Somil Govani

Algorithmic Composition with Pure Data

Table of Contents

Introduction 2

Materials 3

Methods and Procedures 3

Data Collection and Measuring Success 3

Syntax 5

Program Building and Though Process 6

Results/Analysis

Quantitative Data 14

Qualitative Data 21

Calculations 21

Discussion 23

Comparison & Contrast to Other Algorithmic Composition Software 27

Comparison to Human Made Music 27

Conclusion 31

Bibliography 32

**Introduction:**

*Algorithmic Composition* is “the process of using some formal process to make music with minimal human intervention” with algorithms, mathematical models, and (in this case) computerized techniques to set parameters for essentially undetermined and random compositions. The first inklings of Algorithmic Composition came as early as the Ancient Greeks. Pythagoras, Plato, and Ptolemy, among some of the most prevalent Greek philosophers, introduced the idea of using “formalisms” (algorithms) in music with the notion that harmonies hold both natural and mathematic tendencies (Maurer). However, these notions were limited as theories and made no headway in actual algorithmic composition. Guido d’Arezzo formally introduced it when translating vowels of word phrases into notes; Mozart created a similar phenomenon in his game *Musikalisches Würfelspiel* (“Musical Dice”), which created compositions based solely on the probability of dice rolls (Edwards). Iannis Xenakis pioneered the use of Algorithmic Composition in a computerized form through use of stochastic probability and mathematical models in the 1960s (Maurer). After this age, computerized and mathematic composition popularized and made advancements from many hands.

Prior studies of music theory that yielded results of peculiar mathematic and patterned tendencies triggered this idea. To recollect, the utilization of seemingly arbitrary scales for improvisation in Jazz Music fomented this interest. These aspects along with collective interests in music, computer science, and mathematics sprouted this project based on the initial inquiry of musical relations to spontaneous, yet controlled generation. The goal of this project is to create a self-sufficient computer program, which self-generates unique music, through implementation of various concepts of algorithmic composition, computer science, music theory, and mathematics. The purpose of this project is to combine the aforementioned fields into one program through various determined ‘scientific qualities in music’ and algorithms sharing similarities with Markov Chains, stochastic algorithms, and mathematical models with a hopefully genuine (human) product. These various aspects were determined through extensive research/observation and reverse engineering of previously created automated composition. If successful, this project will not only validate the connection between music theory and mathematics, but also further develop the logistics behind the creation of music. This will prove significant in furthering investigations into a computer’s ability to compose music (as well as heightening those abilities), which has become an increasingly large industry, and ‘humanize’ it as much as possible.

**Materials:**

The only significant tool used to accomplish the goal of this project was a visually based data flow programming language known as Pure Data (<http://www.puredata.info>). A free and open source project, Pure Data allows the implementation and connection of various parameters on the three main musical identifiers: pitch, velocity, and duration. To add, Pure Data makes it possible to convert seemingly arbitrary values into ones that are usable through almost any computer interface, midi (in this case, midi piano).

**Methods and Procedures:**

Data Collection and Measuring Success

Prior to initiating programming phases, it was necessary to determine a plan/structure in order to measure the success of the product in fulfilling the goal and to create a skeleton of procedures to accomplish this task efficiently. The generation of a list of *scientific qualities in music* accomplished this (See **Figure 1a and 1b**). This list was derived through various methods, generally accountable to observations from extensive research of music theory/algorithms, observations from musical studies, and reverse engineering of various previously made programs of automated composition. Instead of attempting to implement all of these aspects at once, it is more ideal to approach each one at a time to take periodic data collection and minimize error. The amount of and to the extent, which these aspects could be implemented into the program marks one factor of success in this endeavor. Another parallel method of quantitative success is through the measurement of the 3 main attributes to musical notes: pitch (frequency in Hz converted into Midi in this case), velocity (attack Hz in Midi), and duration (milliseconds). The evaluation of these three results in particular will show the success in limiting the arbitrary values that would result from blind randomization. Additionally, these data will illustrate how effectively computerized music emulates ‘real’ music (human-made) by comparing the computer’s data to ‘genuine’ data. With these methods, it is possible to reduce bias, varied, and subjective measurements of success that would otherwise be associated with musical evaluation.

**Figure 1a: Scientific Qualities in Music Identified (attempted to emulate underlined terminology in project)**

* **Note Variation** **🡪** utilization of an assortment of notes, sounds, and frequencies (vary pitch, velocity (attack), and duration)
* **Musical Randomization & Improvisation 🡪** generate music without premeditation or pre-composition (can apply to most other aspects)
* **Determinacy & Repetition 🡪** repeat notes, riffs, or strings of notes multiple times
* **Frequency Parameters 🡪** control over the parameters of the upper and lower extremes of pitch variation
* **Limit Step Size (Walks) 🡪** regulation of musical randomization & improvisation through limitations on jump ranges
* **Musical Scales 🡪** utilization of groups of notes that hold significance in music theory and frequency harmonics
* **Dynamics 🡪** change in the velocity, attack, and volume of notes
* **Duration 🡪** note lengths
* **Polyphony & Harmony 🡪** utilization of simultaneous and independent musical tendencies in complement to one another
* **Timbre 🡪** tone and quality of notes (instrumental variation)
* **Silence 🡪** periodic breaks in musical progression
* **Consonance & Dissonance 🡪** utilization of notes that do or do not blend well together
* **Imperfection** **🡪** swing, off-beat, and other aspects to make music *real*
* **Musical Structure 🡪** organization of music in bars, beats, and tempos
* **Progression** 🡪 feel of tempo, repeated choruses, beginning end, musical direction

**Figure 1b: Concise & Visual List of Only the Aspects Used in Program**

Inputs/Inlets

Outputs/Outlets

Main Types of Syntax

**Object** – can take on vast variety of attributes depending on value

Shape:

**Message** – sends value to designated output (clickable)

Shape:

**Number/Number 2** – value is always a number; continuously sends value through output; altered through a vertical strafe scroll (number 2) or conventional input

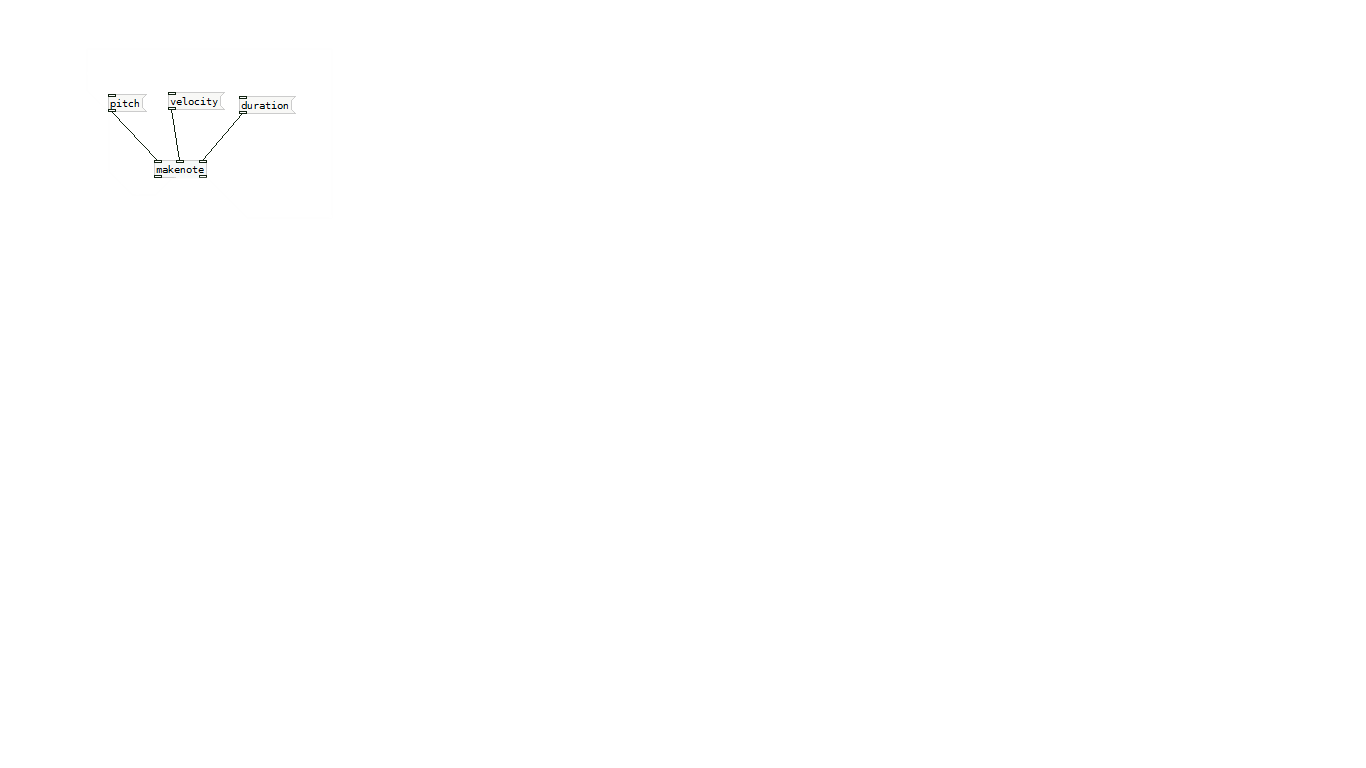
Shape:

**Bang** – initiates action in output (clickable)

Shape:

Useful Object Types

Full list of object are found with right-click on canvas (white) > Help. These are the ones I will possibly use or yield potential for algorithmic composition.

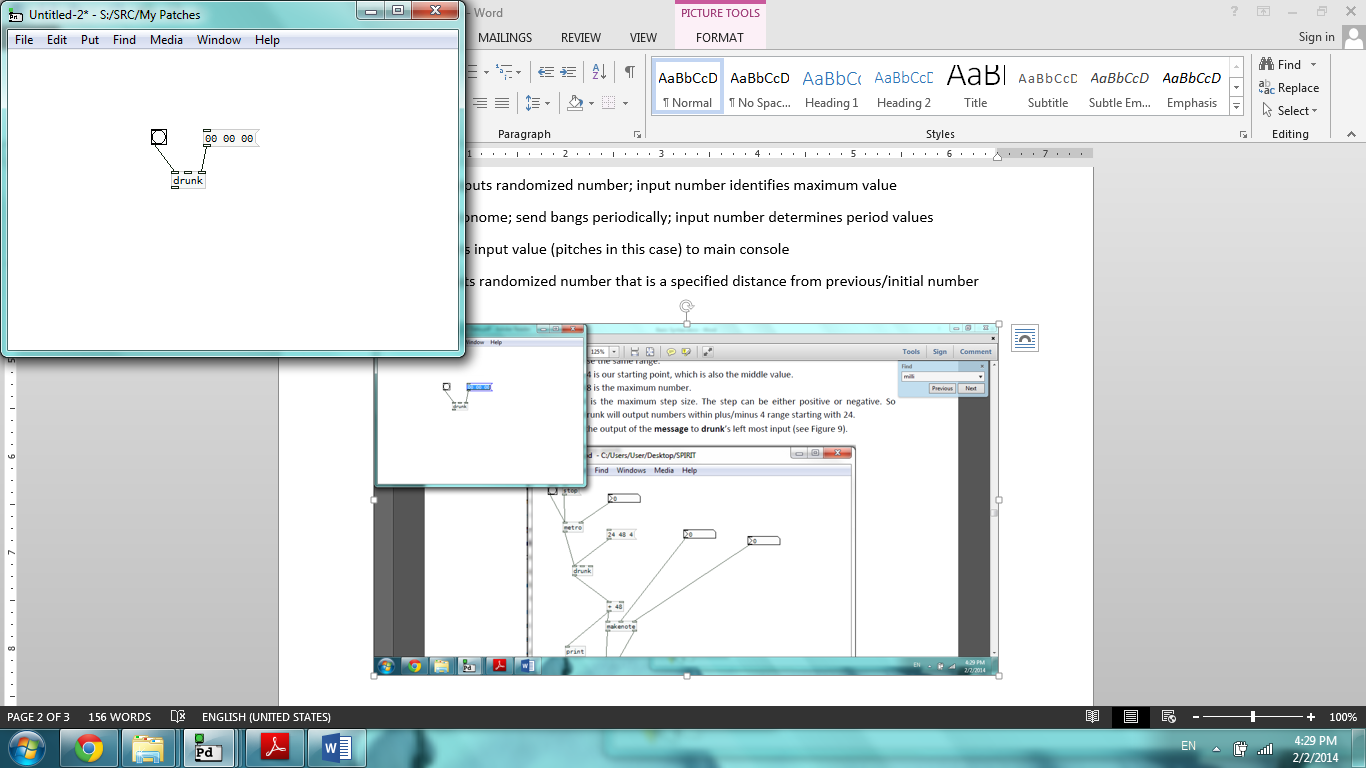
**makenote** – produces sound values based on three input values: *pitch (0-127 Midi), velocity (attack) (0-127 Midi), duration (milliseconds)*

**noteout** – exports the sound values as actual noise when it is an output of *makenote*

**random** – outputs randomized number; input message OR numerical suffix identifies maximum value

**metro** – metronome; send bangs periodically; input number determines period values in milliseconds

**print** – exports input value (pitches in this case) to main console

**drunk** –outputs randomized number that is a specified distance from previous/initial number input

* Needs a bang to initiate
* Input message contains numerical parameters in the format X Y Z
* **X** = Starting/initial point
* **Y** = Maximum value
* **Z** = Maximum step size (next value ranges from 0-Z +/- the previous value)

**table** – links an array (message) to main syntax chain through **set** (message) and **tabread** (object which implements tables); array contains list of values that input (potentially used for scales?)

