow the user to interfere with the process of creating a melody or extracting music. The visual part here serves only to simplify the evaluation of the generation result. The application will as shown in the picture Figure4.2
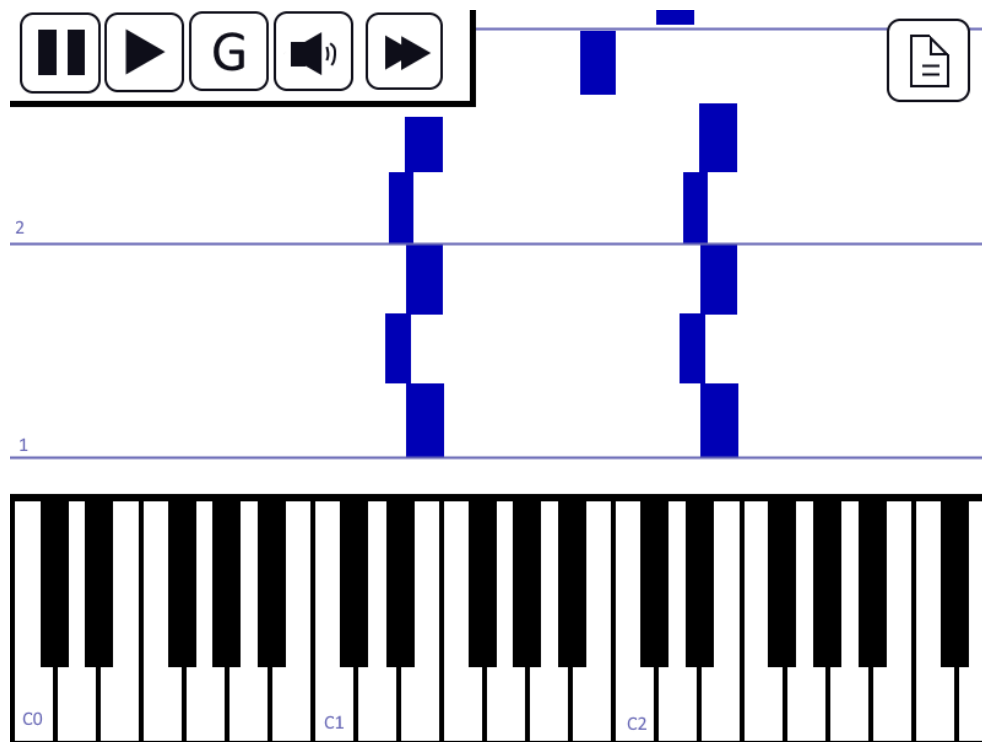

Figure 4.2: The first concept of the application view

It should be noted that this will be the only possible type of user interface.The photo shows some of the following interface details necessary for the full operation of the program:

- The **"Pause" button**, which pauses the playing of the current melody at this moment. Then it is possible to continue playing from the current moment.

- **"Play" button**. Continues or starts (in the case when the melody is just generated) playing the melody.

- **"Generate" button**. Creates a new composition and renders it immediately.

- **Sound control** button. Perhaps there will be a slider.

- **Speed adjustment** button.

- Button for **loading MIDI file**. After downloading, the file will be automatically analyzed.

- **Background panels**. These panels will display the sound scale and the progress of the melody playing. As music is played, they will move downward. On the left side, each panel will have its own serial number. When the melody ends, the panels will loop - the playback will start over without any interruptions.

- **Piano tiles**. They will move down with the panels. Show which notes are currently being played.

- **Piano keys**. They are present in the program so that the user can navigate which notes are being played at the moment. The note will sound the moment the piano tile touches the keyboard. And it will sound until it is completely hidden under the keys. While a note is being played, the corresponding key will also be highlighted. The octaves will also be numbered for better orientation.

Realisation

This chapter will describe the complete process of creating an application. Starting from the previous existing solution, and ending with testing the program according to the requirements described in the design part.

5.1 Development process

This part of the work will describe the entire development process. The versions that have been implemented, the sequence in which the necessary parts are included in the implementation, and the way in which this design is implemented in detail in terms of operating rules and development constraints.

5.1.1 Previous version

The idea of a music generator was implemented by us already last year within the framework of the "Software Team Project 1" project. Despite the fact that the implementation was very far from the current project and did not represent any value as a basis for the development of the current project, this application should still be mentioned.

So, the project was to sound the 3D game with music that will be generated in real time in the background. It was also a Unity project. There was no music synthesis there - it was possible to bypass this aspect by piling up the next system. 88 mp3s were loaded with the sound of the pre-recorded notes being sung by the choir (this is a ready-made asset in the Unity Store, called "ChorusPiano"). Then, the sound source, tied to the main camera, sounded one of the played notes in real time, depending on the generation algorithm. The main and secondary melodies consisted of one line of played notes, which were somehow ordered according to the selected scale.

