Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Creative Robot Composer

Bachelor Thesis

Christian Holzreuter
cholzreu@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Manuel Eichelberger
Prof. Dr. Roger Wattenhofer

August 29, 2018

Abstract

This bachelor thesis is expanding on a library written by a previous student to be able to create new music with the press of a button.

The existing library was written in Python and uses a console interface and a .ini file to pass parameters. In this step a graphical user interface (GUI) for that library was written to make it easier to enter parameters and therefore more accessible. There was some restructuring done to the library as well to allow the use of said GUI.

Contents

Chapter 1

Introduction

Composing a piece of music is an art and not many are able to do it well. It requires a lot of theory and inspiration. Many programs to assist in the process of composing a new piece of music already exist but not a lot of them can do it on their own. The ones that do are mostly based on machine learning which means that they take a lot of human created music to learn how to make music.

The idea behind creating music based on music theory is to see if music written this way can be as good as songs created by human talent or machine learning.

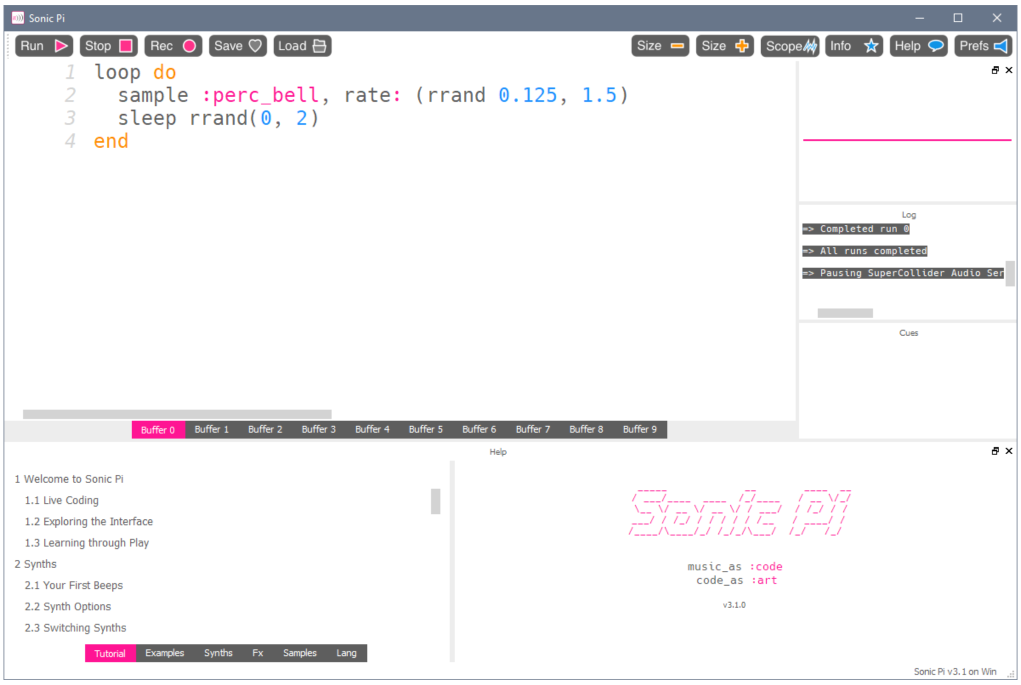
The goal of this thesis is making the act of creating a new song available to everyone by writing an intuitive GUI for the already existing library that can create theory based music.

1.1 Previous Work

The original work created a program that can compose music for everyone was done by Roland Schmid who wrote a paper called “Robot Composer” [1]. It is based on the open source tool Sonic Pi [2] which is a music programming tool. The interface of Sonic Pi looks a lot like an integrated development environment (IDE) as seen in Figure 1.1. Because of this, it makes it hard for people without programming knowledge to get into it.

Noah Studach later took that idea and created the Robot Composer Framework [3]. It is a Python library that uses the same prinicples as the original Robot Composer but was rewritten to function on multiple platforms and has a better interface to be used as a framework in future programs.