**send/receive** – sends/receives values wirelessly

Program Building and Thought Process

This section will walk through the basic syntax and thought process for implementing each aspect of algorithmic composition individually and chronologically. See the changelog (data book) for a detailed description of changes through versions.

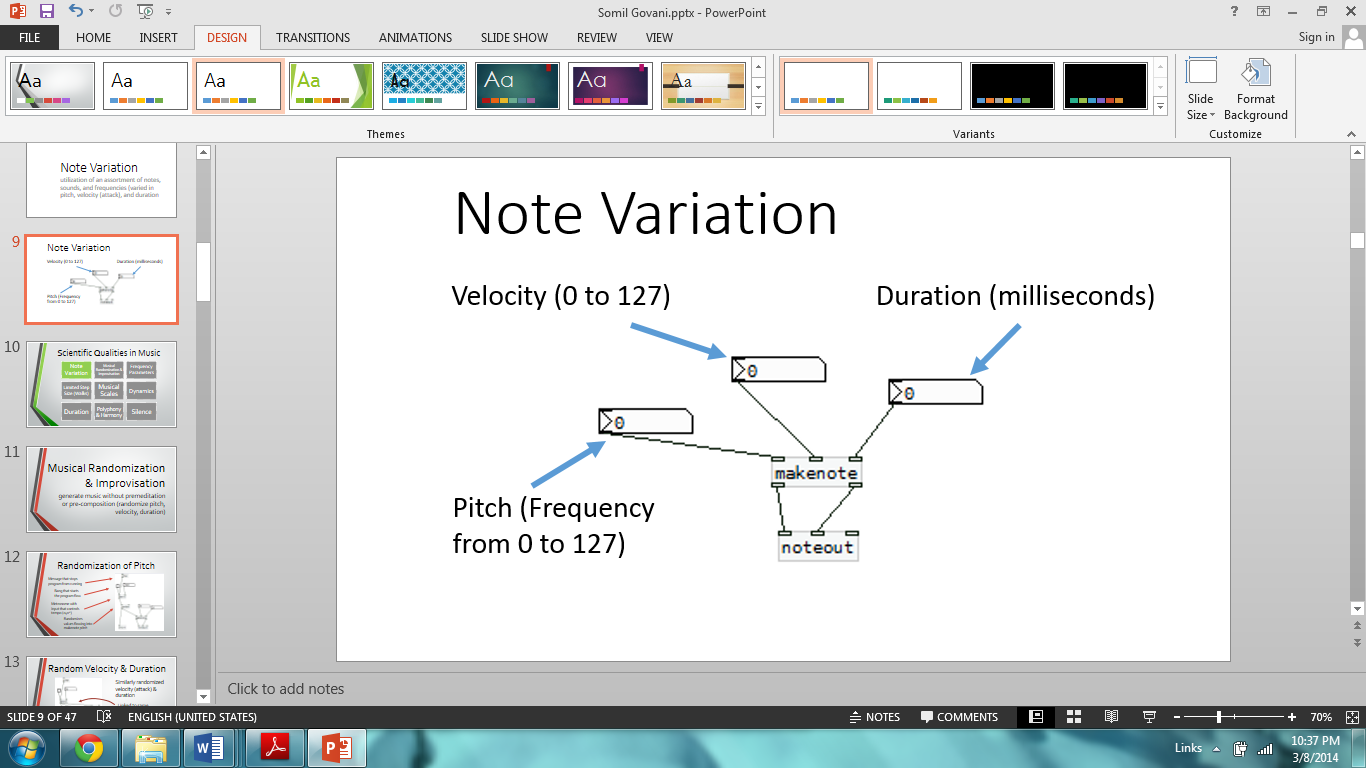
First, note variation is the utilization of an assortment of notes, sounds, and frequencies that vary in pitch, velocity, and duration. The *makenote* object accomplished this with its three inputs that control pitch frequency (midi 0-127), velocity (midi 0-127), and duration (ms) (see Figure 2). (Version 1.0)

Figure 2: Makenote Object

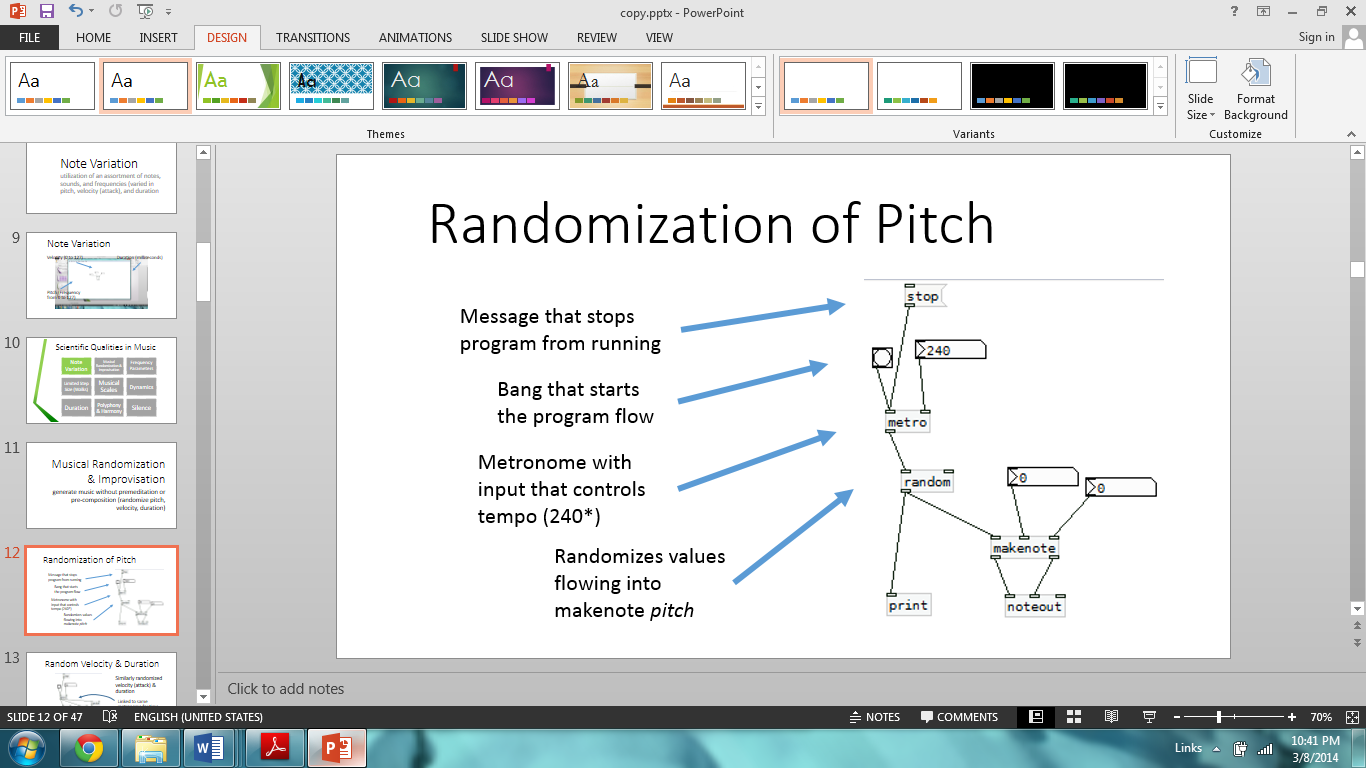
Second, musical randomization and improvisation is the when music is generated without pre-composition, in an essentially random manner. In order to randomize pitch, I random object was implemented into the first *makenote* input. To make the program self-sufficient, a *metronome* object with a controllable tempo was added in to send a continuous pulse for continuous notes once the *bang* initiates it. Consequently, a *stop* object was attached to the *metro* to end the pulse (See Figure 3). Similarly, to randomize velocity and duration, an identical random object was added to the respective branches and synchronized through a common *metro* (See Figure 4). (Version 1.1/1.2)

Figure 3: Randomizing Pitch Frequency

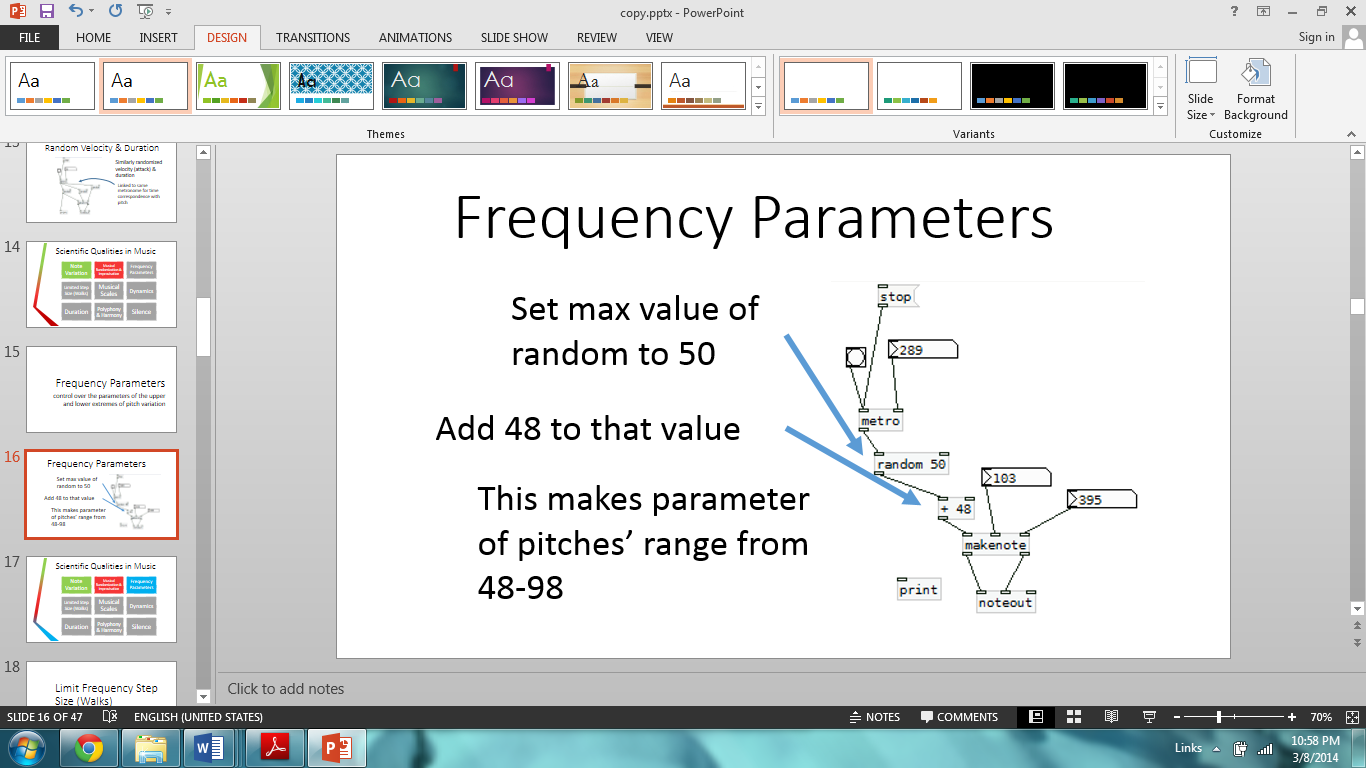


Figure 4: Randomize Velocity and Duration

Third, frequency parameters allow control over the parameters of the upper and lower extremes of pitch variation so it does not result in notes that are abnormally low or high. This was accomplished through adding an argument to the end of the random object. The first number after random identifies the maximum value it can output, 50. Then, a succeeding *+48* object adds 48 to whatever the *random* object outputs. For example, if *random* determines 25 as its value, then the final pitch value sums to become 73. Consequently, the range of frequencies is limited to 48-98 midi (See Figure 5). (Version 1.3)