The quality of the result was poor. Due to the meager implementation of the generation algorithm and the lack of almost all harmony implementation, the melody sounded very unmusical. In addition, due to the use of pre-
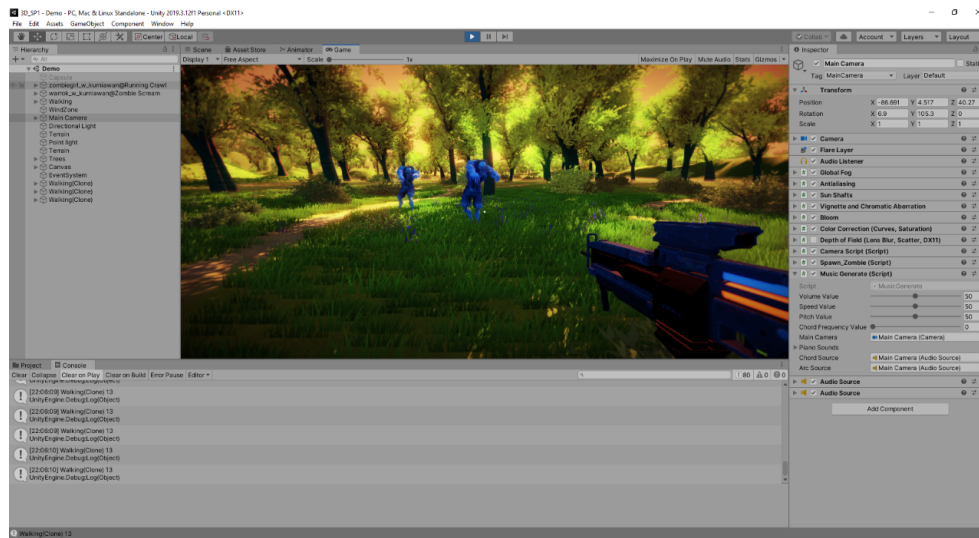
Figure 5.1: The interface of the "Zombies" project opened in Unity

recorded sounds, playing them in the shortened version sounded very jerky and unpleasant. Despite all this, an algorithm was founded that made it possible to start from something when writing this project. The project itself or parts of it were not used, however, some ideas helped to create the basis of the theory.

5.1.2 Visualizer development

It was necessary to start the development process with the development of the visualization algorithm - in this case, the subsequent implementation of the music generation algorithm could be tested right during the development process. The whole development process is the arrangement of a single scene in Unity (according to the design), as well as the programming of scripts in this scene.

The first big part of the implementation was the creation of objects, which were then treated as templates for copying. The color scheme was chosen the same as in the above program. For the working version, it was necessary to create the following objects:

- Tiles of different lengths for chord melody (2 key options).

- Tiles of different lengths for the main melody (2 key options).

- Tiles of different lengths for the second melody (2 key options).

- Key illumination for chord melody (4 key options).

- Key backlight for the main melody (4 key options).

- Key illumination for the second melody (4 key options).

- Template for background steve.

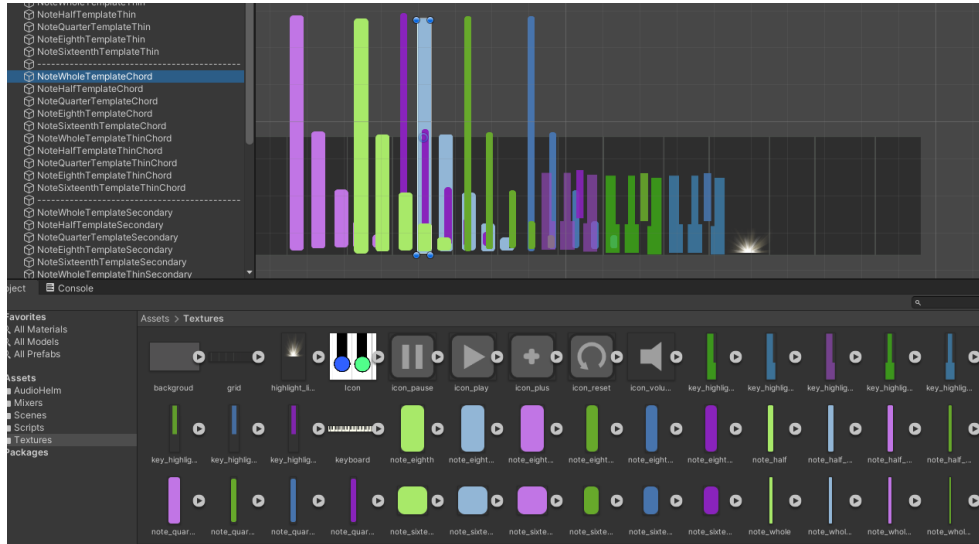- Light flash for top illumination of active keys.

