

UNIVERSITY OF OKLAHOMA  
GRADUATE COLLEGE

GENSYNTH: COLLABORATIVELY EVOLVING NOVEL SYNTHETIC MUSICAL  
INSTRUMENTS ON THE WEB

A THESIS  
SUBMITTED TO THE GRADUATE FACULTY  
in partial fulfillment of the requirements for the  
Degree of  
MASTER OF SCIENCE

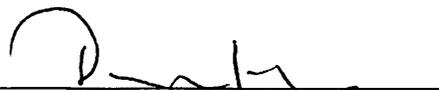
By  
DAVID LUKE RICE  
Norman, Oklahoma  
2015

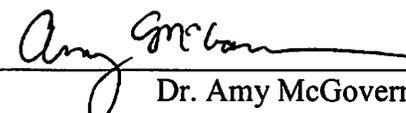
GENSYNTH: COLLABORATIVELY EVOLVING NOVEL SYNTHETIC MUSICAL  
INSTRUMENTS ON THE WEB

A THESIS APPROVED FOR THE  
SCHOOL OF COMPUTER SCIENCE

BY

  
\_\_\_\_\_  
Dr. Deborah Frytten, Chair

  
\_\_\_\_\_  
Dr. Dean Hougen

  
\_\_\_\_\_  
Dr. Amy McGovern

© Copyright by DAVID LUKE RICE 2015  
All Rights Reserved.

## Table of Contents

List of Tables .....	vii
List of Figures.....	viii
Abstract.....	xi
Chapter 1: Introduction.....	1
Chapter 2: Background.....	4
Evolutionary Computation and Art .....	4
Evolutionary Sound Synthesis.....	8
Evolutionary Fitness .....	10
Evolutionary Phenotype .....	13
Evolutionary Genotype.....	14
NeuroEvolution of Augmenting Topologies .....	20
Picbreeder .....	23
GenSynth .....	25
Chapter 3: Design and Implementation .....	26
asNEAT Instrument Evolution and Audio Synthesis Library .....	27
SN Nodes .....	28
Node Connections .....	31
Instrument Mutation .....	31
Split Connection Mutation .....	32
Add Oscillator Mutation.....	32
Add Connection Mutation .....	32
Connections Weight Mutation.....	33

Node Parameters Mutation .....	33
Instrument Breeding .....	33
JSON Serialization .....	34
asNEAT-Visualizer .....	34
Audio Visualizations .....	35
Oscilloscope .....	35
Spectrum.....	36
Spectrogram.....	37
Force Graph .....	38
Instrument Visualization .....	39
GenSynth Web Application.....	40
Chapter 4: Methodology.....	41
GenSynth Exploration Usability Study .....	41
GenSynth Collaborative Platform Usability Study .....	42
Three Platform Goals .....	43
Six CIE Challenges.....	46
CIE Challenge 1. Empower Users Regardless of Skill Level .....	46
CIE Challenge 2. Overcome Single User Fatigue .....	47
CIE Challenge 3. Create Diverse Content with Many Users .....	48
CIE Challenge 4. Enable Collaboration with Individual Choice .....	48
CIE Challenge 5. Encourage Participation .....	49
CIE Challenge 6. Balance Work Exploration & Exploitation.....	49
Chapter 5: Results & Discussion.....	50

Usability Study Results .....	50
GenSynth Platform Study Results .....	54
The Discovery, Exploration, and Publish Goals .....	55
Meeting Six CIE Challenges .....	59
Meeting CIE Challenge 1. Empower Users Regardless of Skill Level .....	59
Meeting CIE Challenge 2. Overcome Single User Fatigue.....	60
Meeting CIE Challenge 3. Create Diverse Content with Many Users .....	61
Meeting CIE Challenge 4. Enable Collaboration with Individual Choice .....	62
Meeting CIE Challenge 5. Encourage Participation.....	62
Meeting CIE Challenge 6. Balance Work Exploration & Exploitation .....	65
Chapter 6: Conclusions & Future Work.....	66
Future Work.....	66
References .....	70
Appendix A: Source .....	73
Appendix B: Usability Study 1 Procedures.....	74
Appendix C: Usability Study 1 Survey Questions .....	76
Appendix D: Usability Study 2 Survey Questions .....	80
Appendix E: Mutation Parameters .....	83

## List of Tables

Table 1: NEAT Based Method Comparison .....	18
Table 2: Comparison of Related GA Sound Synthesis Methods .....	19
Table 3: asNEAT SN Nodes.....	29
Table 4: Mutation Chances per Step .....	83
Table 5: Connection Mutation Parameters .....	83
Table 6: Node Mutation Parameters.....	84

## List of Figures

Figure 1: Genetic Algorithm Process .....	5
Figure 2: Two individuals being crossed and mutated into new individuals. ....	6
Figure 3: IEC process integrates user opinion.....	7
Figure 4: CIE allows multiple users to evolve past the fatigue wall. ....	8
Figure 5: ADSR envelope simulating a real instrument [Russ 2009] .....	9
Figure 6: Synthesizer Topology Example [Russ 2009] – In this example a low frequency oscillator (LFO) controls some aspect of the voice oscillators (VCOs). These then get combined with noise passed into two envelopes (VCF, VCA). ....	10
Figure 7: Interactive Evolution – (a) [Dahlstedt 2001] Mutasynt. (b) [Moore and Spies 2006] djNEAT. (c) [McDermott 2008] sweeping 2d plane. ....	12
Figure 8: Genotypes – (a) [Johnson 1999] defines a set of parameters detailing how the sine waves are combined. (b) [Takala et al. 1993] Timbre Trees and (c) [Donahue 2013] Synthax Trees both define how sound is generated with functional trees. ....	16
Figure 9: Genotypes (continued) – (a) [Wehn 1998] allows multiple connections to a node in its tree structure. (b) [Garcia 2001] generates a tree (top) that builds the actual audio graph (bottom). ....	17
Figure 10: Genotypes (Continued) – (a) [Woolf and Yee-King 2003] Audioserve. (b) [Moore and Spies 2006] djNEAT directional graph. ....	18
Figure 11: Neural Network [Engelbrecht 2007] – (a) A neural network comprised of input, hidden, and output layers. (b) A single neuron combining multiple inputs to determine its output. ....	21
Figure 12: Picbreeder’s Images [Secretan et al. 2011] – The progression of images generated by a CPPN of n neurons and c connections after g generations. ....	24

Figure 13: Picbreeder’s IEC Application [Secretan et al. 2011] – Users are presented with 15 CPPN generated images at a time and they select which ones they like.....	24
Figure 14: GenSynth Platform Organization – GenSynth is a full stack platform using both client and server side technologies. ....	26
Figure 15: asNEAT SN – A synthesizer network consisting of two oscillators, three hidden layer nodes, and an output node. ....	27
Figure 16: Node Internals – (a) A generic asNEAT Node. (b) A Feedback Delay Node that creates a diminishing echo effect. ....	28
Figure 17: Instrument Off/On States –The left image shows an instrument’s off. When it changes to the on state (middle and right), the visualization shows the newly generated sound.....	35
Figure 18: Oscilloscope Visualization [Foobar2000 2014].....	36
Figure 19: Spectrum Visualization [Foobar2000 2014].....	36
Figure 20: Spectrogram Visualization – Four snapshots of this visualization over time. The arrow tracks a key features of the audio as the visualization moves left over time. Spectrograms rendered using [Foobar2000 2014]. ....	38
Figure 21: Force Graph Visualization .....	39
Figure 22: GenSynth Instrument Visualization – This visualization combines both spectrogram (left) and force graph (right) visualizations into a single component.....	39
Figure 23: First Usability Study IEC Software .....	42
Figure 24: GenSynth Front Page .....	44
Figure 25: GenSynth Evolve Page .....	45
Figure 26: GenSynth Instrument Page .....	46

Figure 27: Selected Vs Live - (Top) Initial Study. (Bottom) GenSynth. ....	51
Figure 28: More Options Toggle – (Top) Initial Study. (Bottom) GenSynth. ....	53
Figure 29: Onscreen Keyboard Setting - (Left) Initial Study. (Right) GenSynth. ....	54
Figure 30: Views per App Screen .....	55
Figure 31: Events on Each App Screen .....	56
Figure 32: Tracked Evolution Events – Events on the evolution page. ....	57
Figure 33: Instrument Component Events – Each instrument component contains UI elements for the listed instruments events.....	57
Figure 34: Number of Showing or Hiding Login Events – The number of times a user was prompted with a login popup, how many times they hid it without authenticating, and the number of times feedback was actually left or an instrument saved or starred after authenticating. ....	59
Figure 35: Play events based on the number of connected MIDI devices .....	60
Figure 36: Saved Instrument Node Finger Prints – This graph shows each of the saved instruments in GenSynth divided into the number of each type of nodes it contains ....	62
Figure 37: Instrument Play Event Percentage Based on the Number of Connected MIDI devices – For each number of connected midi devices and a combination of those that never used MIDI (0, 1 or 2) and those that did (3 or 6), this graph shows the percentage of the four instrument play events used.....	64
Figure 38: Number of Connected MIDI Devices Vs Average Session Time – Users with more connected MIDI devices tended to stay longer on the platform. ....	64
Figure 39: Number of Branches from Each Showcase .....	65

## Abstract

Crafting novel synthetic musical instruments has required both in-depth knowledge and intuition of sound synthesis learned only through significant experience in the domain. The Genetic Synthesizer Project, GenSynth, is a Collaborative Interactive Evolution (CIE) online platform designed to determine in what ways software can be developed to ease the challenges inherent in designing these instruments. Utilizing Picbreeder's collaborative evolution approach, GenSynth aims to enable both advanced and novice users to discover, explore, and share their instruments. GenSynth's development is divided into an initial usability study looking at the basic interactive evolutionary software and a second study examining its utilization when implemented in an online collaborative platform. GenSynth meets the discover, explore, and share platform goals, in addition to providing an open source base for future research. It also meets the following four out of the six primary challenges confronting CIE platforms as presented by Picbreeder: overcoming single user fatigue, creating diverse content with many users, enabling collaboration with individual choice, and encouraging participation. The CIE challenge of empowering users regardless of skill requires more data to validate if it was sufficiently met, and the final challenge of balancing artifact exploration and exploitation is not met with the current Picbreeder style platform. Finally, GenSynth utilizes a new NeuroEvolution of Augmenting Topologies variant, Audio Synthesis NEAT, which is the first instrument evolution approach that utilizes a graph genotype, NEAT crossover, multiple waveform oscillators with attack, decay, sustain, and release envelopes, hidden layer filter and composite nodes, and multiple connectable parameters per node.

## Chapter 1: Introduction

Artificial platforms are human produced environments that aid in a set of activities in a given activity space. Email, for example, is an artificial platform for extending the communication activity space by allowing near instant messages to be authored and sent to anyone with an internet connection. A goal for any artificial platform is to not only inspire novice and advanced users to seek out and discover this space and what exists there, but to further explore that space and enable them to introduce their own understandings into the mix by creating and sharing their own new works with others.

The artistic domain provides an excellent realm for software and technology to augment these tasks. Artistic software tools can inspire individuals through a variety of senses and give glimpses into new and exciting possibilities. In the audio realm, these tools can be used to search for and listen to works of music or sound (discover), and compose, edit, and reference them in the process of creating (explore) and sharing (share) new ones. Synthetic musical instrument (**SMI**) design, in particular, is an inherently difficult task that can benefit from technological aid as it requires users to both become proficient in sound production and spend large amounts of time developing an intuition in creating **SMI**'s that produce the desired sounds [Johnson 1999]. Interactive Evolutionary Computation (**IEC**) can be used to integrate human creativity into developing **SMI**s but is limited by a single user's fatigue and inability to evaluate a large number of instruments. In the image evolution domain, Secretan et al. utilized Collaborative Interactive Evolution (**CIE**) [Szumlanski et al. 2006] to limit these

inherent weaknesses of single user IEC and identified the following six challenges in developing a CIE platform [Secretan et al. 2011]:

1. **“Empowering groups, regardless of skill, to collaboratively search a design space.** While groupware often coordinates users by sharing expertise, few such projects empower users who may lack such expertise.”
2. **“Overcoming user fatigue.** In existing single-user IEC applications and CIE services, most users succumb to fatigue before generating significant products.”
3. **“Proliferating content.** Some CIE systems do not encourage a proliferation of content, but instead concentrate the efforts of many users on single decisions.”
4. **“Collaborating without diluting individual contribution.** While existing CIE systems aim to produce more meaningful output by involving many users, they frequently average the contributions of many users to generate [a work] that is not necessarily pleasing to any.”
5. **“Encouraging participation.** CIE systems need to encourage participation through recognizing user achievements and through a flexible interface.”
6. **“Balancing exploitation with exploration.** Any method for searching design space needs to balance exploitation with exploration; that is, users should be able to choose to continue evolving less prominent designs in the hope that future generations may improve.”

The online platform for collaboratively evolving novel SMIs presented in this thesis, GenSynth, utilizes the Picbreeder CIE platform model and takes inspiration from evolutionary biology as it works towards solving an aural version of the open evolutionary music and art problem of creating “unique kinds of evolutionary ‘software instruments’ that offer the possibility of deep creative engagement and enable the creative exploration of generated computational phase-spaces” [McCormack 2008]. Employing ideas from similar visual platforms [Langdon 2005; Secretan et al. 2011],

GenSynth increases the amount of the SMI domain space that can be explored by combining small IEC [Takagi 2001] traversed regions together with Collaborative Interactive Evolution [Szumlanski et al. 2006], the real-time communication of the web, and current web browsers' visual and aural capabilities. This paper demonstrates GenSynth's capabilities by showing how it meets the three artificial platform goals of discovering, exploring, and sharing and how it meets at least four of the six CIE challenges. The first challenge is left as an open question as there was not enough evidence to sufficiently prove it, and the sixth challenge was not met.

The subsequent chapters are laid out as follows. Chapter two gives a brief introduction into the evolutionary computation and sound synthesis domains in which GenSynth falls. When compared to previous approaches, GenSynth is placed in a unique cross section of how individuals are rated during the evolutionary process, what the evolved artifacts are made of, and what the user interacts with. Finally, the NeuroEvolution of Augmenting Topologies (**NEAT**) approach is described since the evolutionary approach created for GenSynth, Audio Synthesis NEAT (**asNEAT**), extends it [Stanley and Miikkulainen 2002]. Chapter three provides an implementation overview of the GenSynth platform. Chapter four lists how both the initial and final usability studies were designed, and chapter five explains the results of those studies. Finally, chapter six lists GenSynth's conclusions and details potential avenues for further study.

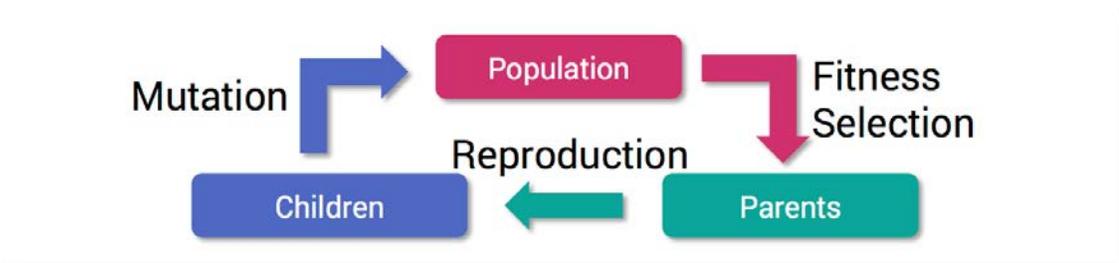
## Chapter 2: Background

### Evolutionary Computation and Art

Motivated by scientific theories of biological evolution, evolutionary computation searches for the best individuals within a given computation space by breeding and mutating populations of artificial individuals [Holland 1992]. These artificial individuals can range from potential answers to difficult questions to pieces of visual artwork. Evolutionary techniques are ideal in artistic situations since there are no specific solutions or techniques to determine an optimized “answer” to art [McDermott 2008].

To start the evolutionary process, an initial generation of individuals is typically randomly generated (**Figure 1**). Each individual in this population is then rated and given a fitness value based on a predefined fitness function characterizing how good that particular individual is or how close it matches an optimal outcome. A subset of these individuals is then selected to be parents for the next generation. This subset is predominantly composed of the highest rated individuals in a current generation, but less fit individuals can be brought in to maintain diversity. The parents then reproduce through crossing one with another or through cloning to create children that are then mutated to generate the next population of individuals. The rating step then begins again and after many cycles of rating and generating new populations, a given population will typically converge to a small region of the solution space, providing a locally optimized solution. For example, a population could consist of two individuals containing a sequence of zeros or ones (**Figure 2**). The automatic fitness metric used ( $F(x)$ , where  $x$

is the individual) is the number of ones contained in the individual, with higher values being more fit. The two parents are bred together using a single point crossover. This generates two new children each containing half of a parent. The children are then mutated by flipping a single element in their DNA. The child with the highest fitness, six in **Figure 2**, is now the best individual of the next population.

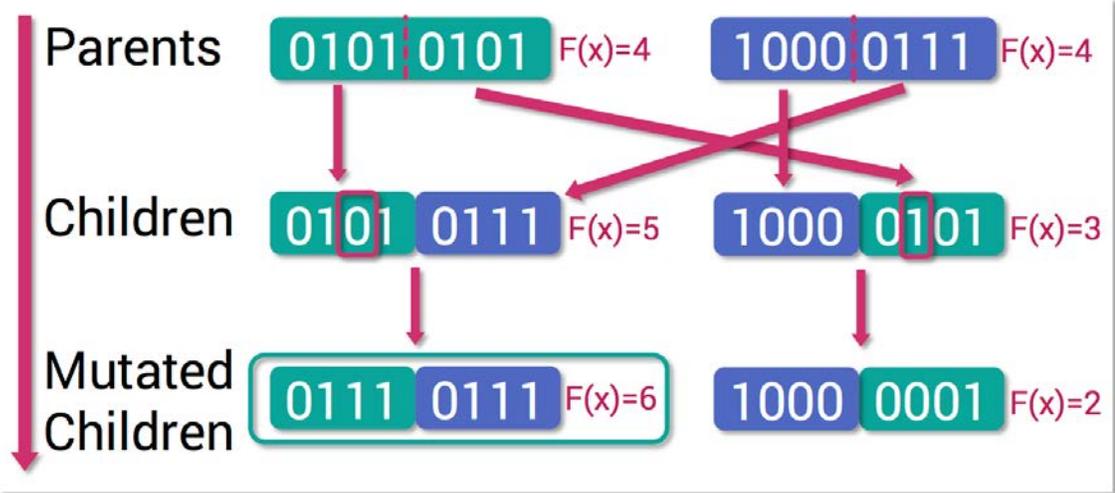


**Figure 1: Genetic Algorithm Process**

Evolutionary art can be divided by their fitness function into the categories of automated fitness, user guided fitness, community guided fitness, and a hybrid of automated and user guided fitness [Greenfield 2008; Takagi 2001; Szumlanski et al. 2006; Moore and Spires 2006].

