

Live Interactive Music Performance through the Internet

by

Charles Wei-Ting Tang

B.M., Piano Performance, The Juilliard School (1992)

M.M., Piano Performance, The Juilliard School (1994)

B.S., Computer Science, Columbia University (1994)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Technology

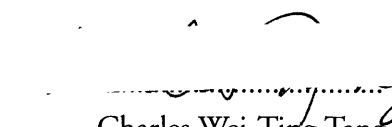
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1996

© Massachusetts Institute of Technology 1996. All rights reserved.

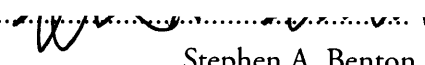
Signature of Author


Charles Wei-Ting Tang
Program in Media Arts and Sciences
August 9, 1996

Certified By


Tod Machover
Associate Professor of Music and Media
Program in Media Arts and Sciences

Accepted By


Stephen A. Benton
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
AUG 21 1996

Live Interactive Music Performance through the Internet

by

Charles Wei-Ting Tang

Submitted to the Program in Media Arts and Sciences
School of Architecture and Planning
on August 9, 1996
in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences at the
Massachusetts Institute of Technology

Abstract

Various models of interactive music systems have been proposed and implemented since the mid-1980's. To date, however, there exists no interactive music system which supports participation of remote users in real-time. With the recent introduction of RealAudio and the new Java programming language, real-time audio transmission and animation on the World-Wide Web (WWW) finally become possible. The proposed project takes advantage of the Internet to allow remote participation. Using the WWW, as well as RealAudio and Java applets, three models of networked interactive music systems will be designed, implemented, and evaluated. This will be followed by an in-depth discussion of the three central research areas of this thesis. They are: 1) design issues such as number of participants allowed, musicianship of the participants, role of composer / conductor, and musical elements or parameters to be controlled by users; 2) presentation issues such as technical difficulties to overcome, concatenation of materials collected, and structure; and 3) usability of such systems in live concert performances - e.g., the premiere of the Brain Opera in Lincoln Center, New York.

Thesis Advisor: Tod Machover

Title: Associate Professor of Music and Media

Program in Media Arts and Sciences

Live
Interactive
Music Performance
through the
Internet

by

Charles Wei-Ting Tang

Thesis Readers:

Reader
Michael Hawley
Assistant Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Reader
Mitchel Resnick
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences

...

ACKNOWLEDGMENTS

I am greatly indebted to my advisor, Prof. Tod Machover, for his support and guidance throughout my stay here at the Media Laboratory.

I would also like to thank:

Prof. Michael Hawley, for his guidance and patience, and Prof. Mitchel Resnick, for offering his valuable perspectives.

The *Performance* and the *Projects in Media & Music* classes.

Teresa Marrin, for her support, friendship, and enthusiasm.

Alvin Fu, for his love of music, and being a passionate Rachmaninoff and *Honglou Meng* fan.

John Yu, Josh Strickon, Jeanne Yu, and Roberto De Prisco, for their collaboration and assistance.

Eric Métois, for his expert advice and kindness.

Manish Tuteja, Christina Manolatu, and Bernd Schoner, for being great office-mates and friends.

Linda Peterson, for her help and advice, and Susan Bottari, for taking care of business.

My parents and family here and above

And of course *Avalokitesvara*, without whom none of anything would be possible.

TABLE OF CONTENTS

1. Introduction

- 1.1 Motivation
- 1.2 The World Wide Web
- 1.3 Java and *RealAudio*
- 1.4 Theoretical Framework
- 1.5 A Very Quick Tour of the Brain Opera

2. ITX – The *Expressive Piano Tutor*

- 2.1 Description of the Project
- 2.2 How Does It All Work?
 - 2.2.1 Buttons
 - 2.2.2 How the Graphing Routine Works
 - 2.2.3 How the Amplitude Curve Works
 - 2.2.4 How the Teach Algorithm Works
 - 2.2.4.1 Philosophy
 - 2.2.4.2 The “Flow”
 - 2.2.4.3 Determining “Places of Musical Interest”
 - 2.2.4.4 Good and Bad Exaggerators
 - 2.2.5 How the Harmony Analyzer Works
 - 2.2.6 How the Tempo Analyzer Works
 - 2.2.7 How the Attribute Field Is Used
- 2.3 Parametrization of Music

3. Interactive Web Systems: Overview and Basic Design Issues

- 3.1 Basic Design Issues
- 3.2 Limitations
 - 3.2.1 Java and MIDI-lessness
 - 3.2.2 Speed

4. Simple Yet Effective Games: *Sound Memory* and *Bouncing Balls*

- 4.1 *Sound Memory*
 - 4.1.1 Making a Match
 - 4.1.2 The Role of Music in *Sound Memory*
 - 4.1.3 Improving *Sound Memory*

4.2 *Bouncing Balls*

4.2.1 From Total Randomness to Rhythmic Perfection

4.2.2 Multiple-Players

4.2.2.1 *WebStar* and *StarClient*

4.2.2.2 Running Things Locally

5. **Creating and Morphing a Music Stream**

5.1 *Draw Music*

5.1.1 General Description of the Game

5.2.2 Preset Modes and Quantization

5.2 The Role of Music in *Draw Music*

5.3 Extending *Draw Music*: Other Possibilities

5.3.1 Sequential Compounded-Morphing

5.3.2 Sequential Segmented-Morphing: The *William Tell* Experience

5.3.3 Theme-and-Variations

5.3.4 Multiple-Players

6. ***The Internet Pong Game***

6.1 Overview of the Game

6.2 The Referee

6.2.1 Sending Messages

6.2.2 Bouncing Angle

6.3 Technical Limitations

6.4 MIDI, not .au!

6.5 Linking *The Internet Pong Game* to *Sharle*: John Yu's Computer Music Generator

6.6 I Just Want to Win!

6.7 A Big Happy Family

6.8 Possible Expansions

6.8.1 More Parameters

6.8.2 *Für Elise*: Linking to Famous Pieces

7. **Broadcasting Live Music with *RealAudio* and *StreamWorks***

7.1 Problems in Delivering Time-based Information over the Internet

7.2 *RealAudio*

7.3 *StreamWorks*

7.4 Comparing Sound Qualities – *RealAudio* versus *StreamWorks*

8. Bigger than Life: the *Internet Hyperinstrument*

8.1 About the *Palette* . . .

8.2 About the Controls . . .

8.3 The Role of Music in *The Palette*

9. Usability of Games in Concert Performances

9.1 *Bouncing Balls*

9.2 *Draw Music*

9.3 *The Internet Pong Game* and *Internet Hyperinstrument*

10. Afterthoughts

Appendix

Bibliography

1. INTRODUCTION

1.1 Motivation

Recently, the Hyperinstruments / Opera of the Future group, led by Prof. Tod Machover, has launched a new interactive performance project called the Brain Opera. An important component of the Brain Opera will be the participation of people around the world through the Internet. In the advertisement of the Brain Opera homepage, visitors are encouraged to send in sound materials to be used in the event. Moreover, a true interactive music system will be designed and implemented so that users who simultaneously log on to the Brain Opera site can all participate and collaborate in the making of a live music performance. In the current design of the Brain Opera, there is a five-to-ten-minute section where the materials presented will almost exclusively come from the events taking place at the Brain Opera website at that particular time. Creating and exploring various models of live, interactive music applications thus become the focus of this project.

1.2 The World Wide Web

Due to its user-friendliness and world-wide popularity, the World-Wide Web – a.k.a. “WWW,” “W3,” and “the Web” – becomes the obvious environment-of-choice where the proposed interactive music system should be built.

The first-generation Web browsers such as Mosaic and Netscape have often been portrayed as “Internet interactivity enablers.” If one examines these browsers more closely, however, it becomes apparent that they only provide an illusion of interactivity. By using these browsers, the user is able to retrieve remotely linked data of various formats (e.g., text, sound, image), download them, and display them on the screen. Although this process is automated and can be performed with only a button click, the fact remains that these Web browsers are but “nice,” convenient data-fetching and displaying devices. And because of the way these browsers are designed, two-way communication over the Web is difficult and slow.

As the teaching assistant of Prof. Tod Machover's *Performance* class and *Projects in Media & Music* class, the students and I have collectively explored perhaps hundreds of musical and game links during the course of the semester. Most of the Web sites we have visited are "information stations." Some are very well laid-out and structured, and some quite poor. In almost all cases, however, the interactive aspect of the sites consists of nothing more than prompting the user for either comments and suggestions or for completing a product-ordering form!

1.3 Java and *RealAudio*

Recently, with the release of the Java programming language and the Hot Java browser, real-time interaction on the Web finally becomes possible. Java is an object-oriented, architecture-neutral, multithreaded, and dynamic programming language [JavaLang, 95]. It supports programming for the Internet in the form of platform-independent Java applets (mini-applications). A Java-enabled browser can thus dynamically download segments of code that are executed right there on the client machine [JavaLang, 95]. In other words, a user can now interact with the applets in real-time. Installation of special software will no longer be necessary.

Another recent technological breakthrough is *RealAudio 2.0*. With the *RealAudio Player*, a user can hear a sound file (encoded by the *RealAudio Encoder*) almost immediately at the touch of a button. One no longer needs to sit for minutes and wait for the downloading process to complete before the content of a sound file can be heard. The user can also navigate through a sound file and select sections to listen to, much like the traditional CD player. Most importantly, using the *RealAudio* client-servers, real-time sound broadcasting over the Internet is finally achievable. *RealAudio* is independent of bandwidth, so it sounds the same on every machine. Currently *RealAudio* sounds slightly better than AM radio, but as its compression algorithms become more efficient, improved sound quality will be made available for lower-end machines. Other Internet streaming software, such as Xing's *StreamWorks*, are being developed to send audio-visual signals over the Internet. *StreamWorks* also takes

advantage of a client's bandwidth, in effect by having a faster connection a user can receive better quality audio and video. The Brain Opera team has been exploring these possibilities as well as other applications that are currently in development.

Equipped with the Web, the Java language, the new Java-enabled Web browsers, and *RealAudio*, the project attempts to create three models of true interactive music systems for the Internet. After they are designed, implemented, and evaluated, one system (or multiple systems) will be used in the actual Brain Opera performance.

1.4 Theoretical Framework

Compressing and sending musical information at the gestural level is by far cheaper and more efficient than transmitting actual audio signals. According to Michael Hawley, the data flow between a performer and an instrument is on the order of 1,000 bytes per minute, whereas a digital recording of the sound at acceptable commercial rates is about 10,000,000 bytes per minute [Hawley, 93]. At the gestural level, fundamental musical information such as note-on time, duration, velocity, and pitch are extracted and stored. This parametrization makes modifying and/or synthesizing music a simpler task. (Consider changing the velocity value of a note in a MIDI file versus redrawing the amplitude of the corresponding sound waves). Furthermore, high-level parameters such as tempo, harmony, and rhythm can be built on top of the fundamental parameters. Like functions and procedures in a computer program, the high-level parametric controllers should be abstract in concept. When a signal is sent to modify a particular high-level parameter, the caller to the controller should not need to know anything about how the controller is written, which fundamental parameters are used, etc., but simply what the controller does and how it is called.

Once the high-level controllers are designed and implemented, different tasks in music become different "agents." Each controller can be a different and independent process, and remote manipulation of musical parameters should be easy to accomplish. A user can send in control information to make a

particular phrase louder; or, if the music is being composed and transmitted in real-time, the user can request for a livelier rhythm, “flatter” melody, and so forth. The nature of this research deals with combining this parametric music making process with the remote-communication capabilities of the Internet. Parametric transmission of musical information and real time interaction on the Web together create the potential for live music performance over the Internet.

1.5 A Very Quick Tour of the Brain Opera

The physical space where the Brain Opera will take place can be roughly divided into two areas: the “lobby” and the “performance space.” The lobby is where the audience physically interact with various stations of “musical agents.” Each station in turn extracts raw materials and/or musical information from the participant, and stores collected data in a central server for later use. The performance space is where the actual performance takes place. Materials collected from users, together with pre-composed sections of music make up the foundation of the piece.

The aforementioned stations of “musical agents” include *the Forest*, *Harmonic Driving*, *Melody Easel*, *Gesture Wall*, and *Rhythm Trees*. They are described as follows [Machover, 95a]:

Forest

The first element in the interactive lobby space will be a large number of obelisks (or “trees”), arranged in a complex pattern (or “forest”). Each “tree” will offer a different experience, seemingly independent at first, but in fact interconnected through a complex network. Visitors will explore each “tree” by peeping into an opening. Some “trees” will give information about music and the mind, others will invite the visitor to record personal spoken or sung material, yet others will connect visitors with the Internet audience.

Harmonic Driving

Of all Brain Opera activities, this will be most like a traditional video game. Three separate “driving”

units will be contained in a circular, translucent kiosk. Players are seated in front of a graphic screen, with a steering wheel, joystick, and foot pedal as controls. Using specially designed software, the player will literally be able to “drive” through a piece of music, negotiating onrushing paths and trajectories from the screen. If the central road is followed, a fairly pleasant - but generic and bland - composition will be experienced. Detours and curvy overpasses must be taken to introduce variation, transformation and harmonic interest. Extra “spice” (accents, articulation, texture, color, etc.) is added by manipulating the left-hand joystick. The foot pedal controls the tempo, or speed that one travels through the piece. Musical “crashes” occur by slamming into roadsides, and points are awarded for the greatest amount of musical interest introduced in the fastest time!

Melody Easel

This station, of which there will be four, will give visitors the experience of playing a sophisticated monophonic instrument, like the violin. Seated at a table-like structure, you play melodies with the touch of a finger. A specially designed surface - made from transparent rubber - subtly measures the movement and touch of your finger; as you move, a beautiful graphic trace (like an exquisite water color) is produced on the display. Melodies can be improvised by moving the finger freely, or can be “rehearsed” by “connecting the dots” of melody fragments from the Brain Opera. In either case, the “instrument” responds to the most delicate variations in movement, making it possible to practice and perform the Melody Easel with remarkable virtuosity and sensitivity.

Gesture Wall

This “instrument” gives the experience of playing in a small musical ensemble. Five separate interactive wedges are placed next to each other, angled so as to form a curvy surface. Each wedge can accommodate one player, whose every gesture is measured by specially designed (and very precise) invisible movement sensors. Rather than creating music from scratch, players will react to musical fragments that flit across the high resolution graphic screens. Players will use arm movements to “throw” splotches of virtual color onto these screens, transforming, modifying, and “perturbing” the music as

it passes by. The five wedges will be interconnected, so that musical actions by one player will have an effect on the whole system.

Rhythm Tree

This circular sculpture surrounds the audience member with many suspended objects of varying shapes, sizes and “feels.” Sounds are produced by hitting or tapping these objects. In turn, each is connected to many others, like branches on a tree or synapses in the brain. When an individual object is struck, the strength of the hit as well as the hand's direction sends a signal ricocheting through the connected circuits. In this way, the rhythm you play animates the entire system, producing intelligent echoes and responses to your physical impulse.

The audience will interact with these stations for approximately an hour, then escorted to the performance space. The performance itself will last approximately 45 minutes and will have three distinct parts. According to the composer and the project leader Tod Machover, the three sections are divided as follows [Machover, 95a]:

- Part 1 will be a Sonic Blitz, as recorded voices of audience members are mixed into a dense sound collage, turning words into beautiful music.
- Part 2 will inject snippets of music produced by audience members, colliding and overlapping, and finally combining and aligning to create fascinating musical mini-pieces.
- Part 3 will bring together all of the musical themes and layers of the Brain Opera, getting ever faster and more intense, as the music builds to a crashing and cathartic climax.

Three performers, playing specially designed gesture instruments and using baton-like “magic wands,” will select, modify, and combine the various strands of music in real-time.

In addition, sensors in the room's carpet and radar beams at head level will measure audience movement and energy level during the performance, using this information to automatically influence the musical build-up and formal shape of the piece.

In Part 3 of the piece, a five-to-ten-minute section will focus almost exclusively on the events taking place at the Brain Opera website. The task of this project is therefore to explore, examine, and analyze possible Internet game models suited for interactive music performances.

2. ITX – THE *Expressive Piano Tutor*

The ITX Project was led by Prof. Tod Machover and Prof. Michael Hawley, and was designed and implemented by Alvin Fu, Teresa Marrin, and myself. This project deals heavily with the various parameters in music performances. Musical parameters are isolated, extracted, modified, and exaggerated so as to explore different aspects of musicality. My involvement with the ITX project prepared me of the necessary skills in manipulating parametrized musical information, as well as in viewing music as interactions of different parameters and musical changes as combinations of different parametric values.

2.1 Description of the Project

Various projects have tried to provide automated musical instrument teaching, especially for piano students. Most such systems provide feedback about the easily quantifiable aspects of piano playing, correcting wrong notes and adjusting inaccurate rhythm. In line with the interest of the *Hyperinstrument Group* in measuring and evaluating human expressive behavior, we have tried to design a system capable of giving qualitative feedback to the student: advising how to play more musically, and correcting errors in phrasing, articulation, and interpretation.

The *Expressive Piano Tutor* first asks the student to play a piece, and then uses two parallel systems to analyze the musicality of the performance. First, the system accesses a statistically prepared database, constructed from a collection of “normal” pianists playing the same piece, which easily identifies where the student has deviated significantly from “normal” musical performance. Second, intelligent music analyzers mark up the piano piece itself, deciding according to its formal and expressive features where a “normal” pianist would make interpretative variations. This information is used to find the places where the student performance needs musical work. A system of automated exaggeration is used to show the student the problems with his playing, and to highlight the musical feature to strive for.

2.2 How Does It All Work?

For the statistical analysis, we applied estimation and pattern classification theory to build a statistical profile of each note from a database of instructional performances. Each note is then characterized by a multi-dimensional feature vector profiling timing and loudness. The instructional performances enabled us to estimate a probability distribution for the feature vector. The student's feature vector can then be compared with this distribution. If the student is unlikely to be correct, he will be told to change his performance.

This approach has a number of advantages and disadvantages. One advantage is that no scores are needed; the instructional performance will provide all necessary data. Another advantage is that a statistical approach easily lends itself to a composite portrait of a piece. Perhaps most important, this approach is able to take in an arbitrarily large number of model performances (the more the better) and use all of them to instruct the student. In fact, with a rather large number of “model” performances, we believe that this system will present a “normal, musical” profile of the piece, rather than the very particular, hard-to-imitate “great artist” profile.

A disadvantage of this approach is that the method is only as good as the model performances. If the model performances are poor, the performance of the system will be poor. A further difficulty is that the number of model performances needed may be quite large, depending on the size of the feature vector and the exact method of estimation used. This number can be minimized by assuming a gaussian or other exponential distribution, and limiting the dimensionality of the feature vector.

Using data gathered from the estimation and pattern classification method, we developed the idea of an “interpretive envelope,” which would represent one or more parameters of interpretation and the range of possible values (both dynamics and tempo) for any given point in a piece. Based on this, we then developed a set of rules for “musical” trajectories through the envelope, and begin trying to

automate that process. In order to fulfill the request of making this a system which is compatible with “normal” musicians and teachers, we made sure that the concept of “interpretive envelope” encompasses a large range of possible and “normal” interpretations of a piece of music, as well as the extreme ones.

