Designing a Hyperinstrument with Gesture Interface for Musical Performance

Mario Cronje

Thesis presented in partial fulfilment of the requirements for the degree of Master of Philosophy in Music Technology in the Faculty of Arts, University of Stellenbosch.

Supervisors:
Mr Theo Herbst
Prof Johan Vermeulen

April 2005

DECLARATION

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

__________________ __________________

Signature Date

ABSTRACT

The field of gesture based research and the interaction between human and computer with the focus falling on musical applications, is well established internationally. However, in South Africa, research in this field appears dormant. The reasons for this state of affairs are complex and can be argued from different angles covering socio-economical, philosophical and educational perspectives.

This document describes the design, creation and implementation of an operational gesture interface environment which holds the potential to be expanded in the future. The implementation draws on cost-efficient hard- and software in the design of elementary to more advanced musical and even non-musical virtual environments (VEs) harbouring potential for further research and performance.

Hard- and software available at Stellenbosch University's Konservatorium were used together with selected free downloadable software from the internet in creating VEs, which to a degree simulate other techniques of sound manipulation. The choice of software was guided by the availability of support and prominence in terms of usage. Software basically had to incorporate hand movement tracking and the mapping of data to manipulate several parameters.

Three independent systems, each representing a different VE, were studied, experimented with and programmed in order to validate the thesis. The first system manipulates a complete electronic musical instrument. The second system incorporates the simulation of a real-life musical performance and the third system focuses on manipulating specific sequencing software by a basic alternative computer mouse implementation.

The outcome of this thesis provides an environment within which several programming techniques are treated and combined to form a template for teaching this field, and future development and research. These techniques incorporate the manipulation of digital audio, deal with a digital communication protocol, basic computer graphics and other necessary programming algorithms. In addition, the thesis strives to provide an outline for the understanding, design and implementation of a VE installation.

The three systems will be installed for operation during a presentation of this thesis. Together with the three operative systems, this document strives to act as an initial platform from which exciting futuristic research and activity can be launched.

OPSOMMING

Die gebied van beweging gebaseerde navorsing en die interaksie tussen mens en rekenaar waar die fokus op musikale toepassings val, is internasionaal stewig gevestig. In Suid-Afrika egter, blyk navorsing op hierdie gebied sluimerend te wees. Die redes vir hierdie stand van sake is kompleks en kan vanuit verskillende hoeke wat sosio-ekonomiese, filosofiese en opvoedkundige perspektiewe insluit, beredeneer word.

Hierdie dokument beskryf die ontwerp, skep en implementering van ‘n operasionele bewegings koppelvlak omgewing met potensiaal tot uitbreiding. Die implementering baseer op koste-effektiewe hardeware en programmatuur in die ontwerp van eenvoudige tot gevorderde virtuele omgewings (VOs) vir musiek, en selfs nie-musikale dissiplines met die potensiaal tot verdere navorsing en implementering binne die musikale uitvoeringspraktyk.

Hardeware en programmatuur beskikbaar aan die Konservatorium is gebruik tesame met ‘n seleksie van gratis programmatuur op die internet beskikbaar om VOs wat ander klankmanipulerings tegnieke simuleer, te skep. Die keuse van programmatuur is gelei deur die beskikbaarheid van ondersteuning en gewildheid en inkorpeer die volg van hand beweging asook die verspreiding van data om verskeie parameters te manipuleer.

Drie onafhanklike sisteme wat elk ‘n ander VO voorstel, is bestudeer, mee ge-ekperimenteer en geprogrammeer om die tesis te valideer. Die eerste sisteem manipuleer ‘n volledige elektroniese musiekinstrument. Die tweede sisteem inkorporeer die simulasie van ‘n werklike musikale uitvoering en die derde sisteem fokus op die manipulasie van spesifieke *sequencing* programmatuur sonder die hulp van ‘n muis.

Die uitkoms van hierdie tesis verskaf ‘n omgewing waarbinne heelparty programmerings tegnieke bespreek en gekombineer word in ‘n aanpasbare templaat vir die onderrig van hierdie veld asook toekomstige ontwikkeling en navorsing. Hierdie tegnieke inkorporeer die manipulasie van digitale klank, die omgang met ‘n digitale kommunikasie protokol, basiese rekenaargrafika en verdere noodsaaklike programmerings algoritmes. Verder streef die tesis daarna om ‘n raamwerk vir die begrip, ontwerp en implementering van ‘n VO daar te stel.

Die drie sisteme wat in hierdie tesis bespreek word, sal operasioneel geïnstalleer word gedurende ‘n demonstrasie daarvan. Saam met die drie werkende sisteme streef hierdie dokument daarna om te dien as platform waarvan af opwindende futuristiese navorsing en aktiwiteite geïniseer kan word.

“A wide range of applications can benefit from advances in research on gesture, from consolidated areas such as surveillance to new or emerging fields such as therapy and rehabilitation, home consumer goods, entertainment, and audio-visual, cultural and artistic applications, just to mention only a few of them."

“…the consolidation of new technologies enabling ‘disappearing’ computers and (multimodal) interfaces to be integrated into the natural environments of users are making it realistic to consider tackling the complex meaning and subtleties of human gesture in multimedia systems, enabling a deeper, user-centered, enhanced physical participation and experience in the human-machine interaction process.”
Camurri & Volpe (2004)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE
INTRODUCTION

This introductory chapter is devoted to the motivation behind the study, as well as its purposes and aims. Most importantly it serves to identify and analyse the relevant key concepts that form the background to the project, an understanding of which is crucial to successfully reach the stated objectives. The chapter concludes with an outline of the thesis' chapter content.

1.1 MOTIVATION OF THIS STUDY

Internationally, the already extensive body of research into the interface between human gesture movement and computer systems, be they musical or non-musical, has rapidly escalated in recent years.[1] However, specifically in terms of human gesture and musical computer interaction, the local (South African) body of research remains minuscule. A stark contrast is visible, mainly because the focus at the majority of local music institutions continues to fall on performance and education. Moreover, a pronounced resistance is harboured against the interdisciplinary connection that exists between music and fields such as computer science and engineering. However, even local computer science and engineering departments and faculties host limited research activities into gesture movement interface in a musical context.[2] Problems and limitations resulting from this state of affairs were exaggerated by the fact that this endeavour was initiated and located within a music department, not an engineering faculty or computer science department. In this instance the host department was Stellenbosch University's Konservatorium.

Initiating activity and implementing applications in the chosen field of study can be time consuming. The principle difficulties can be said to pivot around issues such as identifying and obtaining core literature[3] as well as the exploration of, and experimentation with new, but sometimes actually old and proven fundamental concepts.[4] To this must be added the disadvantage of working single-handedly, in isolation and dealing with the inevitable terminological misunderstandings that accompany the process.

[1] Quantitatively, the portability and availability of computer systems and other technologies have clearly increased dramatically over the last decades and therefore research can be done at more institutions. Qualitatively, the processing power of computer systems has allowed the research focus to shift from mechanical implementation to virtual environments.

[2] The reader is referred to the curricula of other South African universities.

[3] Most literature is available as proceedings and short articles. The reader is referred to http://recherche.ircam.fr/equipes/analyse-synthese/wanderle/Gestes/Externe/references_list.html for a list of primary articles.

This project evolved out of the author's music and computer programming teaching experience coinciding with the supervisor's interest in hand movement manipulation of oscillator parameters such as amplitude and frequency. This was followed by an unproductive research phase concerned with evaluating suitable mechanical tracking devices, which resulted in the identification of expensive and difficult to procure solutions.

During February 2002 Professor Marc Leman, director of IPEM,[5] was invited to pay a visit to the Konservatorium. During one of his lectures concerning research at IPEM the potential inherent to the computer tracking software, *EyesWeb*,[6] was briefly touched upon. This initial reference was explored and lead to a new angle and attitude towards the potential harboured in this field. More specifically the article by Camurri & Leman (1997) triggered an investigation into what came to be regarded as primary and secondary research activities necessary to successfully implement the actions listed under the purpose of this study.

The Konservatorium electronic music studio had by this time expanded necessitating the taking of an inventory of the few and recently acquired hard- and software solutions available to be employed in such a new field. It became clear that the time consuming process of connecting and setting up a gesture environment installation would be exaggerated if specific rooms could not permanently be reserved for this purpose and the hard- and software was used daily by other students for other projects. Also, due to financial restrictions and to prevent future financial loss resulting from a waning of interest and subsequent loss of skills, the author decided that any additional software had to be freely downloadable and accompanied by legal research license agreements.

The pronounced emphasis placed on the value of expanding *Information and Communication Technology* (*ICT*)[7] in the developing world provided further motivation. It was hoped that the successful implementation of the first African gesture research environment, based at a music institution, would pave the way for further expansion, resulting in numerous practical and educational spin-offs.

[4] The thesis strives to introduce concepts to the reader throughout the document.
[5] Instituut voor Psychoacustica en Elektronische Muziek, Department of Musicology, Ghent University, Ghent, Belgium. http://www.ipem.rug.ac.be
[6] Refer to sections 2.4 and 4.2.
[7] The following quotes are adopted from a draft white paper of the South African National Department of Education (2003):
"Africa is a developing continent. The lack of developed infrastructure for information communication technology is exacerbating the gap between Africa and the developed world."
"Our quest for active contextual learning to promote understanding will be supplemented by multi-media applications that require learners to create realistic contexts for problem-solving, data analysis and the creation of knowledge in the learning process."

Closer to home, a more personal motivation for this study flowed from the researcher's interest in unconventional, futuristic musical performance approaches capable of providing musicians and the broader public with new and innovative concepts.

1.2 PURPOSE OF THIS STUDY

The purposes of the study are manifold. Firstly, the reader is introduced in the necessary and to varying degrees to those fields of study on which the implementation draws. In the second place stands the design and creation of a cost-efficient, but high quality system suited to both entry-level and advanced research and performance, by employing suitably applicable software chosen from leading options.[8] Thirdly, surroundings conducive to the teaching of gesture interfaces and the necessary programming techniques, giving students the ability to create and design new music environments, are established. The practical component of this thesis has already been implemented in the departmental Music Technology programming modules with the latter acting as an evaluation of this thesis.

The particular angle taken is to create a versatile system which translates into different setups. The first concerns the control of hardware paraphernalia such as a MIDI-controlled synthesizer, the second a simulation of a real-life setup, for example a club DJ environment and the third the manipulation of a well-known sequencer, such as *Pro Tools*.

"In the interim, the Department of Education will initiate the collection and evaluation of existing digital, multi-media material that will stimulate all South African learners to seek and manipulate information in collaborative and creative ways."

"There are three critical elements that will determine ICT's future as an effective tool for social and economic development. Firstly is cost. Any solution that South Africa adopts has to be cost-effective if we are to meet our developmental demands and to reach the most remote parts of our country. Secondly is sustainability. It is no use having state of the art technology unless it can be sustained. Thirdly is the efficient utilisation of ICT. Deployment of ICT does not guarantee its efficient utilisation. Capacity building and effective support mechanisms must accompany it."

"The ongoing costs of providing access to technology, including teacher development, pedagogical and technical support, digital content and telecommunication charges, as well as maintenance, upgrades and repairs are enormous."

"In response to this under-development, Africa has adopted a renewal framework, the New Partnership for Africa's Development (NEPAD), which identified ICT as central in the struggle to reduce poverty on the continent."

"In order to realise the benefits of ICT, Africa must develop and produce a pool of ICT-proficient youth and students from which we can draw trainee ICT engineers, programmers and software developers."

"The Commission advises Government on the optimal use of ICT to address South Africa's development challenges and to enhance South Africa's global competitiveness."

"The challenge is to roll-out ICT infrastructure that is specifically suited to Africa. Through appropriate technologies, it is hoped that South Africa will leapfrog into the new century, bypass the unnecessary adoption cycle and implement a solution that works now and has the capacity to handle future developments."

[8] As a number of possible pieces of software are mentioned and discussed in related literature, the motivation of this thesis's selection is stated throughout this document.

In this thesis the term "three systems" is used for the above three setups. More specifically, *SystemOne*, *SystemTwo* and *SystemThree* refer to the practical applications forming the core of the endeavour.

1.3 BACKGROUND OF THIS STUDY

The aim of this section is to introduce concepts and terminology relevant to this project, therefore commonly employed by researchers and users of applications. These are regarded as paradigms and are studied and implemented in the practical side of this thesis, so that it is clear that the three systems are based on international trends in research.

1.3.1 Virtual Environment

In its simple form a *virtual environment* (VE) comprises a simulation of real-life interaction, consisting of a controller (user) connected through a computer interface, to manipulate an object, which resolves into a result. The incorporation of VEs is a well-known research area in different disciplines. In the field of medical research, where the outcome is saving lives, the controller (surgeon) and the result (surgery on a patient) are for example connected by computer software and a network. By comparison VEs can be incorporated to enhance and explore musical performance and composition possibilities, as can be seen in the outcome of this thesis.

According to Wilson (2001) there is no standard definition for a VE, but a number of characteristics are noticeable. Firstly, "the user is immersed in an alternative 3D graphical world that simulates a real or imaginary world". Secondly, "users can navigate through the environment, usually assisted by a user embodiment or avatar". Thirdly, "users can examine, interact with and manipulate virtual environment objects". In the case of this thesis, two-dimensional virtual environments are programmed in the form of a *hyperinstrument* and they are therefore based on the hyperinstrument paradigm.

1.3.2 Hyperinstrument

According to Machover's[9] paradigm a hyperinstrument is a virtual musical instrument, incidentally also a virtual environment, which responds to triggers or sensors connected to a computer with the aim of generating principally sound but also imagery. For Ungvary & Vertegaal (2000), Machover's hyperinstrument paradigm revolves around the difference between traditional musical instruments and hyperinstruments. Leman (unpublished) summarises this finding by stating that the former have their "control and sound generation parts tightly coupled with a solid physical interface" and the latter "uncouple the input representation (physical manipulation) from the output representation (physical auditory and visual stimuli of a performance)". It follows that for Machover the focus shifted from traditional instruments to innovative musical environments.

It is important to understand the hyperinstrument paradigm within the context of the comprehensive *multimodal environment* paradigm, as discussed by Camurri & Leman (1997). According to them virtual environments and hyperinstruments remain unchanged over time, therefore constituting static environments that do not adapt their behaviour during the performance of the system. A multimodal environment however can be said to represent a "dynamic hyperinstrument" in that it changes its behaviour and functionality by adapting to the user's input over time.

True to the hyperinstrument paradigm, the functionality of the three systems were conceptualised in principal and programmed beforehand. This enables the user to control the instruments according to specific rules. The project stops short of venturing into the domain of the multimodal environment paradigm which is left to be researched in a future study. See section 2.4.

1.3.3 Gesture Interface

This thesis focuses on capturing hand movement data with a camera and in the process the three systems do not make use of any electronic or mechanical hardware devices. Research concerning hand movement is generally embedded in research on gesture. For the purpose of this thesis the two terms shall act as synonyms. As stated by Cadoz & Wanderley (2000) gesture research is dependent on generating and interpreting hand movement data as the hand is "the primary organ associated to the gestural channel". Although they do not come to a firm conclusion, they do ask the question whether, concerning gesture, one can distinguish between "free" or "manipulative" movement. To a degree Sapir (2002) supports this division by associating gesture with "manipulative or communicative movements on an intent". For the purpose of this thesis the focus shall fall on the former, namely the manipulation of an environment.

9 Refer to section 2.3.

From a different perspective Mulder (1998) distinguishes between “gesture” and “posture” by pointing to the fact that the former is a dynamic process through time while the latter is static. He puts it that “hand posture and gesture describe situations where hands are used as a means to communicate to either machine or human.” He also points to the fact that “empty-handed” and “free hand” gesture is used to describe hand movement where no clasped physical object is manipulated.

Marrin (2002) defines two gesture methods, i.e. “discrete” and “continuous”. The former, a single movement such as up or down represents a single action and is comparable to, for example, an alphabetical letter in one of the sign languages. The latter she defines as following an “ambiguous trajectory” that must be interpreted. For the purpose of this thesis discrete gestures are to be used primarily, although a less often used circular gesture (controlling a rotary knob) could also be defined as continuous.

If gesture as defined above is located within an environment, varying degrees of gesture interface is said to occur. O’Hagan (2001) defines a gesture interface as “body movements which are used to convey some information from one person to another” and in this case, from a person, the user, to a computer via a camera.

Nielsen et al. (2004) states that a gesture interface should only be considered if it proves to be the most efficient and therefore optimized interface for a specific application. As the design of the three systems were governed by the available hard- and software and the author’s interest, their implementation served to establish a field of activity and is therefore not in line with Nielsen recommendation.

1.3.4 Mapping

As discussed previously, the input and output of a hyperinstrument is physically uncoupled, although they are connected by means of *mapping*. Kirk & Hunt (1999) explains mapping as the manner in which "the playing interface are [*sic*] interpreted by the synthesis engine". Marrin (2002) defines mapping as "the way that the musical response reflects the gesture that activates and controls it". For Sapir (2002) mapping follows gesture capturing and concerns the manner in which "gestural data will be related to sound processing". It follows that the basic operating principal of any instrument, including a hyperinstrument, can be said to linearly follow three phases, i.e. input ⇒ mapping ⇒ output. A feedback cycle is observed, as the input adapts according to the output. If follows that if latency for example is present, the user anticipates performance.

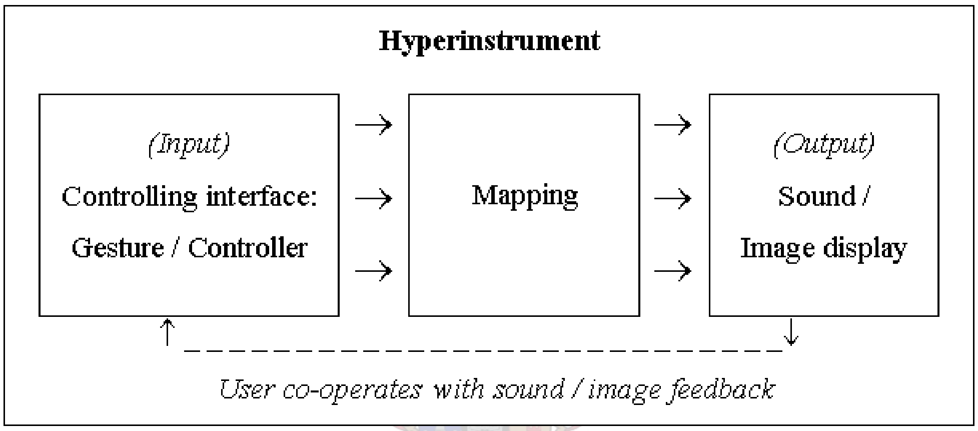


Figure 1. Basic structure of a hyperinstrument.

A common mapping uncertainty that can occur is called "perceptual disconnection" (Marrin: 2002) or the "problem of causality perception" (Camurri & Leman: 1997). This involves the degree of the user's and audience's recognition of the dependency of a particular output on an input. As mapping is done by programming the mapping software, various and numerous parameters can be controlled by a single gesture or as Mulder (1998) puts it "that the possibilities for mapping are endless". He explains that "mapping may be faster to learn when movement features are mapped to sound features of the same abstraction level". An example would be the basic method whereby the hand goes up and frequency rises.

Many music related projects make use of the graphical object programming environment, *Max/MSP*;[10] however, the programming language, *Java*, is used in the three systems, as mapping software. Refer to chapter four for a discussion on Java.

[10] Available from http://www.cycling74.com.

1.3.5 Computer Programming

The design and development of programming software forms an important part of this thesis. Kirk & Hunt (1999) defines programming as “a creative *art*, not just a technique” as it includes several careful steps of development. The programmer (software developer) has to focus on features such as user-friendliness, capability, stability, expandability of software, etc.

A program consists of instructions and data, of which the former is the procedure to be executed on the data. A couple of instructions performed on some data, to accomplish a specific task, is called an algorithm. In addition, a program consists of numerous algorithms, which are to be coupled in an effective way, achieved by frequent programming practice.

1.4 SCOPE OF THIS THESIS

Chapter one, the introductory chapter discusses the motivation and purpose or aim of this study. In addition this chapter provides the reader with an introduction to the terminology used in the field, thereby providing the necessary background. The chapter concludes by outlining the structure of the thesis. Chapter two identifies a basic classification of controllers, followed by an introduction of projects and research from leading figures and at leading institutions.

Chapter three sketches the background to the three systems that are selected to provide a practical enhancement and validation of this thesis. This chapter also demonstrates the installations that will be constructed for a practical performance when using these systems. Chapter four discusses the software that has been used to program the three systems, by providing an overview of the software and how this software was implemented. Chapter five offers a technical discussion of the difficulties that were encountered during the programming of the three systems and how these were solved.

Chapter six forms a conclusion and explains how further research based on the purpose of this thesis can be implemented, and also how the three systems can be improved with new ideas to validate the practical implementation of the thesis. The thesis is accompanied by the obligatory references and appendices.

CHAPTER TWO
RELATED LITERATURE REVIEW

2.1 INTRODUCTION

This chapter involves the brief discussion of research and products of prominent figures at some of the leading institutions and therefore takes the form of a literature review. It starts with a concise classification of controllers, thereafter discussing figures and institutions, and concludes with additional products. A well thought through classification of all interactive music system paradigms falls outside the scope of this thesis, as the number of devices and installations presently researched internationally can not be over-estimated. The focus on the products to be discussed will fall on the controlling interface and not the mapping, although it has to be kept in mind that various controllers can be used with numerous mapping techniques.

The focus of this thesis falls on the development of a gesture-based software environment suitable to install at music institutions unfamiliar with the gesture interface paradigm. As the research towards this thesis was not based on previous research conducted at the host department, the modus operandi boiled down to an investigation of international gesture-based research. The outcome is a not extremely in-depth study on a specific field; however, a general perception of the environment is created. The procedure that was followed was to initiate the development the software product accompanying the thesis soon after a well researched selection of software were made.

By way of a background Marrin (1996) provides a useful point of departure when discussing the very early history of what today amounts to gesture interface. She quotes Roads (1996)[11] who stated that “The original remote controller for music is the conductor's baton”. Combining this perspective with the fact that many contemporary devices require no attachments, the history of gesture interfaces are closely associated with the history of musical conducting which has its roots, according to Spitzer & Zaslaw (2001), in the fifteenth century.

The gap between these early roots and current research can be bridged by pointing to the first documentation describing the “Brussels key-device”, dating from the 1830’s. This electromechanical device consisted of one piano-key which turned on a light when pressed and was used to indicate the conductor’s tempo to an offstage chorus.

[11] Roads, C. (1996) *The Computer Music Tutorial*. Cambridge, Massachusetts: The M.I.T. Press. p.653

Leon Theremin (b. 1896, "Lev Termen" in Russian) invented the *Theremin* in 1919. This monophonic electronic instrument with an idiosyncratic sound covered a range of 3-5 octaves. The device incorporated a vertical antenna and horizontal loop connected to a cabinet on the right top and left side respectively. The cabinet contained the operational circuitry. The performer interacted with the instrument by moving his or her right hand along the antenna to change pitch and the left hand along the loop to change volume.

Current research into developing controllers consistently takes cognisance of these earlier devices and can be said to draw inspiration from them.

2.2 BASIC CLASSIFICATION OF CONTROLLERS

A number of studies, taking a variety of approaches in classifying numerous gesture research applications, have been conducted.[12] Discusssed below are three research studies, by Winkler (1998), Wanderley & Battier (2000) at IRCAM and Mulder (1998), which are sorted from least to most complex.

Winkler (1998) sorts controllers into four categories:

- *Acoustic models* simulating acoustic instruments, such as the MIDI keyboard simulating the piano and organ.
- *New instruments* which are based on the functionality of acoustic instruments, but with the focus on innovative controllers, for example the modulation wheel on a keyboard controlling frequency or amplitude.
- *Spatial sensors* determining the position and motion of an object in a multidimensional environment by using two dimensional image grids, for example the dance/music research conducted by the Mega Project.[13]
- *Body sensors* including devices attached to the body to determine body part movement, of which Marrin's Conductor's Jacket is an example.

[12] Refer to http://www.notam02.no/icma/interactivesystems/wg.html.
[13] Refer to section 2.5.

Wanderley & Battier at IRCAM sorts gesture research on musical applications[14] into three categories:

- *Instrumental gestures*, such as hyperinstruments to control an instrument with movement.
- *Dance / music interfaces*, for example the research of the Mega Project where the focus falls on the coherence between dance and music.
- *Conductor's gestures*, as researched by Teresa Marrin on controlling music by conducting, for example using her Conductor's Jacket.

Wanderley & Battier at IRCAM also distinguishes between *Haptic* and *Non-haptic* representation. The former concerns the use and the latter the absence of physical objects to generate electrical signals from gestures. Although not used consistently in the field, these terminologies are used by Mulder (1998).

According to him controllers can be categorised in touch-, expanded range- and immersive controllers, of which the latter is divided into internal-, external- and symbolic controllers.

- *Touch controllers*, such as traditional instruments, where the device's structure delimits the manner in which it is operated by being touched.
- *Expanded range controllers*, such as the Theremin and "The Hands", where the user touches a physical object or not, and although limited to specific motions to control the device, is freer than in the above case.
- *Immersive controllers*, where few or no restrictions on the user's movements apply. These divide into:
    - *Internal*, such as the Conductor's Jacket, where the controller is a simulation of the human body and determines body movement, for example a finger motion.
    - *External*, this complex controller is not a simulation of the human body and determines movement according to other body movement, such as the changing distance between hands.
    - *Symbolic*, the controller can not be visualised, and operates according to specific movements such as conducting, sign language or dancing.

[14] Non-musical application categories include natural gestures, sign language, HCI (Human Computer Interaction), etc.

By using the above, a classification of the three systems, around which this thesis revolves, amounts to the following: they are non-haptic, expanded range controllers with instrumental gesture making use of spatial sensoring. However, it must be added that the three systems were not conceptualised according to these categories, but by making use of the available hardware and free software.

2.3 TOD MACHOVER AT MIT

Tod Machover (b. 1953) initiated the "Hyperinstruments" project in 1986 at the MIT Media Lab at the Massachusetts Institute of Technology (MIT).[15] The reader is referred to chapter one for a definition of the hyperinstrument paradigm. From 1986 to 1991 Machover concentrated on creating devices for well-trained musicians[16] with the aim of expanding the expressive intentions of performers by the use of computers. From 1991 onwards his focus shifted to the creation of systems for use by not only professional musicians. The motivation behind this shift was summarised by him (1995) as follows: "that any normal, intelligent person is capable of far more sensitivity and creativity than he/she is normally given credit for". From 1995 onwards this approach developed into large-scale interaction, particularly the "Brain Opera" touring[17] installations, which consisted of numerous creative hyperinstrument systems.[18]

Three systems from the Brain Opera, which can be categorised as gesture-based devices, the *Gesture Wall*, *Sensor Chair* and *Digital Baton* are discussed below. The latter will be discussed elsewhere.

[15] As mentioned on the MIT Media Lab website (http://www.media.mit.edu), the laboratory was derived from research by the MIT Architecture Machine Group in 1980 and focused on "cognition and learning to electronic music and holography" research. It is also mentioned that the Media Lab presently focuses on the "study, invention, and creative use of digital technologies to enhance the ways that people think, express, and communicate ideas, and explore new scientific frontiers".

[16] These include the cellist Yo-Yo Ma, Peter Gabriel and even ensembles such as the Los Angeles Philharmonic.

[17] A world tour in 1998 covered United States, Europe, Asia and South America.

[18] For example the "Speaking Tree", "Singing Tree", "Melody Easel", "Rhythm Tree", "Gesture Wall", "Sensor Chair", etc.

2.3.1 Gesture Wall

The Gesture Wall is based on *transmit-mode electric field sensing*[19] and determines the position and movement of a user's hands and body in front of a projection screen with images (Paradiso: 1999). The user stands in front of the screen on top of a brass transmitter panel, which is driven by a 50-100 kHz sinusoidal signal at a voltage of 2-20 Volts. The user becomes a transmitter antenna through capacitive coupling. Next to the screen, corresponding to the four corners of the screen, are four receiver antennae, each measuring the amplitude of the signal transmitted through the body to the hands.

Mapping of the device include the output of MIDI code from a PC to a synthesizer, playing sequences, and also another PC connected to a video projector for graphical display on the screen.

2.3.2 Sensor Chair

The Sensor Chair also uses transmit-mode electric field sensors and has almost the same setup as the Gesture Wall. The transmitting plate is attached to the seat of a chair, resolving in better electrical coupling into the user's body. The four receivers each have a halogen light attached. These glow in accordance with hand positions and provide visual feedback to the user. The system also incorporates two footswitches for additional functionality.

Although the Sensor Chair's original mapping scheme was designed to accompany magicians,[20] it is used in the Brain Opera to control the "Future Music Blender". The latter is basically a sample player, with which samples can be chosen, edited and mapped.

[19] For more information see Paradiso, J., Gershenfeld, N. (1997) Musical Applications of Electric Field Sensing. In *Computer Music Journal*, 21(3), 69-89.

[20] For example the act by Penn & Teller.

2.4 ANTONIO CAMURRI AT DIST

Antonio Camurri (b. 1959) and his colleagues are located at the InfoMus Lab (Laboratorio di Informatica Musicale), DIST (Dipartimento di Informatica, Sistemistica e Telematica) at the University of Genova, Italy.[21] Their aim is to design and build interactive multimodal environments (MEs) with a focus on expressive dance and music analysis, and expressive human-robot interaction. *Expressive gesture*[22] research at DIST is based on the dimensions in dance gesture, as described in Laban's "Theory of Effort".[23]

The EyesWeb software, used in this thesis, was developed at DIST where it continuous to play a prominent role in research projects. EyesWeb is also continuously upgraded. For the purpose of this research project EyesWeb is used in a two-dimensional environment, while research at other institutions often incorporates three-dimensional environments. See section 6.1.3 for future development incorporating 3D.

An additional library of software modules, the EyesWeb Expressive Gesture Processing Library, is used for expressive gesture analysis and forms part of the MEGA project which is discussed below. This library of blocks is to be used in future developments of this thesis and includes:

- The EyesWeb Motion Analysis Library: used for motion tracking.
- The EyesWeb Space Analysis Library: used for tracking an object in a 2D space with grids.
- The EyesWeb Trajectory Analysis Library: used for trajectories in 2D (real or virtual) spaces.

Refer to Camurri & Volpe (2004) for more information on the library and www.eyesweb.org for downloading the software.

[21] As mentioned on the InfoMus Lab website (http://www.infomus.dist.unige.it), the laboratory developed in 1984 as part of DIST and with the aim "to carry out scientific and musical research, design, development and experimentation of key technologies and systems for music, dance, theatre, edutainment and museums".

[22] "Expressive gesture" is also described by the Japanese term, "KANSEI", a keyword that prominently features in research from Camurri and Shuji Hashimoto based at the Department of Applied Physics, Waseda University, Japan.

[23] Refer to publications by the German choreographer Rudolf Laban, such as Laban, R. (1963) "Modern Educational Dance" (2nd Ed.) London: Macdonald & Evans Ltd.

2.4.1 HARP / Vscope

An example of an older research project at DIST is the *HARP / Vscope* system. HARP consists of software modules, called "agents",[24] which are created and removed and influence each other at specific moments. The Vscope sensor system includes wireless markers on the hands, feet and torso of a dancer, and is used to control the HARP system.

2.4.2 "L'Ala dei Sensi"

Another example is the multimedia performance "L'Ala dei Sensi", which is an on-stage installation consisting of a small mobile wheeled robot, dancers, large video projection screens, video cameras, wireless sensors attached to the dancers, EyesWeb, etc. The performance consists of various episodes, such as a dancer-robot dialogue, interaction of two dancers with their real-time edited images displayed on video screens, resolving into complex graphics, and a virtual mirror environment (Camurri: 2000b).

2.5 THE MEGA PROJECT

Since 2001 DIST has in collaboration with other institutions,[25] made up the MEGA project (Multisensory Expressive Gesture Applications), of which Antonio Camurri is the project coordinator. The project's focus falls on "the analysis of expressive and emotional content in non-verbal interaction" (Fagernes & Hagen: 2002). Despite the fact that the institutions engaged in research in different fields, focussing mainly on music and video analysis, the main topics are:

- Analysis of expressive gestures
- Synthesis of expressive gestures
- Mapping strategies
- Integration issues
- Applications in MR (Mixed Reality) environments

Refer to Fagernes & Hagen (2002) or www.megaproject.org for a discussion on these topics.

[24] According to Camurri & Leman (1997) agents "exhibit dynamic, intelligent, real-time and adaptive behavior". They are also described as "program modules which, at run-time, perform certain tasks exhibiting skills such as music analysis, gesture analysis, interactive composition, perception of audio input and so on."

[25] The MEGA project consists of: - University of Genova (DIST), Lab of Musical Informatics, Italy; - University of Padova (DEI), Italy; - Ghent University (IPEM), Belgium; - Royal Institute of Technology (KTH), Department of Speech, Music and Hearing, Sweden; - Uppsala University, Department of Psychology, Sweden; - Telenor R&D, Norway; - Generalmusic, Italy; - Consorzio Pisa Ricerche, Italy; - Eidomedia, Italy.

One of the applications developed in the MEGA project, as discussed in Fagernes & Hagen (2002) includes a virtual walking robot, which changes its walking mood according to the expressiveness of music and sound analysis as well as gesture analysis of dancers present in an installation.

Other project examples include *Ghost in the Cave*[26] and *Groove Machine*.[27]

2.6 MICHEL WAISVISZ, STEIM

Waisvisz (b. 1949), the director of STEIM (studio voor elektro-instrumentale muziek) since 1981, arrived at the institute in 1973. STEIM concentrates on developing new live electronic concepts and applying these to the performing arts. The studio also supports collaboration with international performers and musicians in concerts and workshops. Technology developed at STEIM has also been utilised by DJ's and VJ's (Video Jockeys) eager to expand their environments through new ideas and equipment.

The Hands

Among numerous controllers[28] forthcoming from STEIM, *The Hands*, developed by Waisvisz in 1985, is a remarkable device in the history of gesture interfaces. The controller allows the user to move his/her hands freely in three-dimensional space and incorporates numerous sensors consisting of mercury switches, sonar, and toggle switches. The Hands generates MIDI code, which can be ported to a MIDI instrument.

2.7 TERESA MARRIN

Teresa Marrin Nakra completed a Ph.D. at the MIT Media Lab and was part of the "Brain Opera", discussed elsewhere. She is the artistic director of the non-profit Boston-based organisation, Immersion Music, which focuses on high-tech application connected to the traditional performing arts. Below follows two examples developed by her at MIT in the course of her master's and doctoral studies respectively.

[26] Refer to Rinman, M. et al. (2004) *Ghost in the Cave – An Interactive Collaborative Game Using Non-verbal Communication*. In Camurri, A. & Volpe, G. (Eds.) Gesture-Based Communication in Human-Computer Interaction. Springer-Verlag, Berlin Heidelberg. pp. 549-556.

[27] Refer to Marklund, K. et al. (2002) *Groove Machine*. Available from http://www.megaproject.org/Performances/p7_KTH/groovemachine/GrooveMachine.pdf.

[28] Such as Crackleboxes, LiSa, BigEye, Sensorlab, etc.

2.7.1 Digital Baton

This device incorporates three different sensor systems into one. The first is an infrared LED used for tracking the position of the baton's tip with a camera. Five force-sensitive resistor strips at the bottom of the baton measure the pressure of the fingers and palm. The device also incorporates three orthogonal accelerometers to measure beats and sweeping gestures (Paradiso: 1999). The digital baton is capable of controlling several musical parameters.

2.7.2 Conductor's Jacket

The controller is a shirt which is worn by the user and consists of sixteen sensors, for measuring respiration, heart rate, skin conductivity, temperature, motion, etc. (Marrin: 2002). As this device determines gesture, it is designed to measure other parameters associated with a conductor. Apart from the jacket, the device also employs two networked computers to manage, filter and map the sensor data. A range of MIDI-controllable equipment can be controlled by the output of the computers.

2.8 AXEL MULDER AT INFUSION SYSTEMS

Axel Mulder is engaged in research towards *Virtual Musical Instruments* (VMI). A VMI can be described as a "musical instrument without a physical control surface, but instead a virtual control surface that is inspired more or less by the physical world" (Mulder: 1998). Mulder's focus falls on the creation of gesture interfaces that are specifically suited to determining hand and other body gestures for the multidimensional control of sound. Refer to the PhD dissertation of Mulder (1998) for a classification of numerous controllers.

His Canadian company, Infusion Systems Ltd.,[29] designs and creates numerous innovative controlling and sensor devices, driven by the *I-CubeX systems,* which transmit MIDI code. Controllers include the *TouchGlove v1.6* and sensors include the *TapTile v2.1*, and *Reach v2.0.* Numerous installations have been established and researched and include *I-Rave*, *Integration of music and martial arts*, *Pictures of Sound* and *Do-Be-DJ interactive music installation.*

[29] Refer to http://infusionsystems.com.

2.9 OTHER PROJECTS

2.9.1 The Radio Baton

Max Mathews[30] spent time during the 1960's at Bell Telephone Laboratories on gestural input for human-computer interface. Later he became closely associated with the *Radio Baton*, a transmit/receive system where the co-ordinates (*x*, *y* & *z*) of the baton are determined. A low-frequency radio transmitter is attached to the end of a baton. Several configurations can be used for the placement of the receiving antennae, which determine the distances between the baton and antennae every millisecond. One specific configuration[31] includes four antennae placed in the four corners of an *x*-*y* square and a fifth antenna in the middle. Near the square, accurate positions of *x* and *y* can be determined, but as the baton moves away from the square, the *z* position is tracked, but results in an inaccuracy of both *x* and *y*.

The mapping includes the *Conductor Program*, which is a computer with a sequencer program, controlled by the Radio Baton which sends MIDI messages to external devices, such as a synthesizer. The work to be performed is stored in the program with trigger points (for example every beat) clearly marked. The tempo at which the work is played is calculated by comparing the baton movement to these trigger points, and generates MIDI code to be sent out. Refer to Mathews (1991) for a more elaborate discussion of the functionality of the Radio Baton.

2.9.2 Twin Towers and Imaginary Piano

The Italian Leonello Tarabella developed the *Twin Towers*, which makes use of two groups of infra-red beams. These beams are projected upwards from a base unit. The user moves his/her hands through these beams and the reflections back to the base are calculated, determining the type of movement made. The output data is used to control for example synthesizers.

The *Imaginary Piano*, another device by Tarabella, makes use of video capturing. The user sits on a chair, facing a camera. When his/her hand or finger strays below a virtual horizontal line, a message representing for example a note to be played is compiled. The rapidity with which the line is crossed determines the velocity of the note.

[30] Born in Columbus, NE in 1926. Since 1987 he has held a professorship at Stanford University's CCRMA (Center for Computer Research in Music and Acoustics). Refer to http://ccrma.stanford.edu.

[31] Sometimes called the "radio drum".

2.9.3 The Interactive Dance Club

This electronic dance music project formed a once off happening with the purpose of creating surroundings "where people could have the opportunity to become players in a large, interconnected, interactive musical and visual environment" (Ulyate & Bianciardi: 2002). The outcome was to not hold a DJ and/or VJ responsible for the music and visuals, but to delegate this to numerous interesting devices manipulated by the participants in the club.

Some of the controllers incorporated gesture based movement, such as the *Beam Breaker* with parallel light beams which triggers different samples when beams are broken. In the case of *Tweak* a user sways his/her hips between two infrared proximity sensors which control the cut-off frequency of a bandpass filter with percussion samples passing through it.

2.9.4 Steven Spielberg's film "Minority Report"

Considering the extensive body of international research, it is unfortunate that the use of gesture-based devices was probably first introduced to the South African community through the gestural input portrayed in the 2002 Spielberg film "Minority Report". The concept used in the film was developed by the MIT alumni Prof. John Underkoffler[32] and focused on the manipulation of numerous images through hand movements. Although this concept constituted a fictional interface, and a non-musical application at that, the result was an environment where the user, wearing futuristic gloves, acted like a conductor manoeuvring video chunks on a transparent wide screen.

2.10 Conclusion

The aim of this chapter was to introduce several gesture-based applications. There exist numerous other distinguished figures and institutions, which are not mentioned, because of space limitations and focus considerations of this thesis.

For further examples the reader is referred to
http://www.notam02.no/icma/interactivesystems/wg.htm.

[32] The "virtual reality" pioneer and composer, Jaron Lanier, also exercised a prominent influence. Refer to http://www.21cmagazine.com/issue1/minority.html.

CHAPTER THREE
BACKGROUND AND APPLICATION OF THE THREE SYSTEMS

3.1 INTRODUCTION

The goal of the thesis is to create an environment for gesture interface applications and further research. Three applications, which differ in mapping, are identified, selected and programmed, demonstrating the possibilities and validity of the three systems. The applications were selected according to the available hardware and software and on levels of interest displayed by the researcher and in the host department. The three systems do not strive to use this equipment in the most ergonomic constructive manner, but high importance is paid to the creation of a clean, therefore stable and optimised environment. This chapter discusses the origins of systems one and two and also what can be expected from all three systems. Also, in the case of systems one and two this chapter sets a benchmark in order for their eventual performance to be measured. The chapter's three sections each provide background information, followed by a discussion on the implementation of each application. The chapter concludes with a list of the hardware including their operating systems to be used in the installation of the three systems.

The first system consists of a digital (virtual analog) synthesizer that is controlled by MIDI data generated through gestures. The discussion around the systems includes an overview on *subtractive synthesis* and the detailed operation of the system. The second system comprises a dance music[33] DJ (Disc Jockey) environment, where some of the functions of a DJ, excluding advanced "scratching" (turntablism), are controlled by gestures. Also included is a discussion of the responsibilities and working methods of a DJ mixing on two decks and a mixer. The third system is a *Pro Tools*[34] *MIDI control surface* environment, in which a Pro Tools session is partially or fully controlled through gesture. This section discusses how the MIDI implementation of the *JL Cooper CS-10*[35] MIDI control surface can be overridden.

[33] Peel (2001) defines the term dance music as "20th-century club dance music". According to him "It developed out of DISCO and the invention of the synthesizer into a major worldwide force, eclipsing rock; unlike most others genres, it has developed at a very fast rate, aided largely by the continual invention of sub-genres and frequent artistic collaborations."

[34] Digidesign (2001a) states that Pro Tools "integrates powerful multitrack digital audio and MIDI sequencing features, giving you everything to record, arrange, edit, mix, and master professional-quality music".

[35] Refer to section 3.4.1 for a discussion on MIDI control surfaces for Pro Tools.

The original product accompanying this thesis was planned to be an analog synthesizer programmed in *JSyn*. As research developed and new hardware was obtained, the system divided into the three systems, henceforth called SystemOne, SystemTwo and SystemThree. A hardware synthesizer is used in the place of the original concept and called SystemOne. The use of JSyn is moved to the DJ environment, called SystemTwo, followed by an additional approach, controlling a sequencer and called SystemThree.

As these three systems make use of a two-dimensional environment, the single movement of the user can be compared to the two movements of a computer mouse, i.e. *mouse-moved* and *mouse-dragged* events.[36] In both event types the on-screen cursor, representing the position of the mouse, is moved, but with the latter the mouse button is pressed while moving. Derived from this, *free* or *controlling* movements are identified to be used in the three systems. With these movements, both cases move the cursor, but with the latter a control region can be manipulated when the cursor moves into the region. SystemOne makes use of only free movements, but with SystemTwo and SystemThree both movements are implemented.

3.2 CONTROLLING A HARDWARE ANALOG SYNTHESIZER (SYSTEMONE)

An analog synthesizer can be described (Russ: 1996) as a device that produces audio signals, but also generates control signals which are used to manipulate these audio signals. Both control and audio signals are associated with a range of parameters, and on commercial synthesizers these parameters are manipulated by knobs, buttons, sliders, ribbon-controllers, etc. This section discusses the basics of subtractive synthesis and introduces the synthesizer that is used for this system.

3.2.1 Subtractive synthesis

Russ (1996) states that "the majority of commercial analogue synthesizers use subtractive synthesis" and is based on instruments which can be broken down into smaller modules, of which some produce and other shape the sound. Sams (1999) adds that signals are passed through filters which shape the sound "by 'taking away' (subtracting) frequency components present in the original sound". The following are the most important subtractive synthesis modules (also called unit generators (UGs)) and are categorised according to their function:

[36] These terms are events taken from the Java language's Mouse-Motion Listener interface and discussed by Campione (2004).

To produce and process sound:

- Oscillator: produces simple mathematical repetitive waves, such as sine, triangle, square, pulse and random noise.
- Filter: processes sound by eliminating and emphasising variable bandwidths of harmonics around the cut-off frequency. The most common filters are lowpass, highpass, bandpass and bandreject (notch). Resonance, also labelled Q or Response, can be used to emphasise the cut-off frequency.
- Amplifier: controls the amplitude or sound level of the signal.

To modulate sound:

- Envelope: modulates various parameters, such as amplitude, pitch and filter cut-off. Its time period is connected to when a key is pressed or released and the character depends on the attack, decay, sustain and release (ADSR) stages.
- LFO (Low Frequency Oscillator): also modulates various parameters with repetitive period.

3.2.2 Clavia Nord Lead 2 virtual analog synthesizer

The Swedish company Clavia,[37] made an impression in the analog synthesizer world in 1994 when they introduced the Nord Lead, "the first modelled-analog synth to make it to market" (Vail: 2000). A "virtual analog"[38] design philosophy based on an analog synthesizer's building blocks and interface amalgamate to form a digital instrument. The *Nord Lead 2* (NL2), which is used in this system, appeared in 1997 and incorporated a range of improvements on the previous model. The NL2 interface uses 25 knobs, 21 buttons, pitch bend, modulation wheel and four-octave keyboard, which can all be controlled with MIDI. With no hidden hierarchical menu interface, each button and knob is linked to a single parameter enabling quick and easy manipulation. The NL2 is the synthesizer of choice in research projects and educational programs at a range of institutions.

[37] Refer to http://www.clavia.se.

[38] "The virtual analog concept combines classic ideas with today's technology. Instead of analog components we use digital signal processors (DSP) that we program to emulate analog synthesis." Clavia (1996-2004)

3.2.3 The SystemOne application

This application strives to make use of a larger two-dimensional area as in SystemTwo and SystemThree. The user moves around inside the capturing view of the camera. See Figure 2 below. As will be discussed in chapter four, a dark background, with the object to be tracked in a lighter colour, is used. The accuracy of the tracking data of the object is highly dependent on the contrast between the background and the object. In the absence of a dark background, the same degree of contrast can be generated by using two electrical light sources such as commercial laser pointers, each representing a hand position.

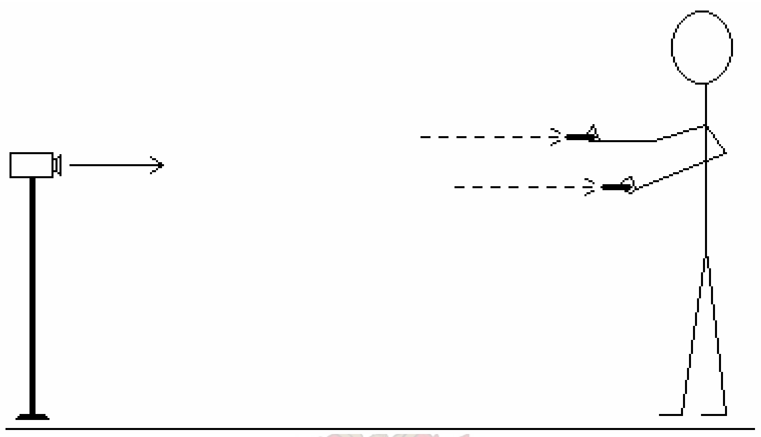


Figure 2. Position of the user and camera.

For the purpose of mapping, an on-screen simulation (see Figure 3) of the NL2 interface is designed and drawn as a background image. For practical reasons concerning gesture tracking, five controlling regions are superimposed upon the simulated background image. These five regions (two selector panes, two semi-transparent rotary knobs and one keyboard) can be manipulated if the light sources are moved into these regions. The background image is used to show the status of buttons and positions of knobs. By using the selector regions at the top and right-bottom of the screen, a button can be pressed or a knob can be associated with a rotary knob region. The two-octave keyboard can be played with the left or right hand, and is provided with a latch button, to sustain a maximum of eight depressed notes. By navigating through and modifying the various controllers, the application transmits the applicable MIDI code to the NL2.

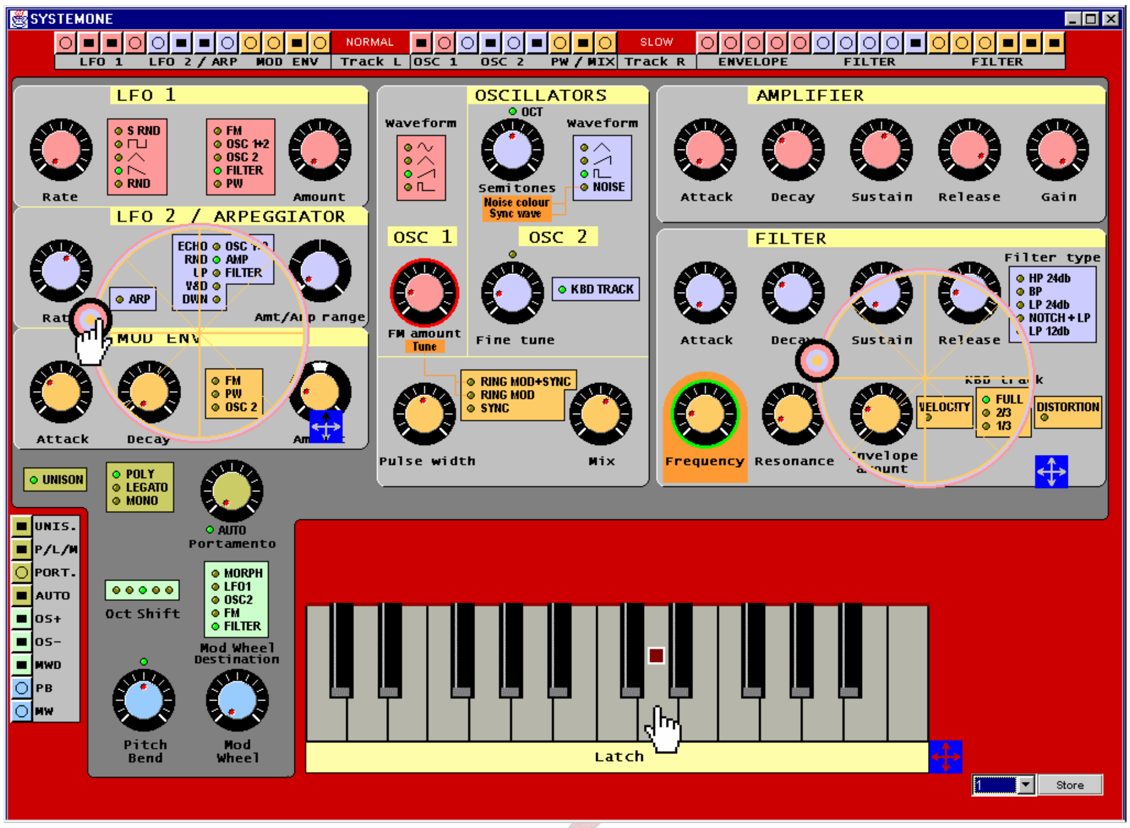


Figure 3. A snapshot of the graphical interface of the SystemOne Java application.

A Java program as mapping software receives MIDI code from EyesWeb and transmits MIDI code to the NL2 synthesizer. The NL2 is connected to a stereo amplifier, which is connected to two speakers. See Figure 4 below.

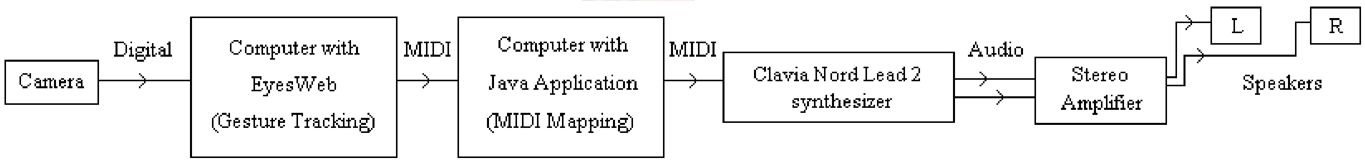

Camera
Digital
Computer with EyesWeb (Gesture Tracking)
MIDI
Computer with Java Application (MIDI Mapping)
MIDI
Clavia Nord Lead 2 synthesizer
Audio
Stereo Amplifier
L
R
Speakers

Figure 4. Basic hardware structure flow of SystemOne.

3.3 VIRTUAL DJ (DISK JOCKEY) EQUIPMENT ENVIRONMENT (SYSTEMTWO)

This environment was selected as part of the thesis, firstly because of the author's interest in some of the dance music genres[39] utilized by DJs. Secondly, because of the need for research in the techniques of the DJ milieu[40] in order to create new control interfaces.

> "…to make a virtual turntable that is controlled with existing or specially designed input devices. From a playing practice point of view, this approach is straight-forward as the performer will need to learn to play in the way as on real equipment." Hansen (2003)

[39] Hoggarth (2002) identifies genres such as "House", "Techno", "Trance", "Garage", "Jungle", "Drum & bass", etc.
[40] Hansen (2003) embodies the closest approximation to an academic analysis of DJ-ing and for example states that "…the acoustics of scratching has been barely studied until now."

For a clear and successful demonstration of SystemTwo, it is necessary for the reader to understand the basics of the DJ environment on which the system is based. This section introduces the equipment used by the DJ, the manner in which the equipment is used, and provides an overview of the working and developing environment of the DJ. The equipment discussion focuses on basic features, while a detailed discussion of the features used in SystemTwo follows in section 5.7.

3.3.1 Definition of a DJ

At a fundamental level Slaney (2002) defines a DJ as "…a person who plays recordings of music (which have been made by other artists) to audiences, who listen and/or dance." However, it has to be kept in mind that the role of the DJ developed[41] throughout the course of the last three decades. A wide range of different types and levels of skills[42] can be associated with this art form. The DJ is often also the producer of the recordings and apart from a turntable, a varied selection of equipment[43] is incorporated into an act. Although somewhat nebulous, the duties of the DJ can be categorised in the following (probably in order of least to most technically skilled) order, although a degree of correlation exists between the various roles:

- Mobile DJ, which plays all types of dance/party music at functions, such as weddings.
- Radio DJ, which basically presents music and talk shows to radio listeners.
- Club DJ, which plays tracks following each other by using several techniques, such as *beat matching* (discussed in section 3.3.3), to combine tracks so that it results in a continuous musical procedure.
- Scratch DJ (also called "hip-hop disk jockey" and "turntablist") which makes more physical contact with the medium with techniques such as scratching[44] and beatjuggling resulting in a performance where the focus falls more on the editing and creative manipulation of recordings.

