Emitting Grains of Learning with Pd

Judy Franklin
Smith College
Computer Science Department
Northampton, MA 01063
jfrankli@scinix.smith.edu
http://www.cs.smith.edu/~jfrankli

Jacob Last
University of Massachusetts
Mathematics Department
Amherst, MA 01002
jacoblast@gmail.com

January 26, 2006

Abstract

We present work in granular synthesis that combines spectral analysis with reinforcement learning, a type of machine learning. We describe an audio granular synthesis generator designed specifically to perform granular synthesis, with programmable controllers that can be accessed by reinforcement learning controllers. The movement of the controllers affects the sound, which is analyzed to produce a vallue called the reinforcement The reinforcement values depend on the spectral centroid of the grains, as well as on a measure of the centroid spread. These values are used to learn how to affect future controls. We have used reinforcment learning in both a conventional machine learning way, as well as in a more artistic, experimental way. The work was carried out in a sound and music software environment called Pure Data. Some implementation details are also included.

1 Background

We are interested in combining machine learning algorithms with granular synthesis. Granular synthesis is the building of sound objects using brief microacoustic events called grains of sounds. A grain of sound may be a pure sine wave passed through an amplitude (volume) envelope, and lasting barely as long as the threshold of human auditory perception. Sound may be synthesized and shaped from such grains by varying the density of grains per second, the frequency range of the sine waves used in synthesis, and the durations of individual grains, among other parameters. These parameters may be time-varying and can be set as means in a gaussian distribution or as parameters in some other kind of random distribution. Harmonics may be added to the sine wave that is oscillated to generate each grain, or other sounds may be sampled, and those waveforms oscillated to generate each grain. Granular synthesis has a history, both in electronic music as well as in computer music. Interested readers will find the Roads text [11] enlightening. Many grains of sound, overlapping in time, with various durations can sound like waterfalls, dripping faucets, or thousands of chicks chirping. It is difficult to characterize the sound in words, and it cannot be readily visualized in a scientific way.

In the past we have manipulated sound grain frequencies [7] using common computer science algorithms such as searches and sorts, implemented in music and sound software called RTCmix [12]. These grains were of fixed duration. The work in this paper goes beyond our previous efforts, with

the goal of varying many parameters of grains of sound to generate sounds that have some global characteristics, using reinforcement learning.

2 The Granular Synthesizer

We created a tailored granular synthesizer patch, called GranWave, that uses wavetable oscillators to generate grains. GranWave is implemented in a graphical music programming environment called Pure Data (PD) [10], as shown in Figure 1. There are many ways to vary grain generation and

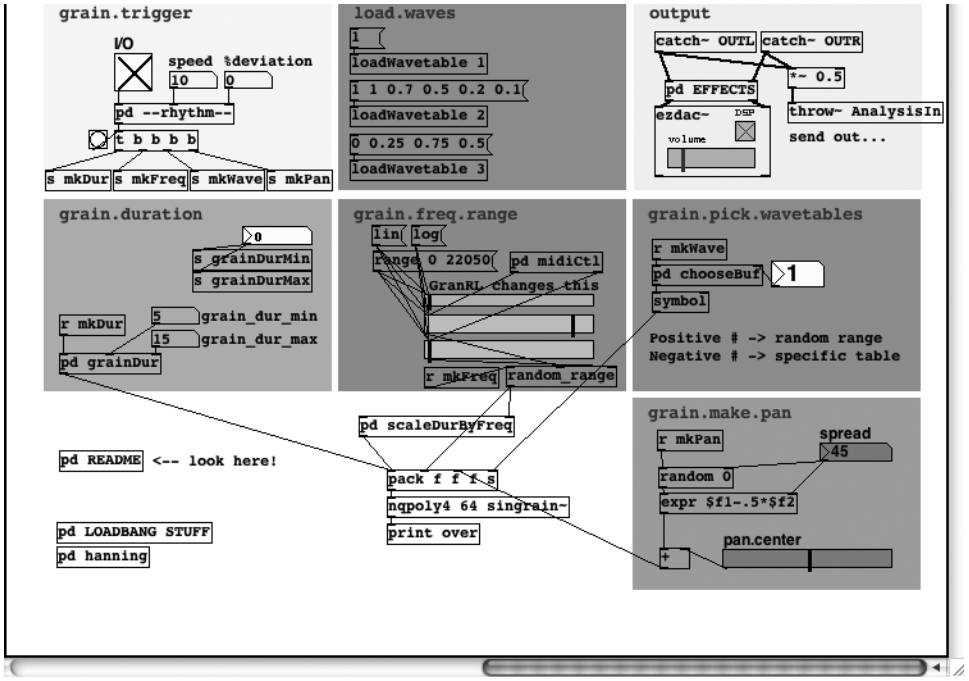

Figure 1: The PD file that holds the granular synthesis engine and the mechanisms that send audio signals to the analyzer, and the slider controllers that are controlled by reinforcement learning.