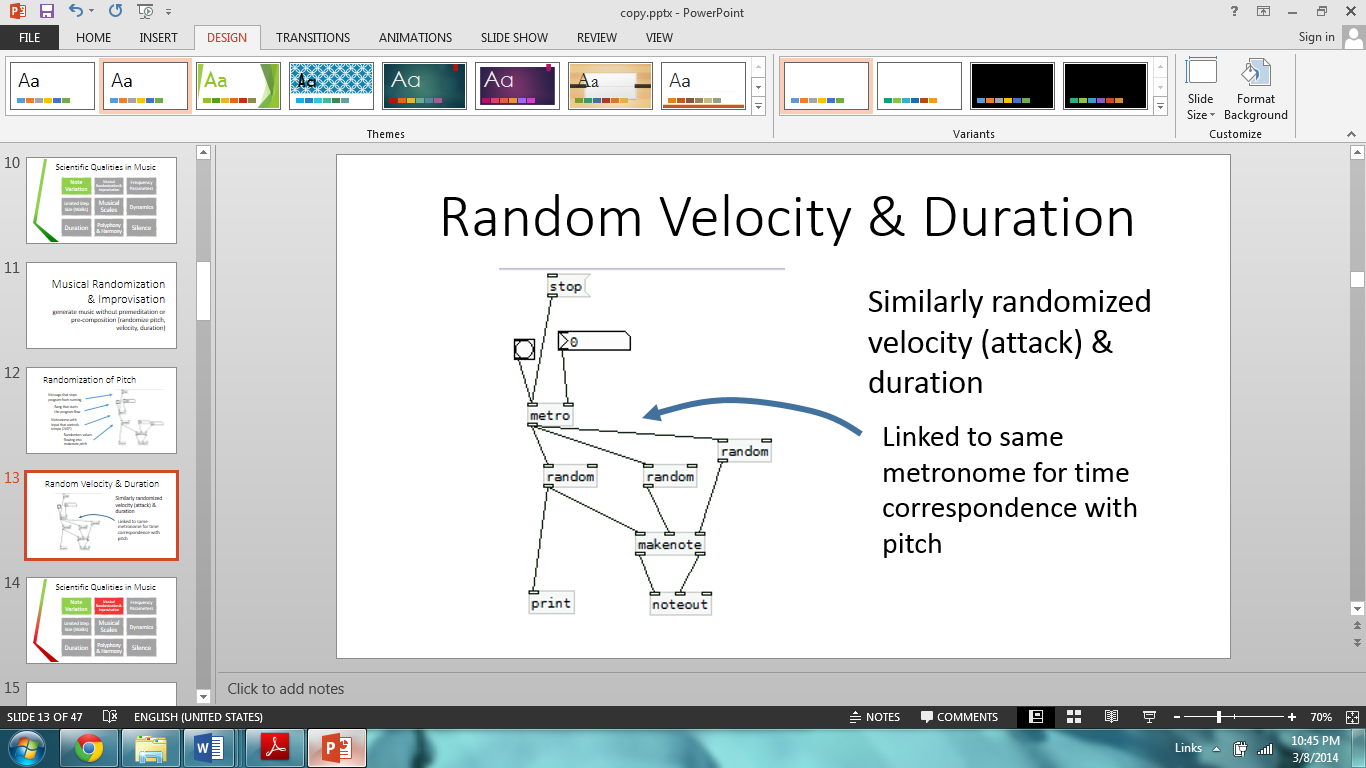


Figure 5: Setting Frequency Parameters

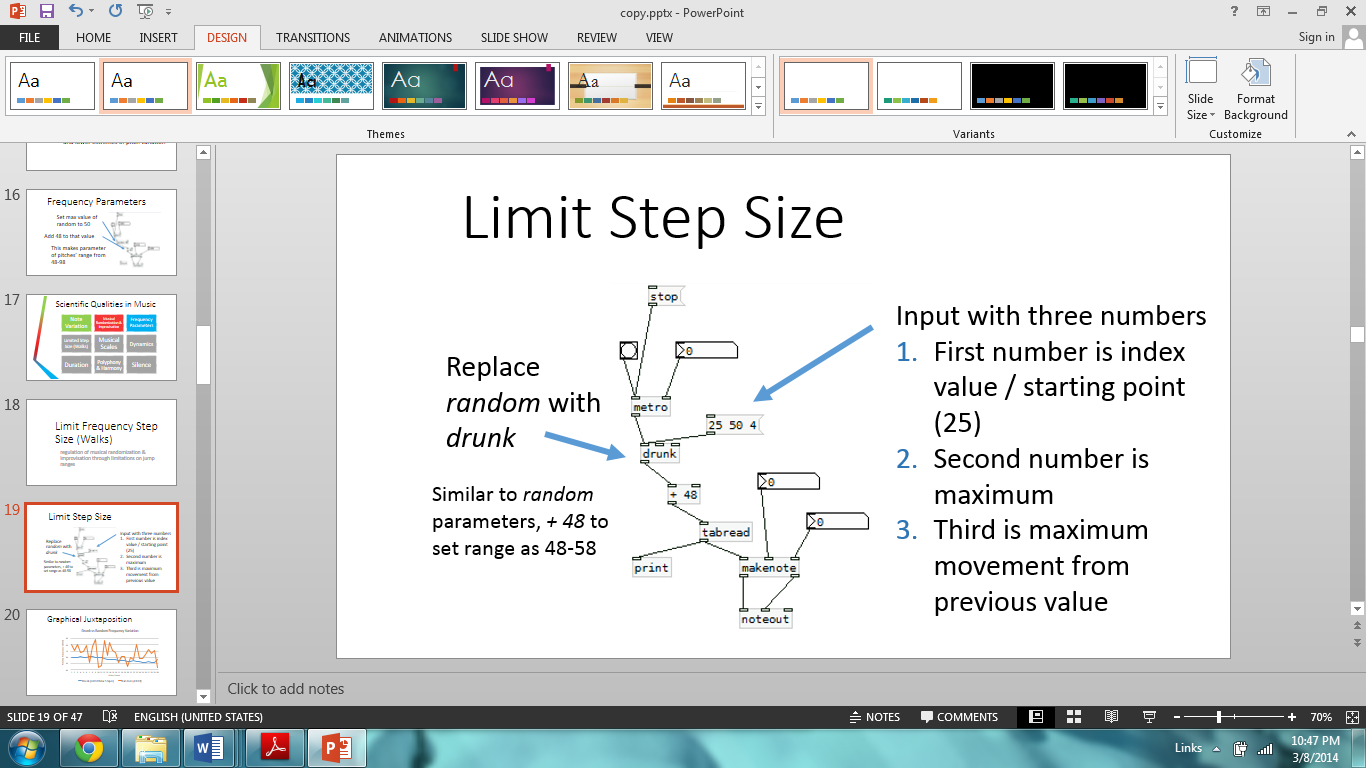
Fourth, the limitation of frequency step size, similar to what musicians refer to as ‘walks’, regulate musical variance through limitations on jump ranges. The goal of this is to make it so the pitch of the note does not jump too drastically, say from 49 to 90 in one pulse, giving the music and unnatural and sporadic feel. To do this, the *random* object in pitch is replaced with a *drunk* object. It is similar to *random* in that it generates a random value, but message inputs allow it to pick an index value (starting number), maximum (similar to random), and *drunk*’s unique resource, a maximum step size to make it so the pitch can only fluctuate up or down a certain number of values (4 in this case) (See Figure 6). Graph 1 illustrates the difference in the value outputs between the *drunk* and *random* objects. (Version 1.4.1)

Figure 6: Limiting Step Size with the Drunk Object

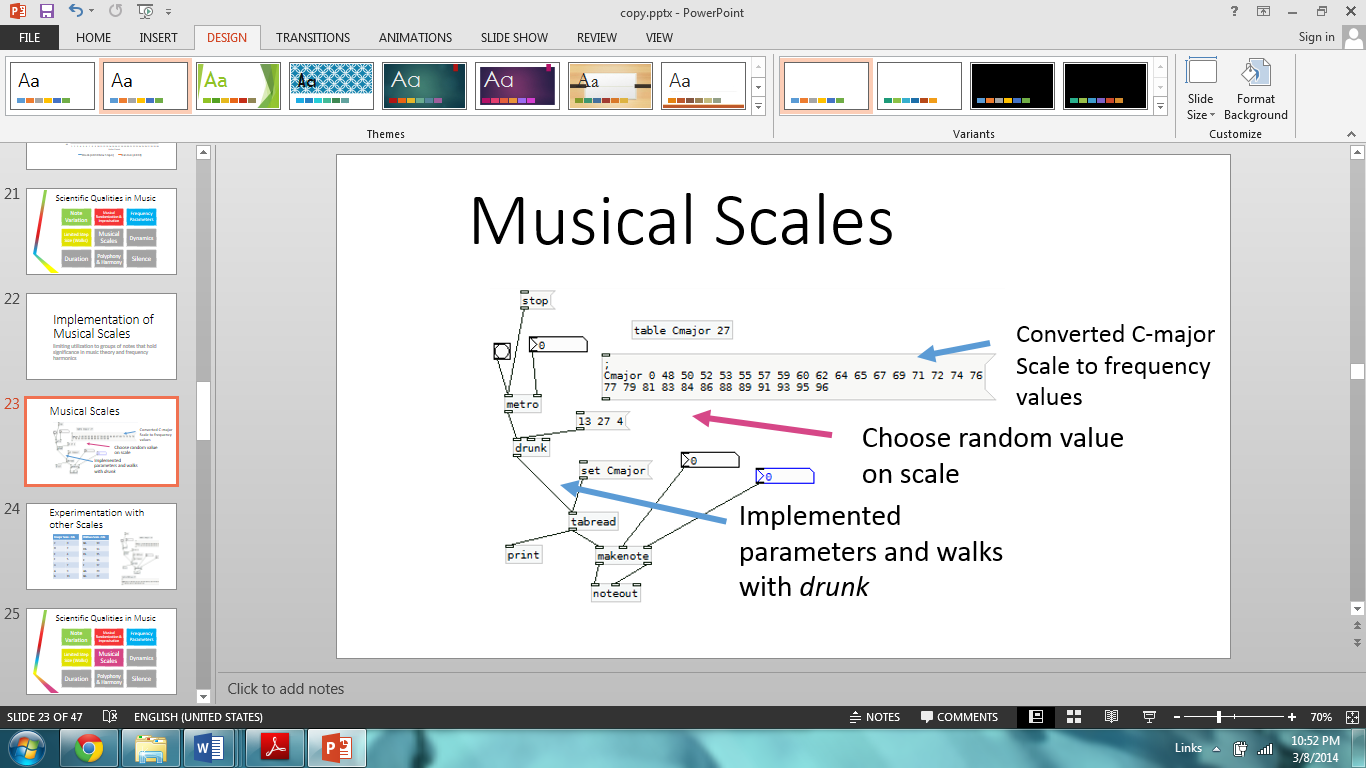
Fifth, the musical scales use concepts of music theory to limit the pitch to a group of notes that are harmonically significant for a number of reasons. To implement these, scales must first be translated from letter notes, to frequencies (Hz), to midi notes (0-127). Using the most basic scale at first, CMajor, it was converted into midi values. Then, the notes were translated into a tabled array and linked up with a *drunk* or *random* object to limit the note values to only notes on the C scale, increasing the patch’s musicality. To add, more scales were created and translated through these means and made available for manual toggle (See Figure 7). See calculations for information on deriving scales. (Version 1.4/1.4.1)

Figure 7: Implementing the Cmajor Scale

Sixth, dynamics, deviating from the pitch branch, are the use of changes in velocity, attack, and volume of notes to add musical interest. This was originally accomplished through randomizing the velocity input. However, because the volumes became too drastic, this was later limited to parameters of 60-100 through similar methods used for frequency parameters (See Figure 8). It should also be taken into consideration to use the drunk object as a method of slowly increasing or decreasing volumes with crescendos and decrescendos. (Version 1.5)

