Capstone Documentation

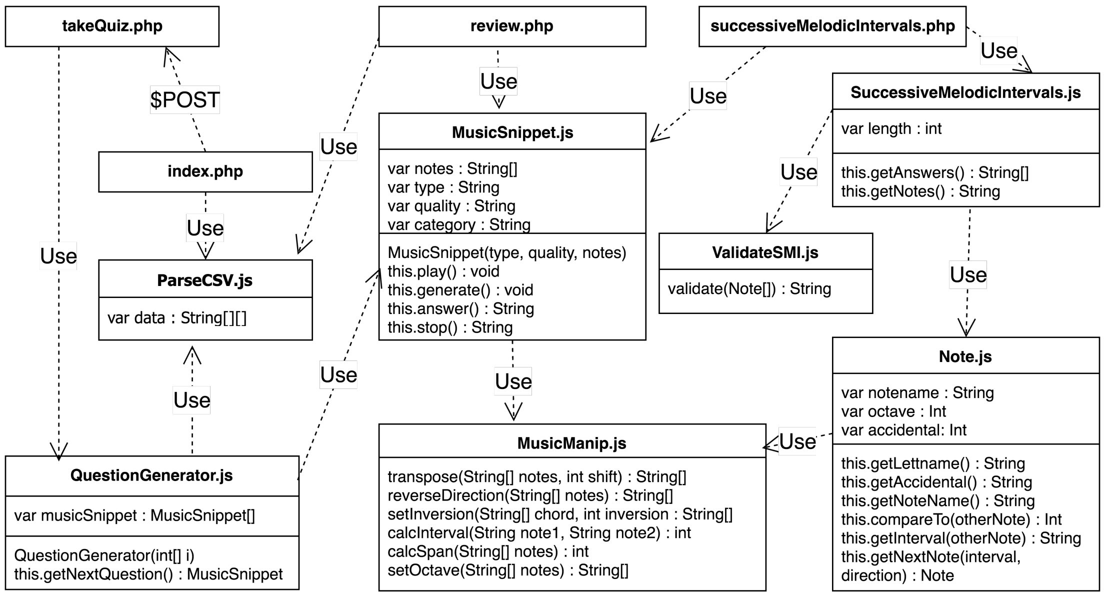
Wesley Stedman, Kaylene Barber, Imanuel Chen

2016

Contents

1 Overview

This project is available on Github at `https://github.com/kbarber-ups/Capstone`.


takeQuiz.php
review.php
successiveMelodicIntervals.php
Use
$POST
Use
Use
Use
SuccessiveMelodicIntervals.js
var length : int
this.getAnswers() : String[]
this.getNotes() : String
MusicSnippet.js
var notes : String[]
var type : String
var quality : String
var category : String
MusicSnippet(type, quality, notes)
this.play() : void
this.generate() : void
this.answer() : String
this.stop() : String
index.php
Use
Use
ParseCSV.js
var data : String[][]
ValidateSMI.js
validate(Note[]) : String
Use
Use
Note.js
var notename : String
var octave : Int
var accidental: Int
this.getLettname() : String
this.getAccidental() : String
this.getNoteName() : String
this.compareTo(otherNote) : Int
this.getInterval(otherNote) : String
this.getNextNote(interval, direction) : Note
Use
Use
Use
MusicManip.js
transpose(String[] notes, int shift) : String[]
reverseDirection(String[] notes) : String[]
setInversion(String[] chord, int inversion : String[]
calcInterval(String note1, String note2) : int
calcSpan(String[] notes) : int
setOctave(String[] notes) : String[]
Use
QuestionGenerator.js
var musicSnippet : MusicSnippet[]
QuestionGenerator(int[] i)
this.getNextQuestion() : MusicSnippet

2 Note Representation

Notes in our program are represented as

[Letter][Sharp/Flat]{0-2}[Octave].

’Letter’ must be capitalized. ’Sharp’ is represented as ’x’ and ’Flat’ is represented as ’b’. ’Octave’ is an integer. As an example, B double flat in the fourth octave is represented as

Bbb4.

Note that in the CSV file that is discussed in the next section, the notes will not contain the octave number since the user should not have to worry about what octave the notes will be played in.

3 Data Storage

3.1 Data Files

Scale and chord data is stored in a CSV file. Each line of the file must have a type (scale or chord), a quality (major, minor, etc.), and note names. Note names must be separated by spaces, and the number of sharps and flats should be minimized. A line may additionally have a category or test. Blank lines and lines starting with ”//” are ignored by the parser.

3.2 ParseCSV.js

Downloads the CSV file from the server, then places its contents into a 2D array called ”data”.

3.3 Audio Files

We store an audio file per note (currently using piano). The audio files are located in ”/audio/piano/” and are in .mp3 format (there is another folder called ”/pianowavs/” which has the files in .wav format). Audio files are named ”piano[midi].mp3”, where [midi] is the midi number of the note it stores. Currently, we have the notes 48-83 (which correspond to C3-B5).

4 Music Algorithms

`MusicManip.js` is a script that contains a variety of methods that perform useful manipulations on an array of musical notes.

