CAPSTONE ECE 492
Group #8

EVOLUTION OF MUSIC

[Fig. 1]

—

Andersen, Stephen | Ingram, Lee | Peard, Jonathan | Schuman, Aaron

SUMMARY

A song will be generated that is passed to the critic algorithm. Using a grade as feedback, the best patterns will propagate to the new generation.

ABSTRACT

The goal of this project is to see if musical theory can be implemented with a genetic algorithm to simulate an “Evolution of Music”, where after many generations, songs that are both unique and acceptable are generated. The Genetic Algorithm (GA) selects many sequences of notes to generate songs. A hardcoded Critic Algorithm will accept the song and will grade it based on musical theory. That score is then sent back to the GA. The GA will interpret the score by adjusting the probabilities of propagation for each song. After many iterations, the GA will find some tracks that could be the basis of a more complete song; and although the GA requires many iterations to create something that is objectively good, the songs it has produced show potential. The Critic Algorithm (CA) identifies bad musical practices and gives a song demerits when such a practice is found. In testing with several well-known songs, the critic seems to prefer the only classical piece that it was given. The multi-voice synthesizer reads the songs and converts them into audio, with several different instruments playing simultaneously; the synthesizer functions according to specifications.

Table of Contents

Functional Requirements

1. Everything is located on a computer.
2. A sequence of random notes are generated by an AI.
    - A random seed is used to determine the starting notes.
    - The AI is a Genetic Algorithm (GA).
3. Demerits for tone sequences are assigned based on musical rules.
    - Accomplished via a Critic Algorithm; based on tone, timing, and phrasing patterns.
4. Changes are applied to the music based on the severity of the grade.
    - Genetic Algorithm makes random changes to the note sequences.
    - This will repeat, for a user-specified number of iterations.
5. Audio files are produced by the synthesizer.
    - Each track of a song (See Software Design) is played by a different instrument
    - Each instrument is created via Frequency Modulation[1]
6. A User Interface displays statistics pertaining to all past generations
    - Statistics can be updated with a press of a button
    - Genetic Algorithm parameters can be updated in real-time

All of the above requirements are completely satisfied.

Design & Operation

1. AI Shell

a. Input:

   i. Prev Songs Grade

b. Output:

   i. New Song [see Data Structures 1 in Software Design]

c. If a song scores 0 and the parent is null, we toss it out and create a new randomly generated song. This way, we can initialize the sequence by giving the AI Shell a score of 0.

d. When mutate() is called, both the current song and the current song score are passed to the GA.

e. The shell then calls pause() to suspend execution until the GA(Python) completes.

2. Genetic Algorithm (GA)

a. Input:

   i. The C Program's ID

   ii. The random seed

   iii. Song score via the file, ex. "main_py_input"

b. Output:

   i. A Generation of Songs via the file, ex. "main_py_output"

c. Stored within the AI Shell.

d. For the first iteration, a 'population' of songs will be randomly generated, in order to establish a starting point for the Genetic Algorithm.

e. In the context of the GA, each instrument (a collection of notes, of fixed size) will be referred to as a Chromosome.

f. The GA will pass the new generation of songs to a Critic Algorithm, which will then assign each song a grade, and pass the grades back to the GA.

g. The GA will then randomly select a subset of the songs (with a significant bias produced by the grade of each song) that will pass on their genetic information to the next generation of songs.

h. During song reproduction the tracks of each song are combined into one song, tracks with the same id, in this new song, will be split in half and then the two halves will be recombined to form a new track, which will replace the two parent tracks that made it.

i. Then the GA sends the continue signal, making the AI Shell resume.

3. Critic Shell

a. Input:

   i. A Song [see Data Structures 1 in Software Design]

b. Output:

   i. A Score based on the inputted Song.

c. The Critic Shell contains the Critic Algorithm.

d. The Critic Shell will pass the song into the Critic Algorithm, and will wait to receive a score. Then the score will be passed back to the composer algorithm

4. Critic Algorithm (CA)

a. Input:

   i. A Song [see Data Structures 1 in Software Design]

b. Output:

   i. A Score based on the inputted Song.

c. Stored within the Critic Shell.

d. Analyzes the song assuming a perfect score of 0, and adds demerits to the score for violations of rules defined by musical theory.

   i. Uses weighted demerits based on the severity of the fault. [8]

   ii. Analysis of the song is performed piecewise in two dimensions as follows.

      1. Each note is compared to several adjacent notes in the same track.

      2. Each note is compared to the notes in other tracks playing at the same time.

5. Synthesizer

a. Input:

   i. Song sample file [.txt file].

b. Output:

   i. Audio waveform [.wav file]

c. Plays multiple tracks and instruments.

d. Can play many notes at the same time.

6. Design Diagrams

a. The Hardware Block Diagram is available at [Appendix, Hardware Documentation, 1].

b. The Data Flow Diagram is available at [Appendix, Source Code, 1].