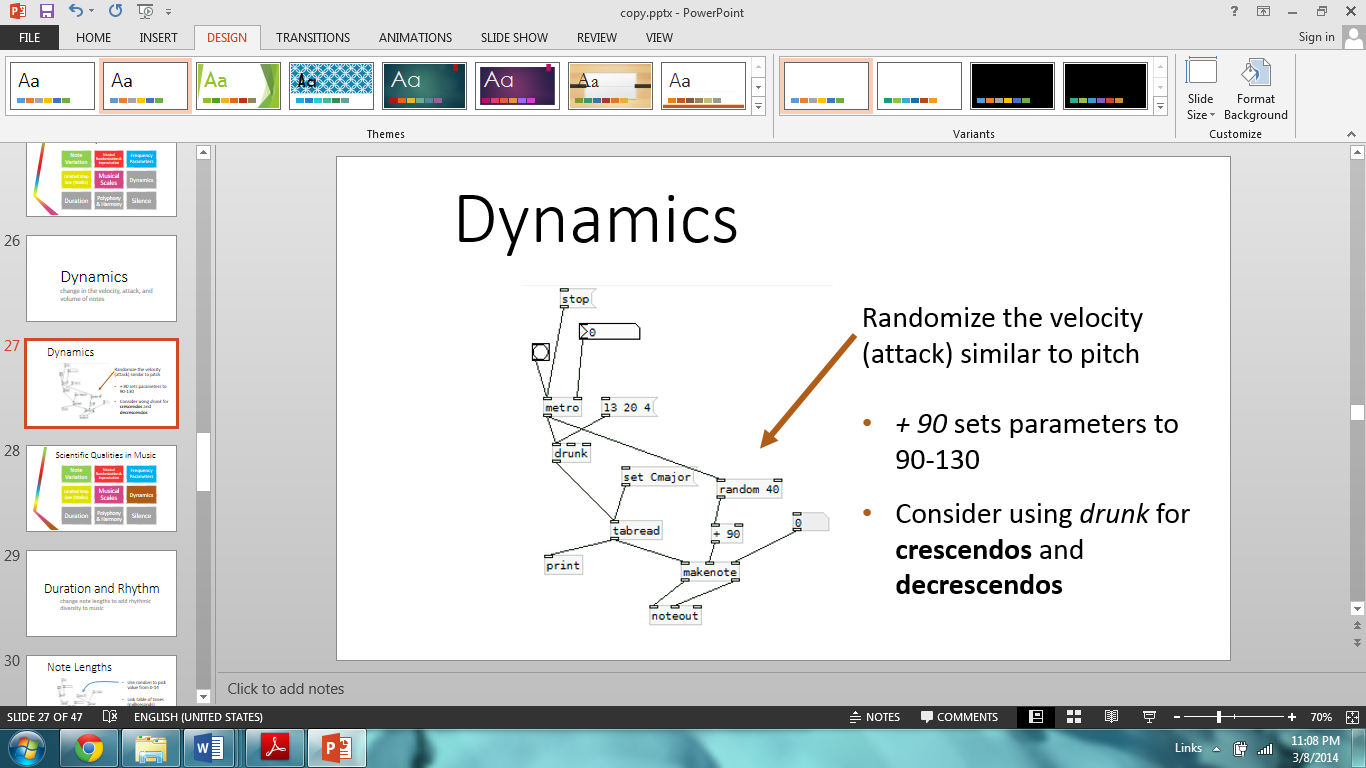


Figure 8: Randomizing Attack with Parameters

Seventh, duration and rhythm is the use of change in note lengths and patterns to add rhythmic diversity. This was originally accomplished through randomization in the respective *makenote* branch. However, because duration has no parameters to begin with, it must have drastic limitations to prevent excessively long or short notes. The combination of the random object and another array containing the desired eighth (500ms), quarter (1000ms), and half (2000ms) note estimates accomplished this. The uneven distribution of these timings added musical aesthetics. For tempos and rhythms, connecting a second *metro* object to generate and independent pulse to a *drunk* object varies the constant beat. This sets the *drunk* parameter from 200 to 400 starting in the middle with a max movement of 10, so the music van lose and pick up speed (See Figure 9). (Version 1.6)

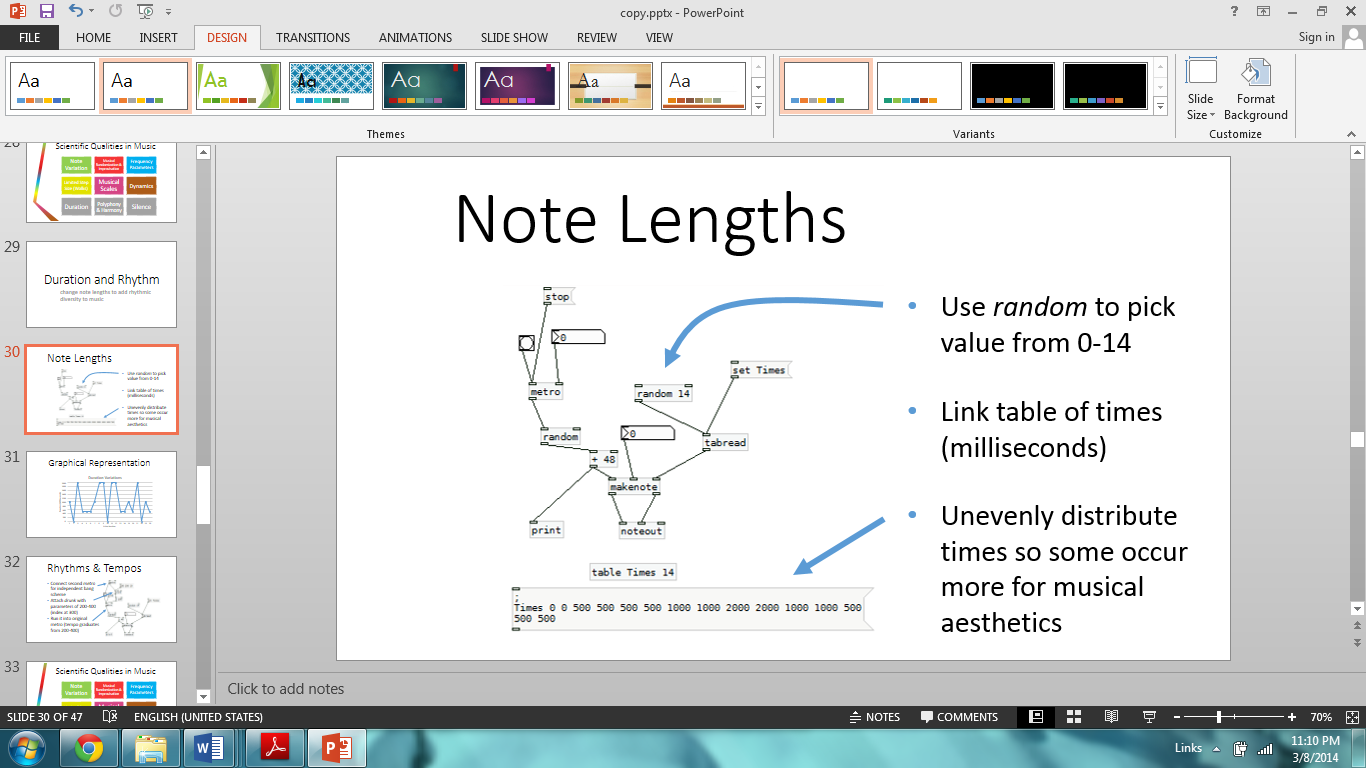


Figure 9: Set Note Durations

Eighth, polyphony and harmony is the utilization of simultaneous and independent musical tendencies in complement to one another to give the music and incredible increase in depth. A bass accompaniment accomplished this. First, it was ideal to send the velocity, time, and *metro* values wirelessly for consistency in the bass line (See Figure 10). These values were received on an entirely separate bass structure, mirrored from the original (See Figure 11). However, in this parallel structure, a lower octave parameter of the same scale gave the low note bass effect while still adding the aforementioned algorithms. These two parallel structures will play their tones independently, but simultaneously on click. Eventually, I negated the use of wireless pings and simply replaced the bass structure with basically identical but deviant pitch, velocity, and duration outputs (See Figure 12). (Version 1.7)