The framework takes a .ini file with input parameters as an argument, generates a midi file with them and then exits. Since editing a text file and then running a program with a file as a parameter in the console is not quite beginner friendly, we decided to expand on the framework by adding a graphical user interface and discarding the text based arguments. This not only allows



```
loop do
  sample :perc_bell, rate: (rrand 0.125, 1.5)
  sleep rrand(0, 2)
end
```

```
=> Completed run 0
=> All runs completed
=> Pausing SuperCollider Audio Ser
```

```
music_as :code
code_as :art
```

Figure 1.1: Screenshot of the Sonic Pi GUI

complete beginners to try the program, it also allows listening to the generated music before saving it and change parameters until the user is satisfied with the outcome.

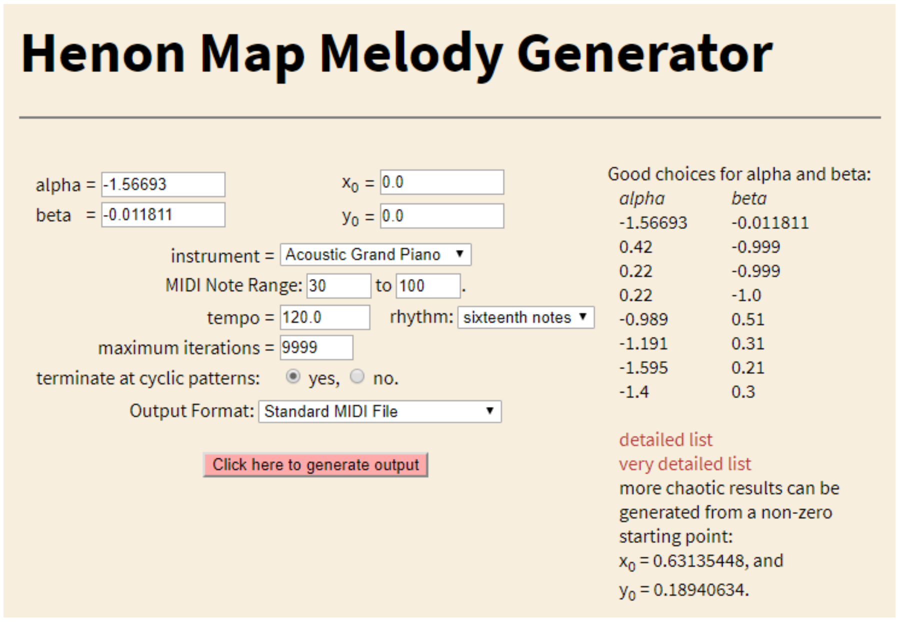
1.2 Related Work

There exist a lot of papers and scientific work about music creation, for example [4, 5, 6, 7, 8]. A lot of the papers are based on machine learning which makes it hard to compare them to our music theory based approach.

The above mentioned examples are all theory only which makes it impossible to compare to an actual software like ours. In contrast to the theory based papers, some music generation programs already exist as well, like the Henon Map Melody Generator [9] and Impro-Visor [10].

The Henon Map Melody Generator relies as the name implies on Henon Maps to create the music. Due to the chaotic behavior of Henon Maps, the music that gets generated sounds quite random. As seen in Figure 1.2, there are not many input options because the mathematical formula of the Henon Map does most of the work.

Impro-Visor is pretty much on the other end of the spectrum in terms of



Henon Map Melody Generator

alpha = -1.56693 x_0 = 0.0

beta = -0.011811 y_0 = 0.0

instrument = Acoustic Grand Piano

MIDI Note Range: 30 to 100.

tempo = 120.0 rhythm: sixteenth notes

maximum iterations = 9999

terminate at cyclic patterns: [✓ yes], no.

Output Format: Standard MIDI File

Click here to generate output

Good choices for alpha and beta:

alpha	*beta*
-1.56693	-0.011811
0.42	-0.999
0.22	-0.999
0.22	-1.0
-0.989	0.51
-1.191	0.31
-1.595	0.21
-1.4	0.3

detailed list
very detailed list
more chaotic results can be generated from a non-zero starting point:
x_0 = 0.63135448, and
y_0 = 0.18940634.