Because of the latency inherent in SystemTwo and the available level of accuracy of gesture tracking the system will be based on the role of the Club DJ, simulating all the basic equipment used by this type of DJ.

[41] In addition, the turntable is the DJ's primary instrument. Souvignier (2003) states that "it has grown from a consumer playback device into a professional musical instrument".
[42] "…it is not just chaining tracks together into one long continuous piece of music." (Pluijms: 2002)
[43] Such as synthesizers, samplers, grooveboxes, acoustical instruments, vocals, etc.
[44] "…art form of manipulating a vinyl record by rhythmically dragging and pushing it…" (Hansen: 2002)

3.3.2 DJ Equipment

The primary set-up for a DJ is two direct-drive[45] turntables,[46] or two CD players, that are connected to a DJ mixer and are placed in front of the user as illustrated in Figure 5.

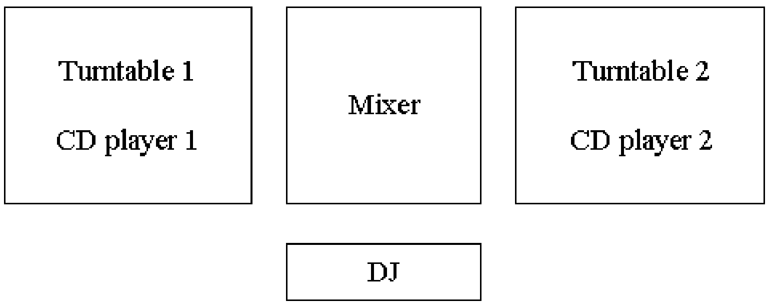


Figure 5. Basic setup of DJ equipment and position of DJ.

SystemTwo simulates the primary set-up's equipment and focuses on CD players rather than turntables. The system is then connected to headphones and an amplifier which is connected to speakers.

Both the turntable and CD player are equipped with a pitch slider to change the rotation speed in the range of –8% to +8%. The CD player, is amongst others, equipped with a Cue[47] button, to store a specific point in a track which can be instantly located, and a *jog wheel* to search or spool easily through a track. Depending on the model, the DJ CD player can have an unlimited number of features.[48]

The basic DJ mixer incorporates two *channels*,[49] each with an *Equalization*[50] (EQ) section, *Fader*[51] and *Cue*[52] facilities. These two channels are connected to a horizontally-placed *Crossfader*[53] in order for a mixture of the channels to be sent to the master audio output.

45 Belt-drive forms the other option.
46 The Technics SL-1200 is the industry standard.
47 Note that two definitions for "Cue" exist. In this case, Sams (1999) states that "to 'cue' a selection of music (or a sound effect) is to position the playback device (a tape or CD, for instance) at the exact beginning of the desired selection". See also footnote 52.
48 An example is the Pioneer CDJ1000, which is a leading model with a seven-inch scratch platter.
49 In the DJ mixer circle the term "channel" refers to a stereo signal.
50 Sams (1999) states that "Equalization" refers to a "circuit or device for frequency-selective manipulation of an audio signal's gain".
51 Sams (1999) identifies a "Fader" as a "potentiometer or similar software construct, with a linear (or rotary) control, typically used for reducing or augmenting an audio level".
52 Note that two definitions for "Cue" exist. Sams (1999) states that "Cue Send", which is used for beat matching, is sometimes abbreviated to "Cue" and that "this function is also very common on DJ and broadcast mixers, as it can be used to preview or "Audition" material before it is played back through the main audio outputs". See also footnote 47.
53 Sams (1999) defines a "Crossfade" as an "overlapping volume fade before/after the boundary between adjacent audio segments. Further, on a Turntablist mixer different fading curves can be selected by using a hamster-switch.

At this point it is necessary to mention a new approach, namely the use of professional DJ software[54] by professional and highly respected DJs,[55] that lead to the selection of this environment as part of this thesis. Native Instruments' *Stanton Final Scratch*,[56] one of the most commonly used software packages allows any audio file in WAV, AIFF (Audio Interchange File Format), and MP3 (MPEG Audio Layer-3) format to be scratched. All tracks are stored on the computer's hard disk in a process analogous to storage on vinyl records and CDs. The DJ still uses turntables or CD players but makes use of special records and CDs containing time code to manipulate tracks loaded in the software.

Broughton & Brewster (2003) explain that "cueing" in the DJ world encompasses two different actions.[47] Firstly, it refers to the DJ mixer's Cue facilities used for listening to the next track over headphones and enabling this track to be beat matched with the current track playing. Each channel has its own Cue button to send the pre-fader signal to the headphones, and on some models there are "kill switches" (Slaney: 2002) to eliminate bass-, mid- or top frequencies. The mixer also incorporates a Cue level knob and some models have a Split-Cue function, with which channel one is routed to the left ear and channel two to the right, or both channels to both ears. Secondly "cueing" refers to the storage a specific point in a track from where the track must play, which is part of the beat match process.

3.3.3 The Beat Matching process

The aim of this section is to familiarise the reader with the functionality of SystemTwo by explaining the beat match process using CD players. As explained elsewhere, it is the club DJ's responsibility to create a musically satisfying continuous composition[57] by using different tracks. Souvignier (2003) states that beat matching[58] (also called "beat mixing") "entails adjusting a turntable's speed (the pitch control) so that the tempo of one record matches another." Numerous ways to accomplish beat matching and dependent on the DJ's own musicality, creativity and dexterity, exist. One basic procedure is discussed below.

[54] Such as Soundgraph's D-Vinyl2020™ (http://www.soundgraph.com),
Native Instruments's Traktor DJ Studio (http://www.native-instruments.com),
Serato's Scratch (plug-in for Digidesign Pro Tools) (http://www.serato.com) and
American DJ's Pro Mix™ Dual MP3 controller (http://www.americandj.com).

[55] For example Paul van Dyk (http://www.paulvandyk.com) and Josh Wink (http://www.joshwink.com).

[56] Refer to http://www.finalscratch.com for more information.

[57] "Francis Grasso was the first DJ to present an uninterrupted flow of music, where the dancing never had to stop. To make this happen, Grasso used two copies of a record, one on each turntable." (Souvignier: 2003)

[58] "…the technique of smoothly layering records with no interruption or variance in the beat." (Souvignier: 2003)

The beat matching process can be divided into the following three steps, of which the first two are monitored over headphones.

- Cueing the track by storing the cue position.

  The *Cue Button* on a CD player has two functions. When in pause mode, the current position is stored when Cue is pressed. While playing, the track will suddenly play from the stored position if Cue is pressed, but will go back to the stored position and pause, when Cue is released. A suitable position must be found, which is usually the first kick drum of the first measure of a phrase. The jog wheel must be used to scan back and forth through the track.
- The actual beat match.

  The track is played from the cue position, by synchronizing it with the other CD player's track. The pitch bend slider and pitch bend buttons are used to match the tempo with the other track. Each time a pitch change is made, the track must be started again from the cue position. This is not an easy process, but several aids[59] exist to improve and hone this skill.
- The Transition.

  Once the two tracks are playing together, the crossfader can fade in the new track and fade out the previous track. Depending on the content of the track, the transition can be a creative and skilled process. Various techniques[60] exist, the slow removing of bass frequencies from the outgoing track while the new track is faded in being the most common.

3.3.4 The SystemTwo application

This application attempts to simulate the DJ equipment environment. Therefore the camera is mounted in such a way that the user's workplace is more or less at the same height and angle as the DJ equipment. White objects such as simple paper finger hats, are attached to the thumb and index-finger of both hands. As fairly accurate data is to be tracked from the hands, a fluorescent light bulb is used to enhance the positions of the four white objects. The light is mounted next to the camera, lightening the objects at the same angle as captured by the camera.

[59] Such as EQ "kill switches", Beats per minute (BPM) display and waveform display.

[60] For example "Chop mix", "Beat mix", "Phrase mix", "Phase mix", "Shadow mix", "Ambient transition", "Split Stereo transition", etc. (Slaney: 2002; Pluijms: 2002)

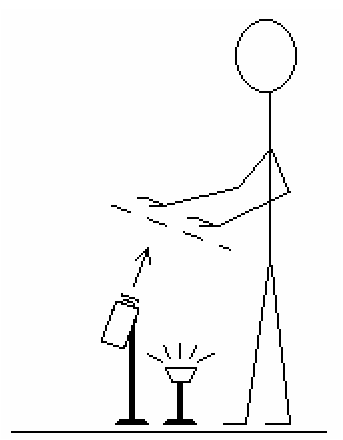


Figure 6. Position of the user, camera and fluorescent light.

With the thumb and index-finger held apart, the movement of the hand is interpreted as *free*, whereas *controlling* movements are tracked when they are held together. The type of movement is determined by independently calculating the total area, delineated as a bounding rectangle, by the white objects of each hand. The central point of the bounding rectangle is interpreted as the position of the hand. The user's hands have to be held so that the fingers, apart or together, generate the same two-dimensional position for each. Figure 7 illustrates images captured by a camera and modified by EyesWeb, which also splits the image into left and right hand images. Refer to chapter four for a discussion on the program flow.

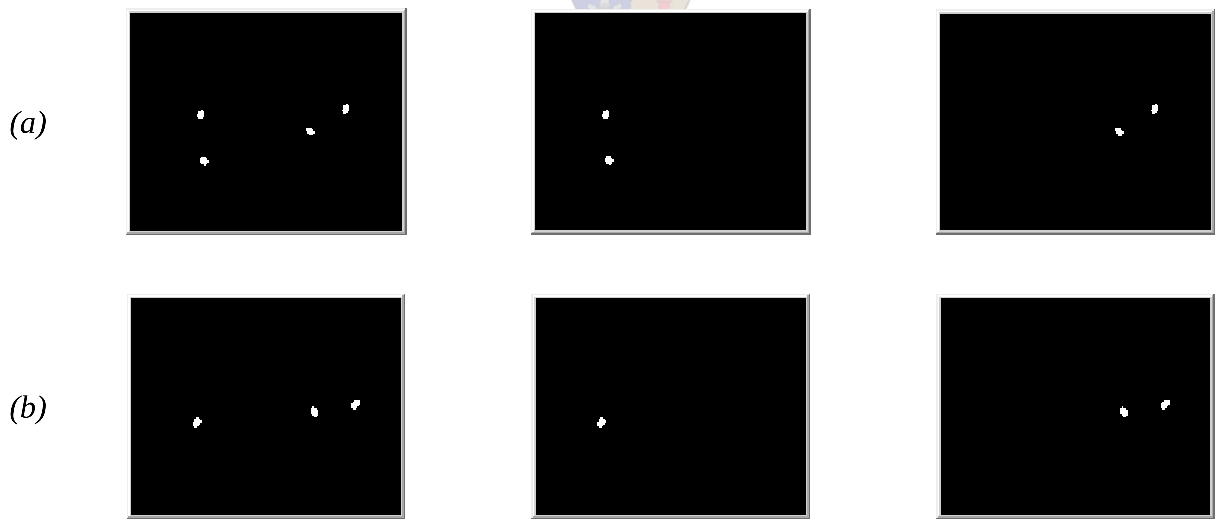


Figure 7. Images captured by the camera and modified by EyesWeb. Row *(a)* illustrates two open hands, followed by the left and then right hand extracted images. Row *(b)* shows a closed left and open right hand, followed by the left and right hand extracted images.

The graphical interface is designed to act as a simulation of the applicable DJ equipment (see Figure 5) that includes the use of CD players. Each individual button or fader constitutes a control region and can be manipulated if *controlling* movement data is received. It is not possible to incorporate the exact rectangular appearance of the equipment on a computer screen. Firstly, because of the small size and almost square dimension of the screen and secondly the user's virtual workplace is to be congruent with the computer screen. To address this shortcoming, two modes, i.e. *Normal* and *Jog wheel* mode, are illustrated in Figure 8 and Figure 9. By entering one of the two jog wheel regions in Normal mode, Jog wheel mode is selected, giving the user more control to manipulate the loaded audio. A *controlling* movement inside the *Exit* region activates Normal mode. In the latter the bottom of the image is reserved for viewing, selecting and loading available audio tracks into the system.

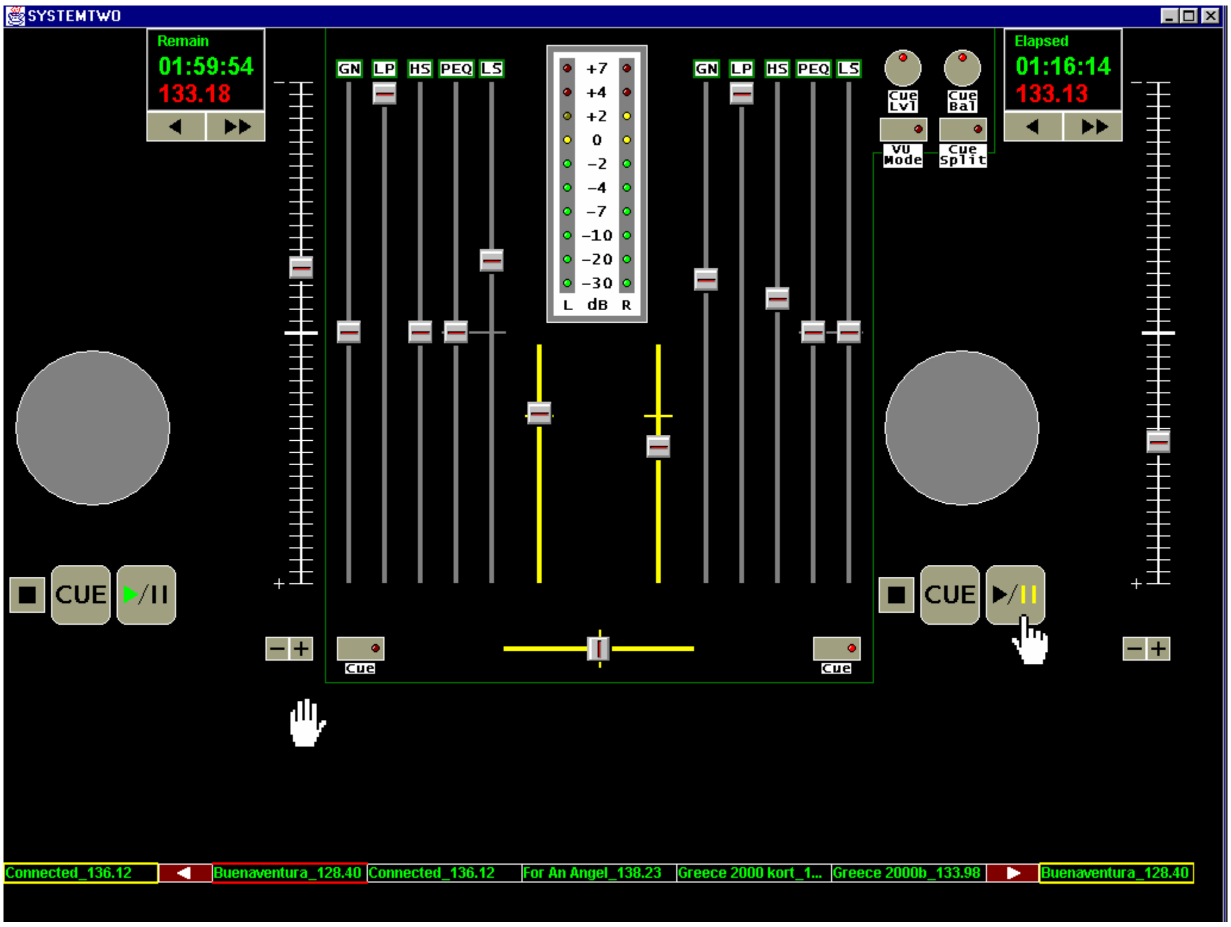


Figure 8. A snapshot of the graphical interface of the SystemTwo Java application (Normal mode).

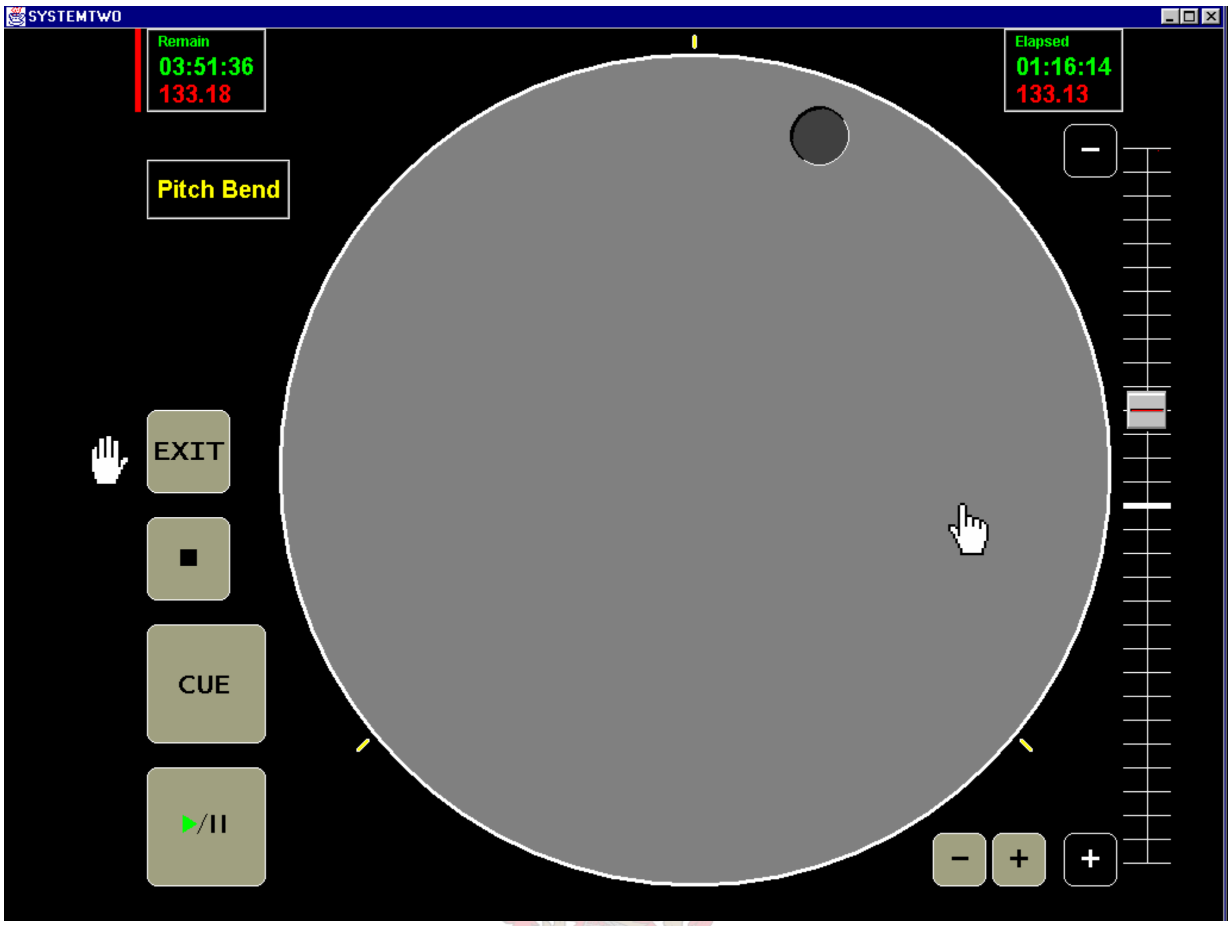


Figure 9. A snapshot of the graphical interface of the SystemTwo Java application (Jog wheel mode).

As opposed to SystemOne and SystemThree, which both make use of an additional device to generate sound, SystemTwo uses Jsyn (discussed in section 4.4) that is embedded within the Java mapping software. The particular soundcard of the computer running Java and Jsyn has eight audio output channels, of which four grouped into two stereo pairs are used. The first stereo pair acts as the master output and is connected to a stereo amplifier, which feeds into speakers for the audience. The second is reserved for cueing and is routed to the headphones used by the DJ. See Figure 10.

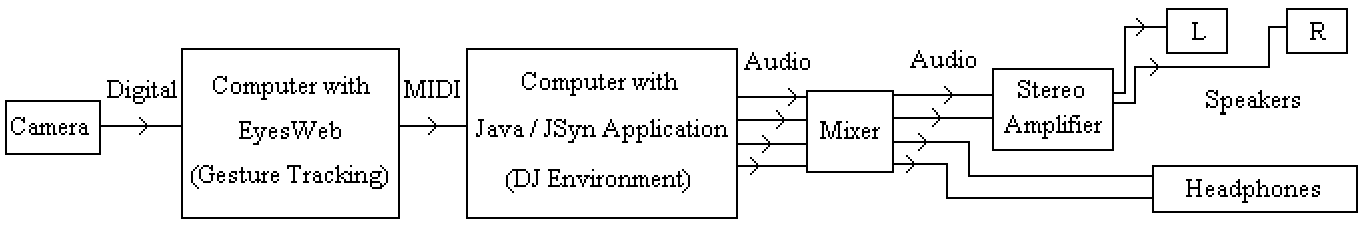


Figure 10. Hardware structure flow of SystemTwo.

3.4 PRO TOOLS MIDI CONTROLLER (SYSTEMTHREE)

3.4.1 Background

As mentioned elsewhere, the three systems use hard- and software that are available and in every day use. The main recording facility in the Konservatorium recording studio is a Pro Tools 24 Mix Plus system and it was obvious to use Pro Tools as sequencer in SystemThree. SystemThree is programmed as part of the KEMUS[61] live act for the composition *Skadu-musiek 2*[62] by Theo Herbst.

Digidesign (2001b) states that a MIDI control surface "add hands-on control to Pro Tools functions", such as faders, knobs, transport controls and also scrub and shuttle functionality.[63] Although SystemThree only manipulates the EQ frequency of a plug-in and panning in a quadraphonic environment, it makes use of the advanced features available in the *Pro Tools MIDI Controller Personalities*[64] and is very simple to expand to incorporate numerous other features.

According to Digidesign (2001b), Pro Tools is designed so that a third-party control surface[65] can be used, if it is programmed to function like the *JL Cooper CS-10* and emulates the CS-10's functional personality. Therefore, if the Pro Tools MIDI Setup is set to the latter as control surface, any MIDI code sent to the MIDI interface, that is the same as valid CS-10 MIDI code, will control the Pro Tools system in the same manner as the CS-10 does.

[61] The "Komitee vir Eietydse Musiek" or contemporary music society ensemble consists of a group of local musicians dedicated to the composition and performance of experimental contemporary electronic music. Past programs included performances of "Mikrophonie 1" (1964) by Karlheinz Stockhausen (b. 1928), "mit / gegen sich selbst" (1969) by Erhard Karkoschka (b. 1923) and "Dialoog" (2003) and "Skadu-musiek 2" (2004) by Theo Herbst (b. 1965). The ensemble has performed in Stellenbosch, Grahamstown and Johannesburg.

[62] The work incorporates the playback of a large number of pre-designed sound files as well as live piano and viola, captured, panned and filtered respectively using SystemThree. The first performance occurred on 21 July 2004 in the Fismer Hall as part of the first International Chamber Music Festival hosted by the Konservatorium. This was most probably the first time that EyesWeb was used locally in a performance.

[63] Digidesign (2001a) defines "scrubbing is a technique that originated in tape editing, where the tape was rocked back and forth past the playhead at slower than normal speeds to find a particular location". Sams (1999) identifies "shuttle" as "the rapid forward or backward playback of the video program" and in the case of Pro Tools the audio.

[64] "*MIDI Controller Personalities* are files that allow Pro Tools to communicate with MIDI controllers such as the Mackie HUI™, any of the JL Cooper CS-10™ series, the Peavey PC-1600™, and the Penny & Giles MM-16™ / DC-16™." (Digidesign: 2001b)

[65] Such as the Yamaha AW4416 Audio Workstation. http://www.yamaha.com

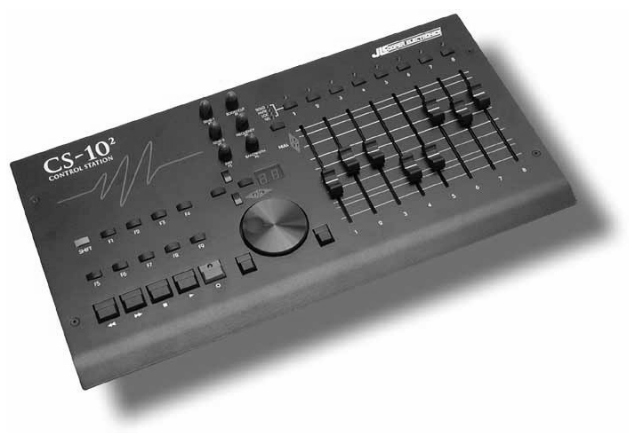


Figure 11. The JL Cooper CS-10 control surface.
(Adopted from http://www.jlcooper.com/images/photos2/jpg/CS_10_2.jpg)

The JL Cooper CS-10 unit is an eight-fader control surface that acts as a controller for Pro Tools. Together with optional *CS-10^x* units, it can be expanded to contain a maximum of 64 faders. The CS-10 incorporates all basic functions such as "play", "stop", "rewind", "fast forward", "solo", "mute", track fading, zooming, scrubbing, basic editing, etc. and also offers control over plug-in parameters. It is important to understand the CS-10's functionality fully before the mapping application can be programmed. The following are some of the features and restrictions for all MIDI controllers, followed by the CS-10 in particular.

For all MIDI Controllers:

- ➢ Only one track's *Output Window*, used for track panning and *Sends*, or one *Plug-in Window*, can be used at a time.
- ➢ The control of different plug-in parameters depends on the current *control page* selected. A control page consists of a combination (six on the CS-10) of controllable parameters. The other control pages combine the remaining parameters. The user has to step through these pages until the required parameter group, which incorporates the required parameter, is selected.

For the CS-10:

- ➢ There are six rotary knobs controlling pan, sends and plug-in parameters.
- ➢ The panning and sends of a selected track ("focused track") are controlled by the rotary knobs if the top control page is selected, otherwise the track with the selected plug-in ("target track for plug-in editing", further called "target track") is influenced by controlling the plug-in's parameters while also changing the control page.
- ➢ The track name of a normal track is outlined in blue, the focused track in green and the target track in red.

3.4.2 The SystemThree application

This application focuses on a Human-computer Interaction (HCI)[66] implementation while the gesture tracking functionality is based on a virtual mouse concept. The hand lightly tracks across a flat surface, such as a desktop, with an open hand representing free and a closed hand as controlling movement. A camera focused on the flat surface captures the hand movements (see Figure 12 and Figure 13). Refer to chapters four and five respectively, for a discussion on the software implementation of tracking the hand movement and the transformation of this movement to structured MIDI code.

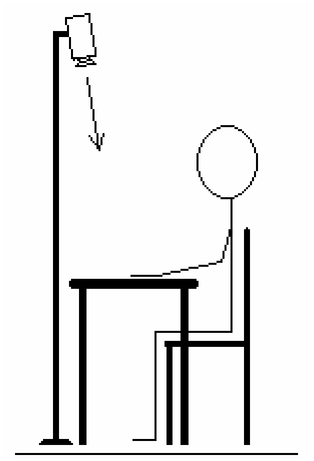


Figure 12. Position of the user and camera.

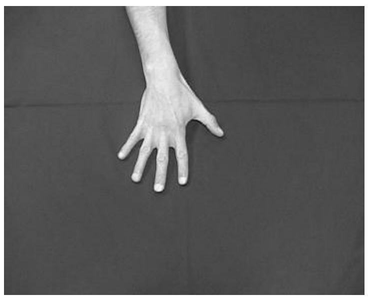


Figure 13. View of the camera.

The graphical interface of the mapping (see Figure 14), which was decided upon for *Skadu-musiek 2*, consists of two control regions, i.e. a controlling square surface (in the centre) and a toggle button (top- left). The latter toggles between two modes of which the first controls quadraphonic panning and the second a plug-in's EQ frequency. In panning mode the control square offers a top view of a quadraphonic setup, with four speakers distributed between four corners. When the EQ mode is selected, the x-axis of the control square is used to represent the center frequency, while the y-axis is ignored. The control square is manipulated by moving the ball image, which is representative of the position and status of the square. Depending on the mode and position of the ball image, MIDI code is generated and sent to Pro Tools. The limitations imposed by the Pro Tools MIDI controller on the outgoing MIDI code are discussed in chapter five.

[66] Kirk & Hunt (1999) defines HCI as "the study of how human beings use computer systems. It has emerged as a subject area of increasing importance as computing technology has developed."

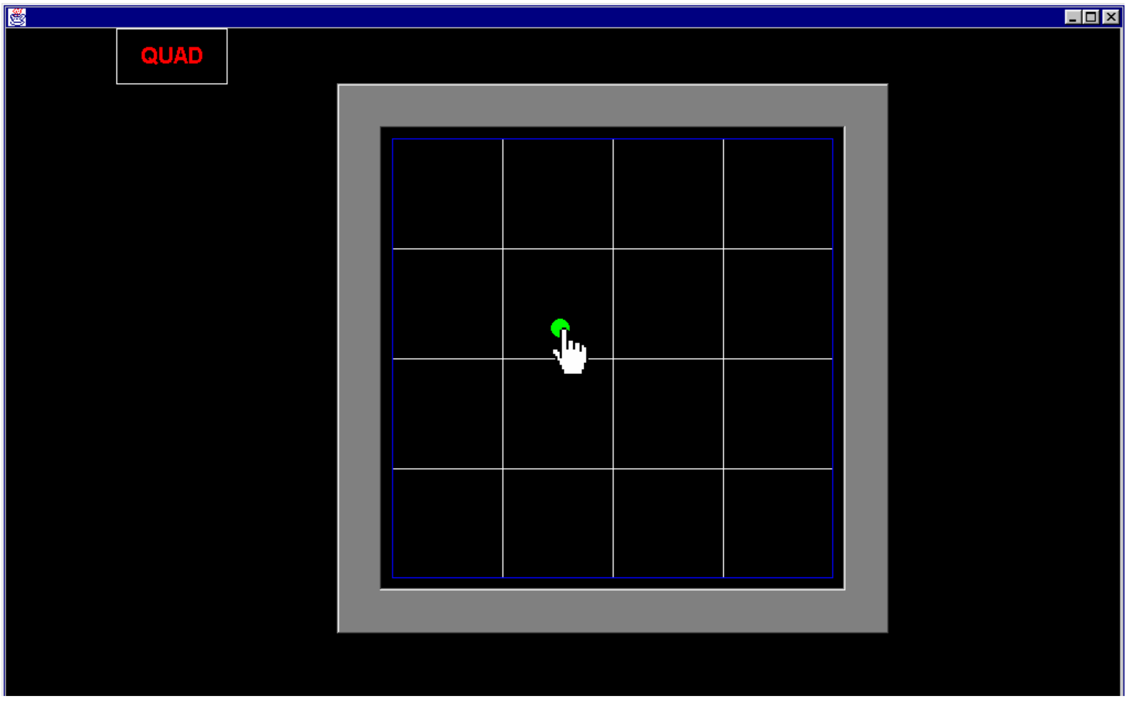


Figure 14. A snapshot of the graphical interface of the SystemThree Java application.

Mapping software, programmed in Java, receives MIDI code from EyesWeb and sends valid CS-10 MIDI code to the Pro Tools external MIDI hardware interface. The MIDI code manipulates parameters in the Pro Tools session and the audio is sent to two stereo amplifiers, which are connected to four speakers. See Figure 15 below.

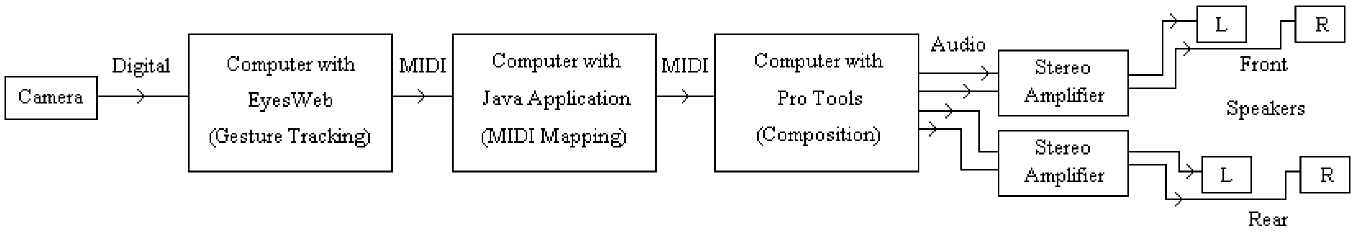

Camera
Digital
Computer with EyesWeb (Gesture Tracking)
MIDI
Computer with Java Application (MIDI Mapping)
MIDI
Computer with Pro Tools (Composition)
Audio
Stereo Amplifier
Stereo Amplifier
L
R
Front
Speakers
L
R
Rear

Figure 15. Hardware structure flow of SystemThree.

3.5 HARDWARE USED BY THE THREE SYSTEMS

This section provides a list of the hardware used in the design, experimentation, programming and installation of the three systems as discussed in the above application sub-sections. The order in which the hardware is specified emulates the direction of the dataflow, from the user to the audio output.

Panasonic NS-DV38 digital video camera
2 x LOHUIS 18 W fluorescent tube lights
2 x laser pointers
FireWire cable with DV and FireWire connectors

Intel Pentium 4

2.8 GHz (2 CPUs), 510 MB RAM

Creative Sound Blaster Live!^{TM} 5.1 card

Microsoft Windows XP Professional

Gameport MIDI cable

Intel Pentium III

1000 MHz, 512 MB RAM

SEK'D SIENA[67] Audio/Midi Recording Card, 8 analog I/Os up to 24-bit / 96 kHz

Microsoft Windows 98

MIDI cables

Power Mac G4

733 MHz, 384 MB RAM

Mac OS 9.2

Digidesign Pro Tools 24 MIXplus system (Pro Tools 5.1.1)

Roland UM-4 (64 Channel USB MIDI Interface)

Clavia Nord Lead 2 virtual analog synthesizer

[67] The sound card has undergone a name change according to Marian (2004). “Version 2.13: Because of legal reasons the name "Siena" cannot be used anymore. The existing hardware and related drivers are now part of the product "Marc 8 Midi".” This fact is corroborated by Marian (2003) who states that “MARC 8 Midi: 02-13-2002: The "Siena" hardware and drivers, which were developed by MARIAN, are now part of the MARIAN product "Marc 8 Midi". The current and future "Marc 8 Midi" drivers are fully compatible to the existing "Siena" hardware.”

CHAPTER FOUR
THE PROJECT'S SELECTION OF SOFTWARE

4.1 INTRODUCTION

As was stated in chapter one, the criteria for this thesis revolve around the fact that all software (fully operational packages and not so-called "demo" versions) had to be freely downloadable from the internet. In addition, software usage had to fall within the framework of the applicable license agreements. Chapter four is devoted to an explanation of the software selection, while the hardware to be used appeared in chapter three. Software suitable for image tracking, an *Integrated Development Environment* (IDE), graphics, sound synthesis and computer communication, is selected. This chapter will focus on the software's characteristics, whereas chapter five will discuss how the software was used to solve problems.

4.2 EYESWEB

EyesWeb[68] is a visual software environment which allows for the connection of software patches, represented by graphical blocks,[69] to create a new real-time program. It provides a tool for musicians and researchers to create new interfaces between humans and other systems. EyesWeb is a multi-thread application, which runs on Win32 and is based on the Microsoft COM standard.

The leading EyesWeb blocks used in this thesis can be categorised in the following libraries:

- *Imaging*: for capturing incoming video, converting images, splitting images, calculating co-ordinates of blobs, etc.
- *Math*: for basic mathematical operations, working with matrixes or scalars.
- *MIDI*: for building MIDI messages and output.

Other categories are:

- *Generic*: such as delay on any data type.
- *Network*: for network input and output.
- *String*: a text-based environment.
- *Serial*: transport with serial devices connected to the computer.

68 The version used for this thesis is "3.1.0".

69 "Software components that carries out operations on ordered data input, supplying data output." (EyesWeb User's Guide)

All three systems comprising this thesis use basically the same approach to the patch complex design. The latter, its constituent software blocks as well as their interaction, are attached as Appendix C, and discussed below by way of the SystemThree patch. As was explained above, EyesWeb essentially computes the two-dimensional Cartesian co-ordinates of an object, which can be determined when the image consists of an object against a sufficiently contrasting background. Parameters appearing in the following sections, will not be allocated specific values, because these parameters differ and must be set according to light, background, camera distance from surface, hand size, etc. Refer to the EyesWeb documentation accompanying the software and the IPL 2.5 manual[70] for further information on blocks.

4.2.1 Imaging

A digital video camera is used to capture the movement from the user. The camera has a DV (Digital Video) output, which can be connected with a cable to a FireWire port on the computer. All video and audio data are then sent digitally from the camera to the computer. The operating system interprets the camera as a hardware device connected to the computer. Inside the EyesWeb patch the *Imaging.Input.DVCamInput* block is used to read the data entering the FireWire port and output data which can be viewed in a different window.

The *Imaging.Output.Display* block is used to display the IPL image in a popup window, which is obtained from imaging blocks. The Display block is very useful to get an idea of how images are changing before and after conversion and operation blocks. The following illustrations (Figure 16 to Figure 21) were acquired by copying the various popup windows.

Imaging.Conversion.ColorToGray converts as a first step a 24-bit color[71] image to an 8-bit grayscale image,[72] allowing less data to be transported. According to the camera setup discussed in section 3.4.2, the initial image has to be rotated or mirrored around the vertical and/or horizontal axis, by using the *Imaging.Operations.Mirror* block. This action demonstrated in Figure 16 obviously facilitates the setting of parameters for the following blocks and provides for a logical visual perspective.

[70] Intel Corporation. (2000) Intel® Image Processing Library Reference Manual. Available at http://www.cs.aue.auc.dk/~rea/eyesweb

[71] Three colors / channels, Red-Green-Blue (RGB) are used, with 8 bits allocated to a color / channel.

[72] EyesWeb refers to the number of bits used as “depth”, for example 8-bit equals a depth of 8.

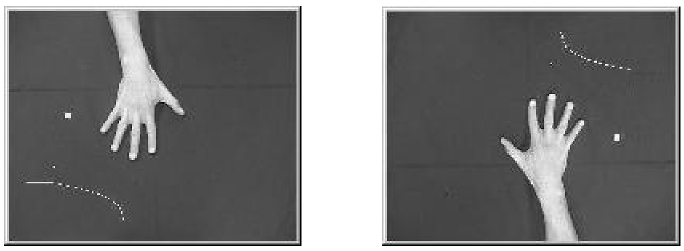


Figure 16. Before and after horizontal and vertical mirroring.

The next step is to convert the 8-bit grayscale image to a 1-bit, bitonal, black and white image, which is done by the *Imaging.Operations.ThresholdBitonal* block. This block has a threshold parameter, with which each pixel is replaced with simply black (0) or white (1). See Figure 17.

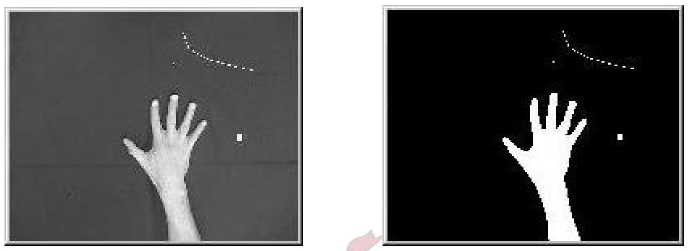


Figure 17. Before and after a threshold is implemented.

The *Imaging.Filters.NonlinearFilter* block is used to change small unwanted white objects or noise to black therefore removing them from the image. The filter type is set to *filter median*, and a median for each pixel is calculated according to the pixels in the neighbourhood, which is the size of *number of columns* by *number of rows*. See Figure 18.

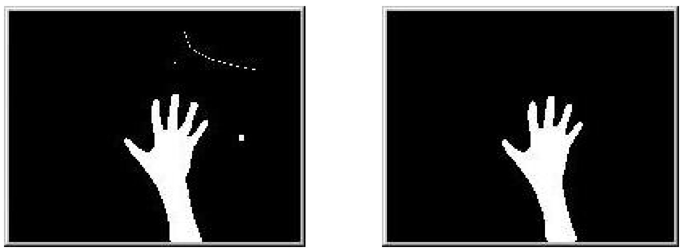


Figure 18. Before and after a median filter is used.

For SystemThree only information about the hand itself is needed, therefore the part of the forearm that is present in the image must be removed. The *Imaging.FeatureCalc.BoundingRect* block determines the bounding rectangular area of the hand and outputs the co-ordinates of the left-top and right-bottom points. Usually the top of the left-top co-ordinate represents the top of the middle finger. The vertical distance from the top of the middel finger to just below the thumb is to be set, whereafter the distance is added to the left-top point. This new value is the y co-ordinate where the image is to be split and then only the top halve is used. The split process is accomplished by using the *Imaging.Operations.Split* block. In SystemOne and SystemTwo the image is also split in left and right images, representing the left and right hand. See Figure 19.

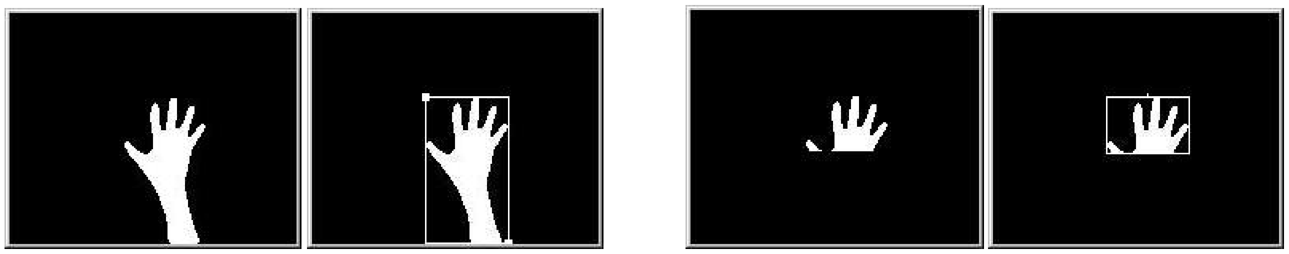


Figure 19. Before and after the image is split, bounding rectangles.

As discussed above, for SystemThree an open hand has the same functionality as a mouse movement, while a closed hand represents a mouse drag or click. The *Imaging.FeatureCalc.BoundingRect* block is again used to determine the width of the portrayed open or closed hand. Thereafter, the horizontal centre and top of the hand is used as the co-ordinates to be sent to the mapping software. See Figure 20.

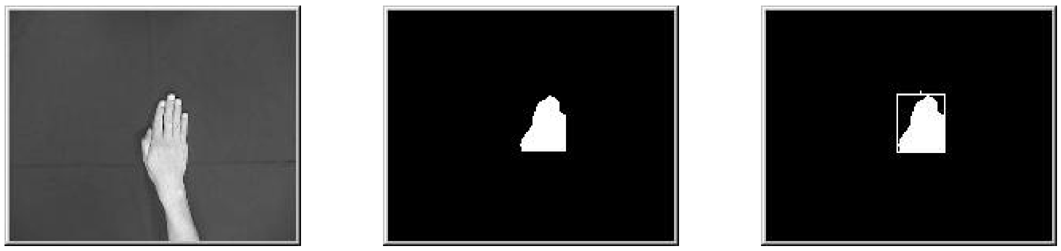


Figure 20. After mirroring and after splitting (closed hand), bounding rectangle.

If a background surface is used, which is a lighter colour than the colour of the hand, the *Imaging.Operations.MonadicLogicalOp* block is used to change black pixels to white and white pixels to black. The *operation type* of the block is set to “not”, which gives the complement of each pixel. See Figure 21.

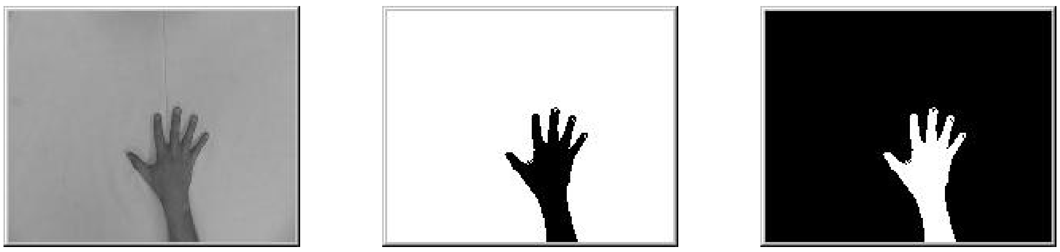


Figure 21. The original image, after threshold, and after the logical operation.

4.2.2 Math

This subsection involves the overview of blocks used to prepare the co-ordinates to be sent as MIDI messages. However the flow of the mathematical operations will be discussed in the next chapter.

The *Math* blocks are divided into *Scalar*, representing integer and real numbers, and *Matrix*, which is an organized group of scalars. The discussion will focus on some of the *Scalar* blocks, followed by the *Matrix* blocks.

A very useful block for observation is *Math.Scalar.Display* and displays a value sent by other scalar blocks. *Math.Scalar.ConstOp* is responsible for doing basic operations, such as addition, multiplication, storing the maximum value, etc. on a single input value. On the other hand, the *Math.Scalar.BinaryOp* block needs two input values in order for these basic operations to be executed. *Math.Scalar.UnaryOpBitwise* focuses on bit-type operations, such as and, or, xor, left shift, etc.

The *Math.Scalar.DomainConv* block converts integer numbers to decimal (real) and *vice versa*. This block is necessary when the following block is used, which is the *Math.Scalar.AlphaFilter* block, as the latter operates on real numbers. The alpha-filter can be used to stabilise incoming values. It uses the formula:

$$y(i) = y(i-1) + \text{Alpha} * (x(i) - y(i-1))$$

where x (i) is the current input, y (i) and y (i - 1) respectively form current and previous outputs, and Alpha is a parameter ranging between 0 and 1. When Alpha is 1, the output of this block is the same as the input; otherwise the previous output is modified with a percentage of the input value. Depending on the change of light, even if a hand is kept still, the *Imaging.FeatureCalc.BoundingRect* block can output small fluctuations which can be corrected by using the *Math.Scalar.AlphaFilter* block.

From the *Matrix* library, *Math.Matrix.Output.Display* is useful for observing the values sent by other matrix blocks. As, for example, the *Imaging.FeatureCalc.BoundingRect* block outputs co-ordinates in matrix form, *Math.Matrix.GetEntry* is helpful to extract a value in scalar form from the matrix. This is done by referring to the row and column number. The *Math.Matrix.Extract* block extracts a submatrix of specified size from the input matrix and outputs therefore a smaller matrix. With *Math.Matrix.SetEntry* a single value in the matrix can be set, again by referring to the row and column number.

4.2.3 MIDI

In EyesWeb, MIDI data is a 32-bit message, consisting of four bytes, of which the most significant byte, the fourth byte, is ignored. The remaining three bytes are called *Status byte*, *Byte1* and *Byte2*, in order of least to most significant. The co-ordinates, explained before, are prepared and built as a single scalar which is converted to a valid MIDI message using the *Midi.Conversions.ScalarToMessage* block.

When the hand is kept on the same spot, therefore providing the same co-ordinates, it is unnecessary to send the same values more than once. The *Midi.Operations.MessageCheckDuplicates* block is used to remove all duplicates, i.e. when the current value is the same as the previous value. Therefore, unnecessary messages are not sent through MIDI, uncluttering the MIDI ports of both the EyesWeb computer and the mapping computer.

Midi.Output.MessageOutput is applied to send the MIDI message to a specified MIDI device, which is in the case of this thesis the MIDI port of the computer's soundcard. The *Midi.Operations.MessageField* block can be used to analyse the message that is sent, by extracting the specific fields, which can be displayed by *Math.Scalar.Display*.

This concludes the section on EyesWeb, and the procedure followed to modify the co-ordinates in order for valid MIDI messages to be created by using the *Math* blocks, discussed below.

4.3 JAVA

This section introduces the reader to the Java programming language. It will explain where Java fits into the computer world and give a historical overview of the language. Programming fundamentals, such as selection, repetition, threads and graphics will be introduced. An important feature of Java is that it is an object-oriented programming (OOP) language. Therefore, an introduction will follow on OOP, explaining classes, methods and the most important techniques, such as *inheritance* and *polymorphism*. It is entirely beyond the scope of this thesis to explain and make these programming concepts understandable to the reader. The thesis will strive to introduce these techniques, firstly to assure the validity of Java as mapping language in this thesis. Secondly, so that the flow of the source code of the three systems, accompanying the thesis document, can be followed and thirdly it will justify the level of complexity of the three systems. It is important to understand that the term "program" refers to a piece of software, with a specific task, written by the programmer.

4.3.1 Background

Motivation

Turbo Pascal[73] was used for years in South African schools as the programming language in the Computer Studies Higher Grade course for grades 10 to 12. The Western Cape Education Department decided in 2000 that a new language had to be implemented. The options lay between Java, C++, Delphi and Visual Basic. In the process the author, then an active teacher, along with the body of provincial teaching staff was consulted and in 2001 Java was implemented, making the Western Cape the first province to change from Turbo Pascal to Java. Other provinces followed in later years.

[73] Koffman & Maxim (1991) explains that Turbo Pascal is a dialect of Pascal and was designed for the use on personal computers. They state that Pascal is a "high-level, general-purpose language developed in 1971 by Professor Nicklaus Wirth of Zurich, Switzerland".

The language transformation[74] in the Western Cape was mainly handled by Donald Cook, a lecturer from the Computer Science Department of the University of Cape Town. Here, Java is the main language during the undergraduate Computer Science course. Since 2002 Java has also formed an important part of the programming component of Music Technology at the Konservatorium. Therefore it is obvious that, locally, Java is a leading programming language on secondary and tertiary education level. The implementation of this language in a non-computer science environment was done in order for a degree of levelling with other disciplines and potential co-operation to materialise.

Java and its ancestor languages (concise history of Java)

In the early 1970s the C programming language was developed by Bell Laboratories to write the UNIX operating system with. Before C, operating systems were written in the low-level assembly language, for which other languages, such as FORTRAN and COBOL, were too slow. As stated by Perry (2000), C was often called a high low-level language, because of the high level of the understandable code and maintenance, but also the speed and efficiency that comes with the low-level communication with the hardware.

During the late 1970s and early 1980s the language C++[75] evolved from C. As Naughton & Schildt (1998) explains, the need for C++ revolved around the lack of complexity of C. He continues that this complexity enables a programmer to "deal with larger, increasingly complex programs using symbolic representations of the machine instructions." The main difference between C and C++ lies therein that the former contains only structured programming, while the latter for the first time implemented object-oriented programming, discussed later in this section.

[74] Regular workshops, where teachers were introduced to the similarities shared between Turbo Pascal and Java were conducted. Also, the new concept Object-oriented Programming was introduced.

[75] C++ was invented in 1979 by Bjarne Stroustrup, while working at Bell Laboratories.

Java was developed in 1991 by James Gosling at Sun Microsystems. According to Naughton & Schildt (1998) it was initially designed as a platform-independent[76] language, which could be used in electronic devices such as microwave ovens and washing machines. With the growth of the World Wide Web and the Internet, the Java language acquired an additional role. Programs written in Java could be executed on different central processing units (CPU) and on different operating systems, making the same program usable by a variety of computer systems connected to a network. Because C and C++ are such prominent languages, but not platform-independent and Java was derived from C++, it became a leading language on the Internet.

The basic characteristics of Java, as discussed widely in the literature,[77] are:

- Simple
- Architecture-neutral
- Object-oriented
- Portable
- Distributed
- High-performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

[76] The goal is to have a system so that the same program can be executed on different platforms, making such a system platform–independent.

[77] For a discussion around these so-called "buzzwords", refer to Gosling, J. & McGilton, H. (1997) *The Java Language Environment: Contents*. Sun Microsystems, Inc. http://java.sun.com/docs/white/langenv.

The Java Platform

Campione (2004) explains that a platform usually consists of hardware and software[78] in which a program can be executed. In the case of the *Java Platform*, it involves only software, which operates on top of the operating system platform. The Java Platform can be divided into the *Java Virtual Machine* (JVM) and the *Java Application Programming Interface* (Java API). The JVM is software, which is dependent on a computer's operating system, and is responsible for the execution of Java programs. The Java API is additional software packages that form part of a program and are necessary for compiling and interpreting a program. When a Java program is compiled, a *bytecode* file is created,[79] which can be transported over a network to be interpreted and executed by the JVM on a computer.

4.3.2 Programming Concepts and Techniques

The following topics are hierarchically sorted according to increasing level of difficulty and approximately in the same order as covered in relevant programming literature. As mentioned at the beginning of this section, the discussion of these programming concepts is of an introductory nature and aimed at a novice programmer prepared to practice the skills associated with them. This section will strive to make the source code readable, but it is advisable to refer to the relevant Java programming literature for further discussions. In order for certain concepts to be clarified it is sometimes necessary to refer to the relevant source code examples.

Statements

A statement can be defined as an instruction, which ends in a semicolon (";"). It can also be a heading (such as "if", "for", "class", etc.) for a next statement or statement block in which case a semicolon is not used.

Statement blocks

It is frequently necessary to group lines of source code ("code") together to form a block of statements. Such a block is declared with beginning and ending braces, "{" and "}",[80] and with a heading statement, such as "if", "try", "class", etc. to group some code together. The indentation method used in this thesis, places the braces of different blocks in the same column.[81]

78 A computer's operating system, such as Windows, Linux and MacOS, form part of the software of a platform.
79 A bytecode file can be recognized by the ".class" extension.
80 In Turbo Pascal "begin" and "end" are used instead.
81 Refer to Vermeulen (2000) for another option concerning indentation.

Comment

To make code understandable to other programmers, it is advisable to make use of comment. All comment are ignored by the compiler, and therefore do not have any influence on the code. A block of comment can be identified by

```
/* some code */
```

but in the case of a single line of comment

```
// some code
```

can be used.

Variables and objects

As stated in Hubbard (1999) variables and objects form two kinds of entities which can store data, and are declared in code with: the type of data, followed by the name (*type-name variable-name*), for example `char c`. A variable stores a singe value of a specific type, of which the following list includes some of the most used types (so-called "primitive types"):

- int (32-bit whole numbers)
- double (64-bit decimal number)
- boolean (true or false)
- char (16-bit Unicode characters)

An object usually comprises more than one variable, and is called an "instance of a class", for example with "`String s`", "`s`" is an instance of the `String` class. The `String` class is probably the most used primitive object and stores a series of characters.