Figure 5.2: List of basic templates for the scene

First, it was decided to implement one template for all note durations, and then change it to the required one in real time. However, when it became clear that the music generation algorithm would only work with a limited set of durations, it was decided to make a separate template for each duration. Perhaps this will change in the future when note values are determined during music generation. The same applies to the color of the notes - perhaps this will be parameterized in the future. Since the algorithm assumes only the use of notes of durations 1, 1/2, 1/4, 1/8 and 1/16, for each type of melody it was necessary to create 10 objects (5 for white and 5 for black keys). In total, this gives us 30 objects for the implementation of tiles.

The next catch comes at the moment when we need to create templates for highlighting each of the different types of keys - white keys have three different shapes. Therefore, we had to add an algorithm for determining the type of a key by its ordinal number (and this is on top of the already existing algorithm for obtaining the horizontal coordinate of a key by its number), which was implemented simply as determining the position of the key in an octave (remainder after division by 12) and the subsequent switch case.

Then a template was created for the background stave (it was decided not to make the sequence number of the stave part of the object, instead the number is a separate object that will move in parallel with the stave). And

39

also the light texture was downloaded and slightly changed from here [20]. Finally, the last object was created, which was a piano keyboard, which is a pre-drawn texture and several text objects that represent octaves.

After creating the objects, the priorities of rendering on the screen (layers) were allocated. The objects of key illumination have the highest priority, then comes the keyboard, then tiles, and only then the stave.

The principle of interaction of objects on the scene is quite simple:

- Staves together with tiles constantly descend downward at a constant speed.

- Periodically, steves with tiles that are far down are deleted and new ones are created in the pre-render zone.

- Synthesizer "Audiohelm" sounds all notes that touch the keyboard and does this only until the tile is completely hidden under the keyboard.

- Key illumination highlights those notes that are sounding at a given time.

- The illumination above the keys is only active while the illumination of the corresponding keys is active.
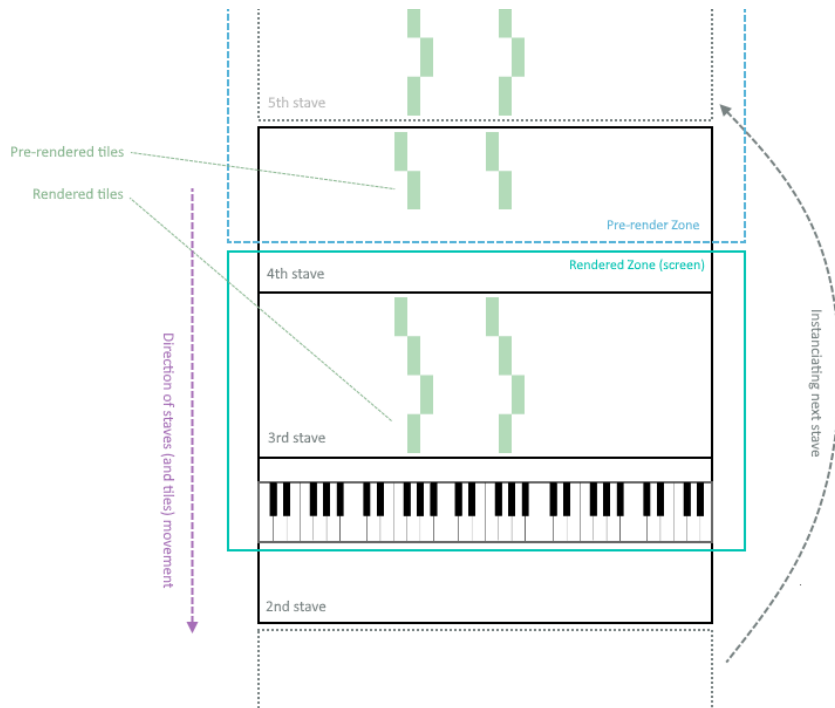

Figure 5.3: The principle of interaction of objects on the scene

The implementation includes a method that removes the stave and all keys that are in it (their lower part lies within the stave) at the moment when the stave reaches a certain critical vertical coordinate. At the same moment, another method will be triggered, which takes the next stave and all the keys with the timestamp of this stave from the queue and draws them in the pre-render zone.

The user can control this movement. The Pause button will simply cause the script to avoid the method that updates the position of objects and the ringtone timer. It was also decided to combine the pause and play keys for convenience: during pause, this button is the play button, during playback, it is the pause button.