Figure 1.2: Screenshot of the Henon Map Melody Generator GUI

music generating software. The idea behind it is to help artists create music by adding percussion and deriving new solos from previous ones instead of creating new music from scratch. The Impro-Visor GUI (Figure 1.3) therefore allows the modification of every single note which even makes completely manual music composing possible.

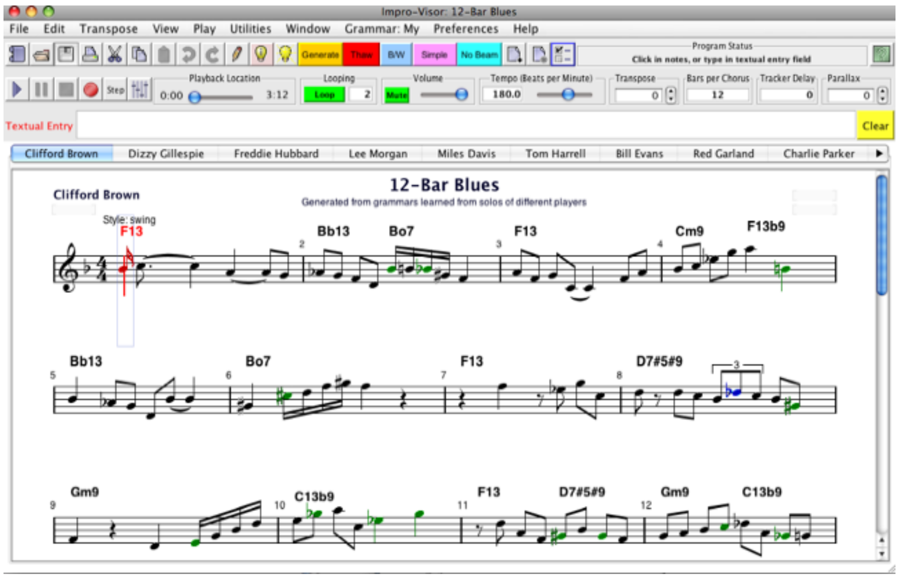

Impro-Visor: 12-Bar Blues
File Edit Transpose View Play Utilities Window Grammar: My Preferences Help
Generate Thaw B/W Simple No Beam
Program Status
Click in notes, or type in textual entry field
Playback Location 0:00 3:12
Looping Loop 2
Mute Volume
Tempo (Beats per Minute) 180.0
Transpose 0
Bars per Chorus 12
Tracker Delay 0
Parallax 0
Textual Entry
Clear
Clifford Brown Dizzy Gillespie Freddie Hubbard Lee Morgan Miles Davis Tom Harrell Bill Evans Red Garland Charlie Parker
12-Bar Blues
Generated from grammars learned from solos of different players
Clifford Brown
Style: swing
F13 Bb13 Bo7 F13 Cm9 F13b9
Bb13 Bo7 F13 D7#5#9
Gm9 C13b9 F13 D7#5#9 Gm9 C13b9

