Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing

Robot Composer

Project Report

Roland Schmid

roschmi@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Manuel Eichelberger, Michael König
Prof. Dr. Roger Wattenhofer

March 22, 2017

Abstract

Automation is an omnipresent topic nowadays; hence, we ask why music composition has not been substituted yet. Therefore, we developed a tool, the Robot Composer, that automatically generates jazz/blues music (or serves as an automated backing group for play along) only based on music theory instead of on machine learning. In this report, we present our selection of applied musical concepts, describe the challenges during the implementation and inform the decisions regarding the music generation process. Finally, we conclude that the Robot Composer is capable of producing melodious pieces and, consequently, that music theory seems to be a suitable foundation to build music generation software.

Contents

Chapter 1

Introduction

Many tasks in our digital world have been automated and were assigned to machines, but composers have not been replaced so far. What makes music composition a hard task for computers? — In order to tackle this question we have implemented the Robot Composer, a tool that automatically generates jazz/blues music. Throughout this practically oriented case study, we intended to use only the enormous stock of knowledge from music theory and no tools from machine learning (ML).

One can easily find similar tools for music/melody generation (like [1, 2, 3, 4, 5, 6, 7]) but while, on the one hand, it seems to be rather easy to generate music based on ML techniques and there are lots of well-functioning student projects on the Internet, on the other hand, most of the non-ML tools are rather sophisticated and have been developed over an extensive period of time. This renders their comparison to the many ML student projects unfair.

With lots of existing comparable and efficient tools based on ML, we thus aim to do the opposite: we impose restrictions on using any form of ML and try to accomplish best possible music generation for the selected genre while the comparison against ML techniques is not part of this project. Subordinately, the Robot Composer is meant to evaluate the open-source tool Sonic Pi and its capabilities regarding programmatic music generation in a close-to-professional context.

Concerning the structure of this report: we will first introduce the underlying music theory that was applied throughout this project, subsequently present the main challenges and technical impediments during the implementation of the Robot Composer, discuss the informed decisions regarding the music generation process and point out the weaknesses and features of this project's implementation. Finally, we draw the conclusions to this attempt and try to briefly put the results into context.

Chapter 2

Background from music theory

With the choice of jazz/blues music, we both gained a wide range of existing basic knowledge to start with, combined with a very liberate interpretation of the question *"What is jazz?"*, but also have to contend within one of the most creatively expressive styles of music out there. We therefore had to pick and focus our work on certain well-known concepts:

2.1 Rhythm and percussion selection

According to several online tutorials [8, 9, 10, 11, 12], jazz drumming follows along very similar lines in general. This led us to the conclusion that, instead of reinventing the wheel at this point, we would merely implement a finite selection of drum patterns with ongoing rhythmic and instrumental deviations to simulate the spontaneous variations a human drummer would apply in order to keep up the dynamics of a piece.

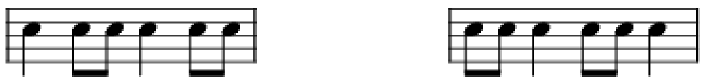


Figure 2.1: The music notation of the (inverse) swing rhythms

The general idea drawn from the aforementioned online tutorials is a simple (potentially inverse) swing rhythm on the ride cymbal to drive the piece, accompanied by suited *comping* patterns (from accompanying or complement [13]) using other percussive elements like the base and the snare drum. More specifically, the ride cymbal swing rhythm denotes a repeated pattern which comprises of one quarter note and two eighth notes (see Figure 2.1) where *swing* denotes an unequal partition of the ground beat (quarter notes) into roughly 72% of its duration to the first and 28% of its length to the second eighth note, while the notation usually remains unchanged [14]. The inverse swing rhythm denotes the shift of this pattern by one beat, i.e., commencing with the two eighth notes. In addition, we have decided to add some combinations of or completely without the ride cymbal rhythms in order to avoid monotonicity. The jazz comping serves to accentuate the swing-offbeats, that is, the second eighth note in a swing partitioning of the ground beat, and therefore helps to set the rhythmic frame of the piece.

