August 2011

GameWave: An Interactive, Educational Sound Exhibit

Rhiannon Chiacchiaro
Worcester Polytechnic Institute

GameWave: An Interactive, Educational Sound Exhibit

Project Number: FB02

A Major Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Art

in Liberal Arts and Engineering

Music and Technology Concentration

by

Rhiannon Chiacchiaro

_____

Date: August 22, 2011

Approved:

_____

Prof. Frederick Bianchi, Project Advisor

_____

Date

Abstract

Sound art is an increasingly relevant and creative branch of art, exploring various social, economic, and academic ideas in real time. Sound artists look to the latest technology to record, replay, and remix sounds around them or to create completely novel noises. GameWave is a museum installation which uses a game-like interface to turn music into a creative learning experience. Participants create their own music from well-known video game tunes via a classic handheld controller. As they alter the song, they have immediate visual and auditory feedback of the changes to the sounds. The waveform of the song displays itself while the changing music plays from the speakers. Overall, GameWave is a fun introduction to sound waves which encourages participants to create unique music in real time.

Acknowledgments

Thanks go out to the following people and organizations for their contributions to this project: Professor Frederick Bianchi, for his guidance in the project and for providing the speaker system and museum contacts necessary to get GameWave on its feet.

Alexander Goldowsky and the EcoTarium of Worcester for their cooperation and willingness to help get GameWave out into the public sphere.

Halsey Burgund and Nina Simon for their advice and for being easily citable references.

The Willisson family for their unofficial sponsorship of the program, multiple hardware donations, and overwhelming moral and technical support.

Rebecca Baron, Matthew Brenner, Solomon Ross, and Wendy Rowe for their professional and thorough last minute editing.

Table of Contents

Executive Summary

The use of sound in the art world has been growing in importance and relevance for over 100 years. Sound art employs noises, music, and ambient sounds for artistic or educational exhibitions. Harmonics, sound interplay, and multimedia are all important aspects of sound art, and have been since Erik Satie performed his first “Furniture Music” piece in 1902 Paris. Since then, sound as an art form has exploded into many different genres and has become a creative and effective tool for communicating social, technological, and academic concepts.

GameWave is an interactive sound art exhibit which teaches participants about the dynamic nature of sound waves through music. Through the use of a handheld controller, participants can cycle through a song, altering tempo, instrumentation, and pitch. A large speaker setup plays the music, while a display screen shows how the sound waves are affected by these changes. GameWave combines musical creativity with basic physics to teach participants about sound through their own unique adaptations of well-known video game tunes.

The GameWave system was coded in Python, using a special library called Pygame. This library was created specifically for the purposes of game design and features an extensive Musical Instrument Digital Interface (MIDI) and music mixer function database. The songs used in GameWave are free MIDI versions of theme songs from classic NES video games: Super Mario Bros., Tetris, and The Legend of Zelda. Songs may be added to or removed from this list thanks to a song loader and playlist queue.

Due to the various logistics involved in setting up a museum installation, this project will be an ongoing collaboration with the EcoTarium of Worcester, and should be on display some time during Fall 2011. The actual time period for exhibition will be determined at a later date.

Introduction

Artists have long used creative media as a tool for communicating social and academic ideas in a language that is easily understandable and translatable. As artistic technology has progressed through carvings to sculptures and paintings to digital media, artists have been able to refine and enhance their works to varying effect. From sophistication to the avant-garde, from tragic snapshots to comedic interactions, the contents of the world's museums, galleries, and even streets have evolved into a fascinating display of emotional and educational expression.

One of the more recent and increasingly prevalent artistic genres is sound art. An expansive branch of creative design, sound art features works defined by recorded and remixed music, ambient sounds, purposeful cacophonies, and audio/visual relationships. From the earliest experiments with synthesizers, Theremins, and tape loops up through today's advanced music mixing and recording software, sound artists have been breaking technological barriers to find new methods of creative communication. Applications of these methods have been used to explore social issues, such as race and poverty, as well as new methods of learning.

At the Museum of Science in Boston, sound art's societal implications have been seen in the exhibit entitled *Voices Without Faces; Voices Without Races,*[1] a very simple yet poignant installation by Halsey Burgund. In a large, square room, three projectors display a looping video of a drive through Eastern Massachusetts, including Boston and its suburbs. As the drive continues, voiceovers are played of people who were asked to comment on race and its effect on their lives. These people's faces are never shown and they are left otherwise anonymous, but their words and voices combine with the varying backdrops of the Greater Boston Area to present a full picture of race's societal impact.

[1] http://halseyburgund.com/work/vwf/

The above example demonstrates only a small portion of sound art's ability to inspire and educate. The GameWave project seeks to expand these possibilities by allowing users to create their own sound art and instantly receive both educational and artistic results. Through the use of interactive multimedia, participants will be able to first hear and see a song's normal pitch, speed, and instrumentation. As they manipulate any or all of these musical aspects, they can watch as the sound waves transform in response. Once the users begin to grasp the concepts of sound manipulation, they can then continue to change the song to create either a sound or a waveform that they like. By experimenting in this manner, users are able to both learn about the dynamic nature of sound and have the satisfaction of creating their own, unique music.

Historical Background of Sound Art and Digital Music

Sound has been an integral part of the art world since the turn of the 20th century. While music has existed as its own genre since cavemen danced to skin drums, it wasn't until the early 1900's that people really began to explore alternative sound and noise as a legitimate basis for art. The earliest forms of sound art came in the form of ambient music. In 1902, Erik Satie performed a piece that he called "Furniture Music" at the opening of a Parisian gallery.[2] He gathered together a small orchestra and played in a corner of the gallery, begging participants to pay no attention to him and simply continue looking about. This performance ended up being somewhat a failure for Satie, as gallery patrons continued to stop and listen even though they were continuously told to pretend the band was not there. Though his performance was unsuccessful, it paved the way for the extremely widespread use of background music. Today, almost every store, restaurant, and shopping center has some sort of ambient music playing.

Satie's "Furniture Music" was by no means his only work. An eccentric Impressionist trained in classical music, Satie explored variations in music composition that appear at first glance to simply be ways to upset conductors and players.[3] For example, various works of his contained instructions such as "Work it out yourself" or "Light as an egg" and at times had no actual bar lines or recognizable form. This extreme version of avant-garde music is similar to works such as "The Faerie's Aire and Death Waltz" by John Stump, a piece arranged "on accident" with instructions such as "Release the penguins." One of Satie's most famous pieces, "Vexations", was comprised of a single sheet of music which was repeated 840 times.[4]

[2] http://newmusicbox.org/article.nmbx?id=2417
[3] http://www.jazclass.aust.com/satie.htm
[4] http://www.wfmu.org/~kennyg/popular/articles/satie.html

