PlayGuru, a music tutor

Marc Groenewegen
Hogeschool voor de Kunsten Utrecht
Ina Boudier-Bakkerlaan 50
3582 VA Utrecht
The Netherlands
marc.groenewegen@hku.nl

Abstract

PlayGuru is a practice tool being developed for beginning and intermediate musicians. With exercises that adapt to the musician, the intention is to help a music student to develop several playing skills and motivate them to practice in-between classes. Because all exercise interaction is entirely based on sound, the author believes PlayGuru is particularly useful for blind and visually impaired musicians. Research currently focuses on monophonic exercises. This paper is a report of the current status and ultimate goals.

Keywords

Computer-assisted music education, DSP, machine learning

1 Project objectives

The main reason for writing this paper is to bring the project to the attention of others so they can use, improve and benefit from the ideas, technology and the intended end product.

Even though few user experiences can be reported at the time of writing, some intermediate results and plans for the near future are given towards the end of this paper.

PlayGuru is a music tutor that operates exclusively in the sound domain. Because the focus is only on music, the author believes it is a very useful tool for beginning and intermediate musicians and particularly useful for blind and visually impaired musicians.

1.1 Practice motivation

How does an amateur musician find the motivation to pick up their instrument and play? What motivates children to practice?

These questions probably have a large spectrum of answers. Let us focus on two motivational forces: personal growth and affirmation.

PlayGuru is a set of music exercises based on an example and response approach. Dialogs usually start with an example being played and base the next action upon the response of the musician. The example and response can also be played simultaneously, creating a sense of playing together. Those synchronised exercises give room to improvisation and exploration.

The way in which PlayGuru aims to keep the user motivated is based on affirmation when the exercise is performed according to predefined objectives.

It will never flag a "wrong note", as this is considered highly demotivating. Instead, it responds by making the exercise slightly easier when it finds you are struggling with the current level, until it gets to a level from which you can continue growing again.

The most important affirmative motivators used are:

- increasing playing speed
- extending the phrase
- increasing complexity of the exercise

The current version contains a very basic user model, which is a starting point for a module that monitors a user's achievements and keep track of their progress, thus supporting their personal growth.

To perform user tests supporting the research, several exercises are being implemented. At the time of writing, a versatile sound-domain guitar tuner with arbitrary tuning is available, as is an exercise for remembering a melodic phrase and a riff trainer for practicing licks at high speed.

Most exercises are developed for the guitar. A great source of inspiration for guitar practice is the book by Scott Tennant: [Tennant, 1995] with a focus on motor skills and automation.

Because all exercises are based on pitch- and onset-detection in sound signals, adaptation for other instruments with discrete pitch and a clear onset should be reasonably straightforward. For instruments with arbitrary pitch

ranges and smooth transitions, as well as human voice, some additional provisions may be necessary.

1.2 Origin of the project

The PlayGuru project started as part of the author's Master's course. While trying to find ways to improve the effectiveness of practice routines for the guitar, some research was done into existing solutions.

Several systems for computer assisted music training were found and some have been put to the test. A shortlist is included at the end. One thing all the encountered solutions have in common is that they rely heavily on visual interaction. In many cases this implies written score or tablature, in other cases a game-like environment in which the user has to act upon events happening in a graphic scene.

In several cases the author found the visual information distracting from the music. Thus the idea arose for a practice tool exclusively working with sound.

Shortly after that, the foundation Connect2Music [1] came into view. Connect2Music, founded in 2013, provides information with respect to music practice by visually impaired musicians.

According to [Mak, 2015], the facilities for blind music students in The Netherlands are limited. Even though the situation is improving, a practice tool which focuses only on the music itself would be a much wanted addition.

Thus a project was born: to find ways to improve the learning path for beginning and intermediate musicians with music as the key element and primarily addressing blind and visually impaired people.

The prototype being developed for performing this research is called PlayGuru. The envisioned end product aims to help and encourage a music student to perform certain exercises inbetween music classes and is meant to complement rather than replace regular classes from a human teacher.

Through the contacts of Connect2Music with the community, several blind and visually impaired musicians and software developers in The Netherlands and Belgium expressed their interest in this project and offered help to assess and assist.

[1] `https://www.connect2music.nl`

2 Research

To support the research with experiences of end users, some application prototypes are being developed. The adaptive exercises used in these prototypes will briefly be introduced separately.

This chapters discusses the software and the chosen methods for interacting with the user.

2.1 Software

The framework and all exercises are currently implemented in C++11. For audio signal analysis, the Aubio [2] library is used. Exercises are composed in real time according to musical rules or taken from existing material like MIDI files and guitar tablature.