2.2 The jazzblues patterns

Building on top of the selected drum pattern as its rhythmic frame, a piece of music needs an underlying harmonic structure as its melodic frame. This will be the main guideline for an improvising soloist and allow us to organize all further melodic voices into an euphonious harmonic composition. In the blues genre, the 12-bar blues schemes are such well-known underlying structures, widely used by artists all over the world. As jazz standards and jam sessions were emerging, the players reused and refined these blues schemes as a basic harmonic structure to accompany their improvisations by adding in more and more typical jazz cadences, e.g. II-V-I [15]. Some (very famous) exemplary results of this process can be seen in Figure 2.2; these are precisely the 12-bar patterns that will be used by the Robot Composer.

‖: F7	Bb7	F7	Cm7 F7
Bb7	Bdim	F7	Am7 D7
Gm7	C7	F7 D7	Gm7 C7alt :‖

(a) standard jazzblues

‖: Cm7	Fm7	Cm7	Gm7b5 C7alt
Fm7	Fm7	Cm7	Cm7
Ab7b5	G7alt	Cm7	Dm7b5 G7alt :‖

(b) minor jazzblues

‖: Bb	Eb7	Bb	Bb7
Eb7	Eb7 E^{o}	Bb	G7
Cm7	F7	Bb Eb7	Bb F7 :‖

(c) Dixieland blues

Figure 2.2: Popular jazzblues patterns according to [16], based on [15]; vertical lines denote separations of bars, two entries within one bar denote a changing harmony after the first 2 beats

2.3 The minor blues scale

In addition to the selected (or generated) harmonic structure of the composition, the Robot Composer should be able to produce a melodic leading voice fitting into the given melodic frame. This corresponds closely to the decisions made by an improvising soloist; hence, we found it natural to make use of the tools that inexperienced players are taught in order to facilitate this task. For improvisations to an accompanying blues scheme, a simple but already powerful tool is to restrict one's note selection to the blues scale (see Figure A.1).

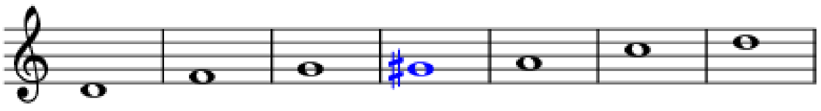


Figure 2.3: The minor blues scale in `d`, the **blue note** is marked by its color

In practice, with only few exceptions, this restriction on the note selection ensures that the soloist is not conflicting with the accompanying rhythm group. In this context, conflicting means to play dissonant notes that clearly do not fit with the other voices. The major issues may arise when the so-called *blue note* is played and not resolved immediately, that is, it builds up some tension in the listeners' minds through dissonance (see "tritone" [17] for details). Experienced soloists use this tension to attract the listeners' attention and transform the dissonant state into a harmonic sequence by moving on to the fifth. When picking the notes from the blues scale at random, however, the blue note may be followed by another dissonant interval like a diminished or augmented quart (e.g. `g#` $\rightarrow$ `d` or `g#` $\rightarrow$ `c`).

Clearly, a melodious piece of music is not guaranteed through avoidance of conflicts between the soloist and the rhythm group alone; an appealing composition most certainly needs to avoid constantly performing arbitrarily large jumps between successive notes and is required to exhibit some sense of melody and rhythm. More on this issue can be found in Section 3.2.3.

Chapter 3

Implementation

3.1 Technical framework

After deciding to implement the Robot Composer based upon the open-source project Sonic Pi, a potentially easy-to-use tool for programmatic music generation, we had to overcome lots of technical problems throughout this project. These include a proper installation and setup of a suitable development environment, as well as some technical limitations regarding the playback of the composed piece.

3.1.1 Installation and usage of Sonic Pi

First of all, the recommended installation procedure using the pre-built package of Sonic Pi did not work smoothly. While all the dependencies and the tool itself were installed successfully, it was not possible to launch Sonic Pi straight away. JACK and PulseAudio, the two prevailing sound servers, cannot be run in parallel. Hence, any user-friendly desktop environment using PulseAudio will cause a conflict here as Sonic Pi requires the use of the JACK audio server. During the following research, we came across two possible solutions to this problem: either combine JACK with PulseAudio via the `pulseaudio-module-jack` or using `pasuspender`. Either works fine, but the `pasuspender` method seems to be more stable and preserve the better overall desktop experience. More on this method can be found in the provided launch script: `./bin/sonic-pi.sh`.

