Musician's Toolbox

Christina Lipson '14
Advisor: Professor John Ridgway
Department of Computer Science, Trinity College, Hartford, CT 06106

1. Introduction

As a longtime musician, I have accumulated a list of tools that I believe would help me in my musical endeavors. The web application I have developed provides a space with a cohesive set of tools for users to train themselves in various musical skills, create their own music, analyze music, and reference musical information. These four components (Train, Create, Analyze, Reference) act as categories and each contain different musical functions. The application was built using Ruby on Rails and other assisting web development languages (HTML, CSS, JavaScript). I utilized open source code for the sound and visual components of my application in order to move quickly and effectively through all of the material I wanted to cover. Using a real-time audio JavaScript library and a music notation rendering API as my building blocks, I created algorithms for advanced chord generation, interval generation/classification, live audio analysis, and optimal audio-visual connection. The resulting application is a space with musical functions that incorporate the visual and auditory components of music.

2. Outline

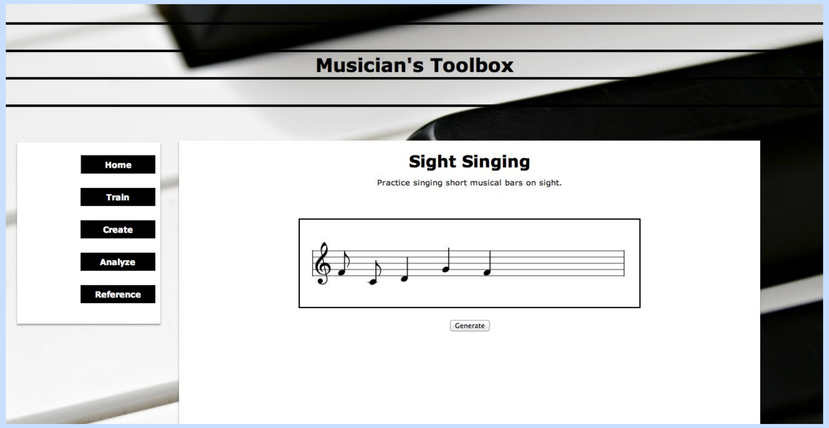
- ♪ Train
  - ♪ Interval Training *(test user on musical intervals)*
  - ♪ Sight Singing *(user sings generated melodies)*
  - ♪ Pitch Training *(test user on pitch)*
- ♪ Create
  - ♪ Note Playground *(pitch grid that user can create music with)*
- ♪ Analyze
  - ♪ Voice to Score *(turn live audio input into a musical score)*
- ♪ Reference
  - ♪ Chord Charts *(displays/plays various chord types)*
  - ♪ Musical Dictionary *(terminology for the user)*

3. Objectives

- ♪ Build framework of application and design layout
- ♪ Real-time audio in browser
- ♪ Visual notation method
- ♪ "Train" exercises system
- ♪ Chord chart generation system
- ♪ Storage of musical terms
- ♪ Note Playground: musical grid
- ♪ Voice to Score: filtering, organizing, and processing live audio

4. Audio/Visual Representation

- ♪ **Audiolet.js**
  - ♪ JavaScript library for real-time audio written by Joe Turner
  - ♪ Graph-based routing and pattern-based scheduling
- ♪ **VexFlow.js**
  - ♪ JavaScript library for music notation rendering written by Mohit Muthanna Cheppudira
  - ♪ Supports HTML5 Canvas (used to draw graphics on the fly on a web browser via scripting) and Scalable Vector Graphics (XML-based vector image format)



Web Interface: menu on the left (with drop-down of options), current activity in the middle

5. Intervals

An interval is the distance between two notes measured by semitones, or half steps. Intervals in a given octave (the span between two notes of the same name) can range from a P1, or unison (0 semitones) to a P8, or octave (12 semitones). Each note has a distinct frequency, and intervals can be measured by the ratio of two frequencies.