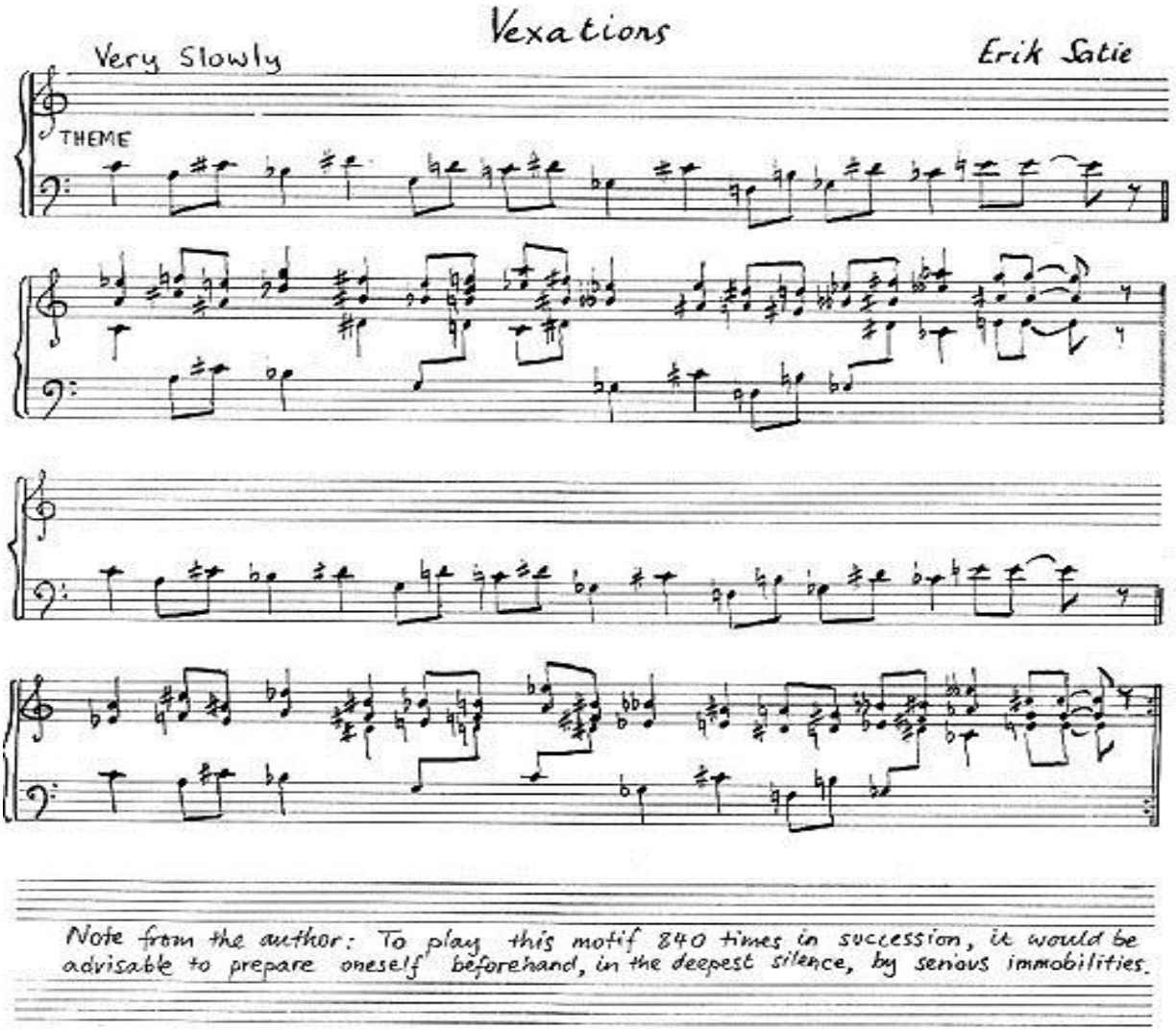

Figure 1: Erik Satie's "Vexations", with a note advising the performer to meditate and prepare for the piece.[5]

Though most musicians were daunted by the task of playing such a piece, it was none other than the famous John Cage who finally put together an ensemble of pianists to play in two hour shifts until, over 18 hours later, the piece was finished.

As the 20th century continued, a group of artists calling themselves Futurists began to take the stage.[6] An Italian group, the Futurists (or *i Futuristi*) were led by F.T. Marinetti, an Italian poet who used his concept of *parole in liberta* ("words in freedom") to give his poetry an

[5] http://www.satie-archives.com/web/jpg/vexscor2.jpg
[6] http://newmusicbox.org/article.nmbx?id=2417

extra tonal quality. His 1926 work "Bombardamento di Adrianapoli" mimicked the sounds of battle with its expressive use of onomatopoeia.[7] Multiple books, articles, and manifestos were written by Marinetti and other Futurists, including Luigi Russolo's *The Art of Noises.*[8] In it, he delves into the exploration of ambient sounds and their inherent musical potential. Russolo categorizes six distinct families of noises that were used extensively by the Futurist movement. Table 1 below lists each of the six families. These six families have remained the basis upon which many sound artists base their work.

1	2	3	4	5	6
Rumbles Roars Explosions Crashes Splashes Booms	Whistles Hisses Snorts	Whispers Murmurs Mumbles Grumbles Gurgles	Screeches Creaks Rumbles Buzzes Crackles Scrapes	Noises obtained by percussion on metal, wood, skin, stone, terracotta, etc.	Voices of animals and men Shouts Screams Groans Shrieks Howls Laughs Wheezes Sobs

Table 1: Russolo's six families of noises.[9]

Sound art remained part of the modern art movement for decades, spilling over into the Impressionist and Dadaist movements. In this forum, artists could expand their creative, sometimes eccentric ideas into a completely new medium. Whereas visual art allowed an artist to show an emotion or idea, sound allowed them to actually tell a story through words and noises. It was during the early to mid-1900's that this experimentation led sound artists to really interlace their work with science and technology. In 1948, French radio broadcaster Pierre Schaeffer

[7] http://youtu.be/Tn0dkz9Polg
[8] http://www.unknown.nu/futurism/noises.html
[9] Ibid.

invented what we know today as electronic music. Named *musique concrete,* Schaeffer created a type of music that took sounds from nature and used tapes and speaker setups to remix the original recordings.[10] From the very first electronic music studio, he took samplings of a variety of sounds, filtering them through tape loops, microphones, and speakers to create music that often gave off a futuristic vibe. In fact, *musique concrete* was the basis for much of the early science fiction movie and television music, including the theme song for the BBC show *Doctor Who*. The song was written by Ron Grainer and recorded by Delia Derbyshire, who used tone generators and tape loops (created by physically cutting and reconnecting ends of magnetic tape into different sized loops) to give the song its signature melody.[11] The enormous popularity of the song fueled the sound art movement and opened new doors to musical experimentation.

The next few decades saw amazing strides regarding technology and the rise of digital music. In the late 1970's and throughout the 1980's, new wave and synth-pop music began to reshape the traditional face of rock and popular music.[12] Bands such as Devo, The Eurythmics, and Soft Cell began to incorporate keyboards and synthesizers into their songs, bringing a more futuristic and electronic sound to their music. Eventually, some synth-pop groups decided to forgo other instruments altogether, relying solely on multiple keyboard synthesizers to emulate everything from percussion to melody and even orchestral harmony. With this new marriage of technology and music, artists began to look even further into what was possible through simple computer manipulation. This revolution made way for the introduction of entirely new genres such as techno, house, and club music. Songs in these genres often employ completely digital sounds, though they may also include recordings of voices or clips from other audio sources.

[10] http://www.musespace.com/writings/essays/musique.html
[11] http://markayres.rwsprojects.co.uk/DWTheme.htm
[12] A History of Rock Music 1951-2000

Figure 2: Devo advertising the Mobile Synthesizer, also called the Keytar.[13]

With these new musical formats, composers and artists became one and the same. Many DJs today, such as the world-famous Dutch artist Tiësto, are also well known for their own songs, which they can conceive, record, and play for worldwide audiences without any external assistance. In fact, much of techno's diversity and popularity stems from the fact that it has become possible and sometimes exceedingly simple to create music from one's own home. Even most lower-end laptops are capable of handling simple music creation and editing software, and keyboard synthesizers have become more compact and available at affordable prices. Of course, a professional setup is much more advanced, but many artists can still fit their entire studio into one or two rooms of their home, as opposed to requiring giant recording studios or churches with full pipe organs.

[13] http://4.bp.blogspot.com/_2JJnXrU37vY/TM2jgjdmdbI/AAAAAAAAARE/YHB8Sh-s-j0/s1600/DevoMoogLiberationMobileSynthesizer1982.jpg

Figure 3: At-home studios like this one have become increasingly common alternatives to full recording booths.[14]

Along with the newly developed hardware, digital music and experimental sound art have led to the creation of music software that can be boiled down to simple programming. Whereas traditional instrumental or vocal music is based on vibrations in the instrument or voice box, music formats such as MIDI (Musical Instrument Digital Interface) can be distilled down to basic data streams and hexadecimal codes. In fact, MIDI programming allows composers, if they so wish, to create entire songs from scratch without needing a synthesizer at all. Free programs such as Anvil Studio even have a built in GUI so that a click of the mouse is all that is needed to create tones that sound like they come from one of 128 possible instruments. Additionally, music editing programs such as Audacity allow the user to alter songs in other formats, such as MP3, through a wide array of musical effects. Between these two types of programs - music synthesis

[14] http://homerecordingstudiomicrophone.files.wordpress.com/2011/02/ed-victor-home-recording-studio.jpg

and editing - digital sound artists have been given the freedom to delve more deeply into complex methods of conveying an artistic idea and creating dynamic, relevant works that would not have been possible with just human beings and instruments.