Later, we discovered that it would be beneficial to make use of the new functionality of Sonic Pi introduced in v2.11, like the addition of the minor blues scale. When following the guideline for the Linux installation from source, no further problems arose and the installation manual even describes the `pasuspender` method mentioned above. However, the decision to use the newer version of Sonic Pi caused problems on one of the originally targeted deployment systems, the Raspberry Pi: the corresponding installation guidelines resulted in an infinite loop producing the same error output over and over. As these issues could not be resolved easily, it seems that using the Robot Composer on a Raspberry Pi requires the patience to wait for the pre-built package of Sonic Pi v2.11.

Once the tool is successfully installed and running, it is easy to make the first steps in Sonic Pi by following its included tutorial. Beyond that, there are two less accessible facilities: there is an included API-documentation-style reference of internal functions in the bottom left corner of the Sonic Pi interface in the register tab "Lang" and an even more elaborate API documentation that can be found online [18].

Finally, on a Raspberry Pi with rather limited resources, it may be desirable to run the Sonic Pi server in headless mode, that is, with no `Xserver` running. In principle, this is possible by manually executing the ruby script:

```
../sonic-pi/app/server/bin/sonic-pi-server.rb
```

and suppressing all error output that the server cannot connect to the GUI. See `./bin/raspberry-pi.sh` for details; please note that this requires package `x11-dbus` to be installed.

3.1.2 Setup a development environment

In its original form, Sonic Pi does not allow to develop in a different IDE other than the very simple provided interface intended to be suitable for beginners in programming. This, however, is barely usable for professional development. We therefore suggest to install the `ruby` gem `sonic-pi-cli` in order to send code fragments to the `sonic-pi-server` directly via command line. For those who favor the text editor `vim`, there is an included suggested setup documented in the `README.md`. Notwithstanding the above, one may use:

```
./bin/send_to_sonic-pi.sh
```

to load all required functionality from multiple files and send it to Sonic Pi accordingly. It provides the following directives:

- `#set: CODE`
  Immediately send `CODE` to Sonic Pi for evaluation; may be used to set global constants like:

  ```
  #set: PROJECT_DIR = "/home/pi/robot/"
  ```

  and will be evaluated just like any other `ruby` or Sonic Pi code.

- `#include: FILE`
  Directly include the contents of `FILE` at the position of the directive and interpret all directives found in the `FILE` accordingly.
- `#require: FILE`
  Recursively call `./bin/send_to_sonic-pi.sh` with `FILE` as an argument. This allows sending a file as a separate request to the `sonic-pi-server`.

This allows using a proper file structure instead of a single file that must contain the complete program.

Aside from the improved readability and maintainability, the last step is indeed mandatory due to Sonic Pi's server implementation: in some cases it only allows a limited number of lines of code to be processed at the same time. In our case we could determine a limit of 528 lines per separate request on a personal computer. This is a known bug [19] and apparently hard to resolve. However, this problem can be worked around by sending function definitions as separate requests as these will reside in a global scope within Sonic Pi. Recall that this can be accomplished using the "`#require: FILE`" directive described above.

3.1.3 Finding suitable instrument samples

Ultimately, the Robot Composer should be able to output the composition as audio. Therefore, the typical jazz/blues instruments need to be made available to Sonic Pi in the form of external samples. As a first step, all samples must be converted into `.wav`, `.aac` or `.flac`.

When handling samples various problems may arise:

- the samples might contain variable-length silence in the beginning that interfere with the precise timing,
- the samples might not be harmonically compatible to each other,
- a note's length is restricted to the sample's duration,
- the sample pack might not contain separate samples for every pitch and thus require the rate to be adjusted,
- this might even restrict the maximal duration of a note further,
- samples at manipulated pitch do not exhibit all the natural sounds of the instruments as usual.

Unfortunately, this is only the list of problems that occurred during the implementation of the Robot Composer – there are potentially many more. In contrary, in the end we were fortunate enough to find a set of samples with clear audio quality, long enough duration, very precise tunings and sufficiently small, equal-length silence prepending the sound. Consequently, we only had to deal with the adjustments of the pitch and tolerate the losses to the originality of the instruments that come along with this process.