When using an automated fitness technique, a deterministic function generates the fitness value for a given individual. This technique is typically used in evolutionary art for evolving a population to match an already created work. If an image of a basket of fruit was supplied, then better fitness values would be given to images evolved that more closely resembled the basket of fruit. There have been various attempts of encoding a value of creativity or aesthetic goodness of evolved individuals in order to

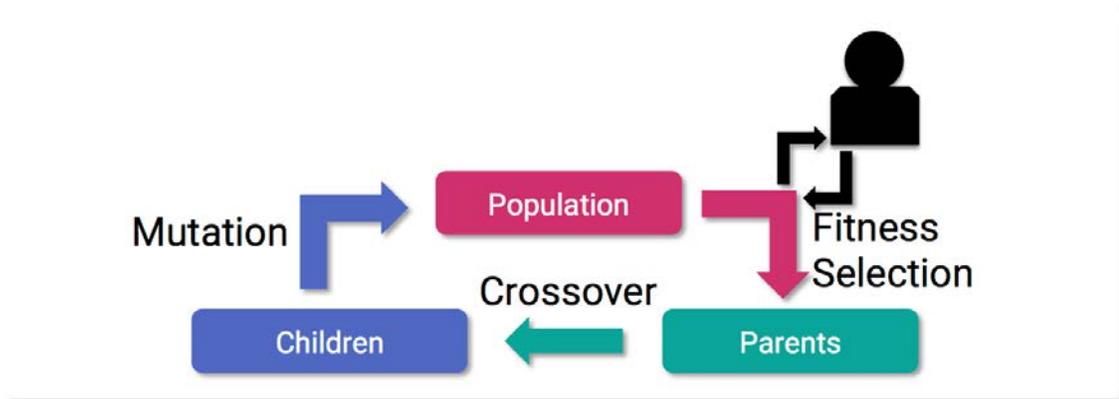
generate novels works automatically [Greenfield 2008]. In some domains, however, the fitness of an individual is subjective enough that a more human directed approach is desirable.



**Figure 2: Two individuals being crossed and mutated into new individuals.**

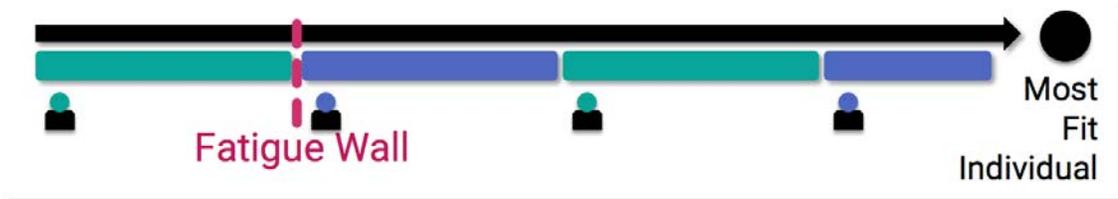
Utilizing a human’s opinions to guide evolution is Interactive Evolutionary Computation, IEC [Takagi 2001]. IEC presents a human with the individuals in a generation and tasks them with rating how good each individual is (**Figure 3**). However, since a user of this technique has to personally rate object after object, it is slow and limited to the user’s fatigue wall. This fatigue wall is how much work the user is willing to do before stopping. This limit requires techniques to reduce the size of each generation, the number of generations required to reach useful individuals, and the time required to evaluate each individual (especially with “time-variant individuals” like sound) in order for users to find the platform useful. Allowing collaboration among

multiple users can exceed this limit of a single user and let IEC be more practical [Takagi 2001].



**Figure 3: IEC process integrates user opinion**

Collaborative Interactive Evolution (**CIE**) aims to solve the issues of IEC caused by single user fatigue by having a community of users collaboratively rate individuals [Szumlanski et al. 2006]. A CIE platform is often an IEC application wrapped by a communication layer that distributes the tasks of evolving individuals across a community. The community can either all vote on a single population and evolve each generation together [Szumlanski et al. 2006], or users can evolve individuals separately and then share them so others can pick up where they left off (**Figure 4**) [Langdon 2005; Secretan et al. 2011].



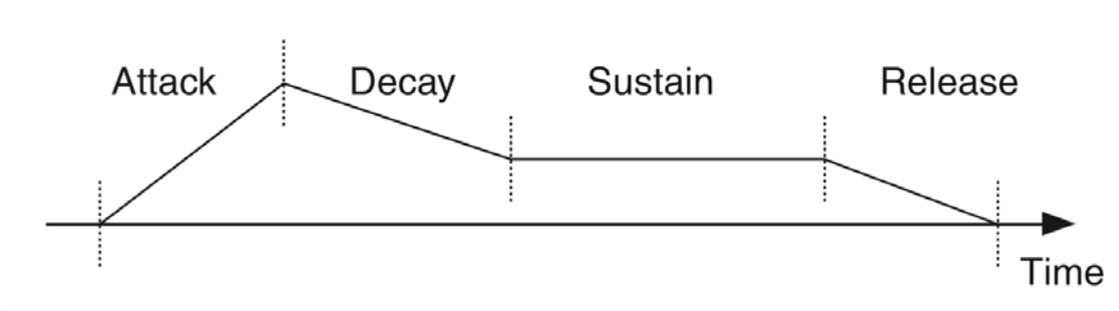
**Figure 4: CIE allows multiple users to evolve past the fatigue wall.**

The final guidance technique is a hybridization of an automated and user guided fitness. Since the number of individuals a user can rate is limited, this technique utilizes an automated fitness function to either seed new individuals to a user in a separate process or sort through a generated population and attempt to filter out individuals a user would dislike and would therefore be wasting their limited time on [Moore and Spires 2006; McDermott 2008].

### **Evolutionary Sound Synthesis**

Evolutionary Sound Synthesis is a technique in evolving the signals used to produce sound with genetic programming. Some of the most used types of sound synthesis include additive, subtractive, and Frequency Modulation (**FM**) synthesis [Russ 2009]. Additive synthesis is the combination of multiple basic oscillator signals, like a sine or square waves, to create a richer sound. Subtractive synthesis then takes an audio signal and filters out frequencies in the signal. Finally, FM synthesis is a complex synthesis method of allowing one signal to directly influence another. These synthesis methods can then be combined. For example, a carrier signal could be initially created by adding a high frequency sine and square waves together. This signal could then have specific parts of its frequency spectrum filtered out by subtractive synthesis and then modulated

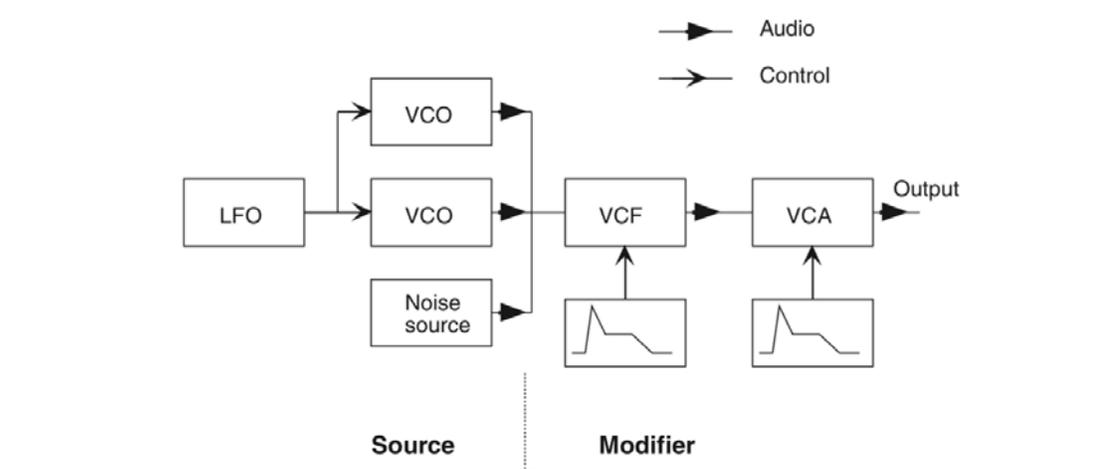
by a low frequency oscillator signal causing a vibrato effect. The overall sound characteristics that are produced through these methods is called the sound's timbre. Often the timbre is enriched by using signal envelopes that change amplitude of a signal over time. Attack-Decay-Sustain-Release (**ADSR**) envelopes are the most common envelopes found on synthetic instruments and attempt to emulate the variation of a real instrument (**Figure 5**). A trumpet, for example, creates an initial burst of sound as vibrations from the players lips begins to move through the instrument. The sound then decays back to a sustained amplitude as the player provides a constant stream of vibrations. When the player releases the note, the amplitude of the sound generated drops back to silence in a quick, but non-zero, amount of time.



**Figure 5: ADSR envelope simulating a real instrument [Russ 2009]**

A synthesizer or instrument, then, is a packaging of various oscillators, filters and other components together into a single or modular unit. What components are included and how they are connected together make up the synthesizer's topology (**Figure 6**).

Moreover, the various evolutionary art techniques developed to generate sound can be categorized by the fitness system used, how the individuals are encoded (genotype), and the observable features of what is evolved (phenotype).



**Figure 6: Synthesizer Topology Example [Russ 2009]** – In this example a low frequency oscillator (LFO) controls some aspect of the voice oscillators (VCOs). These then get combined with noise passed into two envelopes (VCF, VCA).

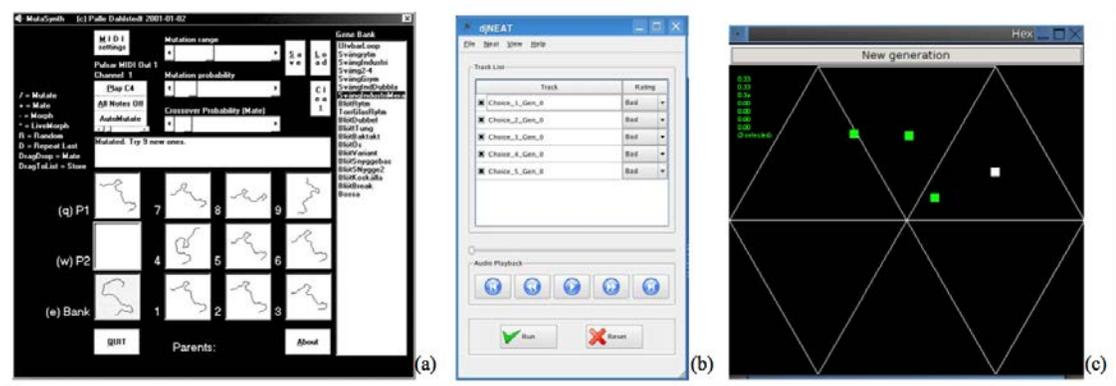
### *Evolutionary Fitness*

Similar to the general evolutionary method described earlier, evolutionary sound synthesis can be categorized by four of the evolutionary fitness guidance techniques: automated, IEC, CIE, and a hybrid of the previous.

For automated fitness, a population is evolved toward a provided sound file. An individual is then compared to the target file based on low level sample comparison [McDermott 2008], spectral analysis [Johnson 1995; Wehn 1998; Garcia 2001; McDermott 2008; Yuksel et al. 2011; Donahue 2013], parameter distances when the

target sound is generated from a defined synthesizer [McDermott 2008], or a composite of all three [McDermott 2008]. Although automated fitness allows individuals to mimic known sounds and instruments and can verify what sounds a given evolutionary system can produce, the requirement of a target file limits aural exploration, and therefore is not appropriate for creating novel sounds.

IEC moves past the limitation of evolving towards a known sound by allowing a human user to guide evolution. The current individuals can be played either one at a time in sequence [Takala et al. 1993] or by selecting them specifically in a provided graphical user interface (GUI) [Johnson 1995; Johnson 1999; Dahlstedt 2001; Moore and Spires 2006; Donahue 2013]. After auditioning the current individuals, the user then picks which ones continue on to the next generation as parents [Dahlstedt 2001], rates which ones were good/bad/average [Moore and Spires 2006], or gives a specific fitness value within a given range to each individual [Johnson 1999; Donahue 2013]. To aid the user in selecting and comparing the individuals, visual representations are often provided. In MutaSynth, a simple “worm-like line” visualization maps the evolving genotype values to curve angles (**Figure 7a**) [Dahlstedt 2001], and djNEAT provides a network graph visualization for viewing the wiring of the synthesizer generating the sounds (**Figure 7b**) [Moore and Spires 2006]. McDermott’s 2D sweeping interface and Yee-King’s SoundExplorer visually relate distances between individuals in a generation by plotting individuals on a 2D plane and allow the creation of new fine-tuned interpolated instruments by selecting points in between (**Figure 7c**) [McDermott 2008; Yee-King 2011]. However, even with more engaging interfaces, the sounds and synthesizers a single user can reach through IEC is limited by the fatigue wall.



**Figure 7: Interactive Evolution** – (a) [Dahlstedt 2001] Mutasynth. (b) [Moore and Spiers 2006] djNEAT. (c) [McDermott 2008] sweeping 2d plane.

Collaborative Interactive Evolution (CIE) brings many users together in order to evolve more advanced and potentially interesting individuals that exist beyond a single user’s fatigue wall. Audioserve integrates community driven evolution by allowing users to name and submit their sound circuits out to a public database [Woolf and Yee-King 2003]. These saved circuits are then periodically sent randomly to others using the software and integrated into their current populations, keeping the collaboration elements out of their control. Sound Gallery, a physical art installation adapted from Audioserve, utilizes the distances of the many visitors to the gallery’s speakers as a means of fitness to select the most liked sounds for generating the next population [Woolf and Yee-King 2003]. Similarly to Audioserve, Evosynth allows users to collaborate by submitting files to a database; however, its Synthax trees (detailed later) must be manually published on the Evosynth website and be manually downloaded and imported into the software in order to try them out [Donahue 2013]. GenSynth is

designed to reduce the user cost of publishing evolved instruments, finding other instruments, and experiencing them by providing a consistent web based platform.

The final guidance technique for evolutionary audio synthesis is a hybrid of the automated fitness and IEC methods. Proposed by McDermott, this technique utilizes the two methods in separate processes [McDermott 2008]. While the user is interactively rating individuals in one process, the other can be automatically evolving towards a given target file and slipping new individuals into the user's population. The hybrid technique can be further extended with an automatic creativity fitness function that can be an initial filter for the individuals presented to the user, allowing a larger amount of space to be explored than would be otherwise possible by an unaided single user [Moore and Spires 2006]. The hybrid technique could also be integrated into a CIE platform by having a background process on the server automatically evolve individuals towards given target files in order to add in extra variation.

### *Evolutionary Phenotype*

The actual artifact a user experiences as an output of evolutionary sound synthesis, an individual's phenotype, is a musical composition, sound, or instrument. Evolving a musical composition is done by evolving either a musical score, and having it played through a software instrument, or evolving the audio waveform itself. Many approaches generate compositions that require no user interaction [Magnus 2010], but a few works can generate musical compositions in real time as responses to a live performer [Biles 1994; Yee-King 2011]. In contrast to evolving an entire musical piece, many techniques

just evolve single sound samples [Johnson 1995; Wehn 1998; Johnson 1999; Garcia 2001; Woolf and Yee-King 2003; Moore and Spires 2006; McDermott 2008]. These techniques are useful for evolving a particular sound effect but are fixed to the evolved static file. For example, if a user wanted to evolve and play an entire piano, it is possible to evolve separate sound files for each piano key and pressure level, but doing so would be inefficient. Instead, an evolutionary system can evolve an entire instrument that can be played with a virtual or real MIDI keyboard [Dahlstedt 2001; McDermott 2008; Yee-King 2011; Donahue 2013]. GenSynth generates entire synthesizer instruments that can be played in the browser through an onscreen keyboard or physical MIDI controller.

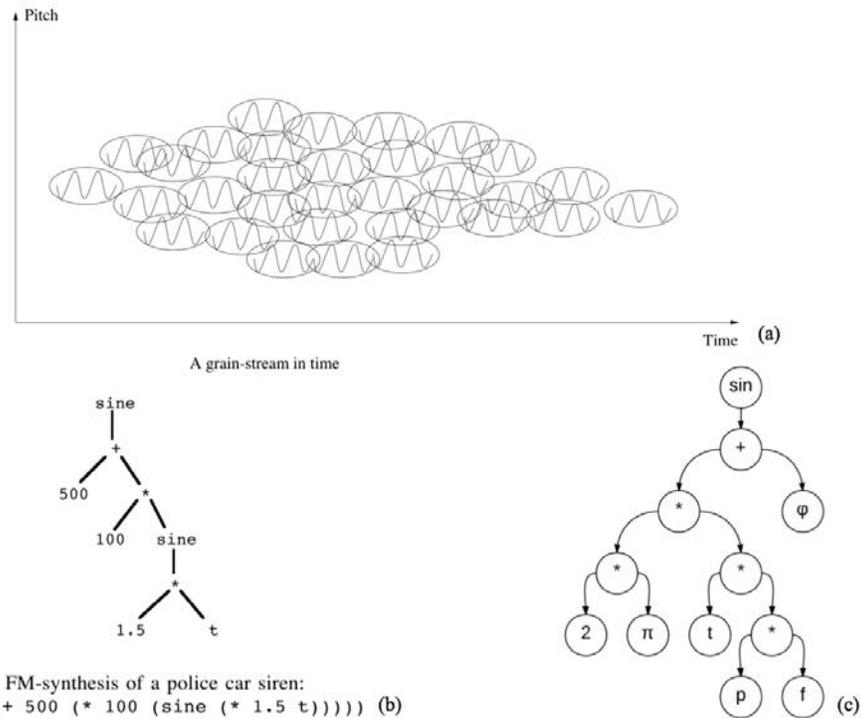
### *Evolutionary Genotype*

The various ways of engaging with an individual is often limited by its genotype, the basic genes or structures the individual is comprised of. For evolving both sound files and playable instruments, evolutionary sound techniques evolve either a list of parameters that map to fields in audio synthesizer software with a static synth topology or evolve the underlying synth topologies themselves that generate audio.

For mapping parameters to synth software, a bit string is utilized to encode what options the software exposes to its users. The software is typically either a popular commercial synthesizer or software modeled after one [Johnson 1995; Dahlstedt 2001; McDermott 2008; Yee-King 2011], but some do allow users to provide their own synthesizer definitions [Yuksel et al. 2011]. The ability to add in arbitrary synth definitions and what parameters can be evolved helps to extend the reach of the parameter mapping

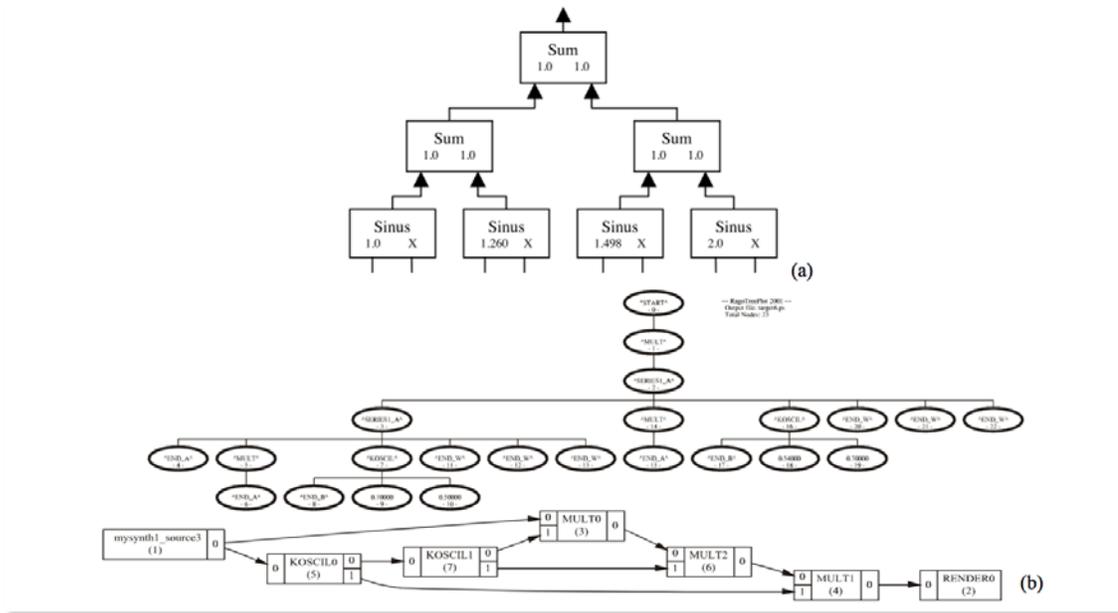
method; however, the sounds that the methods can generate are still limited by the basic synthesizers generating the sounds.