An important detail is that this movement is looped: as soon as the algorithm reaches the last stave of the melody, it will start all over again and draw the first stave.
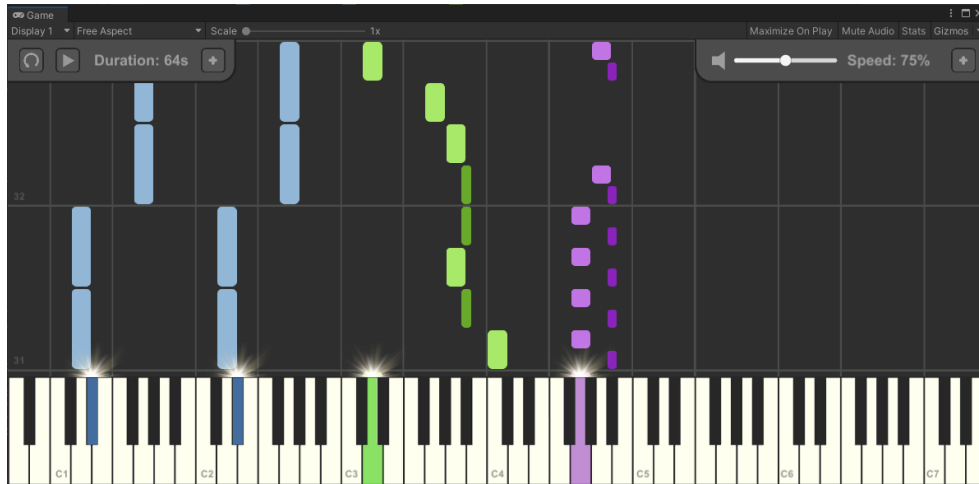

Figure 5.4: One of the latest working versions of the visualizer

The result corresponds (practically without differences) to the original design. Almost all functions are present that allow the user to influence the course of the program. It is also possible to tune the instrument on which the composition is generated. However, this can only be done programmatically by creating and using a new Helm file. In the future, you can add the ability to customize this aspect from the user's side.

5.1.3 Performance improvement

During testing, a performance issue was discovered. It consists in the fact that the application freezes for a split second in the case when the playback speed is 2x and there are many tiles in the scene. In this case, the moment of playback is a fraction of a second late and the sound quality drops dramatically.

After a further analysis, it was concluded that the instantiation function in scripts takes the most time during the operation of such a program. That is, the naive implementation, when we create a new object based on a template, is not suitable for the reason that if we need to add a new object to the pre-render zone (be it a tile or a stave), the slow function "Instantiate" is called. which cannot be used in such quantities.

As a solution to this problem, it was decided to create object pools. The essence of this algorithm is quite simple: at the beginning of the program, we create a large enough array of objects that will be inactive. And then, instead of instantiating new objects, we will get the ones already created from this array, simultaneously making them active again.
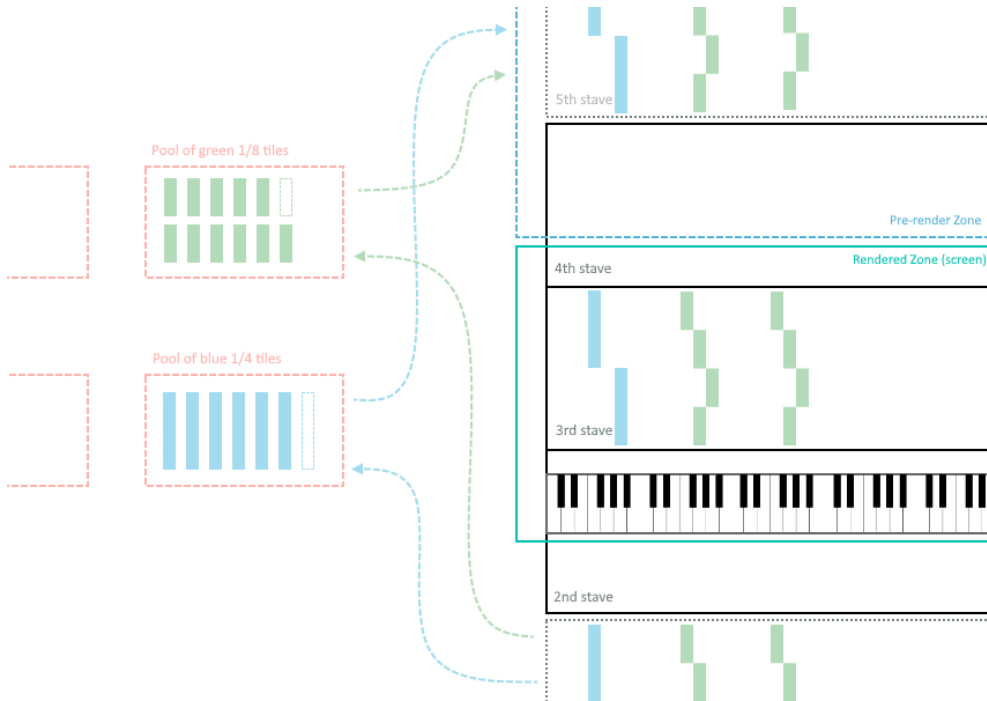

Figure 5.5: A theoretical diagram of how the pools work in our program.

