Empirical Modelling for the Composition and Education of Guitar Tablature Music

0320175

Abstract

Empirical modelling encourages exploration and experimentation during the development process. The most effective way to learn is by trial and error and learning by doing. Empirical modelling thusly appears ideal for use in an educational context. An empirical approach is also adopted for the composition of music since it is necessary to amend a piece, listen to how it sounds and correct if required. This paper will look at building a model in tkeden to take advantage of these aspects to create a platform for learning some music theory whilst composing music.

1 Introduction

Empirical modelling, as suggested by the name, encourages experimentation during development of a computer based model. It is suggested that this approach is better suited to developing software than traditional formal approaches. In addition, the model created can be altered by the user to extend or customise the functionality post development. This allows and encourages exploration of the model to develop a rich 'what-happens-if…' interaction.

This paper will explore how this experimental approach to programming can produce a flexible model suited for education of music theory, in particular the composition using guitar tablature. Guitar tablature is a simple notation for describing a musical score by diagrammatically showing what notes are played in what order and how.

2 Uses of Empirical Modelling

Models differ from applications in that they represent an artefact in the real world. The result is not generally a tool like a word processor but a model of a real world object or system which can be experimented upon.

Formal methods are not agile enough to efficiently accommodate changes and unanticipated problems and also go against the natural trial and error methodology utilised naturally by programmers. The empirical approach allows programmers to develop the program and test it 'on-the-fly' to get immediate feedback on its operation and appearance and to make any changes as required. As mentioned in the previous section, users can also continue modification of the model and this interaction can build the user's own experience of what is being modelled. Experience is the basis for understanding hence encouraging experience of a model educates the user about it.

2.1 EM for Education

Computer programs are ultimately made for the users and an important part of computer usability is enabling new users to quickly learn the system or application. This is best achieved, not by providing a wordy document, but by allowing practical exploration of the tool whilst providing information to fill in background theory and answer questions as they arise in their natural context. This method of learning is true for most subjects, practical or theoretical.

The eden[1] tool provides the basis for exploration of a predefined model by continually allowing the user to redefine any aspect of the model to see what will happen. If the creator of the model provides adequate definitions and functionality, subsequent users can then develop their understanding of the model as they work with it.

When learning something new, answers are found by asking questions which can not all be addressed whilst using an inflexible framework. Developers can not anticipate all of the questions which may be raised during the use of the model or the learning of the underlying concepts. A flexible model accommodates this and allows the user to learn by doing and answering the questions themselves. This encourages improved recollection at a later date.

2.2 EM for Music Composition

One area well suited for modelling empirically is music composition. When composing a new piece of music, it is necessary to create novel combinations of notes. It is then the case that the new note sequence or chord must be heard before being finalised with the music. Similar projects to this are Ashley Ward's Music Score Experimentation[2], and David Wai's Music Program[3],. These models demonstrate the usefulness of using this modelling method for creative composition.

An eden model also avoids stifling creativity by removing bounds and limitations that exist in classical programs. The explorative nature of empirical modelling even allows impossible combinations of notes as defined by the range of motion of the human hand. This allows full experimentation and freedom for the user which provides a good platform for learning music theory as mentioned in Section 2.1.

A fluid trial-and-error approach is even more important for a popular use of guitar tablature on the internet: transcribing existing music. There are many web sites that provide individual's interpretations on how to play a piece of music. To transcribe music by ear, it is necessary to listen to the music, note down potential notes heard then listen to what has been written and amend as required.

3 Implementation in tkeden

The model created to support this paper is intended to provide a model for composing guitar tablature. Tablature is a notation based upon a construal of the guitar fretboard hence this is the central concept upon which this model is based. A diagrammatic representation and mapping of the notes on the fretboard can be seen in Figure 1 (adapted from [5]). As well as the actual tool for constructing the tablature, information is provided to allow the user to make well informed decisions on what notes to use and why. This provides a platform for learning music theory whilst using the model. Using the theory in a practical situation reinforces the new information making the educational tool more effective.

