


5-21-2013

# iGrooving: A Generative Music Mobile Application for Runners

Daniel J. Lepervanche  
daniel.lepervanche@gmail.com

**DOI:** 10.25148/etd.FI13080519

Follow this and additional works at: <https://digitalcommons.fiu.edu/etd>

 Part of the [Composition Commons](#), [Music Performance Commons](#), and the [Other Music Commons](#)

---

## Recommended Citation

Lepervanche, Daniel J., "iGrooving: A Generative Music Mobile Application for Runners" (2013). *FIU Electronic Theses and Dissertations*. 941.  
<https://digitalcommons.fiu.edu/etd/941>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact [dcc@fiu.edu](mailto:dcc@fiu.edu).

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

iGROOVING:

A GENERATIVE MUSIC MOBILE APPLICATION FOR RUNNERS

A thesis submitted in partial fulfillment of

the requirements for the degree of

MASTER OF MUSIC

by

Daniel Jose Lepervanche

2013

To: Dean Brian Schriner  
College of Architecture and the Arts

This thesis, written by Daniel Jose Lepervanche, and entitled iGrooving: A Generative Music Mobile Application for Runners, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

---

Joel Galand

---

Jacek Kolasinski

---

Paula Matthusen

---

Jacob Sudol, Major Professor

Date of Defense: May 21, 2013

The thesis of Daniel Jose Lepervanche is approved.

---

Dean Brian Schriner  
College of Architecture and the Arts

---

Dean Lakshmi N. Reddi  
University Graduate School

Florida International University, 2013

© Copyright 2013 by Daniel Jose Lepervanche

All rights reserved

## DEDICATION

I would like to dedicate this to my son, Lucas. May this accomplishment testify that dreams do come true. You must love what you do, work hard and smart, and have integrity. Also I wish to dedicate this to Leonor, Gabriela, Jose, and Flor. You are my pillars, my motivators, my dream-team, my inspiration, and my unconditional support group. Thank you for believing in me, helping me achieve my dreams, and letting me fly higher. One Love!

## ACKNOWLEDGMENTS

First and foremost I wish to thank God for all of the blessings in my life, for always putting me in the right place and the right time, and for always guiding, protecting and providing. I wish to thank my parents, Leonor, Jose and Flor for bringing me into this world, raising me and shaping me into the man I am today. I wish to thank my wife and son, Gabriela and Lucas, for inspiring, motivating and pushing me to be better everyday, you are the fuel that drives me. I would also like to thank my siblings: Adriana, Arturo, Alejandro, Carolina, Cesar, Gustavo, and Eduardo, for always being there. I wish to deeply thank Paula Matthusen, Jacob Sudol, Orlando Garcia, Jacek Kolasinski, Joel Galand, Adam Drisin, Brian Schriener, Kristine Burns, and Juraj Kojis for playing key roles in the development of this project, as mentors, committee members, advisors, faculty, and friends. I would also like to thank all of the programmers who allowed me to use their codes and contributed tremendously to this project: Peter de Tagyos, Yohann Taieb, Frank Hernandez, and Christopher Rivera. I also thank Preeya Disyanan of The Golden Triangle for allowing to come into her store to record her Tibetan instruments. Also thanks to Luis Arturo Mora for allowing me to use his photograph of the FLEA ensemble. Finally, I wish to thank Meg Galvis from the FIU's School of Music's main office for the endless support and friendship.

ABSTRACT OF THE THESIS

iGROOVING:

A GENERATIVE MUSIC MOBILE APPLICATION FOR RUNNERS

by

Daniel Jose Lepervanche

Florida International University, 2013

Miami, Florida

Professor Jacob Sudol, Major Professor

*iGrooving* is a generative music mobile application specifically designed for runners. The application's foundation is a step-counter that is programmed using the iPhone's built-in accelerometer. The runner's steps generate the tempo of the performance by mapping each step to trigger a kick-drum sound file. Additionally, different sound files are triggered at specific step counts to generate the musical performance, allowing the runner a level of compositional autonomy. The sonic elements are chosen to promote a meditative aspect of running. *iGrooving* is conceived as a biofeedback-stimulated musical instrument and an environment for creating generative music processes with everyday technologies, inspiring us to rethink our everyday notions of musical performance as a shared experience. Isolation, dynamic changes, and music generation are detailed to show how *iGrooving* facilitates novel methods for music composition, performance and audience participation.

## TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION.....	1
Inspirations.....	1
Purpose.....	4
II. HISTORICAL AND RESEARCH BACKGROUND.....	10
Biofeedback in Music.....	10
Algorithmic Composition and Generative Music .....	15
III. MUSICAL AESTHETIC AND STRUCTURE.....	19
IV. APPLICATION DEVELOPMENT.....	24
Programming in iOS.....	24
<i>iGrooving</i> 's Code.....	29
V. CONCLUSION.....	34
BIBLIOGRAPHY.....	36
APPENDICES - Objective-C code for <i>iGrooving</i> .....	40



## LIST OF FIGURES

FIGURE	PAGE
1. The FIU Laptop & Electronic Arts (FLEA) Ensemble performing during Art Basel 2010 (image by Luis Arturo Mora).....	5
2. The FIU Laptop & Electronic Arts (FLEA) Ensemble performing <i>L.gamelan.D</i> at the SEAMUS 2011.....	8
3. Performance of <i>bright planes from euphonious hammers: the story of a sound installation that became a composition</i> at FIU Spring 2010.....	9
4. Nayla Mehdi and Jaclyn Heyen (Not in the picture) performing <i>YouGrooving</i> (Heartbeat version) 2009.....	11
5. Different ranges of Tibetan singing bowls (left) and a set on <i>Tingshas</i> (right).....	20
6. Tibetan bell and <i>dorje</i> .....	20
7. <i>iGrooving</i> 's prototype in Pro Tools.....	23
8. Computer screenshot of <i>iGrooving</i> and the simulator.....	25
9. <i>iGrooving</i> 's code: Accelerometer, step count, automatic start and kick drum trigger (Appendix B).....	30
10. <i>iGrooving</i> 's code: Sound file triggers and behavior (Appendix B).....	31
11. iPhone's screenshot of <i>iGrooving</i> .....	32

# I. INTRODUCTION

## Inspirations

As a composer and technologist, I am compelled to explore new media and aesthetics in music composition and performance in order to refine my compositional style. *iGrooving* is a generative music mobile application for runners that primarily stems from my interest in the concepts of biofeedback and generative music.

Biofeedback uses physiological processes to manipulate an external medium; for example, a human's brainwaves might be used to control a wheelchair's movement. The concept of generative music embraces self-organizing musical compositions with set parameters that may or may not receive additional input from a performer or user.

In earlier research, I explored biofeedback as a means of performance and composition by treating the heartbeat as an instrumentalist or controller and the audience as a composition's sonic material (Lepervanche 2009). I revisited commonly held notions regarding the essence of music, specifically focusing on how our physicality relates to music, how performers interpret it, and how our human bodies are—in a sense—living musical compositions. This research led to my personal philosophical discovery that, without the performer and audience as separate entities, a composition ceases to be a shared experience in the performance context. *iGrooving* goes further, collapsing not only the performer and audience but also the composer into one.

My use of biofeedback owes much to Alvin Lucier's *Music for Solo Performer*, a pioneering work that uses brainwaves to generate sound. In this work, the performer's

brainwaves “are amplified enormously and routed through loudspeakers to vibrate sympathetically a battery of percussion instruments” (Lucier & Margolin, 1982). The sounds arise when the performer produces a specific pattern of brainwaves, namely alpha waves, which occur when one enters a non-visualizing or deeply relaxed state. In *iGrooving* I also explore the correlation of a relaxed state—more specifically, a sort of meditative state induced by running—and generative sound production; however in my work, instead of brainwaves, loudspeakers, and percussion, I use a runner’s steps trigger prerecorded sounds.

Another influence on my work is David Rosenboom’s *On Being Invisible II (Hypatia Speaks to Jefferson in a Dream)*, a self-organizing opera that incorporates brain waves. “The sequencing of this work’s nonlinear narrative content...and the structure of accompanying electronic sounds all result from a similar analysis of the brainwaves of two performers. Consequently, the form of the opera cannot be known in advance” (Rosenboom, 2003). For *iGrooving*, I rely on the runner’s steps to attain a steady rhythm and on the manner in which one’s steps trigger sound files to generate the musical performance and composition.

Also informing my work is Eduardo Reck Miranda’s research on brainwaves and musical systems, which explores how the thoughts of the user have the potential to control a musical system. Reck’s Brain Computer Interfacing (BCI) systems are primarily designed to allow the brainwaves to generate musical content. Describing the biofeedback processes in this work, Reck notes that “[the] spectral information [of the brainwaves] is used to activate generative music rules to compose music on the fly, and the signal complexity is used to control the tempo of the music” (Miranda & Brouse,

2005). Similarly, in *iGrooving* a runner controls the tempo and the time at which the different sound files are triggered via his or her steps. The final biofeedback work that informs my thesis is Pauline Oliveros' *Adaptive Use Musical Instruments (AUMI)*, a "software interface [that] enables students who have very limited controlled (voluntary) movement or other types of impairments to independently engage in making music" (Oliveros, 2012). Likewise, *iGrooving* allows everyone capable of running—regardless of whether one has musical training or not—to engage in the music-making experience. Mobile phone applications that influenced my graduate research include *Bbcut*, *Con:cat* and *iGendyn* by composer and programmer Nick Collins (Collins, 2013). The versatility, programming creativity, and intuitive design displayed in these applications inspired me to find a way to use the iPhone's programmable features, such as the accelerometer, to generate music. I drew more, however, from the mobile phone applications *Bloom* and *Scape* by the composer and sound-artist Brian Eno and the musician and software designer Peter Chilvers (Eno, 2012). *Bloom*'s user-friendliness and intuitive design captivated me as I used it to soothe my newborn child. Moreover Eno and Chilvers' approach to a generative music that features minimalist tendencies and drone structures inspired the overall musical aesthetic in *iGrooving*.