Pools will need to be created for each object: that is, a certain number of objects of each type will be created at the beginning when loading the scene. This will greatly increase the loading time of the scene, but at the same time it will greatly speed up the program while it is running.

Another important aspect of pools is the fact that objects that are no longer used are simply returned to the queue of inactive objects.

Since we do not have a guaranteed maximum for the number of tiles on the screen, the pools were also made so that they can be expanded. That is, if the algorithm tries to pull out an object from the array, in which there are no more objects left, the array will simply expand and add a few more objects

to this pool. To avoid the impact of these extensions on the stability of the program, it is sufficient to increase the initial size of the pool or decrease the amount by which it increases with the expansion.

5.1.4 Adding a Generation Algorithms

The generation algorithm does not differ much from the model described in the design part. For convenience, the mentioned algorithm had to be divided into modules, each of which can be controlled autonomously.

The first part of the algorithm is a legacy of the previous version of our program. In this part, we simply select the scale in which our entire composition will be played, and the chord sequence, which will also be observed throughout the entire melody.

The choice of a scale is carried out in the following steps:

1. Select a random octave note from 1 to 12.

2. Choice of a random of 6 standard frets, which will be built from the selected note.

3. Then a structure is built, which is determined by the array of allowed notes based on the selected scale.
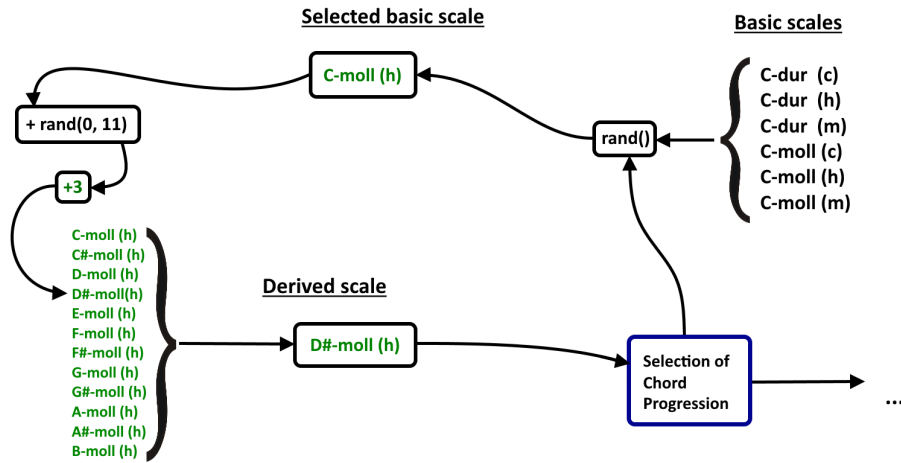

Figure 5.6: Theoretical scheme of the first part of the generation algorithm.

The **Select of Cord Progression** module does what determines the chord progression for a given composition. In our implementation, it simply selects a random sequence from the data. In case of several unsuccessful attempts to select a chord sequence, the module returns the program to the scale selection

43

stage in order to re-select it (since due to the limited number of chords, there are still scales for which none of the sequences are suitable). After the selected sequence is confirmed, the module transmits the output of the next program section.

In the diagram 5.7 you can see that each scale has a certain set of priority chords that can be played within it. Each scale contains a larger number of available chords, however, those that are created according to the specified algorithm have higher priority and SCP, in the presence of these two options, will give priority to the first option.
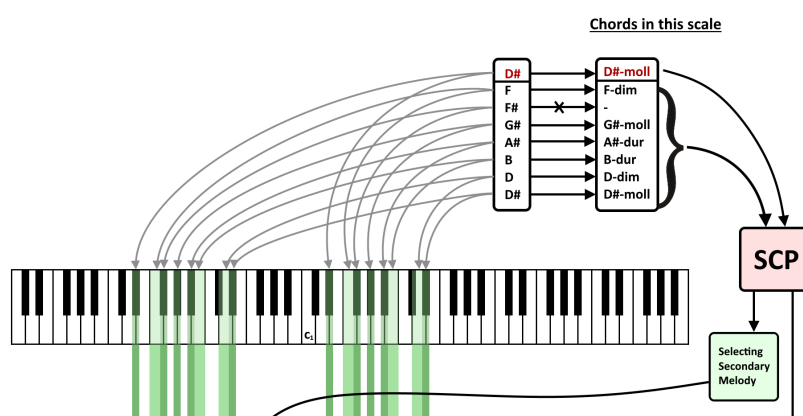

Figure 5.7: An intermediate part of the program with a secondary melody module.

Upon completion of the SCP module, the data will be transferred to the next two modules. The first of these, the Selecting Secondary Melody module, changes the chord sequential so as to form a secondary melody that will reflect the chord being played. This module simply selects one of the given secondary melody patterns in some way. Moreover, several different grids are selected at once - for different stages of the melody.