GranWave makes many variations possible. In our first experiments with reinforcement learning, we set most of these variables to fixed default values that are consistent across experiments. To start grain generation, the user clicks grain.trigger, which generates a " generation rhythm" with a PERIOD of "speed" ms (not frequency), and some randomness set by the percentage of desired deviation (%deviation). We initialize with 10ms speed and 0 percent deviation. grain.trigger triggers the patches that set the other grain parameters, through sent messages: mkDur, mkFreq, mkWave, and mkPan. mkDur triggers the actual grain duration to be set, with some random variation. The user or the reinforcement learning agent can control the range of variation. The default values of the range are 5 msec minimum and 15 msec maximum duration. The mkWave and mkPan messages trigger the choice of wavetable (1,2, or 3 depending on desired harmonics), and panning. Panning is set by default to have a spread of 45, with random variation enabled, and the pan center strictly in the center. The chosen grain wavetable defaults to 1, the pure sinusoid.

mkFreq triggers randomrange. Randomrange receives as input the values of the two lower horizontal sliders as the bounds on the range of frequencies within which grains will be generated. The middle slider sets the minimum range value, and the lowest slider sets the maximum range value. The

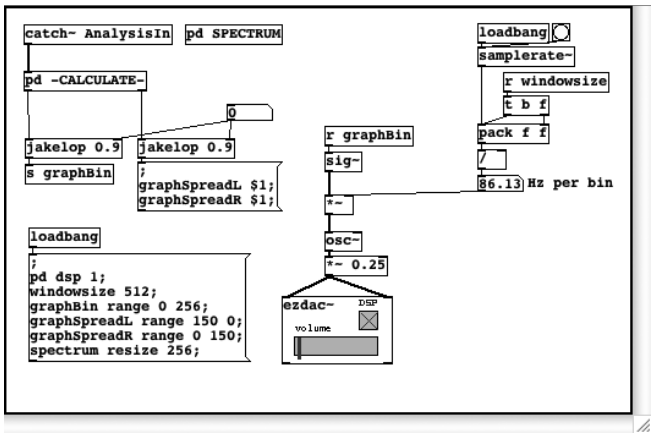

Figure 2: The PD file that holds the spectral analysis patch and mechanisms for receiving audio information and sending analysis results. It contains a means to listen to the spectral centrum as well.

top slider is used to change both of the lower sliders at once, making the range 1, and completely deterministic. It is the lower two sliders that are moved by reinforcement learning agents to affect the range of frequencies of grains.

In the following description, we use PD's syntax of the tilda(˜) character appended to the operation name to indicate it is a signal level operation. The grain generator is built to work with the nqpoly4 [8] polyphonic voice manager patch for PD. nqpoly4 accepts its parameters as a list in the first inlet. Upon loading, nqpoly loads the singrain˜ abstraction that is the producer of each single voice. The parameter list for nqpoly4 is: grain start index (ms), grain duration (ms), grain playback speed scaling, pan (from -45 to 45 degrees) and wavetable source table number. Upon receiving a start time, nqpoly4 throw˜s the grain audio output to OUTL˜ and OUTR˜ summing busses. nqpoly has one outlet that bangs when the voice is done playing the current grain (that tells nqpoly4 that the voice is free).

One of our evaluation criteria is a function of spectral centroids of windows of grain clouds. The spectral centroid is an amplitude weighted average of frequency bins, that corresponds to the preceptual brightness of the sound [1]. The OUTL˜ and OUTR˜ signals are caught to produce the audio, and re-thrown to the spectral centroid analyzer.

In the analyzer, called pd - calculate - as shown in Figure 1, the signal is put through a Hanning Window, and then run through a Fast Fourier-Transform (FFT) patch to find the spectral characteristics of one frame of granular sound. The number of frequency bins used in the FFT analysis is variable, but we have set it to 256 bins.