Figure 1.3: Screenshot of the Impro-Visor GUI

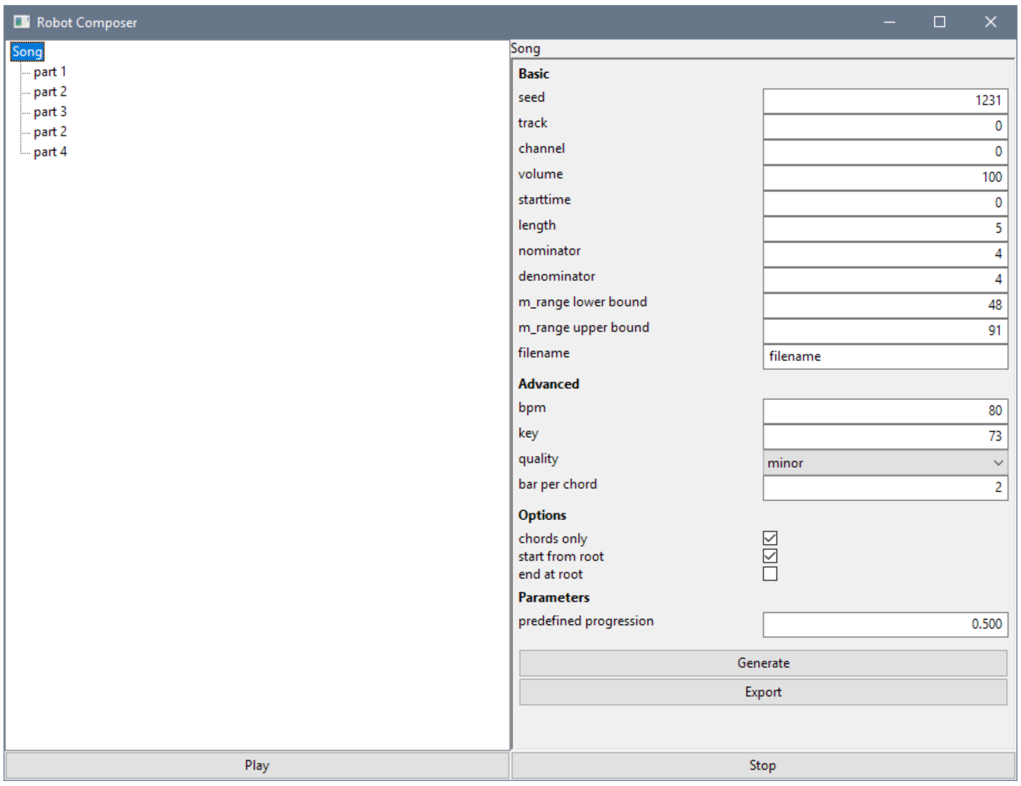
Chapter 2

User Interface

In the following chapters we show what changes were made to the Robot Composer Framework and how we designed and added the user interface.

2.1 Design

For the user interface a simple looking approach was taken as seen in Figure 2.1.



Song
- part 1
- part 2
- part 3
- part 2
- part 4

Song

Basic

seed	1231
track	0
channel	0
volume	100
starttime	0
length	5
nominator	4
denominator	4
m_range lower bound	48
m_range upper bound	91
filename	filename

Advanced

bpm	80
key	73
quality	minor
bar per chord	2

Options

chords only	✓
start from root	✓
end at root	☐

Parameters

predefined progression	0.500

Figure 2.1: Screenshot of the GUI

It has a list on the left side which allows the selection of any single part or the entire song. This is so that the user can listen to or change settings of a single part. Due to limitations of the Robot Composer framework, this is not currently supported as mentioned in Section 2.2.4.

On the right side all the parameters for the generator are listed and can be edited. If an invalid value is entered into a field it is changed to the nearest valid value as soon as the user changes the selection so it is ensured once the music gets generated all variables are accepted. There is also an on-change listener on the input fields so the user does not have to worry about manually saving them. As described in Section 2.2.2, the settings are imported from a JSON [11] file and can easily be extended to fit future need.

The bottom part has simple play and stop buttons so the user can listen to the generated song before exporting it to a Midi file.

The whole window can be resized and all the elements dynamically resize with it. If the list of the settings is too long for the current window size, the list becomes scrollable. The user can also freely move the middle line to make more space for the part list or the settings list.

2.2 Implementation

The implementation of the user interface for the Robot Composer framework currently relies on two libraries, required some restructuring of the framework and still has some ongoing issues which we could not resolve in the time we had.

2.2.1 Libraries

Other than the Robot Composer library and its dependencies we used a GUI creation library and one to play back the Midi files.