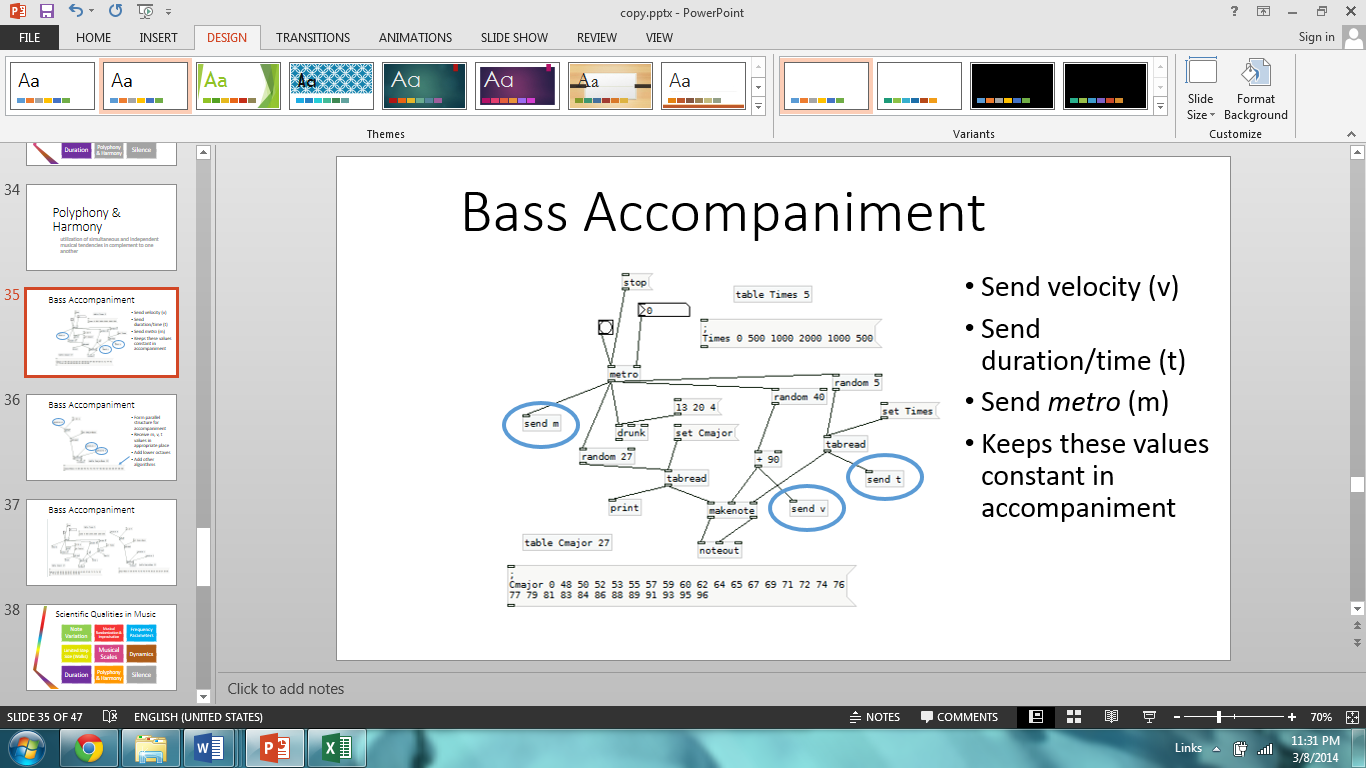


Figure 10: Send Values to Accompaniment

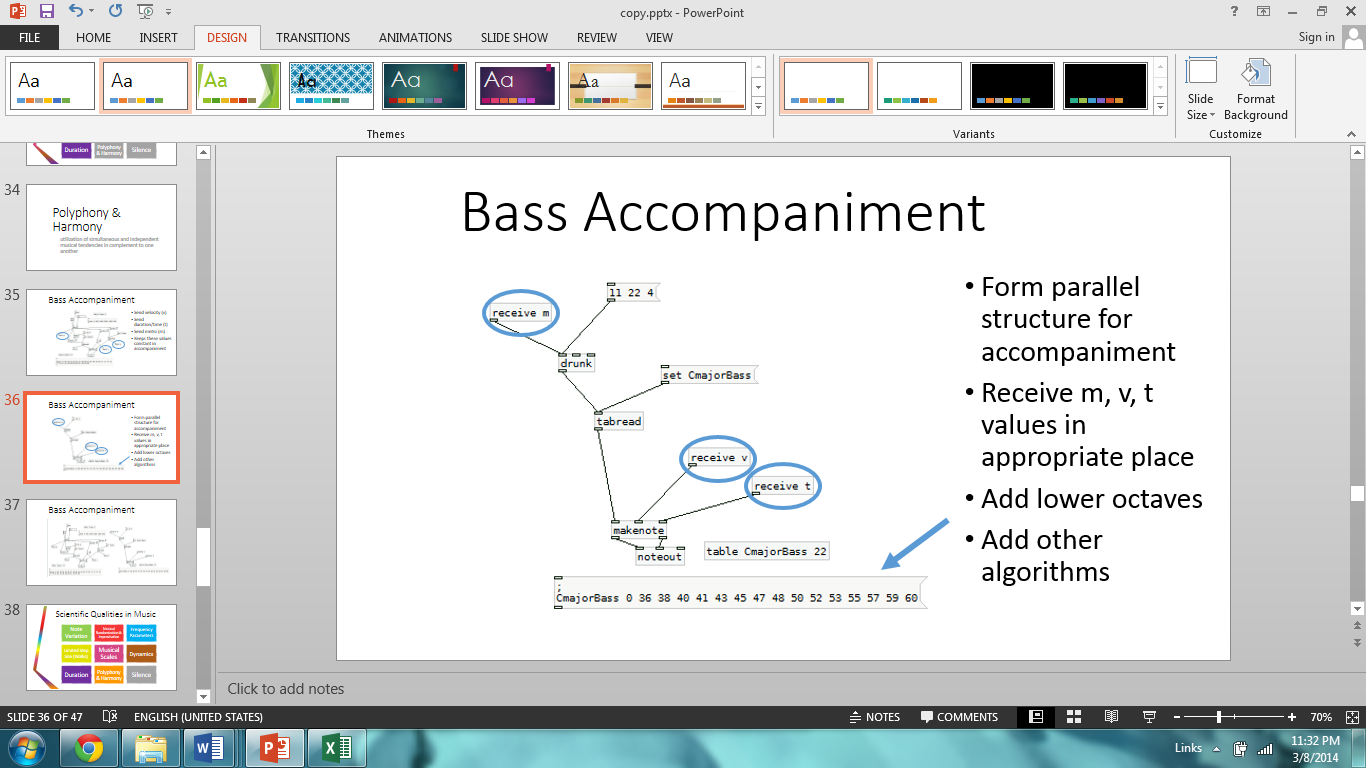


Figure 1: Receive Values in Parallel Structure

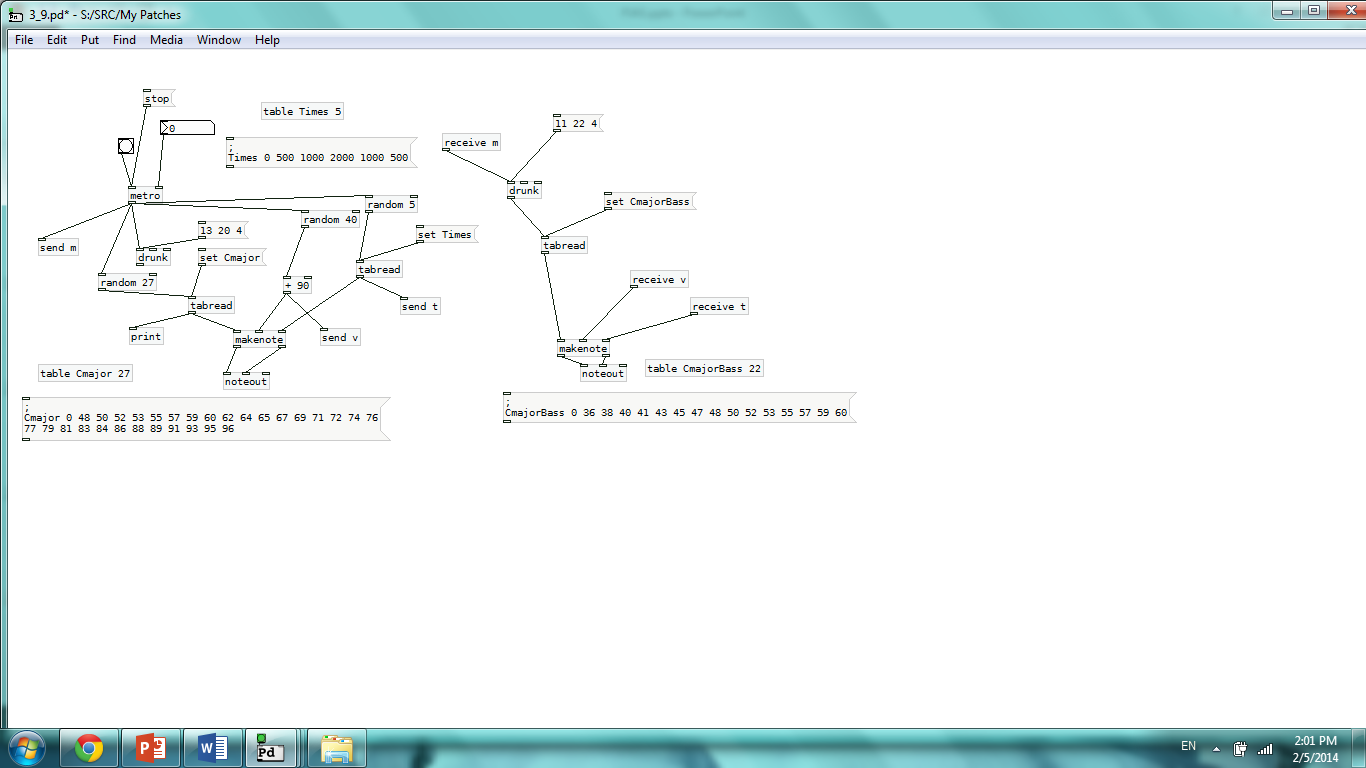


Figure 12: Bass Accompaniment

Ninth and last, silence is the use of breaks or rests in what was an endless stream of rolling notes prior to this engagement. Adding 0s into the duration table to randomly make notes have a length of 0 seconds placed this into action. In turn, this will pose as a rest. The ultimate ratio of 0:500:1000:2000 ms is 1:7:4:2 (See Figure 13 for experimental probability). As with the previous rhythms and tempos section, these were further fine-tuned to add probability aesthetics. Graph 2 shows how the probability aesthetics favor certain note durations over others resulting in notes of aesthetic length being more common. (Version 2.0)

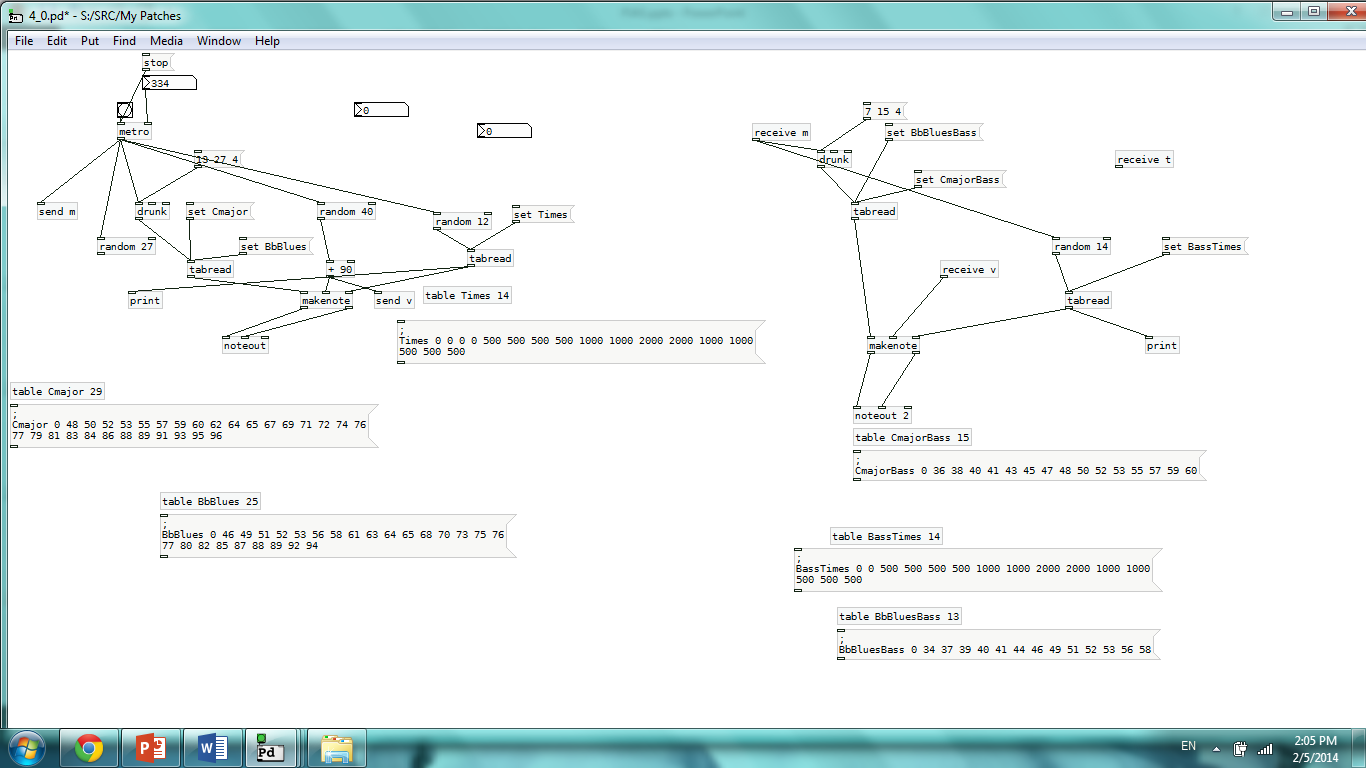


Figure 13: Silence with 0 ms Note Durations

**Results/Analysis:**

Quantitative Data:

During the various phases of implementing the aspects of algorithmic composition, it was ideal to keep periodic tabs on the data that amounted, specifically the 3 main inputs of *makenote*, pitch, velocity, and duration. This was done by compiling the pitch, velocity, and duration from the first 24 pulses in the given version at the time. All the versions of the patch (Pure Data program) are compiled in the data book in a changelog format with a complete version of the raw data. Tables 1-8 below show the first 24 pulses’ values in the 3 aforesaid categories using the C Major Scale.

Table : Numerical Prints for 24 Pulses (1.2)

|  |  |  |  |
| --- | --- | --- | --- |
| Note # | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms) |
| 1 | 69 | N/A | N/A |
| 2 | 46 |  |  |
| 3 | 52 |  |  |
| 4 | 21 |  |  |
| 5 | 4 |  |  |
| 6 | 8 |  |  |
| 7 | 56 |  |  |
| 8 | 97 |  |  |
| 9 | 77 |  |  |
| 10 | 64 |  |  |
| 11 | 79 |  |  |
| 12 | 52 |  |  |
| 13 | 87 |  |  |
| 14 | 37 |  |  |
| 15 | 95 |  |  |
| 16 | 41 |  |  |
| 17 | 20 |  |  |
| 18 | 33 |  |  |
| 19 | 29 |  |  |
| 20 | 67 |  |  |
| 21 | 82 |  |  |
| 22 | 11 |  |  |
| 23 | 35 |  |  |
| 24 | 11 |  |  |

Table : Numeric Prints for 24 Pulses (1.3)

|  |  |  |  |
| --- | --- | --- | --- |
| Note # | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms) |
| 1 | 51 | N/A | N/A |
| 2 | 77 |  |  |
| 3 | 89 |  |  |
| 4 | 52 |  |  |
| 5 | 62 |  |  |
| 6 | 90 |  |  |
| 7 | 66 |  |  |
| 8 | 85 |  |  |
| 9 | 57 |  |  |
| 10 | 62 |  |  |
| 11 | 85 |  |  |
| 12 | 75 |  |  |
| 13 | 95 |  |  |
| 14 | 89 |  |  |
| 15 | 97 |  |  |
| 16 | 96 |  |  |
| 17 | 58 |  |  |
| 18 | 64 |  |  |
| 19 | 60 |  |  |
| 20 | 59 |  |  |
| 21 | 77 |  |  |
| 22 | 65 |  |  |
| 23 | 81 |  |  |
| 24 | 70 |  |  |

Table : Numerical Data for 24 Pulses (1.4)

|  |  |  |  |
| --- | --- | --- | --- |
| Note # | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms) |
| 1 | 48 | N/A | N/A |
| 2 | 62 |  |  |
| 3 | 53 |  |  |
| 4 | 89 |  |  |
| 5 | 53 |  |  |
| 6 | 50 |  |  |
| 7 | 79 |  |  |
| 8 | 62 |  |  |
| 9 | 91 |  |  |
| 10 | 72 |  |  |
| 11 | 79 |  |  |
| 12 | 52 |  |  |
| 13 | 60 |  |  |
| 14 | 86 |  |  |
| 15 | 65 |  |  |
| 16 | 60 |  |  |
| 17 | 48 |  |  |
| 18 | 55 |  |  |
| 19 | 83 |  |  |
| 20 | 83 |  |  |
| 21 | 79 |  |  |
| 22 | 77 |  |  |
| 23 | 69 |  |  |
| 24 | 71 |  |  |