Today's sound artists are able to run ongoing installations for weeks, months, or even years non-stop. The environments available to them are nigh limitless, with sound art exhibits being run indoors or outside, rain or shine, with or without interaction from passersby. Even searching online for sound art exhibits in most cities will produce lists of installations that have been placed in the area from a wide variety of artists and groups. The Greater Boston Area alone boasts sound art festivals, museum features, and educational workshops on sound art and digital music design.[15]

Without a doubt, sound has become an increasingly vital part of the art world, and has only become more popular as technology advances and people are able to explore their creative sides without needing years of instrumental or vocal lessons. Smartphones, computers, MP3 players, and even custom-developed devices are capable of interacting with music and creating interactive experiences for the listener. Thanks to digital sound and music, the normally cacophonous noises of nature and society are being coalesced and shaped in new ways, allowing artists to hold a mirror up to the face of humanity and explore who and what we truly are.

[15] Google Search: sound artist

The GameWave Project

Overview

The GameWave project was first conceived as an attempt to create an interactive sound design involving a specific speaker setup. Originally envisioned as a type of sound wall, the idea eventually shaped itself into something of an educational simulation. The premise is simple: well-known, classic video game songs will be played through a unique speaker setup. The museum participants can interact with the song via a controller, and see how the sound is changing on a fundamental level via a monitor displaying the real-time waveform.

GameWave requires four main hardware components:

- A computer to store the music and run the program
- A monitor to display the changing sound wave
- A handheld device to alter the music
- A speaker setup to play the music

The specific details of each piece are explained in the Hardware section of the Methodology. Additionally, the program itself is coded using the Pygame library for the Python programming language. More details on the program itself can be found in the Software Section of the Methodology, and the entire code is available in Appendix B.

Because GameWave works with MIDI files, there were many qualities that could be altered in the track. After considering which pieces of a song would be the most educational, noticeable, and engaging to play with, it was decided that the exhibit would focus mainly on pitch, tempo, and instrumentation. Pitch and tempo are the two most basic qualities of music, even beyond rhythm or melody. Every note that is played is nothing more than a specific pitch played for a certain duration, which is dictated by the size of the note, the time signature, and the

tempo of the song as a whole. Instrumentation was also selected because of the drastic ways that it can affect the overall sound quality of a song. There is a distinct difference between hearing a note played on a flute and hearing that same note played on a guitar or pipe organ. Instrumentation also adds to the entertainment factor, as it allows for some humorously out of place sounds to play. For additional customization, the controller can pause the current song and can scroll through the playlist. An idle mode has also been implemented, meaning that after five songs have been played with no user interaction, the music will stop until someone else picks up the controller and presses the start button. This additional functionality is mainly for environmental courtesy, so that the songs are not continuously playing when nobody is around the exhibit. As an additional setup feature, the UI is designed so that the program runs automatically when the computer is turned on at the beginning of the day, with a backup icon in case the auto-loader fails.

GameWave goes above and beyond the idea of a simple sound installation. Instead of trying to make an observation about culture or explore a new musical technology, GameWave was created to be an interactive learning experience about the nature of sounds. Blending art, education, and human interaction in this way is extremely important, as it helps further understanding about the way we are able to absorb, process, and retain information.

Educational games and simulations are some of the most effective teaching tools that we have, partly due to the high amount of stimuli surrounding them. In her book, *The Participatory Museum*, interactivity expert and WPI alumna Nina Simon writes, “Interactive exhibits, when successfully executed, promote learning experiences that are unique and specific to the two-way

nature of their design."[16] It is important to recognize this "two-way nature" because of the manner in which it stimulates learning.

In recent years, there has been an emphasis on the fact that people learn in different ways, be it visually (by seeing), aurally (by hearing), or kinesthetically (by doing). GameWave, along with many other successful learning exhibits, is able to teach effectively by engaging all three of these types of learning. By having a physical controller in their hands, users decide which sound qualities they are changing at any given time. This gives them a better grasp of the situation because they can more easily map a specific button to a corresponding change in sound and from there deduce how to recreate that change. From an auditory standpoint, the instant feedback from the speaker system allows the user to hear what is happening as they experiment with the different buttons, instead of just being told what theoretically could happen to pitch, volume, etc. if certain sound qualities were changed. Finally, the sound wave output allows users to physically see why the sounds are changing the way they do. By being able to see the different sizes and shapes of the waves, they can more easily identify how musical qualities like tempo and pitch correlate and better understand the real world effects of scientific terms such as frequency and wavelength. Through this combination of learning and playing, GameWave provides a dynamic, unique experience for every user, and makes exploring the physics of sound easy and entertaining.

[16] http://www.participatorymuseum.org/chapter1/

Main Demographic and Installation

GameWave is an effective educational and entertainment tool for users of a broad age range. Though it was primarily designed for pre-teens and early adolescents, the interface is simple enough that younger children can enjoy the process of making their own music, even if they don't understand the science behind it. Older participants can also get enjoyment out of it thanks to the nostalgia of using a Nintendo controller and being able to play with classic Nintendo Entertainment System (NES) songs. Due to the fact that the exhibit can work simply by experimenting with buttons, as opposed to requiring a specific series of steps to function, learning can begin almost immediately after picking up the controller. Between the simplistic interface and the controller button map that will be provided, users of any age will be able to quickly pick up on the various functions of each button. Even young children who don't understand the musical terms will be able to learn the definitions simply by seeing, for example, that the buttons labeled "Tempo" make the song go faster or slower, a change which is instantly recognizable to the ears.

Because of the target age range and the hands-on approach that GameWave uses, the project will be installed at the EcoTarium, Worcester's local science and nature museum.[17] The EcoTarium is an indoor and outdoor museum with live animals as well as interactive exhibits. Exploring a broad range of subjects, from air currents and weather patterns to minerals and microorganisms, the museum is a local fixture and is extremely popular with the younger demographic. In addition to having the benefit of being close by, the EcoTarium also prides itself on its interactivity. Multiple temporary and permanent exhibits inside the building contain hands-on games, demonstrations, or experiments. Visitors are encouraged to touch many of the pieces

[17] http://www.ecotarium.org

on display, and some exhibits even allow patrons to be part of the creative process by letting them, for example, make their own bubbles or paper helicopters. This philosophy of learning by doing syncs up perfectly with the point of GameWave, and thus is the perfect location to implement the installation.

Methodology

In order to create an installation worthy of display in a museum, the first step was to create a well-defined set of guidelines and requirements for the project. After experimenting with MIDI applications, exploring the most vital qualities of sound and digital music, and determining the best methods for simultaneously teaching and entertaining, the following set of goals was devised.

The GameWave system must be able to:

- Entertain and educate in a brief period of time without being overwhelming
- Receive input from a handheld video game controller
- Update a MIDI file of a classic video game song in real time based on said input
- Alter the following qualities of the song:
    - Pitch
    - Tempo
    - Instrumentation
- Display a visual representation of the current song alterations
- Have an interface that can be learned within a few seconds of being introduced
- Be turned on daily by museum operators with minimal required setup
- Make sense as a cohesive, artistic piece

Throughout the course of this project, all of the objective expectations were met, though some of the more subjective goals, such as entertainment, can only be tested once the project has been in the museum for a while. In addition, extra features, such as pause, song select, and idle mode were also successfully implemented.

Hardware

As was discussed earlier, the GameWave setup requires four main hardware components to run. The following details the specific piece that was used as well as how it was obtained and prepared for use in the GameWave installation.

Main Computer

The main computer for the project had a few vital requirements. First, it would need enough memory to be able to hold an instance of the Ubuntu operating system, the Python program, any MIDI synthesizers that might have to be added in the software, multiple Python library documents, and the MIDI and configuration files for the song. Together, these programs and files require at least 256 MB. For the music aspect, there must be a sound card that is at least capable of playing MIDI songs and, if it didn't come with a MIDI synthesizer (as some older computers do), the CPU must be capable of processing MIDI via a program such as TiMidity. Additionally, both a microphone and speaker port were necessary as well as a VGA connection to ensure that both the audio and visual outputs would function properly.