Software Design

1. Data Structures

a. Song: A series of Tracks for a number of instruments.

   i. Song ID

   ii. Tempo

   iii. Number of Tracks

   iv. Array of Tracks, where a Track is played by a single instrument.

b. Track: A series of Notes throughout ten measures played by a single instrument.

   i. Instrument Number

   ii. Number of Notes

   iii. Array of Notes

   iv. Volume: The amplitude of each Note.

c. Note: Each note can be viewed as a structure containing the following fields,
    i. Pause Time: The space between a Note and the previous note.
    ii. Hold Time: The time the Note is played for.
    iii. Tone: The frequency played by the instrument.

d. The Software Block Diagram is available at [Appendix, Source Code, 2]

2. Python Format

a. The python format involves dividing the hold time and pause times into two and mutating each to increase the chance of mutation. This will increase the chances that small adjustments will be made to the timing of the song.

Pause Time	Hold Time	Tone	Hold Time	Pause Time

[Fig. 2] (The Pause and Hold Times indicated in the figure above are the Pause and Hold Times in the standard format divided by two and placed at opposite sides of the Gene)

3. User Interface

a. The User Interface displays several different graphs of useful data per generation of songs.
    i. Average Score
    ii. Change in Average Score
    iii. Acceleration of Average Score
    iv. Minimum Score
    v. Maximum Score
    vi. Diversity
    vii. Variance

b. The User Interface can also adjust some of the Genetic Algorithm's parameters.
    i. Chromosome Addition Probability
    ii. Chromosome Deletion Probability

iii. Mutation Probability

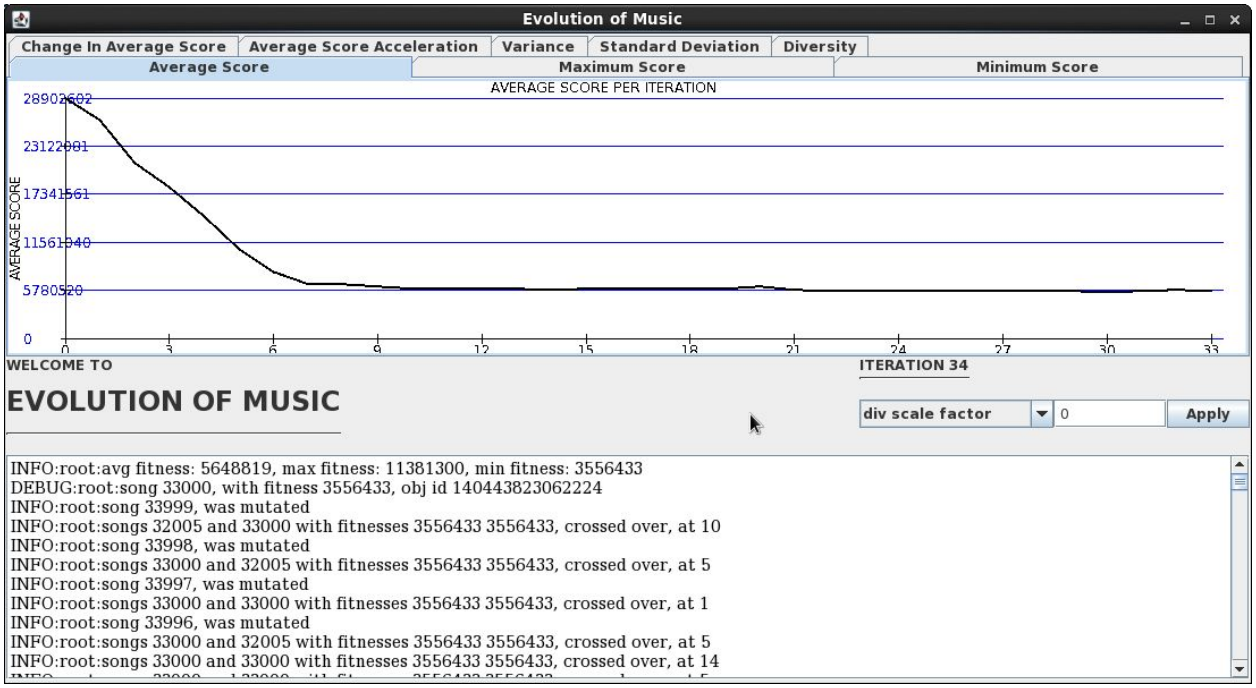
iv. Mutation Step Size

v. Diversity Modifier

vi. Score Modifier

vii. Min/Max Tempo

c. The User Interface takes the score data from “ave_fitness_graph” file and modifies the “pyth_main.config” file.



[Fig. 3] The Graphical User Interface

4. Genetic Algorithm

a. Generates a random set of initial songs.

i. New songs are generated with one track.

ii. The number of songs generated is determined by a config file.

b. After songs have been graded they are assigned a probability based on the number of demerits.

i. First the songs are sorted by score.

ii. Then each song is given a crossover probability, which is given by the following formula: pc * (1 - pc) ^ i, where i is the songs index in the sorted list, and pc is the probability of the best song in that generation.

c. Songs are then combined in an operation called crossover.

i. Where songs have common tracks, those tracks are split in half, at an acceptable crossover point, and combined, during song crossover.

ii. Where songs do not have common tracks, the outlying track is added, as is, into the resulting song.

d. The song that results from crossover is then mutated.

i. The first step of mutation is create something called a delta mask, which is a list of integers, which is randomly generated and has one value for every relevant field in the song data structure.

ii. Not every song is mutated, there is a field in the config file that the GA relies on, which controls the mutation probability for all songs.

iii. Secondly, how much a field in the song data structure (the GeneticSong) is specified by the config file and is referred to as the step size.