4.1 Circle of Fifths

To alter a note collection in a consistent and meaningful way, we have established the circle of fifths to be the natural ordering of the note names. This ordering is crucial for maintaining the correct note names. The circle of fifths is declared as follows:

```
const NOTES = ["Fbb", "Cbb", "Gbb", "Dbb", "Abb", "Ebb", "Bbb", "Fb", "Cb", "Gb", "Db",
    "Ab", "Eb", "Bb", "F", "C", "G", "D", "A", "E", "B", "Fx", "Cx", "Gx", "Dx", "Ax", "Ex",
    "Bx", "Fxx", "Cxx", "Gxx", "Dxx", "Axx", "Exx", "Bxx"];
```

4.2 Transpose

Takes an array of notes and shifts them over by the specified amount in the circle of fifths. Take care with providing the amount to shift; you don't want to shift far enough to create an unused spelling. Since the shift argument must be in fifths, it is a range that can be easily determined.

```
var cmaj = ["C", "D", "E", "F", "G", "A", "B", "C"];
var gmaj = transpose(cmaj, 1); // add one sharp
var dmaj = transpose(cmaj, 2); // add two sharps
var fmaj = transpose(cmaj, 1); // add one flat
```

4.3 Reverse Direction

Reverses the direction of a collection of notes. This is meant to be used for scales and intervals, but *not* chords. This must be done *after* the octave is set!

```
var cmaj = ["C", "D", "E", "F", "G", "A", "B", "C"];
var dmaj = transpose(cmaj, 2); // shift two sharps
```

4.4 Set Inversion

Reorders the notes so that they are in the specified inversion. To be used for major triads, minor triads, and 7th chords only! NEVER to be used for scales, intervals, jazz chords, and 20th century chords!!!

4.5 Calc Interval

Calculate the interval between two notes. Returns the interval as a number of half steps.

4.6 Calc Span

Calculates the span of a collection of notes. Returns an integer representing the number of half steps between the lowest note and highest note in the given note collection.

4.7 Set Octave

Sets an octave appropriate for the given span of notes. Does so by appending a number to the end of each String. DO NOT USE ON A SCALE OR INTERVAL THAT HAS BEEN REVERSED. PLEASE REVERSE AFTER SETTING THE OCTAVE!!! Inverted chords are O.K.

4.7.1 Set Octave Randomly

Given an array of notes, this function will append octave numbers starting with a random octave from 3 to 5. Then, it will return the notes with the octave numbers appended.

4.7.2 Finding Potential Octave Numbers

This function takes an array of notes and finds all potential starting octave numbers for the sonority.

4.7.3 Set Octave Numbers

Given a starting number, appends octave numbers to the provided array of notes.

4.8 Increment

Go up one note name.

4.9 Decrement

Go down one note name.

5 Music Snippet Class

Class that takes in an array of notes and processes them to be played. Located in `scripts/MusicSnippet.js`. Uses the Howler Javascript library (`https://github.com/goldfire/howler.js/`) to load, play and manipulate the audio files.

5.1 Parameters

notes {String[]} A string array of the notes to be played where the format of each note follows our notation format given in section 2. If just the Notes parameter is given, then the MusicSnippet class will just convert the notes to Howl objects and be ready to play them. However, if the rest of the parameters are given, then the Notes MUST NOT have the octave number appended to them. The MusicSnippet class will assign an octave number to them using `setOctave()` in `scripts/MusicManip.js`.

type {String} (Required if notes does not have octave numbers) The type (chord or scale) of the snippet.

quality {String} (Required if *type* is given) The quality of the snippet (i.e. Major/Minor).

category {String} (Optional) The category of the snippet (i.e. 20th century). If not given, then quiz style playing won't work properly.

5.2 Important Class Variables

```
var baseNotes = notes;  //Notes given as parameter
var tempNotes = [];     //Notes after manipulation
var tempSounds = [];    //Howl objects of notes after manipulation
var bpm = 80;           //Default beats per minute
```

5.3 `noteToFileNum.js`

This file contains an associative array that maps all the notes (up to double sharps and flats) to their corresponding midi number. It is located in `scripts/noteToFileNum.js`.

5.4 `this.generate(`*`user_function`*`, `*`key`*`, `*`inversion`*`, `*`octave`*`)`

Calling `this.generate()` function on a MusicSnippet object will transpose and invert the given notes as necessary and gives them a random octave as necessary as well. It then converts the resulting notes into their corresponding midi numbers using `noteToFileNum.js` and then uses those midi numbers to find the correct audio file per note. Finally, it loads the audio files into Howl objects and saves an array of Howl objects to be played.