Operators

Operators are used to change the value that is stored in a variable. Some of the most popular operators can be used in the following way:

- `i = 5`
- `i -= 3` (`i` is decremented with `3`)
- `i++` (`i` is incremented with `1`)
- `j = i / 4` (`j` is assigned the quotient of `i / 4`) (*div*)
- `j = i % 4` (`j` is assigned the remainder of `i / 4`) (*mod*)

Selection

The "if-else" statement is the most important statement when it comes to selection. Eckel (2000) states that the "if-else statement is probably the most basic way to control program flow". Selection involves, during runtime, specific code to be executed or not, depending on a condition formulated in parenthesis ("`(`" and also "`)`"). The basic structure is:

```
if (condition)
    statement1;
else
    statement2;
```

The condition depends on the following symbols: `>` (larger than), `<` (smaller than), `>=` (larger than and equals), `<=` (smaller than and equals), `==` (equals), `!=` (not equals), && (and), `||`(or).
The method, "short-circuiting" as explained in Eckel (2000) is applied in the code and can be seen in the following example. The statement, `if (x<10 && y<10)`, has two conditions, of which the second will not be tested if the first is not true. As in the code, the condition with the least occurrence is put first, contributing into faster and cleaner processing. Refer to relevant literature for information about the "`switch`" statement.

Iteration

This technique repeats the same statement or statement block according to a condition. The three iteration statements are:

- `for (`*`initialisation`*`; `*`condition`*`; `*`modification`*`) `*`statement`*`;`
- `while (`*`condition`*`) `*`statement`*`;`
- `do `*`statement`*` while (`*`condition`*`);`

In the example,

```
for (int i = 0; i < 10; i++) statement;
```

the *`statement`* will be executed 10 times, as the value of `i` will change from 0 up to 9 and the statement is executed each time `i` changes. Refer to relevant literature for information about the "`do while`" and "`while`" statements.

Arrays

Campione (2004) explains that an array "is a structure that holds multiple values of the same type", for example an array of integer values or even an array of sound samples. However when polymorphism is used, which will be discussed later, multiple types can be stored in the same array. An array can be identified by the use of brackets ("`[ ]`") and each array element can be accessed by referring, in any order, to the index of the element. For example, `sample [0]` refers to the first element in the `sample` array, `sample [4]` to the fifth, etc. Multidimensional arrays, also called "arrays of arrays", can be visualised as a structure with rows, columns and even depth. An example is the two-dimensional `node` array, used in SystemTwo, where each row has different number of columns, and can be accessed by using two indexes, for example `node [2][3]` refers to the element stored in row 3, column 4.

Object-oriented Programming (OOP)

As stated by the title of chapter two of Eckel (2000), "Everything is an object". Consequently, it is quite simple to define an object. Campione (2004) explains further that every object does have a state and behaviour, for example the human object's state includes name, age, address, etc. and the behaviour consists of walking, eating, thinking, etc. The interaction between a human object and a bicycle, another object, resulting in a cycling motion and movement can be viewed as object-oriented. However, OOP is based on software objects and involves programs that use classes.

> "Object-oriented programming is just the latest in a long series of solutions that have been proposed to help solve the 'software crisis'. At heart, the software crisis simply means that our imaginations and the tasks we would like to solve with the help of computers, almost always outstrip our abilities." Budd (1997)

For a further discussion of object-oriented programming, the following three sub-sections, *Classes*, *Composition and Inheritance*, and *Polymorphism*, are to be discussed.

Classes

A class is almost the same and also treated the same as a variable type; however, a class usually consists of more than one variable (state) and also does have functionality (behaviour). A class is the outline of objects which have the same functionality, for example the `SampleReader_16V1` class (from SystemTwo) can read and play samples (behaviour), but also keeps track of the number of frames that have been played and at what rate the sample reader is playing (state). The objects used, such as `reader1L` and `reader1R`, are the active objects, and therefore are actual copies of the `SampleReader_16V1` class, also called "instances" of the class.[82]

A class consists of primitive type variables, other objects and methods,[83] with the latter categorized under the behaviour of the class. A method can be identified by the method name followed by parenthesis

(" `()` ") and more frequently these parenthesis will be filled with a "parameter-list", which can include variables and objects. Therefore, the calling of a method can be identified by

```
    object.method (parameter-list)
```

for example

```
    me.riding (myBicycle, for 5 minutes)
```

consists of

```
    object.method (object, variable).
```

Composition and Inheritance

Hubbard (1999) states that a feature of OOP, which makes it very effective, is the reuse of objects through the use of composition and inheritance.

Composition was discussed in the previous subsection and basically amounts to the use of a pre-defined object inside a different object. Composition is often identified by a "HAS A" association, because a specific class incorporates a different class. This technique is very common and is used throughout the three systems accompanying this thesis. As mentioned before, a class name always starts with an uppercase letter, therefore each time a variable is declared and the type name starts with a capital letter, composition occurred.

[82] A class name always starts with an uppercase letter, but an object name starts with a lowercase letter.

[83] Refer to programming literature for information on constructor methods, static methods, void methods, "return" methods (such as boolean methods), overloading, overriding, etc.

Inheritance needs more attention than composition to implement and consists of the creation of a new class, which inherits the state and behaviour from a different class. It is often identified by an "IS A" association, because a specific class forms a subcategory of a different class. Therefore, there exists no need to declare the variables and methods of the *superclass* inside the *subclass* (also called "derived class"). Campione (2004) uses the example of a bicycle as superclass, and identifies three subclasses, mountain-, racing- and tandem bike. The three derived classes share some characteristics, such as the number of gears, which will be declared in the superclass. Additional features to a specific subclass, such as the suspension of a mountain bike, will be only declared in the mountain bike class. Inheritance can be identified by the reserved word `extends`, for example `RacingBike extends Bicycle`.

Inheritance is used at various locations in the three systems, for example in SystemTwo, the `ControlRegion` class, which is the superclass of the `VerticalSlider`, `HorizontalSlider`, `RotaryKnob`, `JogWheel` and `Button1` subclasses. In addition, the `Button1` class is the superclass for the `Button2` derived class.

Polymorphism

This technique is closely associated with inheritance and involves the various responses to a specific method call. The method will be implemented in different object types in a hierarchy of super and subclasses. When using arrays together with polymorphism, different types of objects can be stored in the array, and when a specific method is called, the implementation of that method in the object type will be executed. In SystemTwo a two-dimensional array, `region[][]` is used, which is assigned various types, such as `RotaryKnob`, `JogWheel` and `Button1`, according to the discussion in the previous subsection. These types have different procedures, for example to set the basic value of the region and therefore by calling `region[x][y].setValue()`, independent of the other types in the array, the type stored in `region[x][y]` will force its own `setValue()` implementation.

Threads and "synchronized"

As explained by Campione (2004), programs are usually sequential and "At any given time during the runtime of the program there is a single point of execution". A thread is a sequential program inside another program with its own single point of execution. Multithreading means therefore that a main program can have numerous threads inside the program. To create a thread, a class is to be designed, which is a subclass of the *Thread* class or the class has to implement the *Runnable* interface.

The three systems each have at least one thread located inside the *MidiObject* class. This thread is responsible for receiving MIDI messages from the MIDI port, converting them to understandable values and sending these values to the main program. SystemTwo also has a *TimerThread* to every 0.5 seconds determine the song positions of tracks playing, and a *VUMeterThread*, which updates a VU Meter display every 50 milliseconds.

The *TimerThread* class invokes the *getPosition* method, in which the song position is determined. In the main program, when the track is to be rewound or played fast forward, the *getPosition* method is also invoked to determine which files are to be loaded.[84] It often happens that the latter method is called from different places at more or less the same time, which results in incorrect values for variables used inside the method. The word "synchronized" is added to the method declaration, and the effect is that the method can not be invoked if the previous invocation is not completed. Therefore, classes queue and wait until the method invoked by the previous call has executed. The *setPosition* method and *getPosition* method can also not be executed at the same time, because they share variables, therefore the *setPosition* method's declaration also has the word "synchronized". Therefore, only one of these methods can be executed at a specific time.

Graphics

Java graphics is based on the WIMP (Windows, Icons, Mouse, and Pointer) scheme and an extensive Graphical User Interface (GUI) can be created. However, in the three systems, the traditional method of using a mouse is not employed to control buttons and other controlling interfaces, as these are manipulated by incoming MIDI code. The approach taken involves image files moving on top of each other, the majority representing control regions, and is discussed in chapter five. Java implements graphics in two ways, through *Applets*, a program running in a webpage, and *Frames*, which is a standalone program. The three systems use the frame approach and the inheritance technique is implemented by extending the *Frame* class in the main program classes. Therefore each main class "IS A" Frame and inherits the functionality from the *Frame* class.

Refer to section 6.1.3 for future development incorporating the Java 2D and 3D APIs.

[84] Refer to chapter five for a discussion on the rewinding and fast forwarding of samples.

Exceptions

Eckel (2000) states that "An *exceptional condition* is a problem that prevents the continuation of the method or scope that you're in." The problem is called an *exception* and if exceptions are not treated, they will initiate the termination of a program. A simple example is `z = x / y`, which is accepted, but if y is zero, a "division by zero" exception (traced by the *ArithmeticException* class) will occur. To treat an exception region, firstly the source code must be inside a "*try*" block and secondly a "*catch*" block is to be implemented to deal with the exception.

Exception handling in the three systems includes Exception, SynthException, FileNotFoundException, InterruptedException, ArithmeticException, etc.

This concludes the section on Java as mapping software for the three systems. It is important to note that EyesWeb could also be used to program some of the mapping functionality, but because it is a visual programming language, the programmer does not have the same control as on a text-based language, such as Java. By using Java, it also gives the programmer the opportunity to obtain a higher level of programming skills and technique.

4.4 JSYN

The JSyn API is an important part of SystemTwo, as it forms the audio generator of the system. Therefore JSyn (Java Synthesis), which is a 16-bit real-time audio unit generator based synthesis API, needs to be discussed. JSyn was initially employed to generate sound effects and music for web pages, without the transport or download of large audio files. JSyn, which was and is still developed by Phil Burk, is suitable for teaching sound synthesis, as it includes most basic unit generators and a development environment for subtractive synthesis, as discussed in chapter three. These unit generators, which are represented by classes, can be categorised in the following:

- Oscillators: SineOscillator, PulseOscillator, TableOscillator, etc.
- Noise: WhiteNoise, PinkNoise, RedNoise
- Filters: Filter_LowPass, Filter_HighPass, Filter_PeakingEQ, etc.
- Control: EnvelopePlayer, ExponentialLag, PeakFollower, etc.
- Arithmetic: AddUnit, MultiplyUnit, MultiplyAddUnit, etc.
- Logic: CompareUnit, SchmidtTrigger, MaximumUnit, etc.
- Miscellaneous: SampleReader, SampleWriter, WaveShaper, DelayUnit, SynthMixer, etc.

As SystemOne already focuses on subtractive synthesis and to make the three systems more diverse, SystemTwo concentrates more on the sample facilities of JSyn.

Basic Syntax

Because all modules in JSyn are classes, objects need to be constructed the same way as other objects, for example:

```
SineOscillator myOsc = new SineOscillator ();
Filter_BandPass myFilter = new Filter_BandPass ();
```

All unit generators can be started or stopped, to add or release stress on the CPU, such as:

```
myOsc.start ();
myFilter.stop ();
```

To change parameters of unit generators, the `set()` method is to be used:

```
myOsc.frequency.set (440.0);
myFilter.Q.set (0.5);
```

Each unit has at least one input and one output. Units can be connected with the restriction that a unit's input can have only one unit connected to it, but a unit's output can be connected to several units. This is also called "multiple fan-out". The syntax for connections is for example:

```
myOsc.output.connect (myFilter.input);
myLFO.output.connect (myFilter.Q);
```

As SystemTwo focuses more on the sample facilities of JSyn, it is necessary to explain the basics of the procedure to load samples. Subclasses of the *SampleReader* class can be used and which retains a *samplePort* object, in which a sample is to be loaded, for example:

```
myReader.samplePort.queue (mySample);
```

A fraction of a sample can also be loaded; for example, if the first halve of a sample is to be loaded:

```
    myReader.samplePort.queue (mySample, 0,
                                   mySample.getNumFrames () / 2);
```

so that `0` represents the starting frame and `mySample.getNumFrames () / 2` the number of frames. A samplePort is cleared by using:

```
    myReader.samplePort.clear ();
```

This concludes the discussion of the three major pieces of software, EyesWeb, Java and JSyn. Refer to chapter five for a discussion on problems implementing the three systems in this software.

4.5 JAVAMIDI

JavaMIDI is a library of classes, which runs on Windows and Macintosh platforms in Java programs for MIDI input and output and arbitrary-length *system exclusive* messages. It is to be used for non-commercial projects, but needs the written permission of Robert Marsanyi, the programmer and copyright-holder, when used in commercial projects. As this thesis is not a commercial project, a credit for the programmer should be included. JavaMIDI is selected as part of this thesis's systems, because of its functionality, ease of use and the simple installation procedure, which is accomplished by the copying of a few files.

It is recommended to experiment with the demo's source code, available with the API, and then to modify the code to fit the user's application. For the three systems, the MIDI input abilities are used, and in addition MIDI output is used for SystemOne and SystemThree.

Other network implementations, considered and experimented with during the project, for the real-time transport of data from the EyesWeb computer to the Java computer and other equipment, include *MidiShare* and *JavaSerialComm*. The Java API also incorporates other network options, making use of *TCP/IP* (Transmission Control Protocol / Internet Protocol), *UDP* (User Datagram Protocol), etc., which will be considered in future research.

4.6 JCREATOR™ 3.0 LE

Java programs can be written in any text based editor program, such as *Microsoft Notepad*. These programs need to be compiled, before it can be executed, which can be both accomplished by the use of the Java API and a *DOS prompt* interface. However, the larger and the more integrated the programs become, the more difficult they are to debug, maintain and control. The solution is to start programming by using an IDE (Integrated Development Environment), which behaves like a text-editor and in which programs can be compiled, executed, bundled to form projects, maintaining additional class-libraries, and includes many more advanced functionalities. Such an IDE is *JCreator 3.0 LE* by Xinox Software, which is also used in the three systems.[85] The use of the software is not to be discussed in this thesis, and the reader is referred to the software's own help documents. A more advanced version, *JCreator PRO* can be used, which is not freeware, but the light edition version is adequate for the level of use.

Other IDE's with more or less the same functionality include Modelworks' *JPad Pro*, *NetBeans*, Holt Software's *Ready to Program*,[86] etc. which can be downloaded from their websites.

4.7 JAVA™ 2 SDK, STANDARD EDITION, VERSION 1.4.2 (J2SDK 1.4.2)

As explained in Campione (2004) major versions of Java introduced during the last decade include "Java 1.0" and "Java 1.1". Since "Java 1.2", "Java" is called "Java 2", which was followed by "Java 2 version 1.2", "Java 2 version 1.3" and the latest "Java 2 version 1.4". These versions refer to a library of classes to be used in programs, and are also called the Java API, discussed elsewhere. To create and develop programs, that are using these versions, a SDK (Software Development Kit) is needed. The version used in the three systems is the Java 2 SDK version 1.4.2, which is the latest SDK.

The "Swing" components are part of the Java API since the 1.2 version, and consist of classes to create GUI (Graphical User Interface) programs. The three systems do not make use of many Swing components, but Swing is needed to create the graphical environment, which include the *JLayeredPane* class to perform animation with. Refer to chapter five for an explanation on the graphical implementation.

[85] During the initial stages of programming the three systems, a beta version of Holt Software's *Ready to Program*[86] was used.

[86] Refer to http://www.modelworks.com, http://www.netbeans.org and http://www.holtsoft.com.

CHAPTER FIVE
PROGRAMMING METHODOLOGY

5.1 INTRODUCTION

This chapter discusses problems that were encountered during the process of designing and programming the three systems, that is the practical side of this thesis. Section 5.2 explains an overall MIDI adaptation to the specific hardware and software. The screen output of the Java application is the graphical interface that the user of the project will communicate to. Therefore, section 5.3 discusses the graphical side of the Java applications as a *Graphical User Interface* (GUI). Section 5.4 discusses what lies behind the GUI, explaining an orientation how tasks are executed. Wherever source code is given, it will refer to the Java applications and will be code that is valid Java language used in the project.

5.2 MIDI IMPLEMENTATION

This section serves as a basic overview of how MIDI is applied in this thesis. Refer to numerous available sources about number systems and MIDI implementation. MIDI messages can be categorised in *Channel* and *System messages*.[87] The former can be further categorized in *Channel Voice* and *Channel Mode messages*.[88] The messages that are used in this thesis are based on channel messages as this is the only type supported by the EyesWeb MIDI library. Each message consists of three data bytes:

first byte	*Status byte*
second byte	*Byte1 / Data byte 0*
third byte	*Byte2 / Data byte 1*

The status byte can be divided into two 4-bit commands, of which the first is the type of instruction and the second the channel the message is mend for. The range of the status byte is $8n_{16}$ to En_{16},[89] in which case n is the channel number ($0_{16} – F_{16}$ which represents channel $1_{10} – 16_{10}$). The range of both the two data bytes are 0_{10} to 127_{10} or 0_{16} to $7F_{16}$. With Cn_{16} and Dn_{16} as status byte, the second data byte (Byte2) is ignored.

[87] According to Pressing (1992) a *channel message* is sent to one device channel at a time. A *system message* is sent to control a whole device and consists of *system exclusive-*, *system common-* and *system real-time messages*.

[88] Channel Voice messages includes “Note On”, “Pitch Bend”, etc. Channel Mode messages consist of “Reset All Controllers”, “All Notes Off”, etc.

[89] The use of the subscript to illustrate the base ($_{10}$ for decimal and $_{16}$ for hexadecimal) of the number system is one of numerous ways. For example, $A9_{16}$ is also A9H, A9h, 0xA9, etc.

For example in the MIDI message 94.3C.5A$_{16}$, the "9" indicates a "Note On" message, the "4" is channel 5, "3C" specifies the note to be played which is note 60 (Middle C) and "5A" shows the velocity 90$_{10}$ of the note.

MIDI is used in all three systems by sending messages from the EyesWeb patch to the Java application. Further on MIDI is also used in SystemOne to send messages to the synthesizer and in the case of SystemThree to the Pro Tools MIDI interface.

5.2.1 EyesWeb to Java

As discussed in chapter four, the EyesWeb patch tracks human objects, such as hands, in a specific two-dimensional environment. The Cartesian co-ordinates of the object are sent to the Java applications. The screen of the Java application computer is set to 1024 x 768 (4:3). The resolution of the camera that is used is 360 x 288 (5:4) and therefore, the camera image is to be enlarged to match the Java graphical interface image. Considering the ratio difference, only 360 x 270 (4:3) of the camera image is represented on the interface screen.

The process to transform the state and position of hands to MIDI code is basically the same for all three systems. This process is designed to accommodate SystemTwo, as the system has the most complicated EyesWeb patch setup. Therefore the following is an explanation of this process and based on SystemTwo.

The attempt is to make use of all possibilities of a single MIDI message by sending information such as left- or right hand, *free* or *controlling* movement,[90] and the x and y co-ordinates (according to the camera image) of the hand position. It is not possible to send a value such as 300 with the limitations of the byte size, unless the value is divided by 128, split up in its quotient (*Q*) and remainder (*R*), for example 300 = **2** * 128 + **44** and the **2** and **44** are sent by using two bytes. Constructing the 300 is done by the program receiving the data. With this approach, *Byte1* and *Byte2* are used for the x and y remainders (*xR* and *yR*), respectively. This thesis uses the *status byte* range of 80$_{16}$ - BF$_{16}$ or 128$_{10}$ - 191$_{10}$ and by subtracting 128 from the latter, the decimal value range is 0 to 63. As illustrated in Table 1, the *status byte* can be used to send information about which hand, type of movement and the quotient of x and y (*xQ* and *yQ*).

[90] Discussed in section 3.1.

0 – 31 (*controlling* movement)							
0 – 15 (left hand)				16 – 31 (right hand)			
0 – 3 (xQ = 0) (yQ = 0 to 3)	4 – 7 (xQ = 1) (yQ = 0 to 3)	8 – 11 (xQ = 2) (yQ = 0 to 3)	12 – 15 (xQ = 3) (yQ = 0 to 3)	16 – 19 (xQ = 0) (yQ = 0 to 3)	20 – 23 (xQ = 1) (yQ = 0 to 3)	24 – 27 (xQ = 2) (yQ = 0 to 3)	28 – 31 (xQ = 3) (yQ = 0 to 3)

32 – 63 (*free* movement)							
32 – 47 (left hand)				48 – 63 (right hand)			
32 – 35 (xQ = 0) (yQ = 0 to 3)	36 – 39 (xQ = 1) (yQ = 0 to 3)	40 – 43 (xQ = 2) (yQ = 0 to 3)	44 – 47 (xQ = 3) (yQ = 0 to 3)	48 – 51 (xQ = 0) (yQ = 0 to 3)	52 – 55 (xQ = 1) (yQ = 0 to 3)	56 – 59 (xQ = 2) (yQ = 0 to 3)	60 – 63 (xQ = 3) (yQ = 0 to 3)

Table 1. The representation of the relative status byte possibilities.

It is obvious that the value of *yQ* is the remainder of the *status byte* relative value divided by 4. The range of *xQ* and *yQ* is in the range of 0 – 3 and therefore the maximum co-ordinate value represented by this system for x and y is 3 * 128 + 127 = 511 (*x* = *xQ* * 128 + *xR (Byte1)* and *y* = *yQ* * 128 + *yR (Byte2)*). This maximum resolution is 511 x 511 which is more than 360 x 288. The message received by the Java program is constructed by using *if* statements and basic mathematical operations. An example is A9.48.6B$_{16}$ which represents:

$A9_{16} = 169_{10}$, $169 - 128 = 41$, $yQ = 41 \% 4 = 1$, $xQ = (41 - 32) / 4 = 2$
$48_{16} = 72_{10}$
$6B_{16} = 107_{10}$

therefore *free* movement, left hand, x co-ordinate = 2 * 128 + 72 = 328 and y co-ordinate = 1 * 128 + 107 = 235.

During the study, a previous approach was to use the *status byte* for information only about which hand, type of movement and the co-ordinate of X or Y (and even Z for 3D). The first data byte is used for *Q* and *Byte2* for *R*. With this approach higher values could be sent, but three MIDI messages need to be grouped together to form one X,Y,Z co-ordinate. This aspect is left for future study.

5.2.2 Java to Synthesizer (Nord Lead 2)

The Nord Lead 2 (NL2) sends and receives MIDI messages on all notes, as well as all controllers, except "Master Level". The *Status Byte* for *Note on* and *Note off* is $9n_{16}$, where n is in the range of 0 – 3, representing the 4 channels, because of the 4-part multi-timbrality of the synthesizer. The *Status Byte* for all controllers is Bn_{16} (n = 0 – 3), except *Pitch Bend* that uses En_{16}. In the case of notes, *Byte 1* indicates the specific note and *Byte 2* the velocity of the note (0 with *Note off*). In the case of controllers, the specific controller is indicated with *Byte 1* and *Byte 2* represents the value the controller is set to. Buttons with "on/off" functions (for example "Keyboard Track") have a "0", which indicate the "off" position. Exceptions regarding button controller values and the graphics of the Java application are discussed in section 5.6. The Pitch Bend command's *Byte 1* is always 0 and the value of *Byte 2* falls in the range of 0 – 127 (64 for normal).

5.2.3 Java to Pro Tools

All MIDI instructions transmitted by the CS-10 (see section 3.4.1) and received by Pro Tools make use of the *Control Change* message type. The commands are sent on MIDI channel 16, and therefore the *Status Byte* is BF_{16}. *Byte 1* indicates the specific switch, fader or wheel number and *Byte 2* corresponds with the value of the switch, fader or wheel number.

Switches can be depressed or released, by using $7F_{16}$ for depressing and 00_{16} for releasing. It is important to note that a switch must be released before another switch can be depressed, except in the case where the "Option" switch must be depressed simultaneously with another switch. It is therefore needed to keep track of depressing and releasing MIDI messages.

For faders or knobs, appearing on the Pro Tools screen, to recognize receiving MIDI commands, the incoming messages must first pass through the on-screen position (*null point*). In the initial state when the Java application is running, Pro Tools faders must receive all values from 0 up to 127, to assure that the *null point* was passed through and the Pro Tools faders will follow fader commands.

5.3 JAVA GRAPHICS

There are several ways of graphical manipulation that can be implemented in Java. Some offer painting facilities, such as drawing lines, circles and polygon shapes in different colours. Others offer the loading and using of image files, which is easier to control by the designer of specific images.

5.3.1 Layering images to perform animation

All graphics used in the Java applications are Image files that move on top of each other. A large background image, the size of the application window, is created and placed in the application, and other smaller images move on top of the background. For example, the left and right hand cursors are image files moving on top of the background, in the same way the mouse cursor image moves on the screen. In the case of an "on/off" button the "off" button image is sketched in the background image, but when the "on" button is selected, a new image, representing the "on" button, will be made visible; otherwise it remains invisible.

Each image is assigned a depth layer value (also called Z-order) in the application, which is necessary in the case of images overlapping. As the background image has the lowest layer value, the hand cursors have the highest value, because the cursors must be fully visible and not be hidden behind other images. Therefore, all other images have depth layer values between the background and the cursor level values. Note, that when images with the same layer value are added, the first image added will be on top of other images.

Different methods can be used to load images in Java. The method used in the project is based on the animation implementation by McGrath (2001) and Campione (2004). Each image is loaded as an icon (so called *ImageIcon*) that is connected to a label (*JLabel*). The label is the component, which will move across the screen and can be set visible, or not. A layered pane (*JLayeredPane*) is used to add all the labels and giving each a depth layer value.

The labels can be moved by the method *setLocation (x,y)*, taking the new x and y co-ordinates of the left top position of the image on the screen as arguments. A label can be set visible by using *setVisible (true)* and not visible by using *setVisible (false)*.

5.3.2 Transparency

An image file is represented by a rectangular matrix of values and therefore all images, regardless of shape, are stored as a rectangular image. In the case of a non-rectangle image such as the right-hand cursor (☝), certain regions have to be transparent. In the case of a circle, the four corners have to be transparent, so that whatever is on the background will shine through the corners, making only the circle visible in the circle image as illustrated in Figure 22.

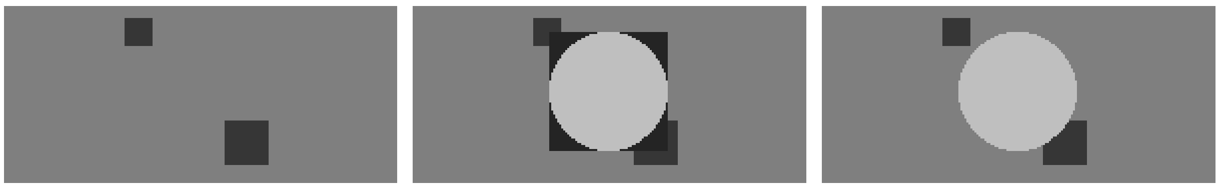


Figure 22. Illustrating transparency with first a background image, a circle image added and then a circle image with transparent regions added instead.

When designing images the unnecessary regions are identified and filled with a specific colour, and all occurrences of that colour in the image are understood as transparent. Purple (RGB: 128, 0, 128) is used for this project as transparency colour.

The Java Abstract Window Toolkit (AWT) allows the programmer to make use of two picture formats: GIF (Graphical Interchange Format) and JPEG (Joint Photographics Experts Group). The JPEG format uses a more successful compression method to represent bigger format images, such as BMP (Windows Bitmap), than the GIF format. All non-rectangle images in the project are designed, created and transparent regions are filled with the specific purple colour in Microsoft Paint and saved as 8-bit 256-color GIF files.

Microsoft Photo Editor 3.0 is used for making unnecessary regions transparent.[91] Photo Editor is able to save files with transparency in three formats: GIF, TIFF (Tagged Image File Format) and PNG. The GIF format was therefore the most suitable format to work with when dealing with transparency and the JPEG format for true rectangle images.

5.4 CONTROL REGIONS

The usage of the three systems focuses on navigating through a virtual environment to trigger functions connected to it. Further study involves emotion and expressive gesture present while navigating, and a time factor is to be included to indicate the velocity with which an object moved from one position to another. However, for this thesis functions are executed as soon as a specific region is entered. A control region (further on called region) can be a representative of any physical controlling object, such as a button, rotary knob or slider. The region is a rectangular shape, with the left top x and y (x1 and y1) and the right bottom x and y (x2 and y2) co-ordinates stored.

The following statement is used, to detect if a hand position (x, y) enters a region.

```
if (x >= x1 && x <= x2 && y >= y1 && y <= y2)
    do function;
```

A superclass (ControlRegion) is programmed, in which the x1, y1, x2, y2 co-ordinates are stored. The value of the specific region is also stored, which can vary from 0 – 1 in the case of an “on/off” button and 0 – 127 in the case of a knob. From the superclass, several subclasses are derived to fulfil the need of the specific applications.[92] These subclasses are knobs (*Knob*) and buttons (*Button*) in SystemOne, but also vertical (*VerticalSlider*) and horizontal sliders (*HorizontalSlider*) in SystemTwo. Instances of these classes are stored in an array, of the superclass type, with the size corresponding to the number of regions. As a result the array consists of a group of buttons, knobs, sliders, etc. with tasks and graphical responsibility connected to it. There are exceptions, because some regions function in different ways. For example, the keyboard and parameter controllers of SystemOne have their own classes and are not derived from any other superclass.

If the hand position is not inside a region, for every *controlling*[90] movement, the array is examined if a region is entered. If so, the necessary functions are executed. From the next movement onwards, the current region will be examined if the hand position is still in the region, because it is futile to examine the whole array. When the hand position is not in the current region anymore, the array will be inspected until a next region is entered.

When a region is entered, the result can be to:

- trigger a specific function.
  A normal button is an example. It is important that while the hand position is inside a region, the corresponding region should not be executed more than once. For example a “play” button should only trigger the play functions once.
- trigger a specific function and also change value
  A rotary knob is an example. The rotary knob differs from the button, so that while the hand position is inside the region the value will keep on increasing or decreasing depending on the clockwise or counter-clockwise direction of the hand movement. The value of a region can change as long as it remains inside the minimum to maximum range.

[91] Microsoft FrontPage also provides a saving facility of transparency files.
[92] By using Inheritance as discussed in section 4.3.2.

- move from position.

  The region, of which sliders are an example, moves in the direction the hand position is heading, depending on if it is a vertical or horizontal slider. The region's value will also change, if it is in the minimum to maximum range. The keyboard and parameter control wheels of SystemOne are also examples, so that they can move to comfortable position for the user.
- trigger a specific function, but also trigger another function when exited.

  Some regions do have a specific task, while entered, but need to be normalised when exited. A function such as a "rewind" must be cancelled, so that playing a sample can proceed, if the button is released.

During the development of this project, a MIDI controller was sometimes not connected to the Java applications. Instead, the Java *MouseMotionListener* interface[36] was used to examine the effect and reliability of control regions. The interface allows Java applications to control mouse moving and dragging events. It consist of two methods, *mouseMoved* and *mouseDragged*, which were used for left and right hand movements respectively.

Rotary Knob

When applying a rotary knob, it is necessary to have a basic knowledge concerning trigonometry and how this can be implemented by using methods from the *Math* class which is provided with the Java library. Rotary knobs can be seen in various software programs and the functionality and appearances differ. Some programs make use of a rotary knob where the knob image rotates,[93] but in the case of the thesis an image, representing a light or a finger groove on a jog wheel, moves around concentrically with the circumference of the knob circle. Determining the x and y co-ordinates of the image position relies on a specific value (*t*), which represents the angle in radians relative to the centre of the circle. The following algorithm is used where *xPos* and *yPos* are the co-ordinates to be placed image, *xCenter* and *yCenter* the co-ordinates of the center of the circle and -pi < *t* ≤ pi ($-\pi < t \leq \pi$):

```
int xPos = xCenter + (int) (Math.cos (t) * radius);
int yPos = yCenter + (int) (Math.sin (t) * radius);
```

[93] The rotation of images can be done by using the Java 2D API available from the official Java website.

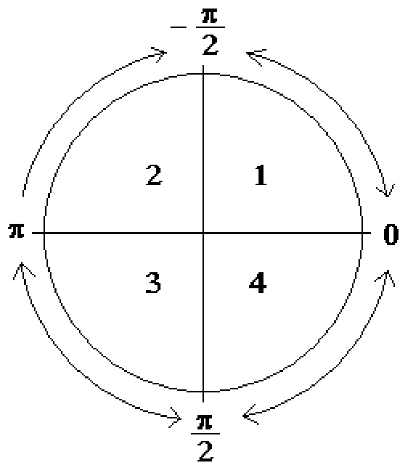


Figure 23. Illustration of the range of *t* ($-\pi < t \leq \pi$) and quadrant numbers (1 to 4).

It is also necessary to determine the value of t by the implementation of x and y co-ordinates. This is done by using:

```
double t = Math.atan2 (yPos – yCenter, xPos - xCenter);
```

The difference (*diffT*) between the previous and current *t* values (*prevT* and *currT*) corresponds to the direction in which a movement is made. For example, it is obvious according to Figure 23 that a decreasing *t* value results in a counter-clockwise movement. An exception occurs when moving from the 2nd to the 3rd quadrant and also from the 3rd to the 2nd when moving clockwise. To deal with this exception the following statement also needs to be implemented:

```
double d = Math.PI / 2;
if (prevT < -d  &&  currT > d)
    diffT = currT – (Math.PI * 2) - prevT;
else if (prevT > d && currT < -d)
    diffT = currT + (Math.PI * 2) - prevT;
```

The value of *t* can further be modified to represent the parameter value of the knob, for example 0 to 127.

5.5 ONE CLASS – MANY INSTANCES

As briefly discussed in section 4.3.2, in OOP a class is designed and programmed and thereafter instances, which are active copies of the class, are created and used. This process can be strengthened by explaining the implementation of the moving objects, two hands, as used in SystemOne and SystemTwo.

It is necessary to store information about each moving object, for example object co-ordinates, the state derived from whether it is inside a control region, and if so which region, the type of movement (*free* or *controlling*), the cursor image files used (for example left or right hand), indication image files (for example outlining the handling control region), etc. Methods to deal with *controlling* movement also need to be implemented.

The procedure is to identify and program all these above mentioned variables and methods into one class. In the thesis this class is called *MovingObject*. Thereafter two instances (one for each hand) are created and constructed with the necessary default parameters. These instances can then be used separately and independently from each other. For further usage numerous moving objects can be created, however, for future development coincidence of the instances probably needs to be investigated.

5.6 SYSTEMONE

This section discusses programming challenges posed by SystemOne. It consists of Java features which are always useful for other programming tasks.

5.6.1 Storing a sound

Each time that a basic program is executed, the program starts with the same basic configuration. During the execution of the program information is added and changes are made. In order for changes to be stored to override the basic configuration when a program again is executed, a separate file on the hard drive has to be used. The result is that the user can proceed where he/she previously stopped the program.

When SystemOne is quitted, a sound's parameters are stored orderly in a simple text file and when the program is executed again these parameters are read from this file. This procedure can be identified in the source code by the use of the *FileWriter* and *FileReader* classes. The configuration is set up by modifying the graphical interface and the internal storage variable while the necessary MIDI code is sent to the synthesizer. Changing the program to store numerous sounds' parameters is straightforward, but this feature was left for future development.

5.6.2 Graphics and MIDI Exceptions

Most selection parameters on the NL2 are modified by a single button connected to a specific parameter. By pressing the button a specific option is selected and indicated by a LED. Different functions of these buttons include typical "on"/"off", "off " and more than one "on" functions and also buttons without an "off" option but always has an option selected. Some parameters use two LEDs which provide more options.

Designing SystemTwo involved the handling of all these diverse functionalities. It was decided to have only one LED image visible at a time which results in the graphical simulation differing slightly from the interface of the NL2. A *Button1* class is designed to deal with button functionality. However, additional methods and array configurations deal with the numerous exceptions present. The construction of MIDI messages combined with the graphical interface leads to more exceptions. For example buttons with "off" options represent a *Byte2* value of 0 (zero) and no LED is visible, but some other buttons with the same *Byte2* value indicates an active option with a present LED. Some buttons differ in the case where the order of *Byte2* values do not correspond to the order options that can be selected. Furthermore, in the case of the LFO2 destination and Arpeggiator direction, these options are interleaved by using the same two buttons and with only one *Byte1* value. The reader is referred to Appendix A for an illustration of these exceptions.

5.6.3 Vector vs. Array

The two-octave keyboard region is designed so that notes can be played by depressing and releasing notes, while the correct *Note on* and *Note off* MIDI messages are sent to the NL2. Because SystemOne makes use of tracking two hands' movements, only two notes at a time can be played by moving in and out of note regions. However, a *Latch* function is added so that notes are kept playing and switched off only when the note regions are exited and entered again. The NL2 has a 16-note polyphony, but 8-note when *Unison* is selected. It was decided that the maximum number of notes to be latched would be set to eight.

The implementation of a note latching environment revolves around three points. Firstly, any number of notes, not more than eight, can be latched, secondly notes can be played and released in any order and thirdly, when the number of notes exceed the maximum, the first played note is released. This unexpectedly results in more complex storing criteria. Information about the latched notes, such as which notes and when the note was depressed according to other latched notes, can be stored in an array. A more efficient and simpler method is the use of the *Vector* class in such a way that elements are connected to form a vector which incorporates the functionality of a linked list.[94] The Java API documentation (2003) states that the *Vector* class "implements a growable array of objects" and that "the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created". Therefore, as a note is depressed it is added as an element to the "right" side of the vector by using *addElement (note)*. If the maximum number of notes (eight) is exceeded, the note element to the extreme "left", is removed from the vector by using *removeElementAt (0)* and the necessary graphics and MIDI code is generated. If a note is released, it is removed from the vector by using *removeElement (note)* and the adjacent latched note elements are connected. Refer to Figure 24 for an example.

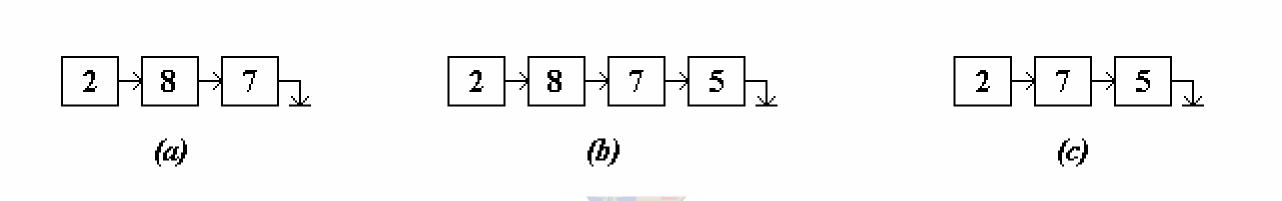


Figure 24. Illustration of a basic usage of the *Vector* class. In *(a)* three notes are already latched. In *(b)* note 5 is depressed and added to the vector. In *(c)* note 8 is released and removed from the vector.

If an array is used, the automatic tasks of the *Vector* class, such as adding, removing and continuously shifting of note elements, have to be done manually. Finally, by deselecting the *Latch* mode, all note elements are removed from the vector, the graphics are updated and the necessary *Note off* MIDI messages are sent.

5.7 SYSTEMTWO

As the following sub-sections focuses on JSyn problems, the solutions to these problems are partially based on the discussions followed on related problems as discussed in the JSyn newsgroup. The author of this thesis experimented with and modified newsgroup response to suit this system. The sub-section captions, derived from newsgroup terminology, are basically sorted in accordance with the chronology of their appearance during the programming process.

To clarify terms used, the following are to be kept in mind. A “track” refers to a song or piece of music which can be played. The “original sample” is a stereo data file representing a full track. A “sample” is usually a short mono data file and can be a small part extracted from the original sample. A “frame” is one value representing amplitude on a specific time. In the case of a stereo sample, each frame has two values, representing the left and right values. With a frame rate / sampling rate of 44.1 kHz a sample consists of 44 100 frames per second.

5.7.1 Hiccups

The main purpose of the DJ is to play tracks one after the other, but also to start playing the next track whilst the current is still playing. In the case of this system, two tracks will be loaded at all times in the memory, except when the program is started and the first track is loaded. The loading of the new track must not have any influence over the current track playing, but should replace the previous track. The average length of tracks is usually about 6 minutes. The tracks are 16-bit, 44.1 kHz sampling rate, stereo samples and with about 5 MB per minute per channel, a 6 minute track occupies approximately 5 MB x 2 channels x 6 minutes = 60 MB. Two tracks simultaneously loaded in memory will occupy 120 MB. Because of the genre of music used, some tracks can be about 10 minutes long. The priority of the current track playing decreases when such a large stereo sample is loaded and totally unacceptable audible “hiccups” occur. When small samples with about 500 KB are loaded, the current track’s priority stays the same. A solution for the hiccups is to split the original sample into many small samples, before the program is started. This is the first reason why an additional program *SamplePreparator* has been programmed to split the original sample into small samples with a numbering system so that the different files are still in the correct order.

5.7.2 Out of memory

The Java heap size (also called *free store*) has an initial size of 64 MB, meaning that 64 MB is reserved in memory to be used by any Java application. Samples that are loaded into the application must be smaller than 64 MB, otherwise an “out of memory” error will appear. Together with the hiccups discussed in the previous section, files must be small enough not to give hiccups, but also may not be larger than 64 MB. The initial and maximum Java heap size can be enlarged using the DOS command *java –Xms[size] –Xmx[size] Class*, so that large samples can be loaded. This forms a second reason why the SamplePreparator program was programmed to deal with large samples while the heap size is enlarged.

---

[94] A linked list is a series of entries where each entry consists of a data object and pointer data about the subsequent entry and /or the preceding entry.

5.7.3 SampleReader_16V1

It is very important that the device used by a DJ must be able to shift the speed of the track playing. Once a sample is loaded in JSyn, there are several ways, which have been implemented and tested intensively, in which the sample can be read and played by classes that can change playing speed.

The *TableOscillator* class can be categorised under Wavetable Synthesis[95] and can be used for small samples, because it plays the same sample repeatedly. The *TableOscillator* changes speed by modifying its frequency parameter and can also reverse a sample when a negative frequency is received. A reason why this class is not the best choice to be used, is because it was originally supposed to function like an oscillator and for SystemTwo the sample only needed to be played once. Another reason is that the output is not stable when the frequency is slightly changed, because a level of accuracy falls short when dealing with small decimal values. For example, if a sample is 5 minutes (300 seconds) in length, the frequency of the oscillator must be 0.0033333. Making the sample playing 1% faster, the sample will take ±297 seconds to play, with a frequency setting of 0.003367. The *TableOscillator* does not accommodate these small changes in a stable enough way.

The *Waveshaper* class can also be categorised under Wavetable Synthesis,[95] but different from the *TableOscillator* class, values can be obtained randomly from the lookup table. The sample is loaded into the WaveShaper and the table is to be looked up, which frame must be played. The WaveShaper must receive -1 for the first frame through to a +1 for the last. Therefore, if a mono sample is 5 minutes in length, with a sampling rate of 44.1 kHz, it will consist of 13,230,000 frames, dividing the range between -1 and +1 into 13,230,000 values will result in too small values to accommodate pitch shifting.

[95] Sams (1999) describes this "sound generation method" as waveforms which "are generated according to parameters in a 'lookup table' stored in memory".

Three classes are derived from the *SampleReader* class, which are *SampleReader_16F1* (for 16-bit fixed rate mono samples), *SampleReader_16F2* (for 16-bit fixed rate stereo samples) and *SampleReader_16V1* (for 16-bit variable rate mono samples). The fixed rate classes are best used when the speed of a sample needs not to be changed. However, the *SampleReader_16V1* will be the most suitable for this application. The default rate is 44100 and will play the sample in its original speed, if the sample is recorded at a sampling rate of 44.1 kHz. The rate can be instantly changed to any value in the range of minimum of 0 and a maximum of 88200. The sample will play an octave higher and twice faster if the rate is 88200, and an octave lower and halve the speed if set to 22050. Unfortunately the sample must be a mono sample, which introduces the third reason why the SamplePreparator program has been programmed to split a stereo sample (two interleaved channels) into two mono samples (mono channels).

5.7.4 The SamplePreparator class

The SamplePreparator is an additional program written to be used before the main program of SystemTwo. As the name states, it prepares a sample before the main program can use the sample. The program splits any 16-bit stereo (44.1 kHz) WAV sample into mono samples of 4 seconds (176400 frames) long each.

Original samples can be selected through a dialog window and be added to or removed from a list which contains all the files that will be split up. After the selection phase, the split phase can be started, which will first create a folder inside the "Sample" folder with the same name as the original sample. The short files will be named Lx.wav for all left and Rx.wav for all right channel samples, where x has the values 000, 001, 002, etc. Figure 25 illustrates how the original stereo sample's frames are stored and then split into mono files.

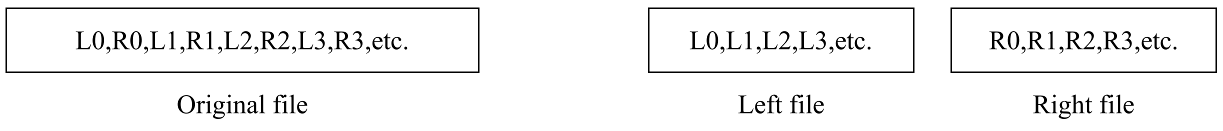


Figure 25. Representation of the frame sequence in a stereo file and mono files.

The program first loads the original stereo sample's frames into a large array the size of the number of frames. The first 352 800 (176 400 x 2) frames are then to be copied to two arrays of 176 400 size each. The two arrays will be saved as two files. This is followed by the next 352 800 frames of the original sample, copied to two arrays and saved as two files. The cycle will proceed until the stereo sample's frames are all saved to new files. The probability that the number of frames in the stereo samples are divisible by 352 800 is small, therefore the last two files saved will not necessarily consist of 176 400 frames. For example, a stereo file of 882 000 frames (10 seconds) is saved to smaller files as illustrated in Figure 26.

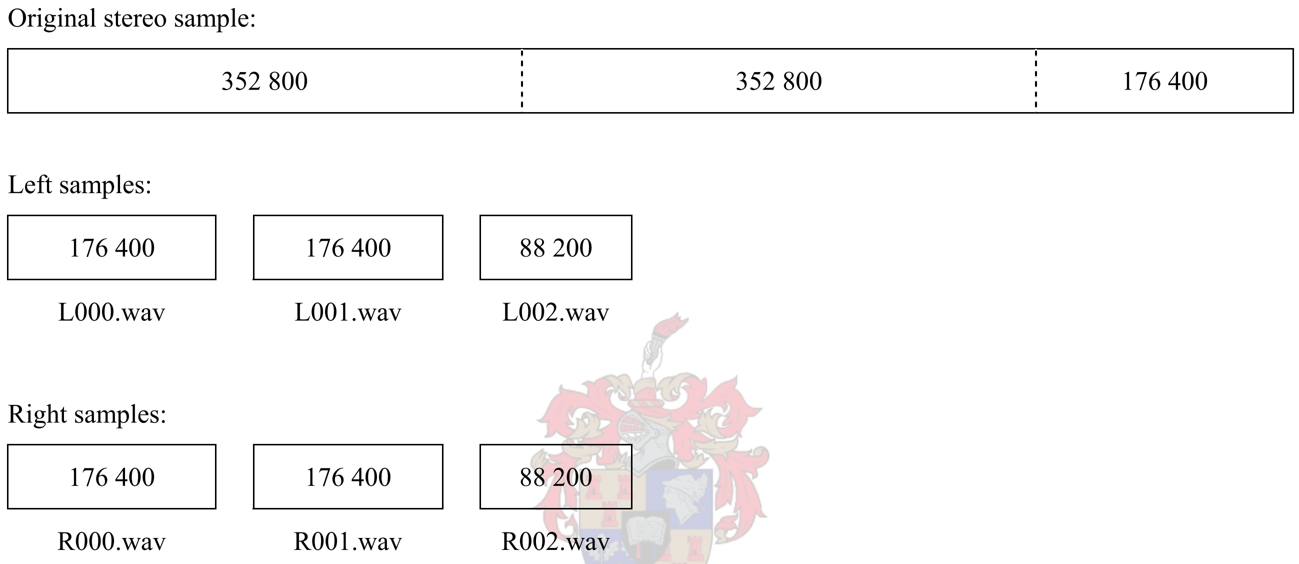


Figure 26. A stereo file is split to smaller mono files.

5.7.5 Reversing samples

For the DJ to perform an accurate beat match, it is necessary to move forward and backward through the track as often and quickly as possible. As discussed in section 5.7.3, the SampleReader_16V1 can not reverse a sample. An additional approach had to be followed, by storing also a reversed copy of the sample in memory when the sample is loaded. The reversed sample can then be played when needed. Figure 27 illustrates how the left channel of the example in section 5.7.4 is stored in memory.

0 – 176 399	0 - 176 399	0 -88 199
(L000)	*(L001)*	*(L002)*

176 399 – 0	176 399 – 0	88 199 – 0
(L000 reversed)	*(L001 reversed)*	*(L002 reversed)*

Figure 27. Representation of storing sequence of normal and reversed sample.

When this example is played in the order *L002 reversed, L001 reversed, L000 reversed*, it will be the exact reversed copy of *L000, L001, L002*. The process, to turn around the frames in a sample, is as follows. When *L000.wav* is loaded in memory, the frames 0 – 176 399 are also extracted and stored in an array of 176 400 size. The first and last frames are exchanged, followed by the second and second last frames, and so forth until the middle of the array. A basic algorithm to reverse an array is used:

```
int t = array.length – 1;
for (int i = 0; i < array.length / 2; i++)
{
    short temp = array [i];
    array [i] = array [t – i];
    array [t – i] = temp;
}
```

The exact processes to change from forward to reverse and reverse to forward are tedious to discuss in detail and do not appear in this document. However, the basics will be briefly explained. When a track is ready to be played, all its samples are queued in the sample reader's samplePort and played. When the review function is called, the position of the track is calculated, the samplePort is cleared, and the reverse samples are queued and played. The same procedure will follow when cue or play is called: calculate the track position, clear the samplePort, queue the original samples.

Obtaining the track position could be easier, if the sample reader was implemented with more methods. The only method, according the movement of frames playing, is the *getNumFramesMoved* method, which returns the number of frames played since samples were queued. The track position is increased or decreased depending on the direction of the track. A method *getPosition* is to be called whenever the direction is changed. The difference of frames since the previous *getPosition* call and the current is added to or subtracted from the track position.

Once the track position is calculated, the current sample number and frame number of the current sample is necessary to queue the correct frames and samples of the compliment direction. Based on Figure 28, the *mod* (/) and *div* (%) operators are used as follows, if the current track position is for example 200 000:

sample number = 200 000 / 176 400 = 1

frame number = 200 000 % 176 400 = 23 600

0 – 176 399	176 400 – 352 799	352 800 – 440 999
(0 – 176 399)	(0 – 176 399)	(0 – 88 199)
(L000)	*(L001)*	*(L002)*

Figure 28. Representation of channel frame number to sample frame number.

Frame number 23 600 in sample L001 is the same as frame number 152 799 in the second to last reversed sample. Therefore frames 152 800 – 176 399 of the second to last reversed sample must be queued, followed by the last reversed sample. The same process will occur when the direction is changed from reverse to forward. The source code gets more complicated when the track position is in the beginning or end of a sample and when many sample are used, but is based on the fundamentals discussed above.

5.7.6 VU Meter

It is important for a DJ to use equipment that has a VU ("Volume Unit") meter for balancing the level between tracks when transitions occur. The JSyn API documentation recommends the *PeakFollower* class for the use of "color organs, vocoders and VU meters". As the class name indicates, it is a unit that follows the peak or absolute value of the amplitude of an input signal. The input parameter *halfLife* setting indicates the number of seconds that the peak takes to drop exponentially to zero. The moment the input signal is higher than the current output signal, the peak follows the input signal. If the input signal suddenly drops to zero, it will take the peak follower 0.5 seconds to drop to zero, if the halfLife is set to 0.5. Figure 29 illustrates an example of an input audio signal with the output signal on top.

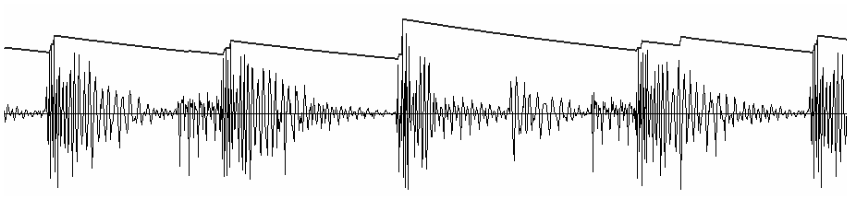


Figure 29. Graphical representation of the input (audio) and output (contour) signals by using the PeakFollower.

A thread is used to get the output of the peak follower every 50 milliseconds. The output value is converted to dB and compared to an array of dB values that often appear on the VU meter of DJ mixers (i.e. -30, -20, -10, -7, -4, -2, 0, +2, +4, +7). These 9 values (18 in the case of stereo) each correspond to an "on LED" image file, which will be set visible or not, according to the comparison of the peak value and array position.

5.7.7 Track position display

It is necessary to know the track position of how much time had elapsed or how much remains. This feature, as on all CD players, is a display in minutes and seconds with an "Elapsed" or "Remain" indication. It uses the same method *getPosition* as in the previous section. To call this method, a thread *TimerThread* is used to determine the track position every 500 milliseconds.

When a program incorporates threads, it can be fatal when a method, such as *getPosition*, is called from different places. It can happen that *getPosition* is called by *TimerThread*, but while executing it is called by the rewind function. Global variables used inside the *getPosition* method are modified by two invocations of the same method, resulting in wrong values in these variables. A solution to this problem is to synchronise *getPosition* with the callers, by adding *synchronized* to the header of the method. From here the method can not be called while executing. The different callers' calls will be queued and have to wait until the synchronized method is available. See the "Threads and 'synchronized'" heading in section 4.3.2.

The application runs with a sampling rate of 44.1 kHz and with the use of the track position, the time can be calculated by dividing the track position by 44100. This will result in the number of seconds, which can be translated into minutes. In the case of calculating the time remaining, the track's overall number of frames is stored from which the track position can be subtracted.

5.7.8 Control regions: changing speed and track position

All the sliders and rotary knobs in this system have 101 increments, so that 0 and 100 represent the least and most values respectively, and 50 represents the normal value. Exceptions are the Cue volume rotary knob and filter sliders, which do not have a normal position at 50.