5. Synthesizer

a. Converts a song with the previously mentioned song structure to a .wav file.

i. Finds the longest track length (in beats).

ii. Creates an empty array proportional to the sampling rate, tempo, and the beat length of the longest track.

iii. For every increment along the song, the increment being the size of the sampling period, the average value of all of the waveform values that should be playing at that beat are taken and stored in the final song array.

6. Critic Algorithm's Musical Rules

a. The critic has three major groupings of rules.

   i. Consonance: How well two notes sound when played at the same time.

      1. Two notes played at the same time in different tracks should obey the ideal frequency ratios of western music.

      2. A note should not be excessively high.

   ii. Tempo and Timing: An evaluation of the beat and rhythm of the song.

      1. A note should start on a beat which is a multiple of its hold time.

      2. Each measure should follow the beat of the first measure.

   iii. Phrasing and Resolution: How cohesive sections of the song are, and how well the whole song is tied together.

      1. A note should be within an octave of the previous two notes.

      2. A note should be held for approximately as long as the preceding note.

      3. The song should end in the same octave it began.

      4. The song should end on a longer note.

      5. The song should end on a down-step.

b. Track Bias

   i. In order to ensure that consonance rules are applied, we assign demerits when a song has too few tracks.

c. Data flow

   i. The critic takes a song given to it by the AI Shell.

   ii. After grading the song, the grade is returned to the AI Shell.

7. Overhead

a. Manages the user's input, the GA, and the Critic to produce songs every time it is ran.
    i. The user must set the number of iterations to run and may set the random seed, the option to print to terminal, suppress the UI, and/or continue from a previous run of the program.

Test Plan

1. The AI Shell will be written in C++.
    a. For Example:
        i. > song1 = ai_shell(0); // Score of 0, but still uninitialized
        ii. if (song1 != null) then Song1 has been initialized
        iii. > song2 = ai_shell(0); // Score of 0, PERFECT
        iv. if (new_max_grade <= prev_max_grade) // either Song 2 is not Song 1 and the song has improved, or Song 1 is still the better candidate
2. The Critic Shell will be written in C++.
    a. For Example:
        i. > grade = c_shell(song)
        ii. > grade = c_shell(create_empty_song())
        iii. if (grade == 0) // the song was perfect
    b. To test the correctness of our algorithm, we translated several well-known songs into our data format and compared the scores given by our critic, to our own interpretation of the quality of each piece of music .
        i. Canon in D[18]: 528120
        ii. Super Mario Theme[19]: 4431730
        iii. Star Wars Theme[20]: 6305559
        iv. Hot Cross Buns[21]: 9789485

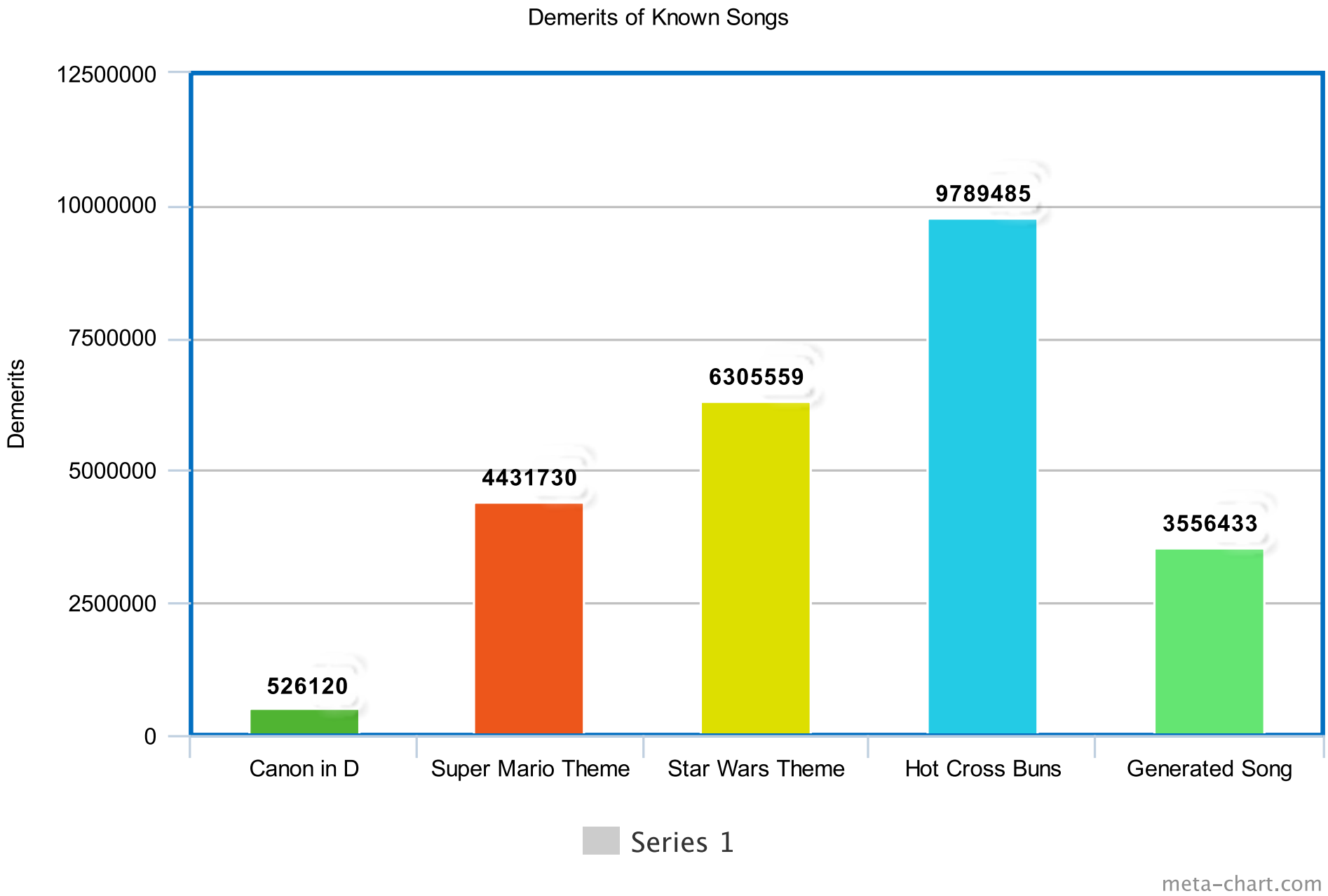
    In order: Canon in D < Super Mario Theme < Star Wars Theme < Hot Cross Buns