The spectral centroid, $\tilde{i}$, is implemented in Python, and made available to PD as an external using pyext [6]. $\tilde{i}$ is computed as:

$$\tilde{i} = \frac{\Sigma_{k=1}^{N-1} kX[k]}{\Sigma_{k=1}^{N-1} X[k]} \tag{1}$$

where $X[k]$ is the magnitude of bin k and N is the FFT length [9].

The spread, implemented in the same python script, is a measurement that we liken to the "bandwidth" of the grain cloud and is defined as:

$$Spread(x) = \sqrt{\frac{1}{N}\Sigma_{i=0}^{N-1} x[i](i-\tilde{i})^2} \tag{2}$$

where $x = x_0, ..., x_{N-1}$ is the signal vector of length N, and $\tilde{i}$ is the spectral centroid (in units of FFT bins). The idea is to use the distance of each bin from the centroid bin, weighted by the amplitude in that bin. This way, the more high amplitude bins that are far away from the centroid bin, the higher the total measure will be. Both the spectral centroid and the resulting spread vary with the transposition range of GranWave, so are useful in evaluating changes in transposition.

One note about this "spread" measurement is that it is linear over the FFT bins, and therefore does not correspond well to what we hear as the subjective pitch spread. A "constant Q" spread [2] may give values that make sense with respect to pitch rather than frequency, if that is desirable.

We wrote the reinforcement learning (RL) algorithm in C++, modifying code from earlier work [4]. The RL algorithm is encapsulated in an interface class that inherits the Flext class [5]. Flext enables a programmer to write externals that can be used as patches within PD. The external is called GrControl, and implements the Sarsa(λ) reinforcement learning algorithm [13].

2.1 Using the Sarsa(λ) algorithm

The tabular Sarsa(λ) algorithm[13] uses a discrete state table containing one Sarsa agent per state. Associated with each agent is some number of actions. An agent contains two sets of numerical values called Q-values and eligibility traces, respectively. One Q-value and one eligibility trace is associated with each action, in every state. Each agent uses its set of Q-values to choose an action when the system is in its state. The eligibility trace $e_t(s, a)$ is used to update the Q-value according to how instrumental the associated action was in acquiring the most recent reinforcement.

Each agent chooses its action according to an ϵ-greedy policy. When s is the current state, action a is chosen if it has the maximum Q-value (the greedy part) as shown in equation (3). Before an action is taken, a random number between 0 and 1 is generated. If it is smaller than a value called ϵ, then a random action is taken instead.

$$a = \begin{cases} \text{arg max} Q(s, a) & \text{if rand}(0, 1) > \epsilon, \\ \text{a random action} & \text{otherwise.} \end{cases} \tag{3}$$

The value of ϵ determines how much agent exploration takes place. We have used $\epsilon = 0.1$.

At the beginning of a learning experiment, the set of Q-values for each agent is initialized to a small random value, and each agent's eligibility traces are initialized to zero. At time step t, the expected Q-value (at time step $t + 1$) is updated for all states and actions (s, a) by the equation

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha \delta_t e_t(s, a), \tag{4}$$

δ_t is the difference between expected reward $r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1})$, the immediate reward plus the Q value of the next action discounted by $\gamma < 1$, and the actual value $Q_t(s_t, a_t)$.

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t). \tag{5}$$

In our granular synthesis application, r_{t+1} is 0 unless the frequency is in the specified range, and then it is 1. The eligibility trace $e_t(s, a)$ is increased by 1 for the current state action pair. All traces are decreased by γ and λ. The higher λ is, the longer a state-action pair's actions are deemed to affect the reinforcement value. This computation provides a decaying trace of eligibility for updating each Q value:

$$e_t(s, a) = \begin{cases} \gamma \lambda e_{t-1}(s, a) + 1 & \text{if } s = s_t \text{ and } a = a_t, \\ \gamma \lambda e_{t-1}(s, a) & \text{otherwise.} \end{cases} \tag{6}$$

The amount a Q-value is updated depends on the corresponding eligibility trace value as well as the difference, positive or negative, between the expected and actual reward. The Tabular SARSA algorithm is shown in pseudocode in Figure 3.

```
1 Initialize Q(s, a) arbitrarily and e(s, a)=0, for all s, a
2 Repeat (for each episode):
3     Initialize s, a
4     Repeat (for each step of episode):
5         Take action a, observe r, s'
6         Choose a' from s' using Q-policy (ε-greedy)
7         δ ← r + γQ(s', a') − Q(s, a)
8         e(s, a) ← e(s, a) + 1
9         For all s, a:
10            Q(s, a) ← Q(s, a) + αδe(s, a)
11            e(s, a) ← γλe(s, a)
12        s ← s'; a ← a'
13    until s is terminal
```

Figure 3: Pseudo-code for Sarsa(λ) algorithm. In this work, an episode is one experiment with a bound on the number of total actions taken.