The pitch of a note can be manipulated by playing it at a modified rate: the new frequency is simply calculated from the original frequency by multiplying it with the rate. Therefore, a rate of $r = 0.5$ decreases the note by an octave and a rate of $r = 2$ increases it by an octave. As an octave consists of 12 logarithmically equidistant semitones, the rate to increase a pitch by k semitones is thus given by the formula [20]:

$$r(k) = \sqrt[12]{2}^{k},$$

more easily given by the frequency quotient of two adjacent notes `a`^{1}, `a#`^{1} [20]:

$$r(k) = \left(\frac{466.164 \text{ Hz}}{440.000 \text{ Hz}} \right)^{k}.$$

The latter of these formulas makes it evident that it also holds for negative values of k in order to decrease the pitch as it simply inverts the quotient within the parentheses.

3.2 Musical concepts

In this part, we will present how the musical concepts from Chapter 2 have been applied during the implementation of the Robot Composer – not by discussing code fragments here, but on a more abstract layer.

3.2.1 Rhythm group generation: play-along mode

First, as indicated in the beginning of Section 2.1, the Robot Composer chooses one of the swing rhythms and an according comping pattern at random from a set of given patterns. These patterns are subsequently tweaked by adding in, exchanging, or leaving out percussive elements depending on a probability distribution to achieve the desired simulation of spontaneous variations.

In this section, we consider the Robot Composer in play-along mode as an automated backing group for improvisation. The Robot Composer selects one of the jazzblues patterns from Section 2.2 and generates a rhythm. Again, there are two cases: either it chooses to play a straight blues using equal-length quarter notes, or it uses the set of predefined rhythm patterns from Figure 3.1 with a high preference on the standard swing beat (four eighth notes) to ensure

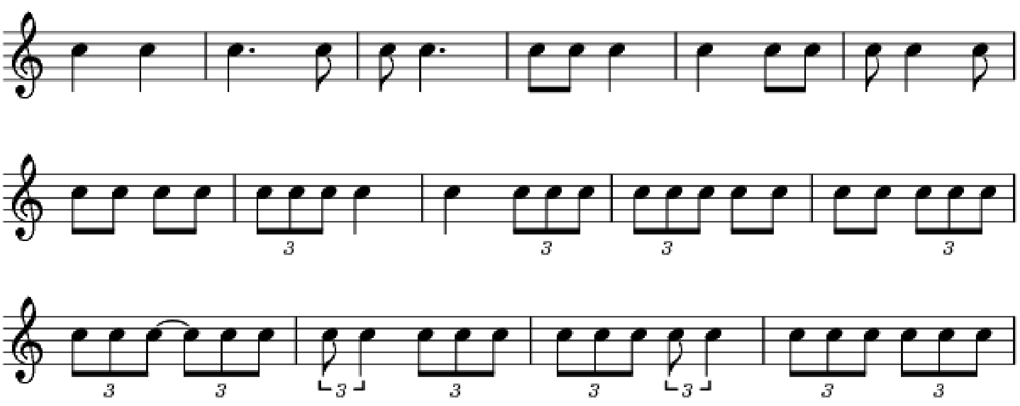


Figure 3.1: All predefined rhythm patterns on two beats

a resulting swing rhythm. Otherwise, it picks one of the patterns uniformly at random.

Rhythm is mainly determined by two elements: notes and rests. Hence, the next step is to assign breaks to the selected rhythm pattern by replacing units of the chosen rhythm pattern. We do so independently for each note with probability either 25% or 40%, depending on the selected rhythm pattern. If the standard swing beat is chosen, the probability to replace a note with a rest is set to be as high as 40%. However, in order to emphasize the rhythmic variation in the output for rhythm patterns chosen at random, notes are only replaced at probability 25% in this case. This latter case is also illustrated in Figure 3.2.

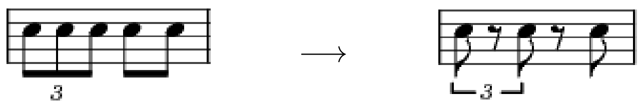


Figure 3.2: An exemplary assignment of rests to the selected rhythm pattern (left) that occurs with a probability of $p = 0.75 \cdot 0.25 \cdot 0.75 \cdot 0.25 \cdot 0.75$.

Ultimately, playing the picked jazzblues is very simple as Sonic Pi knows the structure of musical chords. For harmonic stability, we keep the base note at one instrument while interchanging the other instruments at random so that each instrument plays a different melody with the same rhythm using the notes of the jazzblues chord. Finally, after every iteration of the 12-bar blues pattern, up to two instruments from the rhythm section are exchanged at random in order to alter the musical setting slightly over time.

