Evolution of Music

"It was the best of music; it was the worst of music; then it's children surpassed it; and then it died"

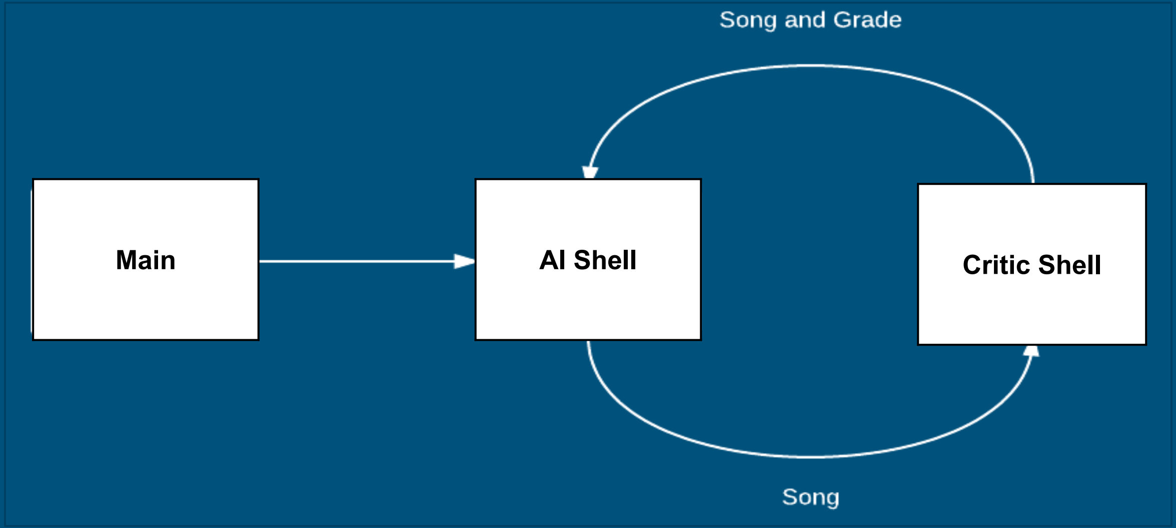
Introduction

- Stephen Andersen	Overhead, Supervisor Algorithm (SA)
- Aaron Schuman	Artificial General Intelligence (AGI)
- Lee Ingram	Genetic Algorithm (GA)
- Jonathan Peard	VHDL, Music Theory (SA Rules)

Motives

- AI interests us
- Music Theory (We are [mostly] Musicians)
- Ability to generate unique music automatically

Sequential Process Interaction Diagram

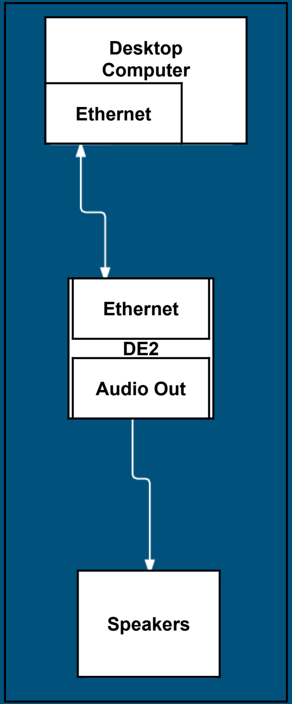


Main: accepts random seed; contains running loop; interacts with DE2 via ethernet,
AI Shell: creates songs based on input and prior songs,
Critic Shell: grades songs