Next, we concentrated on identifying information such as structural sectioning, phrasing, etc. Having this information in hand, together with other tools which have already been written – e.g., the key, chord, cadence identifiers – we wrote the beginnings of a rule-based intelligent system which can take in any MIDI data and know what to do (and where) in order to make the piece sound “more musical.” We designed a range of allowable deviations in volume and timing. When a student uses this program, the key points (or “places of musical interest”) can be identified and analyzed quickly. We can then give the student appropriate and meaningful responses as well as exercise suggestions. Once we have tested a core group of these “musicalizing” algorithms, we can isolate the “places of musical interest” of a piece and compare the student’s MIDI data at those points with the numbers we get from the generative algorithm. If the general curves of the student match those generated by the algorithm, we will consider it “good,” and make appropriate comments like “good, but not enough,” or “good, but that was too much!”

In addition, we started to address the problem of defining a structural description for any piece whose MIDI score is known. This involved feeding a quantized score into a system and receiving a clear description of the form of the piece, as well as an automated interpretive envelope.

The system itself was developed on NeXTSTEP. We wrote an entirely new MIDI toolkit for the platform which is flexible enough to read and write (record and play, patch, etc.) MIDI files in several different formats. We also created an entirely-new file format for MIDI which would adapt well to our analysis methods. This representation was designed especially for this project, and will be discussed in detail below.

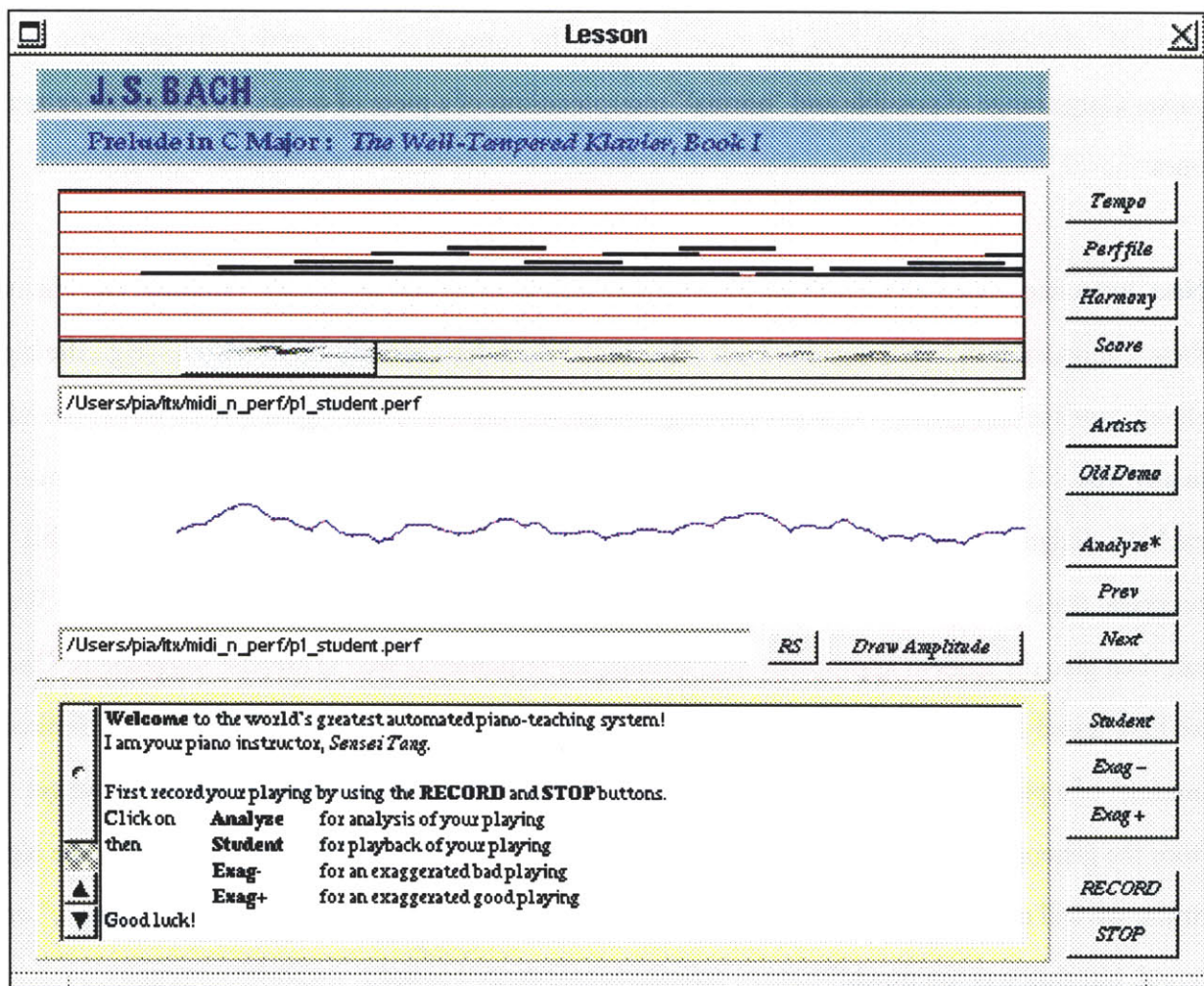


Figure 1: The *Expressive Piano Tutor*.

2.2.1 Buttons

Tempo	shows tempo analysis
Perf file	shows the student PERF file
Harmony	shows harmony analysis
Score	shows the score (in .tiff format)
Artists	launches Alvin's demo
Old Demo	launches the old "Yamaha Demo 1"

Analyze*	performs analysis of student performance
Prev	goes to previous section
Next	goes to next section
Student	plays back student performance of current section
Exag-	plays an exaggerated bad playing of current section
Exag+	plays an exaggerated good playing of current section
RECORD	records student performance
STOP	stops recording and playing invoked by RECORD, Student, Exag-, and Exag+

PRAELUDIUM I



Figure 2: By clicking on the **Score** button, the score of the loaded piece is displayed.

2.2.2 How the Graphing Routine Works

This program draws a graphical representation of the given MIDI file. It simply looks at a MIDI file, extracts information such as timing, duration, pitch, and velocity from the file, and represents them

in the following manner:

- Each note gets a horizontal “bar”
- The higher the note, the higher the bar
- The length of the bar represents the length of the note. The longer the note, the longer the bar
- The color of the bar reflects the velocity of the note. The louder the note, the darker the color

2.2.3 How the Amplitude Curve Works

This ProjectBuilder object (aView) takes in the velocity and timing information from a MIDI file, re-scales them, and draws the curve into a PostScript file. The horizontal axis is the time axis, and the vertical the velocity axis.

2.2.4 How the Teach Algorithm Works

Note: the PERF file format referred to below is our own ASCII representation of MIDI files. It contains the following information: note-on time, duration time, pitch, velocity, and an attribute string which records various attributes about that note. Our attribute string contains information such as right/left hand, dynamic marking, and measure/beat number.

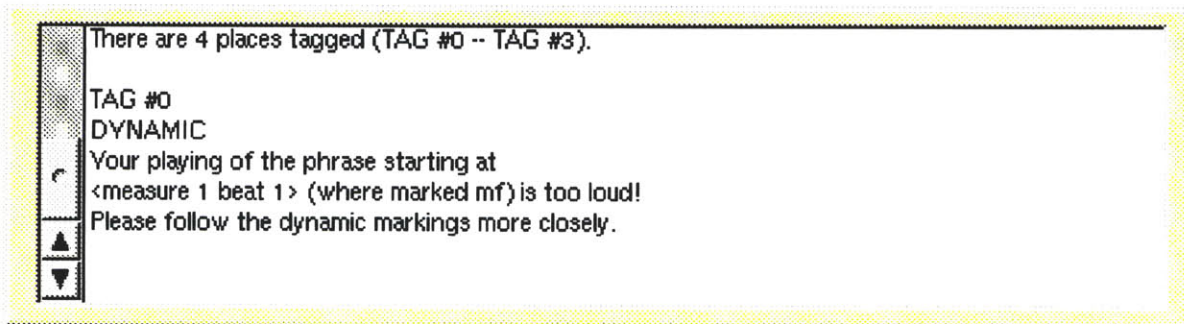
2.2.4.1 Philosophy

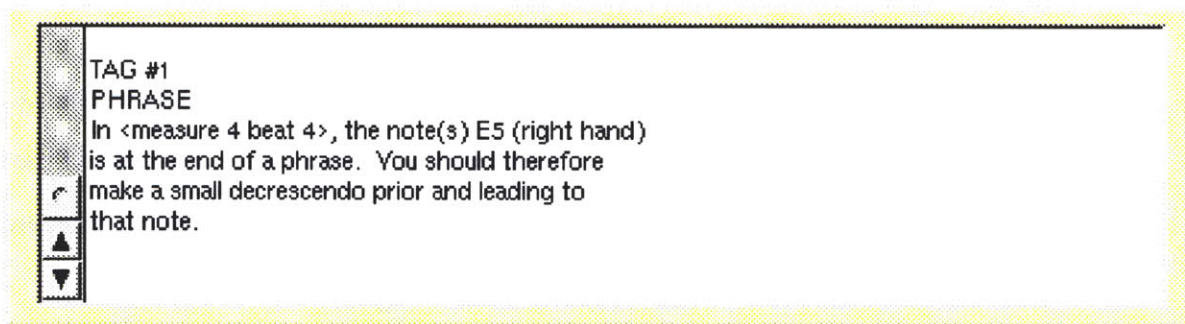
The philosophy behind this teach algorithm is the following: the program, when fed *any* pair of quantized and student performance PERF files, should be able to pick out “places of musical interest” to analyze. The only pre-condition is that the quantized PERF file should have its attribute fields carefully marked.

2.2.4.2 The “Flow”

The program’s flow-of-events is as follows:

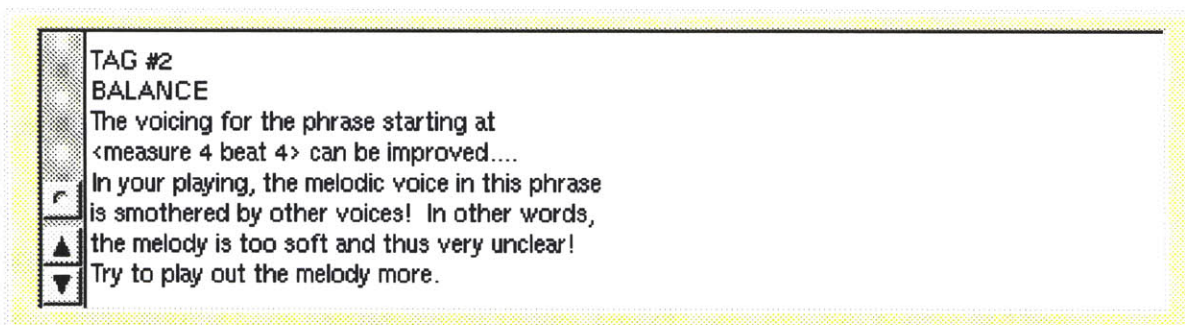
- student records his playing (**RECORD** and **STOP**)
- student presses **Analyze*** for an analysis of his performance
- program compares the quantized file and student file and checks for simple errors (i.e., extra notes and missing notes; takes into account notes played simultaneously). If the student's playing is incorrect, the program tells the student exactly where the mistakes are, and asks the student to record again.
- if the student's performance is error-free, then the program runs the student file through various loops (e.g., the "cadence loop," the "dynamic loop," etc.) and tags all places that "went wrong."
- program prints messages which points out why each tagged place is "bad." It also gives suggestions as to how this section can be improved.
- student can then play back his own playing of that section (**Student**), and/or an exaggerated bad version of his playing (**Exag-**), and/or an exaggerated good version of the same section (**Exag+**).
- after analysis is done, student can also click on **Draw Amplitude** and/or press return at the upper text drag-in field for graphical representations of his playing (general MIDI and dynamic).
- student can also go on to the next tagged section by clicking **Next**, or return to the previous tagged section by clicking **Prev**.





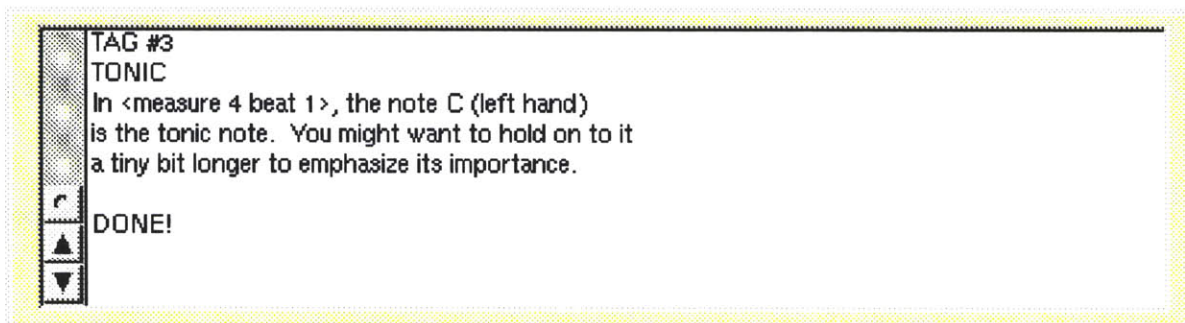
TAG #1
PHRASE
In <measure 4 beat 4>, the note(s) E5 (right hand) is at the end of a phrase. You should therefore make a small decrescendo prior and leading to that note.

Navigation icons: a curved arrow pointing left, an upward-pointing triangle, and a downward-pointing triangle.



TAG #2
BALANCE
The voicing for the phrase starting at <measure 4 beat 4> can be improved....
In your playing, the melodic voice in this phrase is smothered by other voices! In other words, the melody is too soft and thus very unclear!
Try to play out the melody more.

Navigation icons: a curved arrow pointing left, an upward-pointing triangle, and a downward-pointing triangle.



TAG #3
TONIC
In <measure 4 beat 1>, the note C (left hand) is the tonic note. You might want to hold on to it a tiny bit longer to emphasize its importance.

DONE!

Navigation icons: a curved arrow pointing left, an upward-pointing triangle, and a downward-pointing triangle.

Figure 3: The *Expressive Piano Tutor*. After the analysis, the system returns a list of tagged sections which it thinks the student needs to work on. The student can then navigate through the sections by using the Next and Prev buttons, or play the exaggerated good and bad examples to follow along.

2.2.4.3 Determining “Places of Musical Interest”

To determine where the “places of musical interest” should be, I drew heavily from my own experi-

ence both as a piano student and a piano teacher. I have had 19 years of piano lessons (from age 5 to 24) with six different piano teachers. I have also taught piano for quite a number of years. One interesting thing I found is that no matter how elementary or advanced a student is, the teacher invariably spends a big chunk of the lesson time telling the student to *play what the music says*. Seemingly basic instructions like “Play piano where the score says *piano!*” or “keep the tempo! Don’t speed up!” are *not* reserved only to beginning students. My piano teacher at Juilliard – Mr. Martin Canin (who serves in the jury of many international piano competition), for example, was even rumored to have forced his students practice with a metronome at all times. Even though this *is* indeed only a rumor, it shows how world-class teachers are, still, known to care a great deal about the most fundamental elements in music-playing.

Because of this realization, I decided that an important part of the program should be devoted to this book-keeping, “stick-to-the-score!” type of evaluations. The evaluations include procedures that check for dynamic and timing ranges.

Another important feature of a “musical” performance is the projection of the melodic line (or, the right “balance” / “voicing”). I have therefore written a routine that analyzes a phrase at a time, finds the average balance of the primary voice (the “melody”), secondary voice (often the “bass line”), and the accompaniment voice (i.e., everything else), and calculates to see if the balances among the three are acceptable.

Another typical “musical thing to do” is to perform a slight *ritardando* and/or *diminuendo* at an authentic cadence. I have added a procedure that looks at places where perfect and imperfect authentic cadences occur and checks for the presence of *ritardando* or *diminuendo* in the student PERF file.

Likewise, a slight *diminuendo* is almost always performed at the end of a slur – which often marks the end of a *musical motif* or idea. This rule often supersedes the more global *crescendo* or *decrescendo*

markings. I find this to be an extremely important indicator of the performer's phrasing ability.

A sensitive performer is always aware of the tonal center for each section in the piece he plays. Often a slight stretch is done to the tonic note when it occurs in the bass and at the beginning of a new measure. There is also a routine in the program that checks for such a stretch or emphasis of the tonic note.

These are the "places of musical interest" that the system looks for and examines. The rules I have developed here are based on my experience as a piano student and a performance artist, my experience as a piano teacher, what I actually listen to and for in a performance, and my years of study of musical aesthetics and performance practice. These rules are not made up arbitrarily. Some of them may seem too simple at first, but they are accurate indicators of a student's level of musicality. The rules that have been developed are general enough to be applied to most Classical and Romantic music. In the future (if time and resources permit), it is conceivable to develop different rule banks for different composers, and teach students how to play more "Chopinesque" than, say, "Lisztian."

The process of tagging sections to "teach about" is not a simple task. A typical tagging routine usually invokes several other large-scale routines. The "cadence routine," for example, has to invoke a routine that stores places where cadences occur. This routine in turn invokes another one that determines what harmony each beat/unit-of-beats is, which too invokes a key-signature determinator. To know whether a *ritardando* is performed at a the cadence point, a "tempo profile" also needs to be created before the analysis can take place.

2.2.4.4 Good and Bad Exaggerators

Good and bad exaggerators are used for demonstration purposes. They include the following categories:

- tempo and dynamic exaggerators

- octavator – which adds an octave above the melodic voice, or an octave below the bass voice.
- musicalizers – routines which perform “musical procedures” to a file. For example, there are musicalizers which make a *ritard* at cadence points or end of phrases, perform *decre-scendo* at the end of all phrases, perform *decre-scendo* at the end of all slurs, stretch the tonic note whenever it is encountered at the bass and at the beginning of a measure (provided that the first bass note of the previous measure is not the tonic too), etc. The musicalizers are used for “good exaggerations.”

A typical dynamic exaggerator is defined as follows:

```

exagv(Performance *p, int shou, double Degree) {
    int i;
    int new_amp;
    Note *a;

    switch (shou) {
        case (BOTH): {
            for (i = 1, a = p->n; i <= p->nn; i++, a++) {
                new_amp = a->amp + ((Degree - 1) * diffarrayv[i]);
                a->amp = adjustv(new_amp);
            }
        }
        Case (LEFT): {
            for (i = 1, a = p->n; i <= p->nn; i++, a++) {
                if (a->type[0] == '0') {
                    new_amp = a->amp + ((Degree - 1) * diffarrayv[i]);
                    a->amp = adjustv(new_amp);
                }
            }
        }
        [....]
    }
}

```

In this example, the parameter “shou” (“hand” in Mandarin) indicates to the routine that notes for the left hand (0), right hand (1), or both (2) are to be dynamically exaggerated. The differences in vol-

ume for notes in file1 and file2 are stored in *diffarrayu*. Each of the values is then multiplied by a degree factor, and added to the original *amp* of the corresponding note. Ordinarily, the “student file” (PERF file of a student’s performance) is set to be compared against a mathematical, dynamically flat, and quantized version of the same piece. That way, we see exactly what the student is doing in terms of volume and timing. Sometimes a “student file” is set against an “expert performance,” where we can quickly obtain statistical information about the differences between a novice and an expert performer.

2.2.5 How the Harmony Analyzer Works

The harmony analyzer first calls a routine that determines the key of the piece. Then, for each unit of beats specified, it marks the presence of each note in a note-map string. The string is then compared against the pre-stored patterns (e.g., a major chord would contain the pattern 100010010000 where the digits represent the twelve notes in an octave, 1 indicates “present,” and 0 “absent.”)

```
#define CM_CH      "100010010000" // C major triad
#define Cm_CH      "100100010000" // C minor triad
#define CA_CH      "100010001000" // C augmented triad
#define Cd_CH      "100100100000" // C diminished triad
#define C7MM_CH    "100010010001" // Major 7th chord - major 7th,major triad
#define C7mM_CH    "100010010010" // Dominant 7th chord - minor 7th, major triad
#define C7mMx5_CH  "100010000010" // dominant 7th with missing 5
#define C7Mm_CH    "100100010001" // 7th chord - major 7th, minor triad
#define C7mm_CH    "100100010010" // 7th chord - minor 7th, minor triad
#define C7md_CH    "100100100010" // 7th chord - minor 7th, diminished triad
#define C7dd_CH    "100100100100" // 7th chord - diminished 7th, diminished triad
#define Cs7dd_CH   "010010010010"
#define D7dd_CH    "001001001001"
```

A glimpse of how chords are defined in ITX.