Evolutionary sound synthesis techniques that allow the underlying synth topologies to be evolved work to get past this issue by allowing additional sound hardware to be added and removed during the evolutionary process, enabling new types of sound. One of the first attempts of evolving synth topologies extended the parameter mapping technique by utilizing the parameters to define how a set of sine waves are combined and manipulated (**Figure 8**) [Johnson 1999]. When the genotype encodes synth hardware itself though, it's either encoded as a tree or graph structure with each node representing a piece of synth hardware and connections between them. The input or leaf nodes generate sound oscillations that are then piped into intermediary nodes that combine and modify sounds that get piped into a root or output node that sends the audio to a speaker. Takala's Lisp-like Timbre trees (**Figure 8b**) and Donahue's functional Synthax trees (**Figure 8c**), for example, are expressions that define the base oscillators and various processing functions [Takala et al. 1993; Donahue 2013]. Timbre Trees, though, are less geared toward starting from scratch with simple topologies and becoming more complex over time, as initial "guess" trees need to be provided. And although trees are much easier to operate on than more general graph topologies for mutating and crossing parents, they limit the inclusion of more advanced sound synthesis techniques that utilize one to many node output connections and feedback loops.



**Figure 8: Genotypes** – (a) [Johnson 1999] defines a set of parameters detailing how the sine waves are combined. (b) [Takala et al. 1993] Timbre trees and (c) [Donahue 2013] Synthax trees both define how sound is generated with functional trees.

On the other hand, some evolutionary sound techniques utilize a graph structure as its underlying topology. Wehn’s audio circuits’ graph structure enables nodes to connect to multiple inputs but still prevents loops (**Figure 9**) [Wehn 1998]. Alternately, in order to keep the benefits of a tree genotype, Garcia utilizes an acyclic tree with ordered branches as a means of encoding how a cyclic audio graph is built [Garcia 2001]. This hybrid solution, though, complicates the solution space by allowing duplicate trees that represent the same graph topology and makes exposing the underlying structure to more advanced users for direct manipulation of the synth graph more difficult.

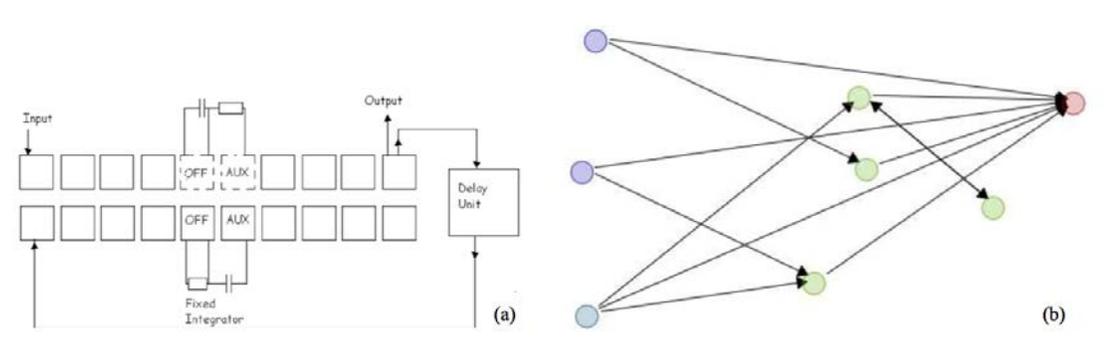


**Figure 9: Genotypes (continued)** – (a) [Wehn 1998] allows multiple connections to a node in its tree structure. (b) [Garcia 2001] generates a tree (top) that builds the actual audio graph (bottom).

To retain the graph genotype, Audioserve encodes each node’s position on 2D plane and the angle and distance ranges of potential connections (**Figure 10a**) [Woolf and Yee-King 2003]. This creates a more direct encoding of the underlying audio synthesis graph but creates a physical limitation of which nodes can connect to each other.

DjNEAT, alternatively, utilizes NeuroEvolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen 2002] in order to allow one to many node output connections, feedback loops, and complexification of the topology to evolve more complex and potentially more interesting sounds (**Figure 10b**) [Moore and Spires 2006]. However, djNEAT does not take advantage of NEAT’s ability to crossover graphs and does not include the ADSR model commonly found in synthesizers that adds richness to the sounds it produces (**Table 1**). In addition, djNEAT’s input nodes

only contain sine oscillators, increasing the generations necessary for a user to wade through and the complexity of the evolved topologies to produce sounds better suited for other types of wave oscillators. Also, the hidden layer nodes in a djNEAT network only modulate incoming signals, leaving out common synthesis techniques such as subtractive synthesis. GenSynth enhances djNEAT’s approach with a variation of NEAT designed for audio synthesis, Audio Synthesis NEAT (**asNEAT**), by allowing multiple kinds of input oscillators and hidden layer nodes, multiple connectible parameters per node, and instrument crossover.



**Figure 10: Genotypes (Continued) – (a) [Woolf and Yee-King 2003] Audioserve. (b) [Moore and Spires 2006] djNEAT directional graph.**

Method	NEAT Crossover	ADSR Envelopes	Oscillators	Connections	Hidden Layer Node Functions
djNEAT	No	No	Sine	Signal Connections	Low Level Signal Modulation
GenSynth	Current Generation	Independent Envelope per Oscillator	Sine, Square, Sawtooth, Triangle	Signal Connections, Parameter Connections	Low and Composite High level Functions (See Table 3)

**Table 1: NEAT Based Method Comparison**

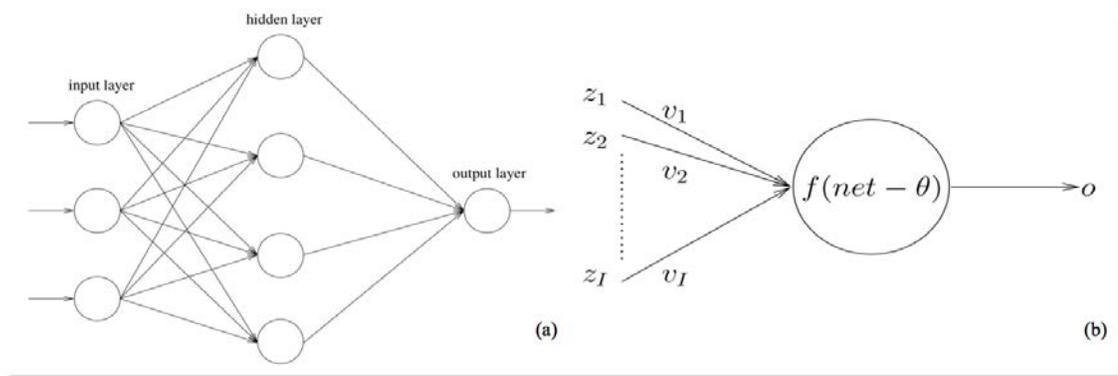
<b>Method</b>	<b>Fitness</b>	<b>Genotype</b>	<b>Phenotype</b>
Timbre Trees [Takala et al. 1993]	IEC	Timbre Tree with Initial Guess Tree	Single Sound
[Johnson 1995]	Automatic, IEC	Parameters	Single Sound
ALS Trees [Wehn 1998]	Automatic	Directed Acyclic Graph	Single Sound
Granular Synth Params [Johnson 1999]	IEC	Parameters for Granular Synthesis	Single Sound
MutaSynth [Dahlstedt 2001]	Automatic, IEC	Parameters	Playable Instrument, Worm Visualization
SSA [Garcia 2001]	Automatic	Ordered Acyclic Tree that Builds a Cyclic Graph	Single Sound
Sound Gallery & AudioServe [Woolf and Yee-King 2003]	IEC, Limited CIE	2D Grid Graph	Single Sound
djNEAT [Moore and Spires 2006]	IEC	NEAT like Graph	Single Sound, Network Graph
[McDermott 2008]	Automatic	Parameters	Single Sound, Playable Instrument, 2D plotting
Synth Space Explorer [Yee-King 2011]	Automatic, Limited IEC	Parameters	Playable Instrument
SynthDefs [Yuksel et al. 2011]	Automatic	Parameters for Synth Definitions	Playable Instrument, 2D Plotting
EvoSynth [Donahue 2013]	Automatic, IEC, External CIE	Synthax Trees	Playable Instrument
GenSynth	IEC, Integrated CIE	asNEAT Networks	Playable Instrument, Network Graph, Spectrogram

**Table 2: Comparison of Related GA Sound Synthesis Methods**

## NeuroEvolution of Augmenting Topologies

NEAT is an evolutionary genetic algorithm designed to evolve both the weights and topology of a neural network by complexifying a simple neural network (NN) in order to find an optimized solution [Stanley and Miikkulainen 2002]. GenSynth utilizes a NEAT variant, Audio Synthesis NEAT (**asNEAT**), developed for GenSynth to evolve synthetic instruments.

A neural network (**Figure 11**) is a computational model inspired by biological brains. The network is a combination of multiple neurons connected together [Engelbrecht 2007]. For a typical NN, data is sent into the network via the neuron nodes in the networks input layer. These neurons then send the data to be combined and processed in the neurons in the hidden layer, which then passes the processed data to one or more output neurons in the output layer. Each neuron in the network can have multiple weighted input connections. The weights on each connection allow inputs to have more or less influence on the neuron. The neuron processes its inputs by multiplying the input by the connections weight and then summing or multiplying all the weighted inputs together to get a net value for the neuron. This net value is then offset by an additional value and passed into a mathematical activation function. Popular activation functions consist of the linear, step, ramp, sigmoid, hyperbolic tangent, and Gaussian functions. The result of the activation function is then output from the neuron to the neurons it is connected to.



**Figure 11: Neural Network [Engelbrecht 2007]** – (a) A neural network comprised of input, hidden, and output layers. (b) A single neuron combining multiple inputs to determine its output.

The NEAT approach is especially helpful in areas where the exact structure of the network is unknown, like in audio synthesis and synth design, since the approach evolves the network from the simplest possible network, a singly connected input node and output node, to find a fit and small solution.

To solve the issue of encoding a NN so that it can be incrementally evolved, a “global innovation number” is assigned to each connection gene. When crossing different networks, this number can be used to match which parts of the two networks are interchangeable. If both networks have a connection gene with the same innovation number, then the nodes are the same but have different output connections. The two variations of output connections and nodes from the genes with the same innovation number can then be randomly selected to cross over and generate new networks.

Since new NN mutations often have a lower fitness than its siblings, NEAT utilizes speciation so that new ideas can be tested away from other networks and not weeded

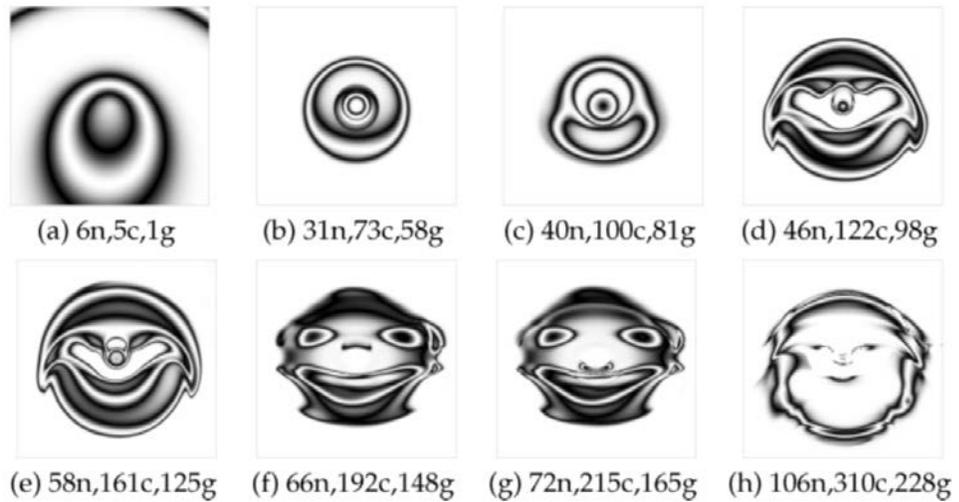
out before they have time to stabilize and show their potential benefits. In order to decide what encompasses a species, NEAT uses a distance function based on the similarity of NN topologies. More specifically, the distance is based on how many genes between two NNs don't share an innovation number (the excess and disjoint genes) and the average difference of the weights for the genes that do match. For each new generation, a random NN in each species is selected to continue on to the new generation and the new crossed and mutated networks are then tested with the old NNs that made it through to the new generation to see if any of the calculated distances are within an allowable range. If one is, the new NN is added to that species. If not, a new species is created. AsNEAT does not currently use speciation; however, it could be utilized to aid in categorizing networks across the GenSynth platform, once a platform wide innovation number is being used, to aid in searching for similarly evolved instruments that have been published.

The final issue involved in evolving an optimized NN that NEAT solves is that the networks should be prohibited from needlessly becoming large and complex. The approach solves this by starting the networks as small as possible (just an input and output layer with no hidden neurons). Since new connections and nodes are only added 'incrementally,' smaller networks are searched first before the network topologies grow. AsNEAT utilizes this attribute of NEAT by starting each new instrument with a simple oscillator and output node. However, the process of complexifying a simple network into an interesting instrument can take many generations, necessitating techniques, like Collaborative Interactive Evolution, found in GenSynth.

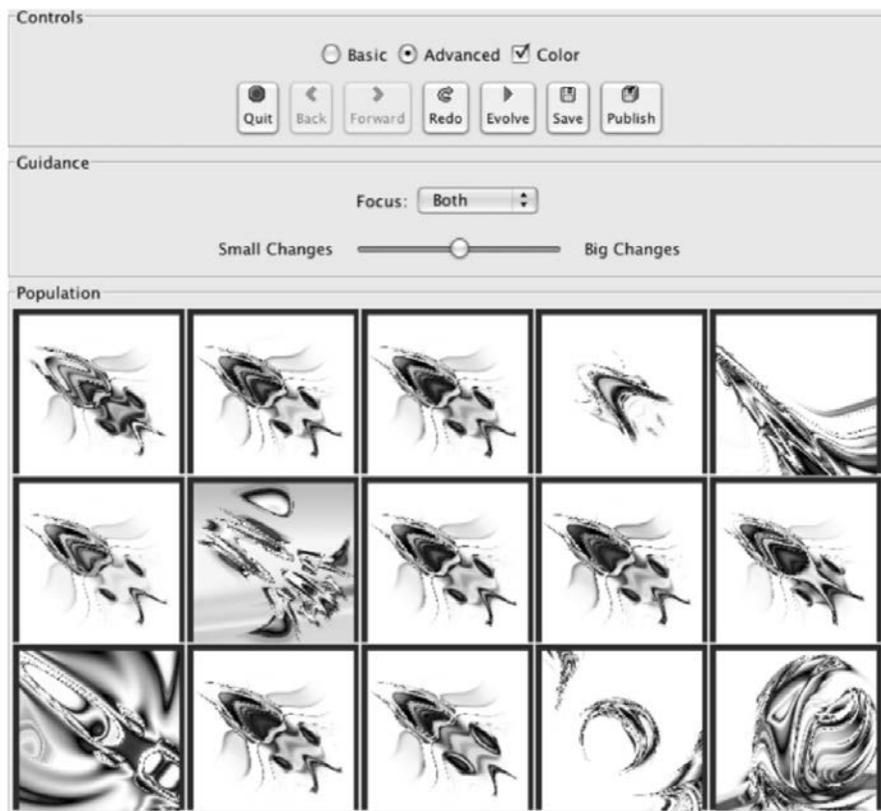
## Picbreeder

GenSynth's CIE platform takes inspiration from Picbreeder's methods of addressing CIE's primary challenges [Secretan et al. 2011]. Picbreeder's IEC application utilizes a NEAT variation to evolve Compositional Pattern Producing Networks (CPPNs) that generate 2D images (**Figure 12**) [Stanley 2007]. During evolution, users select which of 15 CPPN generated images they like to evolve a new generation (**Figure 13**). The CIE layer applied on top of the IEC software addresses the primary CIE challenges by the following [Secretan et al. 2011]:

7. Empowering users to evolve artistic works, regardless of skill
8. Overcoming single user fatigue by allowing users to pick up where previous users left off through branching and only requiring simple pass or fail ratings for individuals
9. Allowing many users to create lots of content, not many users making a single work
10. Enabling collaboration without diluting individual choice by allowing users to evolve images on their own and only publish when they are ready
11. Encouraging participation with well-designed GUI
12. Balancing exploitation of popular works and exploration of unpopular ones by allowing all saved images to be searched for, regardless of popularity



**Figure 12: Picbreeder’s Images [Secretan et al. 2011]** – The progression of images generated by a CPPN of  $n$  neurons and  $c$  connections after  $g$  generations.



**Figure 13: Picbreeder’s IEC Application [Secretan et al. 2011]** – Users are presented with 15 CPPN generated images at a time and they select which ones they like.

## GenSynth

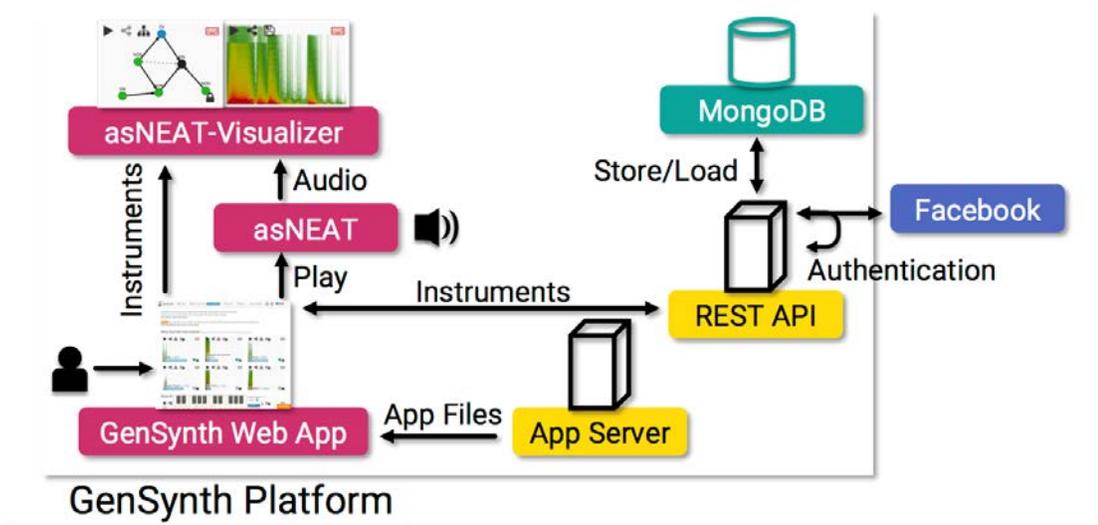
The collaborative evolutionary approach utilized by GenSynth to evolve instruments provides a platform to discover pre-existing instruments through instrument showcase and search pages, explore new instruments through guided evolution, and share instruments by publishing instruments users have evolved. To reduce audio domain specific CIE limitations, various types of initial oscillator waves and abstract processor nodes such as filters and feedback delay loops are utilized to reduce the needed complexity of an instrument and the number of generations required to evolve interesting instruments. In addition, to minimize the user strain of evaluating an instrument, visual representations of the instrument and an interactive environment to play the instrument through a physical or virtual devices are provided. However, evaluation still remains the largest bottleneck for instrument evolution as the user has to spend time playing each one to get a true sense of its timbre. Overall, GenSynth is unique to previous evolutionary audio synthesis methods in the following ways:

- Graph genotype utilizing NEAT crossover, multiple waveform oscillators with ADSR envelopes, hidden layer filter and composite nodes, and multiple connectable parameters per node.
- CIE layer integrated directly into a web platform

In addition, its capabilities as an artificial platform for evolving novel synthetic musical instruments are verified through meeting the discovery, exploration, and sharing platform goals, and through meeting four out of six of the Picbreeder CIE Challenges.