Table : Numerical Prints for 24 Pulses (1.4.1)

|  |  |  |  |
| --- | --- | --- | --- |
| Note # | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms) |
| 1 | 71 | N/A | N/A |
| 2 | 67 |  |  |
| 3 | 62 |  |  |
| 4 | 57 |  |  |
| 5 | 55 |  |  |
| 6 | 55 |  |  |
| 7 | 53 |  |  |
| 8 | 52 |  |  |
| 9 | 53 |  |  |
| 10 | 59 |  |  |
| 11 | 62 |  |  |
| 12 | 65 |  |  |
| 13 | 65 |  |  |
| 14 | 65 |  |  |
| 15 | 71 |  |  |
| 16 | 71 |  |  |
| 17 | 67 |  |  |
| 18 | 67 |  |  |
| 19 | 67 |  |  |
| 20 | 62 |  |  |
| 21 | 59 |  |  |
| 22 | 59 |  |  |
| 23 | 59 |  |  |
| 24 | 60 |  |  |

Table : Numerical Prints for 24 Pulses (1.5)

|  |  |  |  |
| --- | --- | --- | --- |
| Note # | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms) |
| 1 | 71 | 119 | N/A |
| 2 | 65 | 112 |  |
| 3 | 60 | 120 |  |
| 4 | 65 | 109 |  |
| 5 | 62 | 112 |  |
| 6 | 65 | 118 |  |
| 7 | 62 | 97 |  |
| 8 | 65 | 118 |  |
| 9 | 65 | 126 |  |
| 10 | 65 | 90 |  |
| 11 | 67 | 93 |  |
| 12 | 72 | 125 |  |
| 13 | 69 | 106 |  |
| 14 | 64 | 123 |  |
| 15 | 59 | 113 |  |
| 16 | 59 | 109 |  |
| 17 | 59 | 97 |  |
| 18 | 60 | 104 |  |
| 19 | 64 | 104 |  |
| 20 | 67 | 92 |  |
| 21 | 65 | 103 |  |
| 22 | 65 | 101 |  |
| 23 | 69 | 120 |  |
| 24 | 74 | 102 |  |

Table : Numerical Prints for 24 Pulses (1.6)

|  |  |  |  |
| --- | --- | --- | --- |
| Note # | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms) |
| 1 | 71 | 115 | 500 |
| 2 | 71 | 92 | 500 |
| 3 | 67 | 126 | 500 |
| 4 | 62 | 109 | 500 |
| 5 | 67 | 91 | 1000 |
| 6 | 65 | 109 | 1000 |
| 7 | 60 | 94 | 2000 |
| 8 | 55 | 103 | 1000 |
| 9 | 50 | 124 | 500 |
| 10 | 53 | 101 | 1000 |
| 11 | 55 | 117 | 2000 |
| 12 | 55 | 111 | 500 |
| 13 | 52 | 107 | 1000 |
| 14 | 57 | 93 | 500 |
| 15 | 57 | 97 | 2000 |
| 16 | 57 | 126 | 500 |
| 17 | 52 | 121 | 1000 |
| 18 | 55 | 98 | 500 |
| 19 | 55 | 106 | 500 |
| 20 | 52 | 120 | 500 |
| 21 | 52 | 91 | 1000 |
| 22 | 57 | 108 | 500 |
| 23 | 59 | 111 | 1000 |
| 24 | 60 | 90 | 2000 |

Table : Numerical Prints for 24 Pulses (1.7)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Note # | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms) | Bass Frequency (Midi) | Bass Velocity (Midi) | Bass Duration (ms) |
| 1 | 65 | 127 | 1000 | 52 | Same as left | Same as left |
| 2 | 64 | 111 | 500 | 52 |  |  |
| 3 | 93 | 90 | 2000 | 52 |  |  |
| 4 | 48 | 117 | 500 | 52 |  |  |
| 5 | 62 | 105 | 2000 | 52 |  |  |
| 6 | 84 | 92 | 2000 | 48 |  |  |
| 7 | 77 | 118 | 2000 | 43 |  |  |
| 8 | 53 | 116 | 1000 | 43 |  |  |
| 9 | 91 | 121 | 1000 | 45 |  |  |
| 10 | 77 | 111 | 500 | 40 |  |  |
| 11 | 55 | 90 | 500 | 45 |  |  |
| 12 | 64 | 101 | 500 | 43 |  |  |
| 13 | 60 | 127 | 1000 | 43 |  |  |
| 14 | 77 | 100 | 1000 | 40 |  |  |
| 15 | 55 | 99 | 500 | 40 |  |  |
| 16 | 93 | 93 | 1000 | 45 |  |  |
| 17 | 79 | 95 | 500 | 43 |  |  |
| 18 | 48 | 109 | 500 | 45 |  |  |
| 19 | 62 | 126 | 500 | 40 |  |  |
| 20 | 77 | 110 | 1000 | 40 |  |  |
| 21 | 83 | 116 | 1000 | 45 |  |  |
| 22 | 67 | 106 | 2000 | 50 |  |  |
| 23 | 74 | 124 | 500 | 45 |  |  |
| 24 | 76 | 104 | 2000 | 41 |  |  |

Table : Numerical Prints for 24 Pulses (2.0/2.0.1)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Note # | Pitch Frequency | Velocity (Midi) | Duration (ms) | Bass Freq. (Midi) | Bass Velocity (Midi) | Bass Duration (ms) |
| 1 | 69 | 97 | 1000 | 50 | 79 | 500 |
| 2 | 65 | 80 | 1000 | 45 | 98 | 1000 |
| 3 | 65 | 85 | 1000 | 50 | 64 | 2000 |
| 4 | 64 | 68 | 0 | 50 | 69 | 500 |
| 5 | 64 | 92 | 0 | 52 | 79 | 1000 |
| 6 | 62 | 67 | 500 | 52 | 86 | 500 |
| 7 | 62 | 95 | 1000 | 53 | 89 | 500 |
| 8 | 62 | 85 | 2000 | 53 | 93 | 2000 |
| 9 | 65 | 91 | 500 | 52 | 78 | 2000 |
| 10 | 64 | 82 | 500 | 50 | 78 | 500 |
| 11 | 64 | 90 | 1000 | 47 | 76 | 500 |
| 12 | 62 | 95 | 500 | 43 | 87 | 2000 |
| 13 | 62 | 91 | 1000 | 47 | 84 | 1000 |
| 14 | 57 | 82 | 2000 | 41 | 80 | 1000 |
| 15 | 52 | 71 | 1000 | 40 | 89 | 500 |
| 16 | 52 | 62 | 0 | 38 | 72 | 1000 |
| 17 | 52 | 69 | 1000 | 43 | 60 | 500 |
| 18 | 50 | 88 | 0 | 40 | 83 | 500 |
| 19 | 55 | 88 | 500 | 40 | 73 | 500 |
| 20 | 55 | 81 | 500 | 38 | 96 | 500 |
| 21 | 55 | 87 | 500 | 38 | 60 | 0 |
| 22 | 52 | 92 | 500 | 40 | 90 | 0 |
| 23 | 50 | 76 | 1000 | 40 | 87 | 2000 |
| 24 | 50 | 69 | 1000 | 41 | 94 | 2000 |

Table : Numerical Prints for 20 Pulses (Metro) (2.1b)

\*Version 2.1 beta only had one significant change from 2.0. This was the double metronome feature implemented to make changes in tempos. Therefore, all data besides the dynamic metro is omitted.

|  |  |
| --- | --- |
| Double Metro (Tempo in pulse/ms)\* | |
| 300 | 267 |
| 297 | 274 |
| 292 | 281 |
| 286 | 286 |
| 277 | 294 |
| 268 | 298 |
| 267 | 298 |
| 261 | 292 |
| 259 | 284 |
| 260 | 277 |

Qualitative Data:

Qualitatively, the music that results once the bang is initiated sounds recognizable musical. There is considerable depth, attributed to the polyphonic harmonies. The notes do not jump around drastically due to limitations on all three branches; no note is too high or low in frequency to hear, the dynamics do not add excessive or insufficient volume levels, and the durations use of probability aesthetics keep the music dynamic and less monotone. The harmony and melody coincide well (with little dissonance) as the use of the two same scales make it harmonically consonant. This is even further reiterated through the use of silence to add musical interest.

Qualitative disadvantages include the ‘computerized’ sound of the result. This is so because the music has systematic flow, rather than having inspired elements. The music seems like a continuous stream with less structure than human tunes with no recognizable riffs, chorus, or bridges. To add, the single instrument (piano) as well as the ‘cookie-cutter’ rhythms take away from imperfection and variance of human composition.

Calculations:

The only specific calculations in this project involved converting letter notes to frequencies, and then to midi values. Because frequency and notes share a logarithmic relationship the equations below in successfully convert them back and forth to make various scales.

Both are the same equation, just re-oriented in order to make going in the desired direction more direct. Midi uses its own midi scaling, while frequency uses Hertz. The 440 Hz, ‘a value’, is used a base bass index number in order to tune the rest of the notes. An example of converting C from frequency to midi is shown below. Prior knowledge of the pitch frequency of a C in the 5th octave is known for this calclulation.