This procedure is capable of detecting all major, minor, diminished, and augmented triads, as well as major, minor, augmented, diminished, half-diminished, major-minor, and dominant 7th chords. Also, all the notes are weighted by duration. The longer the duration, the more “important” it is. (Think of passing notes, and how “less important” they are in determining the quality of a chord in comparison to a long bass note.) If, after the first pass, no match is found, then the “lightest” note is eliminated, and the matching process is executed again. This loop is repeated until either a match is found, or else a harmony of quality “UNKNOWN” (or “ambiguous”) is returned.

The following is an example of the Harmony Analyzer at work. The piece analyzed here is J.S. Bach’s Prelude in C Major, from *The Well-Tempered Klavier, Book I*:

```
burton> chord -d 4 math2
measure 1 <beats 1 to 4>:      C Major root
measure 2 <beats 1 to 4>:      D minor 2
measure 3 <beats 1 to 4>:      G Dominant 65
measure 4 <beats 1 to 4>:      C Major root
measure 5 <beats 1 to 4>:      A minor 6
measure 6 <beats 1 to 4>:      D Dominant 2
measure 7 <beats 1 to 4>:      G Major 6
measure 8 <beats 1 to 4>:      C Major 2
measure 9 <beats 1 to 4>:      A minor 7
measure 10 <beats 1 to 4>:     D Dominant 7
measure 11 <beats 1 to 4>:     G Major root
measure 12 <beats 1 to 4>:     C#/Db/E/G/A#/Bb diminished 43
measure 13 <beats 1 to 4>:     D minor 6
measure 14 <beats 1 to 4>:     D/F/G#/Ab/B diminished 65
measure 15 <beats 1 to 4>:     C Major 6
measure 16 <beats 1 to 4>:     F Major 2
measure 17 <beats 1 to 4>:     D minor 7
measure 18 <beats 1 to 4>:     G Dominant 7
measure 19 <beats 1 to 4>:     C Major root
```

```

measure 20 <beats 1 to 4>:      C Dominant 7
measure 21 <beats 1 to 4>:      F Major 7
measure 22 <beats 1 to 4>:      C/D#/Eb/F#/Gb/A diminished 43
measure 23 <beats 1 to 4>:      ambiguous
measure 24 <beats 1 to 4>:      G Dominant 7
measure 25 <beats 1 to 4>:      C Major 64
measure 26 <beats 1 to 4>:      ambiguous
measure 27 <beats 1 to 4>:      G Dominant 7
measure 28 <beats 1 to 4>:      ambiguous
measure 29 <beats 1 to 4>:      C Major 64
measure 30 <beats 1 to 4>:      ambiguous
measure 31 <beats 1 to 4>:      G Dominant 7
measure 32 <beats 1 to 4>:      C Dominant 7
measure 33 <beats 1 to 4>:      D minor 2
measure 34 <beats 1 to 4>:      ambiguous
measure 35 <beats -3 to 1>:     C Major root
Cadence found at measure 4 beat 1.    Type: Imperfect Authentic
Cadence found at measure 7 beat 1.    Type: Imperfect Authentic
Cadence found at measure 11 beat 1.   Type: Perfect Authentic
Cadence found at measure 19 beat 1.   Type: Perfect Authentic

```

Harmony analysis of Prelude in C, from J.S. Bach's WTC Book I

2.2.6 How the Tempo Analyzer Works

The tempo analyzer first determines how many beats there are in a measure, then keeps track of the length of all beats. From there, buckets of all the different duration lengths are created. The most-frequently-encountered length is set to be the norm. With this information in hand, a tempo profile can then be created. (e.g., if the length of a beat exceeds a certain percentage of the norm, it is tagged “faster than usual.”) The program also gives a final verdict as to the steadiness of the overall tempo:

```

burton> tempo /Users/pia/itx/midi_n_perf/p1_student.perf
measure 1 beat 1: 0.799000

```



```

measure 1 beat 2: 0.802000
measure 1 beat 3: 0.797000
measure 1 beat 4: 0.803000
measure 2 beat 1: 0.800000
measure 2 beat 2: 0.798000
measure 2 beat 3: 0.802000
measure 2 beat 4: 0.799000
measure 3 beat 1: 0.801000
measure 3 beat 2: 0.800000
measure 3 beat 3: 0.800000
measure 3 beat 4: 0.799000
measure 4 beat 1: 0.800000
measure 4 beat 2: 0.801000
measure 4 beat 3: 0.800000
measure 4 beat 4: 2.355000

```

```

0 length: 0.800000    beats: 5
1 length: 0.799000    beats: 3
2 length: 0.802000    beats: 2
3 length: 0.801000    beats: 2
4 length: 0.797000    beats: 1
5 length: 1.799000    beats: 1
6 length: 2.355000    beats: 1
7 length: 0.798000    beats: 1
8 length: 0.803000    beats: 1

```

Compared to standard tempo (length per beat = 00:00.800),
the student is playing
slower at measure 4 beat 4 by 194.37%
Overall tempo is steady!

Tempo analysis of Prelude in C, from J.S. Bach's WTC Book I

2.2.7 How the Attribute Field Is Used

The attribute field is actually an integer string.

Example:

(note-on time; note duration; pitch; velocity; attribute string)

```
# /Users/pia/itx/midi_n_perf/pl_student_raw.perf: duration
00:14.600, 65 notes, 4.5 notes/sec, avg amp 62
00:01.799 01.554 C4 60 03011000020011500
00:02.000 01.549 E4 63 02010000020011500
00:02.200 00.250 G4 65 11010000020011500
00:02.400 00.250 C5 70 11010000020011500
00:02.598 00.250 E5 75 11010000020012500
00:02.799 00.250 G4 78 11010000020012500
00:03.000 00.250 C5 75 11010000020012500
00:03.200 00.250 E5 68 11010000020012500
00:03.400 01.555 C4 69 03011000020013500
00:03.598 01.550 E4 64 02010000020013500
00:03.799 00.250 G4 61 11010000020013500
00:04.000 00.250 C5 67 11010000020013500
00:04.197 00.250 E5 59 11010000020014500
00:04.400 00.250 G4 57 11010000020014500
00:04.598 00.250 C5 58 11010000020014500
00:04.800 00.250 E5 53 11010000020014500
00:05.000 01.554 C4 56 03011000010021500
00:05.197 01.550 D4 63 02010000010021500
00:05.400 00.250 A4 65 11010000010021500
00:05.598 00.250 D5 64 11010000010021500
00:05.800 00.250 F5 61 11010000010022500
00:06.000 00.250 A4 58 11010000010022500
...
```

A PERF file of Prelude in C, from J.S. Bach's WTC Book I

Each digit represents the following:

digit	range	description
--------------	--------------	--------------------

0	[0/1]	Left/Right hand (1 = right, 0 = left)
---	-------	---------------------------------------

1	[1-9]	voice number (top-most = 1, etc.)
---	-------	-----------------------------------

2	[1-99]	phrase number
3		
4	[0/1/2]	melody (primary = 1, secondary = 2, otherwise 0)
5	[0-999]	chord number (members of same chord get same #. 0 for single-notes)
6		
7		
8	[0/1/2]	time (2 = accel, 1 = rit, 0 otherwise)
9	[0/1/2]	volume (2 = cresc, 1 = decreasc, 0 otherwise)
10	[1-999]	measure number
11		
12		
13	[1-9]	beat number
14	[0-9]	dynamic marking (if not marked, then "5")
		<u>code</u> <u>musical equivalent</u> <u>MIDI-velocity</u>
		9 <i>sf</i> +20
		8 <i>fff</i> 110-127
		7 <i>ff</i> 88-110
		6 <i>f</i> 70-88
		5 <i>mf</i> 55-70
		4 <i>mp</i> 40-55
		3 <i>p</i> 27-40
		2 <i>pp</i> 20-27
		1 <i>ppp</i> 15-20
		0 <i>p sub</i> 27-40
15	[0/1]	double bar encountered? (1 = yes, 0 = no)
16	[0/1/2]	slur (2 = beginning of and/or within, 1 = end of slur, 0 = no slur)

2.3 Parametrization of Music

An important aspect of the ITX project is its treatment of music as an entity of various parameters. As demonstrated in the previous section, performances of piano music can be captured and later re-cre-

ated by storing the following parameters: note-on time, note-off time, pitch, velocity (loudness), pedal-on time, pedal-off time, and pedal velocity (depth). In the ITX project, these information are extracted, calculated, modified, and manipulated so as to create useful teaching tools such as the *octavator*, *tempo exaggerator*, and *dynamic exaggerator*.

In a way, the ITX project can be viewed as a study of the parametrization of musical performances. By parametrizing a CD recording or a live performance, the system becomes capable of analyzing such *qualitative* information as the musicality of the playing, as well as giving useful, intelligent feedback about the performance.

From the above-mentioned parameters (which are the most fundamental of all), “higher parameters” can also be constructed. These second-layer parameters include rhythm, harmony, tempo, voicing, etc. With the knowledge of music theory and performance practice, we were able to take in these higher parameters and determine the performer’s steadiness in tempo, rhythmic accuracy, appropriateness in voicing, etc.

Combining the second-layer parameters, a third-layer parameters can be constructed too. This layer is perhaps the most “musical” and abstract of all. By combining tempo information and harmonic analysis, for example, we were able to see if the performer did anything special at pivot chords and cadence points, or if he just played on as if nothing happened. (A typical “musical” thing to do here would be to slow down a bit so as to emphasize the harmonic change and/or the beginning of a new phrase.)

Procedures such as the good and bad *exaggerators* take in the parameter values, calculate the differences between those values and the “norm” (both dynamic-wise and timing-wise), and exaggerate those values by a factor of between 1 and 1.85. A new MIDI sequence containing the exaggerated segment is then constructed. Likewise, in procedures such as the *musicalizers*, values are taken from a

plain MIDI file. The file's attribute fields are then carefully marked. By applying rules such as

- right-hand notes – the higher the pitch, the louder the amp;
the lower the pitch, the softer the amp.
- left hand notes – the lower the pitch, the louder the amp;
the higher the pitch, the softer the amp.

to the file, and rules which modify other parameters of the performance, an “expressive,” “musical” performance is created. This sort of routines are especially interesting to study, because it allows us to isolate rules of musicality and the parameters affected, and apply them to originally non-musical playings. Studying these routines can also help us in teaching the students precisely what to do in order to gain the desired effects on a piece of music.

Many of the routines used in the interactive Web systems for music which I have designed and implemented later come from the routines used here in the ITX project. Musical segments produced in these systems are often the results of changes in the parametric values. These routines can be best described as “musical filters” or “musical black-box.” When a stream of music is passed through these routines, certain parameters will change, and a new (or modified) stream is produced. Whichever filter to pass through depends on the effect desired. Although oftentimes these parameters are manipulated independently (as in the case of the Brain Opera), with careful design, they can produce coherent and wonderful-sounding music.

3. INTERACTIVE WEB SYSTEMS: OVERVIEW AND BASIC DESIGN ISSUES

3.1 Basic Design Issues

This project explores different dimensions of collaborative music making over the Internet. Four game models are examined, each with a different design philosophy behind it. The first game model, *Sound Memory* and *Bouncing Balls*, are simple games which are fun to play. *Sound Memory* is modelled after the famous card game *Memory*, and is written for a single user. *Bouncing Balls* has both the single-user version and the multiple-player version. The multiple-user version of *Bouncing Balls* is designed so that most of the processing take place locally. Both *Sound Memory* and *Bouncing Balls* use .au sounds, and assume no formal musical training of its players.

The second model, *Draw Music*, also assumes no formal musical training of its players. This model explores *individualism* within a collaborative environment. Each participant is given certain rules to abide by, tools to use, then is left alone to do whatever he wants. He will submit his final creation – which will eventually be concatenated with other users’ submissions – together forming a finished piece.

The third model, *The Internet Pong Game*, is designed for large groups of participants. Players need not to have any formal musical training, and can join or leave at any point of the game. This model explores the philosophy of *unintentional music-making*. In this game, music is a by-product of some other activity (in particular, game-playing), not the focal point. The system also makes no attempt to direct or guide the flow of music. The idea is to just “let things be,” and see what kind of music comes out.

The fourth model, the *Internet Hyperinstrument*, is designed for expert musicians as well as users without extensive musical training. This model explores the possibility of remote improvisation sessions with other users. Participants will collaborate in the music making process. Short solo performances

(“spot-lighting”) within a session are also possible.

These four models are designed with different user groups in mind. They also examine important aspects and types of collaborative music making. These games can serve as the paradigms for Internet music games to be developed in the future.

3.2 Limitations

3.2.1 Java and MIDI-lessness

As of now, Java does *not* support MIDI. Java supports only one sound format: 8 bit, μ law, 8000 Hz, one-channel, Sun “.au” files. The sound quality of .au files is not very good. This presents to be a great obstacle in developing music-related Java programs for the Internet.

Due to Java’s lack of MIDI capability, writing MIDI-style control messages or procedural calls become problematic. As a result, MIDI output can only be imitated through the use of a large number of sound files, each file representing a single note. As John Yu writes in *Computer Generated Music Composition*:

The primary disadvantage of this method is that output is completely limited by the number and quality of the sound files. Given the entire MIDI range of 128 pitches, multiplied by the different volumes desired, and again multiplied by the number of instrument sounds desired, approximation of MIDI through audio files becomes extremely difficult.

The issue of Java and its MIDI-lessness will be discussed further in chapters dealing with *The Internet Pong Game* and *Internet Hyperinstrument*.

3.2.2 Speed

Another technological limitation is speed. An ideal real-time interactive experience should be seamless, fast, and smooth. This is not the case yet with what is available. In the *Internet Pong Game*, for

example, the constant polling, sending, and receiving of information among the clients and server significantly slow down the program. To compensate for this, it became necessary to cut down the number of objects in the game field as well as to reduce the rate at which information is updated. More detailed analyses on the speed issue will follow in subsequent chapters.

4. SIMPLE YET EFFECTIVE GAMES: *Sound Memory* and *Bouncing Balls*

Fun games do not need to be complicated! Simple games can be exciting and effective too. The following two games: *Sound Memory* and *Bouncing Balls* are two examples of such games.

4.1 *Sound Memory*

One of the most effective Java games developed by the *Performance / Projects in Media & Music* class is a simple sound memory game, written principally by Roberto De Prisco.

4.1.1 Making a Match

This game is inspired by the traditional card game “memory.” In the card game version, all the cards are laid out face-down. The rules of the game are as follows:

- Each player takes turn to flip over a pair of cards (so that those two cards are now face-up).
- If a player gets a match (i.e., both cards have the same number), then
he gets to flip over two more cards. Repeat this process until a non-match occurs, in which case the game proceeds to the next player-in-turn.

Otherwise

he loses the turn and the game proceeds to the next player-in-turn.

The rules for *Sound Memory* are somewhat different. Since *Sound Memory* is designed for a single player, there is no one to compete with. Instead of matching numbers, each card – here represented by a brain for its association with “memory” and the “Brain Opera” – now contains a sound. The goal of the game is therefore to match as many pairs of sounds as possible. When the player clicks on a brain, the brain gets a different background color – signifying that the brain has been selected, and the sound of the brain is revealed (played). The player should then click on another brain. If the sound of the second brain is same as that of the first, then both brains will disappear, revealing parts of the picture underneath all the brains. The more matches one makes, the more of the picture one sees. Even-

tually, when all the brains are gone, the entire picture will appear, and some congratulatory message will also be displayed across the screen.

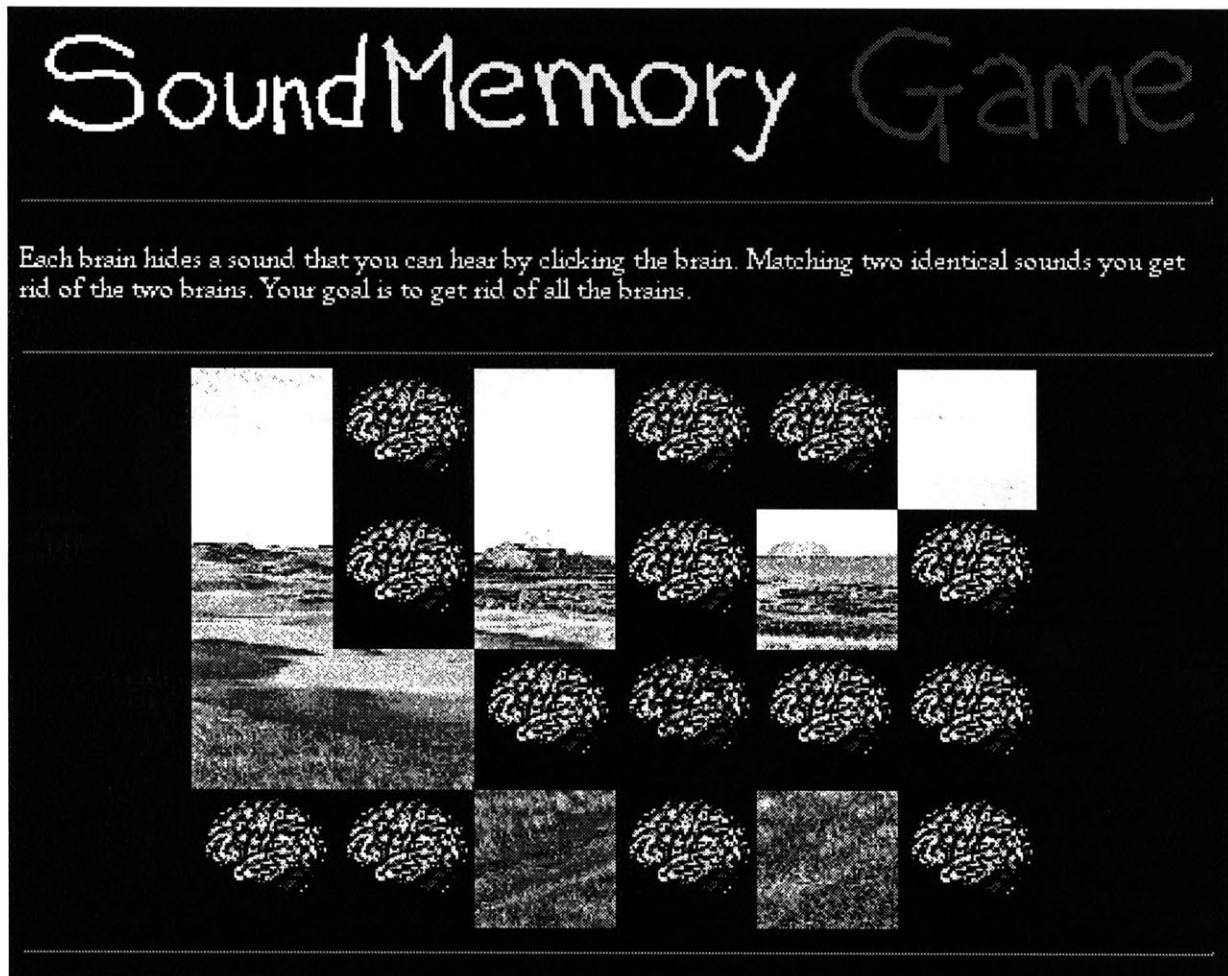


Figure 4: *Sound Memory*. When matches are made, the brains disappear.