The Composition Structure Generator module is responsible for creating the structure of the entire musical composition. For the most part, he is engaged in choosing one of the given schemes for the arrangement of parts of the music and composing a composition according to this scheme.

Music Phrase Architect module creates one phrase of a musical composition. One phrase corresponds to several seconds of music or one (sometimes two) lines in a song. Each such module is responsible for building a small part of the song, so it has only a few parameters as input. For each phrase, start and end points are defined, as well as preferred intervals between sounds. Sometimes a phrase contains two sentences at once, and then two more intermediate control points are added, to which the melody should come. The CSG module handles these segments and arranges them in the correct sequence so
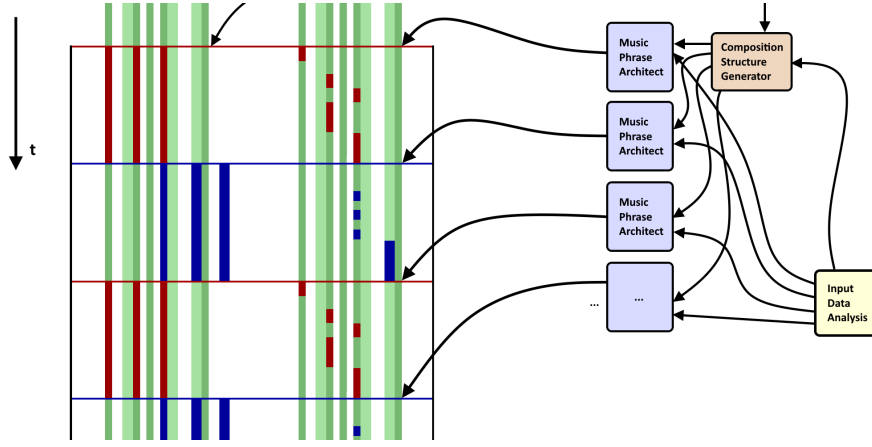
Figure 5.8: The main part of the generation algorithm. The structure and processing of the input.

that the final composition has the necessary structure.

The Input Data Analysis module processes the input. More details about the structure of this module will be discussed in the next part of the work. In a few words, this module parses the file that the user added. And then it passes the data to the aforementioned modules to generate.

5.1.5 Integrating file analysis into an existing algorithm

The presence of a tool for analyzing files is one of the requirements for the final program according to the design. Since the implementation of a serious analysis of a file requires a deeper approach and is worthy of a separate similar work, in our case we focused more on creating a surface analysis that will not affect the general structure of generation, but will affect the algorithm of operation of some finite modules, which will bring closer the type of sound for the specified song. This means that analyzing the file will not affect the selected scale or chord progression.

Instead, the uploaded file will affect:

- The intervals used by the MPA during transitions during the creation of phrases.

- Durations of the selected tiles in MPA modules.

The algorithm, however, will not have the structure of simply defining rules from all two adjacent bundles of notes. The data will be retrieved as follows.

The analysis algorithm is based on the construction of a number of generation rules in the form of a context-sensitive grammar. Figure 5.9 clearly
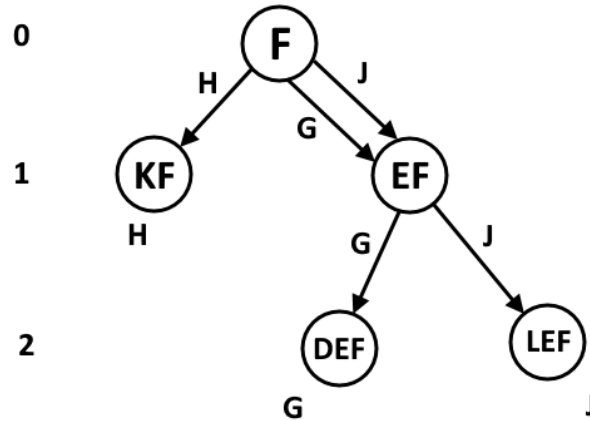
A-B-C-D-E-F-G-I-K-F-H-L-E-F-J

Figure 5.9: Analysis of MIDI files.

shows the principle. The user has uploaded some MIDI file. A sequence of notes in a melody was extracted from this file: ABCDEFGIKFHLEFJ. Some rules will be extracted from this data. For example, the picture shows the diagram for the note F. That is, the situation when the generation algorithm has just played the note F. According to our rules, this note can be followed by three other notes: H, J or G. However, if we look at the previous played note and it turns out to be E, then our algorithm reduces the number of notes that can be played next. Now it can be G or J. If we go to the end in this way (to an unambiguous answer), then sooner or later we will come to a song that is absolutely identical to the one that the user uploaded. Therefore, the algorithm stops at the moment when there are several options and selects one of them at random.