3 First Experiments

In our first experiments, we used RL agents to control one horizontal slider only and the number of states was set to 100. Each agent has the same finite set of actions 0,1,2,3, and 4, correrresponding to sliding the controller -5, -1, 0, +1, and +5 spaces. We found that if the actions are only +1,0,-1, a state space of 100 values is difficult to explore efficiently, so two more actions, +5 and -5, were added to enable the RL agents more flexibility in exploration. As a result of the slider change, the frequencies of the grains change, affecting the spectral centroid and spread, that in turn affect the reinforcement value (R). The state changes to the new position of the slider.

Each of the 100 states corresponds to a 220.5Hz wide subband of the frequency spectrum. Consequently if an agent action is +1, then it moves the horizontal slider (and so its own state) up to the next frequency bin (adding on 220.5Hz). If the action is -5, the horizontal slider moves down 5 frequency bins, adding -1102.5Hz. The action space is simply increasing or decreasing the grain transposition setting in GranWave. A positive (R=1) reward is given if the spectral centroid at a given time step is within the desired range of values. A reward of 0 is given if the centroid at a given time step is outside of that range.

The RL agents are encoded in the GRControl patch, which first gets the state (the current setting of the transposition control), followed by the reward. It outputs one of five actions, which are adjustments of 0,+1,-1,+5,or -5 to the transposition control. The process is then repeated. This is a fairly simple experiment, and the RL agents are always able to find a state that makes the granular sound fall into the target frequency-bin range. The value of ϵ is never decreased, so exploration continues, making the state change. The variation in states due to the agents' exploration ($\epsilon = .1$) yields interesting variations in sound. In a more typical application of the Sarsa algorithm, one might be concerned

with how to algorithmically recognize success and to decrease ϵ to prevent future exploration. We view continued exploration by the algorithm as a way to vary the generated sound.

4 Second experiments

In our second set of experiments, we enabled the GrControl patch to control two horizontal sliders, i.e. setting the min and max of the frequency range as discussed in Section 2. The Sarsa(λ) algorithm would work best if we created a 100x100 square state space since the position of one slider affects how the other slider alters the sound output. This size would tax the system, whose output is heard in real-time. And if another controller is added, we would need a 3-D state space and so on. We decided to use a state space that is just the two slider spaces juxtaposed. That means now there are 200 states, 100 per horizontal slider. The agent actions 0 through 4 cause the current slider to move -5, -1, 0, 1, or 5 steps respectively, as before. But now, an additional action, action 5, causes these actions to be applied to the other slider. After this occurs, the agents in the other slider's state space will move that slider's position, until one of them chooses action 5 and causes the slider switch again. This requires a high level 2-valued state that switches, s0 to s1 or s1 to s0. This configuration is shown in Figure 4.

The RL controller does not receive the high-level slider state as input. This makes the task difficult for RL because the reward from action selection is not consistent. The affect of a single slider's action depends on the position of the other slider as well, and this is not available to the agent that is changing the one slider position. Since slider positions correspond to state values, this is an instance of perceptual aliasing [3]. GRControl is generally successful in meeting the spectral criteria and in achieving continuous reward values of 1, although it can get stuck in a strange state. We used the spread measurement in equation (2) to reward the RL agents if they take actions that may enable them to leave an undesirable state. The reward criteria are now:
For frequency bin range [10, 40],

if spectral centroid is in range,	R = 1
else if spread(t) <= spread(t-1) - 3,	R = 1
else,	R = 0.

The basis for this is that if the spectral centroid is not in range, then if spread(t) <= spread(t-1) - 3 the agents are moving in the right direction, toward the spectral centroid. We found this criterion to produce better results than the simpler version that depends only on the spectral centrum measurement. Better results can mean changing states in hopes of moving to one that produces the correct frequency range. Once again, the exploration involved in moving between states is reflected aurally in the generated grains. The learning process itself produces an attractive soundscape.