With the use of the *pitch bend slider*, the rate of the sample reader can be set to –8% to +8% (40 572 to 47 628) of the original rate (44 100). Therefore, a track with an original speed of 132 BPM can be changed to a minimum of 121.44 BPM and a maximum of 142.56 BPM. The rate, set by the pitch bend slider, can further be temporarily speeded up (+4%) or slowed down (-4%) when moved into the *pitch bend button* regions, but disable the button's speed change when moved out of the button region.

The jog wheel is an important feature on many DJ CD-players to move forth and back in a track. In this system, the jog wheel is based on the RotaryKnob discussed in section 5.4, with a minimum and maximum position. When the track is playing, the wheel will change the rate of the sample reader in the range of 0 – 88 200, with the middle position as the rate set by the pitch bend slider. When moving out of the jog wheel, it will reset to the middle position.

If the track is paused, the jog wheel functionality differs, so that the middle position is set to rate zero. Moving to the right of the middle position will increase the rate and play the track forward, while moving to the left of the middle position will also increase the rate, but reverse the track. The furthest left and right position will skip the track position with 10 second intervals, backward or forward.

An additional jog wheel feature is implemented and simulates basic scratching of a track. The track can play forward or backward, according to the speed set by the pitch slider, or stop. A forward, backward movement and stop can be identified by making clockwise, counter-clockwise or no movement respectively inside the jog wheel control region. This is accomplished by the use of MIDI *timestamp*[96] data. The movement direction is determined by the difference between the current and previous position of the hand relative to the difference of the current and previous timestamp. This implementation tends towards expressive gesture (see section 2.4) and is to be considered for future development.

[96] According to the JavaMidi documentation, the timestamp represents the time that the message was received by the Midi port since the Midi port was opened. It is also mentioned that it "should be accurate to within one or two milliseconds".

5.7.9 Filter structure

DJ mixers provide the user with equalising facilities to create filter effects. These are to be used to balance a track's frequency content with another track but also to apply creativity to DJ practice. For example a *break*[97] can be simulated by taking away bass frequencies. Usually each channel is equipped with high, mid and low equaliser knobs. Numerous professional mixer brands are available and their frequency and decibel ranges differ.[98] The basic filter settings of SystemTwo are based on the Vestax PMC-55[98] although the shelf slope and other functionality of the filters were not researched.

In JSyn three classes *Filter_HighShelf*, *Filter_PeakingEQ* and *Filter_LowShelf* are implemented in this order simulating high, mid and low equalising. For the creation of filter frequency sweeps the *Filter_LowPass* class is implemented before the Filter_HighShelf. As these classes are designed for mono signals each stereo signal needs two instances of each class. Refer to Figure 30 for an illustration of the implementation of filters.

5.7.10 The SynthContext class

From an audience's perspective, a DJ's performance needs to be listened to through stereo speakers, the mixer also has the routing facilities to send the *beat matching* process (as discussed in section 3.3.3) to only the DJ's headphones. From the hardware side, this situation created a problem because the SIENA sound card used (see section 3.5) does not support interleaved audio devices when using JSyn. The sound card has eight output channels and by using JSyn these channels are recognized as four stereo devices (channels 1&2, 3&4, 5&6 and 7&8). When a sound card supports interleaved audio devices, eight channels can be grouped together as for example two quadraphonic devices (channels 1&2&3&4 and 5&6&7&8). This latter example would be the ideal setup for SystemTwo.

[97] The character of a *break* consists of an absence of a kick drum, resulting in a short rest period for dancers. The *break* is followed by an increase in tension in the music and leads to the *climax* where the kick drum is added.

[98] The frequency and decibel ranges on favorite models include:

Model	*High*	*Mid*	*Low*
Behringer DJX700	+12 / -32 dB at 10 kHz,	+12 / -32 dB at 1.2 kHz,	+12 / -32 at 50 Hz
Pioneer DJM-300S	+12 / -26 dB at 10 kHz,	+12 / -26 dB at 1 kHz,	+12 / -26dB at 100 Hz
Vestax PMC-55	+6 / -24 dB at 8 kHz,	+6 / -24 dB at 5 kHz,	+6 / -24 dB at 80 Hz

From the software side, registered JSyn users have access to ASIO[99] support which is enhanced provision for multi-channel functionalities in JSyn. The author of this thesis registered,[100] obtained the ASIO related JSyn software and experimented with this software. The result was access to only one stereo device (channels 1&2), which prompted the author to correspondent with the JSyn newsgroup. The response was that the problem probably lay in the fact that the sound card does not support interleaved audio devices and that the ASIO functionality of the newest sound card driver does not correspond with the JSyn ASIO support. It was decided to leave this issue for future experimentation and investigation.

The outcome was to make use of the *SynthContext* class which is part of the JSyn API, a decision that was discussed on the newsgroup as a possible solution to the creation of a multichannel environment. It involves mounting an entire JSyn stereo circuitry in a context which is then connected to a specific stereo device. With the eight output channels of the SIENA sound card four *SynthContext* instances can be created. For the implementation of this class, for example by moving a mono signal to the four speakers in a quadraphonic setup, two contexts are needed. The sound source is created in both contexts and by modifying stereo panning and the amplitude of these sound sources the four channel environment is simulated.

Because of the use in SystemTwo of large sound files, the main stereo performance with equalising facilities is mounted in one context. A second context incorporates the *cueing* facility (see section 3.3.2) which consists of the signals going to the DJ's headphone. The circuitry of the second context is a duplication of only the necessary components of the first context and includes only the right channels of both sample readers and no equalising facilities. Refer to Figure 30 for a circuitry illustration of both contexts. The functionality of the sample readers includes the precise synchronisation of main and cueing performance.

[99] ASIO (Audio Stream Input/Output) is developed by Steinberg. It is a multi-channel audio transfer protocol and supported by numerous manufacturers of audio and MIDI software applications. It allows software to access the multi-channel capabilities of numerous professional sound cards.
[100] The author is registered as a "Registered Developer". Refer to the "JSyn" section in Appendix B.

5.7.11 Cue structure

As discussed in section 3.3.2, *cueing* forms an important part of a DJ's operation. As the DJ is the only person listening to the cueing signals on his/her headphones while the much louder main performance is listened to by the audience, it is necessary to have a mixer with suitable cueing facilities. SystemTwo implements cueing features which are present on most professional mixers. Refer to Figure 30 for an illustration of the cueing circuitry which is the main feature of the second context as discussed in section 5.7.10.

The features implemented in SystemTwo consist of a switch button for each sample reader to send the signal to the headphones or not. It has a "Cue Volume" rotary knob for controlling global *cueing* level and a "Cue Balance" rotary knob mixing between the two sample readers. A "VU Mode" button is implemented, which permits toggling between displaying the main stereo performance or the *cueing* output on the VU meter. Finally, it also has a very useful "Cue Split" button which toggles between sending reader 1's output to the left and reader 2's output to the right of the VU meter or sending these outputs as a summed output to the VU meter. Further, it is also important to know that the "Gain" faders have an influence on the *cueing* readers.

5.7.12 Sample chooser

A control region is added at the bottom of the graphical interface where all available samples can be seen and loaded into memory. The region shows five samples's titles at a time and with buttons, the following or previous five titles are displayed. By using the correct buttons a sample can be selected and loaded into the desired sample player.

By using `String [] playList = new File ("Samples").list ();` the names of all files and folders in the direct "Samples" folder are stored in the "playList" array. It was found that the order in which these names are stored in the array is dependant on the time and date these files and folders were put in the "Samples" folder. As navigating through a list of entries generally relies on an alphabetical order of entries it is necessary to sort the list before using it. Although numerous sorting methods are used in computer programming, the *bubble sort*[101] method was chosen and implemented. The loaded samples are shown and a specific sample can be loaded in both readers.

[101] Koffman and Maxim (1999) states that the bubble sort method "compares adjacent array elements and exchanges their values if they are out of order".

5.7.13 Beats per minute

Although most club DJs do not use the Beats per minute (BPM) function implemented on mixers and CD players, it is quite useful for novice DJs but also when dealing with new functionalities available on CD players, such as sample looping. The way it is implemented in SystemTwo is very basic in that it does not automatically detect BPM on the playing sample. The value is just calculated according to the initial BPM provided by the user and updated each time the pitch bend slider is adjusted. The name of the sample folder, created by the SamplePreparator program, can be renamed by adding the BPM after an underscore, for example “sample_132.05”. In the absence of an initial BPM value, the percentage deposition of the pitch bend slider is displayed. Refer to chapter six for future developments incorporating automatic BPM detection.

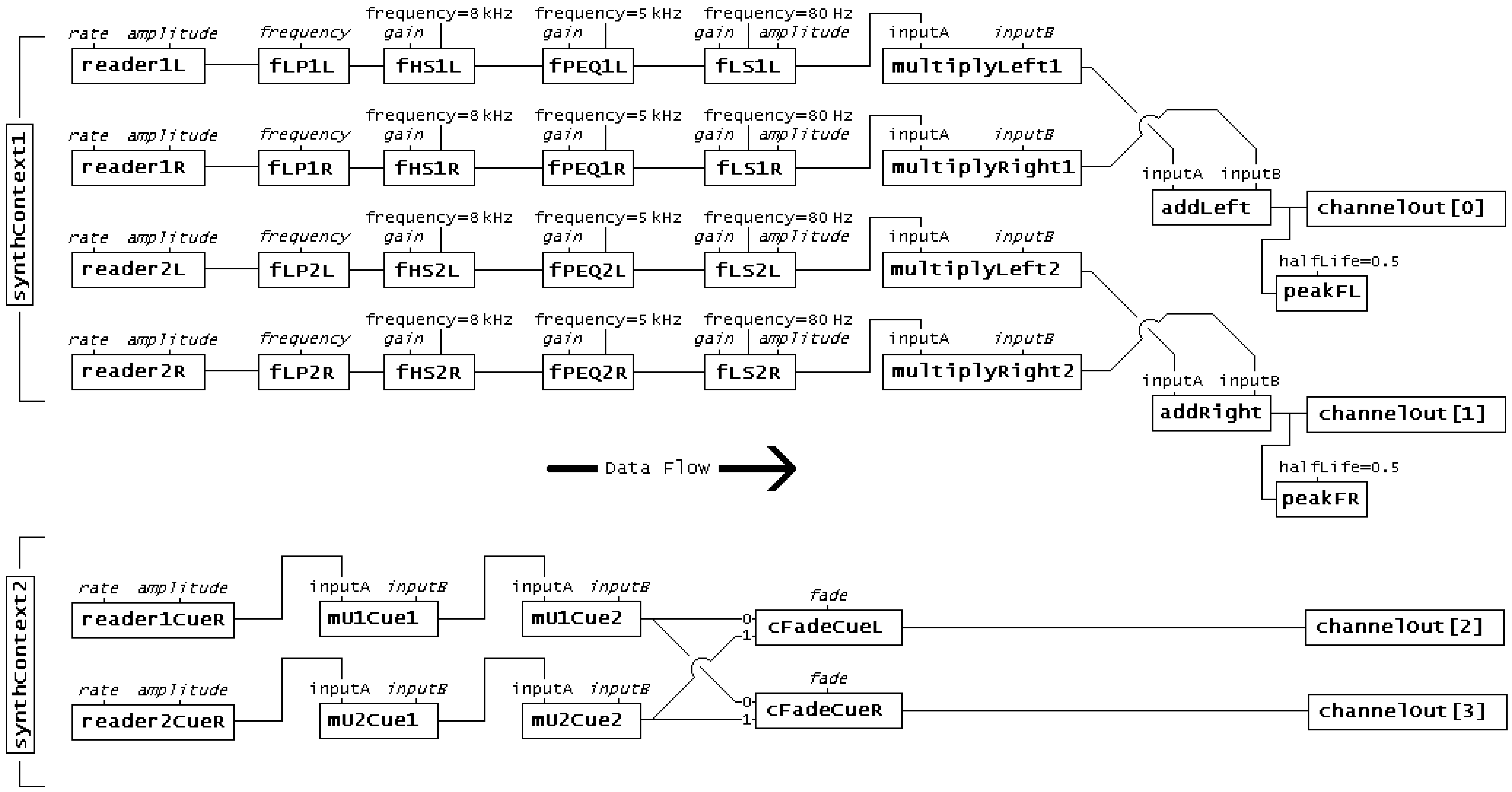


Figure 30. The Jsyn circuit structure of SystemTwo. Instances are bordered and inputs that change during execution are printed italic.

5.8 SYSTEMTHREE

As mentioned in section 3.4.1, this system was especially designed to accommodate specific requests categorised into two approaches. The reader is referred to section 3.4.2 for a discussion of the latter. For future development the system can easily be expanded, as the fundamental operation and limitations of the hard- and software have already been implemented and dealt with. This system manipulates the focused and target track, which could be seen as the more advanced functionalities of the JL Cooper CS-10 to be implemented. This section explains some characteristics and restrictions identified while programming this system. It is unnecessary to discuss Pro Tools terminology that appears in this section, because these terms are only mentioned in relation to the design flow of SystemThree. The reader is referred to Digidesign (2001a) for more information.

5.8.1 Pro Tools session setup (Surround and busses)

The first approach entails the control of Pro Tools so that it functions as a quadraphonic surround panning system. The default method to deal with surround sound includes the use of the *Surround Mixer* plug-in. This plug-in incorporates an *Output Window* for surround panning which is basically similar to the graphical interface of SystemThree as illustrated in Figure 14. However, the Output Window can not be controlled by any MIDI code received from the JL Cooper CS-10. This setback leads to the use of the fact that the audio of each Pro Tools audio track or *Auxiliary Input* track can be sent to a maximum of five *Sends* (excluding the standard *Track Output*). In the case of this system, audio is sent to four Sends (Output 1, Output 2, Output 3 and Output 4) each representing a speaker in the order of Front Left, Front Right, Rear Left and Rear Right. By the systematic manipulation of the audio levels of the four Sends, the same quadraphonic result as with the Surround Mixer plug-in is obtained. Figure 31 illustrates the first track with no standard Track Output but with four Sends, which each can be controlled by an Output Window. In the illustration the fader of Output 2 is set to 0 dB, the other three to ∞, which results in the sound only being perceived on the Front Right speaker.

By using this approach, surround panning is not limited to quadraphonic monitoring only. Five Sends can be used to set up a 5.0 environment. Keeping in mind that by sending the standard Track Output to other Auxiliary Input tracks, which each has a maximum of five Sends, numerous surround setups can be accomplished. Controlling the necessary Sends will depend on which focused track is selected.

The second approach is to manipulate a parameter from a plug-in, in this case the Focusrite d2.[102] Apart from controlling quadraphonic panning, the user can manipulate the centre frequency of the plug-in's *High-Mid Peak Filter*. Refer to section 3.4.1 for a discussion on control pages. In Figure 31 the plug-in is visible behind the four Output Windows.

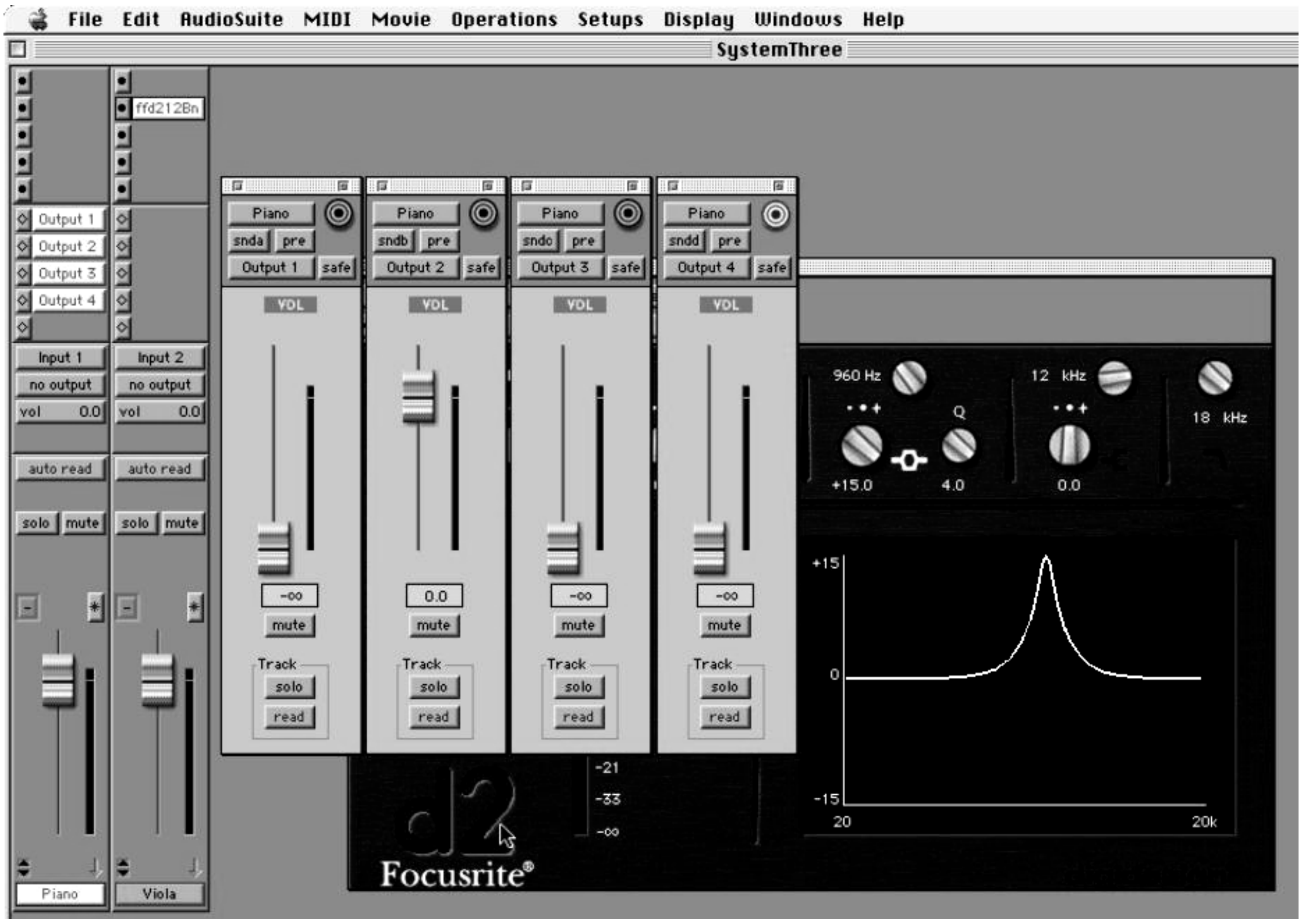


Figure 31. A snapshot of a basic Pro Tools session using SystemThree.

5.8.2 MIDI implementation

When executing the Java program, which is the mapping software for SystemThree, the user is prompted to setup the Pro Tools session. This setup includes the selection of the focused track and the passing through the null point position (as discussed in section 5.2.3) of each applicable fader and knob. The target track and the appropriate plug-in is manually selected by the user by using the Pro Tools computer's mouse.

[102] This Pro Tools TDM plug-in is created by Digidesign and Focusrite (http://www.focusrite.com) and is based on the "Red Range 2^{TM}" dual EQ by the latter company.

Toggling from the control of the Output Window to the Plug-in Window and vice versa is done by transmitting the necessary JL Cooper CS-10 switch MIDI code to Pro Tools. It was found that sending switch MIDI code lead Pro Tools to disregard switch instructions if the messages were sent one directly after the other. The solution was to implement time intervals of at least 40 milliseconds between each switch MIDI message. Intervals of a 100 milliseconds were added by using the Java instruction `Thread.sleep (100)`.

SystemThree is also implemented with an additional window for the use of generating basic MIDI messages. These messages are useful when supplementary functionality needs to be engaged for further manipulation, experimentation and future development of the system.

5.8.3 Determining and setting sound level

This subsection is devoted to the manipulation of the focused track. The process applied to each of the four channels of the quadraphonic setup involves three primary steps. Firstly, the attenuation of the original sound level according to the position of the ball image in the control square, as discussed in section 3.4.2, has to be determined (refer to Table 4). Secondly, the applicable Byte 2 value has to be verified (refer to Table 5) and thirdly the necessary MIDI code is transmitted.

As can be seen in Table 5, values from -∞ to +6 dB are spread through values 0 to 127. Unfortunately this leads to the fact that a precise dB setting can not be made, for example the important "0 dB" value is not represented. It was decided not to deal with the problem by rounding values off to the nearest adjacent value, but to set it to the nearest lower (in the direction of -∞) value. Various searching methods, such as *linear* and *binary*,[103] were considered in finding the appropriate current value in the list. As the current value will usually be in the same area as the previous value, extensive searching is not necessary. The outcome was to make use of a forward and backward linear search method relative to the previous dB value.

It is important to send MIDI messages so that a fader is moved from the previous position to the current (refer to the null point position discussed in section 5.2.3). To implement the rapid setting of the four faders in Figure 31, Byte 2 values are sent in increments and decrements of 2 if the previous and current values were not adjacent.

The reader is referred to the source code of the three systems for dealing with the logic of the algorithms.

[103] Linear search is a sequential process and can be demonstrated by reading a book from the beginning to find a specific word. Binary search can be demonstrated by finding an entry in a dictionary, by repeatedly opening the dictionary on a different page according to the position of the word relatively to the previous opening.

CHAPTER SIX
CONCLUSION

Summary

This chapter strives to consolidate the research process followed in this thesis. It does so by restating the current state of development of the three systems, after which the reader is introduced to developments that can be expected. The project is summed up: firstly, by discussing unsolved problems related to this thesis and secondly, pointing out what the reader can expect to receive in the thesis package as a whole. It closes with a synopsis of the contributions made by this thesis.

6.1 FUTURE DEVELOPMENT AND APPLICATIONS

6.1.1 Limitations of this thesis

The following limitations and exceptions are sorted in order of the data flow of the three systems. Possible solutions are mentioned, but are kept for future study.

Hand related problems

A first exception occurs when determining free or controlling movement (see section 3.1) when a hand is near the border of the camera's capturing view. The total area of finger objects in SystemTwo (see section 3.3.4) and the width of the hand in SystemThree (see section 4.2.1) are determining the type of movement. If the camera does not capture the entire profile of the fingers or hand, controlling movement will be identified. Compare the fourth image of Figure 19 to Figure 32. A simple solution is to ignore the availability of controlling movement near the border of the image. An advanced solution is discussed by Shamaie & Sutherland (2004) and involves velocity and acceleration calculation while tracking hand motion.

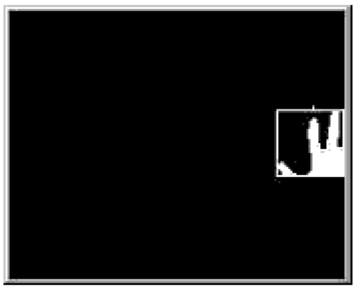


Figure 32. The small width of the white object determines controlling movement, but the hand is open. Also, the position determined for the hand is not the centre of the hand.

A second exception is found in SystemTwo (see section 3.3.4) where, for example, free movement is intended, but both hands' finger objects do not match and the hands are too close together. The image is split incorrectly into left and right hand images which results in false types of movements. A solution is to keep both hands' objects vertically aligned when too close together. See Figure 33.

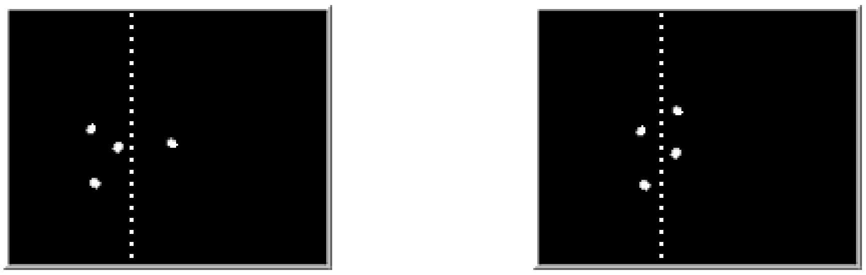


Figure 33. In *(a)* the dotted line represents the image split by separating left from right hand. It is incorrect as the right thumb is grouped with the left fingers and controlling movement is determined for the right hand. In *(b)* a solution is provided when hands are close together.

A third exception is portrayed in SystemOne and SystemTwo and involves the temporarily absence of a hand. This is caused by moving a hand out the capture view of the camera, a light fluctuation that leads to an incorrect threshold operation (see Figure 17), or in the case of occlusion where the emphasized objects are hidden, from the camera, by other body parts. The result is that the visible object is identified as both left and right hands. A simple solution involves the user exercising caution and an averaged threshold setting. For future development, a basic algorithm can be implemented to keep track of each hand's preceding motion.

A fourth exception happens when two hands traverse each other horizontally. In SystemOne and SystemTwo, if the left hand for example is placed on the other side of the right hand, the left hand is identified as the right hand. A basic solution is to not cross over the hands. For future study, research by Shamaie & Sutherland (2004) on bimanual movement is to be considered. A further solution incorporates the use of the EyesWeb *Imaging.BlobAnalysis* library and involves the tracking of multiple pixel concentrations. Related to this, another solution is to make use of different color objects attached to the hands and then the color objects can be tracked by EyesWeb.

EyesWeb related

It is found that the *Imaging.Input.FrameGrabber* block incorporates less latency than the DV facilities of the camera. The computer used for EyesWeb together with the video card does not utilize the Imaging.Input.FrameGrabber block. This problem needs attention for future video capturing for three-dimensional environments. A solution is the use of the Matrox meteor II frame grabber board[104] which is used by some other institutions concerning the use of EyesWeb.

As the MIDI implementation (see section 5.2.1) is designed for the three systems specifically, a different approach needs to be considered when dealing with a three-dimensional environment. As the maximum MIDI message bandwidth provided by EyesWeb is only three bytes, other transport mediums with a larger bandwidth should be investigated. Also when the JavaMIDI class receives too many messages at a time, the class resets itself and a small number of messages are ignored. For the reason that Java provides advanced network facilities, the use of EyesWeb network blocks also needs to be explored. The latter incorporates the use of the *Network.Output.SendToNetwork* block for dealing with *TCP/IP* (Transmission Control Protocol / Internet Protocol) and the *Network.Output.StringToOSC* block for dealing with *OSC* (Open Sound Control).[105]

JSyn related

SystemTwo only incorporates the use of WAV file types, but the loading and playing of AIFF type sound files can also be implemented in JSyn. However, only WAV files can be created in JSyn and to incorporate the use of AIFF files in SystemTwo, other pieces of software are needed to convert WAV into AIFF. The use of MP3 (MPEG, audio layer 3) file types were never considered as part of this thesis.

Exploration was done according to an automatic beat tracking implementation. The *PitchDetector* class could be useful when keeping the repetitive character of the type of music in mind. However, this class does not detect oscillation lower than 50 Hz and for example 120 BPM = 2 Hz is needed. Numerous studies[106] will be investigated in future.

[104] Refer to http://www.matrox.com/imaging/products/meteor2/home.cfm.

[105] Refer to http://www.cnmat.berkeley.edu/OpenSoundControl. The reader is also referred to http://www.mat.ucsb.edu/~c.ramakr/illposed/javaosc.html for information on "Java OSC".

[106] Such as Laroche, J. (2003) Efficient Tempo and Beat Tracking in Audio Recordings. In von Recklinghausen (ed.) *Journal of the Audio Engineering Society*, New York. Vol. 51, No. 4

6.1.2 Other ways of interaction

The implementation of the three systems can directly lead to a number of interactive environments. A system can be programmed to form a graphical interface between the Nord Lead 2 and a computer screen. Sounds can be stored on computer as in SystemOne, but also with the use of *system exclusive* messages, more information can be stored.

The use of *JMSL* (Java Music Specification Language) can lead to a more prominent use of JSyn. JMSL is derived from *HMSL* and focuses on algorithmic composition generated by a computer. The reader is referred to http://www.algomusic.com/jmsl for more information.

As the three systems provide visual and audio output, it is possible to use these studied concepts for non-musical applications. The type of virtual environments that can be designed depends on the developers' programming skills and innovation.

6.1.3 Three-dimensional

This thesis incorporates a practical implementation of gesture interaction in two-dimensional environments. Its value will be increased if studies in three-dimensional installations can be derived from it.

To capture a three dimensional environment by using EyesWeb, more than one camera has to be used for example for frontal and lateral views of the user. An electronic device, the *Video Multiplexer* (Mpx), developed at DIST, is used to interlace the images of two cameras, by switching between the two camera signals at a rate of 50 Hz.[107] Inside the EyesWeb patch, the signal is deinterlaced to create two signals, each corresponding to the original signals.

Considering the improvement of graphics in Java, the following step is the use of the Java 2D API which is part of the Java API. It involves the use of numerous shapes and textures, rescaling and rotating of graphic objects, displaying complex charts, image filtering, etc. A more advanced approach is the use of the Java 3D API, which involves the creation and manipulation of three-dimensional geometric objects and can be downloaded from the official Java website.

[107] Each camera captures at a rate of 25 Hz.

6.2 THE OUTCOME OF THIS THESIS

This study can be labelled a dynamic process and is therefore difficult to be considered an absolute final project. A first reason for this state of affairs is the fact that well supported, prominent and powerful software is used and expansion is limited solely by the software developer's creative and innovative abilities. It was the author who stated a need to select software where the focus fell on basic programming skills (implemented in Java) rather than using applications the way an end-user does. A second reason is that this thesis strives to set a departmental standard on par with related international research projects, and also to constantly improve itself. A third reason is that the thrust of this thesis revolves around conceptualized idealism of the future more than conventionality.

As mentioned before, this thesis makes use of hard- and software available in the Konservatorium. The intention is not to improve on the traditional interaction the three systems are based on, but to implement a degree of existent activity which could lead to future development in the field of gesture research and HCI. An unlimited number of models could have been selected and used instead of the three systems, but the aim was to make use of fairly complex and useful applications.

In a more practical sense, this thesis consists of a written document, introducing gesture research and discussing the three systems. The source code of Java software is printed and attached as Appendix D, but the data remains the author's property, as this thesis's focus falls on development and not usage. Although the source code is appended to this document, it represents the author's personal effort. The overall programming style and algorithms should be evaluated within the parameters set out in the scope of this thesis as in section 1.4. As the used software rely on regular upgrading and are dependent on computer specifications, installation software are not included. The applicable URL's are listed in the "References" section. An oral examination shall take place where the three systems will be installed, demonstrated and examined. This will probably be the only public event where the systems shall be implemented as described in this document. Therefore, video and audio capturing material of this event accompanies this thesis.

6.3 FINALLY

In the final analysis, the contributions made by this thesis touches on the following:

- It consists of a study that incorporates and combines numerous fields. The music technology field can be regarded as large and complex, presenting students with difficulties in understanding how different concepts integrate with one another. It has been observed that, for example, uncertainty concerning the difference between the representation of audio and MIDI information often occurs with students at undergraduate level. In this study MIDI is used conventionally as a protocol, but also as a medium for transporting non-music data, thereby clearly segregating the two. It also, for example, embeds the traditional audio engineering concepts of EQ and filtering within the unrelated computer software development world. It requires a student to conceptualise and think in terms of hard- and software circuit signal flow requirements. However, a lack of space prohibits a substantial investigation into the interdisciplinary nature of the project.
- The project provides a fully functional gesture interface implementation. Future theoretical and applied studies in this field stand to benefit from the lessons learnt during the course of its conceptualisation, design and execution.
- This study leaves the Konservatorium with a competitive edge, as it is the only continental music department in which applied research of this nature is conducted. Music students are afforded the opportunity of becoming involved in frontline applied research. The advantages of furthering this activity in the African artistic context, where music and gesture are inseparable, are obvious. Conversely, the Konservatorium stands to gain from the knowledge base of the students drawn towards this activity.
- The KEMUS[61] ensemble focuses strongly on the implementation of innovative ideas. This study provides a cost-effective platform that stand to benefit future ensemble composition projects.

In addition, a new discipline and creative activities have been initiated locally.

- In terms of computer programming, the project presents an end product towards which programming skills can be aimed. It provides for a focused computer programming curriculum that does not double other curricula, as the focus falls on the implementation of innovative music controllers, music and non-music gesture research and VEs.

- Moderate financial support stands to result in the implementation of a compact and portable environment encompassing peripheral equipment, hard- and software. The potential advantages of merging these in the context of gesture controlled musical and non-musical games and applications with ICT visions outlined in footnote 7, are obvious.
- The successful implementation of this study has resulted in the completion of important groundwork, from which the future large scale implementation shall benefit.

The primary intention for this thesis was to implement an active, functional and groundbreaking study, by designing a hyperinstrument with gesture interface for musical performance. As most of the concepts employed are already implemented at undergraduate level in the Konservatorium, a secondary aim was to create a range of potential post-graduate topics to interest prospective students. The ultimate vision would be to have students researching, designing, creating and installing virtual environments. In addition, these activities are seen as necessary to construct future alliances with other non-music departments. Apart form providing the KEMUS ensemble with creative tools, it is hoped that an informed public opinion will benefit from future activities.

REFERENCES PUBLISHED & PAPERS

Broughton, F. & Brewster, B. (2003) *How to DJ right: the art and science of playing records*. Grove Press, New York.

Budd, T. (1994) *Classic data structures in C++*. Addison-Wesley Publishing Company

Budd, T. (1997) *An introduction to object-oriented programming* (2nd ed.). Addison Wesley Longman

Cadoz, C. & Wanderley, M.M. (2000) *Gesture-Music*. Reprint from Wanderley, M.M. & Battier, M. (Eds.) Trends in Gestural Control of Music. IRCAM - Centre Pompidou.

Camurri, A. & Leman, M. (1997) Gestalt-Based Composition and Performance in Multimodal Environments. In Leman (Ed.) *Music, Gestalt. And Computing*. pp. 495-508, Springer.

Camurri, A., et al. (1999a) *EyesWeb – toward gesture and affect recognition in dance / music interactive systems*. In Proc. IEEE Multimedia Systems '99, Firenze

Camurri, A., et al. (1999b) *KANSEI Analysis of movement in Dance / Music interactive systems*. Proc. Of International Conference of Humanoid and Robot (HURO99), pp. 9-14, Japan

Camurri, A., et al. (2000a) *A real-time platform of interactive performance*. In Proc. ICMC2000, Berlin

Camurri, A., et al. (2000b) Expressiveness and physicality in interaction. In Leman, M. (Ed.), *Journal of New Music Research*. (29:3) pp. 187-198.

Camurri, A., et al. (2000c) *Synthesis of expressive movement*. In Proc. Intl. Conf. ICMC2000, Berlin

Camurri, A., et al. (2002) *Interactive Systems Design: A KANSEI-based Approach*. Proc. NIME2002, Dublin.

Camurri, A. & Volpe, G. (2004) Preface of *Gesture-Based Communication in Human-Computer Interaction*. Springer-Verlag, Berlin Heidelberg.

Department of Education. (2003) *Transforming Learning and Teaching through ICT*. Draft White Paper on e-Education.
Also available from http://www.capegateway.gov.za/Text/2004/11/e-education.pdf

Digidesign Inc. (2001a) *Reference Guide*. Palo Alto: Digidesign

Digidesign Inc. (2001b) *MIDI Control Surfaces Guide*. Palo Alto: Digidesign

Duckworth, W. (1999) Making music on the Web. *Leonardo Music Journal* (9:13-18)

Hahn, J., et al. (1998) Integrating sounds and motions in virtual environments. In *Presence*. (7:1) Massachusetts Institute of Technology. pp. 67-77.

Hamman, M. (1999) From symbol to semiotic: Representation, signification, and the composition of music interaction. In Leman, M. (Ed.), *Journal of New Music Research*. (28:2) pp. 90-104.

Hansen, K.F. (2002) The Basics of Scratching. In Leman, M. (Ed.), *Journal of New Music Research*. (31:4) pp. 357-365.

Hansen, K.F. & Bresin, R. (2003) Analysis of a Genuine Scratch Performance. In Camurri, A. & Volpe, G. (Ed.), *Gesture Workshop 2003, LNAI 2915*. Springer-Verlag, Berlin / Heidelberg. pp. 519-528.

Hoggarth, J. (2002) *How to be a DJ.* London: Penguin Books

Hubbard, J. (1999) *Programming with Java.* The McGraw-Hill Companies

Hubbard, J. (2000) *Programming with C++.* The McGraw-Hill Companies

Kirk, R. & Hunt, A. (1999) *Digital Sound Processing for Music & Multimedia*. Oxford: Focal Press.

Koffman, E.B. & Maxim, B.R. (1991) *Turbo Pascal: problem solving and program design* (3rd ed.). Addison-Wesley Publishing Company

Lovell, R. & Mitchell, J.D. (1995) Using human movement to control activities in theatrical environments. In N. Zahler (Ed.), *Proceedings for the Fifth Biennial Symposium for Arts and Technology*. New London: Connecticut College.

Leman, M. *Musical content processing for artistic and scientific applications.* (Paper in Status Nascendi)

López, J. & González, A. (2001) 3-D Audio with video tracking for multimedia environments. In Leman, M. (Ed.), *Journal of New Music Research*. (30:3) pp. 271-277.

McGrath, M. (2001) *JAVA 2 in easy steps*. Warwichshire: Computer Step.

McLaughlin, B. (2000) *Java^{TM} and XML*. O'Reilly & Associates, Inc: USA.

Machover, T., & Chung, J. (1989). Hyperinstruments: Musically intelligent and interactive performance and creativity systems. In *Proceedings of the 1989 International Computer Music Conference*. (ICMC1989) Columbus, Ohio, USA.

Marrin, T., et al. (1999*) Apparatus for controlling continuous behaviour through hand and arm gestures*. United States Patent no. 5875457, issued 23/02/1999.

Marrin Nakra, T. (2002) Synthesizing expressive music through the language of conduction. In Leman, M. (Ed.), *Journal of New Music Research*. (31:1) pp. 11-26.

Mathews, M. (1991) The radio baton and conductor program, or: Pitch, the most important and least expressive part of music. In *Computer Music Journal*. (15:4) pp. 37-46, Massachusetts Institute of Technology.

Morales-Manzanares, R., et al. (2001) SICIB: An Interactive Music Composition System Using Body Movements. In *Computer Music Journal*. (25:2) pp. 25-45, Massachusetts Institute of Technology.

Mulder, A. (1998) *Design of Virtual Three-Dimensional Instruments for Sound Control*. PhD. thesis. Burnaby, BC, Canada: Simon Fraser University.

Naughton, P. & Schildt, H. (1998) *Java 1.1: The Complete Reference. (2nd edition)* Osborne McGraw-Hill, U.S.A.

Nielsen, M. et al. (2004) A Procedure for Developing Intuitive and Ergonomic Gesture Interfaces for HCI. In Camurri, A. & Volpe, G. (Eds.) *Gesture-Based Communication in Human-Computer Interaction*. Springer-Verlag, Berlin Heidelberg. pp. 409-420.

Otheru, S. & Hashimoto, S. (1992) A new approach to music through vision. In *Understanding music with AI*. AAAI Press.

Paradiso, J. (1999) The Brain Opera technology: New instruments and gestural sensors for musical interaction and performance. In Leman, M. (Ed.), *Journal of New Music Research*. (28:2) pp. 130-149.

Peel, I. (2001) Dance music. In Sadie (Ed.), *The New Grove Dictionary of Music and Musicians*. London: Macmillan Publishers Ltd. pp. 909-911.

Perry, G. (2000) *C by example*. Que Corporation, U.S.A.

Pressing, J. (1992) *Synthesizer Performance and Real-time Techniques*. A-R Editions, Madison, Wisconsin.

Rowe, R. (1993) *Interactive music systems*. Cambridge, MA: Cambridge University Press.

Rowe, R. (2001) *Machine musicianship*. Massachusetts Institute of Technology.

Russ, M. (1996) *Sound Synthesis and Sampling*. Focal Press, Oxford.

Sams, H. W. (1999) *Digital Audio Dictionary*. Indianapolis: Prompt Publications.

Shamaie, A. & Sutherland, A. (2004) A Dynamic Model for Real-Time Tracking of Hands in Bimanual Movements. In Camurri, A. & Volpe, G. (Eds.) *Gesture-Based Communication in Human-Computer Interaction*. Springer-Verlag, Berlin Heidelberg. pp. 409-420.

Slaney, C. (2002) *The DJ handbook*. PC Publishing, Kent, UK.

Sapir, S. (2002) Gestural control of digital audio environments. In Leman, M. (Ed.), *Journal of New Music Research*. (31:2) pp. 119-129.

Souvignier, T. (2003) *The world of DJs and the turntable culture*. Hal Leonard Corporation, Milwaukee, USA.

Spitzer, J. & Zaslaw, N. (2001) *Conducting*. In Sadie (ed.) The New Grove Dictionary of Music and Musicians. Vol. 6. London: Macmillan Publishers Ltd. p.261.

Ulyate, R. & Bianciardi, D. (2002) The Interactive Dance Club: Avoiding Chaos in a Multi-Participant Environment. In *Computer Music Journal*. (26:3) pp. 40-49, Massachusetts Institute of Technology.

Ungvary, T. & Vertegaal, R. (2000) Designing Musical Cyberinstruments with Body and Soul in Mind. In Leman, M. (Ed.), *Journal of New Music Research*. (29:3) pp. 245-255.

Vail, M. (2000) *Keyboard Magazine presents Vintage Synthesizers*. Miller Freeman Inc., San Francisco.

Vermeulen, A., et al. (2000*) The Elements of Java^{TM} Style*. Cambridge University Press.

Winkler, T. (1998) *Composing Interactive Music*. Massachusetts Institute of Technology.

Wright, M., et al. (2000) Analysis / Synthesis comparison. In *Organised Sound*. (5:3) pp. 173-189, Cambridge University Press.

REFERENCES
WEB SITES & OTHER MEDIA

Burk, P. *JSyn – Audio Software Synthesis API and Plugins for Java.*
http://www.softsynth.com/jsyn (accessed 05/03/2002)

Campione, M., et al. (2004) *The Java^{TM} Tutorial.*
http://java.sun.com/docs/books/tutorial

Camurri, A., et al. *The EyesWeb Project.*
http://musart.dist.unige.it (accessed 16/04/2002)

Clark, D.J. (2002) *MIT grad directs Spielberg in the science of moviemaking.* MIT news office. Massachusetts Institute of Technology.
http://web.mit.edu/newsoffice/2002/underkoffler-0717.html

Clavia – digital musical instruments. (1996-2004)
http://www.clavia.se

Crenshaw, B. (2003) *Intellect: Techno House Progressive.* Stepfilm LLC, San Francisco, CA. [DVD]

Cycling '74 – Max/MSP 4.5.
http://www.cycling74.com

Didkovsky, N. *Java Music Specification Language.*
http://www.algomusic.com/jmsl (accessed 05/03/2002)

Eckel, B. (2000) *Thinking in Java.* (2nd ed.)
http://www.mindview.net

Fagernes, S. & Hagen, S. (2002) The MEGA Project - Virtual Emotions in Mixed Reality.
http://www.telenor.no/fou/publisering/notater/N_50_2002.pdf

Hansen, K.F. (2000) *Turntable music.*
http://www.speech.kth.se/~hansen/turntablemusic.html

Holt Software Associates Inc. - Ready to program with Java^{TM} Technology.
http://www.holtsoft.com

Java – Sun Microsystems, Inc.
http://java.sun.com (Sun Microsystems website for Java developers)
http://java.sun.com/j2se/1.4.2/download.html (Java 2 Platform, Standard Edition, v 1.4.2 (J2SE))
http://java.sun.com/j2se/1.4.2/docs/index.html (Java 2 SDK, Standard Edition, Documentation, v 1.4.2)

JCreator (Java IDE) – Xinox Software
http://www.jcreator.com

JL Cooper Electronics
http://www.jlcooper.com

JSyn Newsgroup
http://music.columbia.edu/mailman/listinfo/jsyn

Machover, T. (1992) *"classic" hyperinstruments - 1986-1992 - a composer's approach to the evolution of intelligent musical instruments.*
http://brainop.media.mit.edu/Archive/Hyperinstruments/classichyper.html

Machover, T. (1995) *Technology and Creative Expression.*
http://brainop.media.mit.edu/Archive/Hyperinstruments/creative.html.

Machover, T. (1999) *Technology and the future of music.* Interview by Oteri, F. 18 August 1999. NewMusicBox (1:6).
http://www.newmusicbox.org

Manor, J. *Gestural VJ Software.*
http://acg.media.mit.edu/people/manor/gesture/manor_final_proj.pdf

Marian Digital Audio Electronics (2003)
http://www.marian.de/english/news-history.htm

Marian Digital Audio Electronics (2004)
http://www.marian.de/files/marc8midi/w2k/readme.htm

Marrin, T.A. (1996) *Toward an understanding of musical gesture: Mapping expressive intention with the digital baton.* Master's thesis, Massachusetts Institute of Technology.
http://brainop.media.mit.edu/Archive/Thesis.html

Marsanyi, R. (2001) JavaMIDI library and Documentation.
http://www.softsynth.com/javamidi

Murphy, D. *Extracting Arm Gestures for VR using EyesWeb.*
http://www.maths.tcd.ie/dec (accessed 16/04/2002)

New, J.R. et al. (2003) *Facilitating User Interaction with Complex Systems via Hand Gesture Recognition.* Knowledge Systems Laboratory, Jacksonville State University.
http://ksl.jsu.edu/publications/NewEtAl-ACMSE2003.pdf

O' Hagan, R. (2001) Finger *Track - A Robust and Real-Time Gesture Interface.*
http://syseng.anu.edu.au/rsl/rsl_papers/AI97f.pdf (accessed 16/03/2003)

Pluijms, G. (2002) *Advanced Vinyl Handling, Introduction to DJ'ing and Mixing.*
http://music.hyperreal.org/dj/AVH/index.html

Schmitz, M. (2003) *Human Computer Interaction in Science Fiction Movies.*
http://w5.cs.uni-sb.de/~butz/teaching/ie-ss03/papers/HCIinSF

Stanton Final Scratch
http://www.finalscratch.com

STEIM (studio for electro-instrumental music)
http://www.steim.org

Top DJ Gear (2004) *DJ BASICS / FAQ*.
http://www.topdjgear.com/howtobecomedj.html

Waisvisz, M. (1999) *Gestural round table*. STEIM.
http://www.steim.org/steim/texts.php?id=4

Wanderley, M.M. & Battier, M. (2000) *Trends in Gestural Control of Music*. IRCAM - Centre Pompidou.
http://recherche.ircam.fr/equipes/analyse-synthese/wanderle/Gestes/Externe

Wilson, S. et al. (2001) *Using CORBA Middleware to Support the Development of Distributed Virtual Environment Applications*.
http://wscg.zcu.cz/wscg2001/Papers_2001/R303.pdf (accessed 16/03/2003)

APPENDIX A
TABLES

Parameter	Byte1	Byte2	Result
LFO 1 Waveform	20	0	Random
		1	Sawtooth
		2	Triangle
		3	Square
		4	Soft Random
LFO 1 Destination	21	0	PW
		1	Filter
		2	Osc 2
		3	Osc 1 + 2
		4	FM
LFO 2 Destination / Arpeggio Mode	24	0	Arp Down
		1	Arp Up
		2	Arp Up/Down
		3	Amp
		4	Osc 1 + 2
		5	Arp Random
		6	(Arp) Echo
		7	Filter
		8	(Switch Arp off) + Filter
Modulation Envelope Destination	28	0	Osc 2
		1	FM
		2	PW
		3	(Off)
Osc 1 Waveform	30	0	Pulse
		1	Sawtooth
		2	Triangle
		3	Sine
Osc 2 Waveform	31	0	Pulse
		1	Sawtooth
		2	Triangle
		3	Noise
Osc 2 Keyboard Tracking	34	0	(Off)
		1	(On)
Oscillator Sync	35	0	(Off)
		1	Sync
		2	Ring Modulation
		3	Sync + Ring Modulation

Parameter	Byte1	Byte2	Result
Filter Mode	44	0	LP 12 dB
		1	LP 24 dB
		2	HP 24 dB
		3	BP
		4	Notch + LP
Filter Velocity	45	0	(Off)
		1	(On)
Filter Keyboard Track	46	0	(Off)
		1	1/3
		2	2/3
		3	FULL
Filter Distortion	80	0	(Off)
		1	(On)
Unison	16	0	(Off)
		1	(On)
Poly/Legato/Mono	15	0	Mono
		1	Legato
		2	Poly
Portamento Auto	65	0	(Off)
		1	(On)
Oct Shift	17	0	(2 octaves lower)
		1	(1 octave lower
		2	(Normal)
		3	(1 octave higher)
		4	(2 octaves higher)
Mod Wheel Destination	18	0	Filter
		1	FM
		2	Osc 2
		3	LFO 1
		4	Morph

Table 2. Nord Lead 2 Button MIDI implementation. *Status byte* = $B0_{16} – B3_{16}$.

Parameter	**Byte1**	**Byte2**
LFO 1 Rate	19	0 – 127
LFO 1 Amount	22	
LFO 2 / Arpeggio Rate	23	
LFO 2 Amount / Arpeggio Range	25	
Modulation Envelope Attack	26	
Modulation Envelope Decay	27	
Modulation Envelope Amount	29	0 – 127 (Normal = 64)
Oscillator FM Depth	70	0 – 127
Osc 2 Semitones	78	0 – 120 (Normal = 60)
Osc 2 Fine Tune	33	1 – 127 (Normal = 64)
Oscillator Pulse Width	79	0 – 127
Oscillator Mix	8	
Amplifier Envelope Attack	73	
Amplifier Envelope Decay	36	
Amplifier Envelope Sustain	37	
Amplifier Envelope Release	72	
Gain	7	
Filter Envelope Attack	38	
Filter Envelope Decay	39	
Filter Envelope Sustain	40	
Filter Envelope Release	41	
Filter Cutoff	74	
Filter Resonance	42	
Filter Envelope Amount	43	
Portamento Time	5	
Modulation Wheel	1	
Pitch Bend (*Status byte* = $E0_{16} – E3_{16}$)	0	0 – 127 (Normal = 64)

Table 3. Nord Lead 2 Knob MIDI implementation. *Status byte* = $B0_{16} – B3_{16}$.

Position	dB
-100	0
-98	0
-96	0
-94	0
-92	0
-90	0
-88	0
-86	0.1
-84	0.1
-82	0.1
-80	0.1
-78	0.1
-76	0.2
-74	0.2
-72	0.2
-70	0.2
-68	0.3
-66	0.3
-64	0.4
-62	0.4
-60	0.4
-58	0.5
-56	0.5
-54	0.6
-52	0.6
-50	0.7
-48	0.7
-46	0.8
-44	0.9
-42	0.9
-40	1.0
-38	1.1
-36	1.1
-34	1.2
-32	1.3
-30	1.4
-28	1.5
-26	1.6
-24	1.6
-22	1.7
-20	1.8
-18	1.9
-16	2.0
-14	2.2
-12	2.3
-10	2.4
-8	2.5
-6	2.6
-4	2.7
-2	2.9
0	3
2	3.1
4	3.3
6	3.4
8	3.6
10	3.7
12	3.9
14	4.1
16	4.3
18	4.4
20	4.6
22	4.8
24	5.0
26	5.2
28	5.4
30	5.6
32	5.9
34	6.1
36	6.3
38	6.6
40	6.9
42	7.1
44	7.4
46	7.7
48	8.0
50	8.3
52	8.7
54	9.0
56	9.4
58	9.8
60	10.2
62	10.6
64	11.1
66	11.6
68	12.1
70	12.6
72	13.2
74	13.9
76	14.5
78	15.3
80	16.1
82	17.0
84	18.0
86	19.2
88	20.5
90	22.1
92	24.0
94	26.6
96	30.1
98	36.5
100	∞

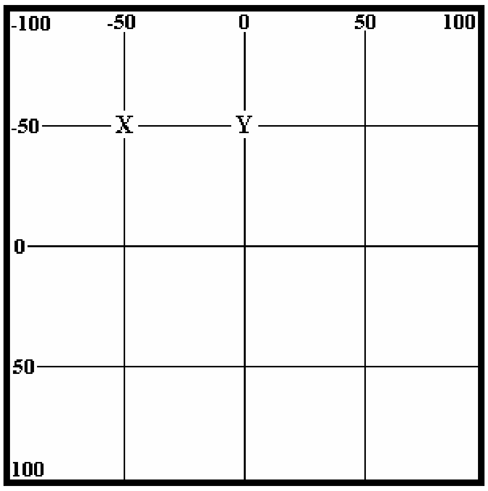


Table 4. Pro Tools quad position for one channel (for example Front Left) and dB attenuation. The illustration shows the position placement of the Front Left speaker. For example, the attenuation at point X is [-50] dB + [-50] dB = 0.7 dB + 0.7 dB = 1.4 dB, and at point Y is [-50] dB + [0] dB = 0.7 dB + 3 dB = 3.7 dB.

Byte 2	dB
0	-∞
1	-110
2	-104
3	-98.8
4	-93.3
5	-87.7
6	-82.1
7	-76.4
8	-70.7
9	-66.3
10	-62.1
11	-58.5
12	-55.4
13	-52.3
14	-49.4
15	-47.0
16	-44.5
17	-42.1
18	-39.8
19	-38.3
20	-36.7
21	-35.2
22	-33.7
23	-32.1
24	-30.6
25	-29.5
26	-28.6
27	-27.7
28	-26.8
29	-26.0
30	-25.1
31	-24.2
32	-23.4
33	-22.5
34	-21.6
35	-20.8
36	-20.0
37	-19.6
38	-19.2
39	-18.9
40	-18.5
41	-18.2
42	-17.8
43	-17.5
44	-17.1
45	-16.7
46	-16.4
47	-16.0
48	-15.7
49	-15.3
50	-15.0
51	-14.6
52	-14.3
53	-14.0
54	-13.7
55	-13.3
56	-13.0
57	-12.7
58	-12.4
59	-12.0
60	-11.7
61	-11.4
62	-11.1
63	-10.7
64	-10.4
65	-10.1
66	-9.8
67	-9.4
68	-9.1
69	-8.8
70	-8.5
71	-8.2
72	-7.8
73	-7.5
74	-7.2
75	-6.9
76	-6.6
77	-6.2
78	-5.9
79	-5.6
80	-5.3
81	-5.0
82	-4.6
83	-4.3
84	-4.0
85	-3.7
86	-3.4
87	-3.0
88	-2.7
89	-2.4
90	-2.1
91	-1.8
92	-1.4
93	-1.1
94	-0.8
95	-0.5
96	-0.2
97	+0.1
98	+0.3
99	+0.5
100	+0.7
101	+0.8
102	+1.0
103	+1.2
104	+1.4
105	+1.6
106	+1.8
107	+2.0
108	+2.1
109	+2.3
110	+2.5
111	+2.7
112	+2.9
113	+3.1
114	+3.3
115	+3.5
116	+3.7
117	+3.9
118	+4.1
119	+4.3
120	+4.5
121	+4.6
122	+4.8
123	+5.0
124	+5.2
125	+5.4
126	+5.6
127	+6.0

Table 5. JL Cooper CS-10 MIDI Byte2 and Pro Tools Fader dB setting.