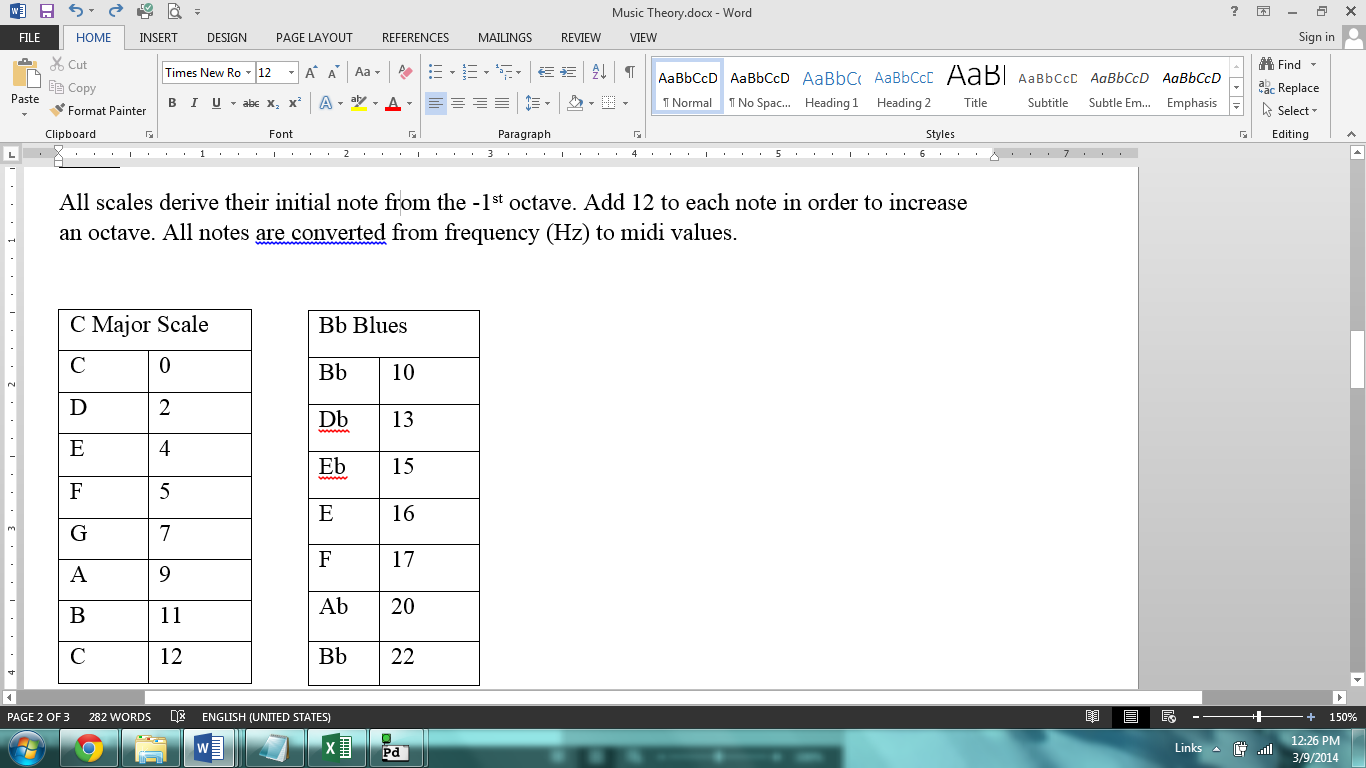
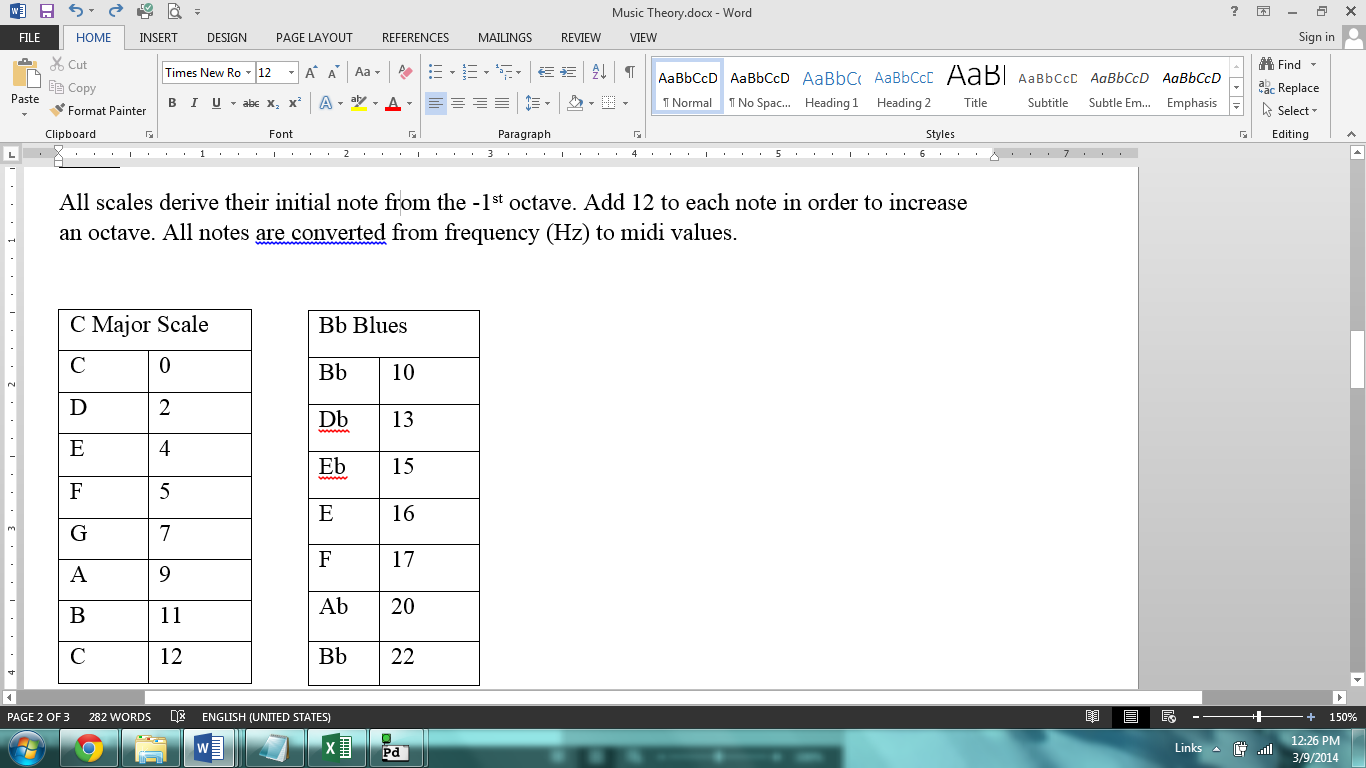
This process is repeated for each note. However, once it is made valid that the difference between each note step is exactly one midi value, it is possible to fill in the table with limited calculations, overlooking Hz altogether. Table 10 shows a full list of all midi notes (0-127).

C = 523.25 Hz

Table : Midi Note Conversions

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Octave | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 1 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| 2 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 3 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| 4 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |
| 5 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |
| 6 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 7 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 |
| 8 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 |
| 9 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |  |  |  |  |

Once this table was made, it was a straightforward process to convert various scales into midi format so they could easily be made into arrays and implemented into the patch. Tables 11-12 show an example of two scales that were converted and implemented into the program.

All scales derive their initial note from the -1st octave. Add 12 to each note in order to increase an octave. All notes are converted from frequency (Hz) to midi values.

Other calculations include implementation of algorithms. However, these are not mathematical calculations, but rather uses of raw probability and randomization with given parameters. The *random* object gives equal chance to all numbers, as does the *drunk* within its range. However, for the *drunk* object its probability parameters range over 9 values (4 left, 4 right, one in the middle), but once a new value is introduced, the range shifts.

**Discussion**

Based on the original goal, the final patch (2.0.1) was able to implement all identified 9 aspects of algorithmic composition. This success is illustrated in the previous methods section where each musical tendency was implemented chronologically and independently. The 9 aspects were: note variation, musical randomization, frequency parameters, limited step size, musical scales, dynamics, duration, polyphony, and silence. To add, the program presents collaboration among computer science, mathematics, algorithmic composition, and music theory, an aim of the original goal. The program successfully self-generates music with no human intervention.

Qualitatively, the patch successfully implemented algorithmic parameters, allowing for musical randomization, but at the same time limiting it to preserve aesthetics. All three branches of musical creation (pitch frequency, velocity, duration) are curbed based on their individual tendencies. Pitch Frequency is limited based on musical scales, limitation on jump ranges, and frequency parameters. Velocity makes use of dynamics, velocity parameters, and musical aesthetics to yield a limited result. Duration uses an array of times to choose certain note lengths more often then others based on probability for musical aesthetics. It also adds the effect of silence that somewhat keeps the music from sounding like an endless flow of computerized notes. Last, polyphony maximizes the depth of the music giving both a melody and harmony that back eachother with harmonic significance.

Although there are many qualitative successes, they are also accompanied by developing areas. One most notable is the fact that the musical sound lacks generic musical structure. This is exemplified in that the notes patterns do not repeat like they would in the riff or chorus of a song or even in a bridge. Too add, the perfection of the pitches, durations, and volumes is almost too exact and gives the music a computerized feel as there is little imperfection or dissonance involved. To add, most music uses additional layer beyond the simple dual-polyphony; multiple instruments and musical scales will aid in increasing depth even further. In addition, the application of Markov Chains may help. This would imply the use of previous note patterns to affect the next ones rather than the new patterns being (almost) completely independent of its preceding ones. For example, in a completely arbitrary way, perhaps each time the notes C and B fall next to each other, they should be followed by another C. Not only would this add more constant patterns in the music, but it would further limit the randomization in its aspects. For a more concrete example, a common theory especially in jazz improvisation is to ‘fall’ or end on certain notes. An algorithm could be implemented (as seen in other algorithmic composition projects) that makes every nth note fall on the nearest interval of the ‘fall’ note.

In general, there are a number of other future implications that could be implemented that follow the same basis that was used when creating this project: setting up 9 aspects and taking them on individually. These aspects would successfully further the developing areas mentioned above if applied (See Figure 14).

Figure 14: Additional Implementations Based on Observed Qualitative Deficiencies

Here are the potential applications in Figure 14 defined and explained along with possible methods of submission:

* **Determinacy & Repetition 🡪** repeat notes, riffs, or strings of notes multiple times
  + Markov Chains (see above)
  + Graphical Storage of Notes followed by Graphical Interpretations (during patch’s run)
* **Timbre 🡪** tone and quality of notes (instrumental variation)
  + Using midi plugins, using oscillators to develop new instruments
  + Genre specification – different musical tendencies, imperfections, and rhythms
* **Consonance & Dissonance 🡪** utilization of notes that do or do not blend well together
  + Adding deviant notes in scales (off by one)
  + Using two different scales that are *mostly* in harmonic touch
* **Imperfection** **🡪** swing, off-beat, and other aspects to make music *real*
  + Change the durations to 497 rather than 500 and even use half values (250)
  + Use double metronomes in a controlled way to add a swing groove
* **Musical Structure 🡪** organization of music in bars, beats, and tempos
  + Write down or record compositions
* **Progression** 🡪 feel of tempo, repeated choruses, beginning end, musical direction
  + Similar to determinacy and repetition

On the quantitative side of things, the data accumulated regarding the 3 main branches of data focus (pitch, velocity, duration) include further insight on the success of limiting utter randomization. For one, this is illustrated through the chronological progression of versions in the changelog. As the data tables in the progress through the various sequential versions show, more data categories were added as it progressed. This proves that as more implementations were made, more output values attribute to the complexity of the patch. Furthermore, as the chronology of the versions progressed, each column category became more and more limited based on laws of music theory and controlled parametric stochastic probability. Similar to Graph 1, Graph 3/4 juxtaposes the difference between utter randomization and the values provided by the 2.0.1 patch. Graph 3 shows that with complete randomization, the probability aesthetics are not present (too many long notes and silences) whereas with the controlled array, there are still silences and long (2000ms) notes, but its shows a central tendency. Graph 4 shows that completely random velocity and pitch jump around drastically with little to no musical aptitude, while the controlled randomization in patch 2.0.1 shows an almost steady horizontal line that is also centrally based. The pitch only hits notes in C major, while the velocity bounces around in a finite 40 value range so it is pleasing to the ear.

Comparison & Contrast to Other Algorithmic Composition Software

Another interesting method, is through comparing the results with those in other compositions. This can be done first with other algorithmic composition programs similar to this one. One of these is Wolfram Tones. Wolfram tones is unique because it not only algorithmically generates music, but also art based on that music. Some similarities between projects include limitations on pitches using one base scale. However, Wolfram proposes a few things that one could reverse engineer. Among these is instrumentation and genre specification, a key aspect of timbre that was aforementioned. To add, Wolfram Tones also utilizes graphical displays to reiterate the musical creation, another aspiration in the PD Patch.

Another program is Eastern Washington University’s musical algorithms page. Like Wolfram Tones, it is a web based program that uses various algorithms such as Markov Chains, Fibonacci Sequences, Pascal’s Triangle, and Chaos Algorithms. These algorithms share similarity to this project in that it limits frequency, velocity, and duration in order to create music. However, it uses completely different algorithms, mostly those based on unrelated mathematical models. Furthermore, I can learn from this through its use of systematic generation with Markov chains and its use of imperfection and humanization in Chaos and entropic algorithms.

Last, is the program known as cgMusic. This is by far the most complex of the three in that the product has the most depth and humanlike qualities in its product. The similarities with this project includes its use of limiting songs to one scale and giving full control over tempo. It expounds on timbre greatly because it has many layers of instruments and complementary harmonies/melodies. In addition, it uses progression exponentially as new layers of music come in and out to build up and strip down the tones. To add, it uses riffs and base lines that are constantly repeated to give it musical identity.

Comparison to Human Made Music

Aside from other algorithmic music software, it is also possible, although somewhat arbitrary and variant, to compare this patch’s result to that of human compositions. This is difficult because human compositions have certain ‘inspire’ elements, therefore it is necessary to use a holistic review of patterns rather than sheer numbers. However, it is possible to convert the sheet music of human compositions into the data value tables like that made previously from printing values from the patches. In this case, it is ideal to use sheet music in C Major scale only (as a precautionary control) because that is the scale used in the data tables.