In another discovery, we found that the RL agents with perceptual aliasing are more successful in keeping the frequency bin in the correct range when the wavetable to be oscillated contains harmonics. Figure 5 is a typical graph of number of steps that R = 1 before it becomes 0, when the wavetable holds one period of a pure sine wave. When harmonics, integer multiples of the fundamental frequency, are added, in some experiments, the performance is much better. The specific harmonics are 1,1,0.7, 0.5, 0.2, and 0.1. Figure 6 shows the same type of graph of number of steps that R is 1, consecutively, but for a wavetable containing the sine wave plus harmonics.

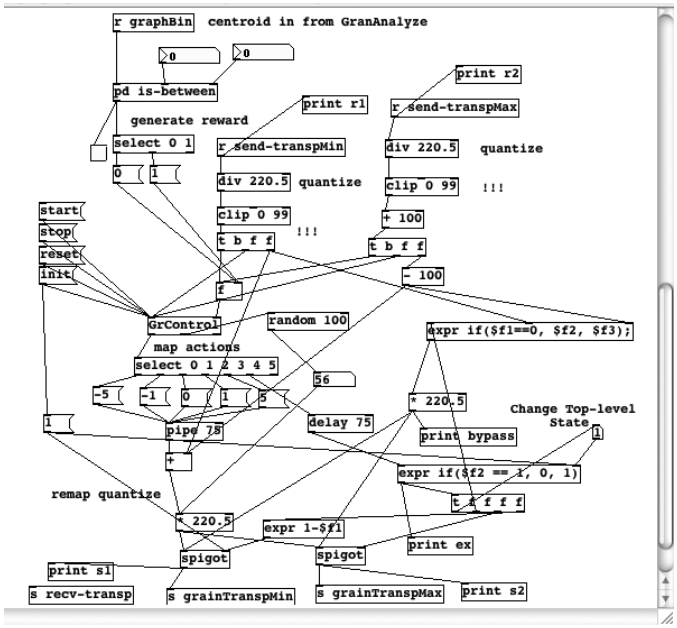

Figure 4: The PD file that holds the Reinforcement Learning patch, GranRL and the mechanisms that generate the reinforcement signal (at top), interpret the actions, and keep track of the low level and high level states

5 Discussion

We have determined that the RL agents can learn to control a simple controller of grain transposition with real-time granular sound generation. We are now poised to explore the intermingling of RL with other granular synthesis parameters. In the future, these would be good as additional reinforcement learning controls.

The addition of further controls in the future may require us to use techniques that explicitly address the perceptual aliasing problem. For now, the perceptual aliasing causes variation in grain transposition that adds to the color of the sound.

Part of this work is an empirical study of how RL behaves when perceptual aliasing is present, and how RL can be coaxed to succeed by adjusting the reinforcement criteria. The other part is an artistic endeavor. We hoped that the learning process itself would generate interesting musical material. We anticipate that by adding more controls and other kinds of evaluation criteria we can generate profound and novel sounds. In fact, we embrace the perceptual aliasing somewhat because it induces dissimilitude in generated clouds of grains.

We are currently experimenting with using the reinforcement value R to change other grain parameters such as minimum duration or %deviation in grain generation density. For example, Figure 7 shows a plot of the number of steps until R = 0, when we use the value of R itself to adjust another control. We constructed a patch that executes whenever R = 1. Then 1 is added to a summing box and the new sum and sent to the %deviation control. If 100% deviation is reached, the summing box is reset to 0. The causes variation in how constant the number of grains generated per second is. When %deviation is 0, it is always one every 10 ms. But the higher the %deviation, the more variability in speed of generation. The episode in Figure 7 shows our best results yet for maintaining a frequently nonzero R value.

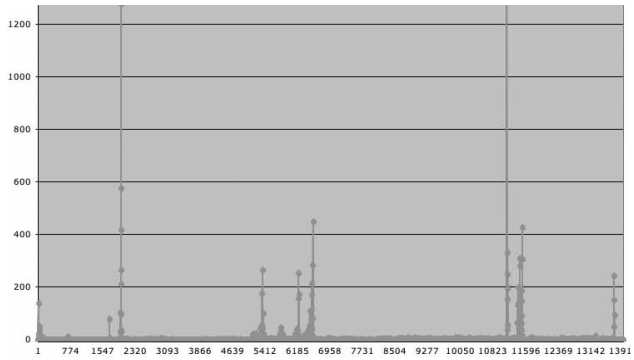

Figure 5: Number of steps until R = 0, when the wavetable is a pure sinusoid. y-axis is in steps, x-axis is in sub-episodes, where one sub-episode corresponds to R having the value of 1 the whole time