Figure 4: The GameWave computer tower.

The computer that GameWave uses is a Dell Inspiron 570. It has 4GB of DDR3 RAM and an AMD Athlon II X2 processor with a clock speed of 2.9 GHz. The SATA hard drive is 500GB and there is 7.1-channel audio support. There are VGA and HDMI connections, as well as multiple USB ports and a microphone and speaker jack. This computer tower is brand new and was purchased for the project by the Willisson family. In order to prepare it for GameWave, a new OS was installed, as well as various programs for program writing, debugging, and playing MIDI (see Methodology: Software).

Monitor

There were only two main considerations when looking at the choice of monitor in terms of functionality. First, it needed to be capable of displaying enough color to show the sound wave in a vibrant and clear image. Secondly, it needed at least a VGA connection. As these requirements are basically standard in most working monitors today, finding a monitor was simple.

Figure 5: The GameWave monitor.

The monitor used in GameWave is a Daewoo C729BBK CRT monitor. It has a VGA connection and is capable of more than high enough resolution for the purposes of the project. This monitor was also donated, courtesy of the Willisson family. Nothing special needed to be done to make it work, and the monitor itself already came with a VGA cable to attach to the computer.

Controller

For the GameWave controller, there were a few possible models that would have been feasible to use. The basic requirements for it included a design that didn't have too many confusing buttons and joysticks, but still had at least five buttons (two for pitch and tempo each and at least one for instrumentation changes). Anything beyond that just allowed for additional functionality. Additionally, the controller had to specifically be for a computer, preferably with a USB connector.

Figure 6: The GameWave controller.

The controller that GameWave uses is a Retrolink NES controller with USB connection. It is a very basic controller, with one directional pad, start and select buttons, and two action buttons. The NES design is a perfect choice for multiple reasons. First, it goes along with the classic NES game music that the project plays, making it thematically appropriate and adding to the nostalgia factor for older users. Secondly, it is extremely simplistic in design, meaning that no extraneous buttons will be left over. Finally, it was extremely inexpensive and easy to find online. This is an important quality, because in a setting like a children-friendly museum, there is always the risk of pieces breaking and needing to be replaced. Additionally, this specific controller works as a Plug-and-Play device, meaning there was no extra software that needed to be purchased or installed to make it work. In order to use it in the program, all that was needed was a basic set of button mapping functions (see Methodology: Software for the button mapping used in the project).

Speakers

The speaker system was the basis for the entire GameWave project. As such, the project itself was built around the speakers that we already had. Since there was a choice of available speaker setups for the project, the decision came down to a few factors. There had to be at least two full speakers to allow for some amount of sound immersion, and just for sake of having an appropriate volume for a crowded museum full of children. Also, it needed to be able to be connected directly into a computer audio jack. The final consideration was aesthetics. This was not so much a problem with the computer and monitor since they will ideally be placed inside an exhibit booth of some sort. However, the idea of a sound wall was one of the foundations of the project, so having visible speakers was important.

Figure 7: The GameWave speakers (1 of 2 shown).

The final speaker setup is actually a left over design from a previous project completed by a group of WPI students. The piece is made of two frames built from wood and copper tubing, through which ten tweeter and two woofers each are strung. The back is also fitted with a strip light, giving the setup a bluish glow when lit. This minimalistic and artistic design fits in perfectly with the GameWave design, as the slightly angled piping is actually somewhat reminiscent of the original Donkey Kong game, which can help add to the enjoyment for older participants. These speakers were provided by Professor Bianchi and the WPI music department and only minor repairs had to be done (such as reaffixing some speaker heads to the pipes) before it was ready to be plugged in to the final system.

Software

Python and Pygame

In order to successfully code the GameWave program, the language used had to meet a group of requirements for ease, compatibility, and versatility. One of the most important aspects of the language was that it needed to employ syntax that a non-programmer could be comfortable using, because the project did not have a heavily experienced coder working on it. Additionally, the language had to include libraries or add-ons that would provide audio and visual support and could work with MIDI files. It was also necessary for there to be some sort of joystick configuration library or an easily customizable method of taking input from a USB controller. Finally, the finished program had to be something universally available and executable on modern hardware.

Taking all of these stipulations into account, the option that immediately jumped out was Python. An open-source programming language, Python has a very basic syntax structure, as well as a wide array of user and developer-created libraries for a variety of applications. One such library is Pygame,[18] which allows users to create well-developed and highly customizable computer games. Though the graphics capabilities are somewhat low-level, they are more than sufficient for the requirements of GameWave. Additionally, Pygame comes complete with joystick compatibility, and can program all aspects of any modern controller, including joysticks, direction hats, buttons, and triggers. The aspect that really settled the decision, however, was the extensive MIDI library. Pygame contains functions that allow the programmer to write out actual MIDI messages and send them to a file that is being streamed in real time. Thanks to this combination of input and output capabilities, as well as the large internet database of relevant,

[18] http://www.pygame.org/

well-documented programs and functions, Pygame became the clear choice for the GameWave project.[19]

GameWave uses three Pygame libraries: pygame, pygame.midi, and pygame.font. Pygame is the basic library, which holds all of the joystick, display, and standard audio output functions. The MIDI library is a special addition which allows the MIDI data to be written directly into the song file as it streams. The font library is a small peripheral library which holds a set of text fonts. This library was used to get a font for the idle screen, which displays "Press START to continue" along with a button map whenever the music isn't playing. Additionally, the program uses the MidiInFile, midi_events, os, and pyaudio libraries. MidiInFile and midi_events are both pieces of an online add-on written by a Python user[20] to help parse MIDI files and create useful Python variables from them. The os library is a Python standard, which was used to help create an operating system-specific directory which would detect and organize the MIDI files that were loaded into the playlist. Finally, pyaudio is another Python library which creates Python bindings for PortAudio, an audio library that the project runs in the background.

The main program process is broken down into four pieces: the playlist class, the controller alteration functions, the screen display, and the idle mode function. The playlist was created by making a class which stores relevant information about each song. By creating corresponding configuration files for each MIDI file, the class is able to read the main MIDI channels used for the song melody, the original instrument of that channel, and a multiplier which determined the tempo of each song. The class itself is comprised of three separate functions. The first is an initialization function which opens and reads the configuration files to get the pertinent data from them. While this is happening, the MIDI files themselves are placed

[19] Note: The following paragraphs describe the overall structure of the program. The full source code is available in Appendix B.
[20] http://www.mxm.dk/products/public/pythonmidi

into the playlist via the load function. Finally, the play function takes the song and prepares to play it at the appropriate tempo. The program starts out with the music on pause, and the play function is called whenever the program is unpaused or taken out of idle mode. The proper construction of the playlist and play functions also requires certain subfunctions in the main body of the program, which process and queue the MIDI data note by note. This is necessary because it allows for real time modifications and only allows the most recently updated notes to play.

The rest of the main body of the program is mostly dedicated to the controller functions that cause the alterations within the songs. Each button on the NES controller has a specific action assigned to it (see Figure 8 below).