The only clear difference between this example and reality is that our algorithm does not analyze the notes themselves, but only the intervals between them. And during the last step, when the algorithm chooses a random interval from the possible ones, in our implementation it also takes into account the range in which we are at the moment.

5.2 Technologies used

During the implementation of this application, many programs, plugins and technologies have been used. Here are the main ones and the purpose of their application:

- **Unity 3d** is the main tool used to create this application. Was used to create the whole scene, describe the interactions. Also Unity Build was used to create the final exe file.

- **Visual Studio** was used to write scripts for the Unity project using **C# language**.

- The **UnityEngine** and **UnityEngine.UI** libraries were used to correctly describe the logic of object behavior in scripts in the Unity project.

- The **Unity Helm** plugin was used to add the necessary functionality for synthesizing music to Unity. The application of the same name was used to study the possibilities of composing an algorithm for this synthesis.

- The **MIDI.NET** library was used to provide a scriptable implementation of reading and analyzing files that the user has uploaded through the Unity Interface.

- **Unity Standalone File Browser** package was used to implement file browser window, that shows up when user clicks on "Upload MIDI" button.

- **Midi Parser in C#** has been used to parse uploaded MIDI file. That's not a unity package, that's C# script which is included to Unity project as other C# scripts.

5.3   Testing the application

Testing the application took place in two main stages.

First, during development, each described segment of code that is responsible for a specific functionality was tested using Unit tests. That is, any stand-alone function that has a specific input and output was included in the final testing using Unit tests, which are launched automatically when the program is launched in debug mode. This made it possible to immediately identify and eliminate errors that were randomly made during development.
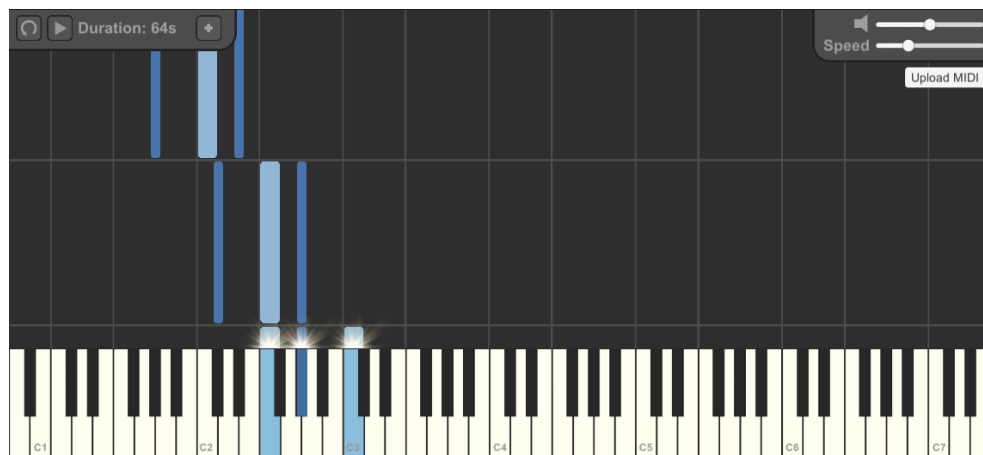

Figure 5.10: Program interface after fixing the flaws

The second stage was full user testing. Both personal feedback on the convenience of the program and criticism from the technical side were taken into account. Several people who were not previously familiar with the program participated in the testing. Testing was carried out from the moment the user installed the program using the link to the installer until the moment when the user thought that he had explored all the available functionality of the program.

During testing of the final product, several shortcomings were found. The user interface turned out to be much less user-friendly than the design required. The problem was primarily in controlling the speed of music playback. Firstly, users did not immediately realize that the button for changing the playback speed is tied to changing the speed: the label and the button are not located in the correct way. Further, it is not at all obvious that the speed of the melody playing will change only after the entire melody is updated (you must press the "Refresh" button). When commenting on this problem, both users agreed that the best solution would be to implement the speed change slider in a similar way, as the slider for changing the sound volume has already been implemented. The only difference is that the speed will be displayed aside from the slider. This recommendation was taken into account

and the interface of the top panel was changed, as well as the functionality of user interaction with a variable speed of playing a musical composition.

During the testing process, we also found a big flaw in the system for determining parameters based on the uploaded file. Due to the specifics of this algorithm, with certain parameters, the music almost exactly repeated the loaded melody or, with other settings, showed a completely random result. From this it was concluded that this algorithm requires more careful tuning. This algorithm has been temporarily replaced by an implementation of an algorithm that gets the distribution of chances to be played over note intervals. This data is directly obtained from the MIDI file and corresponds to the frequency of occurrence of the note A after the note B.

The second issue found was a performance issue. This problem occurred due to excessive use of the "Instantiate" function in scripts. To improve program performance, object creation has been moved to template pools. This solution is described in more detail in the corresponding section.