Finally, I have been inspired by Brian Eno's descriptions of generative music, which Nick Collins defines as "a more rigid description of algorithmic music that happens to produce [an audio] output in real time" (Collins, 2008). To expand on this definition, generative music is a system that uses mathematical or logical structures, with or without external inputs, to create ever-changing music.

## Purpose

"[We] only have three parts for people involved in live performances. Henry Flynt, one of [La Monte] Young's Fluxus cohorts, identifies the division of labour as:

- a) The author of the score, the composer.
- b) Those who realize the score, the performers.
- c) Those who witness the performance—usually from massed seating.

As Flynt pointed out, experimental compositions challenge such distinctions" (Sun, 2006).

I wanted to create an application that primarily questions the roles of the composer, performer, audience, and their relationships as a shared experience in live musical performance. In *iGrooving* a runner becomes the co-composer, performer, and audience in an isolated musical event. As a result, the runners react to his or her performance as an audience member. This allows the runner to make some compositional decisions that will affect the performance and overall sonic outcome. Because this feedback process occurs in an isolated situation, it challenges common notions of composer, performer, and audience's roles. *iGrooving* stimulates self-awareness; this type of musical performance becomes a shared experience with the self, challenging my previous view that the performer and audience needed to be separate entities in order for a composition to exist as a shared experience.

With *iGrooving*, I also propose an alternate way of understanding and defining musical instruments. This interest comes in part from my experiences as a founding member of the Florida International University Laptop & Electronic Arts (FLEA) ensemble, as well as from my research on other experimental ensembles such as the

Princeton Laptop Orchestra (PLOrk), the Stanford Laptop Orchestra (SLOrk), and the Stanford Mobile Phone Orchestra (MoPhO).



Figure 1. The FIU Laptop & Electronic Arts (FLEA) Ensemble performing during Art Basel 2010 (image by Luis Arturo Mora)

These ensembles, by incorporating non-acoustic instruments and controllers such as laptops, mobile devices, and custom-made electronic devices, challenge the traditional conception of a musical instrument as something that uses acoustic means to produce sound, replacing it with something much broader or more nebulous. In spite of this radical position, these ensembles perform music that can be just as formal and organized as that of traditional ensembles and that ranges from improvised to precisely notated music (Scheinin, 2008). These new types of unconventional orchestras are still maturing; their creators “hope to establish some continuity with [their] newly developing performance practice so that [they] do not have to 'reinvent the wheel' each year and so

[they] can reach higher levels of skill and familiarity” (Smallwood, Trueman, Cook, & Wang, 2008).

Some new electro-acoustic music attempts—in ways that I find compelling and suggestive for my own work—to refine the use of laptop computers or electronic devices so that they might gain broader legitimacy, recognition, and use. For example, PLOrk's members have discussed issues arising from the formalization of laptop orchestras, such as how "composers working on pieces for a laptop orchestra have the choice to embrace the given laptop interface as an instrument, or to find ways of providing more suitable control mechanisms for making sound, depending on the kind of instrument they are designing" (Smallwood, Trueman, Cook, & Wang, 2008). Interestingly, ensembles such as PLOrk, SLOrk, and MoPhO have addressed this issue by progressively designing and incorporating gestural devices more often than the laptop interface.

Thus , PLOrk has “a collection of interfacing devices and sensors that can be integrated into any of the meta-instruments to provide physical control of expression. These include off-the-shelf keyboards, percussion pads, and knob/slider controllers, but also custom interfaces using sensors such as accelerometers, pressure pads (using force-sensing-resistors), proximity sensors, light sensors, and so on” (Smallwood, Trueman, Cook, & Wang, 2008). More recently there have even been groups like MoPhO that use mobile devices such as the iPhone as their only musical instrument (Wang, Essl, & Penttinen, 2008).

The accessibility, intuitive design, and programmable features of mobile devices make the iPhone an excellent choice for this thesis project. For groups like MoPho, “Apple’s iPhone has been an enabling technology to more fully consider mobile phones as meta-

instruments for gesture driven music performance.” (Wang, Essl, & Penttinen, 2008).

Indeed, mobile devices have become so prominent and integrated with our everyday lives that they can be easily ignored as music-making entities and become a quasi-natural extension of the human body.

My works mirror this progressive implementation, first of gestural devices and then of mobile devices. Four of my works, *L.gamelan.D*, *bright planes from euphonious hammers: the story of a sound installation that became a composition*, *YouGrooving*, and *iGrooving*, demonstrate this personal compositional progression. In these works I have often aimed to inform the audience of the relationship between the performers, technology, and the music performed.

For example, my composition for laptop ensemble, *L.gamelan.D*, demonstrates some of the performer’s active role in sound production. At the beginning of the work the performers tap piezo-electric contact microphones that they place inside their pants’ pockets or on the floor. The performers use this tapping to make sounds that are first heard in real time and, later, make up the sampled material for the rest of the performance. By solely featuring this tapping on contact microphones at the beginning of the work, I provide a clear visual element that informs the audience about the performer’s active role and how it relates to the sound world. Figure 2 shows a performance where the FLEA ensemble uses this visual reference.





Figure 2. The FIU Laptop & Electronic Arts (FLEA) Ensemble performing *L.gamelan.D* at the SEAMUS 2011 Conference

My interactive audio-visual composition shown in Figure 3, *bright planes from euphonious hammers: the story of a sound installation that became a composition*, explores an interactive relationship between the performer, computer, lights, and color. In this composition, a score instructs any number of instrumentalists and vocalists to play any pitched sound while collectively searching for an agreeable groove. After playing this groove for a few minutes, they must transition to a new groove and repeat this transition and grooving process as often as desired until they agree to finish the composition at an appropriate time. Specific musical pitches are mapped on to different lights, which turn on only when the corresponding pitch sounds. Figure 3 shows colored lights that also correspond to real-time audio digital signal processes, such as reverb and delay, that an autonomous laptop applies to the overall sound world.



Figure 3. Performance of *bright planes from euphonious hammers: the story of a sound installation that became a composition* at FIU, Spring 2010

The relationship and interaction of music, lights, and effects intends to inform the audience of the relationship between the instrumentalists and vocalists, and the audio-visual world. Moreover, by automating how the laptop and technology functions, I focus the audience away from the technological means for creating the relationship and, instead, towards the relationship itself.

In contrast to my previous works, and most other works by other artists that incorporate biofeedback and generative processes, *iGrooving* requires the performer or user—in this case, a runner—to interact with the mobile application alone. As a result, the musical performance becomes an isolated event that only the user experiences. By creating this situation, I explore the following questions that relate to the use of the application: What is a musical instrument? Does this application shatter the notion of musical performance as a shared experience? Can the roles of composer, performer, and audience be successfully combined? And, finally, can the use of biofeedback and generative music in *iGrooving* be a positive addition to the running experience?

## II. HISTORICAL AND RESEARCH BACKGROUND

### Biofeedback in Music

Biofeedback uses “electronic monitoring of a normally automatic bodily function in order to train someone to acquire voluntary control of that function” (“Biofeedback,” 2013). When used in music composition, tracking biofeedback signals offers a new realm of possibilities for music composition. These signals can be tracked by the use of different types of interfaces and sensors.

The use of biofeedback in music has three main components: the physiological body function, the interface or sensor used to translate this function into sound or music, and the feedback loop that attempts to train the user to control said function and relate it to the resultant sound or music. For example, the electrical activity of firing neurons in the brain, also known as brainwaves, can be a physiological body function. One could then use an electroencephalography (EEG) sensor to translate this activity into music. Finally, the feedback loop is the performer’s attempt to control his brainwaves in light of their sonic translation. Beyond this, it is important to note that use of biofeedback in music is in direct contrast to conventional types of performance practices where performers use parts of the body over which they can have much greater conscious control, such as their arms, fingers, and lungs.

My composition *YouGrooving* exemplifies a biofeedback work. Here the physiological bodily function is the heartbeat, digitally captured by a stethoscope connected to a lavalier condenser microphone. In *YouGrooving* the heartbeats have two

purposes—they are the sampled sonic content for the live performance, and they trigger multiple sound files. A separate performer controls the second purpose—the translation component of the biofeedback process—by processing the heartbeats' sound through a resonance filter to find the average loudest amplitude of each heartbeat's "thump" and to generate an amplitude threshold. In performance this separate performer fine-tunes the threshold so that every time the heartbeat "thumps," one of the multiple sound files sounds. The different heart rates, as well as the incredibly difficult task of consciously controlling one's own heart rate, provide subtle elements of unpredictability, which affect the composition. For *YouGrooving* I prefer a larger performance group, yielding a thicker polyrhythmic texture and subtle tempo changes. Looking beyond my thesis, I would like to similarly develop *iGrooving* for an ensemble.