Sound Memory comes in different difficulty levels. The simpler levels contain only highly-distinctive sounds that are easy to recognize and distinguish, whereas the more difficult levels, for example, ask the player to match various seventh chords – augmented, dominant, half-diminished, etc. – all played with the same timbre. In particular, these levels are: notes, chords, sounds, and melodies. Users are

also asked to choose a game of 4 sounds, 8 sounds, or 12 sounds.

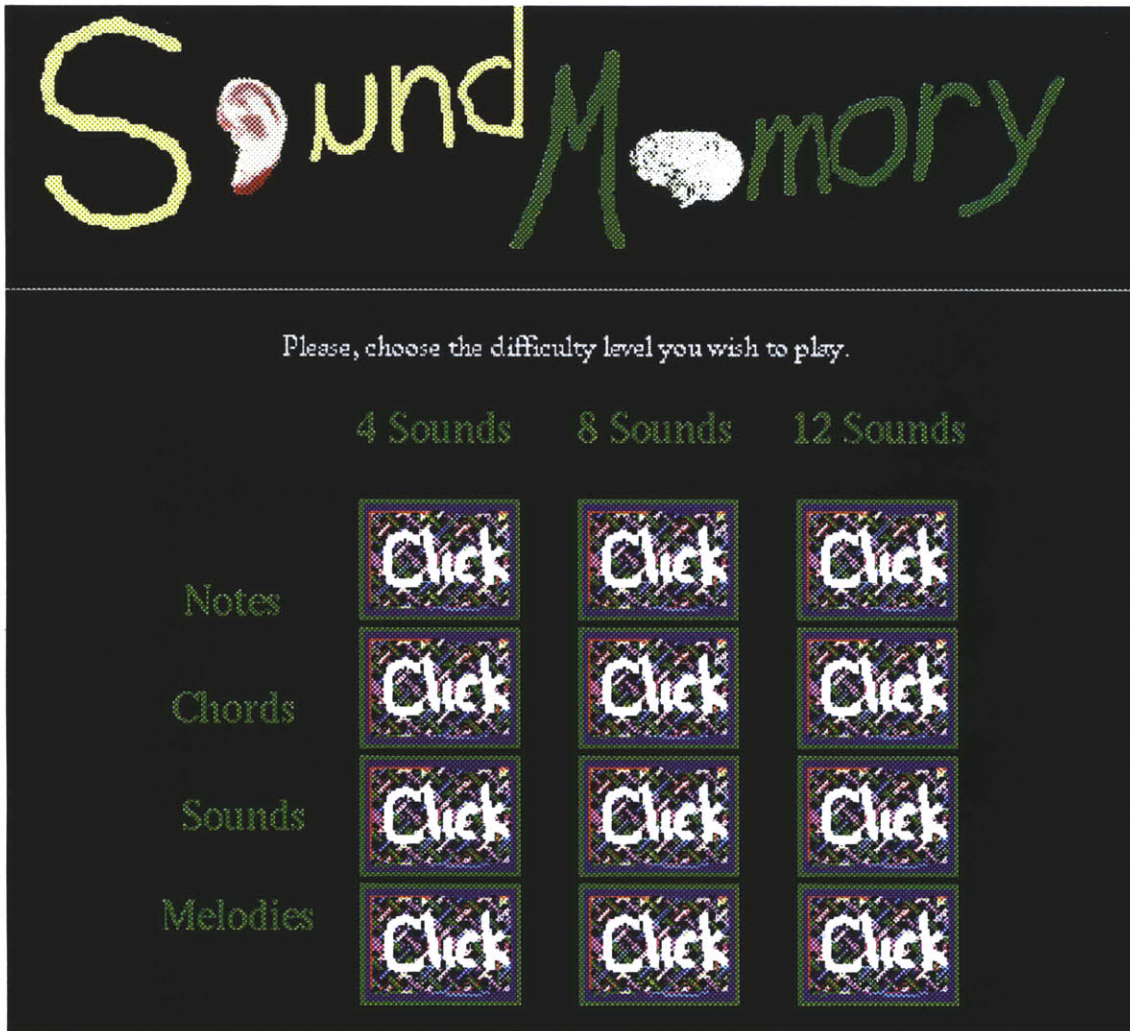


Figure 5: *Sound Memory* and its difficulty levels.

The concept of the game is fairly straightforward. The graphics are simple; the sounds are stored in .au format. There is no animation involved. Nevertheless, we believe that it is an effective and fun game which has great potentials.

4.1.2 The Role of Music in *Sound Memory*

Sound Memory can be categorized as a sound/music game in that it involves remembering and identifying sounds and musical fragments. However, the players are not asked to “make music,” or to do anything that would be considered “musical” in the traditional sense. Nevertheless, a simple game like *Sound Memory* which contains lots of interesting, attention-grabbing sounds can be highly entertaining.

Another aspect of *Sound Memory* is that it is educational in nature. In the “chord” level, for example, users are asked to match various seventh chords of the same timbre. Recognizing chord qualities by ear is actually an important skill for all musicians – composers and performers alike. It is entirely conceivable that a solfège teacher in a music conservatory would instruct his pupils to play a few rounds of *Sound Memory* as a homework assignment. With some minor improvements and new layers (e.g., progressions, intervals, qualities of triads, etc.), *Sound Memory* can be a useful classroom aid of great entertainment value.

4.1.3 Improving *Sound Memory*

Although *Sound Memory* (as it stands right now) is fun to play, many improvements can still be made. First of all, the “prize” awarded for completing the game is rather “bleak.” When a user completes the game, he gets to see 1) a picture in full which was once covered up by the brains, and 2) some congratulatory message. Perhaps a simple animation accompanied by some majestic-sounding music would be more rewarding.

Another suggestion is to add an all-time top-20 score board for each difficulty level. The score board should list the name, e-mail address, and the score of the all-time top-20 players. The score should be inversely related to the amount of time a user takes to complete the game. When a user makes into the top 20, he will be asked to leave his information at the site. The top-20 list will then be updated.

Small improvements like these will make *Sound Memory* more attractive. Moreover, the top-20 chart can make the game more competitive, at the same time give off the air of the existence of a “Internet Community.”

4.2 *Bouncing Balls*

Bouncing Balls is a Java game written principally by Josh Strickon. This game was inspired by Matthew Krom’s Java applet “Musical Balls.” In Musical Balls, whenever the user clicks (randomly) in a designated area, a round ball will appear at the coordinate of the mouse-click. All the balls will then move upward toward the top of the screen. (This upward movement is constant throughout the game.) When a ball reaches the top of the window, a sound whose pitch it represents will be played. The pitches are laid out very much like the traditional piano: the high notes are on the right, and the low notes are on the left; the pitch of any note is always higher than those to its left.

Bouncing Balls is also called “Rhythm-Bouncer.” When the game starts off, the entire window is black. By clicking anywhere within the window, a ball with a given color, speed, and initial angle will be launched. Every time that ball hits a wall it will play the sound associated with that color and bounce off.

There are three menus in this applet. The first pull-down menu represents color. Each color is a unique sound. The second menu is associated with speed. The user can select a speed for each ball he fires off. The greater the number, the faster the speed. The third menu is initial angle, which determines the angle at which the ball will be fired.

The pull-down menu’s and buttons for *Bouncing Balls* are listed below:

[P-d Menu]	[P-d Menu]	[P-d Menu]	[Button]
<u>Color</u>	<u>Speed</u>	<u>Angle</u>	<u>Reset</u>
red	1	0 degree	

yellow	2	45 degrees
green	3	90 degrees
blue	4	135 degrees
orange	5	180 degrees
magenta	6	
white	7	
gray	8	
cyan	9	
pink	10	

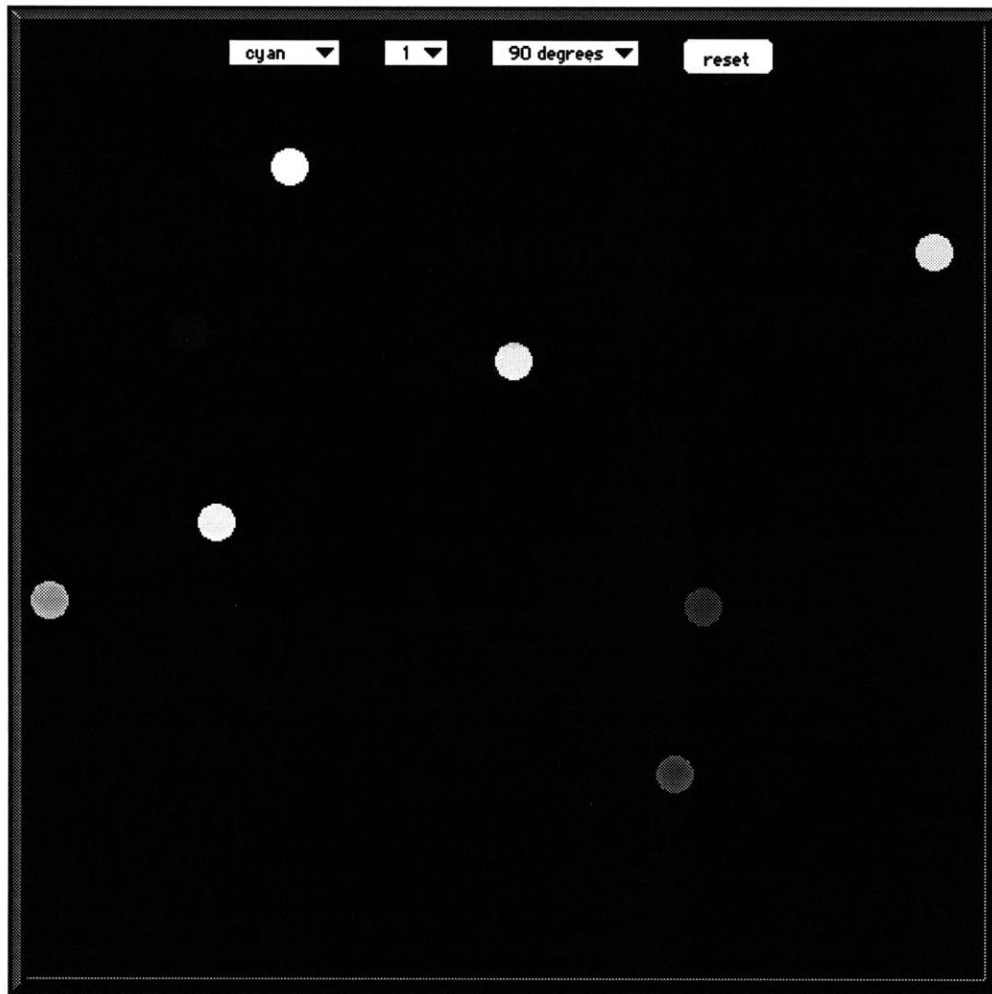


Figure 6: *Bouncing Balls* in action. The balls are travelling in different speed and direction. Each time a ball bounces off the wall, a sound or a musical fragment is played.

The *reset* button clears the screen of all balls. A maximum of 20 balls are allowed per game. When the maximum is reached, clicking on the reset button will start the game over.

4.2.1 From Total Randomness to Rhythmic Perfection

We have observed that when someone plays *Bouncing Balls* for the first time, he would most likely select ball colors, speed, and initial angles randomly, and fire off the balls aimlessly just to see how they all sound together. After a few rounds, however, the user would start to explore more sophisticated rhythmic possibilities. After he becomes familiar with all the sounds available to him, the user might begin to pick different colors more carefully, and fire off balls at pre-meditated moments so as to create a rhythmic texture which he desires.

When a ball is fired off at 0 degree, for example, it will simply travel back and forth across the screen. When the ball bounces off from the left and the right “walls,” the sound associated with its color will be played, thus creating a “beat.” Since the balls all travel in constant speed, a steady rhythmic pattern can be designed and played out by carefully choosing the right combination of colors, speed, and angles, and by firing off the balls at the right moments.

This game contains lots of interesting sounds. It can be appreciated by anyone: by those who are simply clicking away randomly to see “what might happen,” as well as by those who have a clearer design and plan in mind, and use the game as a looping percussion machine.

4.2.2 Multiple-Players

Bouncing Balls was originally written for one player. After the game was completed, the class suggested that a multiple-player version be implemented.

4.2.2.1 *WebStar* and *StarClient*

In order to support multiple-user communication in Java, it is necessary to write some cgi scripts that will handle all the “interactive connections” between the Web server and its clients. When this communication is set up, Web clients can then “talk” to one another through the way of the Web server. This layer of communication is essential for implementing multiple-user games, or any truly interactive communication among the users.

For the multiple-player version of *Bouncing Balls* (and other multiple-user Java applets to follow), we decided to use the *WebStar* package, designed, written, and installed by Eric Métois of the M.I.T. Media Laboratory. The *WebStar* package includes the *WebStar* server and the *StarClient* Java class and interface. The *WebStar* server is a process running constantly on the server side. It is responsible for handling all the “interactive connections” among Web clients. This program keeps track of different users who are connected to the server and dispatches any message that one client sends to the rest of the appropriate community (or group) [WebStar, 96]. According to the reference manual, “a group” is defined to be clients tied to, or related to, a specific experience. For instance, each interactive Web game would have its own group that would be different from the “Web performance” group. So far, this notion of “group” is associated with the actual port on which this communication takes place. A single instance of the *WebStar* server deals with a single group, but there can be many instances of that same program running on the same machine, each listening to a different port (and therefore dealing with a different group or community) [WebStar, 96].

The *WebStar* server first assigns an unique ID number to each client connected. (ID #0 is reserved for the server itself.) When a new client connects, it is immediately assigned a new ID. Clients can talk to the server by sending “messages” to the server. A message is simply a string of characters terminated by a new-line (\n) character. Incoming messages from the clients will be multiplexed and redistributed to all the clients.

Upon connecting to the server, for example, the client might see the following message:


```
0 Welcome to Bouncing Balls! 32\n
```

The first number in the message indicates that it was sent by the server, whose ID number is 0. 32 is the new ID number which the client has just been assigned.

Here is another example:

```
12 new ball at x = 25, y = 141\n
```

This is a message sent by client #12, announcing to everyone in the group the position of the ball mostly-recently fired. (This message is echoed to client #12 itself as well.)

For the actual writing of the Java code, the applet should use the *StarClient* class and implement the *WebStarHandler* interface. That package deals with all the low level communication layer, including fetching client ID's from the server, etc.

WebStar is installed on *w.media.mit.edu*. All the multiple-user games are developed and currently reside on *w*.

4.2.2.2 Running Things Locally

With *WebStar*, modifying the single-user *Bouncing Balls* to a multiple-player version was quite straightforward. The new version allows every client to see and hear all the balls fired by everyone else in the group. In other words, each ball one fires – along with the tempo and sound segment it carries – will appear on everyone else's screen. Different users can therefore collaborate and make music together.

One important issue worth noting here is that each client is running *Bouncing Balls* locally. When some other client in the group fires a ball, information about its position, color, speed, and initial

angle is sent to the central server. The central server then passes along the information to all the clients. When a client receives the information, it immediately paints a new ball on the local screen (at the specified position). Ball positions thereafter are calculated only at the local level. In other words, after the initial passing of information about each new ball, there is no more data transmission between the client and the server.

The obvious advantage for restricting most of the activities to the local level is speed. If we were to perform everything at the server side – for example, to calculate new positions of all the balls (at approximately 10 times a second), pass them to the clients, then have each client draw all the balls locally – it would simply take too long.

One major side effect of running things locally is that once the balls are fired off, each client then completes the rest of the process independently of other clients. Due to different processor speed, connection speed, etc., clients will invariably run at different speed. In other words, other than the firing of the balls, the clients are actually running the game off-sync from one another.

The drawback of running *Bouncing Balls* off-sync is that the game can no longer serve as a rhythmic machine. Rhythm is inseparable from timing. Now that each client is running the game independently, everyone will hear the bouncing of the balls at different times. The multiple-player version of *Bouncing Balls* is therefore not suited for creating collaborative (poly-)rhythmic improvisations. However, it is still an exciting environment for making sound-collages and “tone-painting.”

In *The Internet Pong Game* which we will discuss below, the server becomes an active component of the game itself. All the ball positions, pedal positions, etc. are calculated on the server level, then passed onto the clients. We will see how it differs from this game.

5. CREATING AND MORPHING A MUSIC STREAM

This game model provides users with tools to create from scratch or to alter a given music stream. Suppose a user is handed a stream of music and a box of tools with which to modify the stream, and is given the instruction to play with the stream so that “the end product will best reflect your personality.” What will he do?

The philosophy behind this model is to provide users with as much freedom as possible, and to retain their individualism in the final product. Users are given very simple tools (which anyone can master without much practice). They are then encouraged to play around with the tools and to express themselves in any manner they desire. The final version of their work will stand as is, and remain recognizable to their respective creators.

5.1 *Draw Music*

Draw Music is written principally by Jeanne Yu and myself. In attempt to make the game as simple and less restrictive as possible, we allowed the users with only one tool – the mouse. The entire game can be navigated through and controlled by simple mouse movements.

5.1.1 General Description of the Game

Draw Music resembles a traditional drawing program. The difference here is that the user can *listen to* what he has drawn by clicking the *play* button at any moment during the game. The music produced will correspond directly to his drawing in the drawing window. The user is literally “drawing music,” and hence the name of the game.

Here are the menus and buttons provided by *Draw Music*:

[Pull-down Menu]	Color:	red
		yellow
		blue
		cyan

[Pull-down Menu] **Mode:** chromatic
 major
 minor
 Chinese
 Japanese
 whole-tone

[Pull-down Menu] **Line Type:** simple line
 quantized

[Button] **Play**

[Button] **Pause**

[Button] **Reset Play**

[Button] **Clear**

The user is instructed to use the mouse to draw lines and curves anywhere in the window provided. There are four colors to choose from. The restriction is: for each color, there can only be one y-value for any given x-value. In other words, nowhere in the drawing plane can a perfect vertical line (of the same color) ever appear.

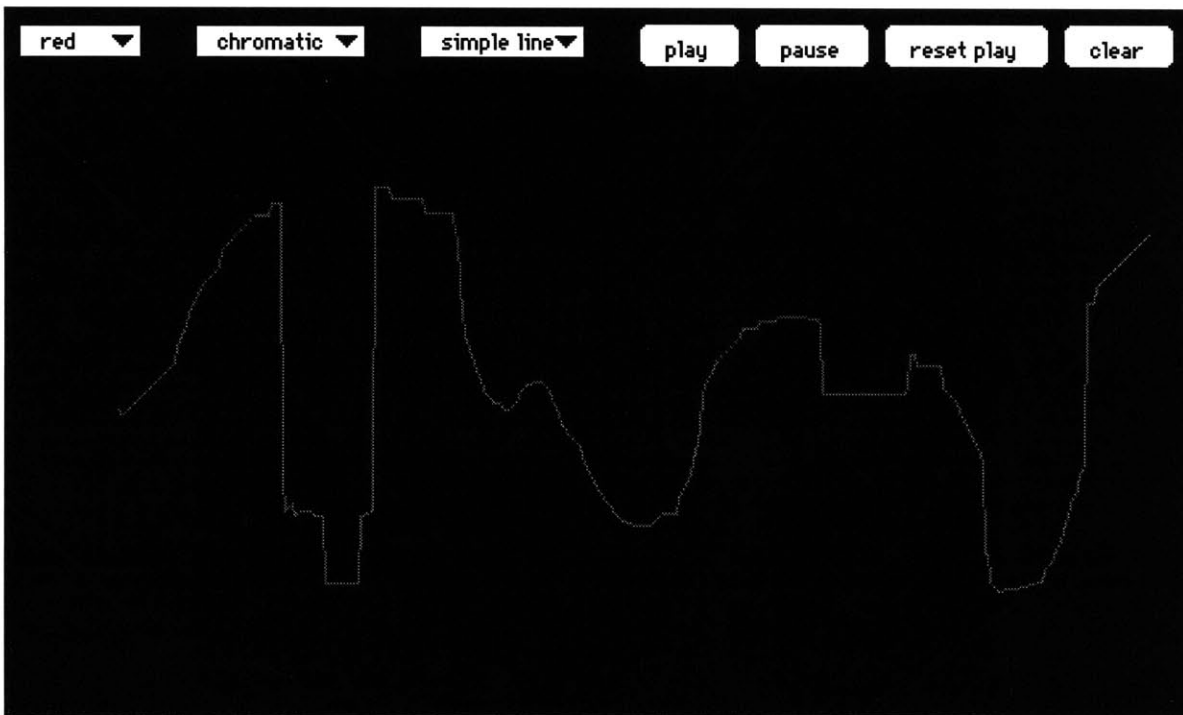


Figure 7: *Draw Music* in action – a melody is drawn.