3.2.2 Rhythm group generation: solo mode

In solo mode, the drum pattern is selected equivalently to the play-along mode as described in the beginning of Section 3.2.1. With lots of similarities to the previous section, we will explain the following ideas for the solo mode in comparison to the play-along mode.

Instead of choosing among the predefined jazzblues patterns, however, in solo mode the Robot Composer generates its own blues harmonics. In play-along mode this was not done because, unlike a computer generated soloist, a human player cannot instantly react to changing harmonics. Therefore, the Robot Composer uses a so-called hidden Markov model [21], that is, a Markov chain whose current state is not known at all times (hidden). In practice, this corresponds to the application of a stochastic transition matrix containing probabilities. As we do not intend to apply any tools concerning ML, we build up such a static Markov model from a (changing) selection of the three jazzblues patterns seen in Figure 2.2. We thereby encode the typical jazz cadences found in these 12-bar blues patterns into our decision matrix. Subsequently, the next chord is chosen based on the previous two chords in correlation to the frequency of appearance of this chord sequence in the jazzblues patterns. The only manual adaption here was to limit the probability of replaying the same chord over and over again, that is, after it was selected twice in a row already.

At this point, the only other difference regarding the rhythm section between the two modes is that solo mode (without the option `--experimental`) only plays the straight blues in order to obtain better rhythmic compatibility with a generated soloist (see Section 3.2.3). This helps to avoid rhythmic chaos that may arise due to the current lack of a direct link between the generated rhythms.

3.2.3 Soloist generation

On top of the generated rhythm group, the Robot Composer has to produce a melodic leading voice that harmonizes with the backing group. Therefore, it picks a soloist among the more dominant instruments (e.g. excluding muted trumpet) and generates a melodic phrase using the minor blues scale. We have discussed in Section 2.3 that it is suited for this purpose and it is available in Sonic Pi starting from version 2.11.

To avoid disharmonies, we start and end the generated phrase with the ground note and the fifth, respectively. In between, the notes are mostly chosen at random from the blues scale with some minor limitations: during each half bar (2 beats), the selected notes are restricted to be within a fifth of the last played note before this half bar. This still leaves more than an octave of range, whereas in practice, it seems to be enough to enforce a melodic output to be generated. We assume that this comes from the fact that the intersection of the blues scale with this interval is only a low number of notes between 7 and 9. Other than that, only the blue note receives special treatment so that it always gets resolved in the next note, hence, avoiding disharmonies as illustrated in Section 2.3.

For the rhythm of the soloist we reuse the exact same idea as above: we select the standard swing beat with higher preference than all other rhythm patterns from Figure 3.1 and replace some of the notes with rests. Last, similar as for the rhythm group, the soloist is exchanged at random every 12 bars; again chosen from the set of more dominantly sounding instruments.

3.2.4 Tweaking the sound

In order to tweak the resulting sound further, we have explored and partly incorporated several ideas. First, we introduced small breaks of 0.25 beats in between any two consecutive notes of the soloist as any human player could not produce arbitrarily small gaps in practice. This yields a more natural feeling when listening to the Robot Composer. For the same purpose, we slightly randomized the times when unisonous accompanying players hit their notes, so that chords sound less mechanical. Lastly, adding some dynamics by means of changing sound volumes based on randomization completes the current set of articulation features. While these ideas only seem to impose minor changes to the program, some of these were already harder to implement than expected, for example, conflicting with Sonic Pi's internal thread timing protection. However, the resulting musical effects are significantly enhanced which makes it well worth experimenting with these attributes.

Naturally, not all of our ideas allowed to produce desirable results. For example, generating fill-in patterns for the rhythm group using the pentatonic scale (= blues scale without the blue note) led to rather uncoordinated and chaotic pieces being produced. Similarly, adding randomized halftone modulations for the chord-based fill-ins, which can frequently be found in common jazz literature, seems to require a thoughtful underlying structure in order to yield pleasant harmonics. Altogether, tweaking the sound appears to mainly involve balancing randomization and control with regard to a (preferably informed) selection of musical parameters.

3.3 Capabilities and limitations of the Robot Composer

With all its features presented above, the Robot Composer manages to produce countless pieces of music, each of which creates its unique feeling through the combination of the drum pattern, tempo, rhythms and harmonies. While it is already capable of producing pieces that can wholeheartedly be called music, it still comes with a few issues.