### Chapter 3: Design and Implementation

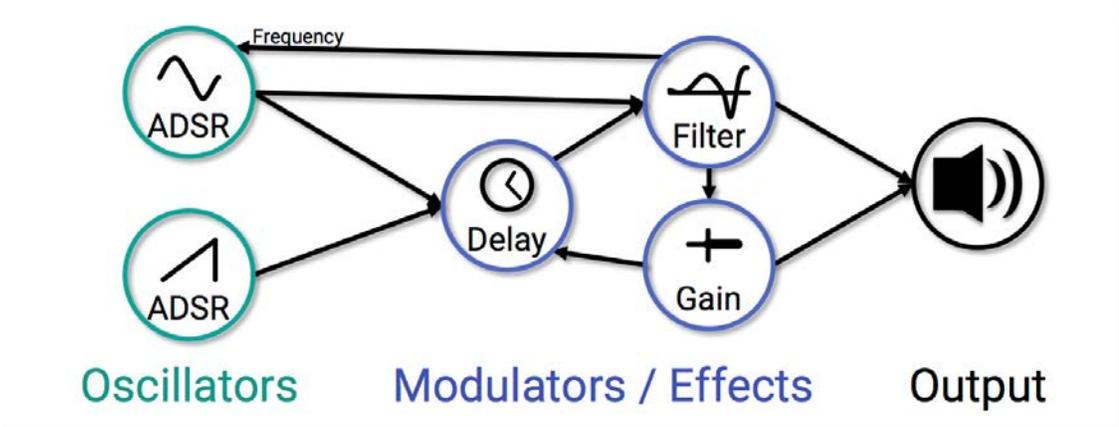
The GenSynth full stack platform utilizes both client and server side technologies (Figure 14). It is primarily divided into asNEAT, an instrument evolution and audio synthesis library; asNEAT-Visualizer, a library for visualizing asNEAT instruments; and the GenSynth Web Application containing both the IEC evolver software and CIE communication layer. It then receives and sends asNEAT instrument definitions to a REST API server. This server authenticates the app's request through local or Facebook authentication and can store and load instrument definitions from its persistent MongoDB database. The web app then sends these instrument definitions to the asNEAT-Visualizer component to show the network visualization. The web app also instantiates instruments through the asNEAT library with the received instrument definitions and sends play actions to the instruments to generate audio for users to hear and to render the spectrogram visualization in the asNEAT-Visualizer component.



**Figure 14: GenSynth Platform Organization** – GenSynth is a full stack platform using both client and server side technologies.

### asNEAT Instrument Evolution and Audio Synthesis Library

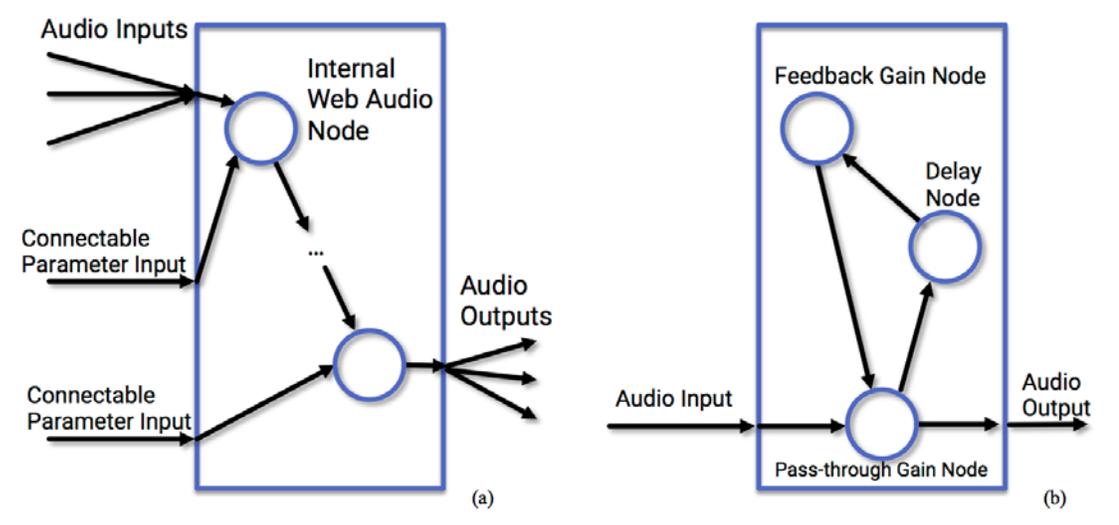
Developed for GenSynth, Audio Synthesis NeuroEvolution of Augmenting Topologies (asNEAT) is an instrument evolution and audio synthesis library utilizing the NEAT algorithm to evolve NN like synthesizer networks (SNs) that synthesize audio through the WebAudio API [Rogers 2012]. Each asNEAT node wraps one or more WebAudio nodes. An SN's input layer comprises of a set of Oscillator Nodes that generate basic waveform signals with an ADSR envelope model (**Figure 15**). These oscillators can then connect to other oscillators' frequency parameters, allowing FM synthesis, and can connect to multiple hidden layer nodes, and their varying input parameters. These hidden layer nodes combine and modify the incoming wave signals, allowing both additive and subtractive synthesis. A set of these input and hidden layer nodes are then connected together into a single Output Node which sends the final signal to be heard by the user.



**Figure 15: asNEAT SN** – A synthesizer network consisting of two oscillators, three hidden layer nodes, and an output node.

### SN Nodes

Each node within an asNEAT SN wraps WebAudio functionality and handles multiple incoming and outgoing connections (**Figure 16**). These inputs are routed internally through one or more WebAudio nodes before connecting to a single output. In addition to basic audio input, each node can have additional exposed parameters that can be connected to and controlled by an incoming signal connection for frequency modulation. Each node also contains a number of parameters that can be mutated during evolution. The varying types of nodes can be found in **Table 3**.



**Figure 16: Node Internals** – (a) A generic asNEAT Node. (b) A Feedback Delay Node that creates a diminishing echo effect.

Node Types	Description	Mutable Parameters	Connectable Parameters
Oscillator	Produces a sine, square, sawtooth, or triangle waveform signals at a set frequency utilizing an ADSR envelope.	Signal type, frequency, attack duration, decay duration, release duration, attack volume	Frequency
Note Oscillator	An Oscillator that is clamped to a specific note frequency based on the tempered audio scale with a base A4 set to 440 Hz. A step distance away from A4 is stored in addition to an offset parameter. This is useful for connecting a virtual or physical MIDI device and producing note harmonies.	Signal type, step, offset, attack duration, decay duration, release duration, attack volume	Frequency
Output	Combines all input signals and sends it to be played by the speakers. All instruments' output nodes share the same NEAT innovation id, allowing any instrument to be crossed over with another since all instruments, even when evolved separately, have this one node in common.		Audio
Compressor	A dynamic audio compressor that increases or decreases the amplitude of an input signal when raised above or below a given threshold to help maintain a more consistent audio level.	Threshold, knee, ratio, reduction, attack, release	Audio
Convolver	Applies a convolution to the incoming signal to simulate a physical location's reverb attributes based on a given waveworm recorded at that location. Currently limited to only a noise waveform.		Audio
Delay	Delays the incoming signal by a given time before outputting it.	Delay time	Audio

**Table 3: asNEAT SN Nodes**

Node Types	Description	Mutable Parameters	Connectable Parameters
Feedback Delay	A delay node with an internal gain loop. The input signal is connected to a pass-through gain node that connects to both the output and a delay node. This delay node then connects to a feedback gain node which reduces the volume of the signal based on a set ratio and passes it back through to the pass-through gain, creating a diminishing echo effect. This effect can be evolved separately with the basic Delay and Gain Nodes but was added so the common audio effect could be achieved more quickly with a simpler SN.	Delay time, feedback ratio	Audio
Filter	Applies a low-pass, high-pass, band-pass, low-shelf, high-shelf, peaking, notch, or all-pass filter to incoming signal.	Filter type, frequency, quality factor (Q)	Audio, gain, frequency, quality factor (Q)
Gain	Multiplies the incoming signal by a given ratio. Negative values invert the signal. Although each connection internally contains a Gain Node, a separate Gain Node is allowed to be added to a SN in order to expose a Gain's ratio to be connected to for modulation.	Ratio	Audio, ratio
Panner	Simulates locational audio effects between an audio source and listener in 3D space. Although this type of node exists in the asNEAT library, it is not enabled during evolution due to current WebAudio API limitations. Currently, a WebAudio Panner Node does not allow FM connections to its x, y, z parameters for automated instrument panning, allowing for rotary speaker effects. With just static parameters, a Panner Node could allow separate left and right audio channels to be evolved; however, it's more likely to imbalance the volume between stereo speakers and make an instrument sound broken. There are plans to enable the node once the WebAudio draft is corrected and browsers support the new feature or the new StereoPannerNode is implemented.	X, Y, Z	Audio

**Table 3 (continued): asNEAT SN Nodes**

### *Node Connections*

Each SN node is connected to at least one other node's audio or other exposed parameter input. Simulating a NN's connection weight, each SN connection has an internal gain node that can be mutated. Connections may be disabled during a split connection mutation and will no longer send a signal through it.

### *Instrument Mutation*

An asNEAT instrument can be mutated by splitting a connection, adding a connection, or mutating connection weights, like a NN in NEAT [Stanley and Miikkulainen 2002]. However, unlike NEAT, asNEAT provides the additional mutations of adding an oscillator and mutating node parameters. During an instrument mutation, a mutation step distance parameter determines how many mutation rounds should take place per mutation. During each round only a single mutation type takes place. During a normal mutation round, the split connection, add connection, mutate connection weights, add oscillator, and mutate node parameters have predetermined chances of being selected (Detailed in **Appendix E**). However, an instrument's topology can be frozen, only allowing mutations that do not change the instrument's topology to take place. In addition, both the connection weight and parameter mutation types are capable of accepting the same mutation step distance parameter to determine how large each mutation should be based on either a linear or exponential interpolation between given ranges (Detailed in **Appendix E**).

### *Split Connection Mutation*

During this mutation, an enabled connection is randomly selected and split with a new hidden layer node. A new connection is added between the original connection's input node and the new node with a gain ratio of 1 and between the new node and the original connection's output node with a gain ratio set to the original connection's gain ratio. The old connection is kept but is disabled.

### *Add Oscillator Mutation*

This mutation has a 50% chance of creating an Oscillator Node used for FM and a 50% chance of creating Note Oscillator Node used to generate an audible signal. When adding an FM Oscillator Node, a random node in the SN that contains connectable parameters is selected. A connection is then made between the new node and a randomly selected connectable parameter with a random weight between a min and max value defined for that connectable parameter. When adding a Note Oscillator Node, a non Oscillator or Note Oscillator node within the SN is randomly selected and connected to with a random weight between 0.1 and 1.

### *Add Connection Mutation*

When adding a new connection mutation, the SN has a 50% chance of creating a basic audio connection and a 50% chance of adding a FM connection. For both types, a list of all possible new connections is built and one is randomly selected and added to the SN. A possible new connection for FM is one that does not have an Output Node for input, the output has connectable parameters, the input and output nodes are not the same, and a connection does not already exist between the two nodes. A possible new basic audio

connection has the same limitations as a new FM connection, except that instead of requiring the output to have connectable parameters, the output cannot be an Oscillator or Note Oscillator Node.

#### *Connections Weight Mutation*

During a connections weight mutation, each connection within an SN has a chance, based on the mutation step distance, to mutate its gain ratio and mutates it either by a small delta or randomly assigning a new weight within a given range. If no connections were mutated based on the determined chance value, a randomly selected connection is forced to mutate.

#### *Node Parameters Mutation*

This mutation is similar to a connection weight mutation. All nodes within a SN have a determined chance to mutate, and if none are mutated, a randomly selected node is mutated. For each node that is mutated, each mutable parameter then has a determined chance to mutate either by a delta value or to a new random value determined by the mutation step distance and min and max ranges for that parameter. If the node has mutable parameters and none of them are mutated, a randomly selected parameter is then forced to mutate.

#### *Instrument Breeding*

The SNs that compose each instrument are bred together as defined by the NEAT algorithm for NNs. For breeding two selected SNs, first a new SN is created from deep cloning the first SN. Then, all the nodes and connections from the second SN are

iterated through and added to the new SN if an element with the same innovation ID does not already exist in the new SN. If there already exists an element with the same innovation ID in the new SN, half of the time the element added from the first SN is swapped with the element from the second and the other half of the time, the element is left alone.

### *JSON Serialization*

SNs can be serialized into and de-serialized from JavaScript Object Notation (JSON) strings. This serialization allows GenSynth and future applications using the asNEAT library to save, load, and transfer instruments.

### **asNEAT-Visualizer**

The asNEAT-Visualizer library is a set of widgets for visualizing an asNEAT SN. Although there are a variety of test visualizations in the library, only the Instrument Visualization is fully developed and used within GenSynth and is a combination of the spectrogram and force network visualizations.

One of the enticing properties of evolving visual art is the immediacy in which the art jumps out at the viewer. One can visit a site like Picbreeder [Secretan et al. 2011] and have an immediate emotional reaction to the wide variety of images brimming on the front page. However, an immediate overload of sounds blaring out to users when they first visit an application would be jarring. The problem, then, is how to prompt intrigue visually for a synthetic musical instrument in a welcoming way. Not only does this visualization need to promote user interest in the software but also needs to be representative of the aural nature of the evolving SN. To complicate matters, an

instrument has both on and off states that require visualizations (**Figure 17**). An instrument is in an off state before it makes any sound and the user interacts with it. When a user plays the instrument, it changes into the on state and produces sound. In addition, since instruments will be mutated and merged together, the visualization should be able to show how the instruments are differentiated.



**Figure 17: Instrument Off/On States** –The left image shows an instrument’s off. When it changes to the on state (middle and right), the visualization shows the newly generated sound.

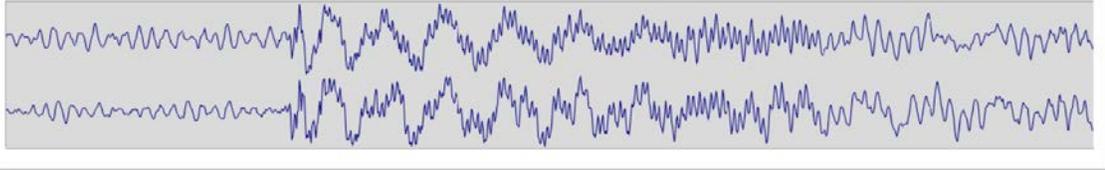
### *Audio Visualizations*

The three main types of audio visualizations are the oscilloscope, spectrum, and spectrogram visualizations. Since the instrument is made of an SN, a force graph visualization is possible that shows how the nodes within the SN are connected.

### *Oscilloscope*

One of the most used audio visualizations, the oscilloscope, displays the actual signal being sent to the speakers (**Figure 18**). It can be used in both on and off states of an instrument. For an off state, a segment of audio that an instrument would play (say a second of playing a c4 note) can be displayed in a single image. When the instrument is being played, the visualization can be scrolled, filling in the new blank section with an

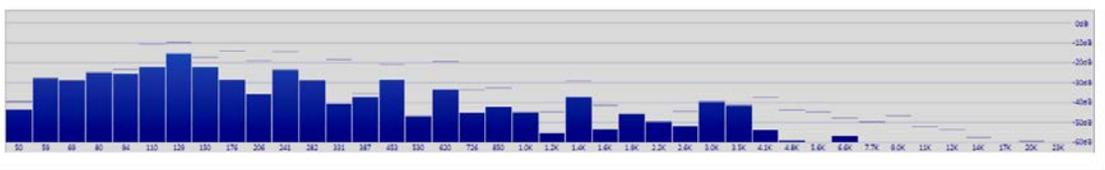
updated portion of the visualization. Although it represents the left and right speakers' positions, the visualization gives little detail to novice users, or even expert musicians, what the instrument actually sounds like.



**Figure 18: Oscilloscope Visualization [Foobar2000 2014]**

### *Spectrum*

A spectrum visualization shows the intensity of all the frequencies currently playing (**Figure 19**). The audio spectrum is subdivided into discrete waveform buckets using a fast Fourier transform and the amplitude of each subdivided waveform is shown as the height of a bar in the visualization. Although averages of a segment of audio can be shown when an instrument is in the off state, this visualization is more useful to show audio in motion. This gives a bit more information to users about what frequencies an instrument utilizes but still does not give that initial emotional response to an instrument that GenSynth needs to intrigue users.



**Figure 19: Spectrum Visualization [Foobar2000 2014]**

### *Spectrogram*

A spectrogram is the combination of spectrum visualizations over time (**Figure 20**).

The visualization not only depicts the amplitudes of the audible frequencies but also relates these amplitudes across time so that distinct aural patterns emerge. Instead of

utilizing the x and y axes for frequency and amplitude like the spectrum visualization, the x-axis defines time, the y-axis defines frequency. The brightness or color of a point

in the visualization shows the amplitude at a given time and frequency. This

visualization is excellent for showing both an instrument in the off state, by simulating

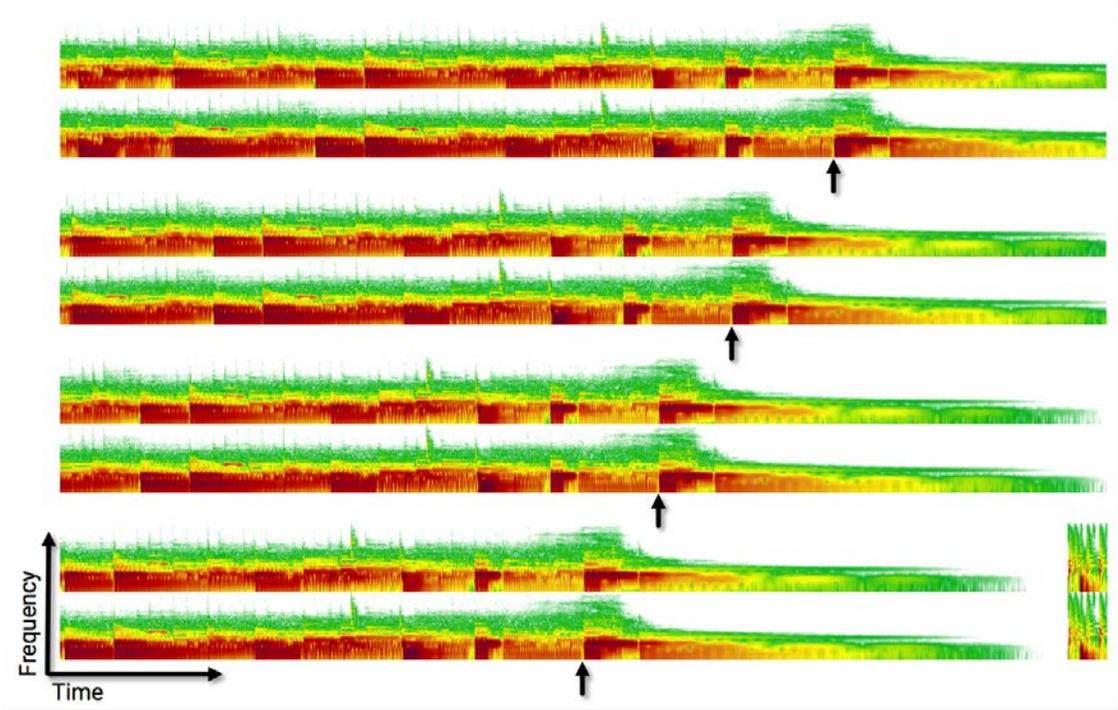
and visualizing a particular note across a full ADSR cycle, and an active on state by

treating the spectrogram as a piece of tape being rolled across the screen with new slices

attaching to one end. Although still a step away from actually listening to an instrument,

a spectrogram can visualize the patterns that an instrument produces over time and in

what frequency ranges.