If the user drags the mouse to a section where some other point of the same color already exists on the same x-coordinate, and if the new mouse position has a different y-value as the previously-drawn point, then the program will automatically erase the previous point, and “paint over” a new point at the new y position.

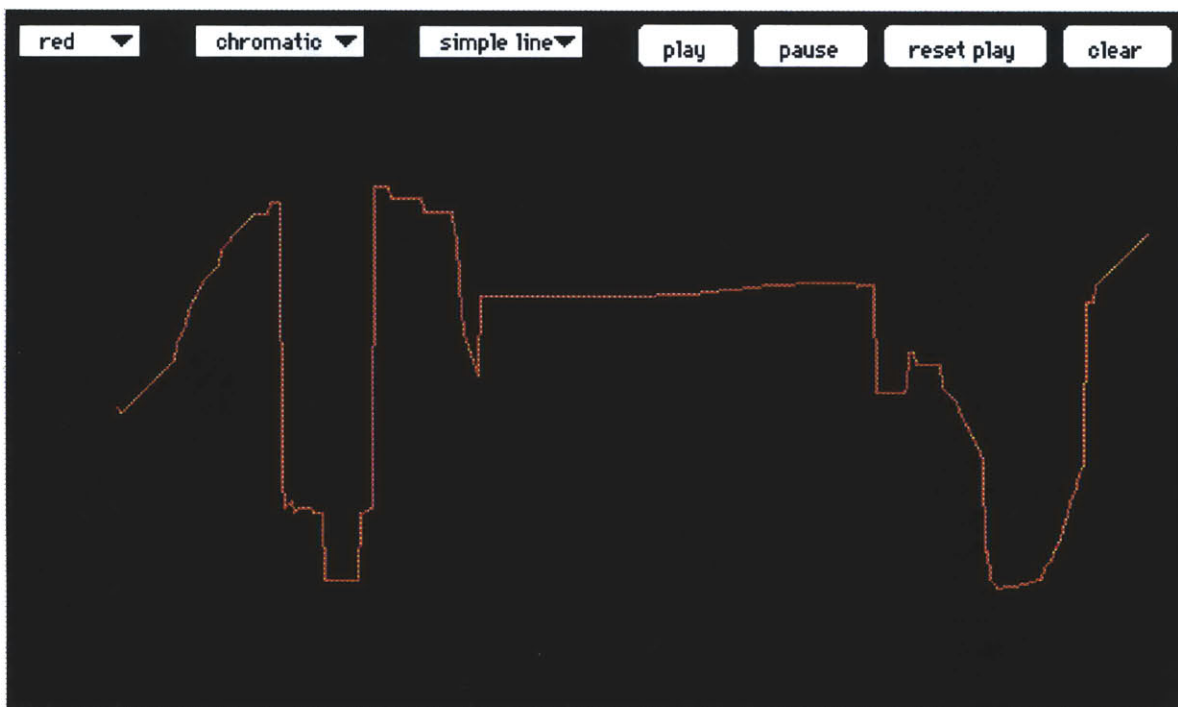


Figure 8: *Draw Music*. When the mouse drags over an area that already contains some drawing (in this illustration, the mouse drags horizontally over the mid-region of the drawing area), the old segment will be replaced by the new.

Note again that this “one y-value for any given x-value” rule applies only to drawings of the same color. Drawings of different colors are not restricted by this rule. There are four colors from which the user may choose. He can choose any color at any point of the game. This means that at any given x-

value, there can be at most four y-values – but each of these points must be of a different color from the other points.

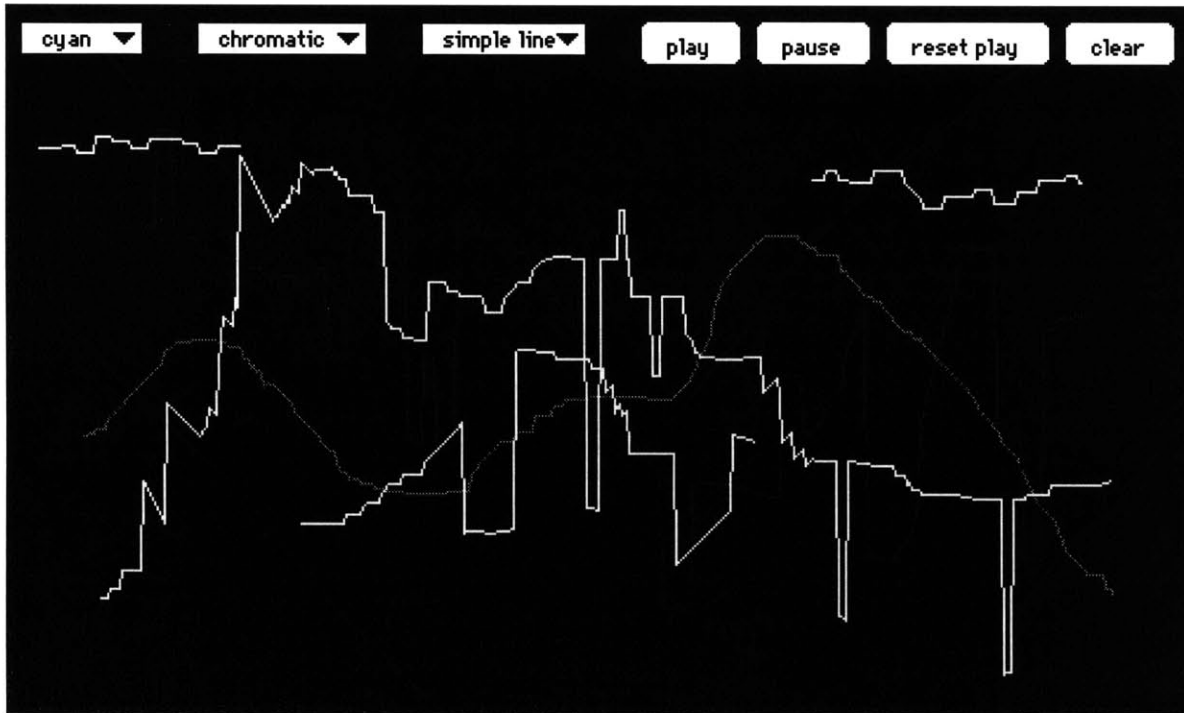


Figure 9: *Draw Music* – all four voices are drawn.

While the mouse is down, if the user drags it above or below the drawing area (i.e., the y-value is out of the vertical range), then the program will erase all points (of the same color) which have the same x-values as the section that was just “dragged over.” This design provides the user with a simple and intuitive “erase” function, and allows him to insert rests (breaks) in between the left and the right margins, as well as to modify his drawing.

By allowing for up to four points of different colors to co-exist at any x-value, the program essentially supports four-voice polyphony. Each drawing color is considered a “voice.” All of the voices have the same range to draw from. In other words, the traditional *soprano*, *alto*, *tenor*, and *bass* division is not

followed here.

When the user decides to “listen to” what he has drawn (by clicking the *play* button), the program will “play the drawing” from left to right at a constant, preset speed. Visually, for each of the four colors, a small white ball will appear at the left-most point of the left-most line/curve. When it is time to play those points, the white balls will start travelling across the screen along those segments at a constant pace. Of course, the music being played corresponds directly to the y-values of the segments, and the positions of the balls travelling across the screen also correspond directly to the music being played. Graphically, the “higher” the point is in the drawing (i.e., the greater the y-value), the higher the pitch of the sound; the “lower” the point, the lower the sound.

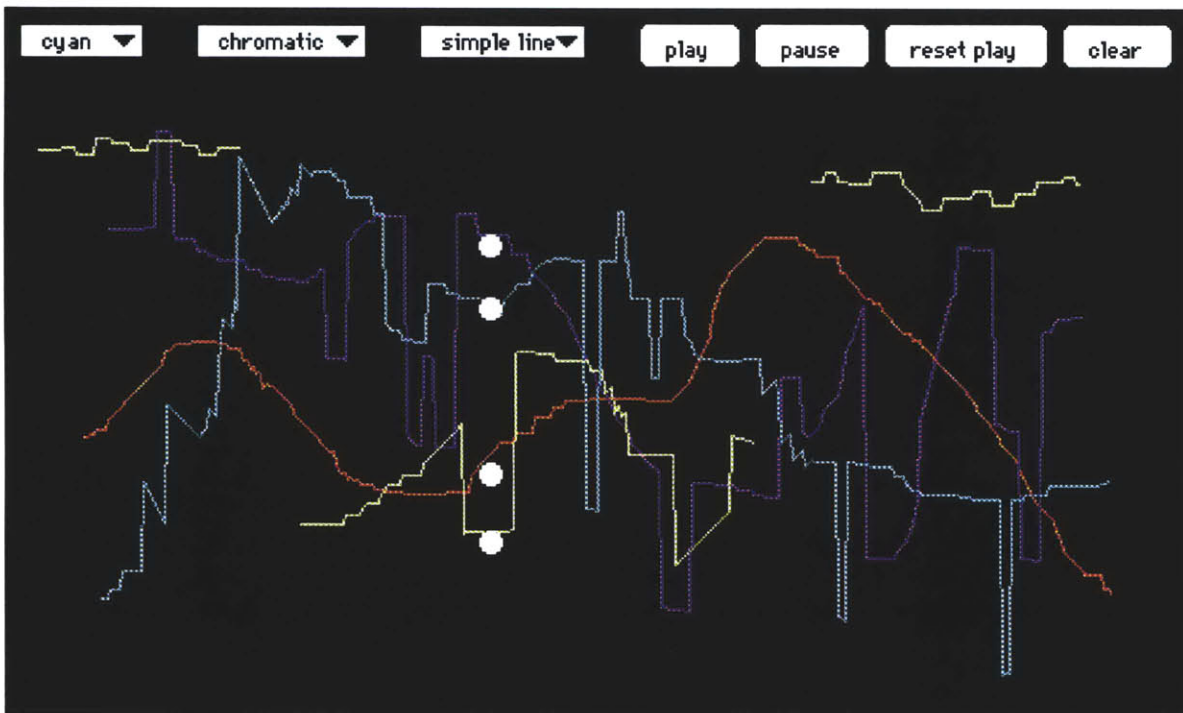


Figure 10: *Draw Music* playing the drawing.

While the drawing is being “played,” the user may wish to stop the playing. This can be accomplished

by clicking the *pause* button. If he wishes to start the piece over, he can *reset play* the drawing. If the user wants to get rid of the entire drawing, he can also *clear* the drawing board, and start his drawing all over again.

The other two pull-down menus *Mode* and *Line Type* will be discussed in detail in the following section.

5.2.2 Preset Modes and Quantization

The default mode for the program is *chromatic*. Without specifically selecting another mode, the user will be drawing in the chromatic scale. If the user wishes to, he may set his drawing to a number of different preset modes. These modes are: chromatic, major, minor, Chinese, Japanese, and whole-tone. (The Chinese and Japanese modes are two different pentatonic scales with different intervals in between the member notes. Of course both traditional Chinese and Japanese music use a number of different pentatonic scales too. Here only one is selected for each label.) The specific note members in each mode are:

- chromatic:** C, C#, D, D#, E, F, F#, G, G#, A, A#, B
- major:** C, D, E, F, G, A, B
- minor:** C, D, D#/Eb, F, G, G#/Ab, B
- Chinese:** C, D, E, G, A
- Japanese:** C, C#/Db, F, G, G#/Ab
- whole-tone:** C, D, E, F#, G#, A#

By selecting different modes, the same drawing of lines and curves will automatically be transformed into notes belonging to member-notes of that mode. For example, suppose an ascending line starts from E₄, travels through F₄, F#₄, G₄, G#₄, and ends at A₄. Under the “Chinese mode,” those notes will be converted to E₄, G₄, and A₄; and under the “whole-tone mode,” the same notes will be converted to E₄, F#₄, and G#₄.

The default *line type* for the program is *simple line*. When *simple line* is selected, all the lines and curves drawn will appear as smooth, connected one-pixel dots in their respective colors. An alternative *line type* is *quantized*, which quantizes the drawn lines and curves into note-regions which they occupy. Using the example from above, suppose there is an ascending line which starts at E4, travels through F4, F#4, G4, G#4, and ends at A4. The quantized chromatic representation will be ascending blocks of rectangles each occupying appropriate spaces which correspond to E4, F4, F#4, G4, G#4, and A4. If instead of *chromatic*, the user is in the “major mode,” then the blocks of rectangles will be occupying spaces which correspond to E4, F4, G4, and A4.

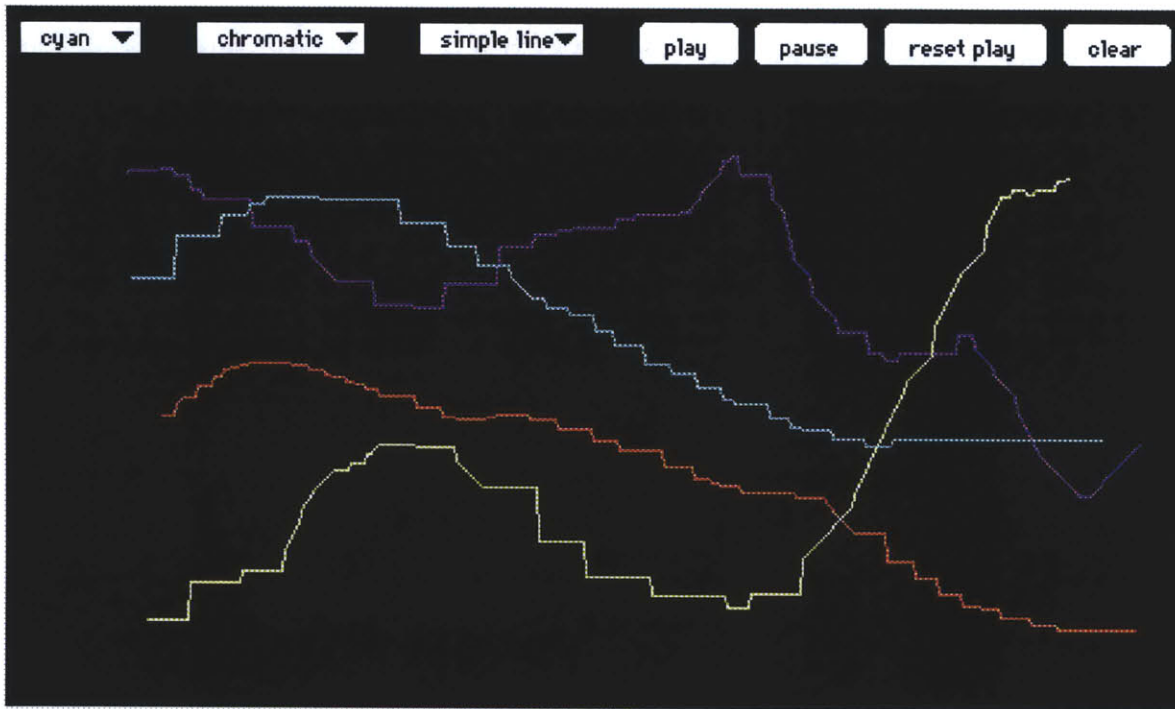


Figure 11: *Draw Music* in simple-lines.

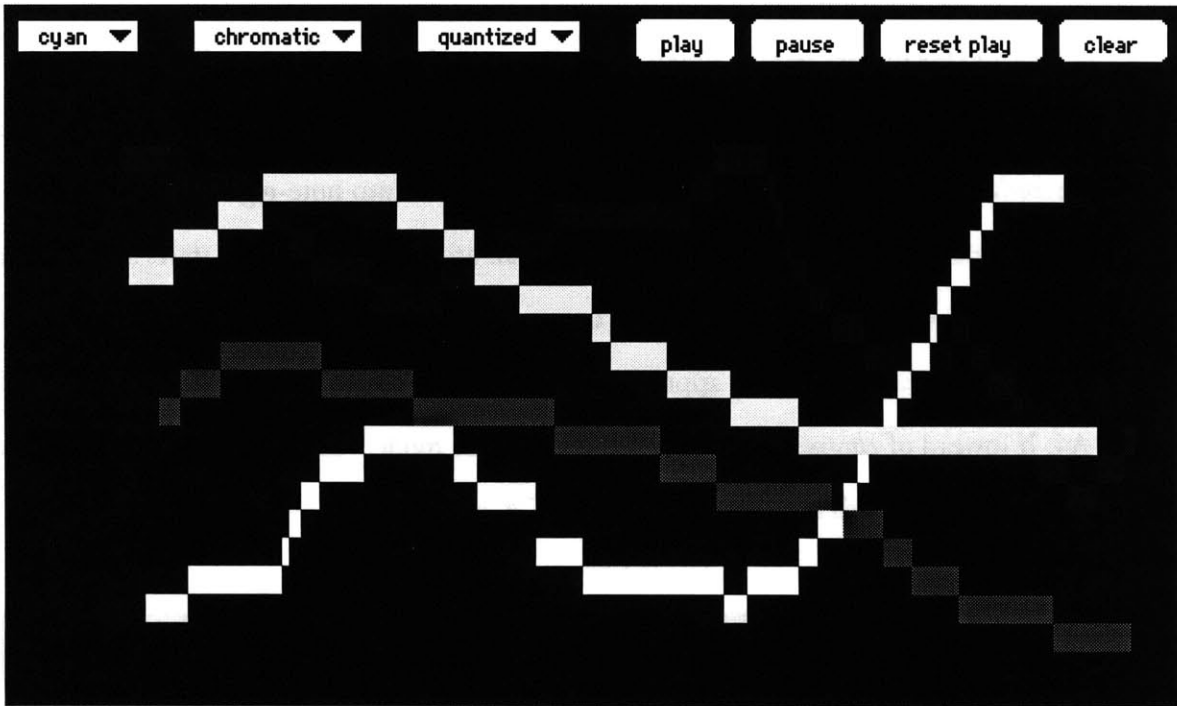


Figure 12: same drawing as Figure 11, but in quantized format and chromatic mode.