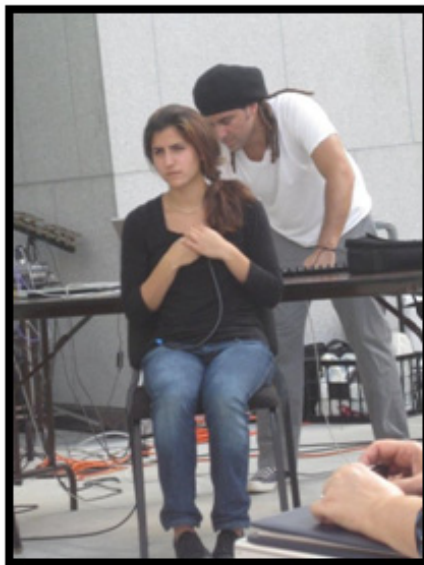


Figure 4. Nayla Mehdi and Jaclyn Heyen (not in the picture) performing *YouGrooving* (Heartbeat version) 2009

Corresponding to the heartbeat in *YouGrooving*, the runner's steps in *iGrooving* determine when various audio samples sound. This brings elements of randomness and

unpredictability to *iGrooving*'s performance and composition. For example, the runner might encounter several unforeseen and unpredictable factors such as difficult terrain, obstacles, exhaustion, and bad weather.

To explore biofeedback further as a means of musical expression and as a performance practice, I will discuss two innovative works—Alvin Lucier's *Music for Solo Performer* and the *AUMI* interface by the Deep Listening Institute led by Pauline Oliveros. In order to explain the choices I made for *iGrooving*, I focus on two issues involved in using biofeedback to create and perform music, namely, the control of biological mechanisms and the interface used to track these functions.

In a video-recorded excerpt of a 1977 performance of *Music for Solo Performer* (1965) (Osterreich, 1977), Lucier begins the performance by entering a calm state while the assistant attaches EEG electrodes to his forehead and a ground electrode to his left hand. Lucier then reaches over the signal amplifier with his left hand to gauge a desired level at which the brain signals will be amplified so that the percussive instruments will sympathetically vibrate with the speakers. Lucier first demonstrates to the audience how the performance works by closing his eyes. This action stimulates the fluctuation of his brain signals, which are then amplified through loudspeakers that cause percussive instruments to sympathetically vibrate. Lucier repeats this informative gesture a couple of times. His brainwaves then seem challenging to control as he makes several attempts to close his eyes with his right hand in order to achieve the necessary non-visualizing state. About six minutes into the excerpt, Lucier finally creates some steady brainwaves

and, as a result, different percussive instruments shake and rattle to create a rich polyrhythmic texture.

In this work, the interface used to translate the brain waves into sound poses a number of challenges. The intricate use of EEG electrodes, a multichannel amplifier, multiple loudspeakers, and a complex percussion set-up make this interface time-consuming to construct and the composition difficult to realize. Beyond this, as the performance discussed above demonstrates, it can be hard to produce the correct brain waves during a performance. This is potentially frustrating for the performer and can lead to long periods of no sonic activity.

In contrast, *AUMI* by Pauline Oliveros is a user-friendly interface that nearly anyone can use. The interface has primarily been used for children with physical disabilities to create a space for them to be part of the music-making process.

The Adaptive Use Musical Instruments software interface enables the user to play sounds and musical phrases through movement and gestures [and] attempts to make musical improvisation and collaboration accessible to the widest possible range of individuals. This approach also opens up the possibility of learning more about the relations between ability, the body, creativity and improvisation (Oliveros, 2013).

The *AUMI* is designed with computers that have video capabilities in mind; therefore, the programmers write the computer code to create video tracking as the initial point of communication with the music-making component of the software and computer. With this video tracking, users of *AUMI* can choose any part of their body, such as their nose, as the tracking point and then use their facial movements to drag the tracking point across the screen and thereby trigger different sound files. In this way, the movement of specific components on one's own body becomes the physiological bodily function in

*AUMI*, while the video tracking and software are the interfaces that translate this function into music.

Because *AUMI* has primarily been used for children with physical disabilities, it is in constant development to adapt to the various users' abilities. Its next steps include developing an iPad application and incorporating Musical Instrument Digital Interface (MIDI). MIDI is protocol that allows multiple electronic instruments, computers and musical devices to communicate with each other. The use of MIDI would allow the *AUMI* users to compose music with the aid of other music production software.

Unlike *Music for Solo Performer*, *AUMI* features an easily accessible and intuitive interface that uses a biological mechanism one can instantaneously control voluntarily. Moving one's nose over a split-screen is obviously more straightforward than controlling one's brain waves. Similarly, *iGrooving* incorporates a voluntarily controllable biological mechanism—the runner's step rate. By exercising control over their pacing, runners can make decisions on how their biological mechanisms affect the feedback loop component in a biofeedback composition. To me, an interesting factor in *iGrooving*, is that—although we have considerable voluntary control of our steps—we don't usually train them to produce musical results. Since the EEGs, EKGs and EMGs sensors used in some of the compositions explained above can be intricate in nature, cumbersome and costly, I decided to choose the widely available iPhone as the translating interface for *iGrooving*. Owing to the intuitive nature of mobile devices, using the iPhone as a biofeedback interface could potentially be as natural as making a phone call.

## Algorithmic Compositions and Generative Music

At their most basic form, algorithms are step procedures to solve a problem or to calculate a solution. Their use in music composition is traceable at least as far back as the eleventh century:

The idea of generating music algorithmically is not new. The earliest recorded work was by the Italian monk Guido D'Arezzo in 1026. Demand for his Gregorian chants was so high that he devised a system to systematically create them from liturgical texts. Mozart, Haydn, and C.P.E. Bach had an interest in generative music; Mozart invented [a] *Musikalisches Würfelspiel* ("musical dice game"), which involved using dice to decide which of a set of pre-defined musical phrases came next in the piece (Worth & Stepney 2005).

Nevertheless, algorithms became far more prominent with the development of computers during the mid-twentieth century. Composers and music researchers of the computer era started using these tools to search for new musical aesthetics and to react to or model previous styles in the evolution of music. Computers provided new sonic materials and tools that allowed composers to explore and formalize new sounds and compositional approaches (Roads, 1996). For example, MUSICOMP, started by Lejaren Hiller and Leonard Isaacson, was a pioneering algorithmic composition software that opened the way for many composers and programmers (Wooller, Brown, Miranda, Berry, & Diederich, 2005). In the 1950s, Hiller and Isaacson used this program to realize a set of Markov Chains to construct works such as the *Illiad Suite* (1956) that primarily imitate earlier classical musical idioms, such as tonal four-part choral writing. Meanwhile, composer Iannis Xenakis' "particular interest [was] in creating complex musical textures [and] us[ing] stochastic functions to organize the general characteristics of these large-scale entities" (Harley, 1995). In contrast to the more formal and musicological



experiments of Hiller and Isaacson, Xenakis' computer-generated algorithmic compositions such as *ST/4* and *ST/10* (1956–62) made greater use of indeterminacy, giving a perceptual priority to chance operations.

In essence, one can describe algorithmic music compositions as works that use a computer or other mathematical means “with the aid from various formalisms, such as random number generation, rule-based systems, and various other algorithms” (Alpen, 1995). Not all algorithmic compositions use random processes. For example, in his algorithmic-generated compositions, Tom Johnson “allows his music to be completely deterministic and predictable, a product of little mathematical machines” (Johnson, 1998).

Like Tom Johnson's works, *iGrooving* solely uses deterministic algorithm procedures. The program maps the runner's steps to a kick drum sample. In addition, the steps trigger twelve different sound files at predetermined step counts. All sounds besides the kick drum, once instantiated, loop until the performance concludes. The runner has complete autonomy to start the performance, establish the tempi used, and trigger the remaining sound files. In contrast to Tom Johnson's deterministic algorithmic scores, however, this autonomy can result in drastically different realizations. For example, if the runner runs below a step count of 700 steps, only three sound files out of twelve are triggered. Furthermore, the runner has the autonomy to establish the length of the performance. Since the probability of running at an exactly perfect cadence more than once is so low, the likelihood that a specific compositional realization will happen only once directs the focus towards the present moment.

Historically, generative music succeeds algorithmic compositions. Generative music uses mathematical or logical structures, with or without external inputs, to create an ever-changing sonic result. Furthermore, “an algorithm tends toward being generative when the resulting data representation has more general musical predisposition than the input” (Wooller et al. 2005). Generative music directly stems from algorithmic compositions, yet in generative music there are always different sonic outcomes, whereas in algorithmic composition the outcomes can be potentially fixed. This is the main difference that separates generative music from what I refer to in this essay as algorithmic compositions.

I consciously try to be less directly involved in predetermining all aspects of the musical outcome. *L.gamelan.D*, *bright planes from euphonious hammers: the story of a sound installation that became a composition*, and *iGrooving* provide clear examples of this. As a composer, I bask in this relinquishment of control. As Brian Eno states, “if you move away from the idea of the composer as someone who creates a complete image and then steps back from it...[Composition becomes] putting in motion [a performance] and letting it make the [composition] for you” (Eno, 1996).

*iGrooving* only works if a user is inputting data. For example, until the user starts running and the first step gets counted, *iGrooving* will not produce sound. In contrast, *Bloom*, by Brian Eno and Peter Chilvers, can generate music without receiving any input from a user. The intuitive design and passive nature of *Bloom* allow the user to become engulfed by the sound world. Almost game-like and simple, *Bloom* effectively allows anyone to feel like part of the music-making experience. The interface of *Bloom*

resembles a sideways keyboard that plays pitched decaying sounds whenever the user touches the screen. Its design is intuitive—the higher on the screen one touches the higher the pitch and *vice-versa*. The application then slowly repeats the patterns that the user inputs. *Bloom* is also capable of being a stand-alone music box and, as such, can play material by itself.