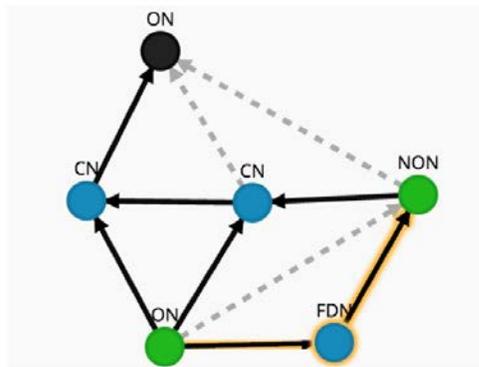


**Figure 20: Spectrogram Visualization** – Four snapshots of this visualization over time. The arrow tracks a key features of the audio as the visualization moves left over time. Spectrograms rendered using [Foobar2000 2014].

### *Force Graph*

This visualization shows directed graphs in an interactive way. Since each instrument is composed of a SN, GenSynth has the opportunity to display the underlying structure of the instrument. A user can see what nodes are producing the base frequencies and how sound travels (via connection arrows), combines, and subtracts through various effects until it finally exits the output node (**Figure 21**). GenSynth uses green nodes to represent oscillators, blue nodes for filter and effect nodes, and the black node is the output node. Gray dashed connections are disabled and highlighted elements are those that just recently changed during evolution. Users can drag nodes around as each tries to stay a set distance from its connected neighbors. Even though a network of audio nodes

is far from an audible sound, the visualization gives insight into how the networks are actually evolving by highlighting what has changed and how the audio is generated.

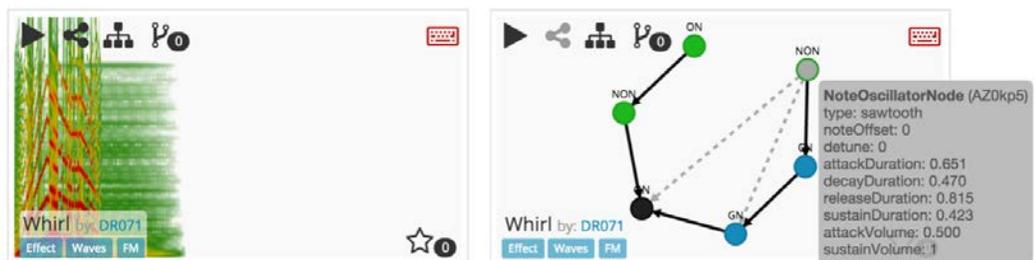


**Figure 21: Force Graph Visualization**

### *Instrument Visualization*

In GenSynth, an Instrument is represented by an Instrument Visualization (**Figure 22**).

This visualization is a combination of the spectrogram and force graph representations as users are allowed to toggle between the two types depending on their need. When hovering over an element in the force graph visualization, a dialog shows more detailed parameters for that element.



**Figure 22: GenSynth Instrument Visualization** – This visualization combines both spectrogram (left) and force graph (right) visualizations into a single component.

## **GenSynth Web Application**

The GenSynth web application is a front facing Ember app that provides an interface connecting the other GenSynth components for users to discover, explore, and share instruments. GenSynth users discover published instruments created by previous users and show their interest by starring or branching off the ones they like to continue their evolution. In addition, users can evolve instruments from scratch, exploring the instrument space from basic two node instruments. When users are done evolving a set of instruments, they can then save, name, tag and publish the instruments for future users to branch. At any location within the web app, an instrument can be selected for auditioning via a play button, the onscreen virtual piano that can be controlled through hotkey presses, or a physical MIDI device connected through WebMIDI. In addition, since there is the potential for evolving harsh sounding instruments or instruments with an escalating feedback loop, an accessible “Panic!” button and spacebar hotkey is provided that immediately cuts all current audio. In addition to both asNEAT and asNEAT-Visualizer, the web application was developed for and enhanced by the GenSynth usability studies.

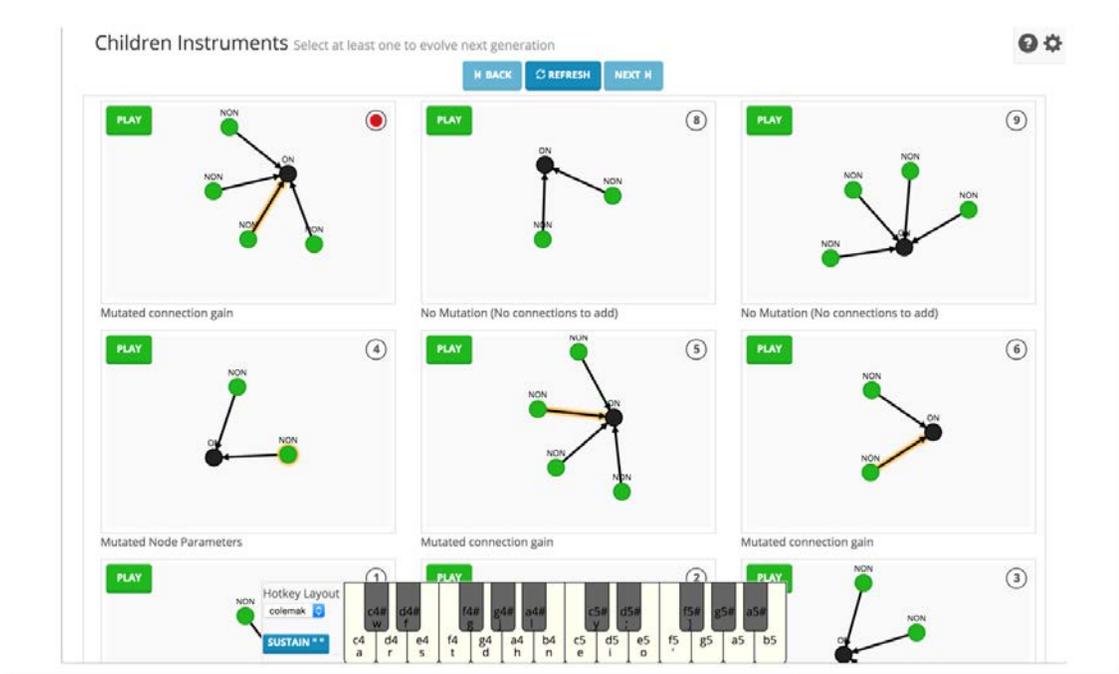
## Chapter 4: Methodology

To determine in what ways software can be developed for collaboratively evolving novel synthetic musical instruments, GenSynth was designed and developed to meet the discovery, exploration, and sharing artificial platform goals and six Picbreeder CIE challenges described earlier. In order to increase GenSynth's usability and provide necessary functionality, an initial IRB approved (#4281) usability study was run focusing on the IEC software for interactively exploring the SMI space through evolution. After taking the first study's feedback into account, a second study added the CIE layer in order to meet both the discovery and sharing goals of the evolutionary platform and the six CIE challenges presented by Secretan et al. [Secretan et al. 2011].

### GenSynth Exploration Usability Study

The initial usability study focused on the interactive software for evolving SMIs and was designed to determine in what ways SMI evolutionary software can be designed and possible directions for user interface improvements to improve clarity for users in the second study. Locally recruited participants were set down at a table with the IEC website preloaded on a laptop with an attached Alesis QX25 MIDI keyboard. After a brief introduction, participants were asked to complete various tasks with minimal help while orating their thought process (Complete procedures listed in **Appendix B**). The software for this first study consisted of a single web page that presented users with nine randomly generated instruments (**Figure 23**). The tasks included finding and reading the help instructions; playing instruments with the play button, virtual piano, and MIDI keyboard; selecting instruments; and traversing forward and backward

through instrument generations. When done, participants then filled out a final questionnaire about using the software (Listed in **Appendix C**). During the study, the participants' screens were recorded for later review and I watched while making notes on specific aspects of each task. The results and feedback in the study were then taken into account for enhancements for the second study.



**Figure 23: First Usability Study IEC Software**

### GenSynth Collaborative Platform Usability Study

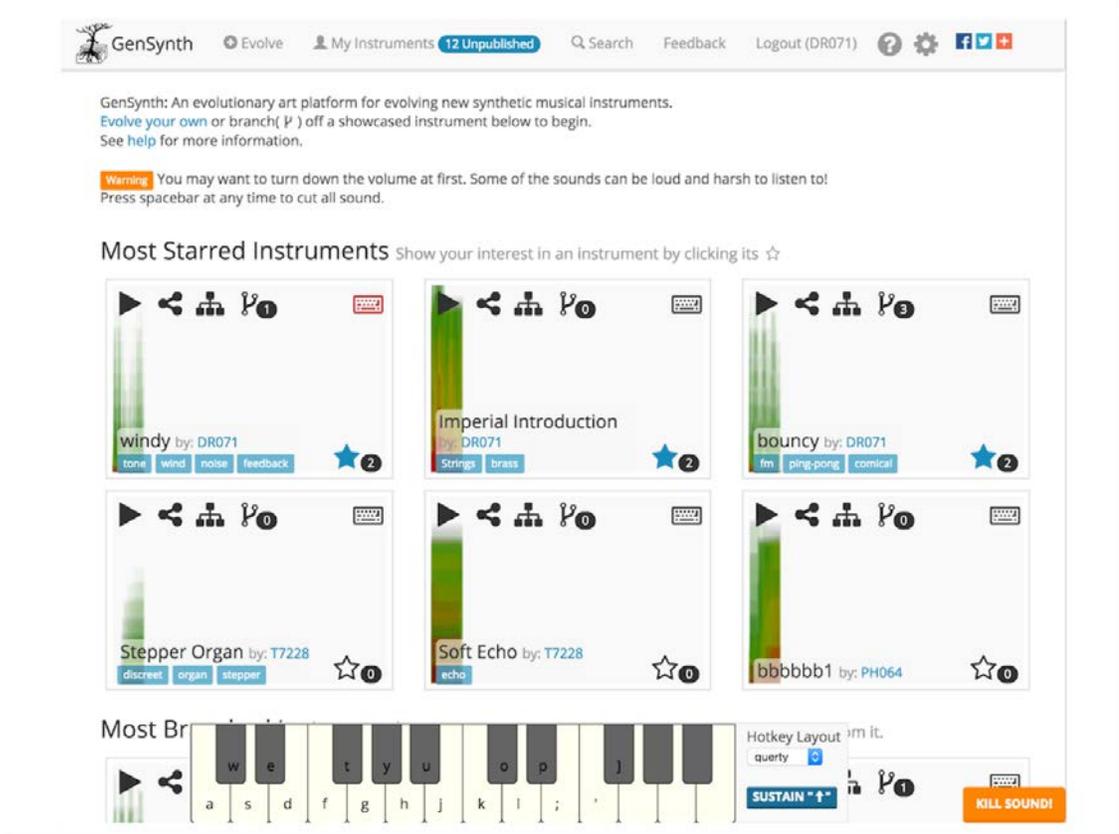
After refining the exploration process of the IEC software in the first study, the second usability study included the collaboration layer into the platform to meet the six CIE challenges and establish how such a platform can enhance the discovery, explore and sharing goals of an evolutionary aural tool for evolving SMIs. The application was publicly hosted online at <https://gensynth.ou.edu> (now accessible from [gensynth.io](https://gensynth.io))

from 2014/10/20 to 2014/03/05 for users to engage in the collaborative process of evolving instruments from anywhere in the world. Users provided qualitative feedback through a quick comment box on the website and an external survey linked to from the site (Listed in **Appendix D**). Quantitative feedback based on user interactions was also tracked with analytics software built into the platform in addition to quantitative fields in the external survey.

### *Three Platform Goals*

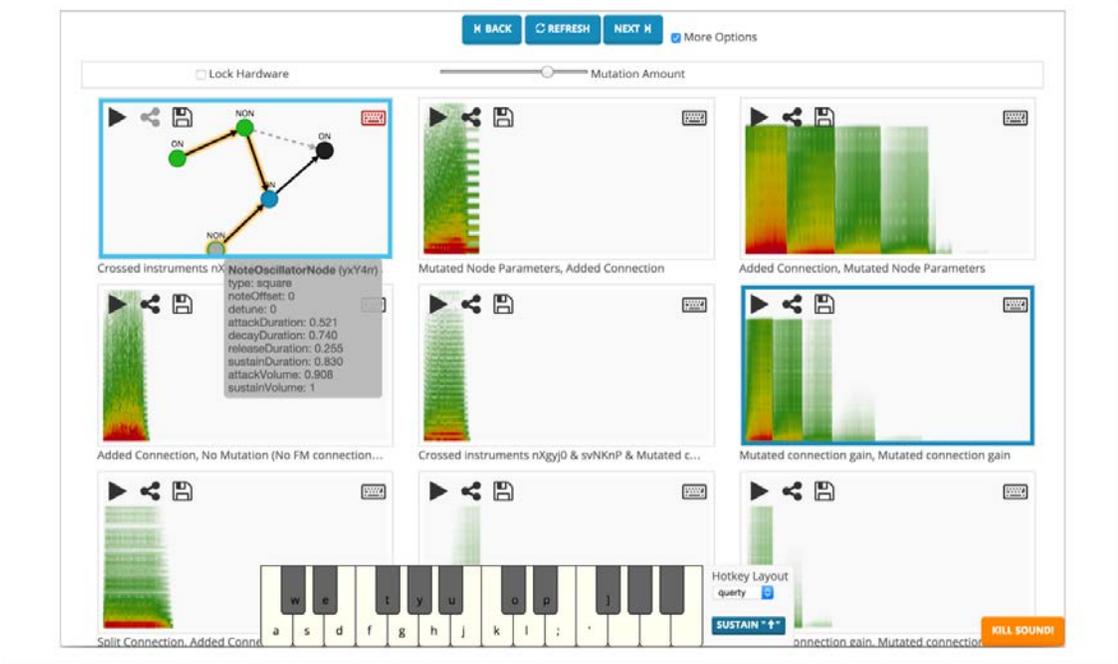
The first method for demonstrating how GenSynth evolves digital instruments is through the discovery, exploration, and sharing goals.

Users discover instruments primarily through showcases on the front page similar to Picbreeder's showcases [Secretan et al. 2011] and searching for instruments by name or tags. The front page is divided into most starred, most branched, newest, and random instrument showcases (**Figure 24**). To tell if this goal is met, users must be shown to be visiting and interacting with the front and search pages.



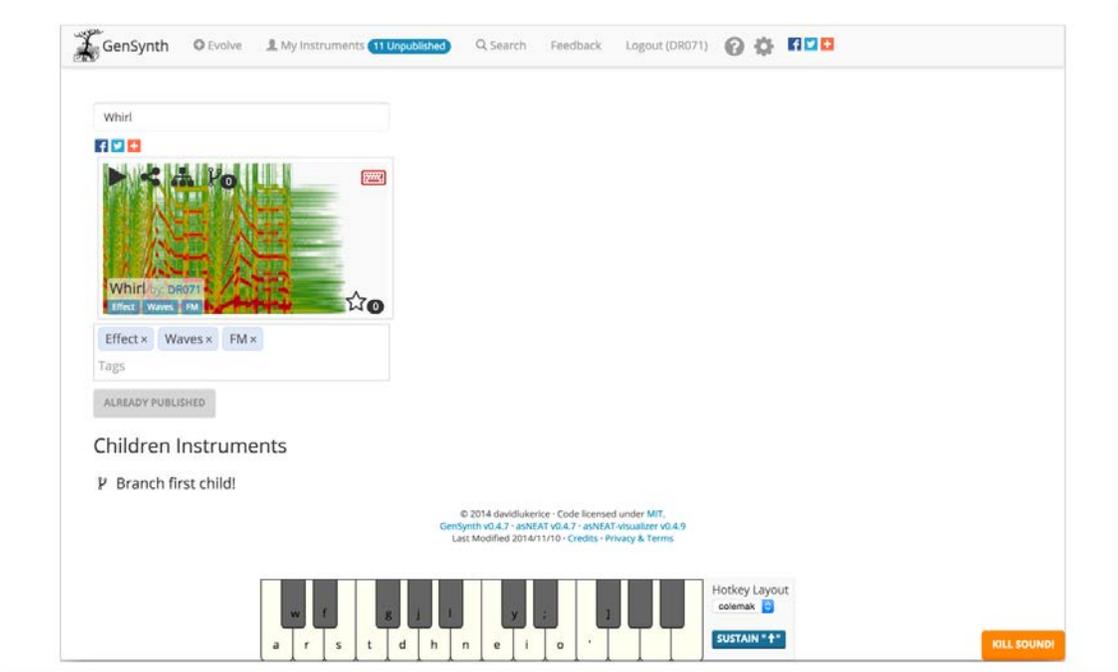
**Figure 24: GenSynth Front Page**

Users explore the instrument space through the IEC Evolve page (**Figure 25**). The page presents the user with a set of nine new instruments or nine instruments evolved from those selected from a previous generation or the branched from instrument. Users can experience the visualized instruments by either playing a single note with the play button or selecting the instrument to be controlled through the virtual onscreen keyboard that reacts to keyboard hotkeys or through a physical device connected through WebMIDI. When one or more instruments are selected, a new generation of instruments can be evolved from the selected instruments. To tell that this goal is met, it must be shown that the evolve page is visited and utilized.



**Figure 25: GenSynth Evolve Page**

Users share the instruments they evolved during exploration by saving and publishing them with a given name and set of tags. Once saved from the Explore page, an instrument is randomly assigned a name. Its creator can then change the instrument's name and publish it on the instrument page. The creator can also add descriptive tags to aid in its discovery by future users (**Figure 26**). For this goal to be met, it must be shown that evolved and branched instruments were both saved and published.



**Figure 26: GenSynth Instrument Page**

*Six CIE Challenges*

The second method for demonstrating how GenSynth evolves digital instruments is through the six CIE challenges presented by Picbreeder. The subsequent sections describe what features of GenSynth were implemented to meet each challenge, followed by what analytics are used to show if they were successful.

*CIE Challenge 1. Empower Users Regardless of Skill Level*

Empowering users to evolve artistic works regardless of their skill level is the first challenge. In order to help novice users, the front page provides the discovery of good preset instruments. These instruments have been declared as better than others through the platform’s rating system, and they allow for an initial starting point for those less experienced in instrument design by allowing these instruments to be branched off of.

In addition, the exploration cycle of generating instruments, auditioning, selecting, and generating the next population enables both advanced and novice users to explore interesting instruments without knowing having to know audio synthesis. The advanced features of locking a current instrument's hardware topology and changing how much instruments evolve from generation to generation give users a bit more control while hiding the complexity of instrument design. In order to show that this challenge is met, the instruments in the front page showcases should be utilized and branched from, the more novice oriented play button should also be used, and the external survey should show that both those with and without musical experience found the platform understandable and usable.

### *CIE Challenge 2. Overcome Single User Fatigue*

The second challenge is overcoming single user fatigue and reducing the fatigue of each generation step. GenSynth allows users to pick up where previous users left off through publishing and branching. To reduce fatigue during evolution, each instrument only requires simple selected or not selected ratings for individuals, and a single note play button is provided to allow quick evaluation. Not having complicated instrument ratings lowers the mental drain of users. The asNEAT library itself also works to reduce fatigue by utilizing a variety of oscillator types and more abstract hidden layer nodes than simple multiplication convolution. This reduces the complexity, and therefore steps required to evolve it, of the internal SNs to represent standard audio themes. To show that this challenge is met, a baseline fatigue wall needs to be established for the average number of generation steps taken per user session. It then needs to be shown that

instruments are branched and saved and that the branched instruments on average have a generation count greater than the found fatigue wall.