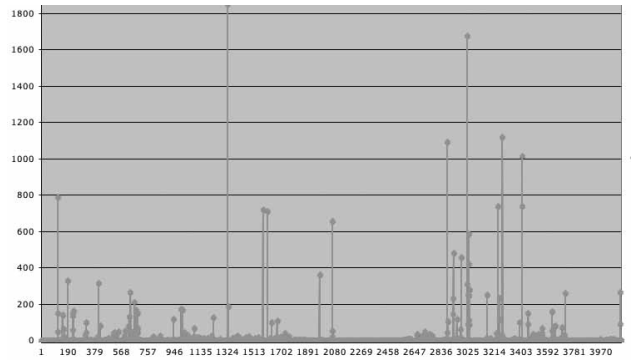

Figure 6: Number of steps until R = 0, when the wavetable is a sinusoid with harmonics added. Since the y-axis is the number of consecutive steps for which R is 1, the shorter span of the x-axis is another indication of a more successful experiment.

We imagine using disjoint sets of learning agents each with its own reinforcement criteria and set of parameters to control. We are also working on human musician interactions with this system.

Acknowledgements

We would like to acknowledge the work of Miller Puckette and Thomas Grill in developing Pure Data, flext, and pyext. We also thank Melody Donoso for her work in encoding the Sarsa algorithm. This material is based upon work supported by the National Science Foundation under Grant No. IIS-0222541 and by Smith College. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

References

[1] Beauchamp, J. 1982. Synthesis by spectral amplitude and ’brightness’ matching of analyzed musical instrument tones. *J. Audio Eng. Soc.* 30(6):396–406.

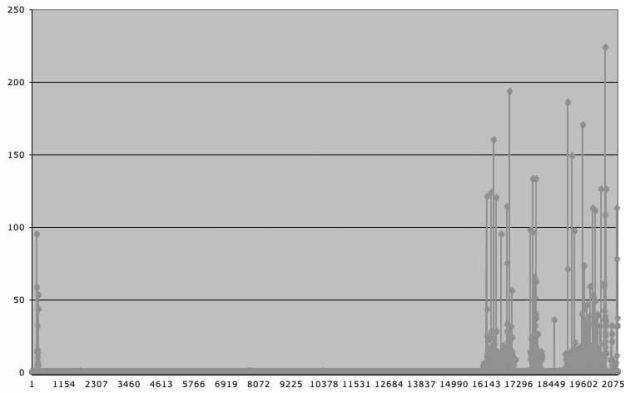

Figure 7: Number of steps until R = 0, when the wavetable is a sinusoid with harmonics added, and with R = 1 changing the %deviation as described in the text.

[2] Brown, J. C. 1991. Calculation of a constant q spectral transform. *J. Acoust. Soc. Am.* 89(1):425–434.

[3] Chrisman, L. 1992. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of AAAI92*. Menlo Park, CA.: AAAI Press.

[4] Donoso, M. 2004. *Improvising Rhythm Using Reinforcement Learning*. Northampton, MA: Senior Honors Thesis, Smith College.

[5] Grill, T. 2001-2005. flext - c++ layer for max/msp and pd (pure data) externals. In *Web URL*. http://grrrr.org/ext/flext/.

[6] Grill, T. 2002-2005. py/pyext - python script objects for pd and maxmsp. In *Web URL*. http://grrrr.org/ext/py/.

[7] J. Franklin, E. Laverty, S. N. 2003. Pitching computer algorithms. In *Proc. of the Ninth Biennial Symposium on Arts and Technology*. New London, CT: Connecticut College.

[8] 2005. nqpoly4, polyphonic voice patch. In *Web URL*. http://pix.test.at/pd/nqpoly/nqpoly4.html

[9] Park, T. H. 2000. *Salient Feature Extraction of Musical Instrument Signals, Masters Thesis*. Salient Feature Extraction of Musical Instrument Signals: Dartmouth College.

[10] Puckette, M. 2005. Pure data (pd). In *Web URL*. http://www-crca.ucsd.edu/ msp/software.html.

[11] Roads, C. 2001. *Microsound*. Cambridge MA: MIT Press.

[12] 2005. Rtcmix (real-time cmix) music software written in c/c++. In *Web URL*. http://www.music.columbia.edu/cmix/

[13] Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning*. Cambridge MA: MIT Press.