2.2 Dependencies

Development is done on Linux and Raspbian. Porting to Apple OSX should be relatively easy but has not been done yet. The most prominent dependencies, as in libraries, are jackd, aubio, fftw3 and portmidi. For generating sound, Fluidsynth is used. Stand-alone versions use a Python script to connect the hardware user-interface to the exercises.

2.3 Practice companion

When playing along with a song on the radio you will need to adjust to the music you hear, as it will not wait for you. Playing with other musicians has entirely different dynamics. People influence each other, try to synchronise, tune in and reach a common goal: to make the music sound nice and feel good about it.

When practicing music with a tool like PlayGuru it would be nice to have a dialog with the tool, instead of just obeying to its rules. This is exactly what makes PlayGuru interact so nicely. It listens to you and adapts, thus behaving like a practice companion.

How this is achieved is shown with reference to the software architecture and indications which parts have been realised and which are being developed.

2.4 Architecture

The modular design of PlayGuru is shown in figure 1, with the Exercise Governor as the module from which every exercise starts.

The Exercise Governor reads a configuration file containing information about the user, the type of exercise, parameters defining the course of the exercise, various composition settings and possibly other sources like MIDI files.

[2] `https://aubio.org`

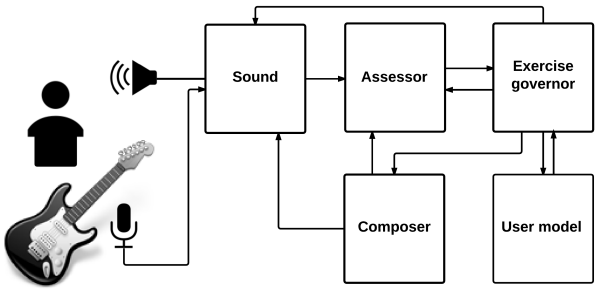

Figure 1: PlayGuru's software architecture

When the exercise starts, the Composer will generate a MIDI track, or read it from the specified file. During the exercise, this generating may take place again, depending on the type of exercise and the progress of the musician.

The MIDI track is played by the sound module, which also captures the sound that comes back from the musician or their instrument.

The Assessor contains all the sound processing and assessment logic and reports back to the Exercise Governor, which calls in the help of the User Model to decide how to interpret the data and what to do next.

2.5 Playback and analysis

Playback and analysis run in separate threads, but share the same time base for relating the output to the input.

Incoming audio is analysed in real time to detect pitch(es) and onsets, which are used to assess the musician's play in relation to the given stimuli. Pitch and onset detection are done using the Aubio library.

2.6 The Exercise Governor

Every exercise type is currently implemented as a separate program. The Exercise Governor is essentially a descriptive name for the main program of each exercise, which uses those parts from the other modules that it needs for a certain exercise. Currently these are compiled and linked into the program. With these building blocks it determines the nature of the exercise.

As an example: to let the musician work on accurate reproduction of a pre-composed phrase, the Exercise Governor will ask the Composer to read a MIDI file, call the MIDI play routine from the Sound module, then let the Assessor assess the user's response and consult the User Model, given the Assessor's data, for determining the next step.

For a play-along exercise using generated melodies, the Exercise Governor uses routines

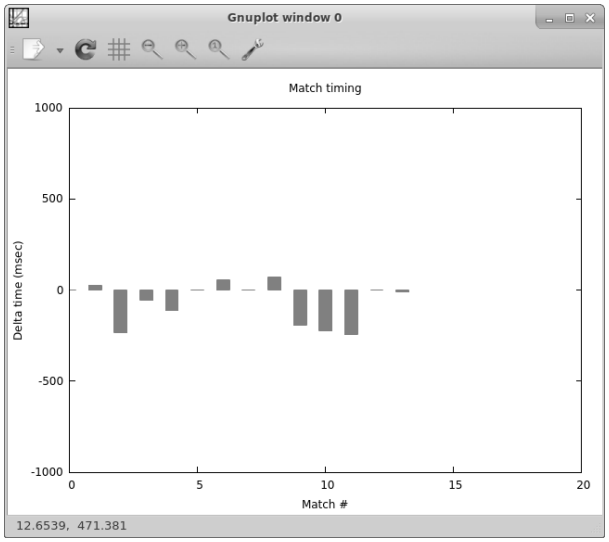

Figure 2: Note onset absolute time difference

from the same modules, but with a different intention. In this case it would let the Composer create a new phrase when needed, ask the Assessor to run a different type of analysis and perform concurrent scheduling of playback and analysis.