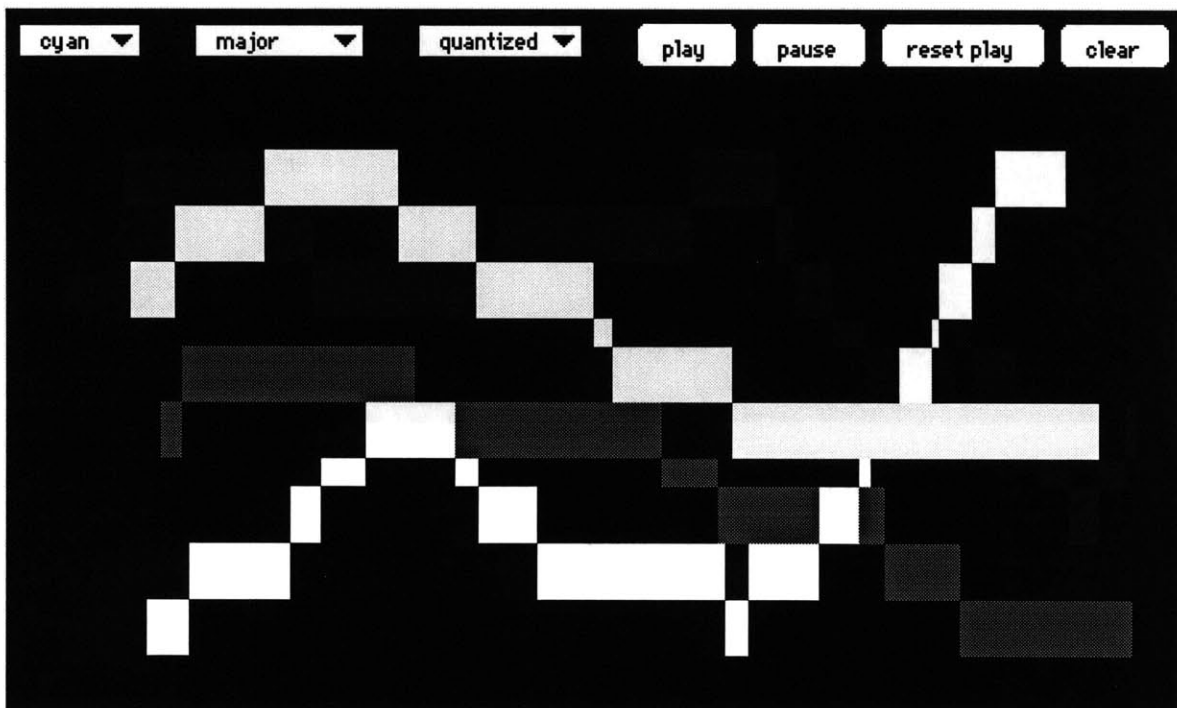


Figure 13: same drawing as Figure 11, but in quantized format and major mode.

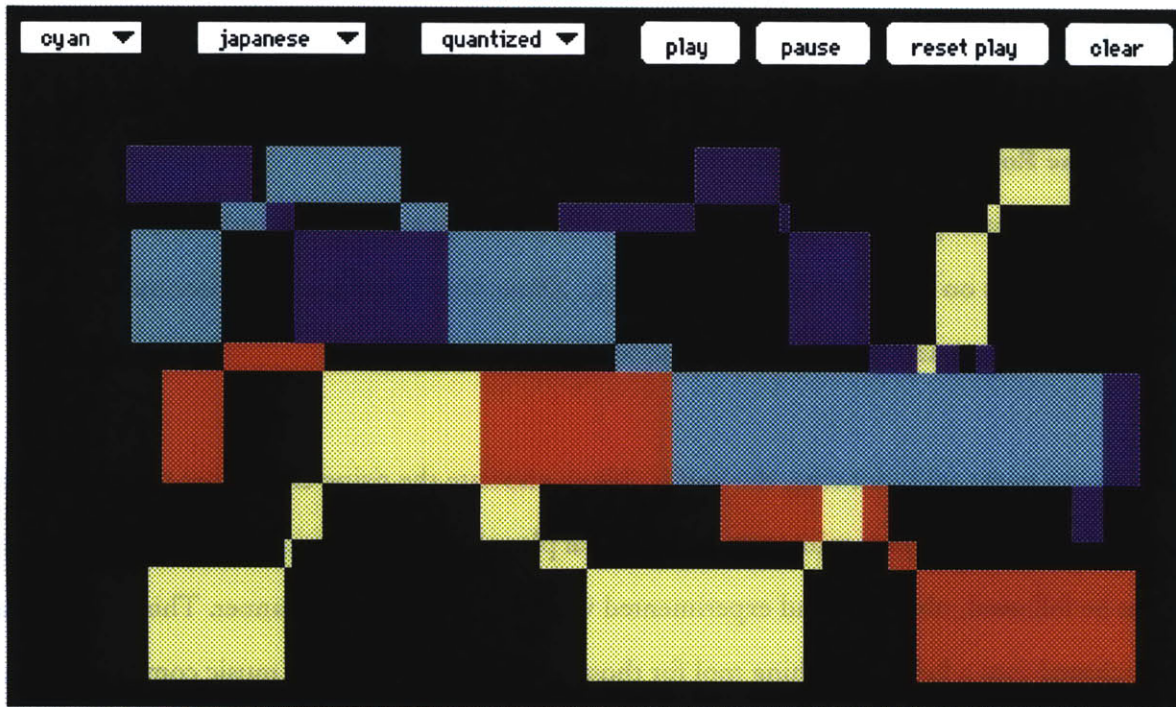


Figure 14: same drawing as Figure 11, but in quantized format and Japanese mode. Note the two major-third leaps in the Japanese scale are clearly represented here by thicker rectangles.

One of the unique qualities about *Draw Music* is that it provides a visual representation of the quantization process taking place in the music part of the program. Thus, the user is able to see exactly how the quantization works, and observe the interval relationships between the member notes in each mode. In the minor mode, for example, the step from G#/Ab to B is greater than the step from C to D or from E to F. Under *quantized* line drawing, these relationships will be shown graphically, and will become clear to the users.

Another unique quality about *Draw Music* is that it is very easy to switch back and forth from one mode to another. After a user finishes the drawing, the drawing is “preserved.” Selecting different modes alters the musical make-up of the piece, but the drawing itself is saved and stored internally. Thus, with a simple click of the button, the user is able to transform his drawing from one mode to

another, as well as to always revert back to the original by selecting the mode with which he started the drawing.

5.2 The Role of Music in *Draw Music*

In *Draw Music*, making music becomes the focal point of the game. In contrast to *Bouncing Balls*, which concentrates on the *rhythmic* aspect of music, *Draw Music* provides an environment in which the users can focus on and explore the *melodic* possibilities in music.

By combining music with drawing, the user is able to associate the rising and falling of pitches with the vertical placement of points in a two-dimensional plane. The melodic contour of a piece of music can thus be followed, observed, and experimented with in a very tangible manner. This program can be re-packaged into a helpful teaching tool for those wanting to learn about music composing or just music in general.

In keeping with the “hands-off” philosophy, I believe this program has just the right amount of guidance and/or restriction. The preset modes are probably more helpful than restricting. And, the user can always reset the mode to *chromatic* if he finds the preset modes to be too limiting.

A possible improvement to the current version of *Draw Music* might be to allow for a library of user-definable modes. That way, in addition to those that already exist, users can also experiment with modes of their own creation.

5.3 Extending *Draw Music*: Other Possibilities

The three extensions of *Draw Music* that naturally comes to mind are: a sequential compounded-morphing program, a sequential segmented-morphing program, and a theme-and-variations morphing program. “Morphing” here is defined as “changing, modifying, or re-shaping of an existing stream.”

In all three extensions, users are given an existing stream of music, and are asked to modify the stream

using the tools with which they are given.

Other than morphing, *Draw Music* can also be turned into a multiple-player game, in where players can each be responsible for a color (voice), and together compose a piece of music.

5.3.1 Sequential Compounded-Morphing

In the sequential compounded-morphing section, I will propose two mid-scale performance games. However, the possibilities here are truly endless. The first game is a straightforward morphing game, where each user is given a set amount of time to modify a given stream, and the stream is passed down sequentially from one user to the next. At the time of the performance, everyone will hear the final version – a piece now containing all of the modifications made by everyone (one on top of another, thus “compounded”), as well as the collective creativity of all of the participants. After the performance, each user can then play back all the stages of the transformation, and see what each player has contributed to the final version of the stream.

A variation of this game is a modified *musical telephone* game. In this game, a music stream is played; the first player is asked to best recreate what he has just heard. When he is done, the player will pass down his rendering to the user after him. The new player will then hear the rendering of his predecessor, and be instructed to, again, best recreate what he has just heard. This process will continue until everyone has had his turn. At the performance, the original stream will be played, followed by the first rendering, second rendering, third rendering, etc. This way, everyone can hear how the stream is altered from one user to the next, and how see how degenerated it became in the last rendering.

5.3.2 Sequential Segmented-Morphing: The *William Tell* Experience

The difference between sequential segmented-morphing and sequential compounded-morphing is that in sequential segmented-morphing, each game participant is assigned a segment of music; all of the morphing processes are independent of the other processes. In sequential compounded-mor-

phing, each morphing process depends on all of the previous morphing processes combined up to that point.

Let function $f(x)$ be the morphing process. Sequential compounded-morphing can be represented as:

$$y_{\text{compounded}} = f(f(f(f(\dots f(x) \dots))))$$

where x is the stream in its entirety.

Sequential segmented-morphing is represented differently:

$$y_{\text{segmented}} = f(x_1) + f(x_2) + f(x_3) + f(x_4) + f(x_5) + \dots + f(x_{\text{last}}),$$

where x_1 is segment 1 of stream x , x_2 is segment 2 of stream x , and so on, and operator “+” denotes a simple concatenation operation.

For the *Performance* class, one of the final projects led by Prof. Tod Machover involves a performance experience which embodies the sequential segmented-morphing game plan. In this project, a section from the MIDI version of Rossini’s *William Tell Overture* is further divided into 10 equal sections. Each of the ten participants is handed a section and asked to “do anything you want with it” using Performer – a popular sequencer and notation software by Mark of the Unicorn, Inc. Each participant is given a limited amount of time to complete his portion of the project. At the end, all sections are concatenated in the original order, and the new (and improved!) version of the *William Tell Overture* is played to the public at the Music Garden.

The *William Tell* experiment turned out to be a positive and encouraging experience. It was received enthusiastically by the participants, the class, as well as by most of the innocent audience who just happened to be in the Music Garden at the time.

Part of the success can be attributed to the great popularity of the *William Tell Overture* itself. The tune is exciting, well-recognized, and well-liked. Also, all of the sections remain untouched after the

participants have completed their respective portions of the project. Therefore, the final piece retained a great deal of individuality, which in turn makes the piece so well received.

An Internet version of such an experience can definitely be implemented and carried out. Like the *Performance* class project, ten or a dozen users can sign up for morphing a piece of familiar-sounding music. They will each be given a section of the piece, and asked to morph the section using *Draw Music*. (Notice that a user can make small changes here and there in the stream, or – if he so desires – wipe out the entire stream and compose from scratch!)

After all the participants have sent back their work, the sections will be concatenated in the original order. At the pre-announced time, the new version of the piece will be broadcasted over the Internet through *RealAudio* or *StreamWorks*. Participants can listen to the piece at their terminal, and communicate remotely with other participants afterwards to share their ideas and thoughts about the project.

5.3.3 Theme-and-Variations

The theme-and-variations extension is designed for a group of users as well. In this model, each user is given the same thematic material and asked to create a variation based on the theme. All of their output will be linked together at the final performance. A theme-and-variations piece can thus be collaboratively composed and performed.

5.3.4 Multiple-Players

A multiple-player version of *Draw Music* is also available. In this version, up to four users can participate in the same game. Each user is responsible for a color, and can draw only with that color. At any point of the game, any participant may click the *play* button to “hear the drawing.” Given the constraints, users may wish to divide up the tasks so that each participant needs to concentrate only on one region (register) of the drawing board. For example, Participant A may want to “take care” of the bass notes, Participant B may want to concentrate on the accompaniment figure, while Participants C

and D may want to work on the principal and secondary voices in the upper registers. These are only suggestions, however. In reality, the participants are free to do anything they desire on the drawing board.

6. *The Internet Pong Game*

The Internet Pong Game is implemented principally by myself. This game was originally intended for a large number of participants. Due to technical limitations, the current version supports only six simultaneous users, but can be modified to accommodate more players. These limitations will be discussed below.

6.1 Overview of the Game

When a user logs onto the game, he will be shown a list of available teams (i.e., teams so far have not been chosen by other users), and will be asked to select a team for which he would like to play. The teams are:

- Tempo Up
- Tempo Down
- Volume Up
- Volume Down
- Transpose Up
- Transpose Down

The current version of *The Internet Pong Game* only allows for one player per team. Once the user makes his selection, he will be matched up with a pre-determined rivalry team (*Tempo Up* will play against *Tempo Down*, *Volume Up* against *Volume Down*, etc.). Together, the two teams will play a game of *Pong*.

On the screen, each player will see his own team on the bottom of the screen, and the rivalry team near the top of the screen. The other pairs of teams will not be shown to him.

The *Pong* game itself resembles the *Bouncing Balls* somewhat. A ball travels up and down the screen with a preset speed. Each team (one on the top of the screen, the other near the bottom of the screen)

is given one horizontal block with which to block the ball. The blocks are placed a quarter of an inch away from the horizontal boundaries (the top one a quarter of an inch below the boundary, the bottom one above the boundary). The blocks can be controlled through the keyboard; however, they can move only horizontally (left and right), and not vertically. The object of the game is to “guard” the horizontal boundary from being touched by the ball. In other words, each team is to successfully block the ball every time it comes close to the boundary so that the ball will not go past the block and touch the boundary.

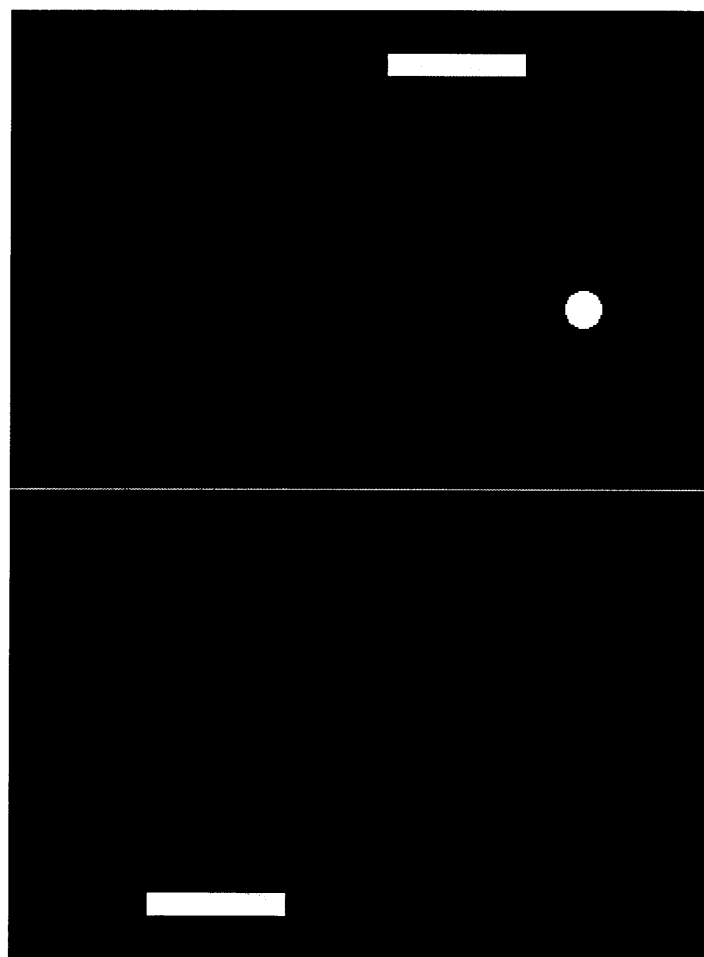


Figure 15: *The Internet Pong Game*.

If the ball is successfully blocked, it bounces off to the other side. The game continues. If a team

“misses,” or, allows a ball to touch its side of the horizontal boundary, then the other team scores a point. No point is awarded for each successful block of the ball

When the game starts, a piece of MIDI music will be played and broadcasted to everyone via the *RealAudio* server. The players will in turn collectively “control” the MIDI parameters: depending on the score each team receives, the corresponding MIDI parameter will be reset accordingly. The central server keeps track of which teams are winning, and by how much. The data are then sent as parameter-altering messages to another program playing the MIDI sequence. The effect of these changes in the scores will be broadcasted and heard by all the participants. For a detailed discussion on broadcasting time-based information over the Internet, please refer to Chapter 7 of this paper.

Let us now take a closer look at the effect of score changes on music. Suppose team *Tempo Up* is currently winning. The tempo of the on-going piece will become faster. The more points team *Tempo Up* is winning by, the faster the tempo becomes. Suppose later on team *Tempo Down* catches up – each time the score difference diminishes, so will the effect of the winning team. Finally, if *Tempo Down* becomes the winning team, the tempo of the piece will become slower.

Similar rules apply to the *Volume* and *Transposition* teams. Instead of *tempo*, they will be controlling *volume* and *transposition* respectively. If team *Volume Up* is winning, the piece becomes louder, otherwise softer; and if team *Transpose Up* is winning, the piece transposes up by a half step, otherwise down a half step.

6.2 The Referee

The *Internet Pong Game* is set up so that there is a referee program running at the server at all times. After setting up rivalry teams, the referee starts to keep track of the positions of the ball and the blocks. It then sends messages to all the clients so that the clients can know the positions of the objects in the game field and update them accordingly.

6.2.1 Broadcasting Messages

Here is a look at how the ball positions are sent using the *WebStar* package:

```
public void run() {
    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
    sc.start(getDocumentBase().getHost(), 8121);

    while (kicker != null) {
        repaint();
        String msg = "";
        msg = "position" + " " + bouncing.x + " " + bouncing.y;
        sc.send(msg);
        try {Thread.sleep(333);} catch (InterruptedException e){}
    }
    [...]
}
```

The ball positions are taken three times a second. The clients also update the screen three times a second. The object “sc” is declared to be of type *StarClient*. The `sc.send` method is called again and again throughout the program. “Send” is defined in *StarClient.java* as follows:

```
/* Sends out a message */
public synchronized void send(String outMsg) {
    if(starID < 0) return;
    /* No anonymous message allowed -> needs a valid starID */
    try {
        output.writeBytes(starIDstr);
        output.writeBytes(outMsg);
        output.writeByte('\n');
    } catch (IOException e) {
        owner.WS_debugLog("Error while sending: " + outMsg, this);
    }
}
```

This routine allows messages to be “broadcasted” from the server to all connected clients. The clients

can then decide for themselves what to do with the information received. (In this program, the clients repaint the ball and blocks at their respective new positions everytime a message is received.)

6.2.2 Bouncing Angles

To keep things simple, the angle at which the ball bounces off the blocks is set to be:

$$\text{angle of reflection} = 90 - (\text{incoming angle} - 90)$$

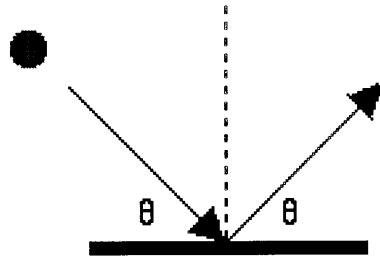


Figure 16: the angle at which the ball is reflected in the *Internet Pong Game*.

6.3 Technical Limitations

The Internet Pong Game is a game in which timing is crucial. The necessity to synchronize every move in the game turns out to be a great limitation. Unlike the multiple-player version of *Bouncing Balls*, where most of the calculations can be performed locally at the client side, *The Internet Pong Game* needs to calculate everything at the server level. To ensure that both teams see the same objects on their respective monitors, the positions of the ball (which is constantly travelling), as well as the positions of the blocks, need to be calculated on the server side, and then passed onto all connected clients. When the clients receive the data, they can then repaint the ball and the blocks as local processes.