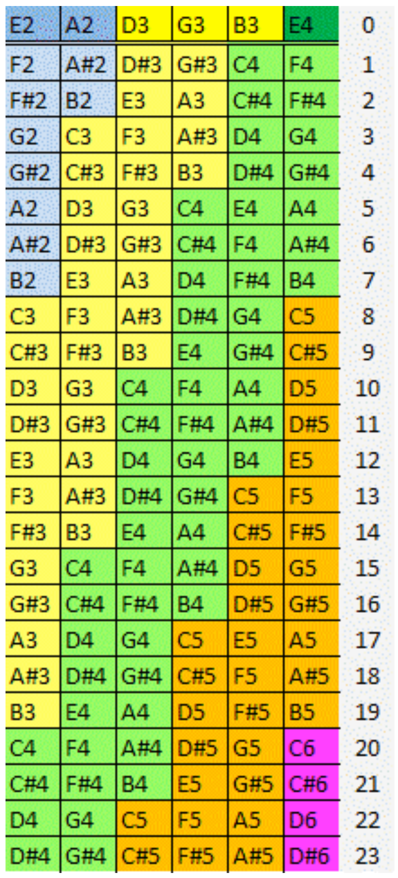
3.1 MIDIModWin – MIDI Module

To obtain useful feedback on tablature, audio output is required in tkeden. There is currently no audio output functionality in tkeden hence it is required that a module is made to output specific notes as defined in eden. The program created is called MIDIModWin as it is a MIDI module for Windows.

3.1.1 MIDIMod Concept

To allow audio output from eden a function has been created in the C language (the same as tkeden itself) that can be bound to an eden function. For the scope of this model, the Windows multimedia library has been used to output specific musical notes using the MIDI audio format. MIDI was used because a message can be constructed to play specific musical notes synthesised through a system's MIDI device[5-9].

To avoid development environment issues associated with compiling eden to bind this method to a function, the program was made executable with arguments being passed to the program defining the note string to be played. To allow future binding, a separate method, `play_tune(char *note_seq, int seq_length)`, was used for all the functionality of the MIDI output, this is simply called by the main method upon execution passing the program arguments as parameters.



E2	A2	D3	G3	B3	E4	0
F2	A#2	D#3	G#3	C4	F4	1
F#2	B2	E3	A3	C#4	F#4	2
G2	C3	F3	A#3	D4	G4	3
G#2	C#3	F#3	B3	D#4	G#4	4
A2	D3	G3	C4	E4	A4	5
A#2	D#3	G#3	C#4	F4	A#4	6
B2	E3	A3	D4	F#4	B4	7
C3	F3	A#3	D#4	G4	C5	8
C#3	F#3	B3	E4	G#4	C#5	9
D3	G3	C4	F4	A4	D5	10
D#3	G#3	C#4	F#4	A#4	D#5	11
E3	A3	D4	G4	B4	E5	12
F3	A#3	D#4	G#4	C5	F5	13
F#3	B3	E4	A4	C#5	F#5	14
G3	C4	F4	A#4	D5	G5	15
G#3	C#4	F#4	B4	D#5	G#5	16
A3	D4	G4	C5	E5	A5	17
A#3	D#4	G#4	C#5	F5	A#5	18
B3	E4	A4	D5	F#5	B5	19
C4	F4	A#4	D#5	G5	C6	20
C#4	F#4	B4	E5	G#5	C#6	21
D4	G4	C5	F5	A5	D6	22
D#4	G#4	C#5	F#5	A#5	D#6	23

Figure 1: Mapping of Notes on Guitar Fretboard

3.1.2 Module Usage

To specify a string of notes and chords to be played by the MIDI module, a definitive notation has been conceived. To call the program the following format must be obeyed:

```
MidiModWin [note_string] [no_of_notes]
[tempo]
```

The `note_string` argument is the sequence of notes and chords to be played and each note is made up of three characters “*NPO*”. ‘*N*’ is the note from ‘A’ to ‘G’, ‘*P*’ is the pitch either ‘#’ for sharp or ‘ ‘ (space) for a standard pitch, and ‘*O*’ is the octave from 2 to 6. These notes correspond to the notes on the fretboard diagram in Figure 1.

To define a chord (where subsequent notes are all played at once) the construct “Ch*n*” is used before the notes in the chord. The *n* defines the number of notes following the construct to be included in the chord. Some examples of note strings are shown in Figure 2 below. The number *n* must be 0-9 which easily allows the six strings to be played at once but also allows additional notes and even the same string played many times should the user wish to try it.

```
(1) MidiModWin “E#3F 3B 4” 3 1
(2) MidiModWin “Ch2B 2E 3” 2 2
```

Figure 2: Example MIDIModWin Usage

Example (1) shows a command to play back a three note sequence at the standard tempo and example (2) plays a single chord composed of two individual notes played at double tempo. A combination of the two note strings would be `“E#3F 3B 4Ch2B 2E 3”` which would need to be followed by the number ‘5’ then the desired tempo. This would result in the first three notes in sequence then the fourth is the chord made of the final two notes.