### *CIE Challenge 3. Create Diverse Content with Many Users*

The third challenge is allowing many users to create lots of diverse content. Unlike some platforms that enable platform wide collaboration on only a single generation of individuals at a time, limiting the amount of content that can be produced, GenSynth inherently allows single user exploration. This allows each user to evolve their own set of instruments, generating more content than a large group of users evolving a single generation at a time. In addition, users should be able to create diverse content. If multiple individuals all evolve similar instruments, there is less reason for users to evolve them separately. To show that this challenge is successfully met, users need to be shown to evolve instruments. These instruments then should contain a variety of different nodes. The composition of the instrument, its genotype, greatly influences the sounds it produces, its phenotype. Although there is the small chance that multiple SNs produce similar sounds, showing a variety of genotypes and informally verifying through listening is sufficient. Utilizing the sum of squares error over the FFTs generated for the spectrogram visualization for each instrument or detected sound features would make the difference testing more rigorous but is left for future study.

### *CIE Challenge 4. Enable Collaboration with Individual Choice*

Closely related to the third challenge, the fourth challenge is enabling collaboration without diluting individual choice. Some platforms enable platform wide collaboration on only a single generation of individuals at a time, limiting the amount of content that

can be produced. GenSynth was designed to allow single users to explore. Therefore, simply showing instruments are evolved, published, and branched is enough to show this challenge is met.

#### *CIE Challenge 5. Encourage Participation*

The fifth challenge of a CIE platform is encouraging participation. GenSynth was designed to have an intuitive interactive UI and benefited from the first usability study. When first visiting a page on the platform, visually intriguing colorful spectrograms are rendered for each instrument on the front page and each continuously scrolls when being played with the four different provided methods. The counterpart force graph visualization can be toggled on for each instrument, allowing users to interactively explore the instrument's topology by zooming, panning, and moving around each node and viewing the node's current parameters. To show this challenge to be met, there should be user participation on the platform and the external survey should report that GenSynth is engaging.

#### *CIE Challenge 6. Balance Work Exploration & Exploitation*

Finally, the sixth and last challenge is balancing exploitation of popular works and exploration of unpopular ones. GenSynth exploits popular instruments by showcasing the best rated and most branched instruments on the front page. This is then balanced by including the newest and random showcases in addition to allowing all published instruments to be searched for, regardless of popularity. To verify that this balance exists, it should be shown that all the showcases are branched from comparable amounts of times and the search page is utilized.

## Chapter 5: Results & Discussion

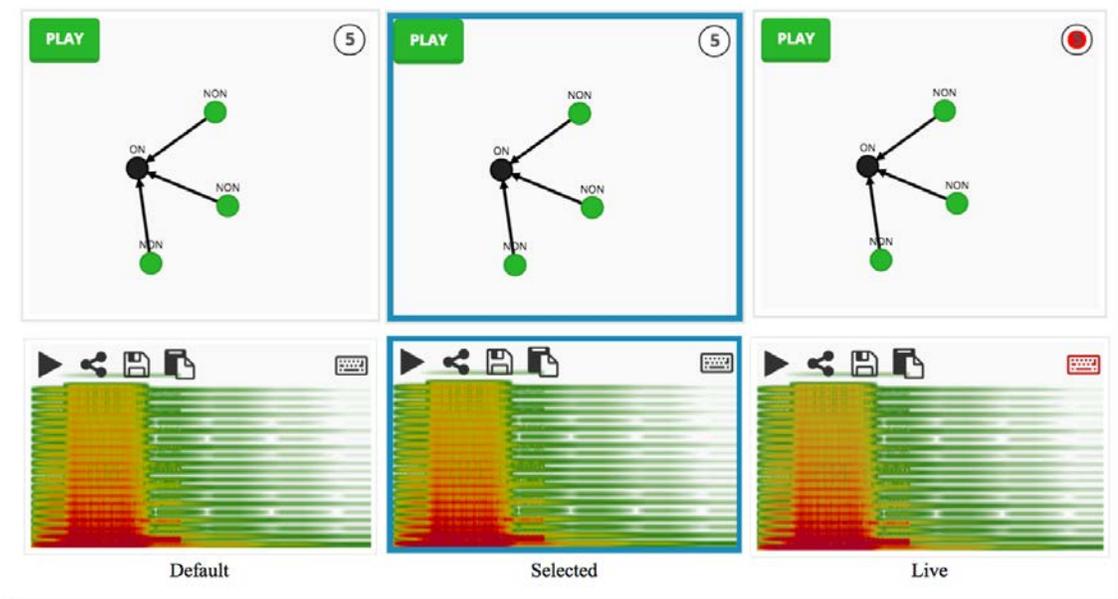
GenSynth was demonstrated to allow instruments to be discovered, evolved, and shared. In addition, the platform demonstrated its ability to meet four of the six Picbreeder CIE challenges, with the first challenge requiring more data for validation and the sixth challenge shown to not be met with the current Picbreeder style platform. The initial usability study provided many UI improvements and helped focus development for GenSynth.

### *Usability Study Results*

The first study focused on the usability of the evolution software and consisted of three university college students in varying age groups and music proficiencies. The study highlighted several areas of confusion in the application and helped prioritize the most desired features to be implemented in the final platform. The main areas of confusion were the selected versus live dichotomy, what the network visualization represented, and what the software was doing overall.

Participants had difficulty understanding the difference between selecting an instrument to be used as a parent for the next generation and selecting an instrument to be played with the virtual and MIDI keyboards. The component utilized imagery of a recording icon found in many recording applications. This icon consisted of a black circle that is filled in red when activated (**Figure 27**). However, even the participant experienced in recording and mixing audio was confused with this interface. To resolve this in the final application, a keyboard icon that is red when selected and black when not was used to directly relate the live instrument state and the instrument that is being auditioned. In

addition, a descriptive tooltip was added when hovering over any of the icons on the instrument to help explain their uses.

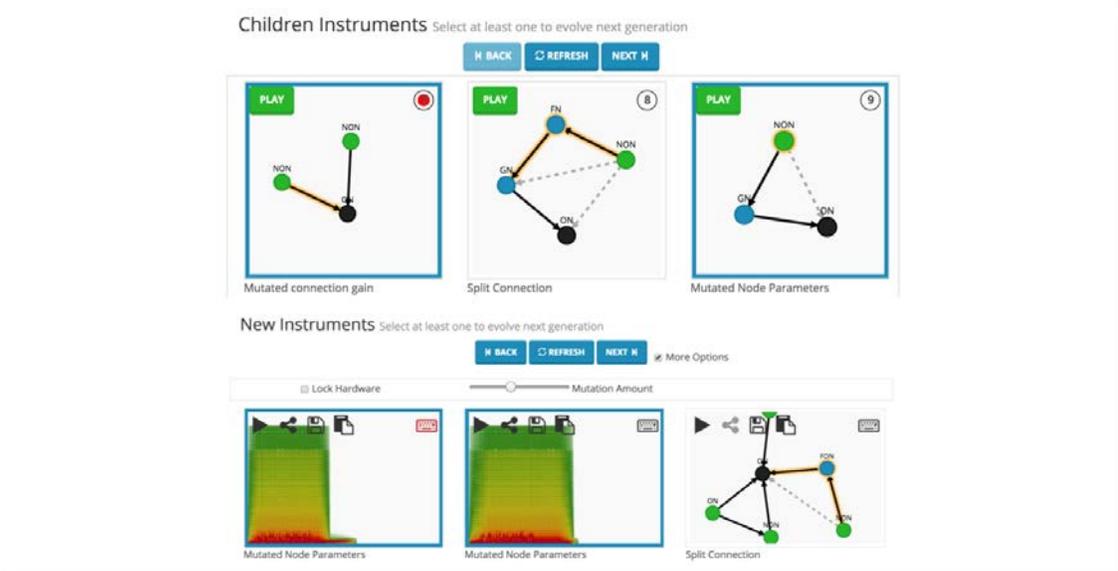


**Figure 27: Selected Vs Live - (Top) Initial Study. (Bottom) GenSynth.**

The second area of confusion was the network visualization. In the study, each instrument was represented solely by a network visualization (Top left in **Figure 27**). The network visualization is abstractly distant from what sounds it generates, so participants not familiar with audio synthesis found it difficult to understand what it represented. Participants either did not understand what the network graph meant or believed that interacting with the visualization would affect the sounds it generated, which is not the case. This called for a more engaging non-interactive visualization closer related to its generated sound to be presented first. For GenSynth, a spectrogram was selected to be the first visualization shown to the user with a toggle button to show the network graph (Bottom left in **Figure 27**). This allows the additional benefit of

having help text on this icon to describe what it is and it does not change the instruments sound.

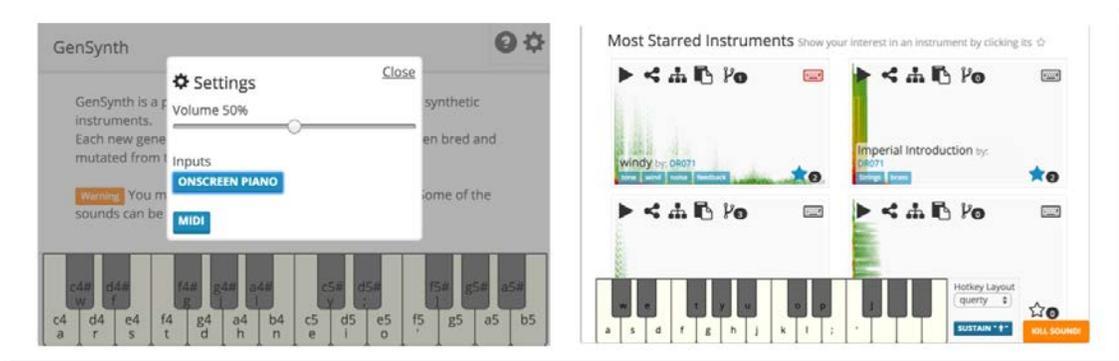
Finally, users found it difficult to understand the overall process of evolving instruments. In the study, users had very little control over the evolutionary process other than selecting instruments and mutation explanations were always listed under each instrument (Top of **Figure 28**). Once participants began to grasp how to select an instrument, they quickly understood how to create new instruments, refresh the current set, and go back to previous ones. However, participants were unsure how the actual evolutionary process was working and what the parent and child symbolism meant in the context of the application. Hiding the specifics of the evolutionary process was done intentionally to make it easier for users unfamiliar with the concepts to utilize the platform. However, as noted by one participant, exposing some of the underlying properties of the evolution behind an advanced options section could allow more control for those that wanted it. For GenSynth, the mutation amount, whether the topology of the network could be mutated, and whether the mutation text below each instrument was shown was placed under a more options toggle. In addition, an introduction video is provided on the front page of GenSynth to give an overview of how the platform works.



**Figure 28: More Options Toggle** – (Top) Initial Study. (Bottom) GenSynth.

The last part of the usability study gathered feedback from the participants on which additional features would be most useful. It was found that the external keyboard was most engaging to participants. This showed that it was important to make sure MIDI continued to work throughout the study as the WebMIDI implementation in browsers changed. The second most engaging way to audition the instruments was through the onscreen piano keyboard. Originally, users had to open up settings and select to show the keyboard, so it was changed to always be visible so everyone would know it existed (Figure 29). Permanently storing and having access to the instruments was another highly desired feature, and participants noted that authenticating through email and password or Facebook would be most preferred, in addition to being able to use the platform while not logged in. These two authentication methods were implemented in GenSynth. Authentication is only required when saving an instrument, allowing users to have instant access to the platform and delaying the overhead of creating an account

until it is necessary. A stop all sound button was also suggested and implemented in GenSynth (can also be quickly applied by tapping the spacebar) in case an instrument generated an infinite feedback loop. Being able to copy an instrument in a plaintext format was also requested in the initial usability study but only implemented near the end of the study due to time constraints. However, importing instruments back into GenSynth was not implemented.



**Figure 29: Onscreen Keyboard Setting - (Left) Initial Study. (Right) GenSynth.**

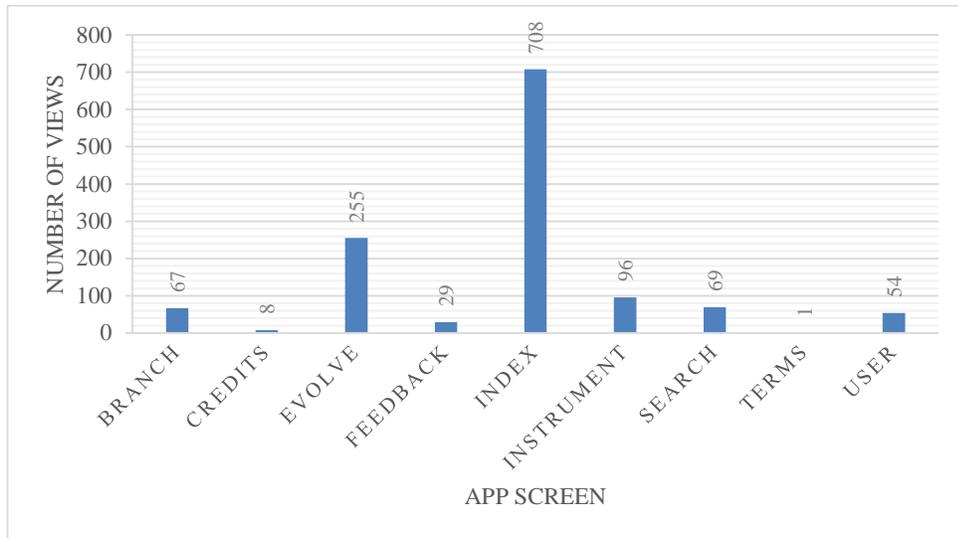
### *GenSynth Platform Study Results*

Over the course of the study, 184 distinct participants visited the site from 21 countries and 17 US states. Of those users, 12 registered for accounts, three utilizing Facebook authentication and nine authenticating with an email address and password. I personally created two accounts, one from each authentication method, in order to test functionality and seed the platform with some instruments to show up on the front page. Of the 12 users, six evolved a total of 35 instruments (nine from non-researcher users), of which 13 were published (five from non-researcher users) using 34 unique tags.

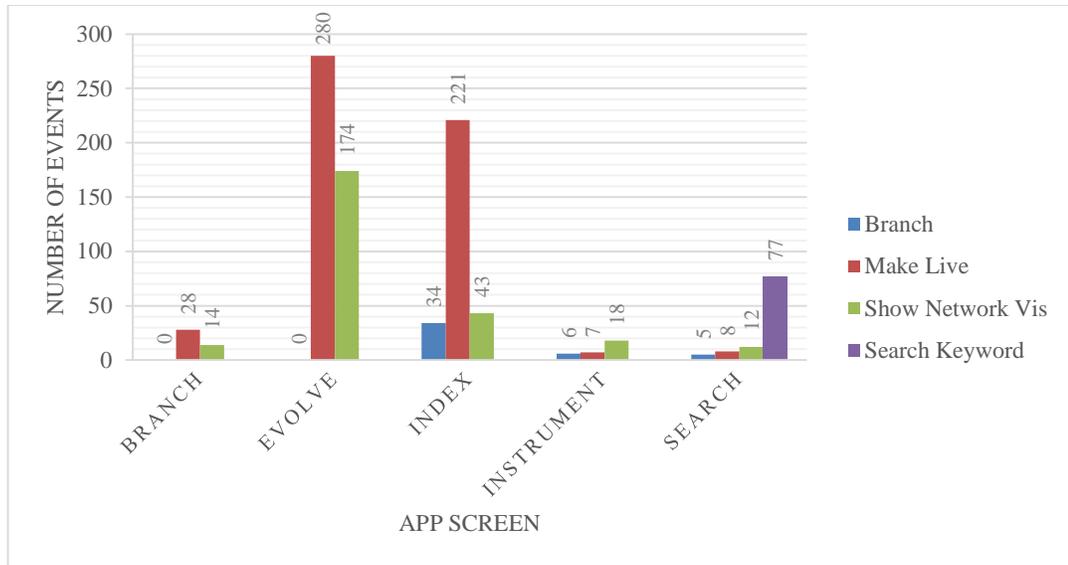
*The Discovery, Exploration, and Publish Goals*

The GenSynth platform was designed and found to aid in the **discovery, exploration,** and **sharing** processes of instrument evolution.

As discussed earlier, to tell if GenSynth meets the **discovery** goal, users must be shown to be visiting and interacting with the front and search pages. Looking at page views, both the index and search pages were accessed (**Figure 30**). When using the application, users are first presented with the index (front) page, so it would make sense it has the higher access rate of the two. Looking the interaction on each page, it can be seen that both the front page and search pages were utilized, with the greater amount of interaction on the front page (**Figure 31**). This shows that the discovery goal was met.

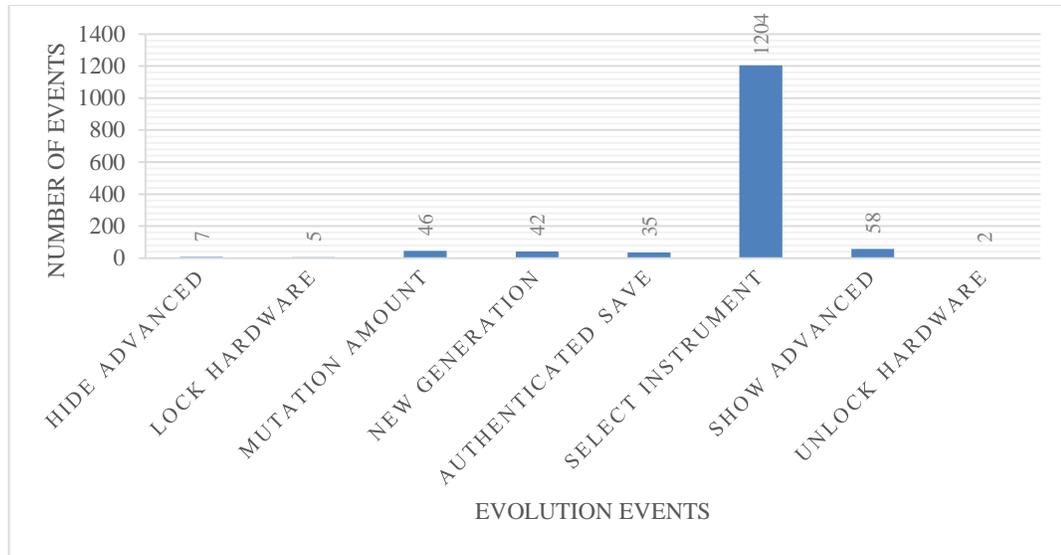


**Figure 30: Views per App Screen**

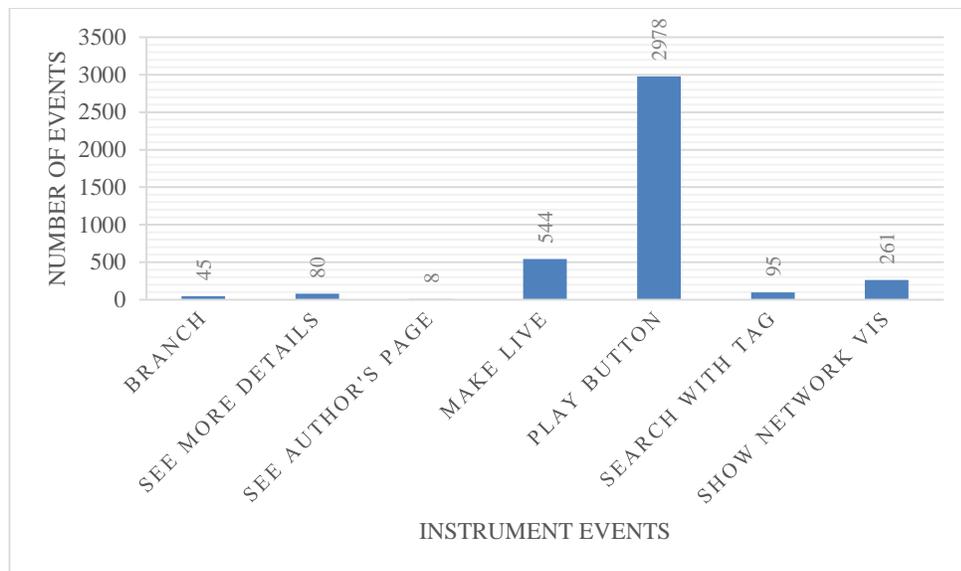


**Figure 31: Events on Each App Screen**

To determine if the **exploration** goal is met, it must be shown that the evolve page is visited and utilized. The exploration and branch screens received the second highest number of visits and the most user interaction (**Figure 30 and 31**). Looking at the tracked evolution events, it is clear that many instruments were played, selected and evolved (**Figure 32 and 33**). So, GenSynth meets the exploration goal.