MIDI	Note	Hz
0	C-1	8.176
1	C#-1	8.662
2	D-1	9.177
3	D#-1	9.723
4	E-1	10.301
5	F-1	10.913
6	F#-1	11.562
7	G-1	12.250
8	G#-1	12.978
9	A-1	13.750
10	A#-1	14.568
11	B-1	15.434
12	C0	16.352
13	C#0	17.324
14	D0	18.354
15	D#0	19.445
16	E0	20.602
17	F0	21.827
18	F#0	23.125
19	G0	24.500
20	G#0	25.957
21	A0	27.500
22	A#0	29.135
23	B0	30.868
24	C1	32.703
25	C#1	34.648
26	D1	36.708
27	D#1	38.891
28	E1	41.203
29	F1	43.654
30	F#1	46.249
31	G1	48.999
32	G#1	51.913
33	A1	55.000
34	A#1	58.270
35	B1	61.735
36	C2	65.406
37	C#2	69.296
38	D2	73.416
39	D#2	77.782
40	E2	82.407
41	F2	87.307
42	F#2	92.499
43	G2	97.999
44	G#2	103.826
45	A2	110.000
46	A#2	116.541
47	B2	123.471
48	C3	130.813
49	C#3	138.591
50	D3	146.832
51	D#3	155.563
52	E3	164.814
53	F3	174.614
54	F#3	184.997
55	G3	195.998
56	G#3	207.652
57	A3	220.000
58	A#3	233.082
59	B3	246.942
60	C4	261.626
61	C#4	277.183
62	D4	293.665
63	D#4	311.127
64	E4	329.628
65	F4	349.228
66	F#4	369.994
67	G4	391.995
68	G#4	415.305
69	A4	440.000
70	A#4	466.164
71	B4	493.883
72	C5	523.251
73	C#5	554.365
74	D5	587.330
75	D#5	622.254
76	E5	659.255
77	F5	698.456
78	F#5	739.989
79	G5	783.991
80	G#5	830.609
81	A5	880.000
82	A#5	932.328
83	B5	987.767
84	C6	1046.502
85	C#6	1108.731
86	D6	1174.659
87	D#6	1244.508
88	E6	1318.510
89	F6	1396.913
90	F#6	1479.978
91	G6	1567.982
92	G#6	1661.219
93	A6	1760.000
94	A#6	1864.655
95	B6	1975.533
96	C7	2093.005
97	C#7	2217.461
98	D7	2349.318
99	D#7	2489.016
100	E7	2637.020
101	F7	2793.826
102	F#7	2959.955
103	G7	3135.963
104	G#7	3322.438
105	A7	3520.000
106	A#7	3729.310
107	B7	3951.066
108	C8	4186.009
109	C#8	4434.922
110	D8	4698.636
111	D#8	4978.032
112	E8	5274.041
113	F8	5587.652
114	F#8	5919.911
115	G8	6271.927
116	G#8	6644.875
117	A8	7040.000
118	A#8	7458.620
119	B8	7902.133
120	C9	8372.018
121	C#9	8869.844
122	D9	9397.273
123	D#9	9956.063
124	E9	10548.08
125	F9	11175.30
126	F#9	11839.82
127	G9	12543.85

Table 6. MIDI, Note and frequency conversion.

(Adapted from http://www.esm.rochester.edu/onlinedocs/Csound/Appendices/table1.html)

STATUS BYTE					DATA BYTES	
1st Byte Value			**Function**		**2nd Byte**	**3rd Byte**
Binary	**Hex**	**Dec**				
10000000	80	128	Chan 1	Note off	Note Number (0-127) See Table 6	Note Velocity (0-127)
10000001	81	129	Chan 2			
10000010	82	130	Chan 3			
10000011	83	131	Chan 4			
10000100	84	132	Chan 5			
10000101	85	133	Chan 6			
10000110	86	134	Chan 7			
10000111	87	135	Chan 8			
10001000	88	136	Chan 9			
10001001	89	137	Chan 10			
10001010	8A	138	Chan 11			
10001011	8B	139	Chan 12			
10001100	8C	140	Chan 13			
10001101	8D	141	Chan 14			
10001110	8E	142	Chan 15			
10001111	8F	143	Chan 16			
10010000	90	144	Chan 1	Note on		
10010001	91	145	Chan 2			
10010010	92	146	Chan 3			
10010011	93	147	Chan 4			
10010100	94	148	Chan 5			
10010101	95	149	Chan 6			
10010110	96	150	Chan 7			
10010111	97	151	Chan 8			
10011000	98	152	Chan 9			
10011001	99	153	Chan 10			
10011010	9A	154	Chan 11			
10011011	9B	155	Chan 12			
10011100	9C	156	Chan 13			
10011101	9D	157	Chan 14			
10011110	9E	158	Chan 15			
10011111	9F	159	Chan 16			
10100000	A0	160	Chan 1	Polyphonic Aftertouch		Aftertouch amount (0-127)
10100001	A1	161	Chan 2			
10100010	A2	162	Chan 3			
10100011	A3	163	Chan 4			
10100100	A4	164	Chan 5			
10100101	A5	165	Chan 6			
10100110	A6	166	Chan 7			
10100111	A7	167	Chan 8			
10101000	A8	168	Chan 9			
10101001	A9	169	Chan 10			
10101010	AA	170	Chan 11			
10101011	AB	171	Chan 12			
10101100	AC	172	Chan 13			
10101101	AD	173	Chan 14			
10101110	AE	174	Chan 15			
10101111	AF	175	Chan 16			
10110000	B0	176	Chan 1	Control/ Mode change	*	*
10110001	B1	177	Chan 2			
10110010	B2	178	Chan 3			
10110011	B3	179	Chan 4			
10110100	B4	180	Chan 5			
10110101	B5	181	Chan 6			
10110110	B6	182	Chan 7			
10110111	B7	183	Chan 8			
10111000	B8	184	Chan 9			
10111001	B9	185	Chan 10			
10111010	BA	186	Chan 11			
10111011	BB	187	Chan 12			
10111100	BC	188	Chan 13			
10111101	BD	189	Chan 14			
10111110	BE	190	Chan 15			
10111111	BF	191	Chan 16			

(to next page)

Table 7. MIDI Expanded Status Bytes List.
(adapted from http://www.midi.org/about-midi/table2.shtml, which is adapted from "MIDI by the Numbers" by D. Valenti, Electronic Musician 2/88, Updated 1995 By the MIDI Manufacturers Association).

(from previous page)

11000000	C0	192	Chan 1	Program change	Program # (0-127)	NONE
11000001	C1	193	Chan 2			
11000010	C2	194	Chan 3			
11000011	C3	195	Chan 4			
11000100	C4	196	Chan 5			
11000101	C5	197	Chan 6			
11000110	C6	198	Chan 7			
11000111	C7	199	Chan 8			
11001000	C8	200	Chan 9			
11001001	C9	201	Chan 10			
11001010	CA	202	Chan 11			
11001011	CB	203	Chan 12			
11001100	CC	204	Chan 13			
11001101	CD	205	Chan 14			
11001110	CE	206	Chan 15			
11001111	CF	207	Chan 16			
11010000	D0	208	Chan 1	Channel Aftertouch	Aftertouch amount (0-127)	
11010001	D1	209	Chan 2			
11010010	D2	210	Chan 3			
11010011	D3	211	Chan 4			
11010100	D4	212	Chan 5			
11010101	D5	213	Chan 6			
11010110	D6	214	Chan 7			
11010111	D7	215	Chan 8			
11011000	D8	216	Chan 9			
11011001	D9	217	Chan 10			
11011010	DA	218	Chan 11			
11011011	DB	219	Chan 12			
11011100	DC	220	Chan 13			
11011101	DD	221	Chan 14			
11011110	DE	222	Chan 15			
11011111	DF	223	Chan 16			
11100000	E0	224	Chan 1	Pitch wheel control	Pitch wheel LSB (0-127)	Pitch wheel MSB (0-127)
11100001	E1	225	Chan 2			
11100010	E2	226	Chan 3			
11100011	E3	227	Chan 4			
11100100	E4	228	Chan 5			
11100101	E5	229	Chan 6			
11100110	E6	230	Chan 7			
11100111	E7	231	Chan 8			
11101000	E8	232	Chan 9			
11101001	E9	233	Chan 10			
11101010	EA	234	Chan 11			
11101011	EB	235	Chan 12			
11101100	EC	236	Chan 13			
11101101	ED	237	Chan 14			
11101110	EE	238	Chan 15			
11101111	EF	239	Chan 16			
11110000	F0	240	System Exclusive		**	**
11110001	F1	241	MIDI Time Code Qtr. Frame		-see spec-	-see spec-
11110010	F2	242	Song Position Pointer		LSB	MSB
11110011	F3	243	Song Select(Song #)		(0-127)	NONE
11110100	F4	244	Undefined (Reserved)		?	?
11110101	F5	245	Undefined (Reserved)		?	?
11110110	F6	246	Tune request		NONE	NONE
11110111	F7	247	End of SysEx (EOX)			
11111000	F8	248	Timing clock			
11111001	F9	249	Undefined (Reserved)			
11111010	FA	250	Start			
11111011	FB	251	Continue			
11111100	FC	252	Stop			
11111101	FD	253	Undefined (Reserved)			
11111110	FE	254	Active Sensing			
11111111	FF	255	System Reset			

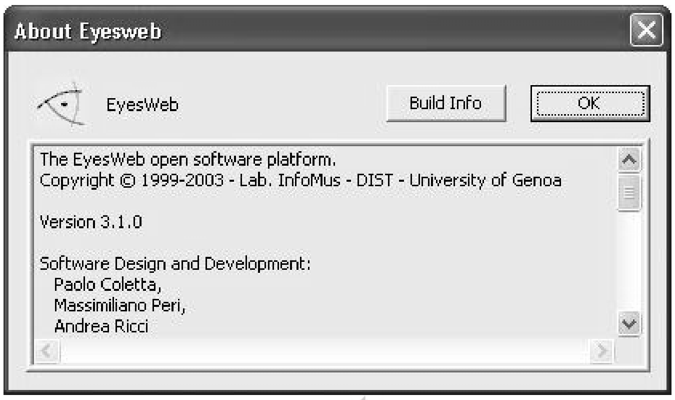
* Note: See “Table 3” from the website source.

** Note: System Exclusive (data dump) 2nd byte= Vendor ID (or Universal Exclusive) followed by more data bytes and ending with EOX.

APPENDIX B
LICENCES

The following sections contain quotes adopted from the applicable “About”, “License Agreement” and website sources.

EyesWeb



The EyesWeb open software platform.
Copyright © 1999-2003 - Lab. InfoMus - DIST - University of Genoa

Version 3.1.0

Software Design and Development:
Paolo Coletta,
Massimiliano Peri,
Andrea Ricci

Figure 34. About EyesWeb.

“EyesWeb Platform Download Page”
“EYESWEB CAN BE FREELY DOWNLOADED BUT PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE”

“License agreement
Use of EyesWeb (hereinafter 'SOFTWARE') is contingent on your agreement to the following terms:

WARRANTY & USE: DIST - University of Genoa grants you a limited, non-exclusive license to use the SOFTWARE free of charge for ANY purpose, commercial or private, without restrictions. DIST - University of Genoa makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty. DIST - University of Genoa is not obligated to provide maintenance or updates for the SOFTWARE.

DISTRIBUTION: the SOFTWARE, in original version or in part, may be freely distributed, provided that this copyright and permission notice appear on all copies and supporting documentation, and that the DIST - University of Genoa copyright notices are referred in the following ways: a) DIST - University of Genoa's copyright notice should be included in the documentation, regardless of the media used to supply the documentation; b) The DIST - University of Genoa and EyesWeb Logos have to appear on software packages based on EyesWeb or using it, and on any related promotional material (DIST - University of

Genoa makes the logos available on the EyesWeb ftp site ftp://infomus.dist.unige.it); c) in the 'about box' of the product, in the case that it is not the EyesWeb about box, DIST - University of Genoa and EyesWeb must be cited in the following manner: EyesWeb is copyright (c) Laboratorio di Informatica Musicale - DIST - University of Genoa (http://infomus.dist.unige.it); d) any public use of EyesWeb, or the distribution of any application based on EyesWeb, must be preliminarily notified to staff@infomus.dist.unige.it; e) this license must be notified to any third party to which EyesWeb is redistributed.

This license only covers EyesWeb and the libraries provided in the original installer. Other extensions (libraries or patches), provided by DIST or by third parties, may be subject to any license, provided that it is not in contrast with this license."

JSyn

"JSyn Licenses

JSyn is available under a variety of licenses.

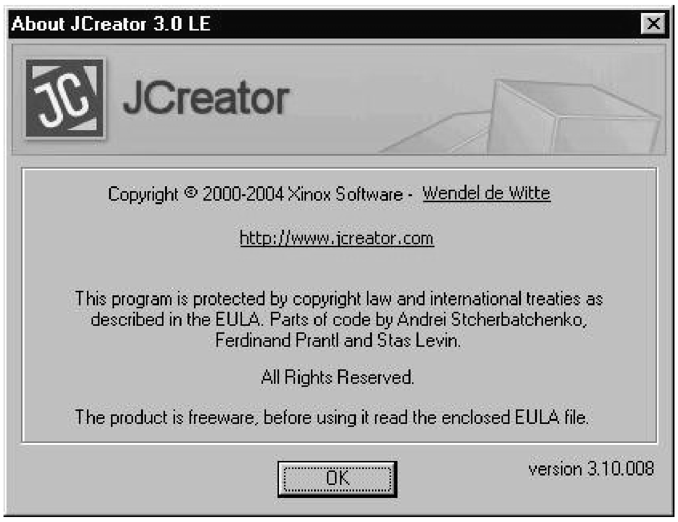
- A Free SDK Lite is available for non-commercial personal and educational use.
- Registered Developer licenses are $90 ($45 for students). Registered developers have access to additional resources and may use JSyn on small-scale commercial site with less than US$ 20,000.00 gross revenue per year.
- An Academic Site Licenses is $500 for the first year, $200 per additional year. This gives all faculty and students of a department the equivalent of the Registered Developer license. They will have access to a shared account on the restricted site. An entire campus may be licensed for $1000 for the first year, $400 per additional year.
- A Redistribution License is required to distribute any parts of JSyn. This is needed if you were to include the JSyn plugin, libraries or classes with your own products.
- A Commercial WebSite License is required to use JSyn on a commercial site with more than US$ 20,000.00 gross revenue per year.

The CSyn 'C' based synthesis engine that underlies JSyn is also available for license for redistribution with games, or musical applications."

JavaMIDI

"JavaMIDI may not be used in a commercial product without the written approval of Robert Marsanyi. This is a beta version for evaluation purposes only. We are not responsible for any damage you incur by using this package - use at your own risk. Standard disclaimers apply. If you use JavaMIDI in your own projects, you should include a credit for JavaMIDI, and a comment to the effect that. Robert Marsanyi retains the copyright."

JCreator 3.0 LE



Copyright © 2000-2004 Xinox Software - Wendel de Witte

http://www.jcreator.com

This program is protected by copyright law and international treaties as described in the EULA. Parts of code by Andrei Stcherbatchenko, Ferdinand Prantl and Stas Levin.

All Rights Reserved.

The product is freeware, before using it read the enclosed EULA file.

version 3.10.008

Figure 35. About JCreator 3.0 LE.

"END-USER LICENSE AGREEMENT FOR JCREATOR LE"
"This End-User License Agreement ('EULA') is a legal agreement between you and Xinox Software for the Xinox Software products identified above, which may include computer software and associated media, electronic documentation and printed materials ('The Software')."

"You are hereby licensed to make as many copies of the installation package for The Software as you wish; give exact copies of the original installation package for The Software to anyone; and distribute the original installation package for The Software in its unmodified form via electronic or other means. The Software must be clearly identified as a freeware or shareware version where described."

J2SE v 1.4.2_06 SDK

"Sun Microsystems, Inc.

Binary Code License Agreement for the

JAVATM 2 SOFTWARE DEVELOPMENT KIT (J2SDK), STANDARD EDITION, VERSION 1.4.2_X"

"'Software' means the identified above in binary form, any other machine readable materials (including, but not limited to, libraries, source files, header files, and data files), any updates or error corrections provided by Sun, and any user manuals, programming guides and other documentation provided to you by Sun under this Agreement. "Programs" mean Java applets and applications intended to run on the Java 2 Platform, Standard Edition (J2SE^{TM} platform) platform on Java-enabled general purpose desktop computers and servers."

"Subject to the terms and conditions of this Agreement, including, but not limited to the Java Technology Restrictions of the Supplemental License Terms, Sun grants you a non-exclusive, non-transferable, limited license without license fees to reproduce and use internally Software complete and unmodified for the sole purpose of running Programs. Additional licenses for developers and/or publishers are granted in the Supplemental License Terms."

APPENDIX C
EYESWEB PATCHES

Blocks			
DV	Imaging.Input.DVCamInput	IMG	Imaging.Output.Display
MIRROR	Imaging.Operations.Mirror		Imaging.Conversion.ColorToGray
	Imaging.Operations.ThresholdBitonal	MEDIAN	Imaging.Filters.NonlinearFilter
	Imaging.FeatureCalc.BoundingRect		Imaging.Operations.Split
	Imaging.FeatureCalc.Baricenter	[ ] x	Math.Matrix.GetEntry
X + Y	Math.Scalar.BinaryOp	X ÷ C	Math.Scalar.ConstOp
[ ] [ ]	Math.Matrix.Extract	x [ ]	Math.Matrix.SetEntry
i r	Math.Scalar.DomainConv	αF	Math.Scalar.AlphaFilter
X LEFT SHIFT	Math.Scalar.UnaryOpBitwise	X > C	Math.Scalar.UnaryOpLogical
Status byte	Output.Math.Scalar.Display	MiDi	Midi.Conversions.ScalarToMessage
MiDi	Midi.Operations.MessageCheckDuplicates	MiDi FiELD	Midi.Operations.MessageField
MiDi OUT	Midi.Output.MessageOutput		

ParamChangers			
Edit Ctrl	EditCtrl		HorizSliderCtrl

Table 8. EyesWeb Block and ParamChanger titles with graphical blocks. As a block setting has an influence on the graphical representation, not all used graphical representations are displayed here.

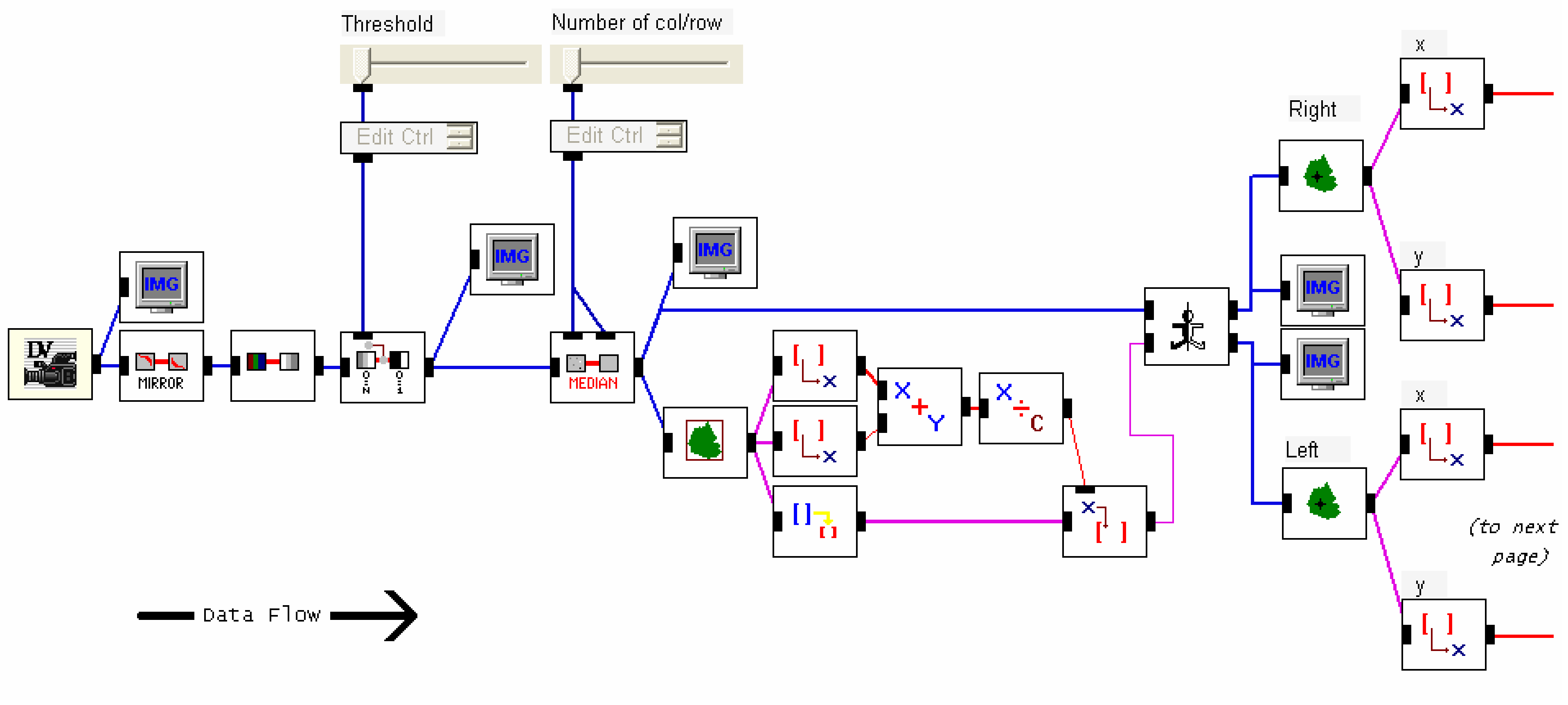


Figure 36. SystemOne EyesWeb structure.

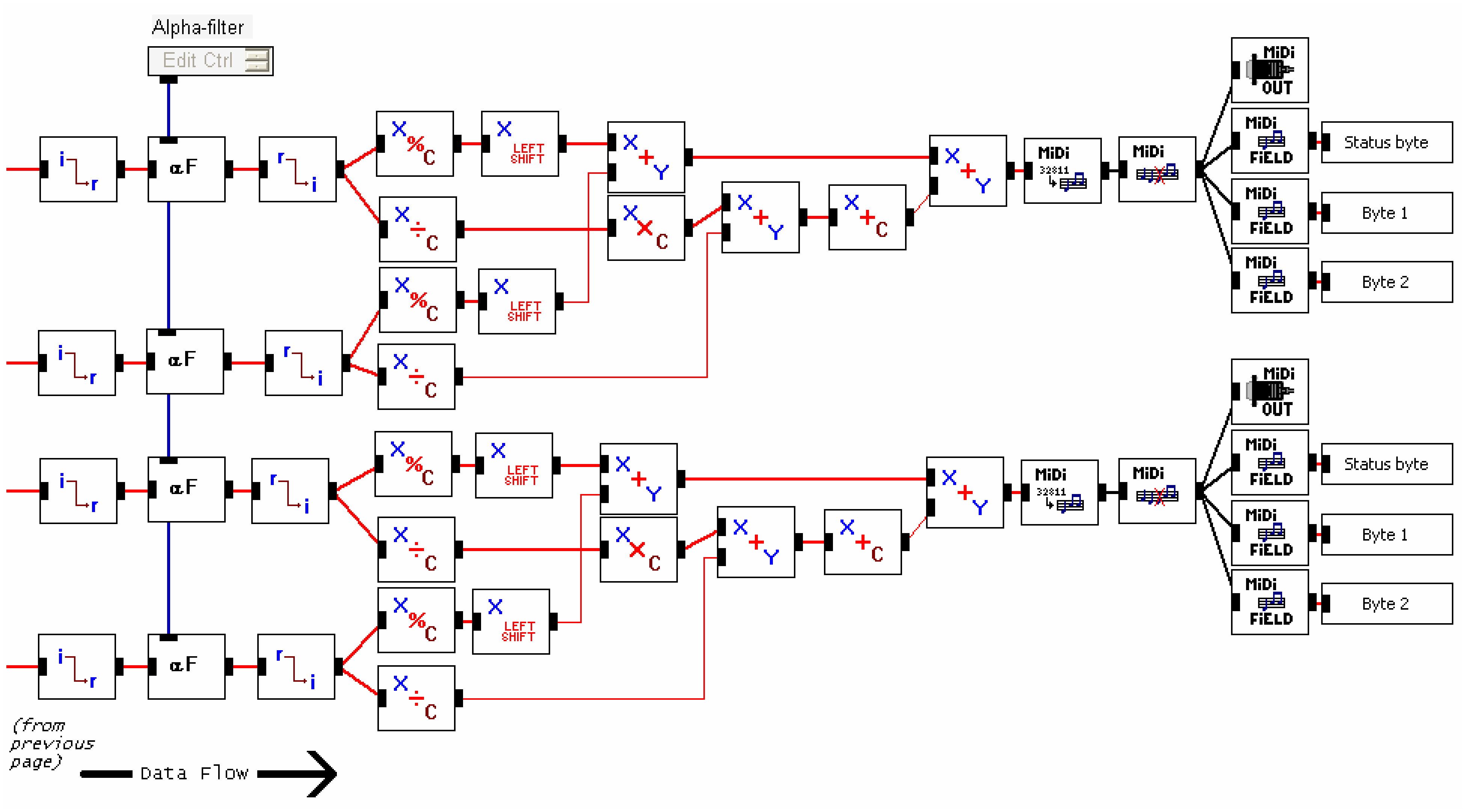


Figure 36. Continued.

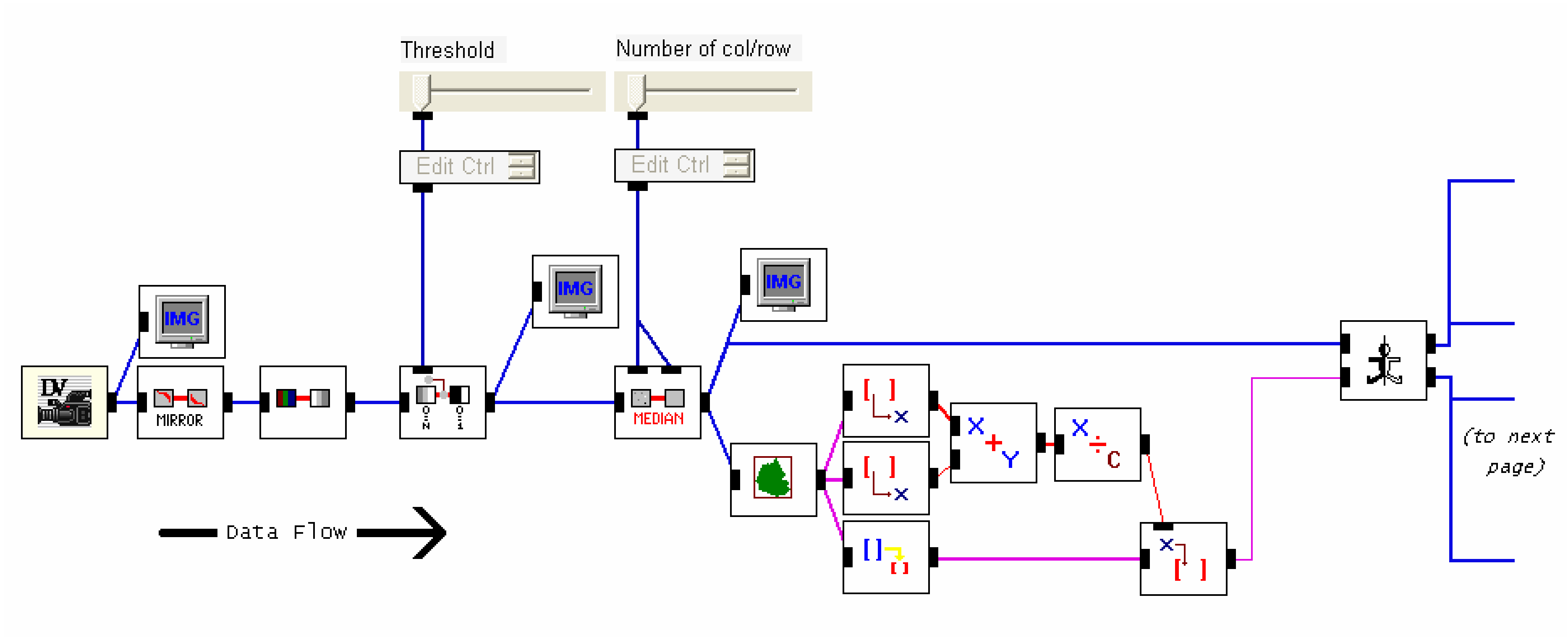


Figure 37. SystemTwo EyesWeb structure.

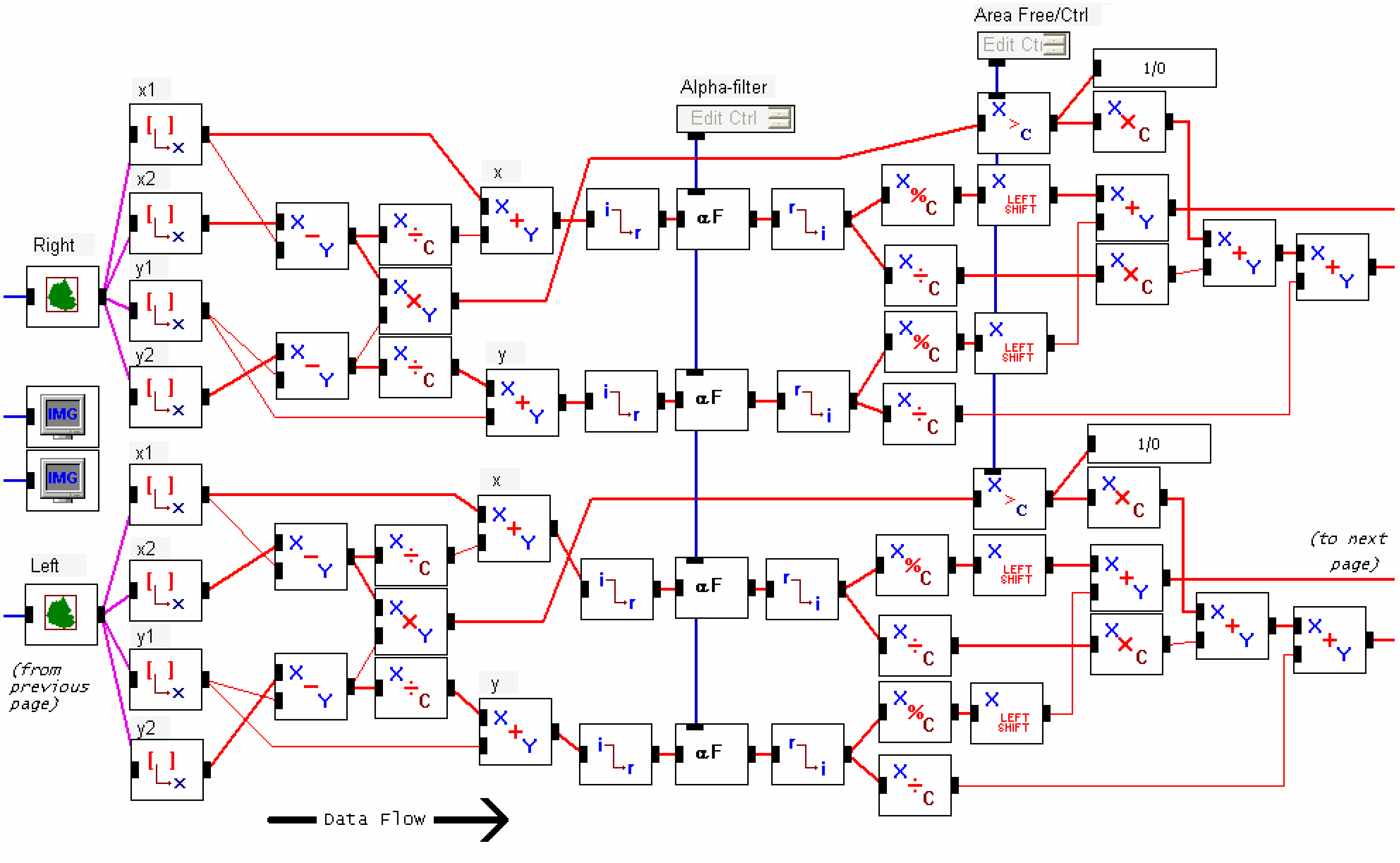


Figure 37. Continued.

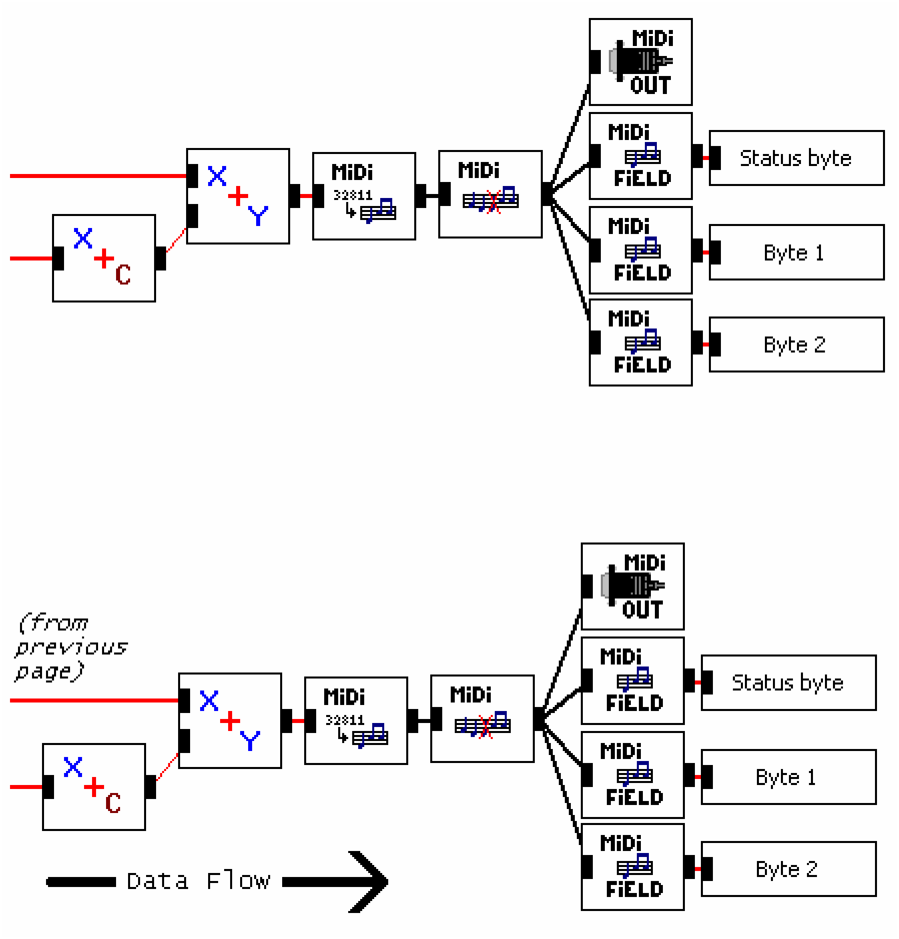


Figure 37. Continued.

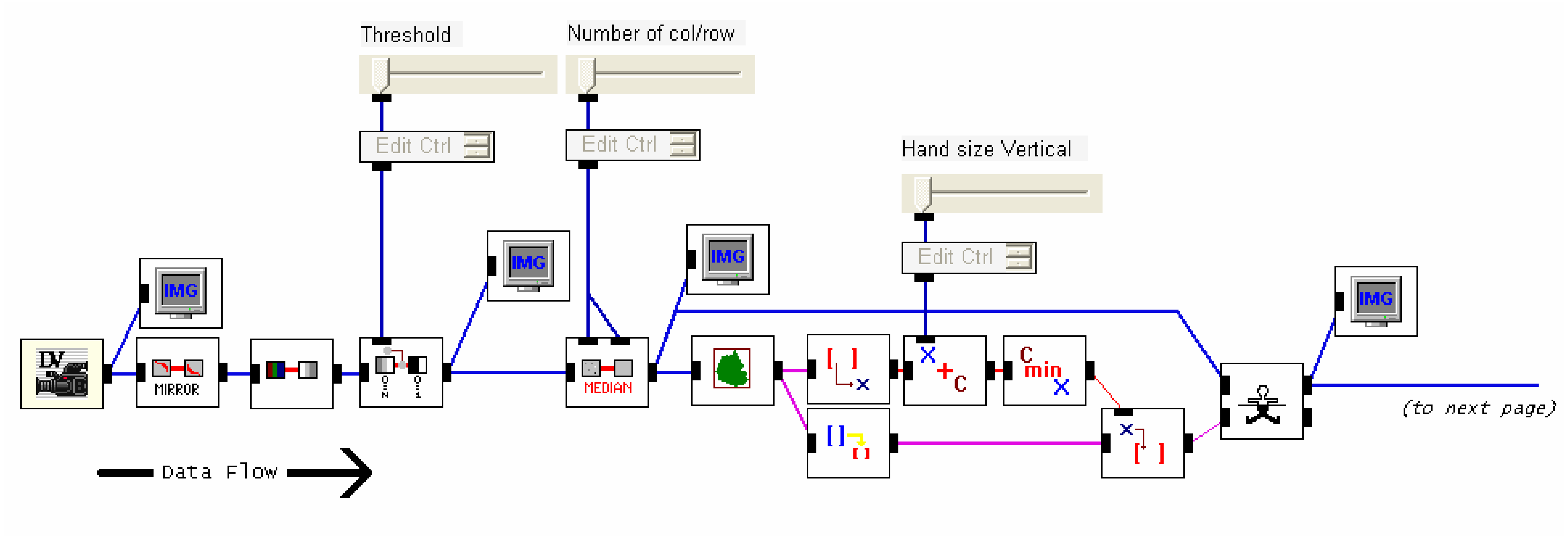


Figure 38. SystemThree EyesWeb structure.

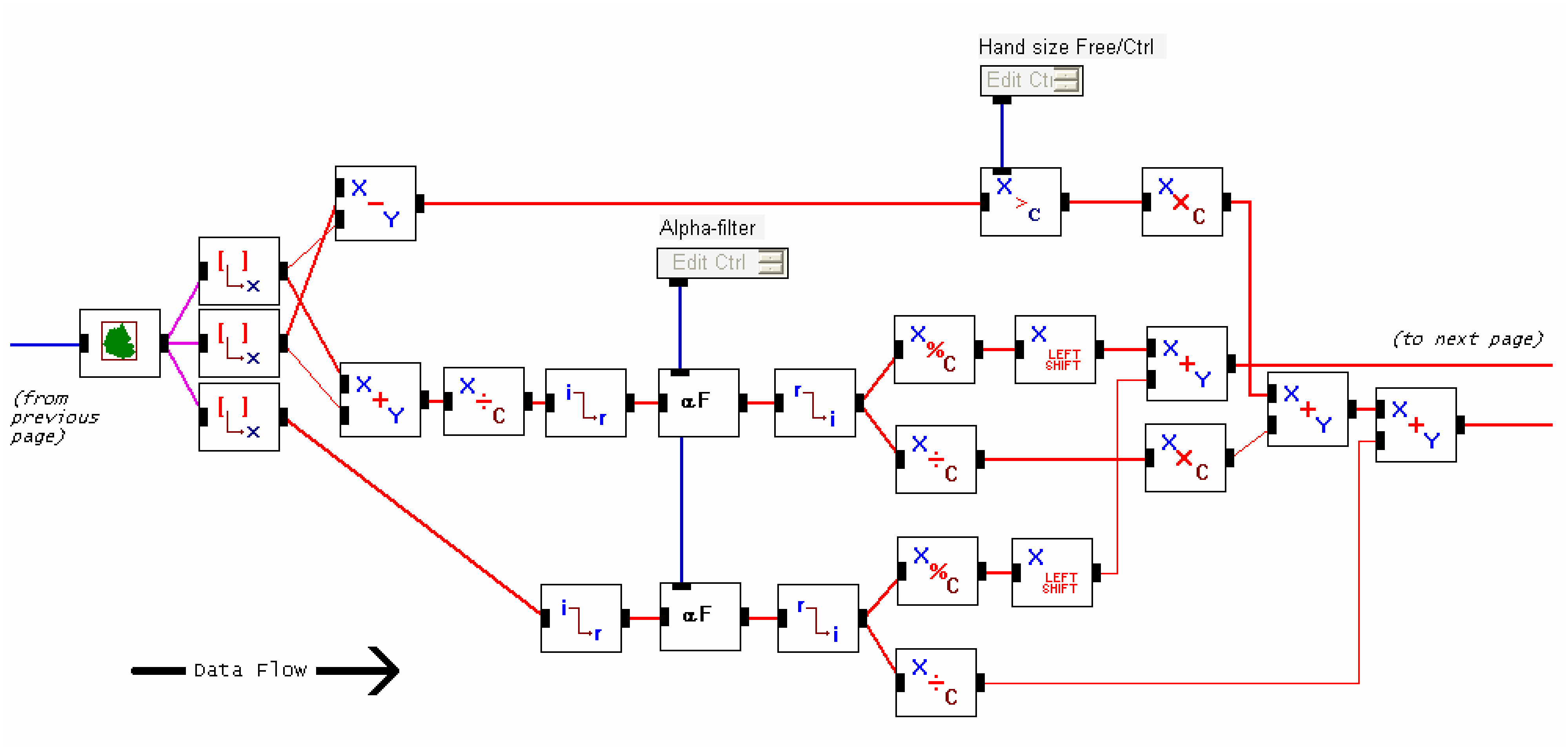


Figure 38. Continued.

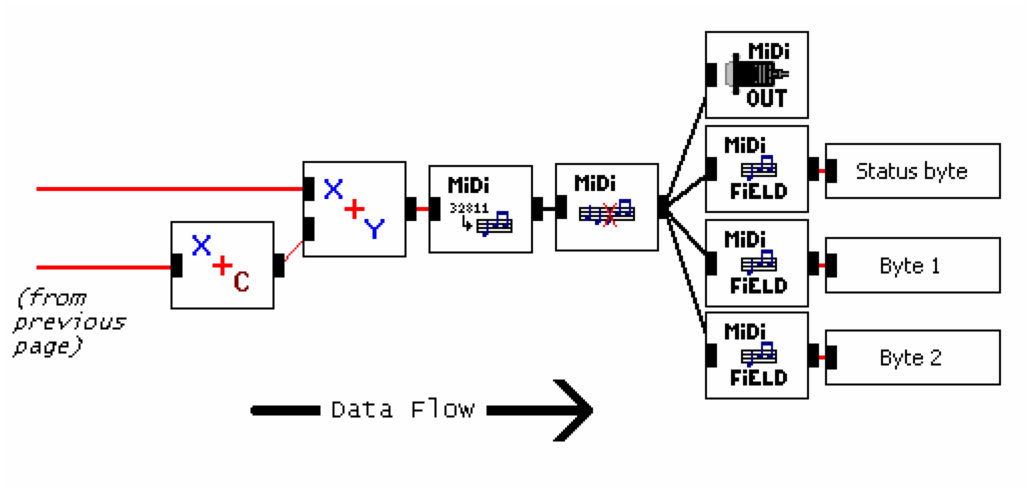


Figure 38. Continued.

APPENDIX D
JAVA SOURCE CODE

System	File name	Classes
SystemOne	"TheMain.java"	public class TheMain extends JFrame
		public class MovingObject
	"TheMidiObject.java"	public class TheMidiObject
		class InputThread01 extends Thread
	"ControlRegion.java"	public abstract class ControlRegion
	"Button1.java"	public class Button1 extends ControlRegion
	"Knob.java"	public class Knob extends ControlRegion
	"RotaryKnob.java"	public class RotaryKnob
	"Keyboard.java"	public class Keyboard
SystemTwo	"TheMain.java"	public class TheMain extends JFrame
		public class MovingObject
		public class TimerThread implements Runnable
		public class VUMeterThread implements Runnable
		class ControlRegion
		class VerticalSlider extends ControlRegion
		class HorizontalSlider extends ControlRegion
		class RotaryKnob extends ControlRegion
		class JogWheel extends ControlRegion
		class Button1 extends ControlRegion
		class Button2 extends Button1
	"TheMidiObject.java"	public class TheMidiObject
		class InputThread01 extends Thread
	"SamplePreparator.java"	public class SamplePreparator extends Frame
SystemThree	"TheMain.java"	public class TheMain extends JFrame
	"TheMidiObject.java"	public class TheMidiObject
		class InputThread01 extends Thread

Table 9. The Java file names and designed classes.