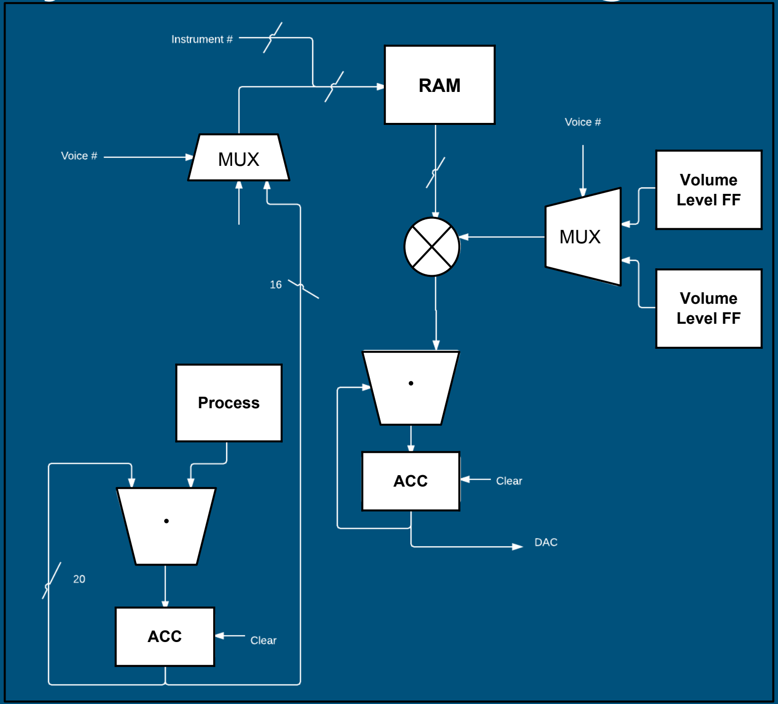
Design

- Song Structures
  - Song, Tracks, Notes
- Genetic Algorithm (GA)
  - Data representation
  - Basics of crossover and mutation
  - Selection and reproduction
- Artificial General Intelligence (AGI)
  - Data format
  - Pattern matching
- Supervisor Algorithm (SA)
  - Requirements
  - Limitations
- Synthesizer
  - Inputs/Outputs
  - Frequency stepper

Hardware Diagram



Synthesizer Diagram



DE2 LUX-Based Audio Synthesizer

- Based on the synthesizer from the Laser Harp project
  - Adding a multiplexer and many frequency counters
  - Therefore adding support for an infinite number of instruments
- Inputs to the synthesizer are an instrument number and a note number
  - The instrument number will select which audio wave (in RAM) will be stepped through
  - The note number selects which frequency counter will be used to step through the audio wave in RAM
- An accumulator will allow multiple notes to be played at once

Data Format C++ (Overhead)

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

{Song ID, Tempo, Array of Tracks}

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

{Instrument Number, Array of Notes, Volume}

Note: Each Note can be viewed as a structure.

{Tone, Pause Time, Hold Time}

Data Format (Python GA)

- Each Note (or NoteGene) has the following format
  - [<left pause time>, <left hold time>, <tone>, <right hold time>, <right pause time>]
- Multiple NoteGenes are appended together to form a track or NoteChomosome
- Similar to the NoteGenes, NoteChromosomes are combined to form songs

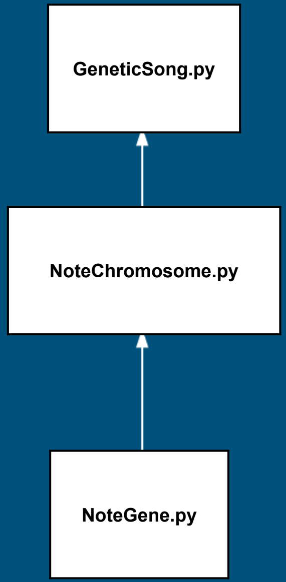
Basics of Crossover and Mutation (GA)

- Mutation is accomplished by adding a list of five integers to a gene
- Crossover is accomplished in two ways:
  - Swapping NoteGene components, such as hold time, pause time, tone…, between chromosomes
  - Swapping left hold and pause times, as well as the tone (More Frequent)

The Basics of Song Reproduction (GA)

- Each Song has a score supplied to it by the Supervisor Algorithm
- This score is used to determine the Inter-song crossover and mutation probabilities
- The songs with the highest score are the most likely to be selected for reproduction
- Each reproducing song creates a copy of itself, which is then modified using the three operations discussed on the previous slide
- Finally the new songs are graded and the cycle repeats