For playing the music in the program we used pygame [12]. It is open source and licensed under the GNU Lesser General Public License [13]. It supports almost every desktop operating system and can play back Midi files in a few lines of code. Pygame also has many more feature like creating windows with hardware acceleration but since its main purpose is to create games, it does not allow the easy creation of menus and mostly text based applications. It does allow further extensions which would add this function but we have decided to go for a more proven library for the GUI instead. This also allows us to replace it with a more lightweight Midi playback library or a more general music playback or editing library in the future, if need be.

For the user interface part, we first decided to go for the Kivy [14] framework. It is an open source MIT licensed [15] library to create applications for many operating systems with even multitouch and mobile support. We thought this would allow us to easily expand to iOS and Android in the future if we decided to do so. Due to the non-native look and some missing features like changing the background color of a button, we needed in our particular case we changed to another library after a while. This set us back quite a bit but in the end it hopefully was the right decision.

The library we settled on is wxPython [16]. It is a Python wrapper for the cross-platform GUI API wxWidgets [17], one of the best known GUI APIs out there. It is licensed under the wxWindows License [18] which is similar to the GNU Lesser General Public License [13]. WxPython has a native look and feel for the supported operating systems with a few small exceptions for some widgets. All the functions and features we needed are supported by this library which makes it a perfect fit for us.

2.2.2 Program Structure

The main function creates a new instance of MusicCreatorApp and initializes it. The MusicCreatorApp then creates a MainFrame which contains all the buttons and UI elements and an instance of Configs which contains all the values the library needs to create music.

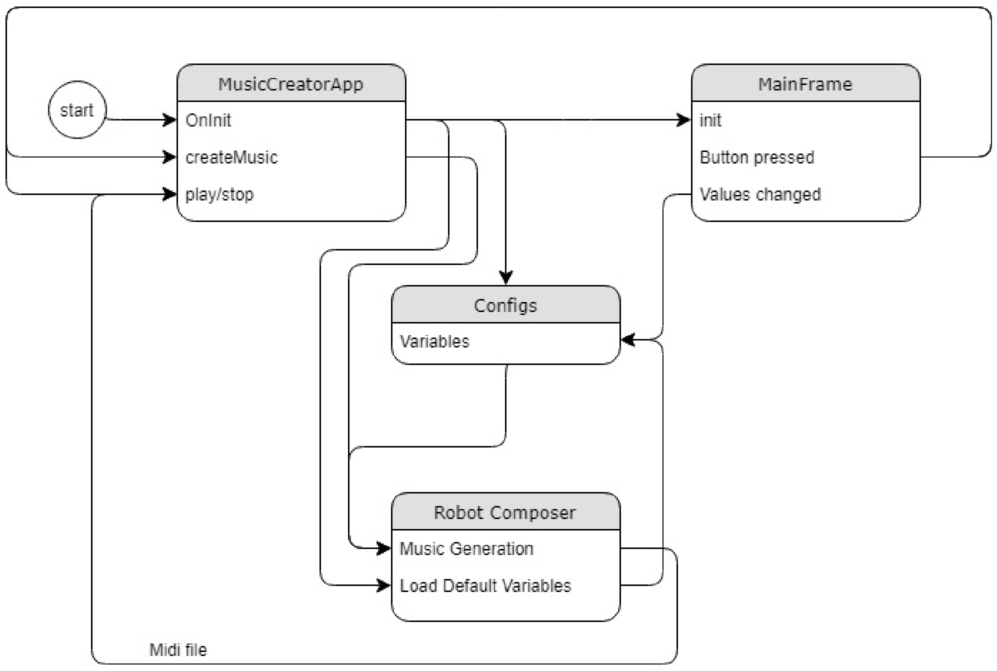


Figure 2.2: Program flow chart of the Robot Composer GUI

The Configs class gets instantiated with all the settings that are listed in the variables.json file. It contains a list with all the variables the library uses including the types and possible values for each variable. This file was created to make it easier for future variables to be shown in the GUI as settings with the right type and allowed input values.