Conclusion

In the theoretical model, the optimal combination of the two methods was chosen. The first simply defined fixed rules and structure according to which the music was created. The second did not contain rules, but relied only on the analysis of already existing melodies. The first method often severely limits the number of possibilities. The second one often generates melodies that don't have the correct musical structure. Therefore, it was decided to combine these two methods.

In the analysis of tools for implementation, the option used in the basic version of this project was chosen. Unity 3D proved to be a universal platform with many useful plugins - for implementation, the AudioHelm plugin was eventually used, which synthesizes the music we generated.

The visualization method was chosen as classical as possible. There are many different solutions, which include non-standard approaches and interesting visual discoveries. However, an option was chosen that helps in analyzing the results of the generated music and maximally demonstrates the structure of the composition - the piano tiles.

During the implementation process, some questions and problems arose that were described above. As an architecture for visualization, a standard version with a moving background, which is at the same time a time scale, was chosen. The user can control the playing of a melody in every possible way, as well as load existing compositions in MIDI format for analysis and generation of a new melody, which is based on the loaded one.

The result is a tool for analysis existing compositions and generating new ones. The expected level of user experience has been achieved - the process of playing and generating music is fully regulated through a graphical interface. The result is also optimized for future improvement, since the program has a music generation and analysis architecture that assumes future extensions.

Bibliography

[1] Armaxis. Music generation in real time. 2013. Available from: `https://habr.com/en/post/185154/`

[2] Sadler. A little more automatic music generation. 2013. Available from: `https://habr.com/ru/post/185606/`

[3] Wolfram Research, Inc. WolframTones. 2005. Available from: `http://tones.wolfram.com/about/how-it-works`

[4] Wolfram, S. *A New Kind of Science.* Wolfram Media, first edition, 2002, ISBN 1-57955-008-8.

[5] SkywalkerY. Generating music based on a given style. 2009. Available from: `https://habr.com/ru/post/69985/`

[6] junshern. How do computers compose music? 2018. Available from: `https://junshern.github.io/algorithmic-music-tutorial/part1.html`

[7] Peter S. Langston. Six Techniques for Algorithmic Music Composition. 1988. Available from: `http://peterlangston.com/Papers/amc.pdf`

[8] Renderforest. Online Music Visualizer. 2013. Available from: `https://www.renderforest.com/music-visualisations`

[9] gheljenor. Sound theory. What you need to know about sound to work with it. 2015. Available from: `https://habr.com/ru/company/yandex/blog/270765/`

[10] Makeman. Fourier transform in action: precise signal frequency detection and note extraction. 2015. Available from: `https://habr.com/en/post/247385/`

[11] Desmos, Inc. Desmos Math Tools. 2021. Available from: `https://www.desmos.com/calculator?lang=en`

[12] Radka Hošková. Fraktální audio vizualizér. 2020. Available from: `https://alfresco.fit.cvut.cz/share/proxy/alfresco/api/node/content/workspace/SpacesStore/5c1a1d74-c485-4aa1-ac2f-400e339cbfa0`

[13] arakisoftware. What is AmazingMIDI? 1999. Available from: `http://www.pluto.dti.ne.jp/~araki/amazingmidi/`

[14] musicmachinery. Visualizing the Structure of Pop Music. 2012. Available from: `https://musicmachinery.com/2012/11/19/visualizing-the-structure-of-pop-music/`

[15] PreSonus Audio Electronics. PreSonus Notion 6. 2021. Available from: `https://www.presonus.com/products/notion`

[16] MuseScore BVBA. MuseScore. 2020. Available from: `https://musescore.org/`

[17] Richard. Can a song have random notes that don't belong to any major or minor scale? 2019. Available from: `https://music.stackexchange.com/questions/78293/do-all-songs-have-to-be-in-a-major-or-minor-scale-can-a-song-have-random-notes`

[18] Dragan MATIĆ. A GENETIC ALGORITHM FOR COMPOSING MUSIC. 2010. Available from: `http://elib.mi.sanu.ac.rs/files/journals/yjor/39/yujorn39p157-177.pdf`

[19] Toms Mucenieks. Pirates Of The Caribbean played on real Piano Tiles. 2019. Available from: `https://www.youtube.com/watch?v=G7eCzvd18Fw`

[20] FreePngPictures. Free PNG Pictures. 2021. Available from: `https://freepng.pictures/download/light/`

Acronyms

GUI Graphical user interface

MIDI Musical Instrument Digital Interface

Contents of enclosed CD

readme.txt .... the file with CD contents description
exe .... the directory with executables
src .... the directory of source codes
    wbdcm .... implementation sources
    thesis .... the directory of LaTeX source codes of the thesis
text .... the thesis text directory
    thesis.pdf .... the thesis text in PDF format
    thesis.ps .... the thesis text in PS format

57