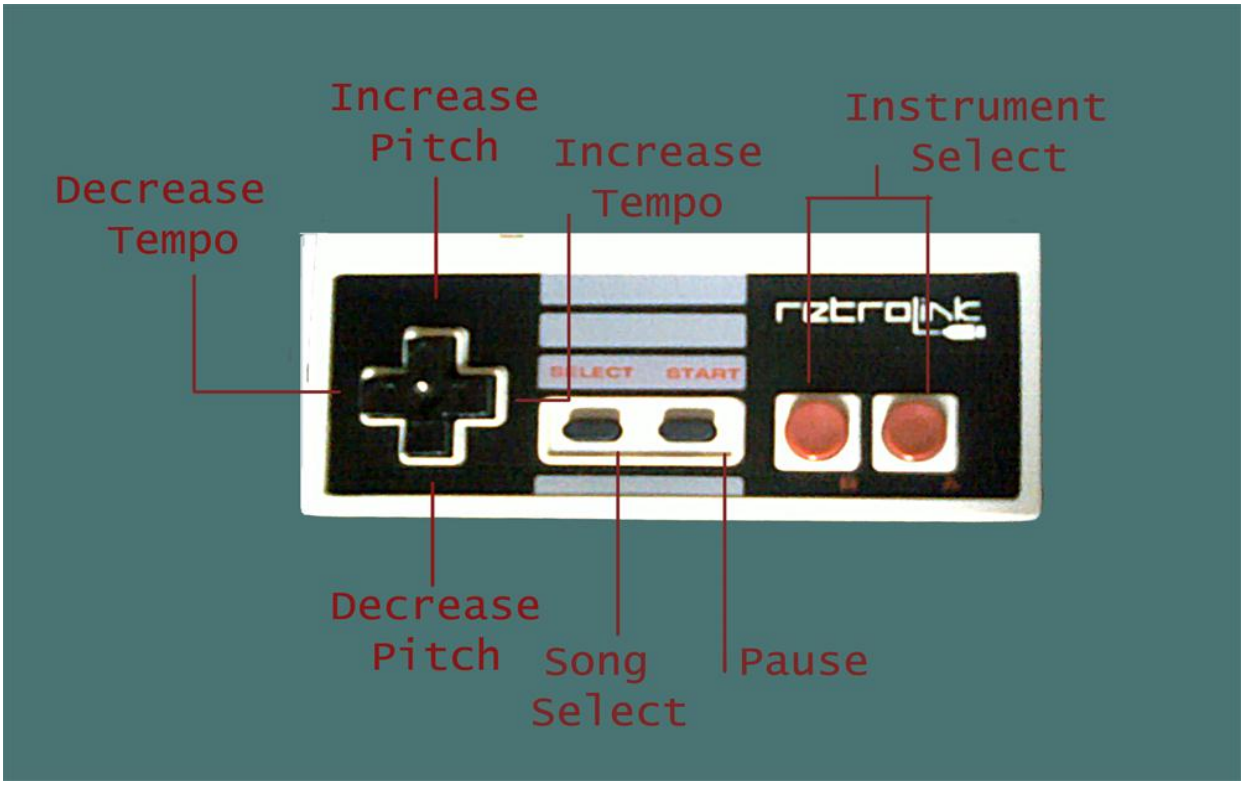

Figure 8: The proper button mapping for the Retrolink NES Controller.

In order to properly map the buttons to their corresponding actions, each button press was assigned an event. First, the button numbers were determined by using jstest, a built-in Linux function that produces a simple readout of which button, joystick, or directional hat is being

activated for a plugged in controller. Once each input had been identified, Pygame events were written for each case. GameWave requires two types of events: button events and axis events. The buttons used in GameWave are A, B, Start, and Select, while an axis is assigned to the left and right (x-axis) and up and down (y-axis) directional buttons. Axis events are also used on the joysticks of more advanced controllers, with the direction pad usually being configured as a directional hat, or joyhat. From there, it was a simple matter of finding the MIDI program codes[21] and sending the appropriate MIDI messages to the speakers via the midi_out.write_short command. For the select button to scroll through the songs, the play function was called as if the previous song had just finished.

Once the audio portion was completed, the visual display was drawn and the sound wave synthesized with Pygame's screen and oscilloscope functions. Pygame's audio oscilloscope displays sound captured from a computer microphone as a real-time sound wave.

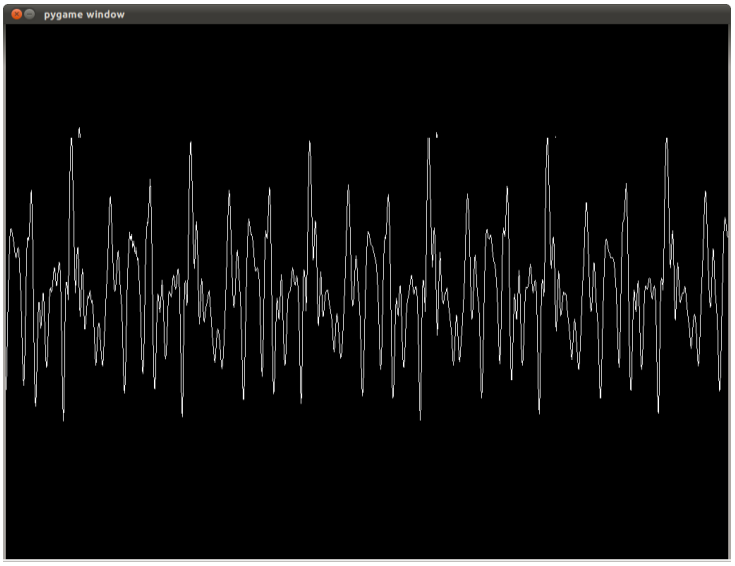

Figure 9: Oscilloscope output of a captured sound wave.

The problem with using an actual microphone to pick up the music is that the oscilloscope would also grab any ambient sound, causing a wave distortion. In order to properly connect the audio

[21] http://www.srm.com/qtma/davidsmidispec.html

and visual ports without causing interference, a cable splitter connected the speaker output to both the speakers and the microphone jack. The pyaudio library allowed for proper communication between the speakers and microphone, ensuring that the data streamed correctly. After that, the programming for the sound wave was straightforward, using the pygame.draw.line function to create a wave based on the data going into the microphone port.

The final piece of programming necessary was the addition of an idle mode. Since it can be a distraction to patrons to have the music continuously playing when nobody is using the exhibit, the program is set to pause itself after five songs have gone by with no user input. This was implemented via a small counter that increments every time the play function calls for a new song. Touching any button on the controller resets the count to zero. After the counter reaches five, the notes stop and a small text box appears, displaying the phrase "Press START to continue" above the aforementioned button map in the center of the screen. Once the start button is pressed, the idle screen disappears and functionality resumes as normal from the spot that the music left off. Because the message was simple to create and can help users who may be confused, the condition was set so that it would also appear at startup and whenever the pause button was pressed. This final piece of programming, though not stated as one of the primary goals, adds to the project's usability and lends it an extra sense of gameplay.

MIDI Software

Because the hardware does not contain built-in MIDI synthesis hardware, a separate program was needed to act as the intermediary between the Python program and the speakers. Fortunately, the Ubuntu operating system has a software repository which contains TiMidity,[22] a fully configured MIDI synthesizer that works by connecting to Advanced Linux Sound

[22] http://lau.linuxaudio.org/TiMidity-howto.html

Architecture and PortMIDI, the two audio outputs to which the GameWave program connects. Once TiMidity was installed, the audio output was fully connected and played without error.

In addition to using TiMidity, a free MIDI composition program called Anvil Studio[23] became the platform for testing the MIDI files. In order to determine the channels, instruments, and tempos of the songs for the purpose of creating configuration files, the songs were opened in Anvil Studio and examined track by track. Anvil Studio was also useful any time there was a playback bug and the audio needed to be tested using a different application.

Operating System

From the beginning of the project, it was obvious that Linux was the best choice in terms of an operating system. Besides the fact that it is open-source, meaning it would add no cost to the project, Linux contains a vast repository of software programs and add-ons for various applications. Also, viruses and malware are very rare on Linux, making it a generally safe OS to run. Additionally, most distributions come with relatively few preloaded programs, ensuring more space on the hard drive and a faster processing speed. Finally, Linux has a very open-ended interface, and the built-in Secure Shell client allows the final computer to communicate easily with the other computer upon which the program was being written.

There are many distributions for Linux, each with their own benefits, but the system chosen for GameWave was Ubuntu 10.04.3 Long Term Support (LTS).[24] This version is the most recent LTS edition to be released (Ubuntu releases a new upgrade every six months or so, and a new LTS version every two years), meaning the software repositories and security upgrades will effectively remain active for as long as the installation is on display. Ubuntu is one of the more well-known Linux distributions, and is among the more user-friendly versions.