On the other hand, *iGendyn*, by Nick Collins, is a noisy synthesizer that uses parameters such as the multi-touch capabilities of the iPhone's screen and the accelerometer. The user has the option of selecting multiple voices with five independent touches. In addition, when the user tilts the phone, *iGendyn* makes probabilistic and deterministic decisions or changes that affect the resultant sonic content. Like *iGendyn*, *iGrooving* uses the accelerometer. However, in contrast to the configurability of *iGendyn*, *iGrooving* uses the accelerometer solely as a step detector. I discuss this feature in Chapter Four.

### III. MUSICAL AESTHETIC AND STRUCTURE

The sound world of *iGrooving* combines Electronic Dance Music (EDM) with subtle, sonic, meditative elements that slowly develop and repeat throughout the running performance. EDM is typically repetitive and accumulative in nature, structured in multiple layers that first progressively build upon each other and then get added or subtracted to create a sense of movement and development. Beyond these structural features, EDM makes extensive use of effects such as filter sweeps and multiple delays. Artists such as Fat Boy Slim, Prodigy, Daft Punk, Rabbit in the Moon, Paul Oakenfold, Mauro Picotto, and, most recently, Deadmaus5 and David Guetta have played an influential role in the shaping of *iGrooving*'s musical structure.

The primary sonic element that I drew from popular music like EDM is the sound of the electronic kick drum. The other sonic elements I use include recordings of Tibetan singing bowls, various *tingshas*, and a set of Tibetan bell and *dorje*. Tibetan singing bowls are classified as standing bells. They can be played by rubbing the ring in a circular motion with a wooden striker or by softly hitting the rim with the striker. *Tingshas* are two small-pitched cymbals joined together by a strap or chain and usually struck together to produce a high pure metallic tone similar to crotales. Figure 5 shows five different sized Tibetan singing bowls and a set of *tingshas*. Figure 6 shows a Tibetan bell and *dorje*.

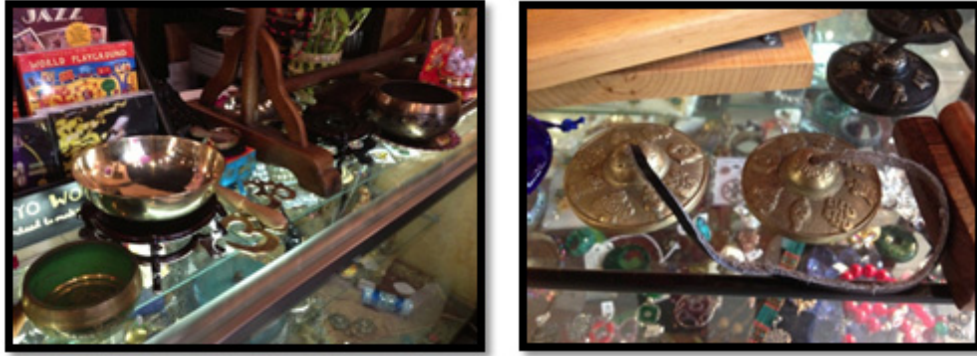


Figure 5. Different ranges of Tibetan singing bowls (left) and a set on *tingshas* (right)



Figure 6. Tibetan bell and *dorje*

I recorded myself rubbing the edge and hitting the side of three different Tibetan singing bowls with the striker; playing single hits and multiple consecutive hits of the *tingshas*; and playing multiple consecutive hits of the Tibetan bell and *dorje* struck together. I then catalogued these sound files in three sound banks: BOWL, HIT and MHIT. Table 1 shows this cataloguing scheme. The BOWL sound files are the Tibetan singing bowls played by rubbing the ring in a circular motion with the wooden striker; the HIT sound files include *tingshas* struck against each other and Tibetan singing bowls struck with the wooden striker; and the MHIT sound files include multiple hits of Tibetan bell and *dorje* struck together, multiple hits of *tingshas* struck together, and multiple hits

of Tibetan singing bowls struck with the wooden striker.

BOWL 1	TIBETAN SINGING BOWL
BOWL 2	TIBETAN SINGING BOWL
BOWL 3	TIBETAN SINGING BOWL
BOWL 4	TIBETAN SINGING BOWL
HIT 1	TIBETAN SINGING BOWL HIT
HIT 2	TINGSHA HIT
HIT 3	TIBETAN SINGING BOWL HIT
HIT 4	TINGSHA HIT
MHIT 1	TIBETAN BELL AND DOJRE MULTIPLE HIT
MHIT 2	TINGSHA MUTLIPL E HIT
MHIT 3	TIBETAN SINGING BOWL MULTIPLE HIT

Table 1. Sound files catalogue for *iGrooving*'s sonic content

For *iGrooving*, the intention is that the runners listen to these sonic materials passively as a subtle supplement to running. I also want the runner to focus on the drum's rhythms, the drone-like sounds of the Tibetan singing bowls, and—ultimately—the overall minimalist sonic texture.

For the sonic content I used a minimalist approach, favoring repetition, drones, and slow development. I chose the Tibetan instruments because of their timbre, long sustain, and drone-like and ritualistic implications. I was inspired by minimalist composers such as La Monte Young and Steve Reich—among many others—in their use of drones and

ritualistic repetitions of simple rhythmic patterns. For example, pieces by Steve Reich that preceded and led up to *Music for 18 Musicians*, such as *Music for Mallet Instruments, Voices and Organ* (Reich, 1987), included “mixing timbres, and mixing very long-held tones” (Kim, 1999) by beautifully blending heavy rhythms with drones. More relevant models that relate to *iGrooving*’s musical structure and aesthetic include Steve Reich’s *Drumming* (Reich, 1971) and Philip Glass’s *Music with Changing Parts* (Glass, 1994). For example, Reich’s *Drumming* develops without:

Unfolding melodies or evolving motivic processes in the context of the sort of contrapuntally harmonic textures central to the common-practice tradition. It stays closer to its roots than an extended-common-practice work of the same duration would do, and generates a special kind of uninhibitedly hypnotic ecstasy...by keeping those common-practice concepts of variety at bay. The trick is to create the effect of 'considerable developments' in ways that would not have been thought considerable, or even developmental, in the past—at least not when spread over '55 to 75 minutes' (Whittall, 2012).

Philip Glass’s structures based on additive and subtractive rhythmic patterns also influenced my research. In particular, I studied *Music with Changing Parts*, which consists of “variations in timbre...sustained tones threading through the typical busy texture of repeated patterns” (Bernard, 2003). Finally, La Monte Young’s and Terry Riley’s studies in Indian music and Steve Reich’s studies in African music inspired my choice to favor a drone-like sound world that demonstrates the influence of non-Western music.

To test and refine the sound world of *iGrooving*, I created the sonic prototype of the resultant sound shown on Figure 7, using the digital audio workstation Pro Tools. Pro Tools is a digital audio workstation that gives the user multiple functionalities, such as

recording, music production and audio mixing. I used this prototype as a simulator to adjust the potential musical outcomes so that I would not have to run every time I wanted to test the application's sound world.

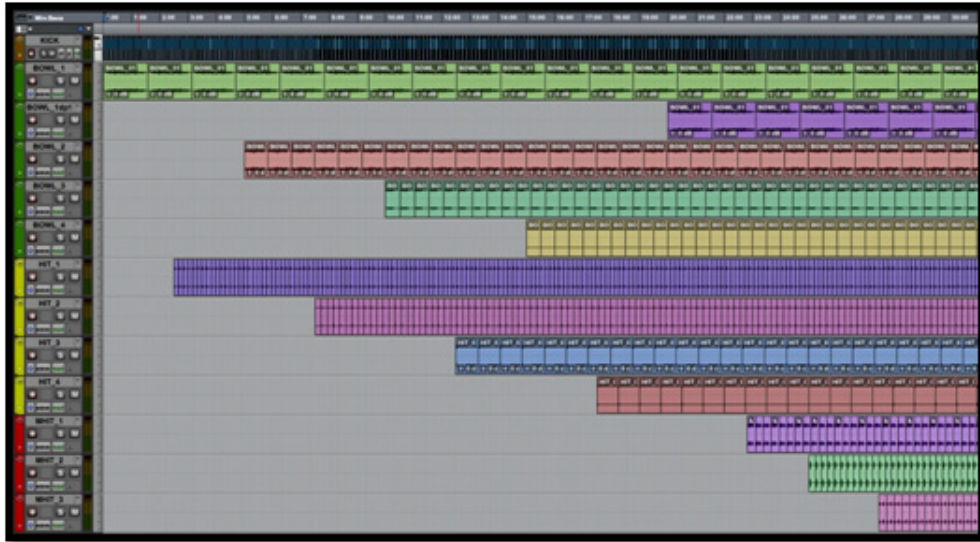


Figure 7. *iGrooving's* prototype in Pro Tools

As the prototype in Figure 7 demonstrates, in a manner similar to many EDM works, *iGrooving* is accumulative and repetitive in structure. Structurally, the trance-like sonic world and steady beat of EDM aim progressively to complement and overshadow the constant repetitive action running offers. Beyond this, my work differs from EDM in that *iGrooving* does not rely on the near-perfect accuracy of a computer's clock or synthesized sounds but, instead, on the more organic rhythm of a runner as well as sampled acoustic sounds. Aesthetically, I aim for the drone-like, meditative, and ritualistic qualities of the sonic material to encourage the listener to focus on these potentially ignored introspective aspects. With these materials and structure, it is my goal that this musical quality in *iGrooving* sheds light on a corresponding meditative quality in running.