Table : “Some Nights” By Fun. Sheet Music Converted into Numerical Values (First 24 pulses)

|  |  |  |  |
| --- | --- | --- | --- |
| Letter Note (Index Octave 5) | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms)(where quarter note equals 1000ms) |
| c | 72 | 80 | 1000 |
| c | 72 | 80 | 500 |
| c | 72 | 80 | 500 |
| e | 76 | 80 | 500 |
| g | 79 | 80 | 500 |
| g | 79 | 80 | 1000 |
| a | 81 | 80 | 500 |
| a | 81 | 80 | 500 |
| a | 81 | 80 | 500 |
| g | 79 | 80 | 500 |
| e | 76 | 80 | 1000 |
| c | 72 | 80 | 1000 |
| REST | 0 | 0 | 0 |
| a | 81 | 80 | 500 |
| a | 81 | 80 | 500 |
| g | 79 | 80 | 500 |
| e | 76 | 80 | 1000 |
| d | 74 | 80 | 750 |
| d | 74 | 80 | 500 |
| d | 74 | 80 | 500 |
| d | 74 | 80 | 500 |
| REST | 0 | 0 | 0 |
| REST | 0 | 0 | 0 |

Table : "Rondo in C Major Op. 51 No. 1" by Beethoven Converted in Numerical Data for the First 24 Pulses

|  |  |  |  |
| --- | --- | --- | --- |
| Letter Note (Index Octave 5) | Pitch Frequency (Midi) | Velocity (Midi) | Duration (ms)(where quarter note equals 1000ms) |
| c | 72 | 70 | 1000 (stacatto) |
| c | 72 | 70 | 1000 (stacatto) |
| c | 72 | 70 | 500 |
| b | 71 | 70 | 500 |
| a | 69 | 70 | 500 |
| b | 71 | 70 | 500 |
| c | 72 | 70 | 1000 |
| c | 72 | 70 | 750 |
| e | 76 | 70 | 250 |
| g | 79 | 70 | 2000 |
| f | 77 | 70 | 500 |
| a (high) | 81 | 70 | 1000 |
| g | 79 | 70 | 250 |
| f | 77 | 70 | 250 |
| e | 76 | 70 | 500 |
| g | 79 | 70 | 1000 |
| f | 77 | 70 | 250 |
| e | 76 | 70 | 250 |
| d | 74 | 70 | 250 |
| e | 76 | 70 | 250 |
| f | 77 | 70 | 250 |
| e | 76 | 70 | 250 |
| g | 79 | 70 | 250 |
| f | 77 | 70 | 250 |

Table : Numerical Print of First 24 Pulses (2.0.1)

|  |  |  |  |
| --- | --- | --- | --- |
| Note # | Pitch Frequency | Velocity (Attack) | Duration (ms) |
| 1 | 69 | 97 | 1000 |
| 2 | 65 | 80 | 1000 |
| 3 | 65 | 85 | 1000 |
| 4 | 64 | 68 | 0 |
| 5 | 64 | 92 | 0 |
| 6 | 62 | 67 | 500 |
| 7 | 62 | 95 | 1000 |
| 8 | 62 | 85 | 2000 |
| 9 | 65 | 91 | 500 |
| 10 | 64 | 82 | 500 |
| 11 | 64 | 90 | 1000 |
| 12 | 62 | 95 | 500 |
| 13 | 62 | 91 | 1000 |
| 14 | 57 | 82 | 2000 |
| 15 | 52 | 71 | 1000 |
| 16 | 52 | 62 | 0 |
| 17 | 52 | 69 | 1000 |
| 18 | 50 | 88 | 0 |
| 19 | 55 | 88 | 500 |
| 20 | 55 | 81 | 500 |
| 21 | 55 | 87 | 500 |
| 22 | 52 | 92 | 500 |
| 23 | 50 | 76 | 1000 |
| 24 | 50 | 69 | 1000 |

Comparing the data between the human-made compositions “Some Nights” and “Rondo” with the data resulting from the latest patch version 2.0.1 results in both similarities and differences. The aspects that favor the patch’s results include the pitch values. The “Some Nights” especially (but also Rondo) pitch values are confined only to the C Major scale, much like the Pure Data Patch 2.0.1. To add, the *drunk* object seems to emulate an almost replica to the pattern presented in “Some Nights” and “Rondo” as the *drunk* object repeats various notes multiple times and when it moves away from a note, it does not do so for more than four places away. The same pattern is evident in the human composition (Table 11).

Heavy contrasts are found in the velocity column. Table 11’s velocity stays completely stagnant throughout the first 24 pulses of the song. This probably illustrates the fact that the patch’s velocity is changing too much for almost no reason. Although this does add imperfection, perhaps a solution would be to create an extensive array with many elements. All the bordering elements should be the same for a long period only spotted with some dynamics. However, there should be a point where the numbers drastically change in value, but continue to do so for another equally extensive period of time. If a *drunk* object was connected to this it would shift from one velocity to another based on periods of time, not just pulses.

Another contrast, although minute, came in the duration’s column. One of the biggest future fixes should come with adding in durations of 750 or 250 ms in order to add the lengths of sixteenth or dotted notes. Other than that, the probability aesthetics quite successfully predicted and emulated the distribution of durations for an actual song.

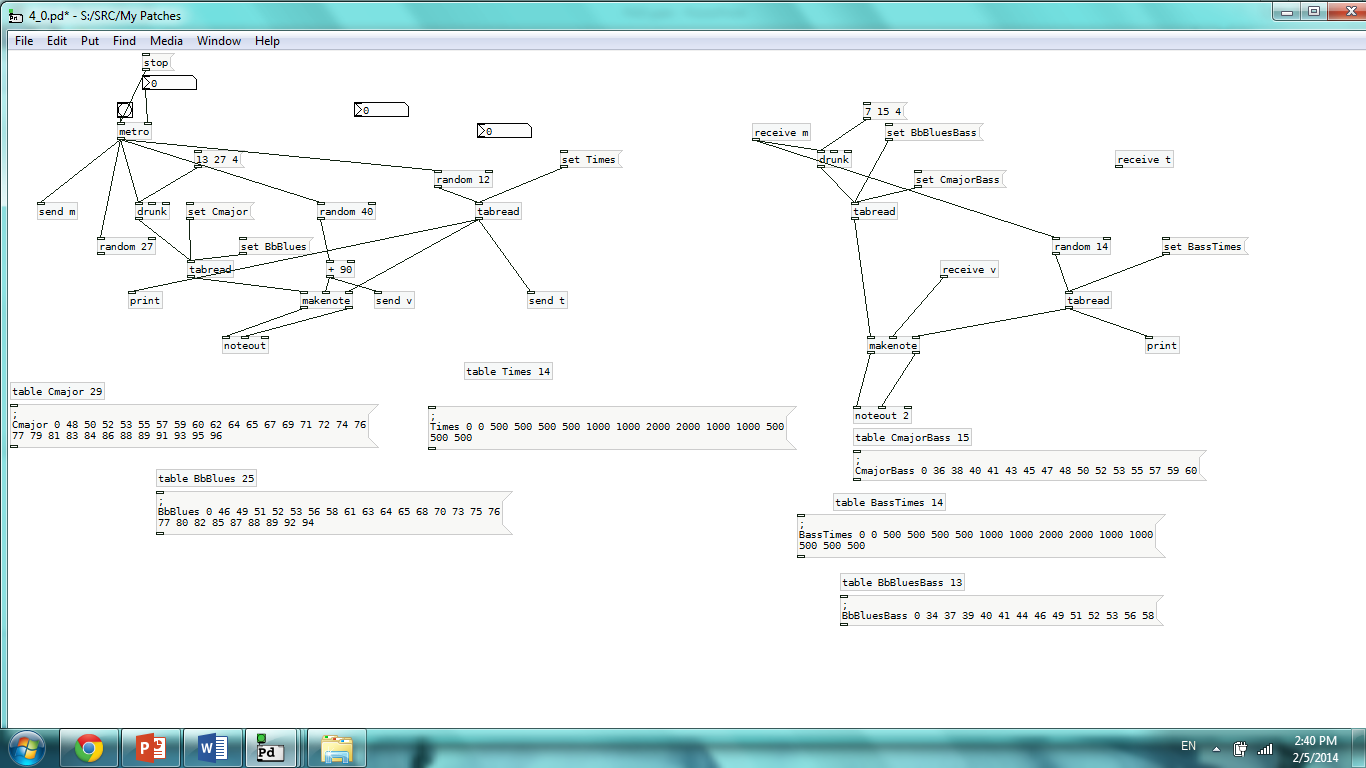


Figure 15: Final Product (Patch 2.0.1)

**Conclusion:**

In conclusion, the results of this project successfully validated the possibility of original goal. The original goal was to create a self-sufficient computer program, which self-generates music, through implementation of various concepts of algorithmic composition, computer science, music theory, and mathematics. In this process, the goal was to make a musical output that was random, but limited in randomness through utilization of parametric algorithms. In these aspects, the project was successful because it synthesized a product that utilized 9 different methods of scientific qualities to emulate in music. These 9 methods illustrate success through their numbers and the extent to which they were successful. To add, the fields of algorithmic composition, music theory, and computer science were all combined in a systematic way to create a product that embodies each of their merits and interests.

Developing areas were also identified in ‘humanizing’ the final composition through use of musical structure and imperfection that is attributed to human execution. These developing areas have been identified and paired with proposals for improvement or future work. To add, the final product has been juxtaposed to both similar algorithmic compositional programs as well as human compositions to notice their differences and similarities both in quantitative and qualitative fields. Based on the data, an accurate statement is to say that as the factors that contribute to randomization of the next value go up and the limitations on that next value increase, the quality of the music increases as well. This means that as the algorithms further envelop the randomization and it becomes more systematic than random, the musical tendencies go up. This means that the timespan of this project should be essentially infinite as more and more combinations, permutations, and variations of algorithms are experimented with to create the ‘ultimate musical machine’.

The practical uses of this program are almost infinite. Among them entails mentioning the trend of computerization in the music industry these days. Much of the music produced and popularized today is becoming less and less instrumental and more computerized. Perhaps the creation of computer software that try their hand in composing music will not only advance this industry through introduction of various new algorithms to implement, but revolutionize it in a way that perhaps the new musical genius will not come from people using computers to make music, but rather people telling computers *how* to make music. After this, it is as easy as sitting back and ‘prooflistening’ the product. To add, it is quite possible that the use of music theory in order to patternize music will lead to advancements in music theory as well. It is quite possible that new patterns will arise as computers compute every musical possibility within a given set of parameters. Moreover, these algorithmic composition programs may be used recreationally. Because each new product is unique from the last, it yields the opportunity to listen infinitely, perhaps in replacement to hearing the same songs repeatedly on the radio. Perhaps, it could inspire human made music through introduction of a riff or pattern than can be utilized on a lrger scale without brain-racking thought processes and creative blocks. In all, with almost infinite practical uses, the future of algorithmic composition is bright.

**References/Bibliography:**

*Algorithmic*. Christopher Ariza, n.d. Web. 18 Nov. 2013. <http://algorithmic.net>.

Antonoff, Jack, et al. *Some Nights*. N.p.: Alfred Music, 2012. Print.

Biedrzycki, Maciej. "cgMusic - Can computers create music?" *Casual Games Harmony*. N.p., n.d. Web. 9 Mar. 2014. <http://codeminion.com/blogs/maciek/2008/05/cgmusic-computers-create-music/>.

Edwards, Michael. "Algorithmic Composition: Computational Thinking in Music." *Communications of the ACM*. ACM, n.d. Web. 8 Mar. 2014. <http://cacm.acm.org/magazines/2011/7/109891-algorithmic-composition/fulltext>.

Essl, Karlheinz. "Algorithmic Composition." *University of Washington*. University of Washington, n.d. Web. 18 Nov. 2013. <http://wiki.dxarts.washington.edu/sandbox/groups/general/wiki/9e11b/attachments/0bead/K.%20Essl%20-%20Algorithmic%20composition.pdf?sessionID=a3f30c5d2db2af680e2cf41f3627b42c99b9bf5f>.

Harnum, Jonathan. "Basic Music Thoery." *Monarch Knights*. Monarch Knights High School, n.d. Web. 18 Nov. 2013. <http://www.monarchknights.com/teacherwebpages/halladay/documents/BasicMusicTheory1ed\_000.pdf>.

Kreidler, Johannes. "Programming Electronic Music in Pd." *Pure Data*. N.p., n.d. Web. 14 Feb. 2014. <http://www.pd-tutorial.com/english/index.html>.

Maurer, John A. "A Brief History of Algorithmic Composition." *Stanford*. Stanford University, n.d. Web. 8 Mar. 2014. <https://ccrma.stanford.edu/~blackrse/algorithm.html>.

Maurer, John A., IV. "A Brief History of Algorithmic Composition." *Center for Computer Research in Music and Acoustics*. Standford, n.d. Web. 18 Nov. 2013. <https://ccrma.stanford.edu/~blackrse/algorithm.html>.

Mungan, R. Yagiz. "Introduction to Algorithmic Composition and Pure Data." File last modified on July 2012. PDF file.

Nierhaus, Gerhard. *Algorithmic Composition: Paradigms of Automated Music Generation*. New York: Springer Wein New York, 2009. *Google Books*. Web. 18 Nov. 2013. <http://books.google.com/books?id=jaowAtnXsDQC&printsec=frontcover&dq=algorithmic+composition&hl=en&sa=X&ei=H9uKUsCzBq\_gsATtroKoDg&ved=0CC0Q6AEwAA#v=onepage&q=algorithmic%20composition&f=false>.

Puckette, Miller S. "Theory and Techniques of Electronic Music." *University of California, San Diego*. UC San Diego, n.d. Web. 14 Feb. 2014. <http://msp.ucsd.edu/techniques/latest/book-html/>.

Smith, Stuart. "Jazz Theory." *Computer Science Department of the University of Massachusetts*. Rev. 4th. University of Massachusetts, n.d. Web. 18 Nov. 2013. <http://www.cs.uml.edu/~stu/JazzTheory.pdf>.

Van Beethoven, Ludwig. *Rondo in C Major Op. 51, No. 1*. N.p.: n.p., n.d. Print.