[23] http://www.anvilstudio.com/
[24] http://www.ubuntu.com/download/ubuntu/download

Additionally, the Ubuntu software repository is expansive, and contained everything the project required for the MIDI output, audio/visual connections, and controller testing. From a hardware standpoint, it was trivial to hook up the visual and audio components since Ubuntu does not have hidden utilities for devices such as speakers and microphones like some larger operating systems do. The final argument for Ubuntu is that there are setup utilities inside the OS that will allow GameWave to launch automatically when the computer is started up, eliminating the need for a museum attendant to manually set the program up every morning.

Installation was also very straightforward; the OS was simply written onto a CD and then installed via the CD onto the project computer. After running through initial setup and downloading the necessary software, all of which took under an hour, the computer was completely set up and ready to run the program.

Results

Overall Results

In general, the outcome of the GameWave project was remarkably positive. The combination of art and technology led to a more stimulating and thus educational experience, and the promise of actually displaying the final product provided impetus for a higher quality piece. By using pre-existing resources combined in new ways, GameWave helped to expand knowledge in programming, digital music, and educational theory.

Apart from personal accomplishments, GameWave also demonstrated that it was in fact possible to create a working product capable of being presented to the public within a small time frame and on a very low budget. Though early planning for the project began in the spring, the complete design and creation of the actual piece, including all of the coding, took place throughout the course of about seven weeks. Thanks to donations and previously owned materials, the final cost of the project only came out to be around $400, not including any additional materials that may be required to set up the actual exhibit.

This project also demonstrated a new way of using the Pygame library. Even though the point of GameWave was to create a simulation as opposed to a full-fledged game, the audio and visual components of Pygame were used to great effect and the hookup of the speaker and microphone jacks presented an alternate use for Python's oscilloscope software. The controller functions also demonstrated an alternate purpose, since normally the joystick and buttons are used to control a visual aspect of a game, usually a character on a screen. In GameWave, the buttons are responsible for controlling sound, and the sound wave visuals were just a display of the results.

Obstacles and Solutions

Despite the positive outcome of the project, there were some problems that presented themselves along the way. These issues dealt mainly with hardware, software, and project organization. Fortunately, each problem was able to be dealt with in a satisfactory manner.

Hardware

The biggest hardware issue had to do with the budget. WPI did not provide any money for the project, so most of the pieces had to be obtained independently. The hardware had to be both affordable on a college budget and hardy enough to withstand the constant flow of children that would be passing by it (or at least inexpensive enough to be replaced without too much concern). Additionally, the main computer requires a certain amount of processing power and audio capabilities in order to properly run the program.

To successfully gather the components, there was a considerable effort put towards gathering donations and searching out affordable pieces. The speakers, luckily, were already provided, which was a big help due to the potential cost of a good speaker system. Additionally, the project received a great deal of help from the Willisson family, who donated a computer and a monitor to the cause. When it was discovered that the first computer was too old and did not have enough RAM for the program, they delivered more RAM to install. When the additional RAM made it clear that the processor was not powerful enough to run TiMidity, they replaced the system with a newer model. Unfortunately, this tower had a disc error and could not load Ubuntu, or any OS, properly. After trying to assist in troubleshooting, they then offered $400 to purchase the Dell Inspiron that is currently being used. Apart from that, the only pieces that needed to be purchased were the controller and cable splitter. As was mentioned in the hardware methodology, a USB version of the NES controller is inexpensive and relatively easy to find. It

is not the highest quality controller, but only cost around $8, shipping included, and is easily replaceable if need be. The cable splitter was also an inexpensive purchase, and did not require any extraordinary effort to obtain.

Software

As with any programming-based project, GameWave required a bit of troubleshooting. There were a few bugs in the program and auxiliary software that led to some setbacks throughout the project. The first hitch was encountered when the MIDI files were unable to play correctly because Pygame's MIDI mixer did not allow real-time editing. The problem was fixed by finding a library that allowed for direct MIDI messages to be written instead of having to pass commands through the mixer.

The next big setback was trying to create the queue properly. Originally, only the first song file would play instead of the whole playlist. This was an easy fix, since there was a playback clock that had not been resetting itself to zero. Resetting the clock and fixing a bug in the MIDI loader to clear data about the previous song solved the problem.

Another issue was related to Ubuntu settings. At first, it did not want to recognize Pygame, so the computer was unwilling to play sound. After adjusting the SDL_AUDIODRIVER settings in Ubuntu, Pygame was able to play the sound correctly.

Finally, the computer had trouble feeding data into the microphone and displaying the proper real-time sound wave. Specifically, the wave would stop flowing even when the music was still playing. The solution to this problem was to lower the quality of sound read from the microphone and ensure that the CPU usage was optimized so the system wouldn't become overtaxed and freeze.

Other

Outside of the hardware and software issues, there were some obstacles related to communication and time restrictions. The first problem arose while trying to get music files for the project. Originally, collaboration with the Video Game Orchestra was in the works, but due to logistical issues and slow response time, it eventually fell through. Instead, more basic MIDI files were pulled from the internet,[25] which turned out to be a better idea since MIDI is much easier to alter in Pygame than the MP3 or WAV files that would have likely been provided by the VGO.

Additionally, extenuating circumstances took out a major portion of time in the middle of the summer. Because of this, the program was not finished until August, at which point there was not enough time to get the installation running at the EcoTarium before the project submission date. This was an unfortunate but unavoidable turn of events. However, it is not a total loss, because now the project can continue as an ongoing, long-term collaboration with the museum.

Possible Future Projects

GameWave is the first in a potential series of projects related to digital sound design, educational art, and interactive media. There are multiple possible expansions for the GameWave setup to enhance the educational capabilities of the project, make the program more customizable, or allow for more in-depth user interaction.

One possibility is to make the playlist more easily expandable. As it currently stands, in order to add a song to the playlist, a configuration file must be made in the folder to capture necessary data. A subprogram that could create that file automatically would help immensely,

[25] http://home.swipnet.se/~w-22134/nmm/mitten.html

especially if there is a large batch of songs to be added to the list. Additionally, there are many more musical alterations possible for the song itself. Changes in distortion, panning, and volume would provide extra educational and entertainment value, as would the ability to alter the harmonies and rhythms in addition to the melody lines. Going along with that, an extra on-screen interface could be devised so that users could decide which features they would like to alter. Upon selection, a customized button map would appear, telling them which button affects which sound quality. This would be a project for an older, more advanced demographic, as it would require some previous knowledge of the musical terms being altered. As a final thought, it would be interesting to create some sort of log that would record how the users interact with the program, which aspects they change, and how it affects the sound wave. If future budgets allow it, perhaps all users could get a summary printout of how they changed the music with screen captures of the changing wave. This could possibly go along with a set of achievements related to making a wave of a certain size, shape, or frequency. This sort of achievement system would make the exhibit seem even more game-like and give children something to literally take away with them.

Conclusion

The GameWave project is the start of an exploration into musical education and interactivity. A combination of art and technology, the piece is meant to entertain users while giving them multiple forms of educationally valuable feedback related to the properties of sound in music. The full impact of the exhibit will be made clear in the months following its completion and installation in the EcoTarium, where it will be made available to the general public. As a first attempt at educational sound art, its effectiveness is not guaranteed. However, unofficial tests with friends who are not music-literate have demonstrated that the interface is indeed intuitive. With the aid of a quick reference guide and the visual and auditory feedback, they were able to understand the button map and notice changes in the sound wave.

GameWave was a very effective culmination project in that it combined previous knowledge, original ideas, creativity, and fast-paced learning in a way that produced a tangible outcome. The open-ended and personally relevant subject matter enhanced the learning experience and made for a project that was mentally stimulating and interesting to explore. GameWave is the sort of project that works well as a finished product, but which can be enhanced and built upon as technology and learning ideologies evolve. Additionally, the compilation of music theory, computer programming, interactive media, and community involvement created a fully comprehensive experience. For these reasons, the GameWave installation is an outstanding success. In terms of educational value, this project was definitely a level up.