SystemOne
“TheMain.java”

```
import java.awt.*;

import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class TheMain extends JFrame implements ActionListener, ItemListener
{
        int numberOfParameters = 45;
        int leftRKX = 120;
        int rightRKX = 700;
        int leftRightRKY = 150;

        TheMidiObject midiObject;
        JLayeredPane layeredPane;
        JLabel bgLabel, keyboardLabel;
        JLabel [] regionLabels = new JLabel [numberOfParameters];
        JLabel [] additionalLabels = new JLabel [3];
        Choice soundName;
        JButton bStore;

        String imagesPath = "images/";
        String bgFile = imagesPath + "nord03.gif";
        String leftHandFile = imagesPath + "handleft01.gif";
        String rightHandFile = imagesPath + "handright01.gif";

        String knobBKFile = imagesPath + "knobBK.gif";
        String knobFGFile = imagesPath + "knobFG.gif";
        String keyboardFile = imagesPath + "keyboard.gif";
        String noteIndicatorFile = imagesPath + "noteIndicator.gif";
        String latchIndicatorFile = imagesPath + "latchIndicator.gif";

        String indFileRed = imagesPath + "indicatorRed.gif";
        String indFileGreen = imagesPath + "indicatorGreen.gif";
        String dotFile = imagesPath + "dot01.gif";
        String ledFile = imagesPath + "led01.gif";

        String memoryPath = "sounds/";
        String patchMemoryFileName = "1";

        ControlRegion [] region = new ControlRegion [numberOfParameters];
        MovingObject leftHand, rightHand;
        Keyboard keyboard;

        boolean storeMode = false;


        public TheMain ()
        {
                Image bgImage = Toolkit.getDefaultToolkit ().getImage (bgFile);
                Image leftHandImage = Toolkit.getDefaultToolkit ().getImage (leftHandFile);
                Image rightHandImage = Toolkit.getDefaultToolkit ().getImage (rightHandFile);
                Image knobBKImage = Toolkit.getDefaultToolkit ().getImage (knobBKFile);
                Image knobFGImage = Toolkit.getDefaultToolkit ().getImage (knobFGFile);
                Image keyboardImage = Toolkit.getDefaultToolkit ().getImage (keyboardFile);
                Image noteIndicatorImage = Toolkit.getDefaultToolkit ().getImage (noteIndicatorFile);
                Image latchIndicatorImage = Toolkit.getDefaultToolkit ().getImage (latchIndicatorFile);
                Image indImageRed = Toolkit.getDefaultToolkit ().getImage (indFileRed);
                Image indImageGreen = Toolkit.getDefaultToolkit ().getImage (indFileGreen);
                Image dotImage = Toolkit.getDefaultToolkit ().getImage (dotFile);
                Image ledImage = Toolkit.getDefaultToolkit ().getImage (ledFile);

                region [0] = new Knob (45, 105);                        // LFO1 Rate
                region [1] = new Button1 (97, 88, 5, false);       //      Wave
```

```
region [2] = new Button1 (187, 88, 5, false);    //      Dest
region [3] = new Knob (280, 105);                //      Amount
region [4] = new Knob (45, 215);                 //      LFO2 Rate
region [5] = new Button1 (98, 241, 2, true);     //      Arp
region [6] = new Button1 (186, 193, 5, false);   //      Dest
region [7] = new Knob (280, 215);                //      Amount
region [8] = new Knob (45, 325);                 //      M E  A
region [9] = new Knob (125, 325);                //           D
region [10] = new Button1 (185, 315, 4, true);   //      Dest

region [11] = new Knob (280, 325);               //      Amount

region [12] = new Button1 (360, 103, 4, false);  //      OSC1 Wave
region [13] = new Knob (375, 235);               //      FM
region [14] = new Knob (455, 105);               //      OSC2 Semi
region [15] = new Button1 (520, 103, 4, false);  //      Wave
region [16] = new Knob (455, 235);               //      Fine tune
region [17] = new Button1 (500, 232, 2, true);   //      KBD Track
region [18] = new Knob (375, 345);               //      PW
region [19] = new Button1 (417, 316, 4, true);   //      Sync/RM
region [20] = new Knob (535, 345);               //      Mix

region [21] = new Knob (630, 105);               //      Amp  A
region [22] = new Knob (710, 105);               //           D
region [23] = new Knob (790, 105);               //           S
region [24] = new Knob (870, 105);               //           R
region [25] = new Knob (950, 105);               //      Gain
region [26] = new Knob (630, 235);               //      Filt  A
region [27] = new Knob (710, 235);               //           D
region [28] = new Knob (790, 235);               //           S
region [29] = new Knob (870, 235);               //           R
region [30] = new Button1 (916, 222, 5, false);  //      Type
region [31] = new Knob (630, 345);               //      Freq
region [32] = new Knob (710, 345);               //      Resonance
region [33] = new Knob (790, 345);               //      Env Amount
region [34] = new Button1 (832, 348, 2, true);   //      Velocity
region [35] = new Button1 (885, 332, 4, true);   //      KBd Track
region [36] = new Button1 (938, 348, 2, true);   //      Dist

region [37] = new Button1 (20, 405, 2, true);    //      Unison
region [38] = new Button1 (95, 400, 3, false);   //      P/L/M
region [39] = new Knob (200, 415);               //      Portamento
region [40] = new Button1 (180, 450, 2, true);   //      Auto
region [41] = new Button1 (95, 505, 5, false);   //      Oct Shift
region [42] = new Button1 (185, 490, 5, false);  //      MWD
region [43] = new Knob (120, 605);               //      Pitch Bend
region [44] = new Knob (205, 605);               //      Mod Wheel

final ImageIcon bgIcon = new ImageIcon (bgImage);
final ImageIcon leftHandIcon = new ImageIcon (leftHandImage);
final ImageIcon rightHandIcon = new ImageIcon (rightHandImage);
final ImageIcon knobBKIcon = new ImageIcon (knobBKImage);
final ImageIcon knobFGIcon = new ImageIcon (knobFGImage);
final ImageIcon keyboardIcon = new ImageIcon (keyboardImage);
final ImageIcon noteIndicatorIcon = new ImageIcon (noteIndicatorImage);
final ImageIcon latchIndicatorIcon = new ImageIcon (latchIndicatorImage);
final ImageIcon indIconRed = new ImageIcon (indImageRed);
final ImageIcon indIconGreen = new ImageIcon (indImageGreen);
final ImageIcon dotIcon = new ImageIcon (dotImage);
final ImageIcon ledIcon = new ImageIcon (ledImage);

bgLabel = new JLabel (bgIcon);
bgLabel.setBounds (0, 0, bgIcon.getIconWidth (), bgIcon.getIconHeight ());

leftHand = new MovingObject (new JLabel (leftHandIcon), new JLabel (knobBKIcon), new JLabel (knobFGIcon),
                new JLabel (indIconGreen), new JLabel ("NORMAL"), leftRKX, leftRightRKY, - 21);
rightHand = new MovingObject (new JLabel (rightHandIcon), new JLabel (knobBKIcon), new JLabel (knobFGIcon),
                new JLabel (indIconRed), new JLabel ("NORMAL"), rightRKX, leftRightRKY, - 11);
```

```
keyboard = new Keyboard (270, 520, 565, 125);
keyboard.setLabel (noteIndicatorIcon, latchIndicatorIcon);

for (int i = 0 ; i < keyboard.noteLabel.length ; i++)
        keyboard.noteLabel [i].setBounds (keyboard.x1 + (i * 22) + 3, keyboard.y1 + 40,
                        noteIndicatorIcon.getIconWidth (), noteIndicatorIcon.getIconHeight ());

keyboard.latchIndicatorLabel.setBounds (keyboard.x1, keyboard.y1 + 126,
                latchIndicatorIcon.getIconWidth (), latchIndicatorIcon.getIconHeight ());

keyboardLabel = new JLabel (keyboardIcon);
keyboardLabel.setBounds (keyboard.x1, keyboard.y1, keyboardIcon.getIconWidth (), keyboardIcon.getIconHeight ());

leftHand.handLabel.setBounds (- 100, - 100, leftHandIcon.getIconWidth (), leftHandIcon.getIconHeight ());
rightHand.handLabel.setBounds (- 100, - 100, rightHandIcon.getIconWidth (), rightHandIcon.getIconHeight ());

leftHand.knobBKLabel.setBounds (leftRKX, leftRightRKY, knobBKIcon.getIconWidth (), knobBKIcon.getIconHeight ());
rightHand.knobBKLabel.setBounds (rightRKX, leftRightRKY, knobBKIcon.getIconWidth (), knobBKIcon.getIconHeight ());
leftHand.knobFGLabel.setBounds (- 100, - 100, knobFGIcon.getIconWidth (), knobFGIcon.getIconHeight ());
rightHand.knobFGLabel.setBounds (- 100, - 100, knobFGIcon.getIconWidth (), knobFGIcon.getIconHeight ());

leftHand.indicatorLabel.setBounds (region [0].left - 28, region [0].top - 28,
                indIconGreen.getIconWidth (), indIconGreen.getIconHeight ());
rightHand.indicatorLabel.setBounds (region [0].left - 28, region [0].top - 28,
                indIconRed.getIconWidth (), indIconRed.getIconHeight ());

regionLabels [0] = new JLabel (dotIcon);
regionLabels [1] = new JLabel (ledIcon);
regionLabels [2] = new JLabel (ledIcon);
regionLabels [3] = new JLabel (dotIcon);
regionLabels [4] = new JLabel (dotIcon);
regionLabels [5] = new JLabel (ledIcon);
regionLabels [6] = new JLabel (ledIcon);
regionLabels [7] = new JLabel (dotIcon);
regionLabels [8] = new JLabel (dotIcon);
regionLabels [9] = new JLabel (dotIcon);
regionLabels [10] = new JLabel (ledIcon);
regionLabels [11] = new JLabel (dotIcon);
regionLabels [12] = new JLabel (ledIcon);
regionLabels [13] = new JLabel (dotIcon);
regionLabels [14] = new JLabel (dotIcon);
regionLabels [15] = new JLabel (ledIcon);
regionLabels [16] = new JLabel (dotIcon);
regionLabels [17] = new JLabel (ledIcon);
regionLabels [18] = new JLabel (dotIcon);
regionLabels [19] = new JLabel (ledIcon);

for (int i = 20 ; i <= 29 ; i++)
        regionLabels [i] = new JLabel (dotIcon);

regionLabels [30] = new JLabel (ledIcon);
regionLabels [31] = new JLabel (dotIcon);
regionLabels [32] = new JLabel (dotIcon);
regionLabels [33] = new JLabel (dotIcon);
regionLabels [34] = new JLabel (ledIcon);
regionLabels [35] = new JLabel (ledIcon);
regionLabels [36] = new JLabel (ledIcon);

regionLabels [37] = new JLabel (ledIcon);
regionLabels [38] = new JLabel (ledIcon);
regionLabels [39] = new JLabel (dotIcon);
regionLabels [40] = new JLabel (ledIcon);
regionLabels [41] = new JLabel (ledIcon);
regionLabels [42] = new JLabel (ledIcon);
regionLabels [43] = new JLabel (dotIcon);

regionLabels [44] = new JLabel (dotIcon);
```

```
for (int i = 0 ; i < regionLabels.length ; i++)
{
        regionLabels [i].setBounds (region [i].xPos, region [i].yPos, 7, 7);
        if (region [i] instanceof Button1 && ((Button1) (region [i])).onOff)
                regionLabels [i].setVisible (false);
}

additionalLabels [0] = new JLabel (ledIcon);
additionalLabels [0].setBounds (455, 70, 7, 7);

additionalLabels [1] = new JLabel (ledIcon);
additionalLabels [1].setBounds (455, 200, 7, 7);

additionalLabels [2] = new JLabel (ledIcon);
additionalLabels [2].setBounds (120, 570, 7, 7);

leftHand.trackLabel.setBounds (294, 1, 70, 19);
leftHand.trackLabel.setFont (new Font ("arial", Font.BOLD, 10));
leftHand.trackLabel.setForeground (Color.white);
leftHand.trackLabel.setHorizontalAlignment (JLabel.CENTER);

rightHand.trackLabel.setBounds (554, 1, 70, 19);
rightHand.trackLabel.setFont (new Font ("arial", Font.BOLD, 10));
rightHand.trackLabel.setForeground (Color.white);
rightHand.trackLabel.setHorizontalAlignment (JLabel.CENTER);

soundName = new Choice ();
soundName.setBounds (875, 675, 60, 20);
for (int i = 0 ; i < 10 ; i++)
        soundName.addItem (i + 1 + "");

bStore = new JButton ("Store");
bStore.setBounds (935, 675, 60, 20);
bStore.setFont (new Font ("arial", Font.BOLD, 10));

layeredPane = new JLayeredPane ();
layeredPane.setPreferredSize (new Dimension (bgIcon.getIconWidth (), bgIcon.getIconHeight ()));

layeredPane.add (bgLabel, new Integer (0));

for (int i = 0 ; i < keyboard.noteLabel.length ; i++)
        layeredPane.add (keyboard.noteLabel [i], new Integer (1));

layeredPane.add (keyboard.latchIndicatorLabel, new Integer (1));

layeredPane.add (keyboardLabel, new Integer (1));

layeredPane.add (leftHand.handLabel, new Integer (2));
layeredPane.add (rightHand.handLabel, new Integer (2));
layeredPane.add (leftHand.knobFGLabel, new Integer (1));
layeredPane.add (rightHand.knobFGLabel, new Integer (1));
layeredPane.add (leftHand.knobBKLabel, new Integer (1));
layeredPane.add (rightHand.knobBKLabel, new Integer (1));
layeredPane.add (leftHand.indicatorLabel, new Integer (1));
layeredPane.add (rightHand.indicatorLabel, new Integer (1));

for (int i = 0 ; i < regionLabels.length ; i++)
        layeredPane.add (regionLabels [i], new Integer (1));

layeredPane.add (additionalLabels [0], new Integer (1));
layeredPane.add (additionalLabels [1], new Integer (1));
layeredPane.add (additionalLabels [2], new Integer (1));
layeredPane.add (leftHand.trackLabel, new Integer (1));
layeredPane.add (rightHand.trackLabel, new Integer (1));
layeredPane.add (soundName, new Integer (1));
layeredPane.add (bStore, new Integer (1));
```

```
		getContentPane ().add (layeredPane, BorderLayout.CENTER);
		setSize (1020, 750);
		setTitle ("SYSTEMONE");
		setVisible (true);

		for (int i = 0 ; i < keyboard.noteLabel.length ; i++)
			keyboard.noteLabel [i].setVisible (false);

		keyboard.latchIndicatorLabel.setVisible (false);

		midiObject = new TheMidiObject (this);

		layeredPane.addMouseMotionListener (new MouseMotionAdapter ()
		{
			public void mouseMoved (MouseEvent e)
			{
				int x = e.getX ();
				int y = e.getY () - 40;
				rightHand.midiIn (x, y);
			}
			public void mouseDragged (MouseEvent e)
			{
				int x = e.getX ();
				int y = e.getY () - 40;
				leftHand.midiIn (x, y);
			}
		}
		);
		soundName.addItemListener (this);
		bStore.addActionListener (this);
		addWindowListener (new WindowAdapter ()
		{
			public void windowClosing (WindowEvent e)
			{
				for (int i = 0 ; i < 128 ; i++)
					midiObject.midiOutAction (0, i, (byte) 0);

				midiObject.midiOutAction (2, 0, (byte) 64);
				midiObject.midiportClose ();
				System.exit (0);
			}
		}
		);
	}


	public class MovingObject
	{
		JLabel handLabel, knobBKLabel, knobFGLabel, indicatorLabel, trackLabel;
		boolean insideRegion = false;
		int regionSelection = 0;
		RotaryKnob rotaryKnob;
		int prevNote = -1;
		int fingerPos;

		public MovingObject (JLabel hLab, JLabel kbLab, JLabel kfLab, JLabel iLab, JLabel tLab, int x, int y, int f)
		{
			handLabel = hLab;
			knobBKLabel = kbLab;
			knobFGLabel = kfLab;
			indicatorLabel = iLab;
			trackLabel = tLab;
			rotaryKnob = new RotaryKnob (x, y, 200, 200);
			rotaryKnob.setLabel (knobFGLabel);
			fingerPos = f;
		}

		boolean sortOutKeyboard = false;

		public void midiIn (int x, int y)
		{
			handLabel.setLocation (x + fingerPos, y);

			if (keyboard.equals (x, y))
			{
				int tNote = keyboard.getNoteNumber (x, y);
				if (tNote != prevNote)
				{
					if (!keyboard.latch)
					{
						if (keyboard.removeFromVector (prevNote))
						{
							if (prevNote >= 0)
								midiObject.midiOutAction (0, prevNote + 48, (byte) 0);
						}
						if (keyboard.addToVector (tNote))
						{
							midiObject.midiOutAction (0, tNote + 48, (byte) 90);
						}
					}
					else
					{
						if (keyboard.addToVector (tNote))
						{
							int temp = keyboard.checkSize ();
							if (temp >= 0)
								midiObject.midiOutAction (0, temp + 48, (byte) 0);
							midiObject.midiOutAction (0, tNote + 48, (byte) 90);
						}
						else if (keyboard.removeFromVector (tNote))
						{
							if (tNote >= 0)
								midiObject.midiOutAction (0, tNote + 48, (byte) 0);
						}
					}
				}
				prevNote = tNote;
				sortOutKeyboard = true;
			}
			else if (keyboard.equalsLatch (x, y))
			{
				if (!insideRegion)
				{
					keyboard.changeLatch ();
					insideRegion = true;
				}
				if (!keyboard.latch)
				{
					for (int i = 0 ; i < keyboard.notes.size () ; i++)
					{
						int temp = keyboard.getNoteNumber2 (i);
						midiObject.midiOutAction (0, temp + 48, (byte) 0);
						keyboard.noteLabel [temp].setVisible (false);
					}
					keyboard.notes.removeAllElements ();
				}
			}
```

```
else if (sortOutKeyboard)
{
    if (!keyboard.latch)
    {
        if (keyboard.removeFromVector (prevNote))
        {
            if (prevNote >= 0)
                midiObject.midiOutAction (0, prevNote + 48, (byte) 0);
        }
    }
    prevNote = -1;
    sortOutKeyboard = false;
}
else if (y <= 21)
{
    if (!insideRegion)
    {
        insideRegion = true;
        int temp = -1;

        if (x >= 42 && x <= 293)
            temp = (x - 42) / 21;
        else if (x >= 365 && x <= 553)
            temp = (x - 365) / 21 + 12;
        else if (x >= 625 && x <= 960)
            temp = (x - 625) / 21 + 21;
        else if (x >= 294 && x <= 364)
            temp = -10;
        else if (x >= 554 && x <= 624)
            temp = -10;

        if (temp >= 0)
        {
            if (region [temp] instanceof Button1)
            {
                region [temp].changeValue ();
                changeButton (temp);
            }
            else
            {
                regionSelection = temp;
                rotaryKnob.value = region [regionSelection].value;
                rotaryKnob.rePaint ();
                indicatorLabel.setLocation (region [regionSelection].left - 28, region [regionSelection].top - 28);
            }
        }
        else if (temp == -10)
        {
            if (trackLabel.getText ().equals ("NORMAL"))
            {
                trackLabel.setText ("SLOW");
                rotaryKnob.changeTrackValue (false);
            }
            else
            {
                trackLabel.setText ("NORMAL");
                rotaryKnob.changeTrackValue (true);
            }
        }
    }
}
else if (x <= 22 && y >= 440 && y < 629)
{
    if (!insideRegion)
    {
        insideRegion = true;
        int temp = (y - 440) / 21 + 37;
```

```
            if (temp == 41)
                region [41].changeValue (region [41].value + 1);
            else if (temp == 42)
                region [41].changeValue (region [41].value - 1);

            if (temp > 41)
                temp--;

            if (region [temp] instanceof Button1)
            {
                if (temp != 41)
                    region [temp].changeValue ();
                changeButton (temp);
            }
            else
            {
                regionSelection = temp;
                rotaryKnob.value = region [regionSelection].value;
                rotaryKnob.rePaint ();
                indicatorLabel.setLocation (region [regionSelection].left - 28, region [regionSelection].top - 28);
            }
        }
    }
    else if (rotaryKnob.equals (x, y))
    {
        if (!insideRegion)
        {
            insideRegion = true;
            rotaryKnob.setFirst (x, y);
        }
        else
        {
            rotaryKnob.changeValue (x, y);
            changeKnob ();
        }
    }
    else if (keyboard.equalsMover (x, y))
    {
        keyboardLabel.setLocation (keyboard.x1, keyboard.y1);
    }
    else if (rotaryKnob.equalsMover (x, y))
    {
        knobBKLabel.setLocation (rotaryKnob.x1, rotaryKnob.y1);
    }
    else
        insideRegion = false;
}

byte arpeggiatorByte;

public void sortOutArpeggiator (int t)
{
    if (t == 5)
        if (region [5].value == 0)
        {
            region [6].changeValue (2);
            arpeggiatorByte = 8;
        }
        else
        {
            region [6].changeValue (0);
            arpeggiatorByte = 0;
        }
    else
    {
        if (region [5].value == 0)
        {
            if (region [6].value <= 2)
            {
                region [6].changeValue (2);
                arpeggiatorByte = 7;
            }
            else
                arpeggiatorByte = (byte) region [6].value;
        }
        else
        {
            byte [] b = {0, 2, 1, 5, 6};
            arpeggiatorByte = b [region [6].value];
        }
    }
}


public byte getValidByte (int t, int v)
{
    if (t == 10)
    {
        byte [] b = {3, 0, 2, 1};
        return b [v];
    }
    if (t == 15)
    {
        byte [] b = {3, 0, 1, 2};
        return b [v];
    }
    if (t == 30)
    {
        byte [] b = {0, 4, 1, 3, 2};
        return b [v];
    }
    if (t == 5 || t == 6)
        return arpeggiatorByte;
    return (byte) v;
}


public void changeButton (int temp)
{
    if (temp == 5 || temp == 6)
    {
        sortOutArpeggiator (temp);
        regionLabels [5].setLocation (region [5].xPos, region [5].yPos);
        regionLabels [6].setLocation (region [6].xPos, region [6].yPos);
    }
    else if (temp == 41)
        regionLabels [41].setLocation (648 - region [41].yPos, region [41].xPos + 410);
    else
        regionLabels [temp].setLocation (region [temp].xPos, region [temp].yPos);

    byte tempByte = getValidByte (temp, region [temp].value);
    midiObject.midiOutAction (1, temp, tempByte);

    if (((Button1) (region [temp])).onOff)
    {
        if (region [temp].value == 0)
            regionLabels [temp].setVisible (false);
        else
            regionLabels [temp].setVisible (true);
    }
}
```

```
    public void changeKnob ()
    {
        region [regionSelection].changeValue (rotaryKnob.value);
        regionLabels [regionSelection].setLocation (region [regionSelection].xPos, region [regionSelection].yPos);

        if (regionSelection == 14)
        {
            int t = region [14].value * 120 / 127;
            additionalLabels [0].setVisible (t % 12 == 0 ? true:
            false);
            midiObject.midiOutAction (1, regionSelection, (byte) t);
        }
        else if (regionSelection == 16)
        {
            additionalLabels [1].setVisible (region [regionSelection].value == 64 ? true:
            false);
            midiObject.midiOutAction (1, regionSelection, (byte) region [regionSelection].value);
        }
        else if (regionSelection == 43)
        {
            additionalLabels [2].setVisible (region [regionSelection].value == 64 ? true:
            false);
            midiObject.midiOutAction (2, 0, (byte) region [regionSelection].value);
        }
        else
            midiObject.midiOutAction (1, regionSelection, (byte) region [regionSelection].value);
    }
}


public void loadSettings (String fileName)
{
    int [] parameterValues = new int [numberOfParameters + 2];

    try
    {
        String f = memoryPath + fileName + ".txt";
        BufferedReader inputFile = new BufferedReader (new FileReader (f));

        String s;
        int c = 0;

        while ((s = inputFile.readLine ()) != null)
            parameterValues [c++] = Integer.parseInt (s);

        inputFile.close ();

        patchMemoryFileName = fileName;
    }
    catch (IOException e)
    {
        System.err.println (e);
    }

    int initLeft = parameterValues [parameterValues.length - 2];
    int initRight = parameterValues [parameterValues.length - 1];
```

```
		for (int i = 0 ; i < parameterValues.length - 2 ; i++)
		{
			if (region [i] instanceof Button1)
			{
				region [i].changeValue (parameterValues [i]);
				leftHand.changeButton (i);
			}
			else
			{
				leftHand.regionSelection = i;
				leftHand.rotaryKnob.value = parameterValues [i];
				leftHand.changeKnob ();
			}
		}

		leftHand.regionSelection = initLeft;
		leftHand.rotaryKnob.value = region [initLeft].value;
		leftHand.rotaryKnob.rePaint ();
		leftHand.indicatorLabel.setLocation (region [initLeft].left - 28, region [initLeft].top - 28);
		rightHand.regionSelection = initRight;
		rightHand.rotaryKnob.value = region [initRight].value;
		rightHand.rotaryKnob.rePaint ();
		rightHand.indicatorLabel.setLocation (region [initRight].left - 28, region [initRight].top - 28);
	}


	public void saveSettings (String fileName)
	{
		try
		{
			String f = memoryPath + fileName + ".txt";
			PrintWriter outputFile = new PrintWriter (new FileWriter (f));

			for (int i = 0 ; i < numberOfParameters ; i++)
				outputFile.println (region [i].value);

			outputFile.println (leftHand.regionSelection);
			outputFile.println (rightHand.regionSelection);

			outputFile.close ();

			patchMemoryFileName = fileName;
		}
		catch (IOException e)
		{
			System.err.println (e);
		}
	}


	public void actionPerformed (ActionEvent e)
	{
		if (e.getSource () == bStore)
			if (storeMode)
			{
				saveSettings (soundName.getSelectedItem ());
				bStore.setBackground (Color.lightGray);
				bStore.setForeground (Color.black);
				storeMode = false;
			}
			else
			{
				bStore.setBackground (Color.red);
				bStore.setForeground (Color.white);
				storeMode = true;
			}
	}
```

```
	public void itemStateChanged (ItemEvent e)
	{
		if (e.getSource () == soundName)
			if (!storeMode)
				loadSettings (soundName.getSelectedItem ());
	}


	public static void main (String [] args)
	{
		new TheMain ();

	}
}
```

"TheMidiObject.java"

```
// TheMidiObject

// JavaMIDI by: Robert Marsanyi
// Copyright CagEnt Technologies, Inc
// All Rights Reserved

import jmidi.*;

public class TheMidiObject
{
	TheMain m;
	static final int BUFSIZE = 3;
	static byte [] buffer = new byte [BUFSIZE];

	static MidiPort midiport;
	InputThread01 inputThread;

	public TheMidiObject (TheMain m)
	{
		this.m = m;

		int inDevice = 1;
		int outDevice = 1;

		try
		{
			midiport = new MidiPort (inDevice, outDevice);
			tryMidiportOpen ();

			inputThread = new InputThread01 (midiport, this);
			inputThread.start ();
		}
		catch (Exception e)
		{
			System.err.println ("WHOOPS! error! TheMidiObject " + e.getMessage ());
		}
	}
```

```
    void tryMidiportOpen ()
    {
        try
        {
            midiport.open ();
        }
        catch (MidiPortException e)
        {
            System.err.println ("tryMidiportOpen " + e);
            tryMidiportOpen ();
        }
    }


    byte [] midiController = {19, 20, 21, 22, 23, 24, 24, 25, 26, 27, 28, 29, 30, 70, 78, 31, 33, 34, 79, 35, 8, 73,
        36, 37, 72, 7, 38, 39, 40, 41, 44, 74, 42, 43, 45, 46, 80, 16, 15, 5, 65, 17, 18, 0, 1};

    public void midiOutAction (int messageType, int p, byte v)
    {
        try
        {
            if (messageType == 0)
                midiport.writeShortMessage ((byte) 0x90, (byte) p, v);
            else if (messageType == 1)
                midiport.writeShortMessage ((byte) 0xB0, midiController [p], v);
            else if (messageType == 2)
                midiport.writeShortMessage ((byte) 0xE0, (byte) 0, v);
        }
        catch (Exception e)
        {
            System.err.println ("WHOOPS! error! midiOutAction" + e.getMessage ());
        }
    }


    public void midiportClose ()
    {
        try
        {
            System.out.println ("Closing port");
            midiport.close ();
            System.out.println ("Done!");
        }
        catch (Exception e)
        {
            System.err.println ("WHOOPS! error! midiportClose! " + e.getMessage ());
        }
    }
}


class InputThread01 extends Thread
{
    MidiPort midiport;
    TheMidiObject t;

    public InputThread01 (MidiPort aPort, TheMidiObject d)
    {
        t = d;
        this.midiport = aPort;
    }


    public void run ()
    {
        int nBytes;
        boolean done = false;
        byte [] buffer = TheMidiObject.buffer;
        MidiPortMessage msg;

        try
        {
            msg = new MidiPortMessage (TheMidiObject.BUFSIZE);
            buffer = msg.messageData;

            System.out.println ("Opening port");

            t.m.loadSettings (t.m.patchMemoryFileName);

            while (!done)
            {
                try
                {
                    while (midiport.messagesWaiting () == 0)
                    {
                        yield ();
                    }
                    nBytes = midiport.readMessage (msg);

                    int statusByte = 128 + buffer [0];
                    int statusByteX = (statusByte % 16) / 4;
                    int statusByteY = statusByte % 4;

                    int x = (int) ((statusByteX * 128 + buffer [1]) * 3.2) - 64;
                    int y = (int) ((statusByteY * 128 + buffer [2]) * 3.2) - 64;

                    if (statusByte < 16)
                        t.m.leftHand.midiIn (x, y);
                    else
                        t.m.rightHand.midiIn (x, y);
                }
                catch (MidiPortException e)
                {
                    System.err.println ("MidiPort Exception (probably overran buffer) - resetting.");
                    System.err.println ("*" + e + "*");
                    midiport.resetInput ();
                }
            }
            System.out.println ("Checking message queue (should be 0 left) = " + midiport.messagesWaiting ());
            t.midiportClose ();
        }
        catch (Exception e)
        {
            System.err.println ("WHOOPS! error! run " + e.getMessage ());
        }

    }
}
```

“ControlRegion.java”

```
public abstract class ControlRegion

{
    int left, top;
    int value = 0;
    int xPos, yPos;
    abstract void changeValue ();
    abstract void changeValue (int i);
}
```

“Button1.java”

```
public class Button1 extends ControlRegion

{
        int maxValue;
        boolean onOff;

        public Button1 (int x, int y, int r, boolean b)
        {
                left = x;
                top = y;
                maxValue = r - 1;
                onOff = b;

                xPos = x;
                value = -1;
                changeValue ();
        }


        void changeValue ()
        {
                value++;

                if (value > maxValue)
                        value = 0;
                changeYPos ();
        }


        public void changeYPos ()
        {
                yPos = (maxValue - value) * 12 + top;
        }


        void changeValue (int i)
        {
                value = Math.max (i, 0);
                value = Math.min (value, maxValue);
                changeYPos ();

        }
}
```

“Knob.java”

```
public class Knob extends ControlRegion

{
        public Knob (int x, int y)
        {
                left = x;
                top = y;
                changeValue (0);
        }


        void changeValue ()
        {
        }

        void changeValue (int p)
        {
                value = p;

                double t = p;
                t = 127 - t;
                t *= 0.042;
                t -= 1.1;

                xPos = left + (int) (Math.cos (t) * 13);
                yPos = top - (int) (Math.sin (t) * 13);

        }
}
```

“RotaryKnob.java”

```
import javax.swing.*;


public class RotaryKnob
{
        int x1, x2, y1, y2, value, trackValue;
        int radius = 100;
        int TVALUE1 = 30, TVALUE2 = 5;
        JLabel label;

        public RotaryKnob (int x, int y, int w, int h)
        {
                x1 = x;
                y1 = y;
                x2 = x + w;
                y2 = y + h;
                value = 0;
                trackValue = TVALUE1;
        }


        public void setLabel (JLabel lab)
        {
                label = lab;
        }


        public void changeValue (int x, int y)
        {
                double newPos = Math.atan2 (y - y1 - radius, x - x1 - radius);

                double diffPos = newPos - prevPos;

                if (prevPos < -1.5708 && newPos > 1.5708)
                        diffPos = newPos - 6.283185 - prevPos;
                else if (prevPos > 1.5708 && newPos < -1.5708)
                        diffPos = newPos + 6.283185 - prevPos;

                double t = diffPos * trackValue;

                if (t >= 1 || t <= -1)
                        prevPos = newPos;

                value += (int) t;
        if (value < 0)
                value = 0;
        else if (value > 127)
                value = 127;

        rePaint ();
}


int xPos, yPos;

public void rePaint ()
{
        double t = value;
        t *= 0.038;
        t -= 4;

        xPos = x1 + 80 + (int) (Math.cos (t) * radius);
        yPos = y1 + 80 + (int) (Math.sin (t) * radius);

        label.setLocation (xPos, yPos);
}


double prevPos = 0;

public void setFirst (int x, int y)
{
        prevPos = Math.atan2 (y - y1 - radius, x - x1 - radius);
}


public void changeTrackValue (boolean b)
{
        trackValue = b ? TVALUE1:
        TVALUE2;
}


public boolean equals (int x, int y)
{
        if (x >= x1 && x < x2 && y >= y1 && y < y2)
        {
                double a = Math.pow (x - x1 - radius, 2);
                double b = Math.pow (y - y1 - radius, 2);

                if (Math.sqrt (a + b) <= radius)
                        return true;
                return false;
        }
        return false;
}


public boolean equalsMover (int x, int y)
{
        if (x >= x2 && x < x2 + 30 && y >= y2 - 30 && y < y2)
        {
                int tX = x - x2 - 15;
                int tY = y - y2 + 15;
                x1 += tX;
                x2 += tX;
                xPos += tX;
                y1 += tY;
                y2 += tY;
                yPos += tY;
                label.setLocation (xPos, yPos);
                return true;
        }
                return false;

        }
}
```

“Keyboard.java”

```
import javax.swing.*;

import java.util.Vector;

public class Keyboard
{
        int x1, x2, y1, y2;
        JLabel [] noteLabel = new JLabel [25];
        JLabel latchIndicatorLabel;
        Vector notes;

        public Keyboard (int x, int y, int w, int h)
        {
                x1 = x;
                y1 = y;
                x2 = x + w;
                y2 = y + h;
                notes = new Vector ();
        }


        public void setLabel (ImageIcon iIcon, ImageIcon latchIcon)
        {
                for (int i = 0 ; i < noteLabel.length ; i++)
                        noteLabel [i] = new JLabel (iIcon);

                latchIndicatorLabel = new JLabel (latchIcon);
        }


        public boolean equals (int x, int y)
        {
                if (x >= x1 && x < x2 && y >= y1 && y < y2)
                        return true;
                return false;
        }

        public boolean equalsMover (int x, int y)
        {
                if (x >= x2 && x < x2 + 30 && y >= y2 && y < y2 + 30)
                {
                        int tX = x - x2 - 15;
                        int tY = y - y2 - 15;
                        x1 += tX;
                        x2 += tX;
                        y1 += tY;
                        y2 += tY;

                        for (int i = 0 ; i < noteLabel.length ; i++)
                                noteLabel [i].setLocation (x1 + (i * 22) + 3, y1 + 40);

                        latchIndicatorLabel.setLocation (x1, y1 + 126);

                        return true;
                }
                return false;
        }
```

```
public boolean equalsLatch (int x, int y)
{
	if (y >= y2 && y < y2 + 30 && x >= x1 && x < x2)
		return true;
	return false;
}


public void changeLatch ()
{
	latch = !latch;
	latchIndicatorLabel.setVisible (latch);
}


int [] whiteNotes = {0, 2, 4, 5, 7, 9, 11, 12, 14, 16, 17, 19, 21, 23, 24};
boolean latch = false;

public int getNoteNumber (int x, int y)
{

	int tX = x - x1;
	int tY = y - y1;
	int tNote;
	if (tY <= 75)
	{
		tNote = tX / 22;
		if (tNote > 24)
			tNote = 24;
	}
	else
	{
		tNote = (int) (tX * .0265);
		tNote = whiteNotes [tNote];
	}

	return tNote;
}


public boolean addToVector (int t)
{
	Integer note = new Integer (t);

	if (!notes.contains (note))
	{
		notes.addElement (note);
		noteLabel [t].setVisible (true);
		return true;
	}
	return false;
}


public int checkSize ()
{
	int temp = -1;

	if (notes.size () > 8)
	{
		temp = getNoteNumber2 (0);
		notes.removeElementAt (0);
		noteLabel [temp].setVisible (false);
	}
	return temp;
}
```

```
	public int getNoteNumber2 (int i)
	{
		String s = notes.elementAt (i).toString ();
		int temp = Integer.parseInt (s);
		return temp;
	}


	public boolean removeFromVector (int t)
	{
		Integer note = new Integer (t);

		if (notes.contains (note))
		{
			notes.removeElement (note);
			noteLabel [t].setVisible (false);
			return true;
		}
		return false;

	}
}
```