3. The GA will be written in Python 3, and as such will be tested using Doctest [11]. Each function will be tested to ensure correct functionality both individually and in the relevant class.
    a. For Example:
        i. >>> gene = NoteGene(4,6,10)
        ii. >>> gene[0] = 1
        iii. >>> gene[1] = 2
        iv. >>> gene[2] = 3
        v. >>> gene[0]
        vi. 1
        vii. >>> gene[1]
        viii. 2
        ix. >>> gene[2]

x. 3
xi. >>> gene[3]
xii. Traceback (most recent call last):
xiii. ...
xiv. IndexError: Index out of range for type NoteGene

b. An input is indicated by '>>>' and the expected result (if any) is on the line below, and is not preceded by '>>>'.
c. Gene is a data structure that contains the hold, pause, and tone values, and is referred to as Note in the rest of the documentation

4. The synthesizer is tested by synthesizing different songs from a test program on the computer. The correctness of the tones played will be determined using an electronic tuner.
    a. Polyphonism is determined by playing one note on a single track and another on a different track. The waveform is exported and viewed. If the notes playing at the same time are equal to the addition of the waveforms separately.
    b. Hold time will be verified using a stopwatch, pause time will have to be verified subjectively, as it determines how the note ends.
    c. The synthesizer is then ran with popular songs, which allows users to determine subjectively that it functions.


Demerits of Known Songs
12500000
10000000
7500000
5000000
2500000
0
Demerits
526120
4431730
6305559
9789485
3556433
Canon in D
Super Mario Theme
Star Wars Theme
Hot Cross Buns
Generated Song
Series 1
meta-chart.com

Results of Experimentation and Characterization

- The genetic algorithm (GA) will take a constant time to compute every new generation of songs. All tests will be run with a random seed of 0.

#iterations	GA Average Times
1	0.6
100	65.3
1000	633.7

*for 50 songs in a generation

*average was taken over ten samples

GA Iterations	% Decrease in Demerits (for min score, when compared to the min score from the initial iteration)
1	21.6
10	34.5
100	37
500	38.5
1000	38.5
1500	39
2000	41.6
3000	48.7
4000	49.3
5000	49.3

Safety

We have deemed this project to be safe for use on a minimum requirements computer while decibels from the speakers are less than 70 dB. This would also comply with Edmonton's bylaw 14600[15].

- Max Voltage 120 V, Canadian Socket
- Max Decibels 70 dB, normal voice level 1ft away [13]
- Power for speakers: 25 W [23]
- Battery Capacity for laptop: 32 Wh [24]

Regulatory and Society

- According to Bylaw 14600[15], "A person shall not cause or permit any sound exceeding 75 dB(A), as measured at the property line of a property zoned for use other than residential, between 7 a.m. and 10 p.m." As a result, we shall limit speakers have a max volume of 75dB and we shall prefer the volume of a normal human voice, 70 dB[13]. (It shall be nowhere near the property line)
- It is possible for a portion of the system to be hacked through one of the desktop computers that will be running the GA, but this would be, at worst, a nuisance.
- max sound frequency 3502 Hz

Environmental Impact

The impact of this project is:

ROHS Status:

- Consumer grade computer - RoHS compliant
- Speakers - RoHS compliant

None of these elements contain hazardous materials. All used materials are RoHS compliant.