If a computer has a fast processor, and the connection speed is extremely fast, the ball positions can be

calculated more frequently; the ball movement will still look fairly smooth. However, if the connection is slow, the ball movement can look intolerably jumpy. In order to accommodate most possible connections, the update rate (the rate at which the positions of the ball and blocks are updated) is set at 3 times a second. Already, at this rate, the animation becomes slow and awkward. It is hard to imagine how unbearably slow the game will become had more objects – balls, players, etc. – been added to the game field. Although this game does not look very appealing to the eye when it runs, we have continued to develop it as an exploration of possible game models for interactive Internet music. As far as we know, there is no true real-time interactive games out there which supports multiple players like *The Internet Pong Game*. Perhaps when the technology catches up, more similar games will be developed.

6.4 MIDI, not .au!

Another breakthrough in *The Internet Pong Game* is the use of MIDI. With its modest bandwidth requirements compared to other audio formats, MIDI is a natural choice for adding a soundtrack to Web pages. Unfortunately, Java suffers from lack of built-in MIDI support. At this point, adding MIDI support to Java requires platform-dependent code, native methods in Java parlance. This means that anyone who wishes to use a Java MIDI applet will have to first download a platform-dependent native method library. Not only is this highly inconvenient to both programmers and application users, it also prevents Java from being truly platform-independent and portable as the language is designed to be. Java's security feature also prevents a Java applet from downloading a MIDI library automatically [St. Hippolyte, 96]. We believe that in order for Java MIDI to gain widespread acceptance, a native library must be compiled and made available to the public; most importantly, it must follow an open standard.

If Java does not support MIDI, why then is the game designed and implemented to manipulate MIDI streams? Through hours of net search and surfing, we located a Java MIDI package written by Michael St. Hippolyte (for Windows). According to the author, "this Java MIDI package only sup-

ports output, does not support System Exclusive messages, and does not have built-in sequencing or MIDI file support – yet.” But it serves our needs. We installed this package, tested it, and decided to – for this game anyway – abandon .au files, go with MIDI, and use *RealAudio* and/or *StreamWorks* to broadcast the MIDI output over the Internet.

6.5 Linking *The Internet Pong Game* to *Sharle*: John Yu’s Computer Music Generator

The natural question that follows is, “where would the music come from? How is it produced” To take advantage of the new MIDI package, we wanted a program that would generate MIDI output – preferably parametrically – so that the scores of each team can be directly linked to the parametric controllers of the program.

In a parametric music generation system, various musical components are identified, selected, and treated as separate, but related, entities. An examples of parametric music generation system is Alexander Rigopulos’ *Seed-Based Music Generation System*, in which music is defined in terms of a set of parameters (e.g., activity, coloration, etc.). Thus, by directly controlling these parameters, the user is able to modify the output of the system [Rigopulos, 93]. Another example is Fumiaki Matsumoto’s *Drum-Boy* system, where users are given a set of high-level rhythmic controls with descriptive labels like “stuttering vs. floating,” “energetic vs. calm,” and “graceful vs. mechanical.” Users can manipulate these controls so as to create rhythmic patterns of their own liking [Matsumoto, 93].

John Yu’s computer music generator is in many ways similar to Rigopulos’ and Matsumoto’s systems. John describes his system as “[a] computer composition engine [that] has been designed in an attempt to capture basic music composition and improvisation knowledge in a variety of styles.” The output of this system is based solely on user-controlled parameters and low-level rules embedded within the generation engine [Yu, 96].

John’s system – nicknamed *Sharle* – does not deal with chords and harmony; rather, it concentrates on

the note-occurrence frequency in each mode, and produces voices of seemingly independent melodic lines. Because of the voice independency, the music produced by this system reminds one somewhat of Bach's music. (That is *not* to say, however, that the music produced here is fugal or contrapuntal in nature.) If all the parameters of the system remain unchanged for a period of time, the music can sound aimless and static. But if the parameters change frequently enough, the music becomes surprisingly lively.

After careful evaluations and joyful discussions with John, we found John's system most suited for *The Internet Pong Game*. Aside from the technical aspect of the system, the music it produces is also interesting and refreshing to listen to, and at the same time "mellow" enough not to be too distracting. (In fact, John sometimes refers to *Sharle* as a "background-music generator.") We therefore decided to go with John's system, and set up a message-sending channel between *The Internet Pong Game's* central server and *Sharle*. After the link is set up, the parameter values collected by the server from all the clients will be sent directly to *Sharle*, and *Sharle's* parameter controls will then be modified in accordance with the values received.

6.6 I Just Want to Win!

An unique aspect of *The Internet Pong Game* is that making music is simply *not* a top priority to the game players. Players join the game to *win* their respective matches; they do not – nor are they expected to – focus on the musical qualities of the piece which they are co-composing and co-performing. To the players, the music being produced is merely a by-product of their matches, and certainly not the focal point of the game.

This type of music composition / production is worth exploring. Traditionally, a piece of music is through-composed, or at least through-designed. Even in "chance composition," for example, instructions are still being given to the performer(s) by the composer. And in the more modern writings where performers are allowed even greater freedom, composers still design the overall structure of the

music, and often through-compose the sequences of events to be “randomly selected” or triggered by performers at the time of the performance.

In *The Internet Pong Game* style of music production, however, the game players *are* the composers and the performers. The development of the piece rests entirely in the hands of the game participants. But since game players are concerned with *winning* the matches, not the beauty of the music, *The Internet Pong Game* can be said to be a truly *anarchic* composition-performance system.

6.7 A Big Happy Family

The Internet Pong Game gives rise to the sense of the “Internet Community.” In the most obvious sense – the opponent against whom one is playing comes from somewhere in this Internet Community. Furthermore, every time the music changes, a responsive user can pick it up and come to the conclusion that “someone out there just scored a point.” So far, with the exception of *The Internet Pong Game*, there is almost no synchronized, graphic-based real-time Internet game out there in which people can participate in. I believe *The Internet Pong Game* is revolutionary in this regard as well.

6.8 Possible Expansions

Of course, some of these expansions can not be implemented effectively until the technology catches up. Nevertheless, the following are some suggestions worth considering.

6.8.1 More Parameters

For a more sophisticated model (not implemented), other musical and/or extra-musical elements can be considered. These elements can be controlled by pairs, triplets, or even groups of teams. For example, the pair of *melodic contour* teams could be playing for the “wildness” or “peacefulness” of the melody, while the *chord quality* teams could be controlling the number of major, minor, augmented, and diminished chords being played in the on-going piece, and so forth. It is precisely this constantly-

changing combination of different musical elements that makes each music performance so unique and fascinating.

6.8.2 *Für Elise*: Linking to Famous Pieces

Another possibility to make *The Internet Pong Game* more fun would be to link it to famous pieces such as Beethoven's *Für Elise*, Mozart's *Eine Kleine Nachtmusik*, or J.S. Bach's *Two-Part Inventions*.

Audience generally enjoy hearing pieces that they are familiar with. Seeing how their game scores are affecting the playing of a familiar piece of music can really make the game participants go wild! Judging from how the audience reacted to the *William Tell Overture* experiment, I believe that linking *The Internet Pong Game* to famous pieces can be another great success story.

7. BROADCASTING LIVE MUSIC WITH *RealAudio* AND *StreamWorks*

RealAudio, by Progressive Networks, and *StreamWorks*, by Xing Technology Corporation, are the two most prominent products (as of August 1996) designed to handle continuous time-based information over the Internet. These two products make live broadcasting of music and/or video possible. In our projects, both *The Internet Pong Game* and the *Internet Hyperinstrument* use this new technology to broadcast music. In the following sections, we will first discuss problems in delivering time-based information over the Internet, then *RealAudio* and *StreamWorks*, and finally the sound-quality of the two products.

7.1 Problems in Delivering Time-based Information over the Internet

Traditionally, the Internet uses two basic methods to send data – the TCP protocol, and the UDP protocol. The TCP protocol is more reliable; however, it is slower in speed, with substantial delays at times. The UDP protocol, on the other hand, can transmit data without significant delay, but its occasional loss of packets can be quite problematic in situations where data accuracy cannot be compromised.

Web servers in particular use the HTTP protocol to transmit data. HTTP in turn sends information using the TCP protocol. Like the other protocols, HTTP was not designed for handling time-based information (such as audio). According to *HTTP versus RealAudio – Client-Server Streaming* [RealAudio, 95], problems in using HTTP for transmitting isochronous information include:

1. when web servers are used to send time-based data such as audio that gets rapidly consumed by the listener, the TCP delay problem can really bog things down. “Even a 2 or 3 percent re-transmission rate can bring even a low bit-rate audio data stream over a standard modem (say 8 kilobits/second over a 14.4 modem) to a grinding halt” [RealAudio, 95].

2 since the HTTP protocol is designed for one-way continuous transmission, it does not allow for a user-friendly implementation of desirable features such as “fast forward,” “rewind,” and “search” – features which are characteristic of digital medium such as CD and LD players.

3. Web servers were designed to deliver large blocks of data to a client as quickly as possible; they were, however, designed to handle only a small number of concurrent accesses. For a live broadcasting of over 100 simultaneous connections, for example, ordinary Web servers will most likely crash.

4. the Web server approach is inefficient in using network bandwidth. When downloading a sound file, for example, a Web server will need to transmit the file in its entirety before the user can hear even the first few seconds of the file. (And oftentimes a user is only interested in hearing the first ten or twenty seconds of a file to determine whether or not to continue.) This results in lots of wasted network resources.

7.2 RealAudio

RealAudio is a software product which allows playback of audio information over the Internet in real-time. As explained above, the Internet was not intended for handling continuous time-based (isochronous) information: both TCP and UDP guarantees neither minimum latency nor throughput rate [RealAudio, 95]. Clearly, in order to better handle isochronous data transfer, special designs or strategies need to be employed.

To solve the speed-versus-accuracy problem, *RealAudio* chooses to deliver audio data via UDP (which is considered by most to be “superior” to TCP), and further corrects the packet-loss problem by a loss-correction system which Progressive Networks developed. This system in essence minimizes the impact of any given lost packet and enables the client to “recreate” the missing pieces of the signal. The system “works very well under normal lossless conditions, degrades gracefully when packet loss is in the 2-5% range, and even works acceptably when packet loss is as high as 8-10%” [RealAudio, 95].

RealAudio supports Internet connections of 14.4 kbps or faster. To broadcast audio information live using *RealAudio*, the following components are necessary:

1. the *RealAudio Encoder*: encodes audio information into a special RealAudio format.
2. the *RealAudio Server*: delivers live audio over the Internet (or local network)
3. the *RealAudio Player*: plays files in *RealAudio* format [RealAudio, 95]

The *RealAudio Server* comes in different “packages.” The smallest model supports up to five simultaneous streams, whereas more expensive models can support up to 500+ simultaneous streams. The sound quality of *RealAudio* (*RealAudio Player* version 2.0) is similar to that of FM mono, which we believe is suitable for our purposes.

RealAudio has the following features:

- two algorithms: 14.4 and 28.8 both are proprietary
- new scalable algorithms for better sound quality on ISDN bandwidth
- players available for Macintosh, Windows, and UNIX
- servers are available for Sun, SGI, PC's (486/66 or better), and for Macintoshes (System 7.5 running Open Transport)
- live transmission possible
- new features in *RealAudio 2.0*:
 - better sound quality (AM quality on 14.4 connection, FM mono quality on 28.8)
 - Netscape plug-in
 - Java controllable
 - multi-casting enabled
 - synchronized multimedia: Player can control the download of graphics

The MIDI output of the game – whether from John Yu's computer music generator or altered MIDI

files of famous pieces – is directed to a mixer, then out through the speakers. The audio information is then encoded by the *RealAudio Encoder*, and broadcasted using the *RealAudio Server*. Participants of the game will need to have installed *RealAudio Player*. When a user clicks to listen to the live broadcast, the *RealAudio Player* will play the (current) encoded audio information, enabling him to hear the effects of each point his team earns or loses.

7.3 *StreamWorks*

StreamWorks delivers “streaming” multimedia - pictures, video and sound - based on the MPEG international standards for video and audio compression [Xing, 95]. These streams fall into three categories: audio-only, video-only, and system (video plus audio). *StreamWorks* uses standard TCP-IP network protocols, and takes advantage of new “multicast IP” protocols (RFC 1112) for data delivery.

A *StreamWorks* setup contains the following components [Xing, 95]:

1. *StreamWorks* Network Encoders
 - video + audio
 - audio only
 - file transmitter / encoder emulator
2. *StreamWorks* Network Servers
3. *StreamWorks* Network Clients
 - for PC Windows
 - for X-Windows
 - Stand-alone
4. *StreamWorks* Network Manager

In general, the faster the connection, the better output quality one can expect from *StreamWorks*. With a 9600 baud modem, one can hear audio “similar to a low-powered AM radio station.” Video quality also varies depending on the connection speed. For VCR-like video quality, for example, a T1

connection would be necessary.

StreamWorks has the following features:

- scalable algorithms: 8.5kb to ISDN bandwidth, with auto detection and auto-adaptation to possible bandwidth reduction (network overload)
- uses the International MPEG-1 and MPEG-2 standards for audio and video
- on 28.8 connection, video capabilities = 3 to 5 frames per second
- players exist for Windows, UNIX, and Macintosh (new)
- multicasting enabled
- live transmission possible
- on 14.4: AM quality; 28.8: FM quality

7.4 Comparing Sound Qualities – *RealAudio* versus *StreamWorks*

From a user's point of view, the biggest difference between *StreamWorks* and *RealAudio* in handling audio information is that *StreamWorks* offers different sound qualities for different bit-rates of stream:

Bit Rate	Sound Quality	Radio Equivalent
8.5 kbps (LBR format)	8 khz mono	AM radio quality
24 kbps (an MPEG audio format)	16/22 kHz mono	in between AM and FM radio quality
56 kbps (an MPEG audio format)	32/44 kHz mono, or 16/22 kHz stereo	approximately FM radio quality
112 kbps	used for 44 kHz stereo	better than FM radio quality

In addition, with *StreamWorks*, one can reduce from a higher bit-rate audio stream to a lower one “on-the-fly.”

At the same bandwidths, although *RealAudio* and *StreamWorks* use different algorithms, they have the

same sound quality. *RealAudio* and *StreamWorks* are both scalable to fit the bandwidth, but only *StreamWorks* offers real-time adaptation to possible bandwidth loss.

As for standards, only *StreamWorks* uses the MPEG standard. *RealAudio* uses its proprietary algorithms.

One advantage that *RealAudio* has over *StreamWorks* is that *RealAudio* can be controlled by Java, and can send signals back out to the HTML. This makes Web interaction easier to accomplish.

Choosing the “right” system really depends on the nature of the project (as well as sponsorship from the companies). *StreamWorks* is obviously more flexible, and has the advantage of being able to handle video data. The ability to handle video data might come in handy should the Brain Opera project decide to broadcast live video over the Internet directly from Lincoln Center. If live audio broadcasting is all that is needed, *RealAudio* is a possible alternative. For the current plan of the Brain Opera, I see no significant advantage in choosing one system over the other.

8. BIGGER THAN LIFE: THE *Internet Hyperinstrument*

Prof. Tod Machover started the *Hyperinstrument Group* at the MIT Media Laboratory in 1986, with the goal of designing expanded musical instruments, using technology to give extra power and finesse to virtuosic performers. Such Hyperinstruments were designed to augment guitars and keyboards, percussion and strings, and even conducting, and have been used by some of the world's foremost musicians such as Yo-Yo Ma, the Los Angeles Philharmonic, Peter Gabriel, and members of Pierre Boulez's Ensemble InterContemporain [Machover, 92]. Since 1992, the focus of the Hyperinstrument Group has expanded in an attempt to build sophisticated interactive musical instruments for non-professional musicians, students, and music lovers in general.

Part of this work is focused upon the principle of engaging participants in an active, kinetic relationship with the musical process – be it with composition, interpretation, improvisation, or some new experience which can only exist with the aid of a computer. Though great technological advances have been made in music for sound production, processing, and compositional aids, these tools have far outpaced our ability to control them in musically interesting ways [Machover, 92]. Musical interfaces, whether for experts or novices, must possess a refined physical control and an intuitive link with the sounds they produce. A keyboard key which produces a giant electronic gong, for example, will not “play” as well as an instrument which requires a swing of the arm to produce the same sound. Moreover, as the musical elements we choose to offer for users' real-time control become more abstract – concepts such as tempo, embellishment, or rhythmic activity – the question of interface becomes even more challenging.

Traditionally, a Hyperinstrument consists of a physical instrument modelled after its acoustic counterpart (e.g., Hypercello and cello), sensor devices to measure bow pressure, hand position, etc., a computer for data analysis, and a sound-generating / altering device. In this model, however, the physical instrument component will be substituted by common computer peripherals such as music

keyboard, mouse, joystick, or computer keyboard. A user can log on to the Brain Opera homepage, and start improvising along with the real-time output of others who are also currently “jamming” on the Internet.

A remote “jamming session” of such magnitude could be exciting, but several problems must first be taken care of before the system can work convincingly. An intelligent “conductor program” needs to be written so that not all 100 or 200 instruments will be heard simultaneously at all times. The “conductor” should be able to “spot-light” a soloist or two at a time. It should also keep track of the activity level of all users, and make intelligent decisions as to which streams it wants to broadcast, which streams it wants to dispose of, and which streams it wants to alter / enhance at any given moment.

One advantage of this system is that it creates an environment where the user can jam along with real-time music that he helps to produce, and whatever he is currently producing locally helps to mold the direction of the piece for the next ten or twenty seconds. It resembles the traditional orchestra or choir in that each user is a contributing member to the music performance; he is not “fighting” to be heard, but he knows that whatever he is doing does have a real and immediate effect on the overall music being produced.

8.1 About the *Palette* . . .

The *Internet Hyperinstrument*, nicknamed *The Palette*, is based on John Yu’s *Sharle* system. The Internet Music Team of the Brain Opera, together with John, modified the original system and added a new user-interface to the Java version of *Sharle*.

Based on John Yu’s *Sharle* system, *The Palette* was designed to be the official Internet Hyperinstrument for the Brain Opera. Users can play the instrument from home; the values of their interactions will be sent, in “real-time,” to the main server at the Brain Opera performance site. Together with values collected from other users, these numbers will be converted into controller information, which

will in turn trigger and/or modify musical events in the system.

