The Learning Agent Based Interactive Performance System

Michael Spicer [1], B.T.G. Tan [2], Chew Lim Tan [1]

[1]School of Computing, [2]Department of Physics, National University of Singapore
email: mspicer@sp.edu.sg

Abstract

An interactive performance system is described that utilizes the concept of intelligent agents to manage its complexity. Real-Time programming techniques adapted from computer games and audio DSP, as well as a basic machine learning technique are utilized to enable high-level user control.

1 Introduction

This paper describes an interactive performance system designed to easily enable the application of many musical formalisms to control various musical attributes, as well as providing intuitive high-level controls for the user/performer. The system uses an "artificial performer" paradigm, and is designed principally for controlling MIDI synthesizers. Interactive performance systems are inherently real-time, and techniques from other inherently real-time application areas, specifically computer games, and audio signal processing, are employed to achieve the desired results. The "artificial performer" paradigm lends itself to one of the current hot topics in Artificial Intelligence, that of Autonomous Intelligent Agents, and this concept is used to help manage the complexity of the system. The detail of the music produced is ultimately determined by the internal representations in the individual performer agents. The human performer achieves high-level control over the music by specifying particular targets for high-level musical behaviors, and the agents use a simple machine learning technique to collectively modify their internal representations to converge on these target behaviors.

2 Overall program structure

The structure of the program mirrors the structure of the music produced. It is assumed that the music produced follows a section-based form. In this system, a musical "piece" consists of a C++ container object that stores a collection of other C++ objects that represent each section of the piece. The base class for this, called CSystem, contains a set of default values and behaviours for the various attributes. The logic of the sequential ordering of each section is determined using a virtual function that implements a finite state machine. The details of this behaviour can be specified for each piece. This high level object also contains the target values of the musical attributes that can be controlled by the performer in real-time. The current system allows the user to specify:

- The textural density.
- The average pitch height.
- The average note duration.
- The degree of dissonance.
- The average dynamic level.

The CSystem object's main task involves controlling the high-level form of the piece, and contains all of the pieces sections. Each of the sections can contain as many agents (artificial performers) as required, subject to the physical limitations of the computer/synthesizer hardware. Because each section is a different C++ object, it is possible to use a completely different set of agents in each section of the piece. Another way of saying this, using the synthetic performer analogy, is that a different "orchestra" could play each section.

2.1 The synthetic performer analogy

Each section contains a collection of intelligent agents, which can be described as a group of synthetic performers. Each performer plays a single instrument. (There are exceptions to this, especially with regard to percussion, in which case the performer would play a sensibly selected small collection of instruments, such as kick drum and snare drum, a set of tom toms, a collection of different cymbals etc.) For a given performance piece, the human performer can control the number of different synthetic performers he would like to have. A band/orchestra analogy is used, with each performer implementing an algorithm that would be best suited to the function of part it is to play. The most important thing is that the collection of performers used can collectively operate fast enough to keep the system playing in time with the expected tempo of the music. Timing delays are not acceptable at all!

3 Basic agent design

This system makes use of various real-time game AI techniques to implement the virtual performers. We have chosen to use an agent design that is very open ended. In developing this agent, we considered the available percepts and actions that the agent can take, as well as any goals it may have, and the environment in which it operates. This can be summarized as:

- Percepts:
  - high-level control data, other agents in the environment, current time.
- Actions:
  - Select and play a particular note at a particular loudness on the synthesizer. Control continuous intensity of the note, add vibrato
- Goals:
  - Coherent pitch and rhythmic contour, appropriate consonance/dissonance within current harmonic environment, coherent form structure.
- Environment:
  - State of other agents and human performer(s), current time, synthesizer. This environment is dynamic and non-deterministic.

3.1 Agent Architecture

The basic agent design was conceived to allow a wide variety of formal compositional procedures to be utilized, as well as being conceptually simple for the composer/user. The idea of music being made up of distinct sections/patterns is inherent in the design. The information needed for the agent to play a note is: what note to play, how long to play it, and how loud to play it. we have given each of these parameters its own data representation. For pitch, duration and loudness contour (which helps create a sense of phrasing) we have used an approach derived from a common analog synthesis technique, commonly referred to as Sample & Hold, and the notes key velocity, which is largely responsible for creating the sense of meter, uses a simple condition action rule (this could be further refined in the future). The actual implementation of the agent is as a finite state machine, currently the standard technique used to create AI in computer games. In this case, the states are represented by the values at the current positions in the pitch lookup table, and transitions are determined by the current value of the duration array, as outlined below. This agent design is a type of learning agent, as its behavior is moderated by a 'critic', in the form of an error calculation, which is used to modify the values in its internal data representations. Russell and Norvig (2003) provide a good overview of approaches to agent design.

There are a number of distinct functional sections, encapsulated in C++ objects that make up an agent in this system. The agent has one object that encapsulates a MIDI "note on" event. This is the part of the agent that implements the actuators and implements the agent's actions, once it has decided what note to play. It contains the MIDI Channel, the current note number, and the current key velocity. There is also information about the current patch (timbre) that the agent plays. Other objects are involved with the representation of pitch and duration information, and are outlined below.