Sustainability

Power consumption for:

- Speakers
    - Uses 25 W of power [23]

- Consumes 0.6 kWh per day of continuous use
- 219 kWh per year of continuous use

- Computer [Laptop]
  - Consumes 32 Wh per 5 hours of continuous use [24]
    - 0.160 kWh per day of continuous use
    - 58.44 kWh per year of continuous use
- Average Use
  - The speakers are only used when the songs are completely generated
    - Approximated as 1 hour of use per day of continuous project use
    - Speakers consume 0.025 kWh per day
      - Total project consumption per day is 0.185 kWh
      - Total project consumption per year is 67.57 kWh

CO2 from coal power:

Based on a CO2 output of 2.18 lb/kWh, we would output 147.3 lb of CO2 per year of continuous use.

Needed Area of Solar Panels in Edmonton

$$Assuming\ PR(\ Performance\ ratio) = 0.75,\ and\ r(Solar\ panel\ yield\ \%) = 0.2\ \%$$

Annual average solar radiation in Edmonton:

[17]$$P = 3.57\ kWh/m^2day * 365\ days/year = 1303.05\ kWh/m^2year$$

Eq:

$$P = A * r * H * PR$$

$$A = \frac{67.57\ kWh}{1303.05\ kWh/m^2year * 0.75 * 0.2} = 0.3457\ m^2\ of\ solar\ panels\ needed\ to\ power\ this\ system$$

Bill of Materials

Part Name	#	Price($CAD)	Supplier	Operating Statistics
Speakers	1	69.99	Logitech [23]	Requires standard AC plugin and 25 W of power
Computer	1	529.99	Lenovo [24]	Requires standard AC plugin and contains a 32 Wh battery
Ubuntu	1	0	Ubuntu	700 MHz processor (about Intel Celeron or better) 512 MiB RAM (system memory) 5 GB of hard-drive space (or USB stick, memory card or external drive but see LiveCD for an alternative approach) VGA capable of 1024x768 screen resolution.
Total	=	599.98		

Note: Can be used on OS X (Ubuntu is cheaper)

Available Source

Opencog, an open-source program that manages hypergraphs, was not used due to time constraints.

OpenCog (AGI) [5]:

- hosted by the OpenCog Foundation
- Source Size: 20.8 MB
- Compiled Size: 2.1 GB
- Performance: 4GB RAM
- Resource Requirements: 4GB RAM

The Laser Harp Synthesizer was not used due to difficulties in implementing hardware.

Lookup Table Audio Synthesizer [10]

- from ECE 492 Winter Capstone 2015
- Source Size: 74.2 MB
- Could not compile

The make_wav.cpp code was taken from [22].

- Hosted by gasstationwithoutpumps Wordpress
- Source Size: 3.2 kB
- Compiled Size: 8.7 kB

Datasheet

Operating System Requirements	Any OS that is bash-enabled & python3-enabled
Minimum Required CPU clock speed	8712 Hz/8.7kbps
*Memory used per generation	21.3 kB
**Compiled Size of Project	98.3 MB

*assumed max track count of 10 and note count of 160

**size of project after compiling C code and generating Python crossover file (assumed 1000 songs, 10 tracks, 160 notes)

Formula for calculating runtime memory use in Bytes:

*Bytes of memory used = songsPerGen(15+[maxTracksPerSong(8+[maxNotesPerTrack*7])])*

Process for calculating required cpu speed:

[8712 bits/sec required for song]=[99 addresses/note_cycle required for song][11 bits/address]*[8 note_cycles/sec]*

**note_cycle = The minimum amount of time it takes to play a sound.*

***addresses/note_cycle = The number of instruments playing at any given time*

[11 bits/address] = [7 bits for tones] + [4 bits for instruments] (85 tones and 10 instruments)

*[8 note_cycles/sec] = [2 notes/sec] * [4 note_cycles/note]*

(1/16th notes at 120bpm is the fastest we allow)

Formula for calculating use in Bytes:

*Bytes of memory used: numberOfSongsInGeneration(5+[numTracks(4+[numNotes*18])])*

Background Research

Instrument Tones

[1] provides examples on how to simulate instrument tones. It specifies the frequency for starting to play, holding and ending the instrument tone. Instruments are simulated with a Frequency Modulator and various amplitude and timbre envelopes. If interested, reading John Chowning's paper[1] is highly recommended.

Musical Theory

With [8] definitions for musical best practice may be constructed for use in the Critic Algorithm.

Synthesizer & Making WAVs

By using the C++ format and a Frequency Modulator[1], different notes are synthesized, then added together. Songs are exported as WAVs through the use of [22]'s WAV-creating code.

Genetic Algorithms

The basic methods for Crossover, Mutation, as well as, the format that the GA will use to store songs, were inspired from [12].