**Figure 32: Tracked Evolution Events** – Events on the evolution page.

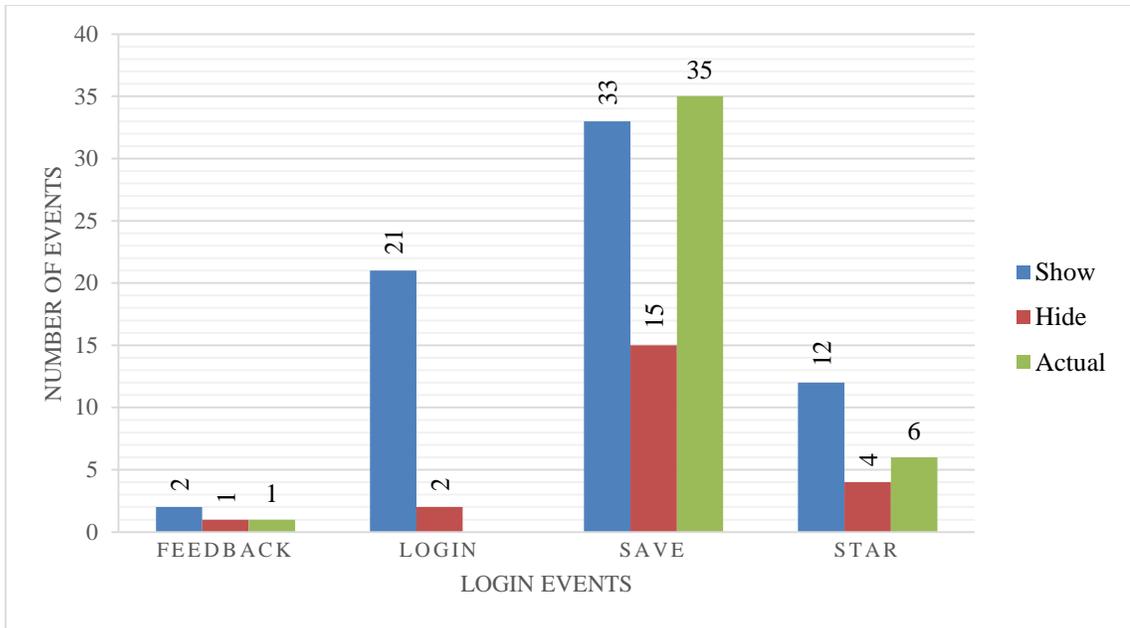


**Figure 33: Instrument Component Events** – Each instrument component contains UI elements for the listed instruments events.

To tell if the **sharing** goal was met it must be shown that evolved and branched instruments were both saved and published. As described earlier, 35 instruments were saved during evolution and 13 published, showing users were able to successfully create instruments. Saving an instrument, however, required user authentication. Users could

use the entire application without authenticating until they attempted to leave feedback in the quick comment box, save an instrument, or star an instrument. At this time they would be provided with a popup dialog that allowed them to either create a new account or login with an existing one with either a Facebook account or email address and password. After completing authentication, they could then perform the action again. Unless the user specifically clicked the login button on the top the page, more than 50% of the time they decided to hide the login popup and not authenticate (**Figure 34**). This means that for at least half of the unauthenticated users that wanted to save an instrument, the barrier for authentication was still higher than a perceived benefit. Additional forms of authentication, such as allowing an account to be created with an unidentifiable username and password, or providing more uses for saved instruments may allow more users to use the system.

Once users authenticated and saved an instrument though, there was only a 37% chance that a user would then publish it for others to use. An unpublished badge was added to the top of the page telling users how many instruments they had saved but not yet published, but was only clicked on 8 times. To allow more content to be exposed to users on the site, saved instruments could automatically be published with a generated name or the publish UI could be shown directly after saving the instrument. Yet, instruments were both saved and published, so GenSynth meets the final platform goal.



**Figure 34: Number of Showing or Hiding Login Events** – The number of times a user was prompted with a login popup, how many times they hid it without authenticating, and the number of times feedback was actually left or an instrument saved or starred after authenticating.

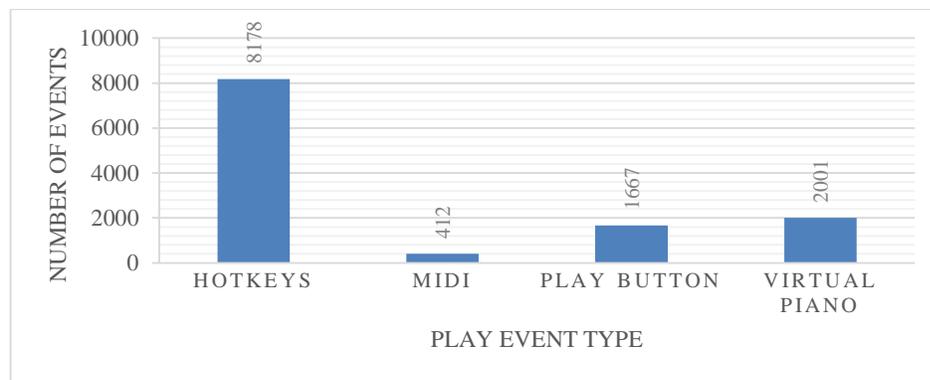
### *Meeting Six CIE Challenges*

The collaborative platform was developed to meet the previously described six CIE challenges presented by Picbreeder. It was found that four of six were met, with one needing additional data, and one not met.

#### *Meeting CIE Challenge 1. Empower Users Regardless of Skill Level*

Empowering users to evolve artistic works regardless of their skill level is the first challenge. This is sufficiently shown if the showcases and play button are used and the external survey show the platform is understandable for a wide range of users. A majority of the branching and make live events happened on the front page (**Figure 31**). This means that a good amount of user interaction happened where novice and advanced users alike could engage with the instruments. Similarly, users interacted with

instruments utilizing the play button that requires no prior experience, the virtual piano that benefits from knowing how to play a piano, and the more advanced hotkey and MIDI setups (**Figure 35**). This means that instrument engagement was not limited to an interaction method that catered to a single skill level. In order to help new users, a help menu was provided and an introductory video was added on 2015/02/02, which was played 29 times. Furthermore, the external survey was supposed to help collect information on how those with varying experience and skill levels, but there was only a single survey submission. The participant self-reported to have some experience with creating music and did not find the site confusing. Although the tracked site interaction and the survey response point in the direction of GenSynth meeting the first challenge, more data would be required to verify if this goal was actually met.



**Figure 35: Play events based on the number of connected MIDI devices**

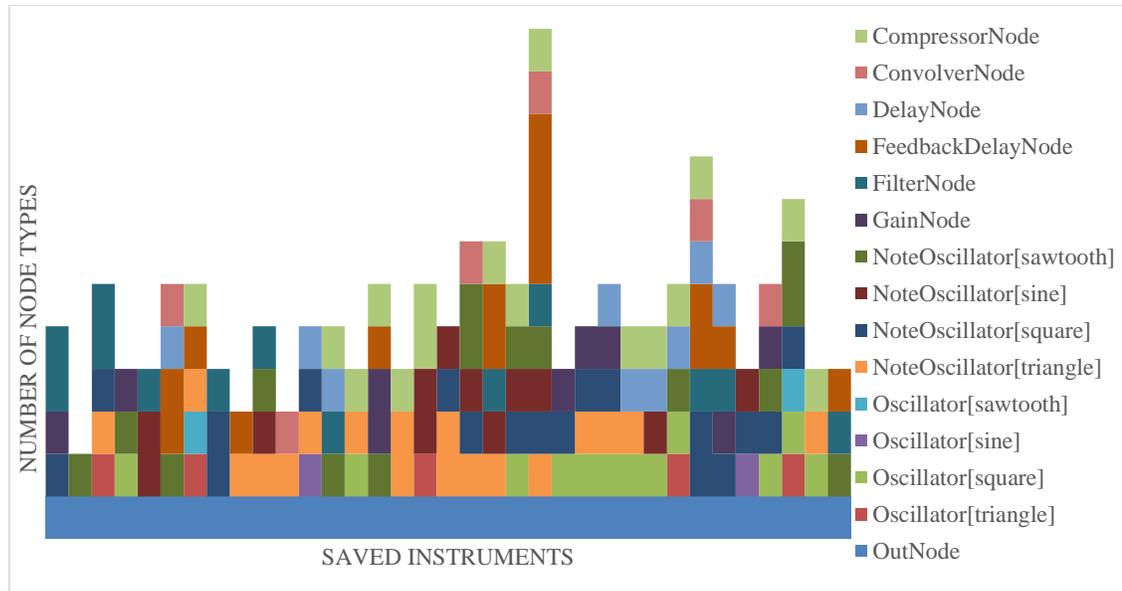
*Meeting CIE Challenge 2. Overcome Single User Fatigue*

Verifying that the platform meets the second challenge of overcoming single user fatigue requires branched instruments to have on average more generations than the baseline fatigue wall. In total, users made 42 generation steps on the platform. For users

that actually took at least one generation step on the evolve or branch screens, this averaged to a fatigue wall of about 3.82 generation steps per evolving session. This number of steps allows for very little space exploration and refinement, so a saving and branching mechanism is shown to be helpful. The 45 branch events (**Figure 31**) resulted in 6 saved children with an average of 5.5 generation steps. This means that branching has allowed instruments to be evolved to that would normally not have been. This is sufficient for the second challenge to be met.

*Meeting CIE Challenge 3. Create Diverse Content with Many Users*

The third challenge is allowing many users to create lots of diverse content and is validated by showing instruments are evolved and diverse in content. It has already been shown that instruments have been successfully evolved by single users. Looking at the fingerprints of all the saved instruments (**Figure 36**), it is clear that the composition of the instruments are diverse in terms of what nodes and how many are utilized in the instruments. Informally testing the instruments phenotypes through listening to them on the platform confirms this as instruments sound very different from one another, finishing out the criteria to meet the third challenge.



**Figure 36: Saved Instrument Node Finger Prints** – This graph shows each of the saved instruments in GenSynth divided into the number of each type of nodes it contains

*Meeting CIE Challenge 4. Enable Collaboration with Individual Choice*

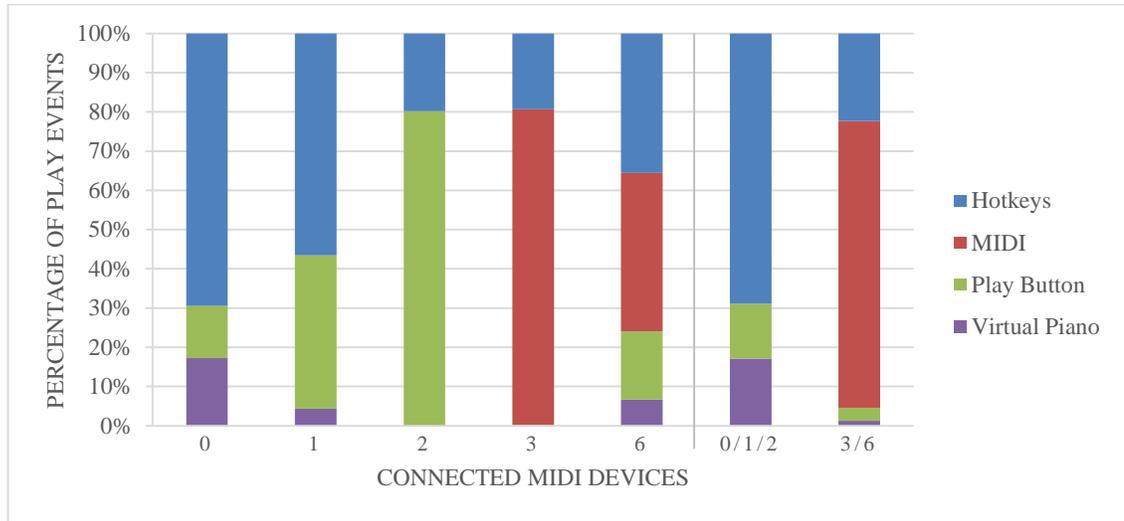
Closely related to the third challenge, the fourth challenge is enabling collaboration without diluting individual choice and is met by showing instruments having been evolved, published, and branched. This has already been sufficiently demonstrated.

*Meeting CIE Challenge 5. Encourage Participation*

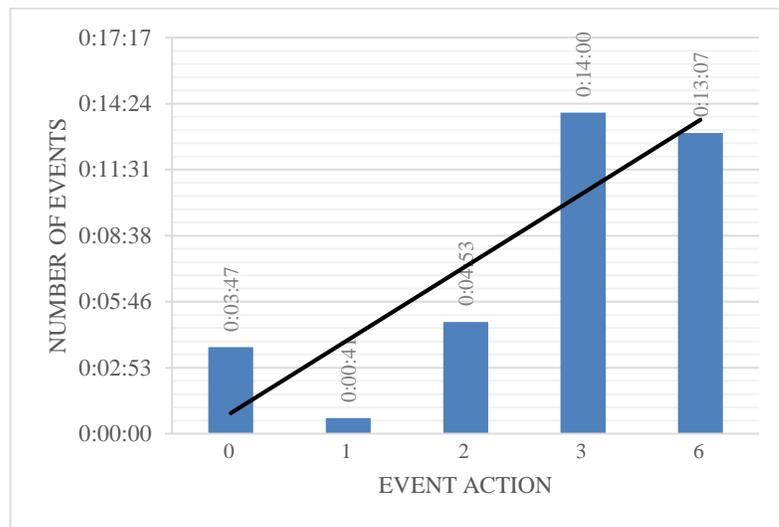
The fifth challenge of a CIE platform is encouraging participation and is met by showing participation on the platform and positive engagement results from the external survey. The usability of GenSynth was greatly improved based on the feedback of the original usability study, and the aforementioned results have shown that users have participated in evolving instruments on the platform. Yet, instrument creation is a much more niche and complex space than simple image evolution, making user traffic and interest the biggest issue of the platform. The single participant that filled out an

external survey selected that as a whole, the platform was very engaging and found the hotkeys extremely engaging. Similarly the one comment in the feedback box thought the site was “pretty cool,” but wished there was a lot more pre-made instruments, especially those emulating well known instruments, to get started with. This is a classic “chicken and egg” problem of needing users to create content but also needing content to pull in users. I seeded the site with some initial novel instruments but as a single user, personally ran into a fatigue wall. Having seed instruments that resemble real world counterparts that users could more immediately relate to could be useful. Moreover, the low number of comments left in the feedback box could be attributed to requiring authentication. This was done to reduce potential message spam, but in retrospect could have provided more qualitative feedback on why participation was low. In addition, the lack of user participation could stem from the lack of things users can actually do with the instruments once they have evolved, saved, and published. At the time of the study, there was no additional use for an instrument other than playing it. Additional internal and external tools, like the in development GenSynthPerformer, could be utilized to increase user participation [Rice and Tsoukalas 2015]. Encouraging the use of MIDI could also increase participation since it was found that users with compatible MIDI devices and browsers tended to use MIDI more than other play methods (**Figure 37**) and there was a correlation between the number of connected MIDI devices and the average time spent on the site each session (**Figure 38**). However, this needs to be verified with more users and it is not always feasible to have a MIDI device present and the WebMIDI API has yet to be fully accepted by browser vendors. So even though is

room for improvement, the interaction logged and feedback given is sufficient to have met this challenge.



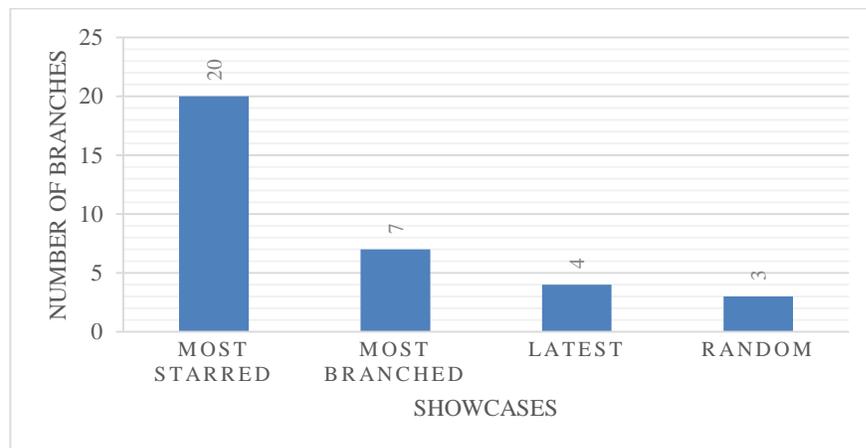
**Figure 37: Instrument Play Event Percentage Based on the Number of Connected MIDI devices** – For each number of connected midi devices and a combination of those that never used MIDI (0, 1 or 2) and those that did (3 or 6), this graph shows the percentage of the four instrument play events used.



**Figure 38: Number of Connected MIDI Devices Vs Average Session Time** – Users with more connected MIDI devices tended to stay longer on the platform.

*Meeting CIE Challenge 6. Balance Work Exploration & Exploitation*

Finally, the sixth and last challenge is balancing exploitation of popular works and exploration of unpopular ones. This would be shown by having comparable branch counts from instruments in all the showcases and the search page being utilized. It is clear that on GenSynth, and potentially other Picbreeder style platforms, exploitation of top rated artifacts dominated exploration in terms of how saved instruments are utilized. The search page was hardly visited (**Figure 31**) when compared to the front page and most branch events only happened on the most starred and most branched showcases, both of which are exploitative (**Figure 39**). So, as it stands, GenSynth does not meet the sixth CIE challenge.



**Figure 39: Number of Branches from Each Showcase**

## Chapter 6: Conclusions & Future Work

GenSynth was designed and developed to allow users to collaboratively evolve novel synthetic musical instruments and was demonstrated to allow instruments to be discovered, evolved, and shared. In addition, the platform demonstrated its ability to meet four of the six Picbreeder CIE challenges, with the first challenge requiring more data for validation and the sixth challenge shown to not be met with the current Picbreeder style platform. The asNEAT library is the first instrument evolution library that utilizes a graph genotype, NEAT crossover, multiple waveform oscillators with ADSR envelopes, hidden layer filter and composite nodes, and multiple connectable parameters per node. In addition, GenSynth is the first instrument evolution platform that fully integrates collaboration into the application and utilize visualizations that can represent both the evolved artifacts' on and off states. For the GenSynth platform, the largest hurdle was getting users since the web does not particularly cater to aural centric content as it does with solely visual content. In addition, users were hesitant to authenticate with either a username and password or Facebook.

### *Future Work*

Future work with GenSynth can be divided into future work with asNEAT, the asNEAT visualization library, and the GenSynth collaborative platform. Due to WebAudio and WebMIDI being recent developments for the web and only available in a few browsers, the GenSynth platform will greatly benefit from the potential new participants and stability that more robust and widespread implementations across browsers can provide.