The main work of the agent during performance is carried out in the udpate() method, which is a polymorphic callback function. This is what would be referred to in the computer science literature as the "agent function" in this design. Specialized versions of the agent override this method in order to determine their specific behavior. The MIDI key velocity is calculated using a deterministic algorithm, which simply calculates the note on velocity, based on whether a new note occurs on a strong, medium or weak beat, as determined by traditional western musical practices.

3.2 Pitch and Duration Representation

One of the appealing features of the design of this system is that, although agents are essentially constructed as a rather unusual type of finite state machine, the human performer can easily understand the how the internal data produces the resulting music by thinking in the familiar terms of musical contours. The pitch, duration and dynamic representations can be thought of as implementing a "Sample and Hold" approach to creating melodic material, a common technique used with modular analog synthesizers since the 1960's. In analog synthesis terms, the idea is that a signal generator of some sort generates a continuously varying voltage (it is traditionally an oscillator or a noise generator), and this signal is fed into the Sample and Hold module. The S&H module has another input, a trigger input. When a trigger signal is applied to the trigger input, the S&H outputs the voltage that the continuous input signal was at that time. This voltage is maintained at the S&H output until it is triggered again.

We am using a digital implementation of this approach. Three C++ digital oscillator objects, of class we named COscil, are instantiated. These objects contain an array (sometimes refered to as a wavetable) consisting of a number (usually 96) of floating-point numbers between 0 and 1, plus associated manipulation methods. These wavetables are used by the agent to represent the pitch, duration and loudness contours. This one dimensional array representation of distinct musical parameters has several advantages. It is relatively small, intuitive, simple to implement and fast, making it ideal for real-time use. By representing contours in cyclically accessed arrays, we automatically capture the idea of sectional musical structures that can be repeated to form patterns. Many intuitive transformations may be directly applied to it, such as transposition, retrograde, inversion. The whole repertoire of audio signal processing can also be applied, such as filtering, biasing, scaling, waveshaping, delay, convolution with some other signal etc. As will be explained later, some of these DSP techniques are applied in the learning algorithm when the agents respond to the high level controls of the user. In addition, many of the game AI techniques, such as tracking/evasion and patterns, can be employed. Compositional formalisms, such as fractional noises, probability distributions, chaotic generators, as well as approaches coming from computer science, such as genetic algorithms, and goal based searches can be adapted to produce contours. Because the results of these generative processes are stored in a lookup table, the calculations themselves can take place during initialization (or before hand) and are freed of the real-time constraint. Loy (1989) provides a good overview of formal compositional techniques.

3.3 Generating the Note Data

In performance, a master clock that is tied to the tempo of the music drives all the oscillator objects within each agent. The current position indexes in the wavetable are updated every tick of this clock, and a comparison is made with the current time and the time the next note is scheduled to be produced. If it is time for a new note, the value at the pitch oscillator's current index position is used to generate the MIDI note number to be played. The agent has attributes that determine the lowest pitch allowed, and the pitch range (the difference between the highest and lowest notes the agent is allowed to play). The MIDI note number is produced by scaling the agents pitch range by the value in the wavetable at the current index, then adding the value of the lowest note allowed.

Note Number = bottomPitch + pitchRange * pitchOscillator.waveArray[currentSample]

This is further processed to map it into the correct musical scale. The more specialized agents may do additional processing of the raw MIDI note number value.

The representation of duration is very similar to the representation of pitch, in that a COscil object maintains the duration's for a particular time span, and samples it when a new note is to be generated. The duration of the new note is proportional to the magnitude of the value stored in the oscillators wavetable.

3.4 Chord Sequence Following

An extra input to the note number calculation for many of the agents comes from an extra percept from another agent whose task is to generate a sequence of chords. This idea is encapsulated in an agent class called CChordSequence. This specialized class is derived from CAgent, and is optimized to represent a series of chords over a particular time interval. It contains a number of methods for generating chord sequences, utilizing techniques such as patterns and Markov chains. An agent can simply add an offset derived from the current state of the CChordSequence object to the final raw MIDI note number calculation. An obvious extension to this is to build an agent that utilizes an incoming MIDI note data stream, from a live keyboard performance that does real-time chord recognition. Rowe (2001) contains such algorithms, and it is a simple matter to integrate these into the system.

3.5 Musical Knowledge in the Agent.

There is a certain amount of musical knowledge pre-programmed into the agent. A basic example of this it the calculation of note velocities using condition action rules. Quite a lot knowledge of musical processes are embedded in the methods implemented in the COscil object. For example, a motivic process such transposition of pitch corresponds to adding a bias to the values in the oscillators wave array. In terms of duration, adding a bias corresponds to uniformly increasing the note durations. Similarly inversion and retrograde are easily implemented on pitch and duration contours. It is also possible utilize these three processes to construct melodic forms, such as period and sentence forms within the pitch and duration dimensions.

The COscil class methods enable a large variety of pitch and duration contours to be generated, utilizing deterministic, stochastic, fractal and path finding techniques. Any computer graphics line generation technique can be implemented. This enables the possibility of generating interesting musical material with a large degree of intuitive control.