## IV. APPLICATION DEVELOPMENT

### Programming in iOS

I developed *iGrooving* for the iPhone 4S. The programming platform for Apple's mobile devices is iOS and the programming language for iOS is Objective-C (Apple Inc., 2013). Objective-C is derived from the standard computer programming language C. Objective-C caters to mobile devices by allowing programmers access to additional programmable elements. Some of these elements—called sensors—include the accelerometer, gyroscope, proximity sensor, touch screen, and GPS. The accelerometer “measures the linear acceleration of the device so that it can report its roll and pitch, but not its yaw. Yaw, pitch and roll refer to the rotation of the device in three axes: X, Y, and Z” (Allan, 2012). The gyroscope “is a rate-of-change device; as the phone rotates around an axis, [it] allows [one] to measure the change in such rotations” (Allan, 2012). The proximity sensor is “used by the device to turn the screen off when [one] puts the phone to [one's] ear to make a call” (Allan, 2012). Because of these programmable elements, it was more advantageous to program *iGrooving* for the iPhone than a desktop or laptop computer.

Apple provides a simulator whereby one can test certain parameters on the iPhone; however, the accelerometer is a parameter that can only be tested on a mobile device. In *iGrooving*, I use the accelerometer to create a step counter. A step counter is a device that tracks when one takes a step and then counts how many steps one has taken since the

first step. As explained above, *iGrooving* maps the step counter to the runner's stride and the playback of prerecorded sound files. Using the accelerometer to calculate a step counter allowed me to start thinking about the relationship of steps counted versus time elapsed. The direct translation of beats-per-minute (BPM) to steps-per-minute allowed me to think musically when designing which step counts would trigger each sound file.

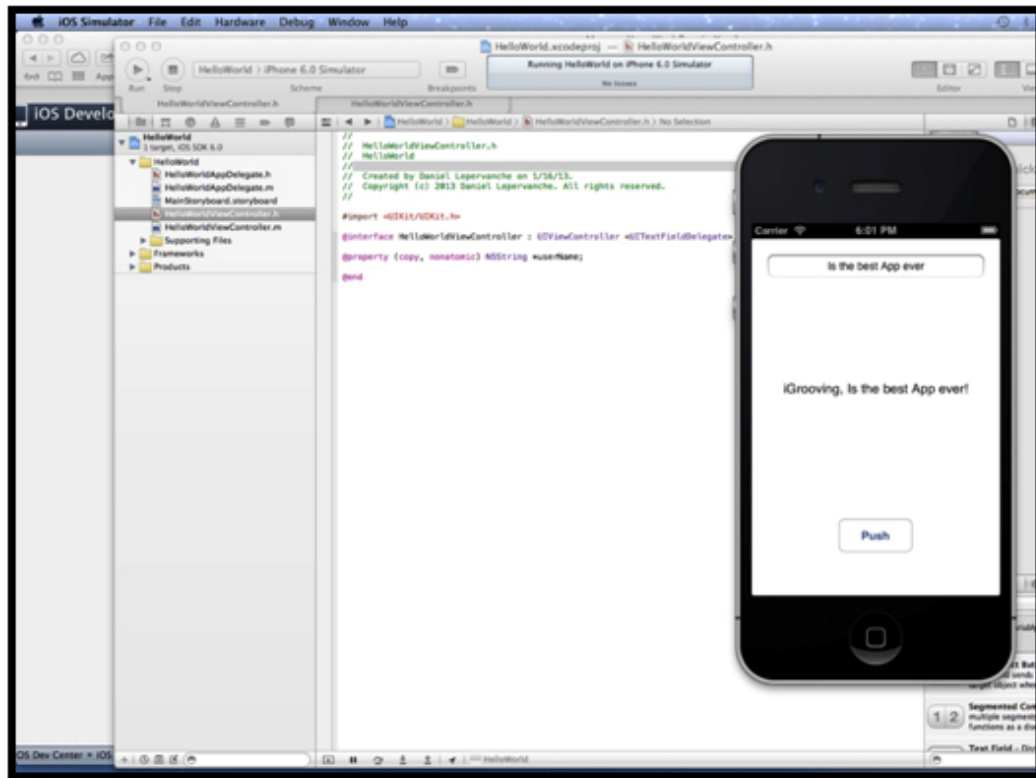


Figure 8. Computer screenshot of *iGrooving* and the simulator

Figure 8 shows the beginning stages of *iGrooving*'s programming. The step counter's source code provides the foundation of *iGrooving*. In order to calculate a step counter, one must first devise a way to detect steps. The step detector uses the accelerometer to report if a movement surpasses a predefined threshold change in movement. When a value surpasses the preset threshold, it reports a step or instance,

which is then sent to a step counter. The musical elements are built around the step detector and the steps counted. When the step detector creates an instance, the sound of the kick drum is triggered. Specific step counts also trigger sound files that loop until the performance concludes. Additionally, users have the option to reset all parameters and start the performance over.

*iGrooving* has three sections—a warm-up, a steady run, and a cool-down.

Following this structure and my experience with running I chose to trigger sound files at intervals of 325 and 375 steps. A common running warm-up section at 130 beats-per-minute should have a time interval of two and half minutes every 325 steps; a steady run section at 150 beats-per-minute should have a time interval of two and half minutes every 375 steps; and a cool-down section at 130 beats-per-minute should have a time interval of two and half minutes every 325 steps. Table 2 shows this timing scheme. Triggering sound files at approximately every two and half minutes felt like a correct pace when I was prototyping and testing *iGrooving*. Furthermore, I decided for aesthetic reasons that the longer the runner runs the more active the performance should become. This creates a progressively richer polyrhythmic experience that could potentially motivate the runner to extend the performance. Currently, *iGrooving* has a limit of 4200 steps. I wish to develop this further in successive versions.

SECTION	STEP COUNT	SOUND FILE
A	1	BOWL_1
A	325	HIT_1
A	650	BOWL_2
B	975	HIT_2
B	1350	BOWL_3
B	1725	HIT_3
B	2100	BOWL_4
B	2475	HIT_4
B	2850	BOWL_1
C	3225	MHIT_1
C	3550	MHIT_2
C	3875	MHIT_3

Table 2. *iGrooving*'s sections and the step count-sound file relationship

The following text defines, in a step sequence, *iGrooving*'s basic algorithm:

### **ALGORITHM**

1. The USER opens *iGrooving* and places the phone on the belt-band, puts the headphones on, and makes sure the headphones are connected to the audio jack in the iPhone.
2. The performance begins when the first step (STEP 1) is taken.
3. The TIMER starts when the performance begins.
4. A label displays the TIMER; another label displays the STEP COUNT.
5. Each STEP precisely triggers the KICK sound file.
6. Audio files get triggered, start looping, and overlap:
  - Section A: sound files get triggered every 325 STEPS
  - Section B: sound files get triggered every 375 STEPS
  - Section C: sound files get triggered every 325 STEPS
7. The USER has the option to START, STOP, and RESET the TIMER.
8. The USER has the option to RESET STEP COUNTER.
9. The USER has the option to RESET ALL parameters.
10. When the USER is done, they quit the application.

## KEY

1. USER: the person using *iGrooving*.
2. TIMER (label): displays the time in hours, minutes , and seconds.
3. START (button): Start of the running exercise/performance (also automated)
4. STOP (button): Stops the TIMER.
5. RESET TIMER (button): Resets the TIMER.
6. STEP COUNT (label): Displays the runner's step count.
7. RESET STEP COUNTER: Resets the STEP COUNT.
8. RESET ALL: Resets all parameters in the application.

## ***iGrooving's Code***

*iGrooving's* foundations are a step detector and step counter. I chose these for their ease of use, simplicity, and instantaneous feedback. The step detector provides the feedback of steps, and these can be linked to time. I chose to rely solely on this method rather than creating a separate controller such as a heart rate monitor that would have been technically too complex and cumbersome for the running experience.

Figure 9 is the main section of the accelerometer's code and the features programmed around it. This code demonstrates how the accelerometer detects a step, how the steps are counted, how each instance triggers the kick drum sound file, and how

the “Steps Count” label—a component of the Graphic User Interface (GUI)—displays the steps counted.

```
// UIAccelerometerDelegate method, called when the device accelerates.
-(void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration
*)acceleration
{
    float xx = acceleration.x;
    float yy = acceleration.y;
    float zz = acceleration.z;
    float dot = (px * xx) + (py * yy) + (pz * zz);
    float a = ABS(sqrt(px * px + py * py + pz * pz));
    float b = ABS(sqrt(xx * xx + yy * yy + zz * zz));
    dot /= (a * b);
    if (dot <= 0.82)
    {
        if (!isSleeping)
        {
            isSleeping = YES;
            [self performSelector:@selector(wakeUp) withObject:nil afterDelay:0.3];
            numSteps += 1;

            if(numSteps == 1)
            {
                [self start];
            }
            self.stepCountLabel.text = [NSString stringWithFormat:@"%d",
            numSteps];

            [self CheckStepTriggers:numSteps];
            NSString *soundFilePath = [[NSBundle mainBundle]
            pathForResource:@"KICK" ofType:@"wav"];
            NSURL *soundFileURL = [NSURL URLWithString:soundFilePath];
            player = [[AVAudioPlayer alloc] initWithContentsOfURL:soundFileURL
            error:nil];

            [player play];
            NSLog(@"%d steps", numSteps);
        }
    }
    px = xx; py = yy; pz = zz;
}
```

Figure 9 *iGrooving*'s code: Accelerometer, step count, automatic start and kick drum trigger (Appendix B)