Appendix A: Glossary of Terms

Sound artists understand that when creating a piece, regardless of how tuneful they wish to make it, they must take certain musical qualities into account. The following is a brief glossary to help familiarize the reader with the concepts and terms that were explored with GameWave, and which may be on the mind of any sound artist or digital musician when creating an installation. The list is not exhaustive, but includes many of the terms that were discussed throughout this report.[26] [27] [28]

Acoustics: the scientific study of sound, its nature, and its properties

Amplification: the increase in magnitude of a signal, in this case, a sound wave

Audio Frequency: the rate of change in a standing wave, measured in cycles/sec or Hertz (Hz)

Damping: the minimization of sound wave vibrations, leading to a muffled sound

Digital Music: any form of music that is created mainly through discrete, digital signals and electronic generation, as opposed to recorded acoustic instruments

Distortion: interference in a digital sound wave that causes it to stray from the intended sound

Duration: the length of time a sound plays, broken into nine time scales in the book Microsound

Harmonics: the sounds which are multiples of, and interact with, a base frequency

Instrumentation: the selection of instruments that are used in a musical piece

Melody: the main line of a song, often the most predominant track

MIDI: Musical Instrument Digital Interface, a digital music standard which uses instruments, programs, and data streams to create, edit, and play purely digital sounds

[26] The Digital Musician
[27] http://www.cardinalproaudio.com/main/terms.htm
[28] http://www.owenscorning.com/around/sound/glossary.asp

Mixer: a sound device which combines two or more signal inputs into a single output

MP3: MPEG audio layer 3, a file format which is used heavily in the recorded music industry

MPEG: Moving Pictures Experts Group, overseers of visual and audio encoding standards

Noise: can refer either to an unwanted sound in an audio sample, or the small extraneous distortions that come from the environment or electrical interferences

Pitch: the perceived frequency of a given sound, based mainly on the perception of the listener

Sound: a pressure wave that excites the inner ear and causes an audible result for the listener

Sound Art: a subculture of fine arts that focuses on audio as opposed to visual media

Synthesizer: an instrument, often times with a keyboard interface, which works not by instantly creating acoustic vibrations, but by sending digital signals to a mixer and speaker setup

Tempo: song timing, defined by how many beats are played per time period (often one minute)

Timbre: a sound's quality that distinguishes it musically based on harmonics, pitch, and volume

Volume: the perceived loudness of a sound, subjective based on the listener

Wavelength: the physical size of a wave from point to point

Appendix B: Program Code

```
#GameWave: An Interactive Educational Sound Exhibit
#By Rhiannon Chiacchiaro
#Program Code

#Imports and initializes pygame and other necessary libraries
import pygame, pygame.midi, pygame.font
pygame.init()
pygame.midi.init()
pygame.font.init()
from midi.MidiInFile import MidiInFile
import midi_events
import os
import pyaudio

#Creates a class playlist for all of the songs used
class Song:
	filename = None
	channel = None
	default_ins = None
	delay = None

	def __init__(self, filename):
		self.filename = filename
		confname = filename[:-3] + "conf"
		f = open(confname)
		confdata = eval(f.read())
		self.channel = confdata["channel"]
		self.default_ins = confdata["default_ins"]
		self.delay = confdata["delay"]
		f.close()
		self.load()

	def load(self):
		self.events = midi_events.MidiEvents()
		midi_in = MidiInFile (self.events, self.filename)
		midi_in.read()
		self.data = self.events.event_list

	def play(self):
		global midi_data
		global main_channel
		global delay
		global program_time
		global note_index
```

```
                global idle_count
                midi_data = self.data
                main_channel = self.channel
                delay = self.delay
                program_time = 0
                note_index = {}
                for index_channel in midi_data.keys():
                        note_index[index_channel] = 0
                idle_count += 1

songlist = []

# Loads songs into the song list for playing
def midi_test(test_file):
        if test_file[-3:] == "mid":
                return True
        return False

def load_songs():
        files = os.listdir("Songfiles")
        midis = filter(midi_test, files)
        for f in midis:
                songlist.append(Song("Songfiles/"+f))

#Turns on the joystick controller.
jstick = pygame.joystick.Joystick(0)
jstick.init()

#Initializes variables
ins_list = [0, 2, 4, 6, 13, 19, 21, 23, 25, 34, 46, 56, 58, 65, 68, 71, 73, 75, 79]
next_ins = -1
stop = True
stop_time = 0.0
on_notes = []
program_time = 0
last_midi_time = 0
off_channel = 0x80
delay = 2.5
pitch_value = 0x40
main_channel = 0
song_number = 0
width = 1024
height = 480
screen = pygame.display.set_mode((width, height))
note_index = {}
```

```
idle_count = -1

#Loads button map
button_map = pygame.image.load ("controller_map.png")
button_map = pygame.transform.scale(button_map, (width/2.5, height/2))

#Configure and initialize the microphone for oscilloscope data
chunk_size = 1024
audio_format = pyaudio.paInt16
num_of_chans = 1
audio_rate = 44100
audio_config = pyaudio.PyAudio()
mic_in = audio_config.open(format = audio_format, channels = num_of_chans, rate =
audio_rate, input = True, frames_per_buffer = chunk_size)

#Set up font for idle mode
font = pygame.font.SysFont(u'tlwgtypewriter', 30, bold = True)

#Set up MIDI output port
midi_out = pygame.midi.Output(2, 0)

#Loads the songlist into the program
load_songs()
songlist[song_number].play()

#Main functions
quitflag = False
last_midi_time = pygame.midi.time()
status = {}
try:
        while quitflag == False:
                for event in pygame.event.get():
                        if event.type == pygame.QUIT:
                                #Quit the program
                                quitflag = True
                        if event.type == pygame.JOYBUTTONDOWN:
                                idle_count = 0
                                #Stop/Start
                                if event.button == 9:
                                        if stop == False:
                                                stop = True
                                                # Stop all currently playing notes
                                                for note in on_notes:
                                                        midi_out.write_short(off_channel+note[1],
note[0])
                                        elif stop == True:
```

```
                                stop = False
                                last_midi_time = pygame.midi.time()
                        # Scroll through songs
                        if event.button == 8:
                                for note in on_notes:
                                        midi_out.write_short(off_channel+note[1],
note[0])
                                song_number += 1
                                if song_number > len(songlist)-1:
                                        song_number = 0
                                songlist[song_number].play()
                        # Change instruments
                        if event.button == 2:
                                next_ins -= 1
                                if next_ins <= 0:
                                        next_ins = len(ins_list) - 1
                                if type(main_channel) == int:
                                        midi_out.write_short(0xc0+main_channel,
ins_list[next_ins])
                                elif type(main_channel) == tuple:
                                        for ins_channel in main_channel:
                                                midi_out.write_short(0xc0+ins_channel,
ins_list[next_ins])
                        if event.button == 1:
                                next_ins += 1
                                if next_ins >= len(ins_list):
                                        next_ins = 0
                        if type(main_channel) == int:
                                        midi_out.write_short(0xc0+main_channel,
ins_list[next_ins])
                                elif type(main_channel) == tuple:
                                        for ins_channel in main_channel:
                                                midi_out.write_short(0xc0+ins_channel,
ins_list[next_ins])

                if event.type == pygame.JOYAXISMOTION:
                        idle_count = 0
                        #Change tempo
                        if event.value == 1 and event.axis == 3:
                                delay -= 0.25
                                if delay < 0.25:
                                        delay = 0.25
                        if event.value < 0 and event.axis == 3:
                                delay += 0.25
                        #Change pitch
                        if event.value < 0 and event.axis == 4:
```

```
                                        pitch_value += 8
                                        if pitch_value > 0x7F:
                                                pitch_value = 0x7F
                                        if type(main_channel) == int:
                                                midi_out.write_short(0xe0+main_channel, 0x00,
pitch_value)
                                        if type(main_channel) == tuple:
                                                for pitch_chan in main_channel:
                                                        midi_out.write_short(0xe0+pitch_chan,
0x00, pitch_value)
                                elif event.value == 1 and event.axis == 4:
                                        pitch_value -= 8
                                        if pitch_value < 0:
                                                pitch_value = 0
                                        if type(main_channel) == int:
                                                midi_out.write_short(0xe0+main_channel, 0x00,
pitch_value)
                                        if type(main_channel) == tuple:
                                                for pitch_chan in main_channel:
                                                        midi_out.write_short(0xe0+pitch_chan,
0x00, pitch_value)

                        #Play the next song in the queue automatically
                        song_done = True
                        for channel_key in midi_data.keys():
                        if len(midi_data[channel_key]) > note_index[channel_key]:
                                        song_done = False
                                        break
                        if song_done == True:
                                song_number += 1
                                if song_number > len(songlist)-1:
                                        song_number = 0
                                songlist[song_number].play()

                        # If program is not paused, process and play MIDI
                        if stop == False:
                                # Update time
                                program_time += (pygame.midi.time() - last_midi_time) / delay

                        last_midi_time = pygame.midi.time()
                                # Get next MIDI events
                                for channel in midi_data.keys():
                                        if len(midi_data[channel]) == note_index[channel]:
                                                continue
                                        if midi_data[channel][note_index[channel]][1] > (program_time):
                                                continue
```

```
                                current_data = midi_data[channel][note_index[channel]]
                                note_index[channel] += 1
                                status[current_data[0][0]] = True
                                # Process MIDI data
                                # Store notes that are currently playing for start/stop function
                                if (current_data[0][0] & 0xF0) == 0x90: # Note on command
                                        if current_data[0][2] > 0:
                                                on_notes.append ((current_data[0][1], channel))
                                        else:
                                                note = on_notes.index ((current_data[0][1],
channel))
                                                del on_notes[note]
                                if (current_data[0][0] & 0xF0) == 0x80: # Note off command
                                        note = on_notes.index((current_data[0][1], channel))
                                        del on_notes[note]
                                midi_out.write ([current_data])

                        #Implement idle mode after 5 songs without user input
                        if idle_count >= 5:
                                stop = True
                                # Stop all currently playing notes
                                for note in on_notes:
                                        midi_out.write_short(off_channel+note[1], note[0])

                        #Draws the graphical frame and creates the oscilloscope waveform
                        screen.fill((0,0,0))
                        samples = mic_in.read(chunk_size)
                        for i in range(0, len(samples) - 2, 2):
                                pygame.draw.line (screen, 0x49E9BD,(i / 2, height / 2 + (struct.unpack
("<h", samples[i] + samples[i+1])[0] * height / 2**16)),(i / 2 + 1, height / 2 + (struct.unpack
("<h", samples[i+2] + samples[i+3])[0] * height / 2**16)))

                        #Makes the paused music text box
                        if stop == True:
                                text = font.render("Press START to continue.", False, (0xFF, 0xFF, 0xFF),
(0x00, 0x00, 0x00))
                                textrect = text.get_rect()
                                textrect.center = (width/2, height/5)
                                screen.blit(text, textrect)
                                maprect = button_map.get_rect(center = (width/2, height/2))
                                screen.blit(button_map, maprect)
                        pygame.display.flip()

except:
raise
```