3.1 DefTab Definitive Model for Tablature Composition

The model created to utilise the MIDI module is called DefTab as it is a definitive tablature model. A screenshot of the model can be seen in Figure 3. The main aspect of the model is the tablature panel, within which the user can construct the tablature music. This is achieved by typing the fret number in the fret textbox and clicking on the string where it is desired for the note to go. This will result in a note appearing on the tab panel. In this part of the model, the main dependency comes from the mapping of tab spots onto the musical notes via the fretboard representation. Using the fretboard to limit the notes a user can put into a composition appears contrary to the concept of exploration and freedom afforded by utilising empirical modelling. It is necessary however, to limit this as if notes can not be played on the guitar, they should not appear in guitar tablature. Whilst this is the case, users are still allowed to experiment with impossible combinations of notes and also the model can be extended to more notes if required. The final aspect of the tablature panel is the ability to playback the music created utilising the MIDIModWin program outlined in the previous section. This gives the user immediate feedback in keeping with the empirical modelling paradigm to allow trial and error without the need for playing the tablature on guitar.

To ensure correct audio output, MIDIModWin needs to reside on the C:/ drive in no folder or the path in DefTab.s needs to be amended to point to the correct location.

The educational aspect of the model comes from allowing the user to investigate the effect of different combinations of notes in conjunction with existing scales and chords being displayed below the tab panel. Music is made up of chords and sequences of notes. Effective sequences of notes are achieved by playing scale segments or by playing around chords hence providing the user this information, they can experience how these sound. Through this experience, the user can learn how well constructed music is composed. A number of chords[10] and scales[11] are provided using the eden list notation and further examples can be added by the user.

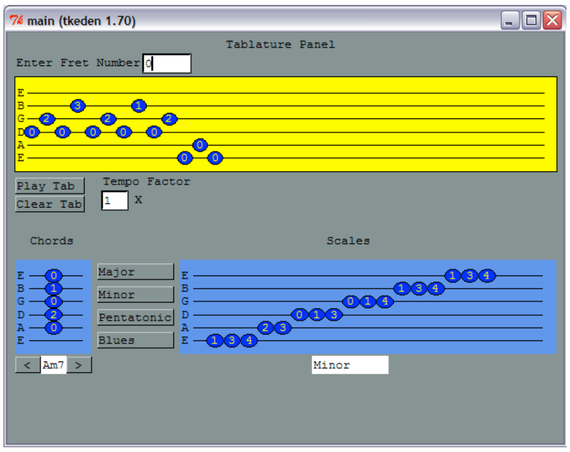


Figure 3: DefTab Screenshot

4 Evaluation

The final model provides the information a user requires to confidently get started with composing music. The additional information also provides hints on what notes are being played in existing mu-

music being transcribed since all music is built the same way. It was hoped that the chords and scales being displayed would automatically be chosen depending upon the current note entered into the tab. This would have provided more contextual assistance rather than relying on the user to select these construct themselves. On the other hand, this way, freedom to explore and experiment is preserved by not limiting the user's interaction too much.

Functionally, the length of the tab panel is a bit short, multiple panels could be employed in the future. For the scope and purpose of this model, the single row of tablature is sufficient. Also a good extension to the model would come from the MIDI module; it currently only works under Windows so other versions should be developed for the different versions of tkeden. Finally, the MIDI module was intended to be bound to an eden function. Sadly this was not realised during this project but could provide built-in functionality for future projects.

Acknowledgements

Special thanks to Meurig Beynon and Antony Harfield for assistance with various tkeden concepts. Thanks also to Daniel Taylor for guidance and confirmation of guitar theory upon which this project is based.

References

[1] Eden Language Reference, *http://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/eden/langref/*

[2] Ashley Ward. Music Score Experimentation *Eden Project,* 1999. *http://empublic.dcs.warwick.ac.uk/projects/musicscoreexptWard1999/*

[3] David Wai. Music Program, 2000. *http://empublic.dcs.warwick.ac.uk/projects/musicWai2000/*

[4] Eric Jake. Where Did Notes and Scales Come From? *http://www.geocities.com/ericjake89/*

[5] MIDI Toolkit in C#. CodeProject.com. *http://www.codeproject.com/KB/audio-video/MIDIToolkit.aspx#Input*

[6] Twinkle.c. Low Level API *http://www.borg.com/~jglatt/tech/lowmidi.htm*

[7] GM Patch Numbers (Instruments). *http://www.borg.com/~jglatt/tutr/gm.htm#Patch*

[8] MIDI Note Numbers. *http://www.harmony-central.com/MIDI/Doc/table2.html*

[9] MIDI Messages. *http://www.midi.org/about-midi/table2.shtml*

[10] Guitar Chords. *http://www.guitaralliance.com/guitar_lessons/guitar_chords/common_open_chords.htm*

[11] Scales. *http://www.all-guitar-chords.com/guitar_scales.php*