Figure 10 displays an edited section of the code that shows how specific step counts trigger sound files and the sound files' behavior. In this code, names such as BOWL\_1 and HIT\_1 correspond to the audio files in Table 2.

```

-(void)CheckStepTriggers:(int)stepCount{
    switch (stepCount){
        case 1:
            [BOWL_1 play];
            break;
        case 325:
            [HIT_1 play];
            break;
        case 650:
            [BOWL_2 play];
            break;
        default:
            break;
    }
}
-(void)InitializeSoundClips{
    NSString *BOWL_1SoundPath = [[NSBundle mainBundle] pathForResource:@"BOWL_01"
ofType:@"wav"];
    BOWL_1 =[[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:BOWL_1SoundPath] error:NULL];
    BOWL_1.delegate = self;
    BOWL_1.numberOfLoops = -1;
    BOWL_1.volume = 1;
    [BOWL_1 prepareToPlay];
    NSString *BOWL_2SoundPath = [[NSBundle mainBundle] pathForResource:@"BOWL_02"
ofType:@"wav"];
    BOWL_2 =[[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:BOWL_2SoundPath] error:NULL];
    BOWL_2.delegate = self;
    BOWL_2.numberOfLoops = -1;
    BOWL_2.volume = 1;
    [BOWL_2 prepareToPlay];
    NSString *HIT_1SoundPath = [[NSBundle mainBundle] pathForResource:@"HIT_01"
ofType:@"wav"];
    HIT_1 =[[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:HIT_1SoundPath] error:NULL];
    HIT_1.delegate = self;
    HIT_1.numberOfLoops = -1;
    HIT_1.volume = 1;
    [HIT_1 prepareToPlay];
}

```

Figure 10. *iGrooving*'s code: Sound file triggers and behavior (Appendix B)



In Objective-C, there are two main files where the programmer codes the background processes—View Controller Header File and the View Controller Implementation File. The programmer declares the main functions of the application in the View Controller Header File. This section is where the programmer inserts the tools needed for the program to function properly. These tools include any supporting material implemented in the application, the actions' parameters, and user interface elements, such as buttons and labels. In *iGrooving*, the sound files are the supporting materials.

In contrast, the View Controller Implementation File defines the application's behaviors, such as the rules, parameters, and actions taken by the application. In *iGrooving*, these behaviors include how to detect the steps, how to count the steps, which files to trigger at which step counts, and how each button and label behaves in the graphic user interface (GUI). Figure 11 shows *iGrooving*'s GUI.



Figure 11. iPhone's screenshot of *iGrooving*

*iGrooving's* View Controller Header File's source code is displayed in Appendix A, and Appendix B shows *iGrooving's* View Controller Implementation File. Since *iGrooving* deals with audio material and building a GUI, the programmer imports and declares the following frameworks and classes: UIKit, AudioToolbox, AVFoundation, AVAudioPlayer. The core infrastructure of an iOS app is built from objects in the UIKit framework. The objects in this framework provide all of the support for handling events, displaying content on the screen, and interacting with the rest of the system. The AudioToolbox framework provides interfaces for recording, playback, and stream parsing. The AV Foundation framework provides an Objective-C interface for managing and playing audio-visual media. An instance of the AVAudioPlayer class provides playback of audio data from a file or memory (Apple Inc., 2013).

## V. CONCLUSION

*iGrooving* is a generative music mobile application that comments on the definition of musical instruments, musical performance as a shared experience, and the roles and relationship of composer, performer, and audience. Beyond this, *iGrooving*'s performer has some compositional autonomy and becomes the only audience member. This situation proposes a new listening environment for compositions. Specifically, the musical performance becomes a solipsistic experience, placing into question the concept of musical performance as a shared experience.

The generative music and biofeedback aspects of *iGrooving* allow the runner to generate the tempi, determine when the remaining sound files get triggered, and choose the length of the performance. Therefore, besides being the performer and audience, the runner—in a sense—becomes a co-composer. In this way, each performance of *iGrooving* has a very low probability of exact replication.

I am confident that a mobile application that generates a musical composition with steps is unprecedented. By combining all these elements, *iGrooving* helps push the envelope of new music, performance practices, musical instruments, and our perceptions of musical performance as a shared experience. Using generative processes, biofeedback, and new media for musical expression has also allowed me to reevaluate my role in creating music.

Further developments in *iGrooving* will include ensuring that the application works when the phone is on lock mode, that the audio output of the application can be recorded from within the application, and that *iGrooving* can be played by an ensemble of runners.

I also wish to develop *iGrooving* for commercial release and conference exposition. I will work on fine-tuning the relationship of a desired running BPM with a musical reward within the performance, thus informing the user when they reach the desired BPM. I will also plan to make the user interface more intuitive and user-friendly. Finally, I will further develop *iGrooving* in order to ask new questions, such as: Can a group of runners collaborate and potentially play together musically as a running ensemble? Can the performance be shared through social networks? And what is the experience when a non-performing audience views a live-streamed performance of *iGrooving*?

## BIBLIOGRAPHY

- Allan, A. (2012). *Learning iOS Programming*, 2<sup>nd</sup> ed. Sebastopol, CA: O'Reilly Media, Inc.
- Alpen, A. (1995). *Techniques for Algorithmic Composition of Music*. Paper, Hampshire College.
- Apple Inc. (2013). *iTunes U*. <http://www.apple.com/education/itunes-u/> (accessed December 10, 2012).
- Apple Inc. (2013) *Developer*. <https://developer.apple.com> (accessed December 10, 2012).
- Bernard, J. W. (2003). Minimalism, Postminimalism, and the Resurgence of Tonality in Recent American Music. *American Music* 21(1), 112-133.
- Boulanger, R., & Lazzarini, V. (2011). *The Audio Programming Book*. Cambridge, Mass.: MIT Press.
- Brown, A. R. (2005). *Generative Music in Live Performance*. Paper presented at the Australasian Computer Music Conference. Brisbane, Australia, ACMA: 23-26.
- Chechile, A. (2013). <http://alexchechile.com/> (accessed December 10, 2009).
- Collins, N. (2008). *INFNO: Generating Synth Pop and Electronic Dance Music on Demand*. Paper, University of Sussex.
- Collins, N. (2013). *iPhone/iPad Apps*. <http://www.sussex.ac.uk/Users/nc81/iphone.html> (accessed June 20, 2012).
- Collins, N. (2009). Musical Form and Algorithmic Composition. *Contemporary Music Review* 28(1), 103–114.
- Collins, N. (2008). The Analysis of Generative Music Programs. *Organized Sound: An International Journal Of Music Technology* 13(3), 237.
- Collins, N. (2002) *The BBCut Library*. (Paper presented at the International Computer Music Conference).
- Cope, D. (1996). *Experiments in Musical Intelligence*. Madison, Wisconsin: A-R Editions.
- De Laubier, S. (1998). The Meta-Instrument. *Computer Music Journal* 22(1), 25-29.

- Eno, B. (2013). *Generative Music*. <http://www.generativemusic.com> (accessed June 20<sup>th</sup>, 2012).
- Eno, B. (1996). *Generative Music*. Transcript of talk from Imagination Conference, San Francisco. In *Motion Magazine*. <http://www.inmotionmagazine.com/enol.html>.
- Fung, J., Garten, A., & Mann, S. (2007). *Deconcert: Bathing in the Light, Sound, and Waters of the Musical Brainbaths*. <http://wearcam.org/icmc2007/cr185882595172.pdf> (accessed December 10, 2009).
- Glass, P. (1994). *Music with Changing Parts* [CD Recording]. New York, NY: Nonesuch.
- Haas, L. F. (2003). Hans Berger (1873-1941), Richard Caton (1842-1926), and electroencephalography. (Neurological Stamp). *Journal of Neurology, Neurosurgery and Psychiatry* 74.1 9(1). Academic OneFile. Gale. Florida International University <http://find.galegroup.com.ezproxy.fiu.edu/gtx/start.do?prodId=AONE&userGroupName=flstuniv> (accessed December 10, 2009).
- Harley, J. (1995). Generative Processes in Algorithmic Composition: Chaos and Music. *Leonardo* 28(3), 221-224.
- Johnson, T. (1998). *Automatic Music*. (Paper presented at the Conference on Musical Information '98 (JIM '98). La-Londe-les-Maures, France).
- Kim, R. Y. (1999). From New York to Vermont: Conversation with Steve Reich. *Current Musicology*, 345-366.
- Lepervanche, D. (2009). *Music of the Body: Biofeedback as a Method of Performance*. (Unpublished Senior Essay, Florida International University).
- Lucier, A., & Margolin, A. (1982). Conversation with Alvin Lucier. *Perspectives of New Music* 20(1/2), 50-58.
- Lyon, E. <http://www.sarc.qub.ac.uk/~elyon/> (last accessed December 9, 2009)
- Miranda, E. R., & Brouse, A. (2005). Interfacing the Brain Directly with Musical Systems: On Developing Systems for Making Music with Brain Signals. *Leonardo* 38(4), 331-336.
- Miranda, E. R., Roberts, S., & Stokes, M. (2004) On Generating EEG for Controlling Musical Systems. *Biomedizinische Technik* 49(1), 75-76.