First Note	Second Note	Interval	Numeric Distance
C	C	P1	0
C	Db	m2	1
C	D	M2	2
C	Eb	m3	3
C	E	M3	4
C	F	P4	5
C	F# or Gb	A4 or D5	6
C	G	P5	7
C	Ab	m6	8
C	A	M6	9
C	Bb	m7	10
C	B	M7	11
C	C	P8	12

Intervals by semitone (left), Interval ratios (down)

Interval	Ratio to Fundamental Equal Temperament
Unison	1.0000
Minor Second	1.05946
Major Second	1.12246
Minor Third	1.18921
Major Third	1.25992
Fourth	1.33483
Diminished Fifth	1.41421
Fifth	1.49831
Minor Sixth	1.58740
Major Sixth	1.68179
Minor Seventh	1.78180
Major Seventh	1.88775
Octave	2.0000

To test the user on an interval, the program selects two random notes, plays them in succession by sending their frequency values to an Audiolet object, determines the interval by the ratio, and stores this interval until the user chooses to reveal it or generates a new interval.

6. Voice to Score

1. Enable use of microphone in browser
2. Audio Context object in JavaScript listens for live audio
3. Classify audio frequencies as pitches based on window of frequencies that surround notes (**Pitchdetect.js**, by Chris Wilson) to store along with frequency (used to determine octave)
4. Filter out superfluous frequencies that are too high or low to be human sound
5. Segment groups of pitches as intentional and constant
6. Send data to Audiolet and VexFlow to produce the audio and notation of the melody

7. Chord Charts Generation

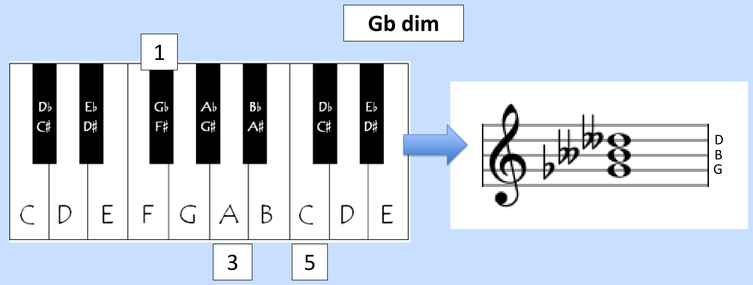
A chord is made up of notes stacked on top of each other, and the intervals between notes determine the variety of the chord. In this application, I am dealing with the triad chord, which is three notes stacked on top of each other. The variety of triads are:

Augmented (Aug): M3 (4 semitones), M3 (4 semitones)
Major (M): M3 (4 semitones), m3 (3 semitones)
Minor (m): m3 (3 semitones), M3 (4 semitones)
Diminished (dim): m3 (3 semitones), m3 (3 semitones)

Each triad must spell the letter names as every other letter from the root (ex: C-E-G). Using accidentals to raise (sharp) or lower (flat) the notes is what maintains the intervals and the letter names. My algorithm organizes the notes into an array of arrays, and each individual array contains the various names that a note could have. The search for the notes moves first by the interval between the notes, then searches for the appropriate letter name of the note within the selected array.

[["B#", "C", "Dbb"] , ["C#", "Db"] , ["C##", "D", "Ebb] , ["D#", "Eb", "Fbb"] , ...]

The example below charts out a Gb dim triad. The letter names must be G-B-D, so although the two m3 intervals land on notes known as A and C for the third and fifth, we must call these Bbb and Dbb.



8. Conclusion

- ♪ Interval, pitch & chord generation complete
- ♪ Sight singing has set interval patterns to randomly pick, but complexity/diversity of these patterns could be expanded up
- ♪ Note Playground (a series of click on/click off values in a grid to generate a melody) could have further use if the creation could be saved and exported
- ♪ Voice to Score is simplified to disregard timing
- ♪ Quality of generated audio could be more pristine
- ♪ Overall, this application is a great basis for me to continue expanding upon

Future Work:

- ♪ Extend Voice to Score to account for time (key signature, note lengths, etc.)
- ♪ Add save capabilities to the application
- ♪ Improve audio quality
- ♪ Continue adding new levels of musical complexity

Acknowledgements: Professor John Ridgway, Trinity College Department of Computer Science