References

PEER-REVIEWED ARTICLES

[1] Chowning, John. “The Synthesis of Complex Audio Spectra by Means of Frequency Modulation”. Stanford Artificial Intelligence Laboratory. Stanford, California. [Online] Available: https://web.eecs.umich.edu/~fessler/course/100/misc/chowning-73-tso.pdf, Accessed on: Jan. 28, 2016

[16] Hyun, Min-Ho, Na, Jong-Hyun, and Hwang, Sun-Young. “DESIGN OF A PIPELINED MUSIC SYNTHESIZER BASED ON THE WAVETABLE METHOD”. Sogang University. Seoul, Korea. [Online] Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=628683, Accessed on: Feb. 18, 2016

OTHERS

[Fig. 1] Zapata, Jorge. “Spanish robot can name that tune”. CNET. [Online] Available: http://www.cnet.com/news/spanish-robot-can-name-that-tune/, Accessed on: Jan 28, 2016

[2] B. H. Suits. “Physics Of Music - Notes”. Physics Department, Michigan Technological University. [Online] Available: http://www.phy.mtu.edu/~suits/notefreqs.html, Accessed on: Jan. 28, 2016

[3] OpenCog. “OpenCog AtomSpace”. OpenCog Foundation. [Online] Available: https://github.com/opencog/atomspace and http://opencog.org/, Accessed on: Jan. 16, 2016

[4] OpenCog. “OpenCog Ocpkg”. OpenCog Foundation. [Online] Available: https://github.com/opencog/ocpkg and http://opencog.org/, Accessed on: Jan. 16, 2016

[5] OpenCog. “OpenCog OpenCog”. OpenCog Foundation. [Online] Available: https://github.com/opencog/opencog and http://opencog.org/, Accessed on: Jan. 16, 2016

[6] Wolfson Microelectronics Ltd. “Wolfson-WM8731-audio-codec.pdf” [Online] Available: http://www.cs.columbia.edu/~sedwards/classes/2008/4840/Wolfson-WM8731-audio-CODEC.pdf, Accessed on: Jan. 28, 2016

[7] Wikipedia. “Ethernet over Twisted Pair”. [Online] Available: https://en.wikipedia.org/wiki/Ethernet_over_twisted_pair, Accessed on: Jan. 28, 2016

[8] Dineen, Joe, and Mark Bridges. *The Gig Bag Book of Theory & Harmony: A Handy Reference Guide for Both Amateur and Professional Musicians* ... New York: Amsco Publications, 2000. Print.

[9] OpenCog. "OpenCog Cogutils". OpenCog Foundation. [Online] Available: https://github.com/opencog/cogutils and http://opencog.org/, Accessed on: Jan. 16, 2016

[10] Rodriguez, Edwin, Crinklaw, Peter, Jiang, Qiushi."Laser Harp". ECE 492 Capstone Winter 2015. Edmonton, Alberta. [Online] Available: https://www.ualberta.ca/~delliott/local/ece492/projects/2015w/g7_laserharp/, Accessed on: Jan 30, 2016

[11] The Python Standard Library: Python Software Foundation [Online] Available: https://docs.python.org/3.5/library/doctest.html, Accessed on: Jan 30, 2016

[12] M.Marques, V.Oliveira, S.Vieira, AC Rosa. "Musical Composition Using Genetic Evolutionary Algorithms"... 049-001 Lisboa, Portugal [Online] Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=870368&newsearch=true&queryText=Music%20Composition%20Using%20Genetic%20Evolutionary%20Algorithms%20, Accessed on: Jan 28, 2016

[13] The Engineering ToolBox. "Voice Level and Distance". The Engineering ToolBox. [Online] Available: http://www.engineeringtoolbox.com/voice-level-d_938.html, Accessed on: Jan 30, 2016

[14] Altera Ltd. "NIOS II core Datasheet". Altera Ltd [Online] Available: https://translate.google.ca/translate?hl=en&sl=ja&u=https://www.altera.co.jp/ja_JP/pdfs/literature/hb/nios2/n2cpu_nii51017_j.pdf&prev=search, Accessed on: Jan. 30, 2016

[15] City of Edmonton. "BYLAW 14600, COMMUNITY STANDARDS BYLAW ". City of Edmonton. [Online] Available: http://www.edmonton.ca/bylaws_licences/C14600.pdf, Accessed on: Jan 30, 2016

[17] Ganske, Peter. "Solar Thermal Forum". Sol Source Inc. [Online] Available: http://www.solaralberta.ca/sites/default/files/events/documents/Peter.pdf, Accessed on: Feb. 28, 2016

[18] A. van Straeten and J. Jansen. "Pachelbel's Canon in D". [Online] Available: http://sheetmusic4saxophones.com/wp-content/uploads/2012/06/Pachelbel-Canon-in-D-SATB-sample-page.jpg, Accessed on Mar. 21, 2016

[19] K. Kondo. “Overworld Theme from Super Mario Bros.”. [Online] Available: http://www.gamemusicthemes.com/sheetmusic/nintendo/supermariobros/overworldtheme/Super_Mario_Bros_-_Overworld_Theme_Sheet_Music_by_BlueSCD_1.png, Accessed on Mar. 31, 2016

[20] J. Williams. “Star Wars (Main Theme)”. Warner-Tamerlane Publishing Corp. [Online] Available: https://s-media-cache-ak0.pinimg.com/736x/c7/7b/11/c77b11d096c9a6d2c17a31eb54b74c20.jpg, Accessed on Mar. 31, 2016

[21] Unknown. ”Hot Cross Buns”. [Online] Available: https://s-media-cache-ak0.pinimg.com/736x/c4/d7/c9/c4d7c97afb6c819a01e68e01674b5479.jpg, Accessed on Mar. 31, 2016