- Moore, K., & Sanders, L. Lucier, Alvin (b 1931). *Grove Music Online*. *Oxford Music Online*. <http://www.oxfordmusiconline.com/subscriber/article/grove/music/47065> (accessed December 10, 2009).
- Oliveros, P. (2013). *Deep Listening Institute*. <http://deeplistening.org/site/adaptiveuse> (accessed June 20, 2012).
- Osterreich, N. (1977). Music with Roots in the Aether. *Perspectives of New Music* 16(1), 214-228.
- Reich, S. (1971). *Drumming* [CD Recording]. New York, NY: Nonesuch.
- Reich, S. (1987). *The Four Sections / Music for Mallet Instruments, Voices, and Organ* [CD Recording]. New York, NY: Nonesuch.
- Roads, C. (1996). *The Computer Music Tutorial*. Cambridge, MA: MIT Press.
- Rosenboom, D. (2003). Propositional Music from Extended Musical Interface with the Human Nervous System. *Annals of the New York Academy of Sciences* 999, 263-271.
- Scheinin, R. (2008, April 28). Laptop Orchestras Bridge the Distance. *Mercury News* <http://slork.stanford.edu/media/2008/rim-of-wire/mercury2008.4.28.html> (accessed October 29, 2012).
- Society for Electro-Acoustic Music in the United States (2013). *SEAMUS*. [http://www.seamusonline.org/?page\\_id=5](http://www.seamusonline.org/?page_id=5) (accessed January 28, 2013).
- Sloboda, J. A., (Ed.). (1988). *Generative Processes in Music: The Psychology of Performance, Improvisation and Composition*. Oxford: Clarendon Oxford, United Kingdom.
- Smallwood, S., Trueman, D., Cook, P. R., & Wang, G. (2008). Composing for Laptop Orchestra. *Computer Music Journal* 32(1), 9-25.
- Straebel, V. (2008). Technological Implications of Phill Niblock's Drone Music, Derived from Analytical Observations of Selected Works for Cello and String Quartet on Tape. *Organised Sound* 13(3), 225-235.
- Sun, C. (2006). The theatre of minimalist music: Analysing La Monte Young's "composition 1960 #7". *Context*, 37-50. <http://ezproxy.fiu.edu/login?url=http://search.proquest.com/docview/1465028?accountid=10901> (accessed March 13, 2013).
- Supper, M. (2001). A Few Remarks on Algorithmic Composition. *Computer Music Journal* 25(1), 48-53.

- Wang, G., Essl, G., & Penttinen, H. (2008). Do Mobile Phones Dream of Electric Orchestras?. *RILM Abstracts of Music Literature, EBSCOhost* (accessed April 25, 2013).
- Whittall, A. (2012). Music Reviews: The Reich Stuff. *Musical Times* 153, 119-120. <http://ezproxy.fiu.edu/login?url=http://search.proquest.com/docview/1266688383?accountid=10901> (accessed April 26, 2013).
- Wooller, R., Brown, A. R., Miranda, E. R., Berry, R., & Diederich, J. (2005). A Framework for Comparison of Processes in Algorithmic Music Systems. Proceedings of the *Generative Arts Practice*, Sydney, Australia. *Creativity and Cognition Studios Press*, 109-124.
- Worth, P., & Stepney S. (2005). Growing Music: Musical Interpretations of L-systems. *Applications of Evolutionary Computing*, 545-550.



## APPENDICES

### Objective-C code for *iGrooving*

#### APPENDIX A

##### View Controller Header File

```
#import <UIKit/UIKit.h>
#import <AudioToolbox/AudioToolbox.h>
#import <AVFoundation/AVFoundation.h>
#import <AVFoundation/AVAudioPlayer.h>

@interface ViewController : UIViewController <UIAccelerometerDelegate,
AVAudioPlayerDelegate>

{
    float px;
    float py;
    float pz;

    int numSteps;
    BOOL isChange;
    BOOL isSleeping;

    IBOutlet UILabel *viewTime;
    NSTimer *viewTimeTicker;

    NSTimer *soundsTimer;

    int timeSpent;

    AVAudioPlayer *playerPad;
    AVAudioPlayer *KickPlayer;
    AVAudioPlayer *BOWL_1;
    AVAudioPlayer *HIT_1;
    AVAudioPlayer *BOWL_2;
    AVAudioPlayer *HIT_2;
    AVAudioPlayer *BOWL_3;
```

```

    AVAudioPlayer *HIT_3;
    AVAudioPlayer *BOWL_4;
    AVAudioPlayer *HIT_4;
    AVAudioPlayer *BOWL_1B;
    AVAudioPlayer *MHIT_1;
    AVAudioPlayer *MHIT_2;
    AVAudioPlayer *MHIT_3;

    BOOL activityHasStarted;

    IBOutlet UIButton *startButton;
    IBOutlet UIButton *stopButton;
    IBOutlet UIButton *resetTimerButton;
    IBOutlet UIButton *resetStepCounterButton;

}

@property (retain, nonatomic) IBOutlet UILabel *stepCountLabel;

- (IBAction)reset:(id)sender;
- (IBAction)stop;
- (IBAction)start;
- (IBAction)resetTimer;
- (IBAction)playSound;
- (IBAction)ResetAllPressed;

- (void) showActivity;

- (NSString*)GetPrettyTime;

@end

```

## APPENDIX B

### View Controller Implementation File

```
#import "ViewController.h"

#define kUpdateFrequency 60.0

@implementation ViewController
@synthesize stepCountLabel;

BOOL isInvalid = false;

AVAudioPlayer *player;

- (IBAction)start
{
    if(activityHasStarted)
        return;

    activityHasStarted = YES;
    viewTimeTicker = [NSTimer scheduledTimerWithTimeInterval:1.0 target:self
selector:@selector(showActivity) userInfo:nil repeats:YES];

    isInvalid = false;

    //[player play];
    startButton.hidden = YES;
    stopButton.hidden = NO;
    resetTimerButton.hidden = YES;
    //resetStepCounterButton.hidden = YES;
}

- (IBAction)stop
{
    if(! isInvalid)
    {
        [viewTimeTicker invalidate];
        isInvalid = true;

        [self StopSoundClips];
        activityHasStarted = NO;
        resetTimerButton.hidden = NO;
        //resetStepCounterButton.hidden = NO;
    }
}
```

```

    }
}

- (IBAction)resetTimer
{
    viewTime.text = @"0:00:00";
    timeSpent = 0;
    activityHasStarted = NO;
    startButton.hidden = NO;
    stopButton.hidden = YES;
    //resetStepCounterButton.hidden = YES;
    resetTimerButton.hidden = YES;
}

- (IBAction)reset:(id)sender
{
    numSteps = 0;
    self.stepCountLabel.text = [NSString stringWithFormat:@"%d", numSteps];
}

- (IBAction)playSound
{
    //CFBundleRef mainBundle = CFBundleGetMainBundle();
    //CFURLRef soundFileURLRef;
    //soundFileURLRef = CFBundleCopyResourceURL(mainBundle, (CFStringRef)
@"PAD1", CFSTR ("wav"), NULL);
    //UInt32 soundID;
    //AudioServicesCreateSystemSoundID(soundFileURLRef, &soundID);
    //AudioServicesPlaySystemSound(soundID);
}

- (void) showActivity
{
    //int currentTime = [viewTime.text intValue];
    //int newTime = currentTime + 1;
    //viewTime.text = [NSString stringWithFormat:@"%d", newTime];
    viewTime.text = [self GetPrettyTime:timeSpent++];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Release any cached data, images, etc that aren't in use.
}

```

```

#pragma mark - View lifecycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    // Enable listening to the accelerometer
    [[UIAccelerometer sharedAccelerometer] setUpdateInterval:1.0 / kUpdateFrequency];
    [[UIAccelerometer sharedAccelerometer] setDelegate:self];

    px = py = pz = 0;
    numSteps = 0;

    self.stepCountLabel.text = [NSString stringWithFormat:@"%d", numSteps];

    [self InitializeSoundClips];

    [UIApplication sharedApplication].idleTimerDisabled = YES;

    NSError *setCategoryErr = nil;
    NSError *activationErr = nil;
    [[AVAudioSession sharedInstance] setCategory: AVAudioSessionCategoryPlayback
error:&setCategoryErr];
    [[AVAudioSession sharedInstance] setActive:YES error:&activationErr];
}

- (void)viewDidUnload
{
    [self setStepCountLabel:nil];
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
}

- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
}

- (void)viewWillDisappear:(BOOL)animated

```

```

{
    [super viewWillDisappear:animated];
}

- (void)viewDidDisappear:(BOOL)animated
{
    [super viewDidDisappear:animated];
}

-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    // Return YES for supported orientations
    return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
}

// UIAccelerometerDelegate method, called when the device accelerates.
-(void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration *)acceleration
{
    float xx = acceleration.x;
    float yy = acceleration.y;
    float zz = acceleration.z;

    float dot = (px * xx) + (py * yy) + (pz * zz);
    float a = ABS(sqrt(px * px + py * py + pz * pz));
    float b = ABS(sqrt(xx * xx + yy * yy + zz * zz));

    dot /= (a * b);

    if (dot <= 0.82)
    {
        if (!isSleeping)
        {
            isSleeping = YES;
            [self performSelector:@selector(wakeUp) withObject:nil afterDelay:0.3];
            numSteps += 1;

            // 2. the first step triggers the timer to start and the timer displays hours, minutes,
            seconds and milliseconds
            if (numSteps == 1)
            {
                [self start];
            }
        }
    }
}

