Interactive Music Composition with a Minimum of Input States

Justin R. Shuttleworth (shuttleworth@cf.ac.uk)
Department of Physics and Astronomy,
University of Wales College of Cardiff,
PO Box 913
Cardiff CF1 3TH
Wales, UK

August 9, 1991

Abstract

Most interactive music composition systems developed previously have demanded a fair degree of physical expression from their operator; for instance, the ability to accurately control a mouse. We have developed a musical sequence generator whose operation can be controlled by a minimum of input states. Such a generator could be controlled by a severely physically handicapped individual. The generator uses Markov-chain arrays of state transition probabilities to decide upon the pitch of forthcoming notes, and 1-dimensional space grammars to create rhythmic motifs. Short sequences of music are generated and played, repeatedly. The operator can express appreciation (or indifference) to the generated sequence by "voting" at the end of each sequence. The probabilities in the transition array (or space grammar) are manipulated after each vote in an effort to disperse generation of unsatisfactory musical output and to encourage generation of sequences similar to those appreciated by the operator. In this way, an operator of the program may participate in the steering of the generator towards musical output that is perceived as "good". We present an example of the evolution of such a sequence, and describe the operation of the present system. We deliberately chose such relatively computationally undemanding generation methods so that the system could be implemented on a cheap and readily available personal computer. With the addition of some inexpensive hardware the generator could indeed be controlled by someone with perhaps only binary or tri-state means of expression.

Motivation and Introduction

It is an obvious statement to make, but there must be a similar proportion of people with some musical ability amongst the physically disabled population as there is in the able-bodied. Even those physically disabled people who struggle to use conventional sequencing packages to realise their musicality are denied the experience of real-time composition or improvisation. I resolved therefore to construct an interactive music composition system that could be controlled by someone with very limited physical ability. To be of general use, the system should be capable of running on a cheap computer with a minimum of specialised hardware. Therefore the
system described below runs on any computer with an IBM PC compatible architecture, provided that it possesses at least 512K of RAM. A snapshot of the system's parameters may be taken at any time and saved for later restoration. A snapshot will fit on a single smallest-capacity PC-based floppy disk and therefore the composition system does not require a hard disk drive to operate. The system generates note events in real-time and transmits these note events to a commercial synthesizer as MIDI commands. The only specialised output hardware required in the computer therefore is a MIDI interface. All event timing is performed in software and thus the simplest of interfaces can be used. All input to the system is through "wait and shoot" style menus, where each menu item is highlighted in turn and the user selects the highlighted item by hitting any key on the keyboard or clicking a mouse button. For most people no specialized input hardware will be required but this binary input state could be taken from some more sophisticated interface (e.g. eye-blinc or breath control). The speed at which the menus are traversed can be tailored to suit an individual user. If a user finds it particularly difficult to time their actions and has at least a tri-state means of expression, the automatic menu traversal can be deactivated. The transition to the third input state can then manually advance the menu-item highlighting.

Generation Method

The pitch and duration of a voice's forthcoming note event is generated by a Markov chain and a 1-dimensional space grammar respectively. A description of these generating methods can be found in [Jones 1981]. For efficiency all probabilities are stored as integers where the probability represented is equal to the integer divided by a thousand.

For each forthcoming note event a result index is generated from the associated voice's Markov chain probability matrix. The result index then selects a diatonic pitch from an array of possible pitches. One of the possible pitches will produce silence, thus creating rests in the voice's melody. The matrix is of size \( m^n \) where \( m \) is the number of possible pitches. The \( n - 1 \) most recent result indices of this voice are used as subscript into the matrix to select a vector of \( m \) probabilities. A new result index is then chosen randomly according to the probabilities contained in this vector.

The duration of each forthcoming note event is generated from a grammar containing a single terminal \( a \) of the form

\[
A \rightarrow AA \\
A \rightarrow a
\]

with associated probability array \( P = \{p_1, p_2\} \). The productions are chosen randomly according to the probabilities in \( P \) where \( p_1 \) is the probability of selecting production (1) and \( p_2 \) that of production (2). The tree generated by the application of these productions is interpreted as dividing some given duration \( t \) into one or more shorter durations. The given duration is a multiple of two in units of the dispatcher's "ticks" (see below). A duration of one cannot be subdivided and therefore the selection of production (1) on a duration of one is prescribed. A separate array \( P \) is associated with each voice.