Dependency Hierarchy for Genetic Algorithm



Genetic Algorithm (Python 3.5)

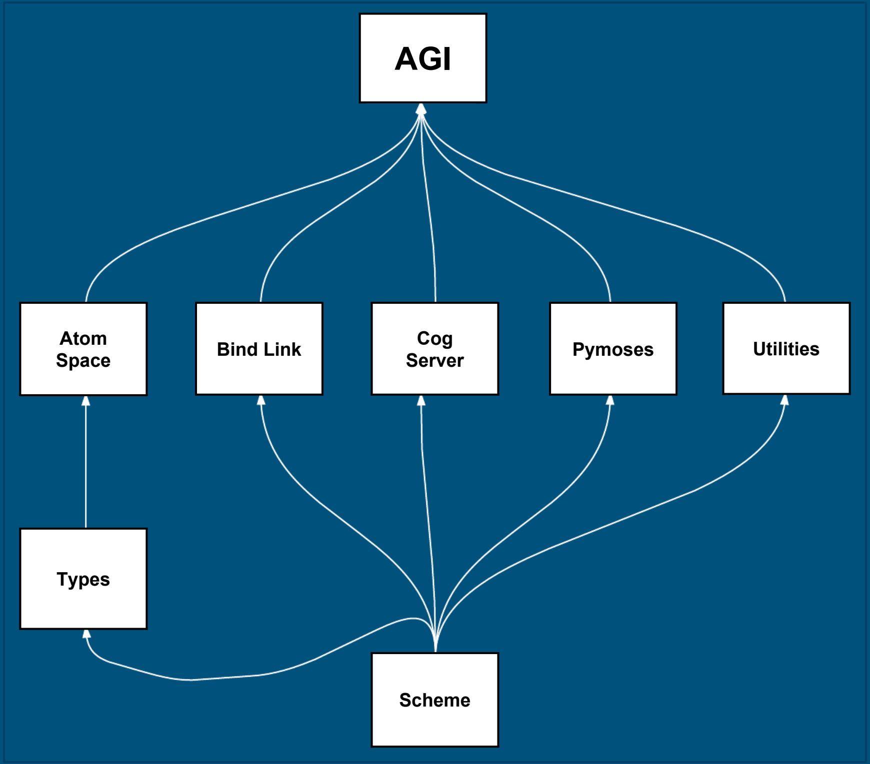
```
def random_gene(max_hold_time=10, max_pause_time=10):
    right_pause_time = randrange(0, max_pause_time//2)
    right_hold_time = randrange(0, max_hold_time//2)
    tone = randrange(0, 96)
    left_hold_time = randrange(0, max_hold_time//2)
    left_pause_time = randrange(0, max_pause_time//2)

    return NoteGene(right_pause_time, right_hold_time, tone,
                left_hold_time, left_pause_time)
```

```
def random_chromosome(length, max_hold_time=10,
max_pause_time=10):
    rv = NoteChromosome()
    for i in range(length):
        rv.append(random_gene(max_hold_time, max_pause_time))
    return rv

def random_song(song_len, max_chromo_len=10, max_hold=10,
max_pause=10):
    rv = GeneticSong()
    for i in range(song_len):
        chromo_len = randrange(1, max_chromo_len)
        rv.append(random_chromosome(chromo_len, max_hold,
max_pause))
    return rv
```

AGI Dependency Hierarchy



Data Format(AtomSpace AGI)

- AtomSpace is an API for storing and querying hypergraphs
  - A hypergraph is a generalization of a graph in which any edge can connect to any number of vertices
- Each vertex has been designed to meet a certain set of properties:
  - Uniqueness of vertices
  - Indexes to provide fast access to vertices
  - Persistence by allowing the contents of AtomSpace to be saved-to/restored-from
  - Distributed computing by sharing vertices on a common backend database
  - Pattern Search for all subgraphs of a particular shape
  - Change notifications that cause a signal to be sent whenever a vertex is added or removed to allow actions to be triggered as contents change

Data Format continued...(AtomSpace AGI)

The hypergraph itself also has to meet certain design requirements:

- Being capable of holding billions vertices and edges that would scale to petabytes worth of memory
- Queries are to be performed as fast as possible
- Be thread safe
- Interactions between hypergraphs with other network-remote atomspaces must be conducted in a quick, coherent manner
- Values associated with each vertex or edge must be accessed in the shortest amount of time possible.