[22] gasstationwithoutpumps. “Making WAV files from C programs”. [Online] Available: https://gasstationwithoutpumps.wordpress.com/2011/10/08/making-wav-files-from-c-programs/, Accessed on Mar. 10, 2016

[23] BestBuy. “Logitech Z313 2.1 Channel Computer Speaker System”. [Online] Available: http://www.bestbuy.ca/en-CA/product/logitech-z313-2-1-channel-computer-speaker-system/10129934.aspx?path=29374a6150358493b23242dea9d75311en02, Accessed on Apr. 08, 2016

[24] BestBuy. ”Lenovo B40-80 14" Laptop - Black”. [Online] Available: http://www.bestbuy.ca/en-ca/product/lenovo-lenovo-b40-80-14-laptop-black-intel-core-i3-4005u-500gb-hdd-4gb-ram-window-10-home-64-bit-b40-80-80ls001jus/10409397.aspx?path=883f7f2c2d94b37a3ff3b0b1d7a36b1cen02, Accessed on Apr. 08, 2016

Appendix

Quick Start Manual

Setup for the Lab Computers

This line enables Python 3 on lab computers

`scl enable python33 bash`

All At Once

Running on a bash-enabled system

`chmod 0700 run.sh`

`./run.sh [-s|-p] RandomSeed NumIterations`

`./run.sh [-c] [-s] NumIterations`

- `-p | p | print`: Print information into the terminal
- `-s | s | supress`: Prevents User Interface from opening
- `-c | c | continue`: Continues from the last generation; Does not initialize anything

Synthesizer

To synthesize samples from a specific generation use

`./wav.sh [AM|ACCORDIAN|BASSOON|BELL|BRASS|FM|GUITAR|HARP|ORGAN|WOODBLOCK] main_py_samples_generation_###`

- `AM`: Amplitude Modulator
- `FM`: Frequency Modulator
- All instruments are hardcoded in `C2Wav.cpp` and all of them but the Organ were created with the Frequency Modulator
- If run without arguments, instruments are assigned by track ID

Future Work

1. Have a synthesizer in Hardware (DE2)
    - Would allow for faster song synthesis.
2. Replace the GA with AGI.
    - Would allow for smarter song generation.
3. Accept a User created song that the algorithms will fix/improve.
    - Would allow for easier remixing of existing songs, might have economic value.
4. Have Manual Override, where a user will grade songs themselves.
    - Would allow the user to directly influence the production of future songs based on their personal taste in music.
7. Parallelize the Genetic and Critic Algorithms.
    - Would speed up the evolution process.
8. Play audio and allow manual grading over the internet.
    - Could expand the platform and might lead to an online community forming

Source Code

Main Program C++ Code

/

- Makefile (Tested/Works)
- run.sh (Tested/Works)

src/

- Overhead.h (Tested/Works)
- Overhead.cpp (Tested/Works)
- ai_shell.h (Tested/Works)
- ai_shell.cpp (Tested/Works)
- critic_shell.h (Tested/Works)
- critic_shell.cpp (Tested/Works)
- frequencies.h (Tested/Works)
- frequencies.cpp (Tested/Works)
- song_structs.h (Tested/Works)
- python_to_cpp_converter.h (Tested/Works)
- python_to_cpp_converter.cpp (Tested/Works)
- sd.h (Tested/Works)
- sd.cpp (Tested/Works)

Synthesizer Program C++ Code

/

- wav.sh (Tested/Works)

/src/C2Wav/

- Makefile (Tested/Works)
- C2Wav.h (Tested/Works)
- C2Wav.cpp (Tested/Works)
- main.h (Tested/Works)
- main.cpp (Tested/Works)
- make_wav.h (Tested/Works)
- make_wav.cpp (Tested/Works)

Critic Tester C++ Code

src/

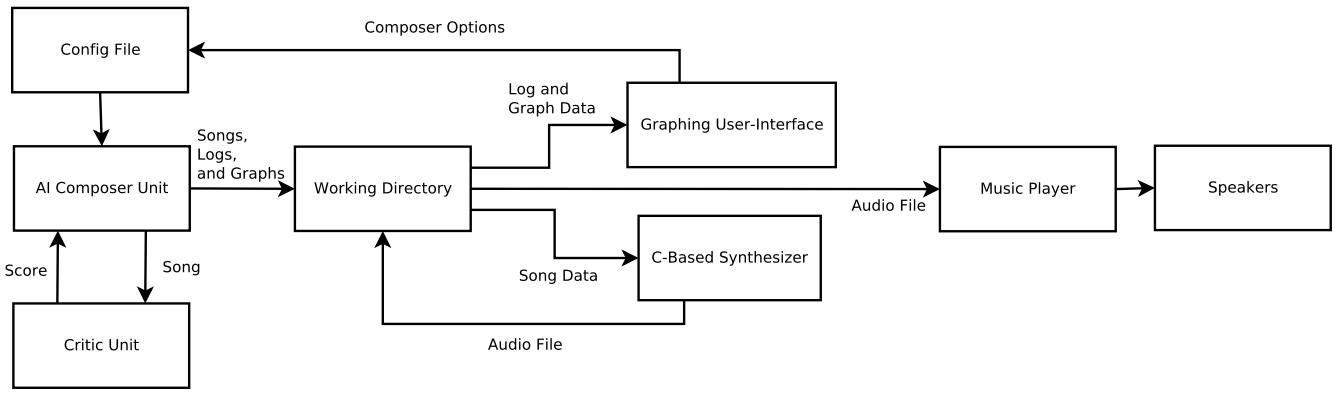
- Makefile (Tested/Works)
- canon_in_d_test.cpp (Tested/Works)