When the GUI gets initialized, it uses the Configs class to look up all the variables it needs to show. When a setting in the GUI gets adjusted, the value in the Configs instance of the MusicCreatorApp will get changed. When the buttons to generate or export the file in the GUI get pressed, the MainFrame calls a function on the MusicCreatorApp which passes the current Configs to the Robot Composer Framework and calls the corresponding function in the library.

2.2.3 Library Restructuring

The Robot Composer Framework had some issues when it was first used with the GUI since it was designed to only run once and then close before generating the next song.

Because of this original design, the main function of the program to use the library contained everything to create a song. This code had to be split and moved into a setup function and a music generation function. The new main function only contains a call to the initialization functions for the user interface. Most of the code from the old main function is now only executed when the corresponding buttons in the user interface get pressed.

Another issue was that when the library was kept open in the background of the user interface it kept some variables between generating songs like for example the random seed. This meant that if the user did not close the window after every song it would not create the next song with all the new variables when they got changed by the user.

To change this, we created a new Configs class that is meant to function as a container for all the parameters used for the creation of the song. We changed the library to not use any internal variables any more and instead rely on this lookup class. When the user changes a parameter in the user interface it is changed in this settings container and when generating a new song the values in the container are used.

To make it easier to add new possible variables to the GUI and the library we decided to replace the .ini file that was used for the default values with a .json file that not only contains the values but also the names, types and possible values for every variable that can be used in the library as mentioned in Section 2.2.2. It gets parsed once in the initialization of the Configs class. The values are used as the default values in the Configs instance and the types and restrictions are used to create the settings entries in the GUI.

2.2.4 Ongoing limitations

Even though we were able to resolve most of the issues we had, we still could not quite get everything to function as we intended.

The original idea was to allow the user to change only parts of the song until he is satisfied with it and can then export it. Due to the way the library generates the song, this change is not as easy as we first thought. The parts that are implemented in the library function parallel to the instruments in the Midi track generation and do not contain those as seen in Figure 2.3. This makes it hard to change one without touching the others and therefore we were not able to get it to work in the time we had.