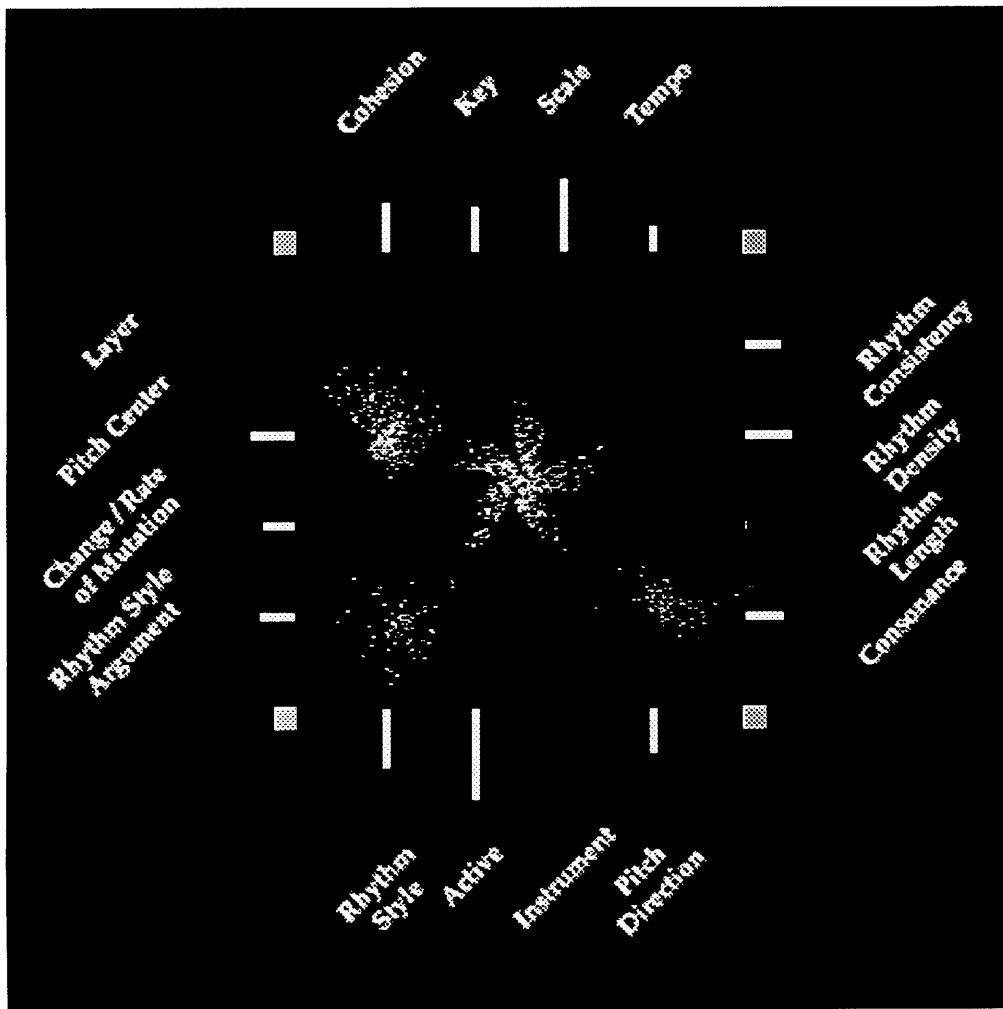


Figure 17: *The Palette*.

In the actual Brain Opera performance, a section of the piece will feature Internet music. Of course, the Brain Opera homepage has continuously been requesting for its visitors to send in sound materials. In this Internet music section, however, everything else takes a back-seat to the music produced by *The Palette* users. In other words, whoever are connected to *The Palette* at the time – from wherever they are – will be spot-lighted at the Brain Opera performance site. In parts of this Internet solo section, certain individuals might be selected and featured; during other parts of the section, all of the Internet input values will be combined and collectively they will produce a coherent stream of music.

Playing *The Palette* is simple. A user simply clicks on the red control bars around the applet, and the music will be modified accordingly. The activities (changes in the controls) of other on-line *Palette* users are represented as the outline of another circle. The user himself is represented as a uniform white circle. This allows each user to see what he is doing in relation to everyone else on-line. The red control bars are laid out as follows [Yu, 96]:

	cohesion	key	scale	tempo	
change of layer					rhythm consistency
pitch center					rhythm density
change/rate of mutation					rhythm length
rhythm style argument					consonance
	rhythm style	activity	instrument	pitch direction	

8.2 About the Controls . . .

Here is a more detailed description of the controls [Yu, 96]:

Cohesion	Causes maximum repetition at full length, and maximum variation at shortest length.
Key	Changes the key (based on its value modulo 12).
Scale	Changes between six distinct settings. From shortest to longest, they are: a major scale, a more consonant major scale, a more consonant minor scale, a minor scale, a pentatonic scale, and a more minor pentatonic scale.
Tempo	The shorter this control is, the faster the tempo.
Rhythm Consistency	The longer this control is, the more regular the rhythm becomes.
Rhythm Density	The shorter this control is, the sparser the rhythm.
Consonance	The shorter this is, the more dissonant the current layer becomes.
Pitch Direction	The longer this is, the more the notes of the current layer will go up in pitch.
Instrument	This changes the instrument used for the current layer. The results depends on the

instrument sounds available.

Active This has two settings, long and short. Short turns the current layer off. Long turns it back on. Layer #0 cannot be turned off.

Rhythm Style One of four transformations on the current layer's rhythm. From shortest to longest, they are:

- (0) a steady beat,
- (1) beats fall between the beats of the normal rhythm,
- (2) a rhythm of a different period, and
- (3) no transformation.

Rhythm Style Argument

Depending on Rhythm Style, this control has the following effects:

- (0) The shorter this is, the faster the steady beat.
- (1) At half length, beats fall exactly halfway between normal beats. At quarter length, beats fall a quarter of the way to the next normal beat. At full length, beats are delayed to the following normal beat.
- (2) The longer this is, the longer the period.
- (3) Not used.

Change/Rate of Mutation

The longer this is, the more the controls will mutate themselves.

Pitch Center This higher this is, the higher the pitches of the current layer.

Layer Changes the current layer to the one displayed in your Java Console. This control can be used to add layers, by adjusting it to maximum length.

As one can see, *Sharle* is a fairly complex system. The *Palette* server takes in control values and sends messages to *Sharle* in the format of P[C | V], where C is the control number (0 through 16), and V is the corresponding new value to be updated. Once *Sharle* receives the new values, it will update the music accordingly.

8.3 The Role of Music in *The Palette*

The Palette is different from other game models in that it is designed for a truly large group of participants. It is conceivable that during the time of Brain Opera performances, hundreds of players world-

wide might join in together to play the game. Note that *timing* for this game is not as critical as in the case of *The Internet Pong Game*. While other users are “represented” in the applet, it is not crucial for a user to keep track of everyone else’s every move, as there is no “winning” or “losing” in this game. Each player is almost like a member of an orchestra: he does what he can to help influence the on-going piece of music; but he is not “important enough” to single-handedly reverse a trend or an unanimous process. Players can still derive satisfaction from playing the game, because he can see exactly what he is doing on the screen, and how he is contributing to the overall shaping of the piece – however big or small his influence might be.

9. USABILITY OF GAMES IN CONCERT PERFORMANCES

Most games discussed in this paper can be adapted for concert performances. The following are some simple suggestions.

9.1 *Bouncing Balls*

Players of *Bouncing Balls* can be grouped into different sections in an ensemble. Two or three players can be in charge of the percussion section of an improvisation session, while another group of players can concentrate on the melodic and harmonic aspects of the music. Of course, there can be different sets of sounds added to the game for those who want to make *Bouncing Balls* a richer instrument. But overall, players / performers of *Bouncing Balls* should take advantage of its repetitive, looping nature, and explore the possibilities of creating syncopations, cross-rhythms, etc. through the use of control settings provided by the game.

9.2 *Draw Music*

Draw Music is a game that emphasizes freedom and individualism. It also calls for its participants to modify or re-create a given stream of music. In order to acquire the best work possible, it is important that each concert experience allows the participants ample time to complete their individual parts. It is also important, in forming the finished product, to concatenate the individual submissions in a cohesive manner.

Consider this scenario: two weeks before a concert, a theme is announced at a Web site. Interested participants are asked to register for the piece, download the theme and the *Draw Music* program, and are given a week to “morph” the theme. When they are done, the registered participants can drop off their work at the Web site from where they downloaded the theme. Then, when all submissions are received, the “super-composer” concatenates all the parts according to the “activity level” of the submissions, and further orchestrates the piece. At the concert, the finished piece is played live on stage

by an orchestra and is simultaneously broadcasted over the Internet for everyone to hear.

This type of collaborative music composing certainly has its traditions. The famous *Hexameron* of 1837 (subtitled *Morceau de Concert - Grandes Variations de Bravoure pour Piano sur la Marche des Puritains de Bellini, composees pour le Concert de Mme la Princesse Belgiojoso au Benefice des pauvres*) is a great example of this type of collaborative music-writing. In this piece, six composers – Liszt, Sigismond Thalberg, Johann Peter Pixis, Henri Herz, Carl Czerny, and Frédéric Chopin – each wrote a variation based on a theme of Bellini. These variations are then put together to form a theme-and-variations piece. Liszt, being the “super-composer” here, not only transcribed the original theme, but also added small sections here and there so as to better unify the piece.

Following the tradition of these great composers, in *Draw Music*, we are using technology to bring together people’s creative talents. Of course, as proposed in Chapter 5 of this paper, there are other possible extensions to *Draw Music*, and each model could easily be turned into a successful concert performance experience. The possibilities are truly endless.

9.3 *The Internet Pong Game and Internet Hyperinstrument*

Both *The Internet Pong Game* and the *Internet Hyperinstrument (The Palette)* use John Yu’s *Sharle* system to generate music. Chapter 8 thoroughly discussed how *The Palette* is to be incorporated and featured in the Brain Opera performances. We are confident that *The Palette* will work well.

The Internet Pong Game, in a way, is a different version of the *Internet Hyperinstrument*. Since both *The Internet Pong Game* and *The Palette* take in control values from clients to modify settings in the *Sharle* system, *The Internet Pong Game* can be thought of as another *Internet Hyperinstrument* – but with a different user interface. (Of course, *The Internet Pong Game* has a different design philosophy behind it, and explores different compositional issues from *The Palette*, but in terms of its relation to the music-generation engine *Sharle*, *The Internet Pong Game* is very much like *The Palette*.) With the

present-day connection speed and bandwidth, however, I believe that *The Internet Pong Game* is not a good choice to be incorporated into a large-scale concert performance. A game like *The Palette* would be a lot more successful, because it does not require fast, continuous updating of positions as in the case of *The Internet Pong Game*.

10. AFTERTHOUGHTS

Since *musique concrète* of the early 1950s, the use of electronics in sound production was introduced to the world of traditional art music. In the mid-1980s, with MIDI and the availability of personal computers, computer music became widespread and easily accessible. Today, with the Internet and its applications, the music world is again being revolutionized.

In the past, the roles of audience and performers do not mix. Performers are the ones out on the stage performing, and the audience listened and watched. Today, with Internet technology, music performances not only can be interactive, but also *remotely* interactive. In other words, concert-goers can now enjoy the concert and *participate* in the music-making process without stepping outside of his livingroom.

This paper explored different possible models for creating interactive music experiences over the Internet. These Internet games are designed for different user groups in mind – but of course they are friendly enough to accommodate users of all types. Most of the games are designed to allow the users the greatest freedom possible – because after all, the point of these games is to provide a friendly environment for the participants to be as creative and imaginative as they can. Writing games with too many rules or restrictions would defeat the purpose.

Friends of mine who are not so musically-inclined sometimes ask me “the things you guys do in your group are fun – but what’s the point?” To me, this is like asking what the purpose is for playing a tennis game, for seeing a movie, or for living. I believe technological development expands people’s vision. Using technology to create new human experiences helps people to learn more, feel more, and think more. Projects like the Brain Opera demonstrate to the public how technology can revolutionize the process of music making, as well as how it can bring people – who might be hundreds or thousands of miles apart – together to participate in and share a common experience.

Appendix: descriptions of the most-used routines in ITX

exagt *[option] file1 file2*

exaggerates tempo change in file2 –
in accordance with the note-on to note-on differences between file1 and file2.

Example: `exag -d 1.5 mathematical.perf interpretive.perf`

exagv *[option] file1 file2*

exaggerates the dynamics in file2 –
in accordance with the dynamic differences between file1 and file2.

Example: `exag -d 1.5 mathematical.perf interpretive.perf`

rexagt *[option] file1 file2*

exaggerates tempo change in file2 –
in accordance with the note-on to note-on differences between file1 and file2.
(RATIO BASED)

Example: `rexag -d 1.5 mathematical.perf interpretive.perf`

invert *[files]*

invert files (ie, invert the pitch)

Example: `invert file.perf`

octavate *[options] [files]*

plays notes of a certain attribute value in octave

Example: `octavate -f 2 -v 1 math.perf`

for all notes in file `math.p`, if the 2nd attribute field is marked '1' then octavate that note

key *[file]*

finds the key of the file (eg., A Major, C minor, etc.)

Example: `key file1`

boost *[options] [files]*

boosts files (in volume)

Example: `exag -d 10 file1` (boosts the volume by 10)

copyattr *[option] file1.attr file2*

copies (note by note) attribute strings in `file1` to `file2`

Example: `copyattr file1.attr file2.perf`

tempoflex *[option] file1*

analyzes timing information, and returns diagnostic messages for places with large tempo variations. It tells the user what the average beat length is, and how much faster or slower the user has played at which measure and beat. This routine can be used to determine whether a student's performance is "too stiff" (mechanic) or "too wild."

chord *[option] file1*

prints the harmony of each beat (or unit of beats) and identifies perfect authentic cadences. This routine handles all major, minor, augmented, diminished triads and their inversions, as well as all major, minor, diminished, half-diminished, dominant, minor-major seventh chords and their inversions.

makeflatv *[options] file1*

sets the volume of all notes to 47, making every note the same loudness.

attredit *[options] -d digit -k mark file1.attrinit file2*

marks the 'd'th digit of all notes in file1 which also appear sequentially in file2 'k'

This routine takes two input files – file1 is a flat, attrinit'ed file, and file2 is a list of notes to be edited (in sequential order). This routine facilitates the attribute-editing process. For example, if the goal is to mark all the left-hand notes in file1 "k", the user would first record a left-hand-only version of file1, and name it file2. By issuing the command "attredit -d d -k k file1 file2", all the left-hand notes in file1 will have 'k' entered in the 'd'th position of their attribute strings.

attrinit *[option] file1*

adds a dummy string "ZZZZZZZZZZZZZZZZZZZZ" (17 Z's) to the "type" field of all the notes. It prepares a newly-recorded midi file for further attribute-field editing.

musicalizev *[option] file1*

"musicalizes" a file by giving it new amp values. This routine finds the highest and lowest pitches in both hands, and reassigns new amp values to all the notes by the following rule:

for right-hand notes – the higher the pitch, the louder the amp; the lower the pitch, the softer

the amp
for left hand – the lower the pitch, the louder the amp; the higher the pitch, the softer the amp.

We can apply `musicalizev` to a “dynamically flattened” midi file (as prepared by `makeflatv`), and convert it into a dynamically expressive file. This sort of routines are especially interesting to study, because it allows us to isolate rules of musicality and apply them to originally non-musical playings. Routines of this type can also help us in teaching the students what to do in order to make a piece sound more musical.

DEMO

```
***harmony analyzer (cadence detector)***
>clear
>chord -d 4 math | more

***key finder (prelude 1)***
>key p1.midi | more
***key finder (chopin polonaise no.3)***
>clear
>play /Surfski/Tunes/chopin/Polonaise3.midi
>key /Surfski/Tunes/chopin/Polonaise3.midi
***key finder (chopin mazurka no.13)***
>clear
>play /Surfski/Tunes/chopin/Mazurka13.midi
>key /Surfski/Tunes/chopin/Mazurka13.midi

***tempo flexibility analyzer***
>copyattr math testt > junk
>tempoflex junk | more

***EXAGGERATE time***
>play testt
>exagt -d 1.8 math testt > junk
>play junk

***EXAGGERATE time (flatten)***
>exagt -d 0.5 math testt > junk
```

```
>play junk
```

```
***EXAGGERATE volume***
```

```
>play testv
```

```
>exagv -d 1.5 math testv > junk
```

```
>play junk
```

```
***EXAGGERATE volume (reverse)***
```

```
>exagv -d -3 math testv > junk
```

```
>boost -d -15 junk > junkk
```

```
>play junkk
```

```
***musicalizer***
```

```
>play noct.flat
```

```
>play noct19.final
```

Note: to get noct19.final from original, do the following:

```
>perf /Surfski/Tunes/chopin/Nocturne19.midi > Nocturne19.perf
```

```
>makeflatv Nocturne19.perf > noct.flat
```

```
>attrinit noct.flat > noct.init
```

```
>attredit -d 0 -k 0 noct.init left_hand > tmp
```

```
>attredit -r -d 0 -k 1 tmp left_hand > noct.hand
```

```
>musicalizev noct.hand > noct19
```

```
>exagv -d 2 -h 1 noct.flat noct19 > noct.final
```

Bibliography

- [Cope, 91] David Cope, *Computers and Musical Style*, Madison, Wisconsin: A-R Editions, Inc., 1991.
- [Gosling, 95] James Gosling and Henry McGilton, *The Java Language Environment: A White Paper*, Sun Microsystems, Inc., 1995. (<http://java.sun.com/whitePaper/java-whitepaper-2.html#HEADING2-0>)
- [Grout, 88] Donald Jay Grout and Claude V. Palisca, *A History of Western Music*, New York: W. W. Norton & Company, Inc., 1988.
- [Hawley, 93] Michael Hawley, *Structure out of Sound*, Ph.D. thesis, MIT Media Laboratory, 1993.
- [HotJava, 95] *The HotJava User's Guide*, Sun Microsystems, Inc., 1995. (<http://java.sun.com/1.0alpha3/doc/misc/using.html>)
- [Horowitz, 95] Damon Matthew Horowitz, *Representing Musical Knowledge*, Master's thesis, MIT Media Laboratory, 1995.
- [JavaTutorial, 96] The Java Tutorial, Sun Microsystems, Inc., 1996 (<http://www.javasoft.com/doc/tutorial.html>)
- [JavaLang, 95] *Java: Programming for the Internet*, Sun Microsystems, Inc., 1995. (<http://java.sun.com/about.html>)
- [Lussy, ?] Mathis Lussy, *Musical Rhythm*.
- [Lussy, ?] Mathis Lussy, *Treatise on Musical Expression*.
- [Machover, 92] Tod Machover, *Hyperinstruments: A Progress Report*, available at MIT Media Laboratory, 1992.
- [Machover, 95a] Tod Machover, *Brain Opera*, available at MIT Media Laboratory, 1995.
- [Machover, 95b] Tod Machover, *ITX: A Progress Report*, available at MIT Media Laboratory, 1995.
- [Matsumoto, 93] Fumiaki Matsumoto, *Using Simple Controls to Manipulate Complex Objects: Application to the Drum-Boy Interactive Percussion System*, Master's thesis, MIT Media Laboratory, 1993.
- [RealAudio, 95] [RealAudio homepage], Progressive Networks, 1995. (<http://www.realaudio.com/>)
- [Rigopoulos, 93] Alexander Rigopoulos, *Parametric Control of Seed-Generated Music for Interactive Environments and Computer-Assisted Composition and Performance*, Master's thesis, MIT Media Laboratory, 1993.

- [Roads, 91] Editor: Curtis Roads, *Composers and the Computer*, Madison, Wisconsin: A-R Editions, Inc., 1991.
- [Rowe, 93] Robert Rowe, *Interactive Music Systems*, Cambridge, MA: MIT Press, 1993.
- [St. Hippolyte, 96] Michael St. Hippolyte, *Mumbo Jumbo*, 1996. (<http://www.users.interport.net/~mash/main.html>)
- [Waxman, 95] David Michael Waxman, *Digital Theremins: Interactive Musical Experiences for Amateurs Using Electric Field Sensing*, Master's thesis, MIT Media Laboratory, 1995.
- [WebStar, 96] Eric Métois, [*Reference Manual:*] *WebStar Server and StarClient Java Class and Interface*, internal document, MIT Media Laboratory, 1996.
- [Xing, 95] [Xing Technology Corporation homepage], Xing Technology Corporation, 1995. (<http://www.xingtech.com/>)
- [Yu, 96] Chong (John) Yu, *Computer Generated Music Composition*, Master's thesis, EECS, MIT, 1996.