Python 3 Code

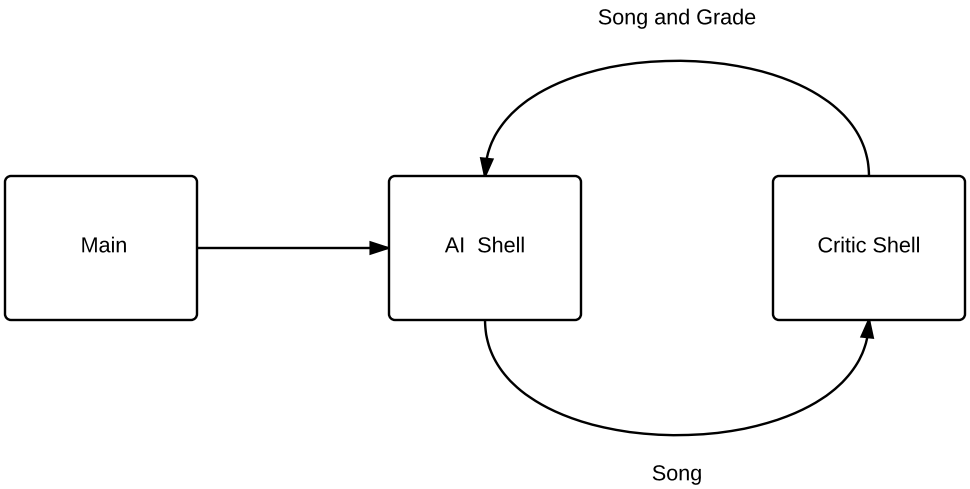
GA_Code/

- NoteGene.py (Tested/Works)
- NoteChromosome.py (Tested/Works)
- GeneticSong.py (Tested/Works)
- GeneticSongRandomizer.py (Tested/Works)
- main.py (Tested/Works)
- BiasedRandomSequence.py (Tested/Works)
- ConfigFile.py (Tested/Works)
- RandomSongGen.py (Tested/Works)
- SongPersistance.py (Tested/Works)

Data Flow Diagram

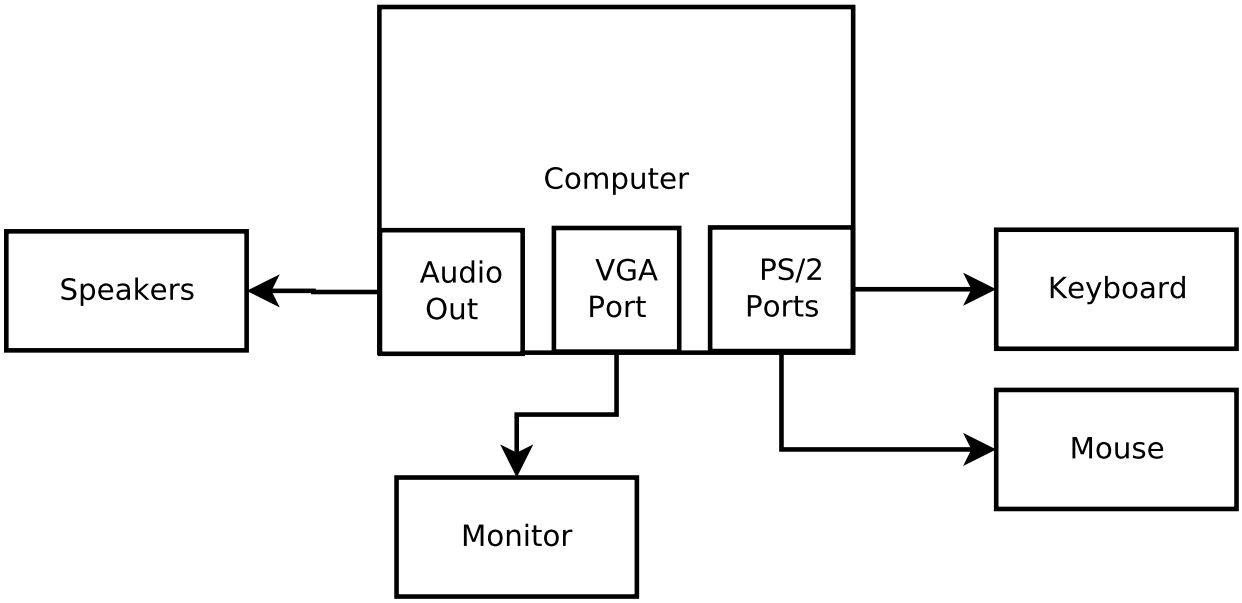


Sequential Process
Interaction Diagram



Hardware Block Diagram

*



*