SystemTwo
“TheMain.java”

```
import java.util.*;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import javax.swing.*;
import com.softsynth.jsyn.*;
import com.softsynth.math.AudioMath;

public class TheMain extends JFrame
{
        JLayeredPane layeredPane;
        JLabel bgLabel, indPlayListLabel, bpmLabel1, bpmLabel2,
                timerLabel1, timerLabel2, elapsedRemainLabel1, elapsedRemainLabel2,
                loadedSampleLabel1, loadedSampleLabel2,
                jogWheelModeLabel, deckIndLabel;

        JLabel [] samplesDisplayLabel = new JLabel [5];
        JLabel [] vuMeterL = new JLabel [10];
        JLabel [] vuMeterR = new JLabel [10];
        JLabel [] [] playPauseLabel = new JLabel [2] [3];
        ControlRegion [] [] region = new ControlRegion [6] [];
        String [] playList;

        String imagesPath = "images/";
        String bgFile = imagesPath + "back3.gif";
        String leftHandFileFree = imagesPath + "handleft03.gif";
        String leftHandFileCtrl = imagesPath + "handleft01.gif";
        String rightHandFileFree = imagesPath + "handright03.gif";
        String rightHandFileCtrl = imagesPath + "handright01.gif";
        String faderVertFile = imagesPath + "bubble01.gif";
        String faderHorFile = imagesPath + "bubble02.gif";
        String ledGreenFile = imagesPath + "ledOnGreen.gif";
        String ledYellowFile = imagesPath + "ledOnYellow.gif";
        String ledRedFile = imagesPath + "ledOnRed.gif";
        String playFile = imagesPath + "play01.gif";
        String pauseFile = imagesPath + "pause01.gif";
        String indPlayListFile = imagesPath + "indPlayList.gif";
        String jogWheelModeFile = imagesPath + "jogWheelMode01.gif";
        String jogWheelFile = imagesPath + "dot02.gif";
        String pitchBendFile = imagesPath + "bubble03.gif";
        String deckIndFile = imagesPath + "deckInd01.gif";

        SynthContext synthContext1, synthContext2;
        SampleReader_16V1 reader1L, reader1R, reader1CueR, reader2L, reader2R, reader2CueR;
        Filter_LowPass fLP1L, fLP1R, fLP2L, fLP2R;
        Filter_HighShelf fHS1L, fHS1R, fHS2L, fHS2R;
        Filter_PeakingEQ fPEQ1L, fPEQ1R, fPEQ2L, fPEQ2R;
        Filter_LowShelf fLS1L, fLS1R, fLS2L, fLS2R;
        AddUnit addLeft, addRight;
        MultiplyUnit multiplyLeft1, multiplyRight1, multiplyLeft2, multiplyRight2;
        CrossFade cFadeCueL, cFadeCueR;
        ChannelOut [] channelOut;
        PeakFollower peakFL, peakFR;
        MultiplyUnit mU1Cue1, mU2Cue1, mU1Cue2, mU2Cue2;
        double rate1 = 44100, rate2 = 44100, bpm1, bpm2;
        int sampleSize = 176400, whichTrack = 0;

        int backgroundMode = 0; // 0 = Normal, 1 = JogWheel Deck1, 2 = JogWheel Deck2
        TheMidiObject thread01;
        TimerThread timerThread;
        VUMeterThread thread03;
        MovingObject leftHand, rightHand;

// ************************************************************************
public TheMain ()
{
        region [0] = new ControlRegion [13];
        region [0] [0] = new Button1 (10, 270, 130, 130, - 1);          //JogWheel Mode Deck1
        region [0] [1] = new Button1 (95, 450, 50, 50, - 1);            //play (0)
        region [0] [2] = new Button1 (120, 70, 50, 25, - 1);            //rew
        region [0] [3] = new Button1 (170, 70, 50, 25, - 1);            //ff
        region [0] [4] = new Button1 (40, 450, 50, 50, - 1);            //cue (2)
        region [0] [5] = new VerticalSlider (240, 45, 21, 21, 55, 455, 11, 200); //+-
        region [0] [6] = new Button1 (0, 700, 130, 15, - 1);            //load sample
        region [0] [7] = new Button1 (130, 700, 45, 15, - 1);           //playlist rew
        region [0] [8] = new Button1 (175, 700, 130, 15, - 1);          //sample1
        region [0] [9] = new Button1 (120, 0, 50, 25, - 1);             //elapsed/remain
        region [0] [10] = new Button1 (5, 460, 30, 30, - 1);            //stop
        region [0] [11] = new Button1 (220, 510, 21, 21, 23);           //PB-
        region [0] [12] = new Button1 (240, 510, 21, 21, 24);           //PB+

        region [1] = new ControlRegion [7];
        region [1] [0] = new VerticalSlider (310, 45, 21, 21, 55, 455, 3, 100);  //lp
        region [1] [1] = new VerticalSlider (340, 45, 21, 21, 55, 455, 4, 100);  //hs
        region [1] [2] = new VerticalSlider (370, 45, 21, 21, 55, 455, 5, 100);  //peq
        region [1] [3] = new VerticalSlider (400, 45, 21, 21, 55, 455, 6, 100);  //ls
        region [1] [4] = new VerticalSlider (280, 45, 21, 21, 55, 455, 13, 100); //gain
        region [1] [5] = new Button1 (305, 700, 130, 15, - 1);                   //sample2
        region [1] [6] = new Button2 (280, 510, 40, 20, - 1);                    //cue1

        region [2] = new ControlRegion [4];
        region [2] [0] = new HorizontalSlider (500, 510, 21, 21, 0, 100);        //Crossfade
        region [2] [1] = new VerticalSlider (440, 265, 21, 21, 275, 455, 1, 45); //Fader 1
        region [2] [2] = new VerticalSlider (540, 265, 21, 21, 275, 455, 2, 45); //Fader 2
        region [2] [3] = new Button1 (435, 700, 130, 15, - 1);                   //sample3

        region [3] = new ControlRegion [7];
        region [3] [0] = new VerticalSlider (610, 45, 21, 21, 55, 455, 7, 100);  //lp
        region [3] [1] = new VerticalSlider (640, 45, 21, 21, 55, 455, 8, 100);  //hs
        region [3] [2] = new VerticalSlider (670, 45, 21, 21, 55, 455, 9, 100);  //peq
        region [3] [3] = new VerticalSlider (700, 45, 21, 21, 55, 455, 10, 100); //ls
        region [3] [4] = new VerticalSlider (580, 45, 21, 21, 55, 455, 14, 100); //gain
        region [3] [5] = new Button1 (565, 700, 130, 15, - 1);                   //sample4
        region [3] [6] = new Button2 (680, 510, 40, 20, - 1);                    //cue2

        region [4] = new ControlRegion [17];
        region [4] [0] = new Button1 (740, 270, 130, 130, - 1);         //JogWheel Mode Deck 2
        region [4] [1] = new Button1 (825, 450, 50, 50, - 1);           //play (1)
        region [4] [2] = new Button1 (840, 70, 50, 25, - 1);            //rew
        region [4] [3] = new Button1 (890, 70, 50, 25, - 1);            //ff
        region [4] [4] = new Button1 (770, 450, 50, 50, - 1);           //cue (3)
        region [4] [5] = new VerticalSlider (960, 45, 21, 21, 55, 455, 12, 200); //+-
        region [4] [6] = new Button1 (870, 700, 130, 15, - 1);          //load sample
        region [4] [7] = new Button1 (825, 700, 45, 15, - 1);           //playlist ff
        region [4] [8] = new Button1 (695, 700, 130, 15, - 1);          //sample5
        region [4] [9] = new Button1 (840, 0, 50, 25, - 1);             //elapsed/remain
        region [4] [10] = new Button2 (736, 75, 40, 20, - 1);           //vumode
        region [4] [11] = new Button2 (786, 75, 40, 20, - 1);           //cuesplit
        region [4] [12] = new RotaryKnob (740, 18, 31, 31, 18, 100); //cue level
        region [4] [13] = new RotaryKnob (790, 18, 31, 31, 19, 100); //cue bal
        region [4] [14] = new Button1 (735, 460, 30, 30, - 1);          //stop
        region [4] [15] = new Button1 (940, 510, 21, 21, 25);           //PB-
        region [4] [16] = new Button1 (960, 510, 21, 21, 26);           //PB+

        region [5] = new ControlRegion [13];
        region [5] [0] = new JogWheel (230, 20, 700, 20, 100);          // JogWheel Mode
        region [5] [1] = new Button1 (120, 0, 100, 70, - 1);            // Deck1
        region [5] [2] = new Button1 (840, 0, 100, 70, - 1);            // Deck2
        region [5] [3] = new Button1 (120, 320, 70, 70, - 1);           // EXIT
        region [5] [4] = new Button1 (120, 410, 70, 70, - 1);           // Stop
        region [5] [5] = new Button1 (120, 500, 100, 100, - 1);         // CUE
region [5] [6] = new Button1 (120, 620, 100, 100, - 1);       // Play/Pause
region [5] [7] = new Button1 (780, 675, 45, 45, - 1);          //PB-
region [5] [8] = new Button1 (830, 675, 45, 45, - 1);          //PB+
region [5] [9] = new VerticalSlider (943, 402, 34, 34, 102, 702, 22, 200); // Pitch Bend Slider
region [5] [10] = new Button1 (120, 110, 120, 50, - 1);       // JogWheel Task
region [5] [11] = new Button1 (890, 80, 45, 45, - 1);          // PB Slider -
region [5] [12] = new Button1 (890, 675, 45, 45, - 1);         // PB Slider +

Image bgImage = Toolkit.getDefaultToolkit ().getImage (bgFile);
Image leftHandImageFree = Toolkit.getDefaultToolkit ().getImage (leftHandFileFree);
Image leftHandImageCtrl = Toolkit.getDefaultToolkit ().getImage (leftHandFileCtrl);
Image rightHandImageFree = Toolkit.getDefaultToolkit ().getImage (rightHandFileFree);
Image rightHandImageCtrl = Toolkit.getDefaultToolkit ().getImage (rightHandFileCtrl);
Image faderVertImage = Toolkit.getDefaultToolkit ().getImage (faderVertFile);
Image faderHorImage = Toolkit.getDefaultToolkit ().getImage (faderHorFile);
Image ledGreenImage = Toolkit.getDefaultToolkit ().getImage (ledGreenFile);
Image ledYellowImage = Toolkit.getDefaultToolkit ().getImage (ledYellowFile);
Image ledRedImage = Toolkit.getDefaultToolkit ().getImage (ledRedFile);
Image playImage = Toolkit.getDefaultToolkit ().getImage (playFile);
Image pauseImage = Toolkit.getDefaultToolkit ().getImage (pauseFile);
Image indPlayListImage = Toolkit.getDefaultToolkit ().getImage (indPlayListFile);
Image jogWheelModeImage = Toolkit.getDefaultToolkit ().getImage (jogWheelModeFile);
Image jogWheelImage = Toolkit.getDefaultToolkit ().getImage (jogWheelFile);
Image pitchBendImage = Toolkit.getDefaultToolkit ().getImage (pitchBendFile);
Image deckIndImage = Toolkit.getDefaultToolkit ().getImage (deckIndFile);

ImageIcon bgIcon = new ImageIcon (bgImage);
ImageIcon leftHandIconFree = new ImageIcon (leftHandImageFree);
ImageIcon leftHandIconCtrl = new ImageIcon (leftHandImageCtrl);
ImageIcon rightHandIconFree = new ImageIcon (rightHandImageFree);
ImageIcon rightHandIconCtrl = new ImageIcon (rightHandImageCtrl);
ImageIcon faderVertIcon = new ImageIcon (faderVertImage);
ImageIcon faderHorIcon = new ImageIcon (faderHorImage);
ImageIcon ledGreenIcon = new ImageIcon (ledGreenImage);
ImageIcon ledYellowIcon = new ImageIcon (ledYellowImage);
ImageIcon ledRedIcon = new ImageIcon (ledRedImage);
ImageIcon playIcon = new ImageIcon (playImage);
ImageIcon pauseIcon = new ImageIcon (pauseImage);
ImageIcon indPlayListIcon = new ImageIcon (indPlayListImage);
ImageIcon jogWheelModeIcon = new ImageIcon (jogWheelModeImage);
ImageIcon jogWheelIcon = new ImageIcon (jogWheelImage);
ImageIcon pitchBendIcon = new ImageIcon (pitchBendImage);
ImageIcon deckIndIcon = new ImageIcon (deckIndImage);

bgLabel = new JLabel (bgIcon);
bgLabel.setBounds (0, 0, bgIcon.getIconWidth (), bgIcon.getIconHeight ());

bpmLabel1 = new JLabel ("000.00");
bpmLabel1.setBounds (130, 45, 65, 20);
bpmLabel1.setFont (new Font ("arial", Font.BOLD, 20));
bpmLabel1.setForeground (Color.red);

bpmLabel2 = new JLabel ("000.00");
bpmLabel2.setBounds (850, 45, 65, 20);
bpmLabel2.setFont (new Font ("arial", Font.BOLD, 20));
bpmLabel2.setForeground (Color.red);

timerLabel1 = new JLabel ("00:00:00");
timerLabel1.setBounds (130, 22, 90, 20);
timerLabel1.setFont (new Font ("arial", Font.BOLD, 20));
timerLabel1.setForeground (Color.green);

timerLabel2 = new JLabel ("00:00:00");
timerLabel2.setBounds (850, 22, 90, 20);
timerLabel2.setFont (new Font ("arial", Font.BOLD, 20));
timerLabel2.setForeground (Color.green);

elapsedRemainLabel1 = new JLabel ("Remain");
elapsedRemainLabel1.setBounds (130, 5, 70, 12);
elapsedRemainLabel1.setFont (new Font ("arial", Font.BOLD, 12));
elapsedRemainLabel1.setForeground (Color.green);

elapsedRemainLabel2 = new JLabel ("Remain");
elapsedRemainLabel2.setBounds (850, 5, 70, 12);
elapsedRemainLabel2.setFont (new Font ("arial", Font.BOLD, 12));
elapsedRemainLabel2.setForeground (Color.green);

loadedSampleLabel1 = new JLabel ("EMPTY");
loadedSampleLabel1.setBounds (2, 702, 129, 12);
loadedSampleLabel1.setFont (new Font ("arial", Font.BOLD, 12));
loadedSampleLabel1.setForeground (Color.green);

loadedSampleLabel2 = new JLabel ("EMPTY");
loadedSampleLabel2.setBounds (872, 702, 127, 12);
loadedSampleLabel2.setFont (new Font ("arial", Font.BOLD, 12));
loadedSampleLabel2.setForeground (Color.green);

for (int i = 0 ; i < samplesDisplayLabel.length ; i++)
{
        samplesDisplayLabel [i] = new JLabel ("");
        samplesDisplayLabel [i].setFont (new Font ("arial", Font.BOLD, 12));
        samplesDisplayLabel [i].setForeground (Color.green);
}

samplesDisplayLabel [0].setBounds (177, 702, 127, 12);
samplesDisplayLabel [1].setBounds (307, 702, 127, 12);
samplesDisplayLabel [2].setBounds (437, 702, 127, 12);
samplesDisplayLabel [3].setBounds (567, 702, 127, 12);
samplesDisplayLabel [4].setBounds (697, 702, 127, 12);

leftHand = new MovingObject (new JLabel (leftHandIconFree), new JLabel (leftHandIconCtrl), - 21);
rightHand = new MovingObject (new JLabel (rightHandIconFree), new JLabel (rightHandIconCtrl), - 11);
leftHand.labelFree.setBounds (- 30, - 60, leftHandIconFree.getIconWidth (), leftHandIconFree.getIconHeight ());
leftHand.labelCtrl.setBounds (- 30, - 60, leftHandIconCtrl.getIconWidth (), leftHandIconCtrl.getIconHeight ());
rightHand.labelFree.setBounds (- 30, - 30, rightHandIconFree.getIconWidth (), rightHandIconFree.getIconHeight ());
rightHand.labelCtrl.setBounds (- 30, - 30, rightHandIconCtrl.getIconWidth (), rightHandIconCtrl.getIconHeight ());

region [0] [5].setLabel (new JLabel (faderVertIcon));
region [1] [0].setLabel (new JLabel (faderVertIcon));
region [1] [1].setLabel (new JLabel (faderVertIcon));
region [1] [2].setLabel (new JLabel (faderVertIcon));
region [1] [3].setLabel (new JLabel (faderVertIcon));
region [1] [4].setLabel (new JLabel (faderVertIcon));
region [1] [6].setLabel (new JLabel (ledRedIcon));
region [2] [0].setLabel (new JLabel (faderHorIcon));
region [2] [1].setLabel (new JLabel (faderVertIcon));
region [2] [2].setLabel (new JLabel (faderVertIcon));
region [3] [0].setLabel (new JLabel (faderVertIcon));
region [3] [1].setLabel (new JLabel (faderVertIcon));
region [3] [2].setLabel (new JLabel (faderVertIcon));
region [3] [3].setLabel (new JLabel (faderVertIcon));
region [3] [4].setLabel (new JLabel (faderVertIcon));
region [3] [6].setLabel (new JLabel (ledRedIcon));
region [4] [5].setLabel (new JLabel (faderVertIcon));
region [4] [10].setLabel (new JLabel (ledRedIcon));
region [4] [11].setLabel (new JLabel (ledRedIcon));
region [4] [12].setLabel (new JLabel (ledRedIcon));
region [4] [13].setLabel (new JLabel (ledRedIcon));
region [5] [0].setLabel (new JLabel (jogWheelIcon));
region [5] [9].setLabel (new JLabel (pitchBendIcon));
region [5] [10].setLabel (new JLabel ("Pitch Bend"));

region [0] [5].getLabel ().setBounds (region [0] [5].x1, region [0] [5].y1, 21, 21);
region [1] [0].getLabel ().setBounds (region [1] [0].x1, region [1] [0].y1, 21, 21);
region [1] [1].getLabel ().setBounds (region [1] [1].x1, region [1] [1].y1, 21, 21);
region [1] [2].getLabel ().setBounds (region [1] [2].x1, region [1] [2].y1, 21, 21);
region [1] [3].getLabel ().setBounds (region [1] [3].x1, region [1] [3].y1, 21, 21);
region [1] [4].getLabel ().setBounds (region [1] [4].x1, region [1] [4].y1, 21, 21);
region [1] [6].getLabel ().setBounds (309, 516, 7, 7);
region [2] [0].getLabel ().setBounds (region [2] [0].x1, region [2] [0].y1, 21, 21);
region [2] [1].getLabel ().setBounds (region [2] [1].x1, region [2] [1].y1, 21, 21);
region [2] [2].getLabel ().setBounds (region [2] [2].x1, region [2] [2].y1, 21, 21);
region [3] [0].getLabel ().setBounds (region [3] [0].x1, region [3] [0].y1, 21, 21);
region [3] [1].getLabel ().setBounds (region [3] [1].x1, region [3] [1].y1, 21, 21);
region [3] [2].getLabel ().setBounds (region [3] [2].x1, region [3] [2].y1, 21, 21);
region [3] [3].getLabel ().setBounds (region [3] [3].x1, region [3] [3].y1, 21, 21);
region [3] [4].getLabel ().setBounds (region [3] [4].x1, region [3] [4].y1, 21, 21);
region [3] [6].getLabel ().setBounds (709, 516, 7, 7);
region [4] [5].getLabel ().setBounds (region [4] [5].x1, region [4] [5].y1, 21, 21);
region [4] [10].getLabel ().setBounds (765, 81, 7, 7);
region [4] [11].getLabel ().setBounds (815, 81, 7, 7);
region [4] [12].getLabel ().setBounds (752, 23, 7, 7);
region [4] [13].getLabel ().setBounds (802, 23, 7, 7);
region [5] [0].getLabel ().setBounds (region [5] [0].x1, region [5] [0].y1, 50, 50);
region [5] [9].getLabel ().setBounds (region [5] [9].x1, region [5] [9].y1, 34, 34);
region [5] [10].getLabel ().setBounds (120, 110, 120, 50);

for (int i = 0 ; i < 6 ; i++)
{
        vuMeterL [i] = new JLabel (ledGreenIcon);
        vuMeterR [i] = new JLabel (ledGreenIcon);
}
for (int i = 6 ; i < 8 ; i++)
{
        vuMeterL [i] = new JLabel (ledYellowIcon);
        vuMeterR [i] = new JLabel (ledYellowIcon);
}
for (int i = 8 ; i < 10 ; i++)
{
        vuMeterL [i] = new JLabel (ledRedIcon);
        vuMeterR [i] = new JLabel (ledRedIcon);
}
for (int i = 0 ; i < 10 ; i++)
{
        vuMeterL [i].setBounds (470, 210 - (i * 20), 7, 7);
        vuMeterR [i].setBounds (520, 210 - (i * 20), 7, 7);
        vuMeterL [i].setVisible (false);
        vuMeterR [i].setVisible (false);
}
for (int i = 0 ; i < playPauseLabel [0].length ; i++)
{
        playPauseLabel [0] [i] = new JLabel (playIcon);
        playPauseLabel [0] [i].setVisible (false);
}
for (int i = 0 ; i < playPauseLabel [1].length ; i++)
{
        playPauseLabel [1] [i] = new JLabel (pauseIcon);
        playPauseLabel [1] [i].setVisible (false);
}

playPauseLabel [0] [0].setBounds (101, 467, playIcon.getIconWidth (), playIcon.getIconHeight ());
playPauseLabel [0] [1].setBounds (831, 467, playIcon.getIconWidth (), playIcon.getIconHeight ());
playPauseLabel [0] [2].setBounds (150, 660, playIcon.getIconWidth (), playIcon.getIconHeight ());
playPauseLabel [1] [0].setBounds (125, 467, pauseIcon.getIconWidth (), pauseIcon.getIconHeight ());
playPauseLabel [1] [1].setBounds (855, 467, pauseIcon.getIconWidth (), pauseIcon.getIconHeight ());
playPauseLabel [1] [2].setBounds (174, 660, pauseIcon.getIconWidth (), pauseIcon.getIconHeight ());

indPlayListLabel = new JLabel (indPlayListIcon);
indPlayListLabel.setBounds (175, 699, indPlayListIcon.getIconWidth (), indPlayListIcon.getIconHeight ());

region [5] [10].getLabel ().setFont (new Font ("arial", Font.BOLD, 20));
region [5] [10].getLabel ().setForeground (Color.yellow);
region [5] [10].getLabel ().setHorizontalAlignment (JLabel.CENTER);

layeredPane = new JLayeredPane ();
layeredPane.setBackground (Color.black);
layeredPane.setPreferredSize (new Dimension (bgIcon.getIconWidth (), bgIcon.getIconHeight ()));
```

```
region [1] [6].getLabel ().setVisible (false);
region [3] [6].getLabel ().setVisible (false);
region [4] [10].getLabel ().setVisible (false);
region [4] [11].getLabel ().setVisible (false);
region [5] [0].getLabel ().setVisible (false);
region [5] [9].getLabel ().setVisible (false);
region [5] [10].getLabel ().setVisible (false);

jogWheelModeLabel = new JLabel (jogWheelModeIcon);
jogWheelModeLabel.setBounds (0, 0, jogWheelModeIcon.getIconWidth (), jogWheelModeIcon.getIconHeight ());
jogWheelModeLabel.setVisible (false);

deckIndLabel = new JLabel (deckIndIcon);
deckIndLabel.setBounds (110, 0, deckIndIcon.getIconWidth (), deckIndIcon.getIconHeight ());
deckIndLabel.setVisible (false);

layeredPane.add (bgLabel, new Integer (0));
layeredPane.add (bpmLabel1, new Integer (3));
layeredPane.add (bpmLabel2, new Integer (3));
layeredPane.add (timerLabel1, new Integer (3));
layeredPane.add (timerLabel2, new Integer (3));
layeredPane.add (elapsedRemainLabel1, new Integer (3));
layeredPane.add (elapsedRemainLabel2, new Integer (3));
layeredPane.add (loadedSampleLabel1, new Integer (1));
layeredPane.add (loadedSampleLabel2, new Integer (1));

for (int i = 0 ; i < samplesDisplayLabel.length ; i++)
        layeredPane.add (samplesDisplayLabel [i], new Integer (1));

layeredPane.add (leftHand.labelFree, new Integer (4));
layeredPane.add (leftHand.labelCtrl, new Integer (4));
layeredPane.add (rightHand.labelFree, new Integer (4));
layeredPane.add (rightHand.labelCtrl, new Integer (4));

layeredPane.add (region [0] [5].getLabel (), new Integer (1));
layeredPane.add (region [1] [0].getLabel (), new Integer (1));
layeredPane.add (region [1] [1].getLabel (), new Integer (1));
layeredPane.add (region [1] [2].getLabel (), new Integer (1));
layeredPane.add (region [1] [3].getLabel (), new Integer (1));
layeredPane.add (region [1] [4].getLabel (), new Integer (1));
layeredPane.add (region [1] [6].getLabel (), new Integer (1));
layeredPane.add (region [2] [0].getLabel (), new Integer (1));
layeredPane.add (region [2] [1].getLabel (), new Integer (1));
layeredPane.add (region [2] [2].getLabel (), new Integer (1));
layeredPane.add (region [3] [0].getLabel (), new Integer (1));
layeredPane.add (region [3] [1].getLabel (), new Integer (1));
layeredPane.add (region [3] [2].getLabel (), new Integer (1));
layeredPane.add (region [3] [3].getLabel (), new Integer (1));
layeredPane.add (region [3] [4].getLabel (), new Integer (1));
layeredPane.add (region [3] [6].getLabel (), new Integer (1));
layeredPane.add (region [4] [5].getLabel (), new Integer (1));
layeredPane.add (region [4] [10].getLabel (), new Integer (1));
layeredPane.add (region [4] [11].getLabel (), new Integer (1));
layeredPane.add (region [4] [12].getLabel (), new Integer (1));
layeredPane.add (region [4] [13].getLabel (), new Integer (1));
layeredPane.add (region [5] [0].getLabel (), new Integer (3));
layeredPane.add (region [5] [9].getLabel (), new Integer (3));
layeredPane.add (region [5] [10].getLabel (), new Integer (3));

for (int i = 0 ; i < 10 ; i++)
{
        layeredPane.add (vuMeterL [i], new Integer (1));
        layeredPane.add (vuMeterR [i], new Integer (1));
}

layeredPane.add (playPauseLabel [0] [0], new Integer (1));
layeredPane.add (playPauseLabel [0] [1], new Integer (1));
layeredPane.add (playPauseLabel [0] [2], new Integer (3));
layeredPane.add (playPauseLabel [1] [0], new Integer (1));
layeredPane.add (playPauseLabel [1] [1], new Integer (1));
layeredPane.add (playPauseLabel [1] [2], new Integer (3));

layeredPane.add (indPlayListLabel, new Integer (1));

layeredPane.add (jogWheelModeLabel, new Integer (2));
layeredPane.add (deckIndLabel, new Integer (3));

getContentPane ().add (layeredPane, BorderLayout.CENTER);
setSize (1032, 776);
setLocation (- 4, - 4);
setTitle ("SYSTEMTWO");
setVisible (true);

playList = new File ("Samples").list ();
sortPlayList ();
writePlayList ();


//  JSyn initialization
try
{
        synthContext1 = new SynthContext ();
        synthContext1.initialize ();
        synthContext1.start (0, Synth.DEFAULT_FRAME_RATE, - 1, 0, 6, 2);        // 6 = Marc8-M Playback 1-2
        synthContext2 = new SynthContext ();
        synthContext2.initialize ();
        synthContext2.start (0, Synth.DEFAULT_FRAME_RATE, - 1, 0, 7, 2);        // 7 = Marc8-M Playback 3-4

        reader1L = new SampleReader_16V1 (synthContext1);
        reader1R = new SampleReader_16V1 (synthContext1);
        reader1CueR = new SampleReader_16V1 (synthContext2);
        reader2L = new SampleReader_16V1 (synthContext1);
        reader2R = new SampleReader_16V1 (synthContext1);
        reader2CueR = new SampleReader_16V1 (synthContext2);

        fLP1L = new Filter_LowPass (synthContext1);
        fLP1R = new Filter_LowPass (synthContext1);
        fLP2L = new Filter_LowPass (synthContext1);
        fLP2R = new Filter_LowPass (synthContext1);

        fHS1L = new Filter_HighShelf (synthContext1);
        fHS1R = new Filter_HighShelf (synthContext1);
        fHS2L = new Filter_HighShelf (synthContext1);
        fHS2R = new Filter_HighShelf (synthContext1);

        fPEQ1L = new Filter_PeakingEQ (synthContext1);
        fPEQ1R = new Filter_PeakingEQ (synthContext1);
        fPEQ2L = new Filter_PeakingEQ (synthContext1);
        fPEQ2R = new Filter_PeakingEQ (synthContext1);

        fLS1L = new Filter_LowShelf (synthContext1);
        fLS1R = new Filter_LowShelf (synthContext1);
        fLS2L = new Filter_LowShelf (synthContext1);
        fLS2R = new Filter_LowShelf (synthContext1);

        multiplyLeft1 = new MultiplyUnit (synthContext1);
        multiplyRight1 = new MultiplyUnit (synthContext1);
        multiplyLeft2 = new MultiplyUnit (synthContext1);
        multiplyRight2 = new MultiplyUnit (synthContext1);

        addLeft = new AddUnit (synthContext1);
        addRight = new AddUnit (synthContext1);
        cFadeCueL = new CrossFade (synthContext2);
        cFadeCueR = new CrossFade (synthContext2);

        channelOut = new ChannelOut [4];
        channelOut [0] = new ChannelOut (synthContext1, 0);
```

```
channelOut [1] = new ChannelOut (synthContext1, 1);
channelOut [2] = new ChannelOut (synthContext2, 0);
channelOut [3] = new ChannelOut (synthContext2, 1);

peakFL = new PeakFollower (synthContext1);
peakFR = new PeakFollower (synthContext1);
peakFL.halfLife.set (0.5);
peakFR.halfLife.set (0.5);

mU1Cue1 = new MultiplyUnit (synthContext2);
mU2Cue1 = new MultiplyUnit (synthContext2);
mU1Cue2 = new MultiplyUnit (synthContext2);
mU2Cue2 = new MultiplyUnit (synthContext2);

// reader to LP
reader1L.output.connect (fLP1L.input);
reader1R.output.connect (fLP1R.input);
reader2L.output.connect (fLP2L.input);
reader2R.output.connect (fLP2R.input);
// LP to HS
fLP1L.output.connect (fHS1L.input);
fLP1R.output.connect (fHS1R.input);
fLP2L.output.connect (fHS2L.input);
fLP2R.output.connect (fHS2R.input);
// HS to PEQ
fHS1L.output.connect (fPEQ1L.input);
fHS1R.output.connect (fPEQ1R.input);
fHS2L.output.connect (fPEQ2L.input);
fHS2R.output.connect (fPEQ2R.input);

fHS1L.frequency.set (8000);
fHS1R.frequency.set (8000);
fHS2L.frequency.set (8000);
fHS2R.frequency.set (8000);
// PEQ to LS
fPEQ1L.output.connect (fLS1L.input);
fPEQ1R.output.connect (fLS1R.input);
fPEQ2L.output.connect (fLS2L.input);
fPEQ2R.output.connect (fLS2R.input);

fPEQ1L.frequency.set (5000);
fPEQ1R.frequency.set (5000);
fPEQ2L.frequency.set (5000);
fPEQ2R.frequency.set (5000);
// LS to Multi
fLS1L.output.connect (multiplyLeft1.inputA);
fLS1R.output.connect (multiplyRight1.inputA);
fLS2L.output.connect (multiplyLeft2.inputA);
fLS2R.output.connect (multiplyRight2.inputA);

fLS1L.frequency.set (80);
fLS1R.frequency.set (80);
fLS2L.frequency.set (80);
fLS2R.frequency.set (80);

multiplyLeft1.output.connect (addLeft.inputA);
multiplyRight1.output.connect (addRight.inputA);
multiplyLeft2.output.connect (addLeft.inputB);
multiplyRight2.output.connect (addRight.inputB);

addLeft.output.connect (channelOut [0].input);
addRight.output.connect (channelOut [1].input);

addLeft.output.connect (peakFL.input);
addRight.output.connect (peakFR.input);
```

```
        reader1CueR.output.connect (mU1Cue1.inputA);
        reader2CueR.output.connect (mU2Cue1.inputA);
        mU1Cue1.inputB.set (1);
        mU2Cue1.inputB.set (1);

        mU1Cue1.output.connect (mU1Cue2.inputA);
        mU2Cue1.output.connect (mU2Cue2.inputA);
        mU1Cue2.inputB.set (1);
        mU2Cue2.inputB.set (1);

        mU1Cue2.output.connect (0, cFadeCueL.input, 0);
        mU2Cue2.output.connect (0, cFadeCueL.input, 1);
        mU1Cue2.output.connect (0, cFadeCueR.input, 0);
        mU2Cue2.output.connect (0, cFadeCueR.input, 1);

        cFadeCueL.output.connect (channelOut [2].input);
        cFadeCueR.output.connect (channelOut [3].input);

        fLP1L.start ();
        fLP1R.start ();
        fLP2L.start ();
        fLP2R.start ();

        fHS1L.start ();
        fHS1R.start ();
        fHS2L.start ();
        fHS2R.start ();

        fPEQ1L.start ();
        fPEQ1R.start ();
        fPEQ2L.start ();
        fPEQ2R.start ();

        fLS1L.start ();
        fLS1R.start ();
        fLS2L.start ();
        fLS2R.start ();

        multiplyLeft1.start ();
        multiplyRight1.start ();
        multiplyLeft2.start ();
        multiplyRight2.start ();
        addLeft.start ();
        addRight.start ();
        cFadeCueL.start ();
        cFadeCueR.start ();

        channelOut [0].start ();
        channelOut [1].start ();
        channelOut [2].start ();
        channelOut [3].start ();

        peakFL.start ();
        peakFR.start ();

        mU1Cue1.start ();
        mU2Cue1.start ();
        mU1Cue2.start ();
        mU2Cue2.start ();

        region [0] [5].setValues (100);
        region [1] [0].setValues (100);
        region [1] [1].setValues (50);
        region [1] [2].setValues (50);
        region [1] [3].setValues (50);
        region [1] [4].setValues (50);
        region [2] [0].setValues (500);
        region [2] [1].setValues (33);
        region [2] [2].setValues (33);
        region [3] [0].setValues (100);
        region [3] [1].setValues (50);
        region [3] [2].setValues (50);
        region [3] [3].setValues (50);
        region [3] [4].setValues (50);
        region [4] [5].setValues (100);
        region [4] [12].setValues (50);
        region [4] [13].setValues (50);
        region [5] [0].setValues (50);
        reader1CueR.amplitude.set (0);
        reader2CueR.amplitude.set (0);

        changeParameters (region [0] [5].iD, region [0] [5].value);
        changeParameters (region [1] [0].iD, region [1] [0].value);
        changeParameters (region [1] [1].iD, region [1] [1].value);
        changeParameters (region [1] [2].iD, region [1] [2].value);
        changeParameters (region [1] [3].iD, region [1] [3].value);
        changeParameters (region [1] [4].iD, region [1] [4].value);
        changeParameters (region [2] [0].iD, region [2] [0].value);
        changeParameters (region [2] [1].iD, region [2] [1].value);
        changeParameters (region [2] [2].iD, region [2] [2].value);
        changeParameters (region [3] [0].iD, region [3] [0].value);
        changeParameters (region [3] [1].iD, region [3] [1].value);
        changeParameters (region [3] [2].iD, region [3] [2].value);
        changeParameters (region [3] [3].iD, region [3] [3].value);
        changeParameters (region [3] [4].iD, region [3] [4].value);
        changeParameters (region [4] [5].iD, region [4] [5].value);
        changeParameters (region [4] [12].iD, region [4] [12].value);
        changeParameters (region [4] [13].iD, region [4] [13].value);
    }
    catch (SynthException e)
    {
        System.err.println ("TheMain constructor");
        SynthAlert.showError (this, e);
    }

    thread01 = new TheMidiObject (this);
    thread03 = new VUMeterThread ();
    timerThread = new TimerThread ();

    layeredPane.addMouseMotionListener (new MouseMotionAdapter ()
    {
        public void mouseMoved (MouseEvent e)
        {
            int x = e.getX ();
            int y = e.getY () - 40;
            //leftHand.midiIn (false, x, y, 0);
            rightHand.midiIn (false, x, y, 0);
        }
        public void mouseDragged (MouseEvent e)
        {
            int x = e.getX ();
            int y = e.getY () - 40;
            //leftHand.midiIn (true, x, y, 0);
            rightHand.midiIn (true, x, y, 0);
        }
    }
    );
    addWindowListener (new WindowAdapter ()
    {
        public void windowClosing (WindowEvent e)
        {
            stop ();
        }
    }
    );
}
```

```
// ****************************************************************************
public void stop ()
{
    thread03.stop ();
    timerThread.stop ();

    try
    {
        synthContext1.sleepForTicks (1000);
        synthContext2.sleepForTicks (1000);
        reader1L.stop ();
        reader1R.stop ();
        reader1CueR.stop ();
        reader2L.stop ();
        reader2R.stop ();
        reader2CueR.stop ();

        fLP1L.stop ();
        fLP1R.stop ();
        fLP2L.stop ();
        fLP2R.stop ();

        fHS1L.stop ();

        fHS1R.stop ();
        fHS2L.stop ();
        fHS2R.stop ();

        fPEQ1L.stop ();
        fPEQ1R.stop ();
        fPEQ2L.stop ();
        fPEQ2R.stop ();

        fLS1L.stop ();
        fLS1R.stop ();
        fLS2L.stop ();
        fLS2R.stop ();

        multiplyLeft1.stop ();
        multiplyRight1.stop ();
        multiplyLeft2.stop ();
        multiplyRight2.stop ();
        addLeft.stop ();
        addRight.stop ();
        cFadeCueL.stop ();
        cFadeCueR.stop ();

        channelOut [0].stop ();
        channelOut [1].stop ();
        channelOut [2].stop ();
        channelOut [3].stop ();

        peakFL.stop ();
        peakFR.stop ();

        mU1Cue1.stop ();
        mU2Cue1.stop ();
        mU1Cue2.stop ();
        mU2Cue2.stop ();

        System.out.println ("Stop the Engine");
        synthContext1.stop ();
        synthContext1.delete ();
        synthContext2.stop ();
        synthContext2.delete ();
        System.out.println ("Exit System");
        System.exit (0);
    }
    catch (SynthException e)
    {
        System.err.println ("stop all");
        SynthAlert.showError (this, e);
    }
}


// ****************************************************************************
public static void main (String [] args) throws Exception
{
    new TheMain ();
}


// ****************************************************************************
public class MovingObject

{
    JLabel labelFree, labelCtrl;
    int rRow = -1;
    int rCol = -1;
    boolean activeNode = false;
    boolean controlMoveState = false;
    int scratchDiffPos = 0;
    int fingerPos;

    public MovingObject (JLabel labF, JLabel labC, int f)
    {
        labelFree = labF;
        labelCtrl = labC;
        fingerPos = f;
    }

    public void midiIn (boolean ctrlMove, int x, int y, double tStamp)
    {
        try
        {
            if (controlMoveState != ctrlMove)
            {
                if (ctrlMove)
                {
                    labelFree.setVisible (false);
                    labelCtrl.setVisible (true);
                }
                else
                {
                    labelCtrl.setVisible (false);
                    labelFree.setVisible (true);
                }
                controlMoveState = ctrlMove;
            }

            if (ctrlMove)
            {
                labelCtrl.setLocation (x + fingerPos, y);
                if (backgroundMode == 0)
                {
                    if (rRow >= 0)
                    {
                        if (region [rRow] [rCol].setValues (x, y))
                            changeParameters (region [rRow] [rCol].iD, region [rRow] [rCol].value);
                    }
                    else if (x < 280)
                    {
                        if (region [0] [0].equals (x, y))
                        {
                            backgroundMode = 1;
                            jogWheelModeLabel.setVisible (true);
    deckIndLabel.setLocation (110, 0);
    deckIndLabel.setVisible (true);

    region [5] [0].iD = 20;
    region [5] [7].iD = 23;
    region [5] [8].iD = 24;
    region [5] [9].iD = 11;
    region [5] [9].setValues (
            region [0] [5].value * region [5] [9].max /
            region [0] [5].max);
    region [5] [0].getLabel ().setVisible (true);
    region [5] [9].getLabel ().setVisible (true);
    region [5] [10].getLabel ().setVisible (true);
    if (state1 == 1)
        playPauseLabel [0] [2].setVisible (true);
    else if (state1 == 2)
        playPauseLabel [1] [2].setVisible (true);

    if (region [0] [0].setValues (x, y))
        changeParameters (region [0] [0].iD, region [0] [0].value);
    rRow = 0;
    rCol = 0;
}
else if (region [0] [1].equals (x, y))
{
    playPauseStop (0);
    rRow = 0;
    rCol = 1;
}
else if (region [0] [2].equals (x, y))
{
    reverseTrack (0, true);
    activeNode = true;
    rRow = 0;
    rCol = 2;
}
else if (region [0] [3].equals (x, y))
{
    fastForward (0, true);
    activeNode = true;
    rRow = 0;
    rCol = 3;
}
else if (region [0] [4].equals (x, y))
{
    setCuePos (true);
    rRow = 0;
    rCol = 4;
}
else if (region [0] [5].equals (x, y))
{
    if (region [0] [5].setValues (x, y))
        changeParameters (region [0] [5].iD, region [0] [5].value);
    rRow = 0;
    rCol = 5;
}
else if (region [0] [6].equals (x, y))
{
    loadSample (true, playList [whichTrack]);
    loadedSampleLabel1.setText (playList [whichTrack]);
    rRow = 0;
    rCol = 6;
}
```

```
    else if (region [0] [7].equals (x, y))
    {
        if (whichTrack >= 5)
        {
            whichTrack -= 5;
            writePlayList ();
        }
        rRow = 0;
        rCol = 7;
    }
    else if (region [0] [8].equals (x, y))
    {
        if (!samplesDisplayLabel [0].getText ().equals (""))
        {
            whichTrack = (whichTrack / 5) * 5;
            moveIndPlayListLabel (whichTrack);
        }
        rRow = 0;
        rCol = 8;
    }
    else if (region [0] [9].equals (x, y))
    {
        timerThread.changeER (true);
        if (timerThread.eR1)
            elapsedRemainLabel1.setText ("Elapsed");
        else
            elapsedRemainLabel1.setText ("Remain");
        rRow = 0;
        rCol = 9;
    }
    else if (region [0] [10].equals (x, y))
    {
        playPauseStop (2);
        rRow = 0;
        rCol = 10;
    }
    else if (region [0] [11].equals (x, y))
    {
        double t = rate1 - (.04 * rate1);
        setFreq (true, t);
        activeNode = true;
        rRow = 0;
        rCol = 11;
    }
    else if (region [0] [12].equals (x, y))
    {
        double t = rate1 + (.04 * rate1);
        setFreq (true, t);
        activeNode = true;
        rRow = 0;
        rCol = 12;
    }
}
else if (x < 420)
{
    if (region [1] [0].equals (x, y))
    {
        if (region [1] [0].setValues (x, y))
            changeParameters (region [1] [0].iD, region [1] [0].value);
        rRow = 1;
        rCol = 0;
    }
    else if (region [1] [1].equals (x, y))
    {
        if (region [1] [1].setValues (x, y))
            changeParameters (region [1] [1].iD, region [1] [1].value);
        rRow = 1;
        rCol = 1;
    }
    else if (region [1] [2].equals (x, y))
    {
        if (region [1] [2].setValues (x, y))
            changeParameters (region [1] [2].iD, region [1] [2].value);
        rRow = 1;
        rCol = 2;
    }
    else if (region [1] [3].equals (x, y))
    {
        if (region [1] [3].setValues (x, y))
            changeParameters (region [1] [3].iD, region [1] [3].value);
        rRow = 1;
        rCol = 3;
    }
    else if (region [1] [4].equals (x, y))
    {
        if (region [1] [4].setValues (x, y))
            changeParameters (region [1] [4].iD, region [1] [4].value);
        rRow = 1;
        rCol = 4;
    }
    else if (region [1] [5].equals (x, y))
    {
        if (!samplesDisplayLabel [1].getText ().equals (""))
        {
            whichTrack = (whichTrack / 5) * 5 + 1;
            moveIndPlayListLabel (whichTrack);
        }
        rRow = 1;
        rCol = 5;
    }
    else if (region [1] [6].equals (x, y))
    {
        if (region [1] [6].setValues (0))
            reader1CueR.amplitude.set (reader1R.amplitude.get ());
        else
            reader1CueR.amplitude.set (0);
        rRow = 1;
        rCol = 6;
    }
}
else if (x < 580)
{
    if (region [2] [0].equals (x, y))
    {
        if (region [2] [0].setValues (x, y))
            changeParameters (region [2] [0].iD, region [2] [0].value);
        rRow = 2;
        rCol = 0;
    }
    else if (region [2] [1].equals (x, y))
    {
        if (region [2] [1].setValues (x, y))
            changeParameters (region [2] [1].iD, region [2] [1].value);
        rRow = 2;
        rCol = 1;
    }
    else if (region [2] [2].equals (x, y))
    {
        if (region [2] [2].setValues (x, y))
            changeParameters (region [2] [2].iD, region [2] [2].value);
        rRow = 2;
        rCol = 2;
    }
    else if (region [2] [3].equals (x, y))
    {
        if (!samplesDisplayLabel [2].getText ().equals (""))
        {
            whichTrack = (whichTrack / 5) * 5 + 2;
            moveIndPlayListLabel (whichTrack);
        }
        rRow = 2;
        rCol = 3;
    }
}
else if (x < 740)
{
    if (region [3] [0].equals (x, y))
    {
        if (region [3] [0].setValues (x, y))
            changeParameters (region [3] [0].iD, region [3] [0].value);
        rRow = 3;
        rCol = 0;
    }
    else if (region [3] [1].equals (x, y))
    {
        if (region [3] [1].setValues (x, y))
            changeParameters (region [3] [1].iD, region [3] [1].value);
        rRow = 3;
        rCol = 1;
    }
    else if (region [3] [2].equals (x, y))
    {
        if (region [3] [2].setValues (x, y))
            changeParameters (region [3] [2].iD, region [3] [2].value);
        rRow = 3;
        rCol = 2;
    }
    else if (region [3] [3].equals (x, y))
    {
        if (region [3] [3].setValues (x, y))
            changeParameters (region [3] [3].iD, region [3] [3].value);
        rRow = 3;
        rCol = 3;
    }
    else if (region [3] [4].equals (x, y))
    {
        if (region [3] [4].setValues (x, y))
            changeParameters (region [3] [4].iD, region [3] [4].value);
        rRow = 3;
        rCol = 4;
    }
    else if (region [3] [5].equals (x, y))
    {
        if (!samplesDisplayLabel [3].getText ().equals (""))
        {
            whichTrack = (whichTrack / 5) * 5 + 3;
            moveIndPlayListLabel (whichTrack);
        }
        rRow = 3;
        rCol = 5;
    }
    else if (region [3] [6].equals (x, y))
    {
        if (region [3] [6].setValues (0))
            reader2CueR.amplitude.set (reader2R.amplitude.get ());
        else
            reader2CueR.amplitude.set (0);
        rRow = 3;
        rCol = 6;
    }
}
```

```
else
{
	if (region [4] [0].equals (x, y))
	{
		backgroundMode = 2;
		jogWheelModeLabel.setVisible (true);
		deckIndLabel.setLocation (830, 0);
		deckIndLabel.setVisible (true);

		region [5] [0].iD = 21;
		region [5] [7].iD = 25;
		region [5] [8].iD = 26;
		region [5] [9].iD = 12;
		region [5] [9].setValues (
				region [4] [5].value * region [5] [9].max /
				region [4] [5].max);
		region [5] [0].getLabel ().setVisible (true);
		region [5] [9].getLabel ().setVisible (true);
		region [5] [10].getLabel ().setVisible (true);
		if (state2 == 1)
			playPauseLabel [0] [2].setVisible (true);
		else if (state2 == 2)
			playPauseLabel [1] [2].setVisible (true);

		if (region [4] [0].setValues (x, y))
			changeParameters (region [4] [0].iD, region [4] [0].value);
		rRow = 4;
		rCol = 0;
	}
	else if (region [4] [1].equals (x, y))
	{
		playPauseStop (1);
		rRow = 4;
		rCol = 1;
	}
	else if (region [4] [2].equals (x, y))
	{
		reverseTrack (4, true);
		activeNode = true;
		rRow = 4;
		rCol = 2;
	}
	else if (region [4] [3].equals (x, y))
	{
		fastForward (4, true);
		activeNode = true;
		rRow = 4;
		rCol = 3;
	}
	else if (region [4] [4].equals (x, y))
	{
		setCuePos (false);
		rRow = 4;
		rCol = 4;
	}
	else if (region [4] [5].equals (x, y))
	{
		if (region [4] [5].setValues (x, y))
			changeParameters (region [4] [5].iD, region [4] [5].value);
		rRow = 4;
		rCol = 5;
	}
	else if (region [4] [6].equals (x, y))
	{
		loadSample (false, playList [whichTrack]);
		loadedSampleLabel2.setText (playList [whichTrack]);
		rRow = 4;
		rCol = 6;
	}
```

```
else if (region [4] [7].equals (x, y))
{
	if (whichTrack < playList.length)
	{
		whichTrack += 5;
		if (whichTrack >= playList.length)
		{
			whichTrack = playList.length - 1;
			moveIndPlayListLabel (whichTrack);
		}
		writePlayList ();
	}
	rRow = 4;
	rCol = 7;
}
else if (region [4] [8].equals (x, y))
{
	if (!samplesDisplayLabel [4].getText ().equals (""))
	{
		whichTrack = (whichTrack / 5) * 5 + 4;
		moveIndPlayListLabel (whichTrack);
	}
	rRow = 4;
	rCol = 8;
}
else if (region [4] [9].equals (x, y))
{
	timerThread.changeER (false);
	if (timerThread.eR2)
		elapsedRemainLabel2.setText ("Elapsed");
	else
		elapsedRemainLabel2.setText ("Remain");
	rRow = 4;
	rCol = 9;
}
else if (region [4] [10].equals (x, y))
{
	if (region [4] [10].setValues (0))
		thread03.setOutput (false);
	else
		thread03.setOutput (true);
	rRow = 4;
	rCol = 10;
}
else if (region [4] [11].equals (x, y))
{
	if (region [4] [11].setValues (0))
	{
		cFadeCueL.fade.set (- 1);
		cFadeCueR.fade.set (1);
	}
	else
	{
		cFadeCueL.fade.set (0);
		cFadeCueR.fade.set (0);
	}
	rRow = 4;
	rCol = 11;
}
else if (region [4] [12].equals (x, y))
{
	if (region [4] [12].setValues (x, y))
		changeParameters (region [4] [12].iD, region [4] [12].value);
	rRow = 4;
	rCol = 12;
}
```

```
        else if (region [4] [13].equals (x, y))
        {
            if (region [4] [13].setValues (x, y))
                changeParameters (region [4] [13].iD, region [4] [13].value);
            rRow = 4;
            rCol = 13;
        }
        else if (region [4] [14].equals (x, y))
        {
            playPauseStop (3);
            rRow = 4;
            rCol = 14;
        }
        else if (region [4] [15].equals (x, y))
        {
            double t = rate2 - (.04 * rate2);
            setFreq (false, t);
            activeNode = true;
            rRow = 4;

            rCol = 15;
        }
        else if (region [4] [16].equals (x, y))
        {
            double t = rate2 + (.04 * rate2);
            setFreq (false, t);
            activeNode = true;
            rRow = 4;
            rCol = 16;
        }
    }
}
else if (backgroundMode == 1 || backgroundMode == 2)
{
    if (rRow >= 0)
    {
        if (region [rRow] [rCol].iD == 20 || region [rRow] [rCol].iD == 21)
        {
            boolean b = region [5] [10].getLabel ().getText ().equals ("Scratch");
            if (region [rRow] [rCol].setValues (x, y, b))
            {
                if (!b)
                    changeParameters (region [rRow] [rCol].iD, region [rRow] [rCol].value);
                else
                    doScratch ();
            }
        }
        else if (region [rRow] [rCol].setValues (x, y))
            changeParameters (region [rRow] [rCol].iD, region [rRow] [rCol].value);
    }
    else if (region [5] [0].equals (x, y))
    {
        if (region [5] [0].setValues (x, y,
                    region [5] [10].getLabel ().getText ().equals ("Scratch")))
            changeParameters (region [5] [0].iD, region [5] [0].value);
        activeNode = true;
        rRow = 5;
        rCol = 0;
    }
    else if (region [5] [1].equals (x, y))
    {
        if (backgroundMode != 1)
        {
            region [4] [5].setValues (region [5] [9].value);
            region [5] [9].setValues (region [0] [5].value);
            playPauseLabel [0] [2].setVisible (false);
            playPauseLabel [1] [2].setVisible (false);
            if (state1 == 1)
                playPauseLabel [0] [2].setVisible (true);
            else if (state1 == 2)
                playPauseLabel [1] [2].setVisible (true);
            if (!region [5] [10].getLabel ().getText ().equals ("Scratch"))
                setJogWheelTaskLabel (state1);
        }
        region [5] [0].iD = 20;
        region [5] [7].iD = 23;
        region [5] [8].iD = 24;
        region [5] [9].iD = 11;
        backgroundMode = 1;
        deckIndLabel.setLocation (110, 0);
        rRow = 5;
        rCol = 1;
    }
    else if (region [5] [2].equals (x, y))
    {
        if (backgroundMode != 2)
        {
            region [0] [5].setValues (region [5] [9].value);
            region [5] [9].setValues (region [4] [5].value);
            playPauseLabel [0] [2].setVisible (false);
            playPauseLabel [1] [2].setVisible (false);
            if (state2 == 1)
                playPauseLabel [0] [2].setVisible (true);
            else if (state2 == 2)
                playPauseLabel [1] [2].setVisible (true);
            if (!region [5] [10].getLabel ().getText ().equals ("Scratch"))
                setJogWheelTaskLabel (state2);
        }
        region [5] [0].iD = 21;
        region [5] [7].iD = 25;
        region [5] [8].iD = 26;
        region [5] [9].iD = 12;
        backgroundMode = 2;
        deckIndLabel.setLocation (830, 0);
        rRow = 5;
        rCol = 2;
    }
    else if (region [5] [3].equals (x, y))
    {
        if (backgroundMode == 1)
            region [0] [5].setValues (region [5] [9].value);
        else if (backgroundMode == 2)
            region [4] [5].setValues (region [5] [9].value);
        backgroundMode = 0;
        jogWheelModeLabel.setVisible (false);
        deckIndLabel.setVisible (false);
        region [5] [0].getLabel ().setVisible (false);
        region [5] [9].getLabel ().setVisible (false);
        region [5] [10].getLabel ().setVisible (false);
        playPauseLabel [0] [2].setVisible (false);
        playPauseLabel [1] [2].setVisible (false);
        rRow = 5;
        rCol = 3;
    }
```

```
else if (region [5] [4].equals (x, y))
{
    if (backgroundMode == 1)
    {
        playPauseStop (2);
        if (!region [5] [10].getLabel ().getText ().equals ("Scratch"))
            setJogWheelTaskLabel (state1);
    }
    else
    {

        playPauseStop (3);
        if (!region [5] [10].getLabel ().getText ().equals ("Scratch"))
            setJogWheelTaskLabel (state2);
    }
    playPauseLabel [0] [2].setVisible (false);
    playPauseLabel [1] [2].setVisible (false);
    rRow = 5;
    rCol = 4;
}
else if (region [5] [5].equals (x, y))
{
    if (backgroundMode == 1)
    {
        setCuePos (true);
        if (!region [5] [10].getLabel ().getText ().equals ("Scratch"))
            setJogWheelTaskLabel (state1);
    }
    else
    {
        setCuePos (false);
        if (!region [5] [10].getLabel ().getText ().equals ("Scratch"))
            setJogWheelTaskLabel (state2);
    }
    playPauseLabel [0] [2].setVisible (false);
    playPauseLabel [1] [2].setVisible (true);
    rRow = 5;
    rCol = 5;
}
else if (region [5] [6].equals (x, y))
{
    if (backgroundMode == 1)
    {
        playPauseStop (0);
        if (state1 == 1)
        {
            playPauseLabel [1] [2].setVisible (false);
            playPauseLabel [0] [2].setVisible (true);
        }
        else if (state1 == 2)
        {
            playPauseLabel [0] [2].setVisible (false);
            playPauseLabel [1] [2].setVisible (true);
        }
        if (!region [5] [10].getLabel ().getText ().equals ("Scratch"))
            setJogWheelTaskLabel (state1);
    }
    else
    {
        playPauseStop (1);
        if (state2 == 1)
        {
            playPauseLabel [1] [2].setVisible (false);
            playPauseLabel [0] [2].setVisible (true);
        }
        else if (state2 == 2)
        {
            playPauseLabel [0] [2].setVisible (false);
            playPauseLabel [1] [2].setVisible (true);
        }

        if (!region [5] [10].getLabel ().getText ().equals ("Scratch"))
            setJogWheelTaskLabel (state2);
    }
    rRow = 5;
    rCol = 6;
}
else if (region [5] [7].equals (x, y))
{
    if (backgroundMode == 1)
    {
        double t = rate1 - (.04 * rate1);
        setFreq (true, t);
    }
    else
    {
        double t = rate2 - (.04 * rate2);
        setFreq (false, t);
    }
    activeNode = true;
    rRow = 5;
    rCol = 7;
}
else if (region [5] [8].equals (x, y))
{
    if (backgroundMode == 1)
    {
        double t = rate1 + (.04 * rate1);
        setFreq (true, t);
    }
    else
    {
        double t = rate2 + (.04 * rate2);
        setFreq (false, t);
    }
    activeNode = true;
    rRow = 5;
    rCol = 8;
}
else if (region [5] [9].equals (x, y))
{
    if (region [5] [9].setValues (x, y))
        changeParameters (region [5] [9].iD, region [5] [9].value);
    rRow = 5;
    rCol = 9;
}
else if (region [5] [10].equals (x, y))
{
    if (region [5] [10].getLabel ().getText ().equals ("Scratch"))
    {
        setJogWheelTaskLabel (backgroundMode == 1 ? state1:
        state2);
        region [5] [0].setValues (50);
    }
    else
        region [5] [10].getLabel ().setText ("Scratch");
    rRow = 5;
    rCol = 10;
}
```

```
        else if (region [5] [11].equals (x, y))
        {
            if (region [5] [9].setValues (region [5] [9].value + 1))
                changeParameters (region [5] [9].iD, region [5] [9].value);
            rRow = 5;
            rCol = 11;
        }
        else if (region [5] [12].equals (x, y))
        {
            if (region [5] [9].setValues (region [5] [9].value - 1))
                changeParameters (region [5] [9].iD, region [5] [9].value);
            rRow = 5;
            rCol = 12;
        }
    }
}
else //if (!ctrlMove)
{
    labelFree.setLocation (x + fingerPos, y);

    if (rRow >= 0)
    {
        if (activeNode)
        {
            if (region [rRow] [rCol].iD == 20)
            {
                if (region [5] [10].getLabel ().getText ().equals ("Scratch"))
                {
                    if (forward1 == false)
                        reverseTrack (0, false);
                    else if (state1 == 1 && forward1 == true)
                        setFreq (true, rate1);
                    else if (state1 == 2)
                        setFreq (true, 0);
                }
                else if (region [rRow] [rCol].setValues (50))
                    changeParameters (region [rRow] [rCol].iD,
                            region [rRow] [rCol].value);
            }
            else if (region [rRow] [rCol].iD == 21)
            {
                if (region [5] [10].getLabel ().getText ().equals ("Scratch"))
                {
                    if (forward2 == false)
                        reverseTrack (4, false);
                    else if (state2 == 1 && forward2 == true)
                        setFreq (false, rate2);
                    else if (state2 == 2)
                        setFreq (false, 0);
                }
                else if (region [rRow] [rCol].setValues (50))
                    changeParameters (region [rRow] [rCol].iD,
                            region [rRow] [rCol].value);
            }
            else if (rCol == 2)
                reverseTrack (rRow, false);
            else if (rCol == 3)
                fastForward (rRow, false);
            else if (region [rRow] [rCol].iD == 23 ||
                    region [rRow] [rCol].iD == 24)
                setFreq (true, rate1);
            else if (region [rRow] [rCol].iD == 25 ||
                    region [rRow] [rCol].iD == 26)
                setFreq (false, rate2);
            activeNode = false;
        }
        rRow = -1;
    }
}
```

```
        }
        catch (Exception e)
        {
            System.err.println ("midiIn " + e);
        }
    }

    public void doScratch ()
    {
        int border = 10;
        scratchDiffPos += (int) (0.8 *
                (((JogWheel) region [5] [0]).diffPos - scratchDiffPos));

        if (backgroundMode == 1)
        {
            if (scratchDiffPos < border && scratchDiffPos > -border)
                setFreq (true, 0);
            else if (scratchDiffPos > border)
            {
                if (forward1)
                    setFreq (true, rate1);
                else
                    reverseTrack (0, false);
            }
            else if (scratchDiffPos < -border)
            {
                if (!forward1)
                    setFreq (true, rate1);
                else
                    reverseTrack (0, true);
            }
        }
        else
        {
            if (scratchDiffPos < border && scratchDiffPos > -border)
                setFreq (false, 0);
            else if (scratchDiffPos > border)
            {
                if (forward2)
                    setFreq (false, rate2);
                else
                    reverseTrack (4, false);
            }
            else if (scratchDiffPos < -border)
            {
                if (!forward2)
                    setFreq (false, rate2);
                else
                    reverseTrack (4, true);
            }
        }
    }
}


boolean skipable1 = true;
boolean skipable2 = true;
// ************************************************************************
public void changeParameters (int regionID, int regionValue)
{
    if (regionID == 20)
    {
        if (state1 == 2)
        {
```

```
			if (regionValue == 50)
			{
				setFreq (true, 0);
				if (!forward1)
					reverseTrack (0, false);
				skipable1 = true;
			}
			else if (regionValue < 50 && regionValue > 0)
			{
				int t = 50 - regionValue;
				t *= 900;
				setFreq (true, t);
				if (forward1)
					reverseTrack (0, true);
				skipable1 = true;
			}
			else if (regionValue > 50 && regionValue < 100)
			{
				int t = regionValue - 50;
				t *= 900;
				setFreq (true, t);
				if (!forward1)
					reverseTrack (0, false);
				skipable1 = true;
			}
			else if (regionValue == 0)
			{
				if (skipable1)
				{
					int t = getPosition (true, 2);
					t -= 441000;
					if (t < 0)
						t = 1;
					setPosition (true, t);
					reverseTrack (0, true);
					skipable1 = false;
				}
			}
			else if (regionValue == 100)
			{
				if (skipable1)
				{
					int t = getPosition (true, 2);
					t += 441000;
					if (t >= sampleNumFrames1)
						t = sampleNumFrames1 - 2;
					setPosition (true, t);
					forwardTrack (true, t);
					skipable1 = false;
				}
			}
		}
		else
		{
			int t = (int) ((rate1 / 50) * regionValue);
			setFreq (true, t);
		}
	}
	else if (regionID == 21)
	{
		if (state2 == 2)
		{
			if (regionValue == 50)
			{
				setFreq (false, 0);
				if (!forward2)
					reverseTrack (4, false);
				skipable2 = true;
			}
			else if (regionValue < 50 && regionValue > 0)
			{
				int t = 50 - regionValue;
				t *= 900;
				setFreq (false, t);
				if (forward2)
					reverseTrack (4, true);
				skipable2 = true;
			}
			else if (regionValue > 50 && regionValue < 100)
			{
				int t = regionValue - 50;
				t *= 900;
				setFreq (false, t);
				if (!forward2)
					reverseTrack (4, false);
				skipable2 = true;
			}
			else if (regionValue == 0)
			{
				if (skipable2)
				{
					int t = getPosition (false, 2);
					t -= 441000;
					if (t < 0)
						t = 1;
					setPosition (false, t);
					reverseTrack (4, true);
					skipable2 = false;
				}
			}
			else if (regionValue == 100)
			{
				if (skipable2)
				{
					int t = getPosition (false, 2);
					t += 441000;
					if (t >= sampleNumFrames2)
						t = sampleNumFrames2 - 2;
					setPosition (false, t);
					forwardTrack (false, t);
					skipable2 = false;
				}
			}
		}
		else
		{
			int t = (int) ((rate2 / 50) * regionValue);
			setFreq (false, t);
		}
	}
	else if (regionID == 0)
	{
		try
		{
			if (regionValue < 50)
			{
				double t = regionValue * .02;
				multiplyLeft1.inputB.set (1);
				multiplyRight1.inputB.set (1);
				multiplyLeft2.inputB.set (t);
				multiplyRight2.inputB.set (t);
			}
```

```
                else if (regionValue > 50)
                {
                        double t = (100 - regionValue) * .02;
                        multiplyLeft1.inputB.set (t);
                        multiplyRight1.inputB.set (t);
                        multiplyLeft2.inputB.set (1);
                        multiplyRight2.inputB.set (1);
                }
                else
                {
                        multiplyLeft1.inputB.set (1);
                        multiplyRight1.inputB.set (1);
                        multiplyLeft2.inputB.set (1);
                        multiplyRight2.inputB.set (1);
                }
        }
        catch (SynthException e)
        {
                System.err.println ("Crossfade");
                SynthAlert.showError (this, e);
        }
}
else if (regionID == 1)
{
        double t = regionValue * .030303;
        try
        {
                fLS1L.amplitude.set (t);
                fLS1R.amplitude.set (t);
        }
        catch (SynthException e)
        {
                System.err.println ("Fader 1");
                SynthAlert.showError (this, e);
        }
}
else if (regionID == 2)
{
        double t = regionValue * .030303;
        try
        {
                fLS2L.amplitude.set (t);
                fLS2R.amplitude.set (t);
        }
        catch (SynthException e)
        {
                System.err.println ("Fader 2");
                SynthAlert.showError (this, e);
        }
}
else if (regionID == 3)
{
        double t = frequencyArray [regionValue * 127 / 100] + 12;
        try
        {
                fLP1L.frequency.set (t);
                fLP1R.frequency.set (t);
        }
        catch (SynthException e)
        {
                System.err.println ("LP 1");
                SynthAlert.showError (this, e);
        }
}
else if (regionID == 4)
{
        double t;
        if (regionValue < 50)
                t = regionValue * .01874 + .0631;
        else
                t = (regionValue - 50) * .0199 + 1;
        try
        {
                fHS1L.gain.set (t);
                fHS1R.gain.set (t);
        }
        catch (SynthException e)
        {
                System.err.println ("HS 1");
                SynthAlert.showError (this, e);
        }
}
else if (regionID == 5)
{
        double t;
        if (regionValue < 50)
                t = regionValue * .01874 + .0631;
        else
                t = (regionValue - 50) * .0199 + 1;
        try
        {
                fPEQ1L.gain.set (t);
                fPEQ1R.gain.set (t);
        }
        catch (SynthException e)
        {
                System.err.println ("PEQ 1");
                SynthAlert.showError (this, e);
        }
}
else if (regionID == 6)
{
        double t;
        if (regionValue < 50)
                t = regionValue * .01874 + .0631;
        else
                t = (regionValue - 50) * .0199 + 1;
        try
        {
                fLS1L.gain.set (t);
                fLS1R.gain.set (t);
        }
        catch (SynthException e)
        {
                System.err.println ("LS 1");
                SynthAlert.showError (this, e);
        }
}
else if (regionID == 7)
{
        double t = frequencyArray [regionValue * 127 / 100] + 12;
        try
        {
                fLP2L.frequency.set (t);
                fLP2R.frequency.set (t);
        }
        catch (SynthException e)
        {
                System.err.println ("LP 2");
                SynthAlert.showError (this, e);
        }
}
```

```
else if (regionID == 8)
{
	double t;
	if (regionValue < 50)
		t = regionValue * .01874 + .0631;
	else
		t = (regionValue - 50) * .0199 + 1;
	try
	{
		fHS2L.gain.set (t);
		fHS2R.gain.set (t);
	}
	catch (SynthException e)
	{
		System.err.println ("HS 2");
		SynthAlert.showError (this, e);
	}
}
else if (regionID == 9)
{
	double t;
	if (regionValue < 50)
		t = regionValue * .01874 + .0631;
	else
		t = (regionValue - 50) * .0199 + 1;
	try
	{
		fPEQ2L.gain.set (t);
		fPEQ2R.gain.set (t);
	}
	catch (SynthException e)
	{
		System.err.println ("PEQ 2");
		SynthAlert.showError (this, e);
	}
}
else if (regionID == 10)
{
	double t;
	if (regionValue < 50)
		t = regionValue * .01874 + .0631;
	else
		t = (regionValue - 50) * .0199 + 1;
	try
	{
		fLS2L.gain.set (t);
		fLS2R.gain.set (t);
	}
	catch (SynthException e)
	{
		System.err.println ("LS 2");
		SynthAlert.showError (this, e);
	}
}
else if (regionID == 11)
{
	rate1 = 47628 - (regionValue * 35.28);
	double tBpm = (rate1 / 44100) * bpm1;
	if (state1 == 1)
		setFreq (true, rate1);
	bpmLabel1.setText (new DecimalFormat ("000.00").format (tBpm));
}
else if (regionID == 12)
{
	rate2 = 47628 - (regionValue * 35.28);
	double tBpm = (rate2 / 44100) * bpm2;
	if (state2 == 1)
		setFreq (false, rate2);
	bpmLabel2.setText (new DecimalFormat ("000.00").format (tBpm));
}
else if (regionID == 13)
{
	double t = regionValue * .02;
	try
	{
		reader1L.amplitude.set (t);
		reader1R.amplitude.set (t);
		if (region [1] [6].value == 1)
			reader1CueR.amplitude.set (t);
	}
	catch (SynthException e)
	{
		System.err.println ("Gain 1");
		SynthAlert.showError (this, e);
	}
}
else if (regionID == 14)
{
	double t = regionValue * .02;
	try
	{
		reader2L.amplitude.set (t);
		reader2R.amplitude.set (t);

		if (region [3] [6].value == 1)
			reader2CueR.amplitude.set (t);
	}
	catch (SynthException e)
	{
		System.err.println ("Gain 2");
		SynthAlert.showError (this, e);
	}

}
else if (regionID == 18)
{
	double t = regionValue * .02;
	try
	{
		mU1Cue1.inputB.set (t);
		mU2Cue1.inputB.set (t);
	}
	catch (SynthException e)
	{
		System.err.println ("Cue Level");
		SynthAlert.showError (this, e);
	}
}
```

```
        else if (regionID == 19)
        {
                double t = regionValue * .02;
                try
                {
                        mU1Cue2.inputB.set (2 - t);
                        mU2Cue2.inputB.set (t);
                }
                catch (SynthException e)
                {
                        System.err.println ("Cue Balance");
                        SynthAlert.showError (this, e);
                }
        }
}

// ***************************************************************************
public void fastForward (int k, boolean deck1)
{
        try
        {
                if (deck1 && k == 0)
                        setFreq (true, 88200);
                else if (deck1 && k == 4)
                        setFreq (false, 88200);
                else if (!deck1 && k == 0)
                {
                        if (state1 == 1)
                                setFreq (true, rate1);
                        else
                                setFreq (true, 0);
                }
                else if (!deck1 && k == 4)
                {
                        if (state2 == 1)
                                setFreq (false, rate2);
                        else
                                setFreq (false, 0);
                }
        }
        catch (Exception e)
        {
                System.err.println ("fastForward");
        }
}

// ***************************************************************************
public void setFreq (boolean deck1, double f)
{
        try
        {
                if (deck1)
                {
                        reader1L.rate.set (f);
                        reader1R.rate.set (f);
                        reader1CueR.rate.set (f);
                }
                else
                {
                        reader2L.rate.set (f);
                        reader2R.rate.set (f);
                        reader2CueR.rate.set (f);
                }
        }
        catch (SynthException e)
        {
                System.err.println ("setFreq");
                SynthAlert.showError (this, e);
        }
}

// ********************************
int cuePos1, cuePos2;
public void setCuePos (boolean deck1)
{
        if (deck1)
        {
                if (state1 == 1)
                {
                        playPauseStop (0);
                        setPosition (true, cuePos1);
                        forwardTrack (true, cuePos1);
                }
                else if (state1 == 2)
                {
                        cuePos1 = getPosition (true, 2);
                }
        }
        else
        {
                if (state2 == 1)
                {
                        playPauseStop (1);
                        setPosition (false, cuePos2);
                        forwardTrack (false, cuePos2);
                }
                else if (state2 == 2)
                {
                        cuePos2 = getPosition (false, 2);
                }
        }
}

// ********************************
int position1 = 0;
int position2 = 0;
int getNumFramesMovedOld1 = 0;
int getNumFramesMovedOld2 = 0;
boolean forward1 = true;
boolean forward2 = true;

public synchronized void setPosition (boolean deck1, int t)
{
        if (deck1)
        {
                getNumFramesMovedOld1 = reader1L.samplePort.getNumFramesMoved ();
                position1 = t;
        }
        else
        {
                getNumFramesMovedOld2 = reader2L.samplePort.getNumFramesMoved ();
                position2 = t;
        }
}
```

```
public synchronized int getPosition (boolean deck1, int c)
{
    if (deck1)
    {
        try
        {
            if (c == 0)
                forward1 = false;
            else if (c == 1)
                forward1 = true;

            int tempF = reader1L.samplePort.getNumFramesMoved ();
            int numFramesMoved = tempF - getNumFramesMovedOld1;
            getNumFramesMovedOld1 = tempF;

            if ((c == 2 && forward1) || c == 0)
                position1 += numFramesMoved;
            else if ((c == 2 && !forward1) || c == 1)
                position1 -= numFramesMoved;

        }
        catch (Exception e)
        {
            System.err.println ("getPosition");
        }
        return position1;
    }
    else
    {
        if (c == 0)
            forward2 = false;
        else if (c == 1)
            forward2 = true;

        int tempF = reader2L.samplePort.getNumFramesMoved ();
        int numFramesMoved = tempF - getNumFramesMovedOld2;
        getNumFramesMovedOld2 = tempF;

        if ((c == 2 && forward2) || c == 0)
            position2 += numFramesMoved;
        else if ((c == 2 && !forward2) || c == 1)
            position2 -= numFramesMoved;

        return position2;
    }
}


public void forwardTrack (boolean deck1, int t)
{
    if (deck1)
    {
        reader1L.samplePort.clear ();
        reader1R.samplePort.clear ();
        reader1CueR.samplePort.clear ();

        int whichSample = t / sampleSize;
        int f1 = t % sampleSize;
        int numberOfFrames = sampleRev1L [whichSample].getNumFrames () - f1;

        if (numberOfFrames > 0)
        {
            reader1L.samplePort.queue (sample1L [whichSample], f1, numberOfFrames);
            reader1R.samplePort.queue (sample1R [whichSample], f1, numberOfFrames);
            reader1CueR.samplePort.queue (sample1CueR [whichSample], f1, numberOfFrames);
        }

        reader1L.start ();
        reader1R.start ();
        reader1CueR.start ();

        for (int i = whichSample + 1 ; i < sample1L.length ; i++)
        {
            reader1L.samplePort.queue (sample1L [i]);
            reader1R.samplePort.queue (sample1R [i]);
            reader1CueR.samplePort.queue (sample1CueR [i]);
        }
    }
    else
    {
        reader2L.samplePort.clear ();
        reader2R.samplePort.clear ();
        reader2CueR.samplePort.clear ();

        int whichSample = t / sampleSize;
        int f1 = t % sampleSize;
        int numberOfFrames = sampleRev2L [whichSample].getNumFrames () - f1;

        if (numberOfFrames > 0)
        {
            reader2L.samplePort.queue (sample2L [whichSample], f1, numberOfFrames);
            reader2R.samplePort.queue (sample2R [whichSample], f1, numberOfFrames);
            reader2CueR.samplePort.queue (sample2CueR [whichSample], f1, numberOfFrames);
        }

        reader2L.start ();
        reader2R.start ();
        reader2CueR.start ();

        for (int i = whichSample + 1 ; i < sample2L.length ; i++)
        {
            reader2L.samplePort.queue (sample2L [i]);
            reader2R.samplePort.queue (sample2R [i]);
            reader2CueR.samplePort.queue (sample2CueR [i]);
        }
    }

}


//  ***************************************************************************
public void reverseTrack (int k, boolean movingForward)
{
    try
    {
        if (movingForward && k == 0)
        {
            reader1L.stop ();
            reader1R.stop ();
            reader1CueR.stop ();

            int tempP = getPosition (true, 0);

            int whichSample = tempP / sampleSize;
            int numberOfFrames = tempP % sampleSize;
            if (numberOfFrames == 0 && whichSample > 0)
                whichSample--;
            int frame1 = sample1L [whichSample].getNumFrames () - numberOfFrames;

            reader1L.samplePort.clear ();
            reader1R.samplePort.clear ();
            reader1CueR.samplePort.clear ();
            setFreq (true, rate1);
            if (tempP > 0 && numberOfFrames > 0)
            {
                reader1L.samplePort.queue (sampleRev1L [whichSample], frame1, numberOfFrames);
                reader1R.samplePort.queue (sampleRev1R [whichSample], frame1, numberOfFrames);
                reader1CueR.samplePort.queue (sampleRev1CueR [whichSample], frame1, numberOfFrames);
            }

            reader1L.start ();
            reader1R.start ();
            reader1CueR.start ();

            for (int i = whichSample - 1 ; i >= 0 ; i--)
            {
                reader1L.samplePort.queue (sampleRev1L [i]);
                reader1R.samplePort.queue (sampleRev1R [i]);
                reader1CueR.samplePort.queue (sampleRev1CueR [i]);
            }
        }
        else if (movingForward && k == 4)
        {
            reader2L.stop ();
            reader2R.stop ();
            reader2CueR.stop ();
            int tempP = getPosition (false, 0);
            int whichSample = tempP / sampleSize;
            int numberOfFrames = tempP % sampleSize;
            if (numberOfFrames == 0 && whichSample > 0)
                whichSample--;
            int frame1 = sample2L [whichSample].getNumFrames () - numberOfFrames;

            reader2L.samplePort.clear ();
            reader2R.samplePort.clear ();
            reader2CueR.samplePort.clear ();
            setFreq (false, rate2);

            if (tempP > 0 && numberOfFrames > 0)
            {
                reader2L.samplePort.queue (sampleRev2L [whichSample], frame1, numberOfFrames);
                reader2R.samplePort.queue (sampleRev2R [whichSample], frame1, numberOfFrames);
                reader2CueR.samplePort.queue (sampleRev2CueR [whichSample], frame1, numberOfFrames);
            }

            reader2L.start ();
            reader2R.start ();
            reader2CueR.start ();

            for (int i = whichSample - 1 ; i >= 0 ; i--)
            {
                reader2L.samplePort.queue (sampleRev2L [i]);
                reader2R.samplePort.queue (sampleRev2R [i]);
                reader2CueR.samplePort.queue (sampleRev2CueR [i]);
            }
        }
        else if (!movingForward && k == 0)
        {
            reader1L.stop ();
            reader1R.stop ();
            reader1CueR.stop ();
            int tempP = getPosition (true, 1);
            forwardTrack (true, tempP);

            if (state1 == 1)
                setFreq (true, rate1);
            else
                setFreq (true, 0);
        }
        else if (!movingForward && k == 4)
        {
            reader2L.stop ();
            reader2R.stop ();
            reader2CueR.stop ();
            int tempP = getPosition (false, 1);
            forwardTrack (false, tempP);

            if (state2 == 1)
                setFreq (false, rate2);
            else
                setFreq (false, 0);
        }
    }
    catch (SynthException e)
    {
        System.err.println ("Synth reverseTrack");
        SynthAlert.showError (this, e);
    }
    catch (Exception e)
    {
        System.err.println ("reverseTrack");
    }
}


// ************************************************************************
int state1 = 0; //0 stop, 1 play, 2 pause
int state2 = 0;

public void playPauseStop (int k)
{
    try
    {
        if (k == 0)
        {
            if (state1 == 0)
            {
                setFreq (true, rate1);
                queueSamples (true);
                state1 = 1;
            }
            else if (state1 == 1)
            {
                setFreq (true, 0);
                state1 = 2;
            }
            else
            {
                setFreq (true, rate1);
                state1 = 1;
            }

            if (state1 == 1)
            {
                playPauseLabel [1] [0].setVisible (false);
                playPauseLabel [0] [0].setVisible (true);
            }
            else
            {
                playPauseLabel [0] [0].setVisible (false);
                playPauseLabel [1] [0].setVisible (true);
            }
        }
        else if (k == 1)
        {
            if (state2 == 0)
            {
                setFreq (false, rate2);
                queueSamples (false);
                state2 = 1;
            }
            else if (state2 == 1)
            {
                setFreq (false, 0);
                state2 = 2;
            }
            else
            {
                setFreq (false, rate2);
                state2 = 1;
            }

            if (state2 == 1)
            {
                playPauseLabel [1] [1].setVisible (false);
                playPauseLabel [0] [1].setVisible (true);
            }
            else
            {
                playPauseLabel [0] [1].setVisible (false);
                playPauseLabel [1] [1].setVisible (true);
            }
        }
        else if (k == 2)
        {
            clearQueue (true);
            setFreq (true, rate1);
            state1 = 0;
            playPauseLabel [0] [0].setVisible (false);
            playPauseLabel [1] [0].setVisible (false);
        }
        else
        {
            clearQueue (false);
            setFreq (false, rate2);
            state2 = 0;
            playPauseLabel [0] [1].setVisible (false);
            playPauseLabel [1] [1].setVisible (false);
        }
    }
    catch (Exception e)
    {
        System.err.println ("playPauseStop");
    }
}


public void setJogWheelTaskLabel (int s)
{
    if (s == 0 || s == 1)
        region [5] [10].getLabel ().setText ("Pitch Bend");
    else if (s == 2)
        region [5] [10].getLabel ().setText ("Search");
}


// ************************************************************************
public void moveIndPlayListLabel (int i)
{
    int t = whichTrack % 5;
    if (t == 0)
        indPlayListLabel.setLocation (175, 699);
    else if (t == 1)
        indPlayListLabel.setLocation (305, 699);
    else if (t == 2)
        indPlayListLabel.setLocation (435, 699);
    else if (t == 3)
        indPlayListLabel.setLocation (565, 699);
    else
        indPlayListLabel.setLocation (695, 699);
}


// ************************************************************************
public void sortPlayList ()
{
    for (int i = playList.length - 1 ; i > 0 ; i--)
        for (int j = 0 ; j < i ; j++)
            if (playList [j].compareTo (playList [j + 1]) > 0)
            {
                String temp = playList [j];
                playList [j] = playList [j + 1];
                playList [j + 1] = temp;
            }
}


// ************************************************************************
public void writePlayList ()
{
    int group = whichTrack / 5;

    for (int i = 0 ; i < samplesDisplayLabel.length ; i++)
        samplesDisplayLabel [i].setText ("");

    for (int i = 0 ; i < samplesDisplayLabel.length ; i++)
    {
        if (group * 5 + i < playList.length)
            samplesDisplayLabel [i].setText (playList [group * 5 + i]);
    }
}


// ************************************************************************
SynthSample [] sample1L;
SynthSample [] sample1R;
SynthSample [] sample1CueR;
SynthSample [] sampleRev1L;
SynthSample [] sampleRev1R;
SynthSample [] sampleRev1CueR;
SynthSample [] sample2L;
SynthSample [] sample2R;
SynthSample [] sample2CueR;
SynthSample [] sampleRev2L;
SynthSample [] sampleRev2R;
SynthSample [] sampleRev2CueR;
int sampleNumFrames1 = 0;
int sampleNumFrames2 = 0;

public void loadSample (boolean deck1, String fileName)
{
    DecimalFormat f = new DecimalFormat ("000");
    try
    {
        String bpmString = "120";
        bpmString = fileName.substring (fileName.indexOf ('_') + 1);
        int numberOfFiles = 0;
        int firstFile = 0;

        fileName = "Samples/" + fileName;
        System.out.println (fileName);

        String tempString = f.format (firstFile);
```

```
while (!new File (fileName + "/L" + tempString + ".wav").exists ())
    tempString = f.format (++firstFile);

int w = firstFile;
System.out.println (bpmString + " " + firstFile);

tempString = f.format (w);
while (new File (fileName + "/L" + tempString + ".wav").exists ())
{
    tempString = f.format (++w);
    numberOfFiles++;
}

if (deck1)
{
    sample1L = new SynthSample [numberOfFiles];
    sample1R = new SynthSample [numberOfFiles];
    sample1CueR = new SynthSample [numberOfFiles];
    sampleRev1L = new SynthSample [numberOfFiles];
    sampleRev1R = new SynthSample [numberOfFiles];
    sampleRev1CueR = new SynthSample [numberOfFiles];

    for (int i = 0 ; i < numberOfFiles ; i++)
    {
        tempString = f.format (i + firstFile);
        //System.out.println ((i + 1) + "/" + numberOfFiles);
        sample1L [i] = new SynthSampleWAV (synthContext1);
        InputStream stream = (InputStream) (new FileInputStream (fileName + "/L" + tempString + ".wav"));
        sample1L [i].load (stream);
        sample1L [i].byteData = null;

        int nF = sample1L [i].getNumFrames ();
        short [] s1 = new short [nF];

        sample1L [i].read (0, s1, 0, nF);

        int t = nF - 1;
        for (int m = 0 ; m < nF / 2 ; m++)
        {
            short temp = s1 [m];
            s1 [m] = s1 [t - m];
            s1 [t - m] = temp;
        }

        sampleRev1L [i] = new SynthSampleWAV (synthContext1);
        sampleRev1L [i].allocate (nF, 1);
        sampleRev1L [i].write (s1);
        sampleRev1L [i].byteData = null;

        sample1R [i] = new SynthSampleWAV (synthContext1);
        sample1CueR [i] = new SynthSampleWAV (synthContext2);
        stream = (InputStream) (new FileInputStream (fileName + "/R" + tempString + ".wav"));
        sample1R [i].load (stream);
        stream = (InputStream) (new FileInputStream (fileName + "/R" + tempString + ".wav"));
        sample1CueR [i].load (stream);
        sample1R [i].byteData = null;
        sample1CueR [i].byteData = null;

        sample1R [i].read (0, s1, 0, nF);

        for (int m = 0 ; m < nF / 2 ; m++)
        {
            short temp = s1 [m];
            s1 [m] = s1 [t - m];
            s1 [t - m] = temp;
        }

        sampleRev1R [i] = new SynthSampleWAV (synthContext1);
        sampleRev1CueR [i] = new SynthSampleWAV (synthContext2);
        sampleRev1R [i].allocate (nF, 1);
        sampleRev1CueR [i].allocate (nF, 1);
        sampleRev1R [i].write (s1);
        sampleRev1CueR [i].write (s1);
        sampleRev1R [i].byteData = null;
        sampleRev1CueR [i].byteData = null;
    }
    sampleNumFrames1 = (numberOfFiles - 1) * sampleSize + sample1L [numberOfFiles - 1].getNumFrames ();
}
else
{

    sample2L = new SynthSample [numberOfFiles];
    sample2R = new SynthSample [numberOfFiles];
    sample2CueR = new SynthSample [numberOfFiles];
    sampleRev2L = new SynthSample [numberOfFiles];
    sampleRev2R = new SynthSample [numberOfFiles];
    sampleRev2CueR = new SynthSample [numberOfFiles];
    for (int i = 0 ; i < numberOfFiles ; i++)
    {
        tempString = f.format (i + firstFile);
        //System.out.println ((i + 1) + "/" + numberOfFiles);
        sample2L [i] = new SynthSampleWAV (synthContext1);
        InputStream stream = (InputStream) (new FileInputStream (fileName + "/L" + tempString + ".wav"));
        sample2L [i].load (stream);
        sample2L [i].byteData = null;

        int nF = sample2L [i].getNumFrames ();
        short [] s1 = new short [nF];

        sample2L [i].read (0, s1, 0, nF);

        int t = nF - 1;
        for (int m = 0 ; m < nF / 2 ; m++)
        {
            short temp = s1 [m];
            s1 [m] = s1 [t - m];
            s1 [t - m] = temp;
        }

        sampleRev2L [i] = new SynthSampleWAV (synthContext1);
        sampleRev2L [i].allocate (nF, 1);
        sampleRev2L [i].write (s1);
        sampleRev2L [i].byteData = null;

        sample2R [i] = new SynthSampleWAV (synthContext1);
        sample2CueR [i] = new SynthSampleWAV (synthContext2);
        stream = (InputStream) (new FileInputStream (fileName + "/R" + tempString + ".wav"));
        sample2R [i].load (stream);
        stream = (InputStream) (new FileInputStream (fileName + "/R" + tempString + ".wav"));
        sample2CueR [i].load (stream);
        sample2R [i].byteData = null;
        sample2CueR [i].byteData = null;

        sample2R [i].read (0, s1, 0, nF);

        for (int m = 0 ; m < nF / 2 ; m++)
        {
            short temp = s1 [m];
            s1 [m] = s1 [t - m];
            s1 [t - m] = temp;
        }

        sampleRev2R [i] = new SynthSampleWAV (synthContext1);
        sampleRev2CueR [i] = new SynthSampleWAV (synthContext2);
        sampleRev2R [i].allocate (nF, 1);
        sampleRev2CueR [i].allocate (nF, 1);
        sampleRev2R [i].write (s1);
        sampleRev2CueR [i].write (s1);
```

```
                sampleRev2R [i].byteData = null;
                sampleRev2CueR [i].byteData = null;
            }
            sampleNumFrames2 = (numberOfFiles - 1) * sampleSize + sample2L [numberOfFiles - 1].getNumFrames ();
        }

        System.out.println ("LOADED");

        if (deck1)
        {
            bpm1 = 0;
            bpm1 = Double.parseDouble (bpmString);
            double tBpm = (rate1 / 44100) * bpm1;
            bpmLabel1.setText (new DecimalFormat ("000.00").format (tBpm));
            timerThread.display1 ();
            cuePos1 = 0;
        }
        else
        {
            bpm2 = 0;
            bpm2 = Double.parseDouble (bpmString);
            double tBpm = (rate2 / 44100) * bpm2;
            bpmLabel2.setText (new DecimalFormat ("000.00").format (tBpm));
            timerThread.display2 ();
            cuePos2 = 0;
        }
        thread01.midiport.resetInput ();
    }
    catch (SynthException e)
    {
        System.err.println ("Synth loadSample");
        SynthAlert.showError (this, e);
    }
    catch (IOException e)
    {
        SynthAlert.showError (this, e);
    }
    catch (Exception e)
    {
        System.err.println ("loadSample");
    }
}

// ************************************************************************
public void queueSamples (boolean deck1)
{
    try
    {
        if (deck1)
        {
            position1 = 0;
            getNumFramesMovedOld1 = 0;
            for (int i = 0 ; i < 1 ; i++)
                for (int j = 0 ; j < sample1L.length ; j++)
                {
                    reader1L.samplePort.queue (sample1L [j]);
                    reader1R.samplePort.queue (sample1R [j]);
                    reader1CueR.samplePort.queue (sample1CueR [j]);
                }
            reader1L.start ();
            reader1R.start ();
            reader1CueR.start ();
        }
        else
        {
            position2 = 0;
            getNumFramesMovedOld2 = 0;
            for (int i = 0 ; i < 1 ; i++)
                for (int j = 0 ; j < sample2L.length ; j++)
                {
                    reader2L.samplePort.queue (sample2L [j]);
                    reader2R.samplePort.queue (sample2R [j]);
                    reader2CueR.samplePort.queue (sample2CueR [j]);
                }
            reader2L.start ();
            reader2R.start ();
            reader2CueR.start ();
        }
    }
    catch (SynthException e)
    {
        System.err.println ("queueSamples");
        SynthAlert.showError (this, e);
    }
}

// ************************************************************************
public void clearQueue (boolean deck1)
{
    try
    {
        if (deck1)
        {
            reader1L.stop ();
            reader1R.stop ();
            reader1CueR.stop ();
            reader1L.samplePort.clear ();
            reader1R.samplePort.clear ();
            reader1CueR.samplePort.clear ();

            double t = reader1L.amplitude.get ();

            reader1L = new SampleReader_16V1 (synthContext1);
            reader1R = new SampleReader_16V1 (synthContext1);
            reader1CueR = new SampleReader_16V1 (synthContext2);

            reader1L.amplitude.set (t);

            reader1R.amplitude.set (t);
            reader1CueR.amplitude.set (t);

            reader1L.output.connect (fLP1L.input);
            reader1R.output.connect (fLP1R.input);
            reader1CueR.output.connect (mU1Cue1.inputA);

            if (region [4] [10].value == 1)
                thread03.setOutput (false);
            else
                thread03.setOutput (true);

            position1 = 0;
            forward1 = true;
            getNumFramesMovedOld1 = 0;
            timerLabel1.setText ("00:00:00");
        }
        else
        {
            reader2L.stop ();
            reader2R.stop ();
            reader2CueR.stop ();
            reader2L.samplePort.clear ();
				reader2R.samplePort.clear ();
				reader2CueR.samplePort.clear ();

				double t = reader2L.amplitude.get ();

				reader2L = new SampleReader_16V1 (synthContext1);
				reader2R = new SampleReader_16V1 (synthContext1);
				reader2CueR = new SampleReader_16V1 (synthContext2);

				reader2L.amplitude.set (t);
				reader2R.amplitude.set (t);
				reader2CueR.amplitude.set (t);

				reader2L.output.connect (fLP2L.input);
				reader2R.output.connect (fLP2R.input);
				reader2CueR.output.connect (mU2Cue1.inputA);

				if (region [4] [10].value == 1)
					thread03.setOutput (false);
				else
					thread03.setOutput (true);

				position2 = 0;
				forward2 = true;
				getNumFramesMovedOld2 = 0;
				timerLabel2.setText ("00:00:00");
			}
		}
		catch (SynthException e)
		{
			System.err.println ("clearQueue");
			SynthAlert.showError (this, e);
		}
	}


	// ***********************************************************************
	public class TimerThread implements Runnable
	{
		Thread timer;

		public TimerThread ()
		{
			timer = new Thread (this);
			timer.start ();
		}


		public void stop ()
		{
			timer = null;
		}


		public void run ()
		{
			Thread me = Thread.currentThread ();
			while (timer == me)
			{
				try
				{
					Thread.currentThread ().sleep (100);
				}
				catch (InterruptedException e)
				{
				}
				doIt ();
			}
		}


		public void changeER (boolean deck1)
		{
			if (deck1)
			{
				eR1 = !eR1;
				display1 ();
			}
			else
			{
				eR2 = !eR2;
				display2 ();
			}
		}


		int timeOld1 = 0;
		int timeOld2 = 0;
		boolean eR1 = false, eR2 = false;
		DecimalFormat f = new DecimalFormat ("00");

		public void display1 ()
		{
			if (!eR1)
				timeNew1 = sampleNumFrames1 - timeNew1;
			timeNew1 /= 441;
			int minutes = timeNew1 / 6000;
			timeNew1 %= 6000;
			int seconds = timeNew1 / 100;
			timeNew1 %= 100;

			timerLabel1.setText (f.format (minutes) + ":" + f.format (seconds) + ":" + f.format (timeNew1));
		}


		public void display2 ()
		{
			if (!eR2)
				timeNew2 = sampleNumFrames2 - timeNew2;
			timeNew2 /= 441;
			int minutes = timeNew2 / 6000;
			timeNew2 %= 6000;
			int seconds = timeNew2 / 100;
			timeNew2 %= 100;

			timerLabel2.setText (f.format (minutes) + ":" + f.format (seconds) + ":" + f.format (timeNew2));
		}


		int timeNew1, timeNew2;

		public void doIt ()
		{
			timeNew1 = getPosition (true, 2);

			if (timeNew1 != timeOld1)
			{
				timeOld1 = timeNew1;
				display1 ();
			}

			timeNew2 = getPosition (false, 2);
		if (timeNew2 != timeOld2)
		{
			timeOld2 = timeNew2;
			display2 ();
		}
	}
}


// ************************************************************************
public class VUMeterThread implements Runnable
{
	Thread timer;

	public VUMeterThread ()
	{
		timer = new Thread (this);
		timer.start ();
	}


	public void stop ()
	{
		timer = null;
	}


	public void run ()
	{
		Thread me = Thread.currentThread ();
		while (timer == me)
		{
			try
			{
				Thread.currentThread ().sleep (50);
			}
			catch (InterruptedException e)
			{
			}
			doVUMeter ();
		}
	}



	public synchronized void setOutput (boolean b)
	{
		if (b)
		{
			addLeft.output.connect (peakFL.input);
			addRight.output.connect (peakFR.input);
		}
		else
		{
			reader1R.output.connect (peakFL.input);
			reader2R.output.connect (peakFR.input);
		}
	}


	int dbIndOldL = 0;
	int dbIndOldR = 0;
	int [] range = { - 500, - 30, - 20, - 10, - 7, - 4, - 2, 0, 2, 4, 7, 500};

		public synchronized void doVUMeter ()
		{
			try
			{
				double dbL = AudioMath.amplitudeToDecibels (peakFL.output.get ());
				double dbR = AudioMath.amplitudeToDecibels (peakFR.output.get ());

				if (dbL >= range [dbIndOldL + 1])
					vuMeterL [dbIndOldL++].setVisible (true);
				else if (dbL < range [dbIndOldL] && dbIndOldL > 0)
					vuMeterL [--dbIndOldL].setVisible (false);

				if (dbR >= range [dbIndOldR + 1])
					vuMeterR [dbIndOldR++].setVisible (true);
				else if (dbR < range [dbIndOldR] && dbIndOldR > 0)
					vuMeterR [--dbIndOldR].setVisible (false);
			}
			catch (SynthException e)
			{
				System.err.println ("doVUMeter");
			}
		}
	}


	double [] frequencyArray = {
		8.176, 8.662, 9.177, 9.723, 10.301, 10.913,
		11.562, 12.250, 12.978, 13.750, 14.568, 15.434,
		16.352, 17.324, 18.354, 19.445, 20.602, 21.827,
		23.125, 24.500, 25.957, 27.500, 29.135, 30.868,
		32.703, 34.648, 36.708, 38.891, 41.203, 43.654,
		46.249, 48.999, 51.913, 55.000, 58.270, 61.735,
		65.406, 69.296, 73.416, 77.782, 82.407, 87.307,
		92.499, 97.999, 103.826, 110.000, 116.541, 123.471,
		130.813, 138.591, 146.832, 155.563, 164.814, 174.614,
		184.997, 195.998, 207.652, 220.000, 233.082, 246.942,
		261.626, 277.183, 293.665, 311.127, 329.628, 349.228,
		369.994, 391.995, 415.305, 440.000, 466.164, 493.883,
		523.251, 554.365, 587.330, 622.254, 659.255, 698.456,
		739.989, 783.991, 830.609, 880.000, 932.328, 987.767,
		1046.502, 1108.731, 1174.659, 1244.508, 1318.510, 1396.913,
		1479.978, 1567.982, 1661.219, 1760.000, 1864.655, 1975.533,
		2093.005, 2217.461, 2349.318, 2489.016, 2637.020, 2793.826,
		2959.955, 3135.963, 3322.438, 3520.000, 3729.310, 3951.066,
		4186.009, 4434.922, 4698.636, 4978.032, 5274.041, 5587.652,
		5919.911, 6271.927, 6644.875, 7040.000, 7458.620, 7902.133,
		8372.018, 8869.844, 9397.273, 9956.063, 10548.08, 11175.30,
		11839.82, 19988 //12543.85
		};
}


// **********************************
class ControlRegion
{
	int value, x1, y1, x2, y2, width, heigth, iD, max;
	JLabel label = null;
```

```
    public ControlRegion (int x, int y, int w, int h, int id, int m)
    {
        x1 = x;
        y1 = y;
        width = w;
        heigth = h;
        iD = id;
        x2 = x + w;
        y2 = y + h;
        value = 0;
        max = m;
    }


    public void setLabel (JLabel lab)
    {
        label = lab;
    }


    public JLabel getLabel ()
    {
        return label;
    }


    public boolean setValues (int k, int k2)
    {
        return false;
    }


    public boolean setValues (int k, int k2, boolean b)
    {
        return false;
    }


    public boolean setValues (int k)
    {
        return false;
    }


    public boolean equals (int x, int y)
    {
        if (x >= x1 && x < x2 && y >= y1 && y < y2)
            return true;
        return false;
    }


    public boolean equals (int x, int y, double tS)
    {
        return equals (x, y);
    }


    public boolean stillEquals (int x, int y)
    {
        return false;
    }
}
```

```
//  ************************************
class VerticalSlider extends ControlRegion
{
    int min1, max1;
    double pixelsPerInc;

    public VerticalSlider (int x, int y, int w, int h, int v1, int v2, int id, int m)
    {
        super (x, y, w, h, id, m);
        min1 = v1;
        max1 = v2;
        pixelsPerInc = (max1 - min1) / max;
    }


    public boolean setValues (int x, int y)
    {
        int t = max - (int) ((double) (y - min1) / pixelsPerInc);

        return setValues2 (t);
    }


    public boolean setValues (int k)
    {
        return setValues2 (k);
    }


    private boolean setValues2 (int k)
    {
        k = Math.min (Math.max (k, 0), max);

        if (value == k)
            return false;

        value = k;

        y1 = min1 - (heigth / 2) + (int) ((max - value) * pixelsPerInc);
        y2 = y1 + heigth;

        if (label != null)
            label.setLocation (x1, y1);
        return true;
    }


    public boolean stillEquals (int x, int y)
    {
        if (x >= x1 - 10 && x < x2 + 10 && y >= y1 - 50 && y < y2 + 50)
            return true;
        return false;
    }
}


//  ************************************
class HorizontalSlider extends ControlRegion
{
    public HorizontalSlider (int x, int y, int w, int h, int id, int m)
    {
        super (x, y, w, h, id, m);
    }
```

```
	public boolean setValues (int x, int y)
	{
		int t = (int) ((x - 430) * 0.7143);

		t = Math.min (Math.max (t, 0), max);

		if (value == t)
			return false;

		value = t;

		x1 = (int) (value * 1.4) + 420;
		x2 = x1 + width;

		if (label != null)
			label.setLocation (x1, y1);
		return true;
	}


	public boolean setValues (int k)
	{
		return setValues (k, 0);
	}


	public boolean stillEquals (int x, int y)
	{
		if (x >= x1 - 50 && x < x2 + 50 && y >= y1 - 10 && y < y2 + 10)
			return true;
		return false;
	}
}


//  ********************************
class RotaryKnob extends ControlRegion
{
	public RotaryKnob (int x, int y, int w, int h, int id, int m)
	{
		super (x, y, w, h, id, m);
	}


	public boolean setValues (int x, int y)
	{
		double newPos = Math.atan2 (y - y1 - 15, x - x1 - 15);
		double diffPos = newPos - prevPos;

		if (prevPos < -1.5708 && newPos > 1.5708)
			diffPos = newPos - 6.283185 - prevPos;
		else if (prevPos > 1.5708 && newPos < -1.5708)
			diffPos = newPos + 6.283185 - prevPos;

		prevPos = newPos;

		int t = value + (int) (diffPos * 25);

		return setValues2 (t);
	}


	public boolean setValues (int k)
	{
		return setValues2 (k);
	}


	private boolean setValues2 (int k)
	{
		value = k;

		if (value < 0)
			value = 0;
		else if (value > max)
			value = max;

		if (label != null)
		{
			double t = value;
			t *= 0.048;
			t -= 4;

			int xPos = x1 + 12 + (int) (Math.cos (t) * 10);
			int yPos = y1 + 12 + (int) (Math.sin (t) * 10);

			label.setLocation (xPos, yPos);
		}
		return true;
	}


	double prevPos = 0;

	public boolean equals (int x, int y)
	{
		if (super.equals (x, y))
		{
			prevPos = Math.atan2 (y - y1 - 15, x - x1 - 15);
			return true;
		}
		return false;
	}


	public boolean stillEquals (int x, int y)
	{
		return super.equals (x, y);
	}
}


//  ***********************************
class JogWheel extends ControlRegion
{
	public JogWheel (int x, int y, int d, int id, int m)
	{
		super (x, y, d, d, id, m);
	}


	int diffPos, prevPos, lockPos;

	public boolean setValues (int x, int y, boolean scratch)
	{
		int currPos = (int) (1000 * Math.atan2 (y - y1 - 350, x - x1 - 350));
		diffPos = currPos - prevPos;

		if (prevPos < -1571 && currPos > 1571)
			diffPos = currPos - 6283 - prevPos;
		else if (prevPos > 1571 && currPos < -1571)
			diffPos = currPos + 6283 - prevPos;

		lockPos += diffPos;
		prevPos = currPos;

		value = (int) ((double) lockPos / 44.88);
```

```
        if (!scratch)
        {
            if (value < 0)
            {
                value = 0;
                lockPos = 0;
            }
            else if (value > max)
            {
                value = max;
                lockPos = 4488;
            }
        }

        if (label != null)
            rePaint ();
        return true;
    }


    public boolean setValues (int k)
    {
        value = k;
        lockPos = (int) (k * 44.88);
        rePaint ();
        return true;
    }


    public void rePaint ()
    {
        double t = value;
        t *= 0.04488;
        t -= 3.8148;

        int xPos = x1 + 325 + (int) (Math.cos (t) * (300));
        int yPos = y1 + 325 + (int) (Math.sin (t) * (300));
        label.setLocation (xPos, yPos);
    }


    public boolean equals (int x, int y)
    {
        if (super.equals (x, y))
        {
            double a = Math.pow (x - x1 - 350, 2);
            double b = Math.pow (y - y1 - 350, 2);

            if (Math.sqrt (a + b) <= 350)
            {
                prevPos = (int) (1000 * Math.atan2 (y - y1 - 350, x - x1 - 350));
                diffPos = 0;
                return true;
            }
        }
        return false;
    }


    public boolean stillEquals (int x, int y)
    {
        return super.equals (x, y);
    }
}
```

```
//  **********************************
class Button1 extends ControlRegion
{
    public Button1 (int x, int y, int w, int h, int id)
    {
        super (x, y, w, h, id, - 1);
    }


    public boolean stillEquals (int x, int y)
    {
        return equals (x, y);
    }
}


//  **********************************
class Button2 extends Button1
{
    public Button2 (int x, int y, int w, int h, int id)
    {
        super (x, y, w, h, id);
        value = 0;
    }


    public boolean setValues (int k)
    {
        if (value == 0)
        {
            value = 1;
            if (label != null)
                label.setVisible (true);
            return true;
        }
        value = 0;
        if (label != null)
            label.setVisible (false);
        return false;

    }
}
```