The asNEAT library has many areas of possible future study and enhancement. First, asNEAT currently only allows a single connection between two nodes. Allowing multiple connections for different parameters could add to what sounds could be produced. Many new types of nodes could also be extended or developed to help the library more easily evolved to well-known and novel audio affects. The convolver node, for example, currently only uses a noise signal when processing incoming audio and could be extended to utilize multiple types of convolution, even using audio generated from another asNEAT network. A noise oscillator could also be included to help build more percussive like sounds. Finally, the library can benefit from more utilization of the NEAT algorithm. In current implementations, GenSynth only utilizes asNEAT to crossover the selected instruments in the current population and previous parents.

Interface improvements could enable a site wide backpack that would let users select instruments they like from any page in the platform to be used during evolution, potentially integrated greater diversity into the population. Similarly, innovations could be tracked across all evolved instruments platform wide to better match similar mutations when evolved separately, allowing for more nodes to be matched during crossover.

Next, the asNEAT visualization library could be extended in many ways to study how varying visualizations help aid in the instrument evolution process. An oscilloscope or spectrum visualization could be included to see if they are preferred to the current spectrogram and network hybrid visualization.

Finally, many parts of the collaborative GenSynth could be further developed and studied in how evolving instruments is affected. First off, many more parameters and evolution options could be exposed to the user. If more advanced users wanted to change something specific to an instrument, they could manually edit the topology and varying parameters. Although the topology can be locked down, future implementations could allow the user to lock in only certain nodes or parameters as described by Cristyn [Magnus 2010]. Similarly, the mutation distance interpolation could be set to interpolate parameters within the system in new ways. To aid in searching for instruments, a tag cloud could be provided, similar to Picbreeder [Secretan et al. 2011]. This visualization of the varying tags applied to instruments would highlight the most utilized tags to aid in searching for what kind of instruments exist. Tags could also be applied automatically via clustering and asNEAT speciation to determine genres of instruments and easing some of the work users would have to put into evolving instruments.

Additional ways could be developed to help lessen user fatigue during evolution and add to what can be done with an instrument. During evolution, instruments could be selected based on how much they were used by the evolver, similar to Woolf and Yee-King's Sound Gallery [Woolf and Yee-King 2003]. If a user played one instrument a lot, it would be automatically be selected as a parent for the next generation. Automated fitness functions could also be utilized to help seed GenSynth with an initial set of interesting instruments evolved towards varying sounds. Automated fitness could additionally be used in a hybrid like manner by evolving a larger population each

generation and filtering out instruments that were complex sounding or were too noisy [Moore and Spires 2006]. Furthermore, providing additional ways for users to audition instruments, like sweeping interfaces, could aid in their enjoyment of the platform [Tubb and Dixon 2014; Yee-King 2011]. Finally, users may be more motivated to use the software by provided more ways to use the instruments after they have been evolved. A song could be automatically generated after selected selecting what instruments should be in it, or an interface could be provided for users to compose their own works. Current research is focusing on combining song auto generation and user composing for live performances [Rice and Tsoukalas 2015].

## References

- Biles, J. 1994. GenJam: A genetic algorithm for generating jazz solos. Proceedings of the International Computer Music Conference, 131–131.
- Dahlstedt, P. 2001. Creating and exploring huge parameter spaces: Interactive evolution as a tool for sound generation. Proceedings of the 2001 International Computer Music Conference, 235–242.
- Donahue, C. 2013. Applications of genetic programming to digital audio synthesis. The University of Texas at Austin, Department of Computer Science.
- Engelbrecht, A.P. 2007. Computational intelligence: An introduction. John Wiley & Sons.
- Foobar2000. 2014. Foobar2000. <http://www.foobar2000.org/>.
- Garcia, R. 2001. Growing sound synthesizers using evolutionary methods. Proceedings ALMMA 2001: Artificial Life Models for Musical Applications Workshop, (ECAL 2001).
- Greenfield, G.R. 2008. Co-evolutionary methods in evolutionary art. In: The Art of Artificial Evolution. Springer, 357–380.
- Holland, J.H. 1992. Genetic algorithms. Scientific American 267, 1, 66–72.
- Johnson, C.G. 1999. Exploring the sound-space of synthesis algorithms using interactive genetic algorithms. Proceedings of the AISB'99 Symposium on Musical Creativity, 20–27.
- Johnson, D. 1995. Sound Synthesis using a Genetic Algorithm. BSc. The Department of Electrical and Electronic Engineering at the University of Cape Town.
- Langdon, W. 2005. Pfeiffer-A distributed open-ended evolutionary system. AISB, 7–13.
- Magnus, C. 2010. Evolutionary Sound: a Non-Symbolic Approach to Creating Sonic Art with Genetic Algorithms. Diss. University of California, San Diego.
- Mccormack, J. 2008. Facing the future: Evolutionary possibilities for human-machine creativity. In: The Art of Artificial Evolution. Springer, 417–451.
- Mcdermott, J.M. 2008. Evolutionary computation applied to the control of sound synthesis. Diss. University of Limerick.
- Moore, B. and Spires, W. 2006. Signal Complexification using Frequency Modulation and Neuroevolution. BSc. School of Computer Science Orlando, FL

- Rice, D. and Tsoukalas, K. 2015. GenSynthPerformer.  
<https://github.com/davidlukerice/genSynthPerformer>.
- Rogers, C. 2012. Web audio API. Draft, Version 1. <https://webaudio.github.io/web-audio-api/>.
- Russ, M. 2009. Sound Synthesis and Sampling. CRC Press.
- Secretan, J., Beato, N., D'Ambrosio, D.B., ET AL. 2011. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evol. Comput.* 19, 3, 373–403.
- Stanley, K.O. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines* 8, 2, 131–162.
- Stanley, K.O. and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10, 2, 99–127.
- Szumanski, S.R., WU, A.S., and Hughes, C.E. 2006. Conflict resolution and a framework for collaborative interactive evolution. *Proceedings of the National Conference on Artificial Intelligence*, 512.
- Takagi, H. 2001. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE* 89, 9, 1275–1296.
- Takala, T., Hahn, J., Gritz, L., Geigel, J., and Lee, J.W. 1993. Using physically-based models and genetic algorithms for functional composition of sound signals, synchronized to animated motion. George Washington Univ Washington DC Dept of Electrical Engineering and Computer Science.
- Tubb, R. and Dixon, S. 2014. The Divergent Interface: Supporting Creative Exploration of Parameter Spaces. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 1257.
- Wehn, K. 1998. Using ideas from natural selection to evolve synthesized sounds. *Proceedings of the Digital Audio Effects DAFX98 workshop, Barcelona*.
- Woolf, S. and Yee-King, M. 2003. Virtual and physical interfaces for collaborative evolution of sound. *Contemporary Music Review* 22, 3, 31–41.
- Yee-King, M.J. 2011. Automatic sound synthesizer programming: techniques and applications. Diss. University of Sussex.

Yuksel, K.A., Bozkurt, B., and Ketabdar, H. 2011. A software platform for genetic algorithms based parameter estimation on digital sound synthesizers. Proceedings of the 2011 ACM Symposium on Applied Computing, 1088–1089.

## Appendix A: Source

All code used in the study is available on GitHub

- asNEAT instrument evolution and audio synthesis library:  
<https://github.com/davidlukerice/asNEAT>
- asNEAT-visualizer library for visualizing asNEAT Instruments:  
<https://github.com/davidlukerice/asNEAT-visualizer>
- GenSynth Collaborative Web Platform (Both client and server software):  
<https://github.com/davidlukerice/genSynth>

## Appendix B: Usability Study 1 Procedures

### 1. Brief Introduction

This is a study about “Evolving” new digital instruments by mutating and breeding previous instruments to search for interesting combinations of the underlying audio network. This particular study is looking at the user experience as opposed to the actual sound the instruments currently make.

I’m going to ask you to do a set of tasks without any initial help. During the tasks, the screen and mouse/keyboard inputs will be recorded, the webcam or laptop microphone will **not** be used to retain anonymity.

Please describe what you’re thinking while doing the task and feel free to ask any questions as they arise if the task or a particular term is unclear. Also feel free to make suggestions at any point in the study. After the tasks, there will be a short questionnaire to complete.

### 2. 15 minutes of doing various tasks (Speak Aloud Policy)

- a. Please read the help instructions (45 secs)
  - i. Were they able to find it?
  - ii. Other Notes
- b. Listen to a few instruments (optimal: 1 click per listen)
  - i. \_\_\_ Clicks for \_\_\_ instruments
  - ii. Notes
- c. **Select three instruments** that you like and generate the next set of instruments (optimal: 4 clicks)
  - i. \_\_\_ Clicks
  - ii. Did they select any parents? If not, did they know they could select the parents?
  - iii. Did they know they could select anywhere on the visualization to select?
  - iv. Did they try to use any hotkeys for selecting?
  - v. How did they determine that they liked the instruments?  
(Visualization? Play button?, &c...)
  - vi. Other Notes
- d. **Regenerate** the current population and then go **back** to the previous generation (1 click per action)
  - i. \_\_\_ Clicks
  - ii. Other Notes
- e. Turn on the **onscreen piano** (1 click)
  - i. \_\_\_ Clicks
  - ii. Other Notes
- f. Play several notes of the second child instrument with the onscreen piano
  - i. Did they understand how to make an instrument “live”?
  - ii. Did they select through mouse? Or key presses?

- iii. Did they play the piano by mouse or by key presses?
  - iv. Other Notes
- g. Turn on **MIDI** (1 Click)
  - i. \_\_\_\_ Clicks
  - ii. Did they know what MIDI was? Have to explain it's for the hardware keyboard?)
  - iii. Other Notes
- h. Select a different instrument and play several notes
  - i. Other Notes
- i. Give participants time to play around with the platform however they desire (max 5 minutes)
  - i. Other Notes

### 3. GenSynth Study 1 Survey

## Appendix C: Usability Study 1 Survey Questions

How engaging was the following?

	Not Engaging	Slightly Engaging	Moderately Engaging	Very Engaging	Extremely Engaging
Software as a whole	<input type="radio"/>				

How engaging was each method of interacting with the instruments?

	Didn't Use	Not Engaging	Slightly Engaging	Moderately Engaging	Very Engaging	Extremely Engaging
Play Button	<input type="radio"/>					
Clicking the Onscreen Keyboard	<input type="radio"/>					
Using hot-keys for the onscreen keyboard	<input type="radio"/>					
External MIDI Keyboard	<input type="radio"/>					

How useful would the following ways to save an instrument be to you?

	Not Useful	Slightly Useful	Moderately Useful	Very Useful	Extremely Useful
Copy/Paste Code	<input type="radio"/>				
Temporary "Backpack"	<input type="radio"/>				
Permanent "Backpack" you can access through logging in	<input type="radio"/>				

If you had to log in to the application, how useful would each login method be for you?

	Not Useful	Slightly Useful	Moderately Useful	Very Useful	Extremely Useful
Email / Password	<input type="radio"/>				
Facebook Login	<input type="radio"/>				
Google+ Login	<input type="radio"/>				
Twitter Login	<input type="radio"/>				
Other	<input type="radio"/>				

How likely would you be to share your instrument with the following?

	Very Unlikely	Unlikely	Somewhat Unlikely	Undecided	Somewhat Likely	Likely	Very Likely
Copy/Paste Code	<input type="radio"/>						
Email	<input type="radio"/>						
Facebook	<input type="radio"/>						
Google+	<input type="radio"/>						
Twitter	<input type="radio"/>						
Other	<input type="radio"/>						
Other	<input type="radio"/>						

Was there any part of the interface that you didn't understand?

Was there any part of the interface that seemed out of place?

Was there any part of the instrument or interface you wished to have more or less control over?

Rate your experience with each of the following Musical Activities

	No Experience	Very Little Experience	Some Experience	A lot of Experience	Professional Experience
Listening to Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Playing Music by Ear	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Playing Music from Tabs	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Playing Music from Sheet Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Playing Music in a Group (Jamming, Orchestra, Band, &c.)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Composing Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Recording Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Producing/Mastering Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creating Interactive Media with an audio component (Max, Pure Data, &c.)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Rate your proficiency with the following instruments

	No Experience	Beginner	Intermediate	Advanced	Professional
Drums	<input type="radio"/>				
Guitar (Acoustic / Electric)	<input type="radio"/>				
Guitar (Bass)	<input type="radio"/>				
Piano / Keyboard	<input type="radio"/>				
Violin	<input type="radio"/>				
Voice	<input type="radio"/>				
Other	<input type="radio"/>				
Other	<input type="radio"/>				
Other	<input type="radio"/>				

What age range do you fall into?

- 18-24
- 25-34
- 35-44
- 45-64
- Click to write Choice 5

What degree(s) do you currently have or are pursuing?

- Undecided \_\_\_\_\_
- Computer Science
- Computer Engineering
- Engineering Other \_\_\_\_\_
- Music (Please Enter Specialization, eg. Composition, Violin Performance, Instrument Education) \_\_\_\_\_
- Music Education (Please Enter Specialization) \_\_\_\_\_
- Other \_\_\_\_\_
- Other \_\_\_\_\_
- None
- Minor \_\_\_\_\_
- Minor 2 \_\_\_\_\_

Please enter any additional comments or remarks you may have about the software

## Appendix D: Usability Study 2 Survey Questions

How engaging was the following?

	Not Engaging (1)	Slightly Engaging (2)	Moderately Engaging (3)	Very Engaging (4)	Extremely Engaging (5)
Software as a whole	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

How engaging was each method of interacting with the instruments?

	Didn't Know About (11)	Didn't Use (1)	Not Engaging (6)	Slightly Engaging (2)	Moderately Engaging (3)	Very Engaging (4)	Extremely Engaging (5)
Play Button	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Clicking the Onscreen Keyboard	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Using hot-keys for the onscreen keyboard	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
External MIDI Keyboard	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Was there any part of the interface that you didn't understand or seemed out of place?

Was there any part of the instrument or interface you wished to have more or less control over? If so, In what ways?

Rate your experience with each of the following Musical Activities

	No Experience (1)	Very Little Experience (2)	Some Experience (3)	A lot of Experience (4)	Professional Experience (5)
Listening to Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Playing Music by Ear	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Playing Music from Tabs	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Playing Music from Sheet Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Playing Music in a Group (Jamming, Orchestra, Band, &c.)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Composing Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Recording Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Producing/Mastering Music	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creating Interactive Media with an audio component (Max, Pure Data, &c.)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Rate your proficiency with the following instruments

	No Experience (1)	Beginner (2)	Intermediate (3)	Advanced (4)	Professional (5)
Drums	<input type="radio"/>				
Guitar (Acoustic / Electric)	<input type="radio"/>				
Guitar (Bass)	<input type="radio"/>				
Piano / Keyboard	<input type="radio"/>				
Violin	<input type="radio"/>				
Voice	<input type="radio"/>				
Other	<input type="radio"/>				

What age range do you fall into?

- 18-24 (1)
- 25-34 (2)
- 35-44 (3)
- 45-64 (4)

What degree(s) do you currently have or are pursuing?

- Undecided (1) \_\_\_\_\_
- Computer Science (2)
- Computer Engineering (3)
- Engineering Other (4) \_\_\_\_\_
- Music (Please Enter Specialization, eg. Composition, Violin Performance, Instrument Education) (5) \_\_\_\_\_
- Music Education (Please Enter Specialization) (6) \_\_\_\_\_
- Other (7) \_\_\_\_\_
- Other (8) \_\_\_\_\_
- Minor (10) \_\_\_\_\_
- Minor 2 (11) \_\_\_\_\_
- None (9)

Please enter any additional comments or remarks you may have about the software

## Appendix E: Mutation Parameters

During each mutation step, a mutation is selected based on the chances listed in **Table 4**. When a connection is mutated, the connection's weight has an 80% chance of mutating by a small delta within the given delta range and a 20% chance of mutating to a random number within the random range (**Table 5**). The delta range is determined based on a linear interpolation between the mins and maxes at the user set mutation step distance. The table lists [min1, max1] to [min2, max2]. Some random ranges allow the values to be inverted as well. Similar parameters for when node parameters are mutated are listed in **Table 6**.

	Split Connection	Add Connection	Add Oscillator	Mutate Connection	Mutate Parameter
Topology Unlocked	20%	20%	10%	25%	25%
Topology Locked	0%	0%	50%	0%	50%

**Table 4: Mutation Chances per Step**

asNEAT Node	Parameter	Delta Range	Allowable Delta Range	Random Range
Normal Connection	-	[0.05, 0.3] to [0.1, 0.6]	-1 to 1	0.1 to 1
Filter	Frequency	[10, 100] to [300, 700]	0 to 1500	0 to 1500
	Q	[0.0001, 1] to [3, 10]	0.0001 to 20	0.0001 to 20
	Gain	[0.1, 1] to [2, 6]	-5 to 5	-5 to 5
Gain	Gain	[0.1, 0.3] to [0.5, 1]	0.1 to 1.5	0.5 to 1.5
Note Oscillator	Frequency	[10, 200] to [300, 700]	-2000 to 2000	-2000 to 2000
Oscillator	Frequency	[10, 200] to [300, 700]	-2000 to 2000	-2000 to 2000

**Table 5: Connection Mutation Parameters**

asNEAT Node	Parameter	Type	Delta %	Mutation Delta	Allowable Delta Range	Random Range
Compressor	Threshold	Exp	80%	[1,10] to [5,15]	-50 to 0	-50 to 0
	Knee	Exp	80%	[1,10] to [5, 15]	0 to 40	20 to 40
	Ratio	Exp	80%	[0.01, 0.5] to [1, 4]	1 to 20	8 to 16
	Reduction	Exp	80%	[0.01, 0.5] to [1, 4]	-20 to 20	-10 to 0
	Attack	Exp	80%	[0.005, 0.05] to [0.1, 0.2]	0 to 1	-10 to 0
	Release	Exp	80%	[0.005, 0.02] to [0.01, 0.05]	0 to 1	0 to 0.1
Delay	Delay Time	Exp	80%	[0.05, 0.5] to [0.1, 1]	0 to 3	0 to 3
Feedback Delay	Delay Time	Exp	80%	[0.1, 0.4] to [0.4, 0.8]	0 to 3	0 to 3
	Feedback Ratio	Exp	80%	[0.05, 0.1] to [0.1, 0.3]	0 to 3	0 to 3
Filter	Type	-	0%	-	-	0 to 7
	Frequency	Exp	80%	[10, 100] to [300, 700]	0 to 1500	0 to 1500
Gain	Gain	Exp	80%	[0.02, 0.1] to [0.2, 0.4]	-1.5 to 1.5	0.5 to 1.5
Note Oscillator	Type	-	0%	-	-	0 to 4
	Note Offset	Exp	80%	[1, 4] to [5, 15]	-20 to 20	-20 to 20
	Attack Duration	Exp	80%	[0.01, 0.05] to [0.1, 0.3]	0.01 to 1.0	0.01 to 1
	Decay Duration	Exp	80%	[0.01, 0.05] to [0.1, 0.3]	0.01 to 1.0	0.01 to 1
	Release Duration	Exp	80%	[0.01, 0.05] to [0.1, 0.3]	0.01 to 1.0	0.01 to 1
	Attack Volume	Exp	80%	[0.01, 0.05] to [0.1, 0.3]	0.5 to 1.5	0.5 to 1.5
Oscillator	Type	-	0%	-	-	0 to 4
	Frequency	Exp	80%	[10, 200] to [50, 800]	-4186 to 4186	27.5 to 4186
	Attack Duration	Exp	80%	[0.01, 0.05] to [0.1, 0.3]	0.01 to 1.0	0.01 to 1
	Decay Duration	Exp	80%	[0.01, 0.05] to [0.1, 0.3]	0.01 to 1.0	0.01 to 1
	Release Duration	Exp	80%	[0.01, 0.05] to [0.1, 0.3]	0.01 to 1.0	0.01 to 1
	Attack Volume	Exp	80%	[0.01, 0.05] to [0.1, 0.3]	0.5 to 1.5	0.5 to 1.5

**Table 6: Node Mutation Parameters**