Pattern Matching Process(AGI)

Pattern Matching is the process where using a song's score the program calculates the score associated with a pair of notes.

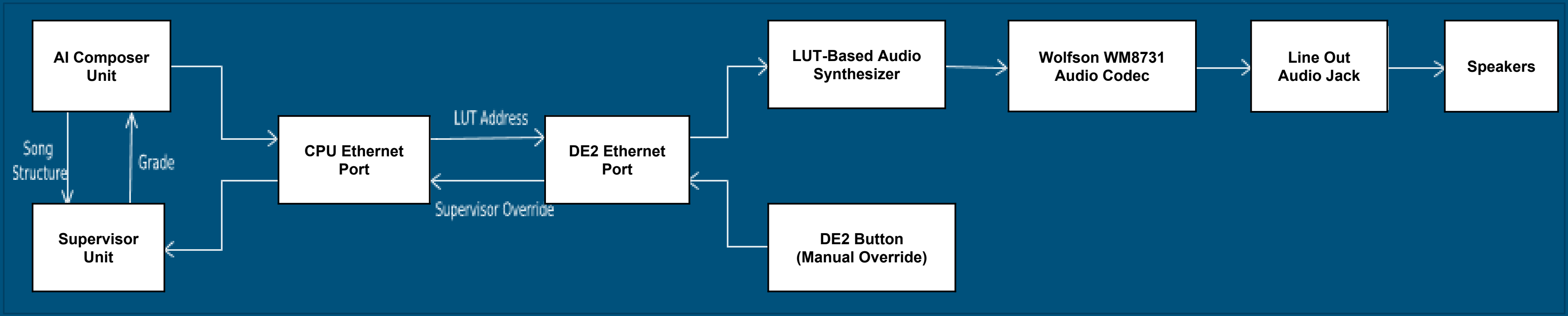
Pattern matching process:

1. Generate 2 random songs of equal length
2. Break the 2 songs into 4 sub-songs, 2 of which will be singular tracks and the other 2 will be made from the rest of the tracks
3. Combine the 4 sub-songs into 2 song of every single combination between them and submit them
4. Break one of the tracks into halves and generate 3 subtracks of equal size
5. Repeat step 3 with sub-tracks instead of sub-songs and tracks until all subtracks are known.
6. Repeat this process for the other sub-song, the other song, and for new songs

Pattern Matching Algorithm(AGI)

Utilizing the memoryless properties of the critic, some algebra, and the restrictions on the first and last note of each sub-track being the same as the others, then the values for a smaller portion of a musical track can be acquired. By repeating this process, the value associated with every note pair can be discovered.

Data Flow Diagram



Testing

- IO testing
  - Desktop to DE2
  - DE2 to Audio Out
  - DE2 buttons to Desktop
- Synthesizer/Hardware testing
  - Static waveform tests
  - Waveform switching
- Composer testing
  - Basic Operations: complete
- Supervisor testing
  - Consistency testing
  - Comparison to human evaluation

Composer Testing

- Testing will be primarily accomplished using docstrings, which will test all relevant edge cases (ie, inappropriate inputs, crossover between chromosomes or songs of differing length, ... etc.)
- The testing of the genetic algorithm's effectiveness will be observation based, using the average score of all of the songs in the population

Optional Features

- AGI (the implementation may not work as intended)
- Note Volume
- Musical Styles
- Both AGI and the GA running on different computers competitively (score based)
- User Criticism UI (Manual Override)

Questions Anyone?