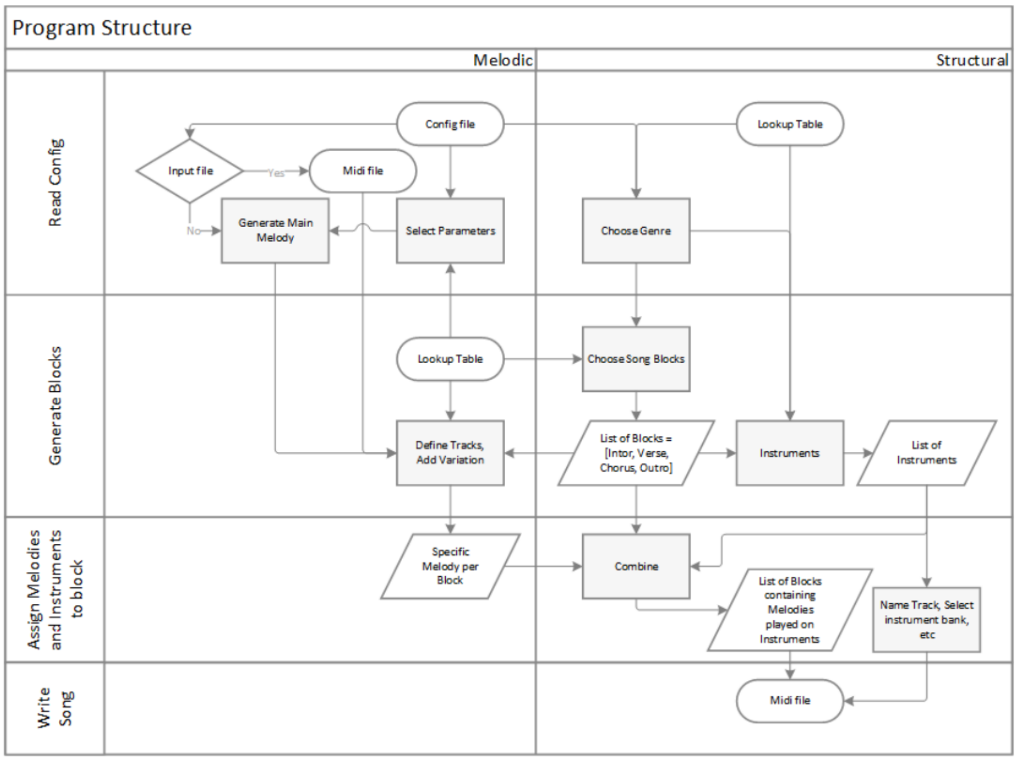


Figure 2.3: Program flow chart of the Robot Composer Framework (from [3])

The user interface already supports changing parts only but the library itself does not currently. This is one of the things that could be added in the future to expand on the functionality of the program.

Chapter 3

Conclusion and Future Work

The GUI that we implemented during this thesis successfully replaces the previous .ini file for the parameters. This makes it more intuitive to use for beginners so everyone can generate songs without having to work with a console and text files. It also allows the playback of the new song before saving it to the disk so the user can adjust the parameters and generate new songs until the music is satisfying.

3.1 Future Work

The library and GUI as it is now works on its own but a lot can be done to improve it. Due to how the library is structured it is not currently possible to adjust parts without changing the whole song. This could be added by reworking parts of the Python library. The GUI already implements the possibility to list and select parts to support the library.

Another quite obrious thing would be to implement more music styles. Since the whole program does not make use of machine learning, it would be necessary to implement the music theory behind other music styles which requires quite a bit of work.

Bibliography

[1] Schmid, R.: Robot Composer.
https://pub.tik.ee.ethz.ch/students/2016-HS/SA-2016-63.pdf
(2017)

[2] Website: Sonic Pi.
https://sonic-pi.net/
Accessed: 2018-08-14.

[3] Studach, N.: Robot Composer Framework.
https://pub.tik.ee.ethz.ch/students/2017-HS/SA-2017-95.pdf
(2018)

[4] Kathiresan, T.: Automatic Melody Generation (06 2015)

[5] Povel, D.J.: Melody Generator: A Device for Algorithmic Music Construction. Journal of Software Engineering and Applications Vol. 3 No. 7 (2010)

[6] Rinchiera, S., Nagler, D., Davison, J., Karunaratne, C.: Youtube: Intelligent Jazz Improvisation Generation using Markov Chains.
https://www.youtube.com/watch?v=zploY043Gx8
Accessed: 2018-08-14.

[7] Youtube: Computer-Generated Jazz Improvisation.
https://www.youtube.com/watch?v=Cbb08ifTzUk
Accessed: 2018-08-14.

[8] Temperley, D., Sleator, D.: Melisma Stochastic Melody Generator.
http://www.link.cs.cmu.edu/melody-generator/
Accessed: 2018-08-14.

[9] Website: Henon Map Melody Generator.
http://henon.sapp.org/
Accessed: 2018-08-14.

[10] Keller, R.: Impro-Visor.
https://www.cs.hmc.edu/~keller/jazz/improvisor/
Accessed: 2018-08-14.

[11] Website: JSON Standard.
https://www.json.org/
Accessed: 2018-08-22.

[12] Website: pygame.
https://www.pygame.org/
Accessed: 2018-08-14.

[13] Website: GNU Lesser General Public License.
https://www.gnu.org/licenses/lgpl-3.0.en.html
Accessed: 2018-08-14.

[14] Website: Kivy.
https://kivy.org/
Accessed: 2018-08-14.

[15] Website: The MIT License.
https://opensource.org/licenses/MIT
Accessed: 2018-08-14.

[16] Website: wxPython.
https://wxpython.org/
Accessed: 2018-08-14.

[17] Website: wxWidgets.
https://www.wxwidgets.org/
Accessed: 2018-08-14.

[18] Website: wxWindows Library Licence.
https://www.wxwidgets.org/about/licence/
Accessed: 2018-08-14.