“TheMidiObject.java”

```
// TheMidiObject

// JavaMIDI by: Robert Marsanyi
// Copyright CagEnt Technologies, Inc
// All Rights Reserved


import jmidi.*;

public class TheMidiObject
{
    TheMain m;
    static final int BUFSIZE = 3;
    static byte [] buffer = new byte [BUFSIZE];
    static MidiPort midiport;
    InputThread01 inputThread;
    public TheMidiObject (TheMain m)
    {
        this.m = m;

        int inDevice = 1;
        int outDevice = 0;

        try
        {
            midiport = new MidiPort (inDevice, outDevice);
            midiport.open ();

            inputThread = new InputThread01 (midiport, this);
            inputThread.start ();
        }
        catch (Exception e)
        {
            System.err.println ("WHOOPS! error! " + e.getMessage ());
        }
    }
}


class InputThread01 extends Thread
{
    MidiPort midiport;
    TheMidiObject t;

    public InputThread01 (MidiPort aPort, TheMidiObject d)
    {
        t = d;
        this.midiport = aPort;
    }


    public void run ()
    {
        int nBytes;
        boolean done = false;
        byte [] buffer = TheMidiObject.buffer;

        MidiPortMessage msg;

        try
        {
            msg = new MidiPortMessage (TheMidiObject.BUFSIZE);
            buffer = msg.messageData;
            double prevTStamp = 0;

            System.out.println ("Opening port");

            while (!done)
            {
                try
                {
                    while (midiport.messagesWaiting () == 0)
                    {
                        yield ();
                    }
                    nBytes = midiport.readMessage (msg);
                    double tStamp = msg.timeStamp;
                    if (prevTStamp != tStamp)
                    {
                        int statusByte = 128 + buffer [0];
                        int leftRigthFreeCtrl = statusByte / 16; //0=LC,1=RC,2=LF,3=RF
                        int statusByteX = (statusByte % 16) / 4;
                        int statusByteY = statusByte % 4;

                        int x = (int) ((statusByteX * 128 + buffer [1]) * 3.2) - 64;
                        int y = (int) ((statusByteY * 128 + buffer [2]) * 3.2) - 64;

                        if (leftRigthFreeCtrl == 0 || leftRigthFreeCtrl == 2)
                            t.m.leftHand.midiIn (statusByte < 32 ? true:
                            false, x, y, tStamp);
                        else
                            t.m.rightHand.midiIn (statusByte < 32 ? true:
                            false, x, y, tStamp);
                        prevTStamp = tStamp;
                    }
                }
                catch (MidiPortException e)
                {
                    //System.err.println ("MidiPort Exception (probably overran buffer) - resetting.");
                    System.err.println ("MidiPort resetting.");
                    midiport.resetInput ();
                }
            }

            System.out.println ("Quit");
            System.out.println ("Checking message queue (should be 0 left) = " + midiport.messagesWaiting ());
            System.out.println ("Closing port");
            midiport.close ();
            System.out.println ("Done!");
            //****************************************************************************
            t.m.stop ();
            //****************************************************************************


        }
        catch (Exception e)
        {
            System.err.println ("WHOOPS! error in  inputthread! " + e.getMessage ());
        }

    }
}
```

"SamplePreparator.java"

```
// The "SamplePreparator" class.


import java.io.*;
import java.text.*;
import java.awt.*;
import java.awt.event.*;
import com.softsynth.jsyn.*;
import com.softsynth.jsyn.util.*;

public class SamplePreparator extends Frame
{
    FileDialog fd;
    TextArea tA;
    List theListDir, theListName;
    Button startButton, addButton, removeButton;
```

```
public SamplePreparator ()
{
    super ("SamplePreparator");

    setLayout (null);

    addWindowListener (new WindowAdapter ()
    {
        public void windowClosing (WindowEvent e)
        {
            System.exit (0);
        }
    }
    );

    fd = new FileDialog (this);
    tA = new TextArea ();
    add (tA);
    tA.setBounds (340, 90, 350, 300);
    theListName = new List ();
    theListDir = new List ();
    theListName.setBounds (10, 90, 320, 300);
    add (theListName);

    startButton = new Button ("Start");
    startButton.setBounds (230, 30, 100, 50);
    add (startButton);
    startButton.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            startButton.setEnabled (false);
            addButton.setEnabled (false);
            removeButton.setEnabled (false);
            prepare ();
        }
    }
    );

    addButton = new Button ("Add");
    addButton.setBounds (10, 30, 100, 50);
    add (addButton);
    addButton.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            fd.setFile ("*.wav");
            fd.show ();
            String s1 = fd.getFile ();
            String s2 = fd.getDirectory ();
            tA.append (s1 + "\n");
            if (s1 != null)
            {
                theListName.add (s1);
                theListDir.add (s2);
            }
        }
    }
    );

    removeButton = new Button ("Remove");
    removeButton.setBounds (120, 30, 100, 50);
    add (removeButton);
    removeButton.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            if (theListName.getSelectedIndex () >= 0)
            {
                int i = theListName.getSelectedIndex ();
                theListName.remove (i);
                theListDir.remove (i);
            }
        }
    }
    );

    setBackground (Color.lightGray);
    setSize (700, 400);
    show ();
}


void prepare ()
{
    String [] listName = theListName.getItems ();
    String [] listDir = theListDir.getItems ();
    int i = 0;
    while (i < listName.length)
    {
        prepareNow (listDir [i], listName [i]);
        i++;
    }
}


String destinationPath = "Samples";
int fileSize = 176400;

void prepareNow (String fileDir, String fileName)
{
    DecimalFormat f = new DecimalFormat ("000");

    try
    {
        boolean proceed = true;
        tA.append ("\n\n" + fileName + "\n");

        SynthContext synth = new SynthContext ();
        synth.initialize ();
        synth.start (0, Synth.DEFAULT_FRAME_RATE, - 1, 0, 9, 2);

        SampleReader_16F2 reader = new SampleReader_16F2 (synth);

        LineOut lineOut = new LineOut (synth);

        reader.output.connect (0, lineOut.input, 0);
        reader.output.connect (1, lineOut.input, 1);

        reader.start ();
        lineOut.start ();

        InputStream stream = (InputStream) (new FileInputStream (fileDir + "/" + fileName));

        SynthSample mySamp = new SynthSampleWAV (synth);
```

```
            try
            {
                mySamp.load (stream);
                mySamp.byteData = null;

                reader.samplePort.queue (mySamp);
            }
            catch (SynthException e)
            {
                tA.append ("Inside " + e + "\n");
                proceed = false;
            }

            if (proceed)
            {
                new File (destinationPath + "/" + fileName).mkdir ();

                int nF = mySamp.getNumFrames ();
                short [] shrtData = new short [nF * 2];
                mySamp.read (0, shrtData, 0, nF);

                stream = null;
                mySamp = null;

                int howManyFiles = shrtData.length / (fileSize * 2);
                int howManyExtraSamples = shrtData.length % (fileSize * 2);

                tA.append ("number of files " + (howManyFiles + 1) + "\n");

                for (int i = 0 ; i < howManyFiles ; i++)
                {
                    tA.append ((i + 1) + " / " + (howManyFiles + 1) + "\n");

                    short shrtData2 [] = new short [fileSize];

                    for (int j = 0, k = i * fileSize * 2 ; j < fileSize ; j++, k += 2)
                        shrtData2 [j] = shrtData [k];


                    String newFileName = destinationPath + "/" + fileName + "/L" + f.format (i) + ".wav";
                    RandomAccessFile rfile = new RandomAccessFile (newFileName, "rw");
                    WAVFileWriter wavWriter = new WAVFileWriter (rfile);
                    wavWriter.write (shrtData2, 1, 44100);
                    wavWriter.close ();

                    for (int j = 0, k = i * fileSize * 2 + 1 ; j < fileSize ; j++, k += 2)
                        shrtData2 [j] = shrtData [k];

                    newFileName = destinationPath + "/" + fileName + "/R" + f.format (i) + ".wav";
                    rfile = new RandomAccessFile (newFileName, "rw");
                    wavWriter = new WAVFileWriter (rfile);
                    wavWriter.write (shrtData2, 1, 44100);
                    wavWriter.close ();
                }

                tA.append ((howManyFiles + 1) + " / " + (howManyFiles + 1) + "\n");

                short shrtData2 [] = new short [howManyExtraSamples / 2];
                int temp = howManyExtraSamples / 2;

                for (int j = 0, k = howManyFiles * fileSize * 2 ; j < temp ; j++, k += 2)
                    shrtData2 [j] = shrtData [k];

                String newFileName = destinationPath + "/" + fileName + "/L" + f.format (howManyFiles) + ".wav";
                RandomAccessFile rfile = new RandomAccessFile (newFileName, "rw");
                WAVFileWriter wavWriter = new WAVFileWriter (rfile);
                wavWriter.write (shrtData2, 1, 44100);
                wavWriter.close ();

                for (int j = 0, k = howManyFiles * fileSize * 2 + 1 ; j < temp ; j++, k += 2)
                    shrtData2 [j] = shrtData [k];

                newFileName = destinationPath + "/" + fileName + "/R" + f.format (howManyFiles) + ".wav";
                rfile = new RandomAccessFile (newFileName, "rw");
                wavWriter = new WAVFileWriter (rfile);
                wavWriter.write (shrtData2, 1, 44100);
                wavWriter.close ();

                tA.append ("FINISH\n");
            }
            reader.stop ();
            reader.samplePort.clear ();
            lineOut.stop ();
            synth.stop ();
            synth.delete ();
        }
        catch (Exception e)
        {
            tA.append ("INVALID FILE\n");
            tA.append (e + "\n");
        }
    }


    public static void main (String [] args)
    {
        new SamplePreparator ();

    }
}
```

SystemThree

“TheMain.java”

```
import java.awt.*;

import java.awt.event.*;
import javax.swing.*;
import java.lancs.*;
import hsa.Console;

public class TheMain extends JFrame
{
    static Console c;
    TheMidiObject midiObject;
    JLayeredPane layeredPane;
    JLabel backgroundLabel, handLabel01, handLabel02, greenBallLabel, quadLabel;

    String imagesPath = "images/";
    String backgroundFile = imagesPath + "background01.gif";
    String handFile01 = imagesPath + "handright03.gif";
    String handFile02 = imagesPath + "handright01.gif";
    String greenBallFile = imagesPath + "ballgreen01.gif";

    public TheMain ()
    {
        c = new Console ();

        Image backgroundImage = Toolkit.getDefaultToolkit ().getImage (backgroundFile);
        Image handImage01 = Toolkit.getDefaultToolkit ().getImage (handFile01);
        Image handImage02 = Toolkit.getDefaultToolkit ().getImage (handFile02);
        Image greenBallImage = Toolkit.getDefaultToolkit ().getImage (greenBallFile);

        final ImageIcon backgroundIcon = new ImageIcon (backgroundImage);
        final ImageIcon handIcon01 = new ImageIcon (handImage01);
        final ImageIcon handIcon02 = new ImageIcon (handImage02);
        final ImageIcon greenBallIcon = new ImageIcon (greenBallImage);

        backgroundLabel = new JLabel (backgroundIcon);
        handLabel01 = new JLabel (handIcon01);
        handLabel02 = new JLabel (handIcon02);
        greenBallLabel = new JLabel (greenBallIcon);
        quadLabel = new JLabel ("QUAD");

        quadLabel.setFont (new Font ("arial", Font.BOLD, 20));
        quadLabel.setForeground (Color.red);
        quadLabel.setHorizontalAlignment (JLabel.CENTER);

        backgroundLabel.setBounds (0, 0, backgroundIcon.getIconWidth (), backgroundIcon.getIconHeight ());
        handLabel01.setBounds (- 100, - 100, handIcon01.getIconWidth (), handIcon01.getIconHeight ());
        handLabel02.setBounds (- 100, - 100, handIcon02.getIconWidth (), handIcon02.getIconHeight ());
        greenBallLabel.setBounds (542, 292, greenBallIcon.getIconWidth (), greenBallIcon.getIconHeight ());
        quadLabel.setBounds (100, 0, 100, 50);

        layeredPane = new JLayeredPane ();
        layeredPane.setPreferredSize (new Dimension (backgroundIcon.getIconWidth (), backgroundIcon.getIconHeight ()));

        layeredPane.add (backgroundLabel, new Integer (0));
        layeredPane.add (greenBallLabel, new Integer (1));
        layeredPane.add (handLabel01, new Integer (2));
        layeredPane.add (handLabel02, new Integer (2));
        layeredPane.add (quadLabel, new Integer (1));

        getContentPane ().add (layeredPane, BorderLayout.CENTER);
        setSize (1020, 750);
        setVisible (true);
handLabel02.setVisible (false);

makeMidi ();

layeredPane.addMouseMotionListener (new MouseMotionAdapter ()
{
    public void mouseMoved (MouseEvent e)
    {
        int x = e.getX ();
        int y = e.getY () - 50;
        midiIn (false, x, y);
    }
    public void mouseDragged (MouseEvent e)
    {
        int x = e.getX ();
        int y = e.getY () - 50;
        midiIn (true, x, y);
    }
}
);


try
{
    sop ("Setup1 (Y/N): ");
    char c = BasicIo.readCharacter ();

    if (c == 'Y' || c == 'y')
    {
        midiObject.midiOutAction (8, 127);
        midiObject.midiOutAction (8, 0);
        Thread.sleep (100);
        midiObject.midiOutAction (8, 127);
        midiObject.midiOutAction (8, 0);
        Thread.sleep (100);
        midiObject.midiOutAction (8, 127);
        midiObject.midiOutAction (8, 0);
        Thread.sleep (100);
        midiObject.midiOutAction (8, 127);
        midiObject.midiOutAction (8, 0);
        Thread.sleep (100);
        midiObject.midiOutAction (0, 127);
        midiObject.midiOutAction (0, 0);
        Thread.sleep (100);
    }

    sop ("Setup2 (Y/N): ");
    c = BasicIo.readCharacter ();

    if (c == 'Y' || c == 'y')
    {
        midiObject.midiOutAction (14, 127);
        Thread.sleep (100);
        midiObject.midiOutAction (26, 127);
        Thread.sleep (100);
        midiObject.midiOutAction (26, 0);
        Thread.sleep (100);
        midiObject.midiOutAction (14, 0);
        Thread.sleep (100);

        midiIn (true, 100, 0);

        for (int i = 350 ; i <= 753 ; i += 2)
            midiIn (true, i, 300);
        Thread.sleep (100);
        for (int i = 754 ; i >= 340 ; i -= 2)
            midiIn (true, i, 300);
        Thread.sleep (1000);
```

```
            inControlRegion = false;
            midiIn (true, 100, 0);
            Thread.sleep (1000);

            for (int i = 100 ; i <= 500 ; i += 5)
                midiIn (true, 350, i);
            Thread.sleep (1000);
            for (int i = 350 ; i <= 750 ; i += 5)
                midiIn (true, i, 500);
            Thread.sleep (1000);
            for (int i = 500 ; i >= 100 ; i -= 5)
                midiIn (true, 750, i);
            Thread.sleep (1000);
            for (int i = 750 ; i >= 350 ; i -= 5)
                midiIn (true, i, 100);
            Thread.sleep (1000);
        }
    }
    catch (Exception e)
    {
    }
    promptWindow ();
}


double [] lookup01 = {
    0, 0, 0, 0, 0, 0, 0, - 0.1, - 0.1, - 0.1,
    - 0.1, - 0.1, - 0.2, - 0.2, - 0.2, - 0.2, - 0.3, - 0.3, - 0.4, - 0.4,
    - 0.4, - 0.5, - 0.5, - 0.6, - 0.6, - 0.7, - 0.7, - 0.8, - 0.9, - 0.9,
    - 1.0, - 1.1, - 1.1, - 1.2, - 1.3, - 1.4, - 1.5, - 1.6, - 1.6, - 1.7,
    - 1.8, - 1.9, - 2.0, - 2.2, - 2.3, - 2.4, - 2.5, - 2.6, - 2.7, - 2.9,
    - 3.0, - 3.1, - 3.3, - 3.4, - 3.6, - 3.7, - 3.9, - 4.1, - 4.3, - 4.4,
    - 4.6, - 4.8, - 5.0, - 5.2, - 5.4, - 5.6, - 5.9, - 6.1, - 6.3, - 6.6,
    - 6.9, - 7.1, - 7.4, - 7.7, - 8.0, - 8.3, - 8.7, - 9.0, - 9.4, - 9.8,
    - 10.2, - 10.6, - 11.1, - 11.6, - 12.1, - 12.6, - 13.2, - 13.9, - 14.5, - 15.3,
    - 16.1, - 17.0, - 18.0, - 19.2, - 20.5, - 22.1, - 24.0, - 26.6, - 30.1, - 36.5,
    - 200};

double [] lookup02 = {
    - 500, - 110, - 104, - 98.8, - 93.3, - 87.7, - 82.1, - 76.4, - 70.7, - 66.3,
    - 62.1, - 58.5, - 55.4, - 52.3, - 49.4, - 47.0, - 44.5, - 42.1, - 39.8, - 38.3,
    - 36.7, - 35.2, - 33.7, - 32.1, - 30.6, - 29.5, - 28.6, - 27.7, - 26.8, - 26.0,
    - 25.1, - 24.2, - 23.4, - 22.5, - 21.6, - 20.8, - 20.0, - 19.6, - 19.2, - 18.9,
    - 18.5, - 18.2, - 17.8, - 17.5, - 17.1, - 16.7, - 16.4, - 16.0, - 15.7, - 15.3,
    - 15.0, - 14.6, - 14.3, - 14.0, - 13.7, - 13.3, - 13.0, - 12.7, - 12.4, - 12.0,
    - 11.7, - 11.4, - 11.1, - 10.7, - 10.4, - 10.1, - 9.8, - 9.4, - 9.1, - 8.8,
    - 8.5, - 8.2, - 7.8, - 7.5, - 7.2, - 6.9, - 6.6, - 6.2, - 5.9, - 5.6,
    - 5.3, - 5.0, - 4.6, - 4.3, - 4.0, - 3.7, - 3.4, - 3.0, - 2.7, - 2.4,
    - 2.1, - 1.8, - 1.4, - 1.1, - 0.8, - 0.5, - 0.2, 0.1, 0.3, 0.5,
    0.7, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.1, 2.3,
    2.5, 2.7, 2.9, 3.1, 3.3, 3.5, 3.7, 3.9, 4.1, 4.3,
    4.5, 4.6, 4.8, 5.0, 5.2, 5.4, 5.6, 6.0};


boolean controlMoveState = false;
boolean surroundState = true;
boolean inControlRegion = false;

int x1 = 350, y1 = 100, x2 = 750, y2 = 500;

int ballX = 0, ballY = 0;
int speakerBallX = 0, speakerBallY = 0;
int frequencyBallX = 0, frequencyBallY = 50;
int [] speaker = {96, 0, 0, 0};
int prevFrequency = 0;
```

```
void midiIn (boolean ctrlMove, int x, int y)
{
    if (controlMoveState != ctrlMove)
    {
        if (ctrlMove)
        {
            handLabel01.setVisible (false);
            handLabel02.setVisible (true);
        }
        else
        {
            handLabel02.setVisible (false);
            handLabel01.setVisible (true);
        }
        controlMoveState = ctrlMove;
    }

    if (ctrlMove)
    {
        handLabel02.setLocation (x - 11, y);
        if (x >= x1 - 100 && x < x2 + 100 && y >= y1 - 100 && y < y2 + 100)
        {
            ballX = (x - x1) / 4;
            if (ballX < 0)
                ballX = 0;
            else if (ballX > 100)
                ballX = 100;

            ballY = (y - y1) / 4;
            if (ballY < 0)
                ballY = 0;
            else if (ballY > 100)
                ballY = 100;

            greenBallLabel.setLocation (ballX * 4 + x1 - 8, ballY * 4 + y1 - 8);

            sendMidi02 ();
        }
        else if (y < 50 && x < 200 && x >= 100)
        {
            if (!inControlRegion)
            {
                surroundState = !surroundState;
                sendMidi01 ();

                greenBallLabel.setLocation (ballX * 4 + x1 - 8, ballY * 4 + y1 - 8);
                inControlRegion = true;
            }
        }
        else
            inControlRegion = false;
    }
    else
    {
        handLabel01.setLocation (x - 11, y);
    }
}
```

```
void sendMidi01 ()
{
        try
        {
                if (surroundState)
                {
                        midiObject.midiOutAction (14, 127);
                        Thread.sleep (100);
                        midiObject.midiOutAction (26, 127);
                        midiObject.midiOutAction (26, 0);
                        midiObject.midiOutAction (14, 0);

                        frequencyBallX = ballX;
                        ballX = speakerBallX;
                        ballY = speakerBallY;
                        quadLabel.setText ("QUAD");
                }
                else
                {
                        midiObject.midiOutAction (27, 127);
                        midiObject.midiOutAction (27, 0);
                        Thread.sleep (100);
                        midiObject.midiOutAction (27, 127);
                        midiObject.midiOutAction (27, 0);
                        Thread.sleep (100);
                        midiObject.midiOutAction (27, 127);
                        midiObject.midiOutAction (27, 0);

                        speakerBallX = ballX;
                        speakerBallY = ballY;
                        ballX = frequencyBallX;
                        ballY = frequencyBallY;
                        quadLabel.setText ("Focusrite");
                }
        }
        catch (Exception e)
        {
        }
}


int getFromLookup (int dBPos, double dB)
{
        if (lookup02 [dBPos] == dB)
                return dBPos;

        boolean found = false;

        if (lookup02 [dBPos] < dB)
        {
                while (!found)
                {
                        dBPos++;
                        if (lookup02 [dBPos] > dB)
                                found = true;
                }
                return dBPos - 1;
        }

        while (!found)
        {
                dBPos--;
                if (lookup02 [dBPos] <= dB)
                        found = true;
        }
        return dBPos;
}
```

```
void sendMidi02 ()
{
        if (surroundState)
        {
                double dB1 = lookup01 [ballX] + lookup01 [ballY];
                double dB2 = lookup01 [100 - ballX] + lookup01 [ballY];
                double dB3 = lookup01 [ballX] + lookup01 [100 - ballY];
                double dB4 = lookup01 [100 - ballX] + lookup01 [100 - ballY];

                int speaker1 = getFromLookup (speaker [0], dB1);
                int speaker2 = getFromLookup (speaker [1], dB2);
                int speaker3 = getFromLookup (speaker [2], dB3);
                int speaker4 = getFromLookup (speaker [3], dB4);

                if (speaker [0] < speaker1)
                {
                        for (int i = speaker [0] + 2 ; i < speaker1 ; i += 2)
                                midiObject.midiOutAction (75, i);
                        midiObject.midiOutAction (75, speaker1);
                }
                else if (speaker [0] > speaker1)
                {
                        for (int i = speaker [0] - 2 ; i > speaker1 ; i -= 2)
                                midiObject.midiOutAction (75, i);
                        midiObject.midiOutAction (75, speaker1);
                }

                if (speaker [1] < speaker2)
                {
                        for (int i = speaker [1] + 2 ; i < speaker2 ; i += 2)
                                midiObject.midiOutAction (76, i);
                        midiObject.midiOutAction (76, speaker2);
                }
                else if (speaker [1] > speaker2)
                {
                        for (int i = speaker [1] - 2 ; i > speaker2 ; i -= 2)
                                midiObject.midiOutAction (76, i);
                        midiObject.midiOutAction (76, speaker2);
                }

                if (speaker [2] < speaker3)
                {
                        for (int i = speaker [2] + 2 ; i < speaker3 ; i += 2)
                                midiObject.midiOutAction (72, i);
                        midiObject.midiOutAction (72, speaker3);
                }
                else if (speaker [2] > speaker3)
                {
                        for (int i = speaker [2] - 2 ; i > speaker3 ; i -= 2)
                                midiObject.midiOutAction (72, i);
                        midiObject.midiOutAction (72, speaker3);
                }

                if (speaker [3] < speaker4)
                {
                        for (int i = speaker [3] + 2 ; i < speaker4 ; i += 2)
                                midiObject.midiOutAction (73, i);
                        midiObject.midiOutAction (73, speaker4);
                }
                else if (speaker [3] > speaker4)
                {
                        for (int i = speaker [3] - 2 ; i > speaker4 ; i -= 2)
                                midiObject.midiOutAction (73, i);
                        midiObject.midiOutAction (73, speaker4);
                }

                speaker [0] = speaker1;
                speaker [1] = speaker2;
```

```
			speaker [2] = speaker3;
			speaker [3] = speaker4;
		}
		else
		{
			int currFrequency = ballX * 127 / 100;

			if (prevFrequency < currFrequency)
			{
				for (int i = prevFrequency + 2 ; i < currFrequency ; i += 2)
					midiObject.midiOutAction (75, i);
				midiObject.midiOutAction (75, currFrequency);
			}
			else if (prevFrequency > currFrequency)
			{
				for (int i = prevFrequency - 2 ; i > currFrequency ; i -= 2)
					midiObject.midiOutAction (75, i);
				midiObject.midiOutAction (75, currFrequency);
			}

			prevFrequency = currFrequency;
		}
	}


	void promptWindow ()
	{
		try
		{
			sop ("choice: ");
			String choice = BasicIo.readString ();
			int p = 0;

			while (!choice.equals ("q"))
			{
				if (choice.equals ("b"))
				{
					sop ("button: ");
					int b = BasicIo.readInteger ();
					midiObject.midiOutAction (b, 127);
					midiObject.midiOutAction (b, 0);
				}
				else if (choice.equals ("p"))
				{
					sop ("parameter: ");
					p = BasicIo.readInteger ();
					sop ("						parameter: " + p + "\n");
				}
				else if (choice.equals ("v"))
				{
					sop ("value: ");
					int v = BasicIo.readInteger ();
					midiObject.midiOutAction (p, v);
				}
				else if (choice.equals ("m"))
				{
					sop ("value1: ");
					int v1 = BasicIo.readInteger ();
					sop ("value2: ");
					int v2 = BasicIo.readInteger ();
					for (int i = v1 ; i <= v2 ; i += 2)
						midiObject.midiOutAction (p, i);
					for (int i = v1 ; i >= v2 ; i -= 2)
						midiObject.midiOutAction (p, i);
				}
				else if (choice.equals ("ms"))
				{
					midiObject.midiOutAction (8, 127);
					midiObject.midiOutAction (8, 0);
				}
				else if (choice.equals ("fp"))
				{
					midiObject.midiOutAction (14, 127);
					Thread.sleep (100);
					midiObject.midiOutAction (26, 127);
					midiObject.midiOutAction (26, 0);
					midiObject.midiOutAction (14, 0);
				}
				else if (choice.equals ("lp"))
				{
					midiObject.midiOutAction (14, 127);
					Thread.sleep (100);
					midiObject.midiOutAction (27, 127);
					midiObject.midiOutAction (27, 0);
					midiObject.midiOutAction (14, 0);
				}
				sop ("choice: ");
				choice = BasicIo.readString ();
			}
		}
		catch (Exception e)
		{
			promptWindow ();
		}
		stopIt ();
	}


	void makeMidi ()
	{
		midiObject = new TheMidiObject (this);
	}


	public void sop (String s)
	{
		System.out.print (s);
	}


	void stopIt ()
	{
		midiObject.midiportClose ();
		System.exit (0);
	}


	public static void main (String [] args)
	{
		new TheMain ();

	}
}
```

“TheMidiObject.java”

```
// TheMidiObject

// JavaMIDI by: Robert Marsanyi
// Copyright CagEnt Technologies, Inc
// All Rights Reserved


import jmidi.*;

public class TheMidiObject
{
        TheMain m;
        static final int BUFSIZE = 3;
        static byte [] buffer = new byte [BUFSIZE];
        static MidiPort midiport;
        InputThread01 inputThread;

        public TheMidiObject (TheMain m)
        {
                this.m = m;

                int inDevice = 1;
                int outDevice = 0;

                try
                {
                        midiport = new MidiPort (inDevice, outDevice);

                        midiport.open ();

                        inputThread = new InputThread01 (midiport, this);
                        inputThread.start ();
                }
                catch (Exception e)
                {
                        System.err.println ("TheMidiObject error! " + e.getMessage ());
                }
        }


        public void midiOutAction (int p, int v)
        {
                try
                {
                        midiport.writeShortMessage ((byte) 0xBF, (byte) p, (byte) v); //(byte) 0x90
                }
                catch (Exception e)
                {
                        System.err.println ("midiOutAction error! " + e.getMessage ());
                }
        }


        public void midiportClose ()
        {
                try
                {
                        System.err.println ("Checking message queue (should be 0 left) = " + midiport.messagesWaiting ());
                        System.err.println ("Closing port");
                        midiport.close ();
                        System.err.println ("Done!");
                }
                catch (Exception e)
                {
                        System.err.println ("midiportClose error! " + e.getMessage ());
                }
        }
}


class InputThread01 extends Thread
{
        MidiPort midiport;
        TheMidiObject t;

        public InputThread01 (MidiPort aPort, TheMidiObject d)
        {
                t = d;
                this.midiport = aPort;
        }


        public void run ()
        {
                int nBytes;
                boolean done = false;
                byte [] buffer = TheMidiObject.buffer;

                MidiPortMessage msg;

                try
                {
                        msg = new MidiPortMessage (TheMidiObject.BUFSIZE);
                        buffer = new byte [3];

                        while (!done)
                        {
                                try
                                {
                                        while (midiport.messagesWaiting () == 0)
                                        {
                                                yield ();
                                        }
                                        nBytes = midiport.readMessage (buffer, buffer.length);

                                        int statusByte = 128 + buffer [0];
                                        int statusByteX = (statusByte % 16) / 4;
                                        int statusByteY = statusByte % 4;

                                        int x = (int) ((statusByteX * 128 + buffer [1]) * 3.2) - 64;
                                        int y = (int) ((statusByteY * 128 + buffer [2]) * 3.2) - 64;

                                        t.m.midiIn (statusByte < 32 ? true:
                                        false, x, y);
                                }
                                catch (MidiPortException e)
                                {
                                        t.m.c.print ("MidiPort resetting\n");
                                        midiport.resetInput ();
                                }
                        }
                        t.m.stopIt ();
                }
                catch (Exception e)
                {
                        System.err.println ("WHOOPS! error! " + e.getMessage ());
                }

        }
}
```