4 Response to user input

A delta-learning rule has been adopted in order to allow agents to modify their internal representations, so as to respond to the human performers wishes. At first glance, gradient descent methods of learning, which are normally associated with time-consuming tasks such as training neural networks, don't seem suitable for real-time interactive performance systems. It turns out that their long training time can be exploited to produce smooth gradual transitions from one state to another, that sound musically coherent.

In this system we use delta learning to allow the user to specify the target average pitch and duration values of all the agents pitch and duration contours. This is how the user/human performer achieves high-level control over the detail of the music produced. The user can specify the desired target average pitch, duration and loudness values, using a set of sliders on the screen (external controllers could also easily be used). At the end of each update, the current values of the average pitch and duration are calculated for all agents that are currently active. The current pitch and duration errors are then calculated, by subtracting the current calculated averages from the desired target values as determined by the user controls. During the next update these error values are used to modify the pitch and duration data in the agent. If the error is sufficiently large, an appropriate small change to the values is calculated by multiplying the overall error by a learning rate I.E. delta = error * learning rate. This delta value is then used to either scale the values or add a bias to the values, depending on which learning mode is chosen.

The mechanisms for providing the high level control of the textural density are far simpler. They both essentially use a threshold approach. In the case of textural density, each agent has a threshold value, on a scale of 0 to 1.0, and if the global current textural density value is above the threshold, then the agent will play, otherwise it will be silent. The user sets the target for the global threshold value. Each clock cycle, the current actual textural density value is calculated, by calculating the proportion of agents active. This value is used to calculate a delta for the current global textural density. The similar approach is used to control the degree of dissonance.

5 Specialized Agents

A number of specialized agent performers have been developed, each optimized to play a specific musical role. Many of the specialized agents further constrain the choice of notes played by the virtual performers. An example of this would be an agent that "listens" to the chord sequence agent, and only plays notes that are contained with the current chord.

Another useful set of agents that place constraints on the resulting MIDI note numbers produced have been developed to play percussion parts. These limit the notes numbers generated to as to only play the appropriate percussion timbres continued in the General MIDI drum key layout.

An interesting class of agents is those that utilize some one of the classic game "AI" techniques, Tracking, and its compliment Evasion. In this situation the agent has a target agent, and it essentially follows the targets pitch and/or duration behavior or avoids it. There are various extra attributes that control useful musical behavior.

6 A Musical Example

Several compositions have been produced using this system for several different contexts. It is often used in conjunction with live performance of acoustic or electronic instruments. The configuration of the system is usually carefully prepared in advance, so as to provide the appropriate high-level control over the musical detail (pitch, note rate etc.) as well as large scale form. One piece that demonstrates some of the capabilities of the system is "South West Monsoon". This piece is designed as a stand-alone piece that can be performed on any Microsoft Windows based Multimedia PC. It utilizes the General MIDI sound set and is distributed as a Win32 executable. The agents in each section make up a collective ensemble of 16 performers. These virtual performers are broken into functional groups, similar to sections in an orchestra. The breakdown of the performers includes players creating melodic lines, a bass line, arpeggio patterns, pads, and pitched and percussive rhythm patterns.

The high-level form of the piece has four sections, all of which are instances of the same C++ class. Random choices are made during the initialization phase, so each of the agents in the sections will have different timbres and contours. The chord sequence for each section is generated using a 1st order Markov chain to create “strong” chord progressions, adhering to the rules of tonal harmony. The overall form is also generated using a 1st order Makov chain. This determines the default order of playback of the four sections. The high-level controls that set the target values for the textural density, mean Pitch, mean duration, mean intensity and degree of dissonance are controlled by instances of the COscil class that is used in the agents. The period of the oscillators is set to be the total duration of the piece, so the waveforms dictate the default evolution of the piece, which has been preset by the "composer".

At any time the human performer can play the system like an instrument, modifying the target values, so that the program will be directed towards creating the user’s desired musical effect. The user can go into completely into manual control mode, and can override the overall form, selecting which section to play next, using the arrow keys on the keyboard. At all times, the human performer can alter the overall balance using a virtual mixer.

Conclusion

The concept of utilizing simple game AI techniques, encapsulated in a learning agent design, has been demonstrated to be a useful strategy for build interactive performance systems. This, coupled with the approach of using various contours for representing musical data, enables very intuitive control of the agent’s behavior by the human performer.

References

Loy, G.. 1989. “Composing with Computers- a Survey of Some Compositional Formalisms and Music Programming Languages.” *Current Directions in Computer Music Research.* Cambridge, Massachusetts: MIT Press. Pp. 291-396. ed. Mathews, M. V., and J.R.Pierce

Rowe, R. 2001. *Machine Musicianship*. Cambridge, Massachusetts: MIT Press.

Russell, S. J.,and P. Norvig. 2003. *Artificial Intelligence A Modern Approach.* Upper Saddle River, New Jersey: Prentice Hall.

Spicer, M.J. 2003.*An Interactive Performance System.* M.Sc. thesis. unpublished