First, all of its created pieces lack the articulation performed by musicians on every single note and, in particular, the sound when initiating a new note on an actual instrument due to technical limitations arising from the use of recorded sound samples. To us, this seems to be one of the major problems regarding the produced musical quality. Having said that, we cannot specify clear opportunities for improvement here while one may, however, observe that these features are not so widely present for some instruments or in some types of music other than jazz. This could render the problem at hand obsolete for other use cases (consider the example from [22]).

Furthermore, the Robot Composer currently uses a selection of three jazzblues patterns. While this allows beginners to quickly adapt to the easily distinguishable harmonies, it also provides very limited input for the hidden Markov model that is used to produce the backing harmonies in solo mode. In addition, experienced soloists might prefer a greater variety of jazzblues patterns for play along. From a technical point of view, adding further jazzblues patterns is a simple task, while musically, we cannot predetermine the resulting behavior of the Robot Composer so easily.

Regarding the melody generation of the soloist we have achieved surprisingly good results based on random note selection with only few imposed selection rules (refer to Section 3.2.3). A potentially efficient step of improvement might consider using another hidden Markov model to generate the melody which seems to be the standard approach for melody generation [23, 21]. This Markov model may be constructed either by using ML or, somewhat more complex than before, by hand.

Finally, by default, the Robot Composer is limited to an accompanying straight blues in solo mode (see Section 3.2.2). However, this is not true with the `--experimental` option, but the rhythms of the soloist and the backing group might not fit well together because they are currently selected independently. In order to improve this combination, one might attempt to choose a global rhythm for each (half) bar or find some other suitable direct link between these rhythms. Technically, this global rhythm could be implemented in the form of a singleton object that is accessible from all threads at any time.

Chapter 4

Conclusion

Is the Robot Composer usable as an automatic backing group for play along? — Yes it is! While the generated soloist still requires some refinements to compete with international artists, we have succeeded to create an easy-to-use tool that allows to practice for jam sessions without the need for other players to accompany the soloist. Evidently, two soloists could also share the joy and perform together using a *"call and response"* scheme. Hence, despite all problems during the implementation and with limited time and resources, the outcome of this project suggests that music theory is an efficient and suitable starting point to build up music generation software.

Unlike ML techniques, the Robot Composer not only rearranges/reuses learned patterns, but may actually produce completely new musical ideas. Furthermore, ML does not allow to deduce general musical ideas that easily, while non-ML software could clearly exhibit what restrictions/rules do yield melodious outcomes. However, considering the high amount of randomization used throughout the music generation process, we may doubt that there is a high probability of generating a true musical masterpiece. Also, without performing an in-depth comparison study, the directly comparable ML tools appear to produce more appealing results for now.

Ultimately, concerning Sonic Pi, Section 3.1 allows us to conclude that it may be an appropriate interface for children to learn programming in a playful way, but should currently not be relied on in a professional context: at times it feels inflexible to work with, does not come with a suited development environment and fundamental bugs do not necessarily get resolved in acceptable time even after discovery.

Bibliography

[1] Kathiresan, T.: Automatic Melody Generation. Master's thesis, KTH Royal Institute of Technology Stockholm (2015)

[2] Povel, D.J., et al.: Melody generator: A device for algorithmic music construction. Journal of Software Engineering and Applications **3**(07) (2010)

[3] Website: Henon Map Melody Generator.
http://henon.sapp.org/
Accessed: 22.09.2016.

[4] Keller, B.: Impro-Visor.
https://www.cs.hmc.edu/~keller/jazz/improvisor/
Accessed: 22.09.2016.

[5] Rinchiera, S., Nagler, D., Davison, J., Karunaratne, C.: Youtube: Intelligent Jazz Improvisation Generation using Markov Chains.
https://www.youtube.com/watch?v=zploY043Gx8
Accessed: 23.09.2016.

[6] Youtube: Computer-Generated Jazz Improvisation.
https://www.youtube.com/watch?v=Cbb08ifTzUk
Accessed: 23.09.2016.

[7] Temperley, D., Sleator, D.: Melisma Stochastic Melody Generator.
http://www.link.cs.cmu.edu/melody-generator/
Accessed: 23.09.2016.

[8] Beland, M.: Youtube: Drum Lesson: Jazz Drumming.
https://www.youtube.com/watch?v=0f0xIjvML_Q
Accessed: 23.09.2016.