```

```

    }
    self.stepCountLabel.text = [NSString stringWithFormat:@"%d", numSteps];

    [self CheckStepTriggers:numSteps];

    //CFBundleRef mainBundle = CFBundleGetMainBundle();
    //CFURLRef soundFileURLRef;
    //soundFileURLRef = CFBundleCopyResourceURL(mainBundle, (CFStringRef)
@"KICK", CFSTR ("wav"), NULL);
    //UInt32 soundID;
    //AudioServicesCreateSystemSoundID(soundFileURLRef, &soundID);
    //AudioServicesPlaySystemSound(soundID);

    NSString *soundFilePath = [[NSBundle mainBundle]
pathForResource:@"KICK" ofType:@"wav"];
    NSURL *soundFileURL = [NSURL fileURLWithPath:soundFilePath];
    player = [[AVAudioPlayer alloc] initWithContentsOfURL:soundFileURL
error:nil];

    [player play];

    NSLog(@"%d steps", numSteps);

    }
    }
    px = xx; py = yy; pz = zz;
}

- (void)wakeUp
{
    isSleeping = NO;
}
- (void)dealloc
{
    [stepCountLabel release];
    [super dealloc];
}
#pragma pretty timer
- (NSString*)GetPrettyTime :(int)duration
{
    int hours = duration / 3600;
    int remainingMinutes = duration % 3600;

    int minutes = remainingMinutes / 60;
    int seconds = remainingMinutes % 60;

```

```

NSString *minutesString = [NSString stringWithFormat:@"%d", minutes];
if(minutes < 10)
    minutesString = [NSString stringWithFormat:@"0%d", minutes];

NSString *secondsString = [NSString stringWithFormat:@"%d", seconds];
if(seconds < 10)
    secondsString = [NSString stringWithFormat:@"0%d", seconds];

return [NSString stringWithFormat:@"%d:%@:%@",hours, minutesString,
secondsString];
}
#pragma STEPS and TRIGGERS
-(void)CheckStepTriggers:(int)stepCount
{
    //1. triggering multiple audio files at different step counts and have them loop
    throughout out the performance.
    /*STEPS and TRIGGERS
    STEP 1 - BOWL_1
    STEP 325 - HIT_1
    STEP 650 - BOWL_2
    STEP 975 - HIT_2
    STEP 1350 - BOWL_3
    STEP 1725 - HIT_3
    STEP 2100 - BOWL_4
    STEP 2475 - HIT_4
    STEP 2850 - BOWL_1
    STEP 3225 - MHIT_1
    STEP 3550 - MHIT_2
    STEP 3875 - MHIT_3*/

    switch (stepCount)
    {
        case 1:
            [BOWL_1 play];
            break;
        case 325:
            [HIT_1 play];
            break;

        case 650:
            [BOWL_2 play];
            break;
    }
}

```



```

    case 975:
        [HIT_2 play];
        break;

    case 1350:
        [BOWL_3 play];
        break;

    case 1725:
        [HIT_3 play];
        break;

    case 2100:
        [BOWL_4 play];
        break;

    case 2475:
        [HIT_4 play];
        break;

    case 2850:
        [BOWL_1B play];
        break;

    case 3225:
        [MHIT_1 play];
        break;

    case 3550:
        [MHIT_2 play];
        break;

    case 3875:
        [MHIT_3 play];
        break;

    default:
        break;
}
}
-(void)InitializeSoundClips
{
    NSString *soundFilePathKick = [[NSBundle mainBundle]
pathForResource:@"KICK" ofType:@"wav"];
    NSURL *soundFileURLKick = [NSURL URLWithString:soundFilePathKick];

```

```

KickPlayer = [[AVAudioPlayer alloc] initWithContentsOfURL:soundFileURLKick
error:nil];
[KickPlayer prepareToPlay];

NSString *soundFilePathPad = [[NSBundle mainBundle] pathForResource:@"PAD2"
ofType:@"wav"];
NSURL *soundFileURLPad = [NSURL fileURLWithPath:soundFilePathPad];
playerPad = [[AVAudioPlayer alloc] initWithContentsOfURL:soundFileURLPad
error:nil];
playerPad.numberOfLoops = -1; //infinite
[playerPad prepareToPlay];

NSString *BOWL_1SoundPath = [[NSBundle mainBundle]
pathForResource:@"BOWL_01" ofType:@"wav"];
BOWL_1 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:BOWL_1SoundPath] error:NULL];
BOWL_1.delegate = self;
BOWL_1.numberOfLoops = -1;
BOWL_1.volume = 1;
[BOWL_1 prepareToPlay];

NSString *BOWL_2SoundPath = [[NSBundle mainBundle]
pathForResource:@"BOWL_02" ofType:@"wav"];
BOWL_2 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:BOWL_2SoundPath] error:NULL];
BOWL_2.delegate = self;
BOWL_2.numberOfLoops = -1;
BOWL_2.volume = 1;
[BOWL_2 prepareToPlay];

NSString *BOWL_3SoundPath = [[NSBundle mainBundle]
pathForResource:@"BOWL_03" ofType:@"wav"];
BOWL_3 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:BOWL_3SoundPath] error:NULL];
BOWL_3.delegate = self;
BOWL_3.numberOfLoops = -1;
BOWL_3.volume = 1;
[BOWL_3 prepareToPlay];

NSString *BOWL_4SoundPath = [[NSBundle mainBundle]
pathForResource:@"BOWL_04" ofType:@"wav"];
BOWL_4 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:BOWL_4SoundPath] error:NULL];
BOWL_4.delegate = self;
BOWL_4.numberOfLoops = -1;

```

```
BOWL_4.volume = 1;  
[BOWL_4 prepareToPlay];
```

```
NSString *HIT_1SoundPath = [[NSBundle mainBundle]  
pathForResource:@"HIT_01" ofType:@"wav"];  
HIT_1 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL  
fileURLWithPath:HIT_1SoundPath] error:NULL];  
HIT_1.delegate = self;  
HIT_1.numberOfLoops = -1;  
HIT_1.volume = 1;  
[HIT_1 prepareToPlay];
```

```
NSString *HIT_2SoundPath = [[NSBundle mainBundle]  
pathForResource:@"HIT_02" ofType:@"wav"];  
HIT_2 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL  
fileURLWithPath:HIT_2SoundPath] error:NULL];  
HIT_2.delegate = self;  
HIT_2.numberOfLoops = -1;  
HIT_2.volume = 1;  
[HIT_2 prepareToPlay];
```

```
NSString *HIT_3SoundPath = [[NSBundle mainBundle]  
pathForResource:@"HIT_03" ofType:@"wav"];  
HIT_3 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL  
fileURLWithPath:HIT_3SoundPath] error:NULL];  
HIT_3.delegate = self;  
HIT_3.numberOfLoops = -1;  
HIT_3.volume = 1;  
[HIT_3 prepareToPlay];
```

```
NSString *HIT_4SoundPath = [[NSBundle mainBundle]  
pathForResource:@"HIT_04" ofType:@"wav"];  
HIT_4 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL  
fileURLWithPath:HIT_4SoundPath] error:NULL];  
HIT_4.delegate = self;  
HIT_4.numberOfLoops = -1;  
HIT_4.volume = 1;  
[HIT_4 prepareToPlay];
```

```
NSString *MHIT_1SoundPath = [[NSBundle mainBundle]  
pathForResource:@"MHIT_01" ofType:@"wav"];  
MHIT_1 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL  
fileURLWithPath:MHIT_1SoundPath] error:NULL];  
MHIT_1.delegate = self;  
MHIT_1.numberOfLoops = -1;
```

```

MHIT_1.volume = 1;
[MHIT_1 prepareToPlay];

NSString *MHIT_2SoundPath = [[NSBundle mainBundle]
pathForResource:@"MHIT_02" ofType:@"wav"];
MHIT_2 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:MHIT_2SoundPath] error:NULL];
MHIT_2.delegate = self;
MHIT_2.numberOfLoops = -1;
MHIT_2.volume = 1;
[MHIT_2 prepareToPlay];

NSString *MHIT_3SoundPath = [[NSBundle mainBundle]
pathForResource:@"MHIT_03" ofType:@"wav"];
MHIT_3 = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:MHIT_3SoundPath] error:NULL];
MHIT_3.delegate = self;
MHIT_3.numberOfLoops = -1;
MHIT_3.volume = 1;
[MHIT_3 prepareToPlay];

}
-(void)StopSoundClips
{
    if([playerPad isPlaying])
        [playerPad stop];

    if([KickPlayer isPlaying])
        [KickPlayer stop];

    if([BOWL_1 isPlaying])
        [BOWL_1 stop];

    if([HIT_1 isPlaying])
        [HIT_1 stop];

    if([BOWL_2 isPlaying])
        [BOWL_2 stop];

    if([HIT_2 isPlaying])
        [HIT_2 stop];

    if([BOWL_3 isPlaying])
        [BOWL_3 stop];
}

```

```

if([HIT_3 isPlaying])
    [HIT_3 stop];

if([BOWL_4 isPlaying])
    [BOWL_4 stop];

if([HIT_4 isPlaying])
    [HIT_4 stop];

if([BOWL_1B isPlaying])
    [BOWL_1B stop];

if([MHIT_1 isPlaying])
    [MHIT_1 stop];

if([MHIT_2 isPlaying])
    [MHIT_2 stop];

if([MHIT_3 isPlaying])
    [MHIT_3 stop];

KickPlayer.currentTime = 0;
playerPad.currentTime = 0;
BOWL_1.currentTime = 0;
HIT_1.currentTime = 0;
BOWL_2.currentTime = 0;
HIT_2.currentTime = 0;
BOWL_3.currentTime = 0;
HIT_3.currentTime = 0;
BOWL_4.currentTime = 0;
HIT_4.currentTime = 0;
BOWL_1B.currentTime = 0;
MHIT_1.currentTime = 0;
MHIT_2.currentTime = 0;
MHIT_3.currentTime = 0;
}
-(IBAction)ResetAllPressed
{
    timeSpent = 0;
    numSteps = 0;
    viewTime.text = [self GetPrettyTime:timeSpent++];
    self.stepCountLabel.text = [NSString stringWithFormat:@"%d", numSteps];
}
@end

```