2.7 The Assessor

PlayGuru's exercises generally consist of a play- and evaluation loop. For some exercises, the evaluation process is run after the playback iteration, while for others they run simultaneously.

Figure 2 shows the measured timing of a musician playing along with an example melody. The time of each matched note is compared to the time when that note was played in the example. In this chart we see that the musician played slightly "before the beat".

This absolute timing indicates whether the musician is able to exactly copy the example, which can be seen clearly for a melody constisting of only equidistant notes.

More interesting however is relative timing. This indicates whether the musician keeps the timing structure of the example intact. In this case we calculate the differences in spacing of the onsets of successive notes, either numerically or as an indication of "smaller/equal/larger" and compare the result to the structure of the example. In figure 3 this is shown. Here we can see that the musician started out with confidence and needed more time to find the last notes of the phrase. The example consisted of equidistant notes, which would result in a chart of zeros and is therefore

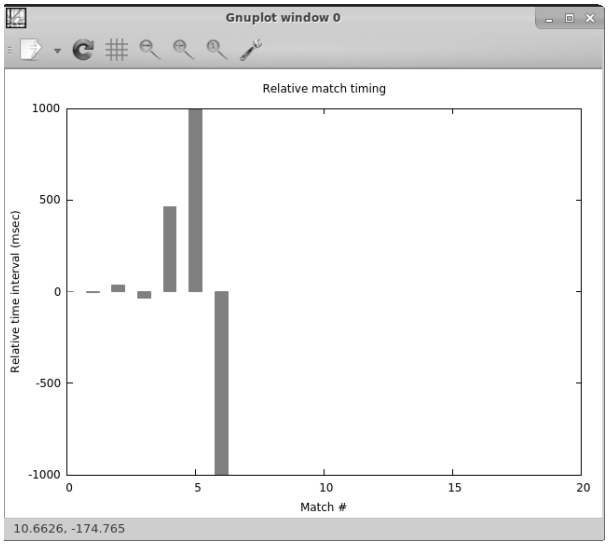

Figure 3: Note onset relative time difference

omitted.

2.8 The Composer

The Composer generates musical phrases based on given rules. The current implementation uses melodic intervals in a specified range, with an allowed set of intervals and within a given scale. The scale is listed as a combination of tones (T) and semitones (S), as in the examples in table 1 and can be specified as needed.

T,T,S,T,T,T,S	major
T,S,T,T,T,S,T	minor
S,S,S,S,S,S,S,S,S,S,S	12-tone

Table 1: Scale examples

2.9 The User Model

At the time of writing, the User Model is partly implemented.

The part that has been implemented and is currently tested by end users is the mapping from analysed properties of the user's playing to parameters that are musically meaningful or significant for the user's ambitions.

In general this mapping is a linear combination of those properties. For example, the user's melodic accuracy can be expressed as a combination of hitting the correct notes and the lack of spurious or unwanted notes.

The weight factors are empirically determined, as are several parameters in the exercises, such as the number of repetitions before moving on or the required proficiency for increasing the level of an exercise.

It is here where the assistance of a Machine Learning algorithm is wanted: to learn which weight factors and other parameters contribute to the user's goals and to optimise these. This has not been implemented yet and is currently being studied.

2.10 Melodic similarity

There are several ways to find out the similarity between the given example and the musician's response. In the current research, only the onset (i.e. start) and pitch of notes are taken into account. Although timbre, loudness and various other features are extremely useful, these are ignored for the time being.

In the exercises where the user is asked to memorise and copy an example melody, the accomplishment of this task is purely based on hitting the correct notes in the correct order. The similarity however is also reflected in the timing. The extent to which the musician keeps the rhythm of the example intact is a property that is measured and evaluated.

In the exercises where the musician plays along with a piece of music, we have much more freedom in the assessment. In this case, playing the exact same notes as in the example is not always necessary. For some exercises it would suffice to improvise within the scale or play certain melodic intervals.

In these situations, similarity measures also allow for more freedom.

A method that is used in an exercise called "riff trainer", focused on automation of and creating variations on a looped phrase, observes notes in the proximity of example notes and draws conclusions based on the objectives of the exercise. This allows for both very strict adherence to the original melody as well as melodic interval-based variations, depending on the assumed objectives.

Another method compares the Markov chain of the example with that of the musician's response.

Some inspiration is gained from this book about melodic similarity: [Walther B. Hewlett, 1998]

3 Personal objectives

Figure 4 shows that an exercise is 'composed' and played. The response of the musician is assessed and mapped to temporary skills. Long-term skills are accumulated in a user model, which tries to construct an accurate profile of the musician and their personal objectives.