[9] Xepoleas, J.: Youtube: Jazz Snare Drum Comping Patterns #1 - Online Jazz Drum Lesson with John X.
https://www.youtube.com/watch?v=Yet9KPR7wfw
Accessed: 23.09.2016.

[10] Thomas, P.: Taming the Saxophone: Arranging for Rhythm Section.
https://tamingthesaxophone.com/jazz-piano-guitar
Accessed: 24.09.2016.

[11] Michalkow, M.: FreeDrumLessons: Basic Jazz Drum Pattern.
http://www.freedrumlessons.com/drum-lessons/basic-jazz-pattern.php
Accessed: 26.09.2016.

[12] Musicradar: How to program a jazz drum beat in MIDI.
http://www.musicradar.com/tuition/tech/how-to-program-a-jazz-drum-beat-in-midi-582172
Accessed: 26.09.2016.

[13] Wikipedia entry: Comping.
https://en.wikipedia.org/wiki/Comping
Accessed: 20.02.2017.

[14] Marchand, U., Peeters, G.: Swing Ratio Estimation. In: Proceedings of the 18th International Conference on Digital Audio Effects. (2015) pages 423 – 428

[15] Weitzmann, B.: Per Anhalter durch das Real Book - Part II.
http://www.justchords.de/bass/bwfiles/jazzblues.html
Accessed: 11.01.2017.

[16] Wikipedia entry: Jazzblues.
https://de.wikipedia.org/wiki/Jazzblues
Accessed: 11.01.2017.

[17] Wikipedia entry: Tritone.
https://en.wikipedia.org/wiki/Tritone
Accessed: 03.03.2017.

[18] Sam Aaron: Sonic Pi – The Live Coding Synth for Everyone.
http://www.rubydoc.info/github/samaaron/sonic-pi/
Accessed: 11.11.2016.

[19] Joseph Wilk: [Bug] Maximum limit on file size.
https://github.com/samaaron/sonic-pi/issues/146
Accessed: 18.01.2017.

[20] Wikipedia entry: Frequenzen der gleichstufigen Stimmung.
https://de.wikipedia.org/wiki/Frequenzen_der_gleichstufigen_Stimmung
Accessed: 12.01.2017.

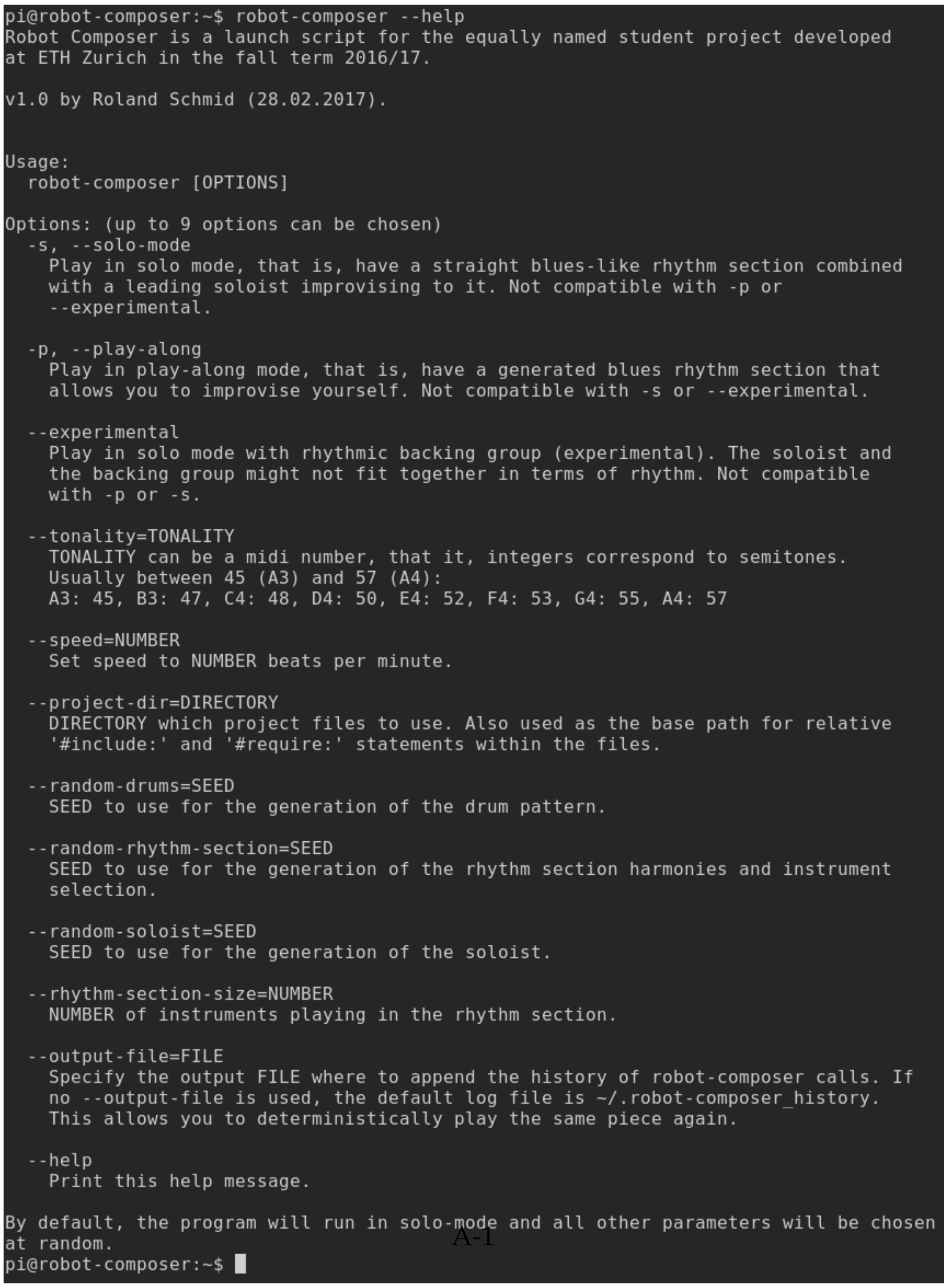
[21] Victor, J.: Jazz Melody Generation and Recognition. (2012)