Appendix C: Index of Figures and Tables

References

"A Brief History of Sound Art." *New Music Box*. N.p., n.d. Web. 20 May 2011. <newmusicbox.org/article.nmbx?id=2417>.

"Anvil Studio Catalog." *Anvil Studio Catalog*. N.p., n.d. Web. 22 Aug. 2011. <http://www.anvilstudio.com/>.

Ayres, Mark. "A History of the Doctor Who Theme." *The Mark Ayres Doctor Who pages!*. N.p., n.d. Web. 22 Aug. 2011. <http://markayres.rwsprojects.co.uk/DWTheme.htm>.

Burgund, Halsey. "Voices Without Faces; Voices Without Races." *Halsey Burgund*. N.p., n.d. Web. 22 Aug. 2011. <http://halseyburgund.com/work/vwf/>.

"Download | Ubuntu." *Homepage | Ubuntu*. N.p., n.d. Web. 22 Aug. 2011. <http://www.ubuntu.com/download/ubuntu/download>.

"EcoTarium: A Museum of Science & Nature." *EcoTarium: A Museum of Science & Nature*. N.p., n.d. Web. 22 Aug. 2011. <http://www.ecotarium.org>.

GOLDSMITH, KENNETH. "Flabby Preludes for a Dog: An Erik Satie Primer." *WFMU-FM 91.1/Jersey City, NJ; 90.1/Hudson Valley, NY* . N.p., n.d. Web. 22 Aug. 2011. <http://www.wfmu.org/~kennyg/popular/articles/satie.html>.

Hugill, Andrew. *The Digital Musician*. New York: Routledge, 2008. Print.

"JAZCLASS : About Erik SATIE - the eccentric Impressionist French composer and musician." *JAZCLASS : Jazz Improvisation Lessons, Blues Lessons, Saxophone Lessons, Keyboard Lessons, Jazz Theory and Music Notation Lessons by Michael Furstner*. N.p., n.d. Web. 22 Aug. 2011. <http://www.jazclass.aust.com/satie.htm>.

"Linux Audio Users Guide -TiMidity Howto ." *Linux Audio Users Guide* . N.p., n.d. Web. 22 Aug. 2011. <http://lau.linuxaudio.org/TiMidity-howto.html>.

"museSpace: writings....essays....musique concrete." *Welcome to museSpace!*. N.p., n.d. Web. 22 Aug. 2011. <http://www.musespace.com/writings/essays/musique.html>.

"mxm, IT's mad science - Python Midi Package." *mxm, IT's mad science - mxm - Den Gale Videnskabsmands Hule*. N.p., n.d. Web. 22 Aug. 2011. <http://www.mxm.dk/products/public/pythonmidi>.

N/A. *Erik Satie's Vexations*. N.d *Erik Satie Audio and Video Archive*. Web. 22 Aug. 2011.

N/A. *Devo Advertisement*. N.d. *4.bp.blogspot.com*. Web. 22 Aug. 2011.

"News - pygame - python game development." *News - pygame - python game development*. N.p., n.d. Web. 22 Aug. 2011. <http://www.pygame.org/>.

"Nintendo Midi Music." *Nintendo Midi Music*. N.p., n.d. Web. 22 Aug. 2011. <http://home.swipnet.se/~w-22134/nmm/mitten.html>.

Russolo, Luigi. "The Art of Noises." *The Niuean Pop Cultural Archive*. N.p., n.d. Web. 22 Aug. 2011. <http://www.unknown.nu/futurism/noises.html>.

Scaruffi, Piero. *A History of Rock Music 1951-2000* . New York: IUniverse, 2003. Print.

Shome, Ethe. *At-Home Music Studio*. N.d. N/A, N/A. *Home Recording Studio Microphone*. Web. 22 Aug. 2011.

Simon, Nina. "Chapter 1: Principles of Participation â€“ The Participatory Museum." *The Participatory Museum*. N.p., n.d. Web. 22 Aug. 2011. <http://www.participatorymuseum.org/chapter1/>.

"sound artist - Google Search." *Google*. N.p., n.d. Web. 22 Aug. 2011. <http://www.google.com/search?rlz=1C1TSND_enUS413US413&sourceid=chrome&ie=UTF-8&q=sound+artist#sclient=psy&hl=en&rlz=1C1TSND_enUS413US413&source=hp&q

=sound+art+installation+boston&pbx=1&oq=sound+art+installation+boston&aq=f&aqi=&aql=&gs_sm=e&gs_upl=1150l64>.

"Sound Quality Glossary Terms." *Owens Corning: Making the World More Energy Efficient with Quality Building Products & Fiberglass Composites*. N.p., n.d. Web. 22 Aug. 2011. <http://www.owenscorning.com/around/sound/glossary.asp>.

"Sound Terms." *Cardinal Sound & Communications... Sound Solutions Since 1970* . N.p., n.d. Web. 22 Aug. 2011. <http://www.cardinalproaudio.com/main/terms.htm>.

Van Brink, David. "David's MIDI Spec." *Synthestra Realtime Music | blips and software*. N.p., n.d. Web. 22 Aug. 2011. <http://www.srm.com/qtma/davidsmidispec.html>.

"Zang Tumb Tumb - Tribute to Marinetti- YouTube." *YouTube - Broadcast Yourself.* . N.p., n.d. Web. 22 Aug. 2011. <http://youtu.be/Tn0dkz9Polg>.

MLA formatting by BibMe.org.