Implementation

The system was implemented in C using Borland's Turbo C V2.0 under MSDOS V2.30. The system consists of three main tasks running concurrently, the midi scheduler/dispatcher, the user interface and the note event generator. A freely available package "CTask" as described in [Wagner 1989] provided the facilities for multi-tasking, inter-process communication, and real-time programming.
Midi events are scheduled and dispatched in the system by an implementation of the algorithm described in [Dannenberg 1988]. This efficient algorithm performs scheduling and dispatching in constant time, whilst sorting events in the background. A pipelined approach optimizes the performance of this scheduler dispatcher algorithm when compared with others designs. Under CTask a task may be woken a maximum of 72 times per second. A "tick" is defined by waking the dispatcher every \texttt{timeint} seconds. The dispatcher increments a counter each time it is woken, and all event timings are performed relative to the value of this counter.

The system provides eight voice generators running in parallel whose output note events are sent to MIDI channels 0-7. The voices are represented on screen by eight boxes each of which contains text indicating whether the voice is audible or not, the MIDI on-velocity of note events generated by that voice, the contents of the space-grammar array \textit{P} and the length in ticks of the voice's sub-divisible duration. On start-up every voice's parameters are set to default values, the probabilities of the space-grammar and the Markov chain are set so that all productions/events are equally likely, \texttt{timeout} is set to a default and only voice one is made audible.

The user interacts with the system through four interconnected wait-and-shoot menus. Initially the system is at the top-level menu where the menu items are the eight voice-boxes and an item for moving to the main menu. Selecting any of the voice-boxes moves to the voice menu where choosing an item performs an action on the selected voice. The voice-menu items are for increasing or decreasing the on-velocity associated with the selected voice and making the voice audible or inaudible. Two further items on the voice menu are for moving to the pitch or rhythm vote menu. The vote menu's operation is described below. Choosing the main menu item from the top-level menu moves to that menu, where the user can increase or decrease the tempo of the generated music, save/load a snapshot to/from disk, exit from the program or restore the initial default parameters. Except for the top-level, each menu also has a quit item which returns control to the previously operational menu.

**Voting**

The user can elect to "vote" on the pitches or the rhythmic motives being generated by a particular voice. In both cases the user is presented with the vote menu from which the items \textit{Good/Long}, \textit{Bad/Short} and \textit{Abstain} may be chosen. On entering the vote menu, a short sequence generated by the voice will be heard. A user may then vote repeatedly until they are more satisfied with the voice's output. The \textit{Abstain} option just causes a new short sequence to be generated and played without manipulation of the voice's parameters.

Changes in the pitch selection are achieved by manipulating the Markov probability array values. The result indices chosen during the generation of the short sequence are recorded. These indices as subscripts into the array indicate each probability vector and the index chosen from that vector involved in the generation of the sequence. If the user chose the \textit{Good/Long} option, then the probability at the chosen index in the vector is increased, and all the other probabilities in the vector decreased proportionally. If the user chose the \textit{Bad/Short} option then the chosen-index probability is decreased, and the others increased.

An example can be seen in Figure 1 of voting from the default set-up. Each bar contains the generated short sequence, and the user's response is shown underneath. The user was attempting to favour ascending pitch sequences. The example is rather contrived for reasons of space — *"winning"* probabilities were made four times more probable, and only five events were possible.

---

1 This rate can be improved by up to a factor of four (to 288 times per second). However, this can result in interrupt overruns on slower computers.
The technique can work with larger numbers of events with less savage probability manipulations, but more votes are needed.

The rhythmic generation parameters $p$ are adjusted directly according to how the user votes. If the user votes Good/Long then the $p_0$ is increased and $p_1$ decreased, and if the user votes Bad/Short vice versa. If $p_0$ reaches 1 then the sub-divisible length of this voice is doubled, and both probabilities reset to 0.5 (the default). Similarly if $p_1$ reaches 1 then the sub-divisible length is halved and both probabilities reset to 0.5.

Conclusions

We are only just beginning to explore the prototype system. We would like to improve the voting mechanism so that the user could home in quickly on an area of the generator’s parameter-space, and then make more accurate local adjustments (rather like a genetic optimisation strategy). More interesting mappings from the Markov generated indices to musical events could be investigated.

References