[22] Newman, R.: Adding a new sample based flute voice for Sonic Pi.
https://rbnrpi.wordpress.com/project-list/adding-a-new-sample-based-flute-voice-for-sonic-pi/
Accessed: 25.10.2016.

[23] Wikipedia entry: Pop music automation.
https://en.wikipedia.org/wiki/Pop_music_automation
Accessed: 23.02.2017.

Appendix A

Appendix



```
pi@robot-composer:~$ robot-composer --help
Robot Composer is a launch script for the equally named student project developed
at ETH Zurich in the fall term 2016/17.

v1.0 by Roland Schmid (28.02.2017).


Usage:
  robot-composer [OPTIONS]

Options: (up to 9 options can be chosen)
  -s, --solo-mode
    Play in solo mode, that is, have a straight blues-like rhythm section combined
    with a leading soloist improvising to it. Not compatible with -p or
    --experimental.

  -p, --play-along
    Play in play-along mode, that is, have a generated blues rhythm section that
    allows you to improvise yourself. Not compatible with -s or --experimental.

  --experimental
    Play in solo mode with rhythmic backing group (experimental). The soloist and
    the backing group might not fit together in terms of rhythm. Not compatible
    with -p or -s.

  --tonality=TONALITY
    TONALITY can be a midi number, that it, integers correspond to semitones.
    Usually between 45 (A3) and 57 (A4):
    A3: 45, B3: 47, C4: 48, D4: 50, E4: 52, F4: 53, G4: 55, A4: 57

  --speed=NUMBER
    Set speed to NUMBER beats per minute.

  --project-dir=DIRECTORY
    DIRECTORY which project files to use. Also used as the base path for relative
    '#include:' and '#require:' statements within the files.

  --random-drums=SEED
    SEED to use for the generation of the drum pattern.

  --random-rhythm-section=SEED
    SEED to use for the generation of the rhythm section harmonies and instrument
    selection.

  --random-soloist=SEED
    SEED to use for the generation of the soloist.

  --rhythm-section-size=NUMBER
    NUMBER of instruments playing in the rhythm section.

  --output-file=FILE
    Specify the output FILE where to append the history of robot-composer calls. If
    no --output-file is used, the default log file is ~/.robot-composer_history.
    This allows you to deterministically play the same piece again.

  --help
    Print this help message.

By default, the program will run in solo-mode and all other parameters will be chosen
at random.
pi@robot-composer:~$ █
```

Figure A.1: The `--help` output of the `robot-composer` invocation script