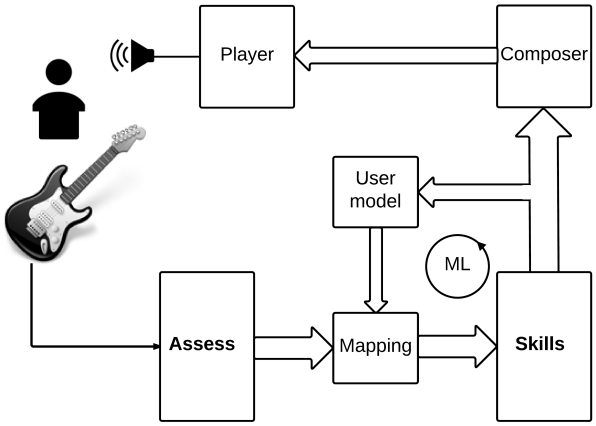

Figure 4: Machine assisted learning supported by machine learning

Examples of these objectives are to strive for faster playing, memorise long melodies or improve timing accuracy. These need to be expressed as quantifyable properties. Faster playing can be expressed as playing a phrase faster than before or playing it faster than the given example, which can be measured.

Likewise, memorising a melody involves the length of the melody that can accurately be played at reasonable speed. Obviously, the concepts 'accurately' and 'reasonable' have to be quantified.

An indication of accuracy in playing is obtained by measuring the number of spurious notes, missed notes and timing.

Some approaches for machine learning (ML) will be investigated. The term "machine learning" is used here to express that the machine itself is learning and does not refer to the "machine assisted learning", which is the main topic of this paper. A machine learning algorithm is thought to be able to achieve the user's objectives by adjusting the mapping parameters that translate measured quantities to short-time skills and several properties of the exercises.

Depending on the exercise, various factors are measured, such as matched notes, missed notes, spurious notes, adherence to the scale, speed and timing accuracy.

These quantities are mapped to short-term skills according to table 2.

Apart from these measured data, the exercises also contain configuration parameters that can be optimised for each user. These are found in composer settings and the curves used to control the playing speed and exercise complexity.

Measured quantity	Mapped to
timing deltas	timing accuracy
missed notes	melodic accuracy
spurious notes	clutter

Table 2: Mapping measured data to skills

4 Hardware

The starting point for this project is to assess the sound of an unmodified instrument. In this section, the current choice of hardware is discussed.

A brief side-project was undertaken to equip an acoustic guitar with resistive sensors for detecting the point where strings are pressed against the fretboard, but because this doesn't look and feel natural, would imply that all users would need to install a similar modification and would exclude all instruments other than guitar, this was discarded.

Because nylon-string acoustic guitar is the primary instrument for the author as well as for lots of beginning music students, the decision was to analyse the sound of the instrument with a microphone or some kind of transducer.

Using a microphone raises the problem that the sound produced by PlayGuru interferes with the sound of the instrument. Source separation techniques are not considered viable for this project due to the added complexity and because we want to be able to play and listen simultaneously, often to the exact same notes. This would justify a study of itself.

Requiring the musician to use a headset is also considered undesirable. So the only option left seems to use a transducer attached to the instrument.

After some experimenting with various combinations of guitar pickups, sensors, preamps and audio interfaces, it turned out that a combination of a simple piezo pickup and a cheap USB audio interface does the job very well.

Successful measurements were done with the piezo pickup attached to the far end of the neck of a guitar. It is advised to embed the pickup into a protective cover to prevent the element itself and the cable from being exposed to mechanical strain and mount it with the piezo's metal surface touching the wood of the guitar using a rubber padded clamp from the DIY store.

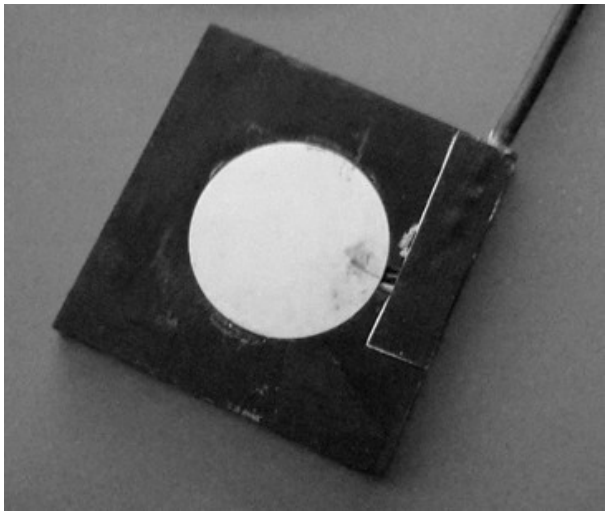

Figure 5: Embedded piezo pickup

5 Dissemination

User's experiences and feedback are of crucial importance for the development of this tool. While a browser application or mobile app seem obvious ways to reach thousands of musicians, development is currently done on Linux and Raspbian. This is a deliberate choice, partly inspired by the author's lack of experience with Webaudio intricacies and the acclaimed large round-trip audio latency of Android devices, for which a separate study may be justified.

For a large part however, this choice is supported by the wish to have an inexpensive stand-alone, self-reliant, single-purpose device.

A series of stand-alone PlayGuru test devices are being developed, based on a Raspberry Pi with a tactile interface meant to be intuitive to blind people. The idea is to attach a piezo pickup to the instrument, plug in, select the exercise and start practicing.

6 Conclusions

Several beginners, intermediate guitar players and some people with no previous experience have used PlayGuru's exercises in various stages of development. The two most mature exercises used are copying a melody and playing along with a melody. In most cases the melodies were generated in real time based on the aformentioned interval-based rules. In some cases a precomposed MIDI file was used.

From the start it was clear that users enjoy the fact that PlayGuru listens to them and rewards "well played" responses with an increase in speed or making the assignment slightly more challenging.

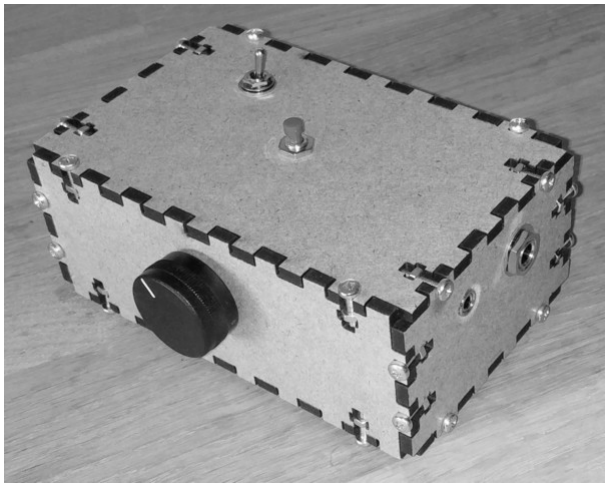

Figure 6: Stand-alone device for user tests

Several users mentioned a heightened focus, meaning that they were very concentrated for a longer time to keep the interaction going and the level rising. Mistakes bring down the speed or level in a suble way and do lead to a slight disappointment, which in many cases proved to be an incentive to get back into the 'flow' of the exercise.

The project is work in progress. For a well-founded opinion on the practical use, a lot more user tests need to be done but the author's conclusion, based on results so far, is that the approach is promising.

7 Future Work

At the time of writing, research concentrates on monophonic exercises with a basic machine learning algorithm.

For the near future the author has plans to perform a study with a larger group of users who can use the system by themselves for a longer time. This requires creating test setups and/or porting the software to other platforms.

Monophonic exercises may be the preferred way to develop several skills, but being able to play, or play along with, your favourite music is much more motivating, particularly for children. An approach resembling Band-in-a-Box® is taken, using a multi-channel MIDI file as input, with the possibility of indicating which channels will be heard and which channel will be 'observed'. This requires both polyphonic play and analysis, which are largely implemented but currently belong in the Future Work section. On the analysis side, a technique based on chroma vectors [Tzanetakis, 2003] is being tested.

Machine learning strategies are being studied but have not yet been implemented. The assistance of co-developers would be much appreciated.

8 Acknowledgements

I want to thank the HKU for supporting and facilitating my work and in particular Gerard van Wolferen, Ciska Vriezenga and Meindert Mak for their insights and ideas. Gerard van Wolferen and Pieter Suurmond were of great help proof-reading and correcting this paper. Lastly I would like to thank the LAC review committee for their excellent observations.

9 Other solutions for computer-assisted music education

- i-maestro
- Bart's Virtual Music School
- Rocksmith ™
- yousician.com
- bestmusicteacher.com
- onlinemuziekschool.nl
- gitaartabs.nl

References

Meindert Mak. 2015. Connecting music in the key of life. 1.2

Scott Tennant. 1995. *Pumping Nylon, The Classical Guitarist's Technique Handbook*. Alfred Music. 1.1

Ning Hu; Roger B. Dannenberg; George Tzanetakis. 2003. Polyphonic audio matching and alignment for music retrieval. *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 7

Eleanor Selfridge-Field Walther B. Hewlett. 1998. *Melodic Similarity*. The MIT Press. 2.10