5.4.1 Parameters

user_function {function} (Optional) Pointer to a function that will run once the audio files are loaded. This function will be saved as a class variable so if the next time this.generate() is called and *user_function* is not given, it will use the previous function given. If this parameter is never used, then an empty function is used.

key {String} (Optional) The letter name of the key to transpose the MusicSnippet's notes to. If *key* is not given, a random key is selected and transposed to.

inversion {Integer} (Optional) The inversion to set the MusicSnippet's notes to. If not given, a random inversion will only be given to 7th chords that are not diminished 7ths. The *quality* will also be updated to match the given inversion (i.e. applying 2nd inversion to an 'mm7' chord will change the *quality* to 'mm4|3'.) If the given category is not "7th," no inversion is given.

octave {Integer} (Optional) The octave to set the MusicSnippet's notes to. Assumes that the user checked which octaves are valid using `getOctaveLocations()` in `scripts/MusicManip.js`. If not given, a random octave is assigned to the notes.

5.4.2 Examples

Alerting user when all audio files have been loaded:

```
function doneLoading() {
    alert("Finished loading all audio files!");
}

var snippet = new MusicSnippet(["C4", "C5"]);
snippet.generate(doneLoading);
```

Generating given notes with given key but random inversion and octave:

```
var snippet = new MusicSnippet(["D", "Fx", "A"], "chord", "major",);
//Quiz styling won't work properly because category not given
snippet.generate(undefined, "A");   //Transpose to the key of A
```

5.4.3 Private Helper Functions

The following helper functions are used to simplify the body of `this.generate`. If `type` was not given, the function assumes that `notes` were given with the octave and that the user wants to simply play the given notes. Therefore, `this.generate` simply calls `loadFiles()`.

`function TOI(`*`key`*`, `*`invert`*`, `*`octave`*`)` stands for 'Transpose, Octavize, Invert' (yes, octavize is a made up word). It simply does all the transformations mentioned in the parameters section.

`function loadFiles(`*`notes`*`)` is a three step process. First, it converts the given notes to their corresponding midi numbers using an associative array defined in `scripts/noteToFileNum.js`. Then, it converts these midi numbers to their corresponding file names. Finally, it creates new Howl objects using the obtained file names and returns an array of these Howl objects.

`function invert7thQuality(`*`quality`*`, `*`inv`*`)` is only for the use of 7th chords. Because 7th chords have different quality names for different inversions, it is necessary to update `quality` to match this. This function does this and represents fractions using a pipe. (i.e. 2nd inversion would be represented as "4|3".)

5.5 `this.play(`*style*`)`

`this.play()` plays the Howl array that was generated by generate(). If `this.play()` is given no arguments, the notes are played 'quiz style': 20th century chords are played blocked, all other chords are played blocked then broken (ascending); all scales are played broken but may be ascending or descending (chosen randomly).

5.5.1 Parameters

style {String} (Optional) How to play the MusicSnippet's notes. Currently supports: `BLOCK` (play the notes blocked), `ASCENDING` (play the notes broken and ascending), and `DESCENDING` (play the notes broken and descending).

5.5.2 Private Helper Functions

The building blocks of `this.play` consist of helper functions that define how to actually play the loaded audio.

`function playNote(i, fade)` simply calls play on the Howl object given in the `i`th index in `tempSounds`. If fade is given, the Howl object will immediately begin to fade out to silence over the course of `fade` beats. When the fade finishes, the Howl object will be reset to max volume and starting position. This function is used in the next two functions.

`function playBlock(fade)` plays all Howl objects in `tempSounds` at the same time. It does so by using the `playNote` function given above. So the `fade` parameter simply gets passed into `playNote`.

`function playBroken(fade, style)` plays the given notes in succession with one beat of space in between (one beat is defined as $(60/bpm) * 1000$ milliseconds). `style` can be either `ASCENDING` or `DESCENDING`. `fade` is applied through calling `playNote` for each note. This function had to be designed recursively because of the use of the `fadeOut`'s optional `callback` parameter. Since `callback` is not called until the fade out finishes, we need to remember which Howl object to apply the `callback` function on; if a loop was used, the index at the time of fade out would be used. Doing it recursively stores the desired index on the stack for us.

5.6 `this.fadeOut()`

The fadeOut() function fades out all sound in `FADE_ALL_LENGTH` milliseconds and stops any further sound from playing. After the fade is finished, all Howl objects will be reset to the starting position and full volume.

5.6.1 Private Helper Functions

Like `playBroken`, this function needs to be recursive because of the use of the `fadeOut` function.

`function fade(note)` is the recursive helper function for `this.fadeOut`. It simply fades out the note in the `note` entry of `tempSounds`, and then immediately calls itself on the next note.

`function clear()` prevents any queued up sounds from playing. Sounds are "queued" for playing using Javascript's `setTimeout()` [1] function. `clear()` simply prevents any delayed sound caused by `setTimeout()` to not play.

5.7 `this.answer()`

The answer() function simply returns a String containing the quality followed by the type with a space in between. In our implementation, we used buttons that change once the files are done loading. Clicking the button would call `snippet.play()`.

[1] See http://www.w3schools.com/jsref/met_win_settimeout.asp for more info on `setTimeout()`

5.8 `this.setBPM(newbpm)`

The default beats per minute for a MusicSnippet is 80. By calling `this.setBPM`, you simply change the beats per minute of the given MusicSnippet to `newbpm`.

5.9 Examples of Usage

The following examples assumes that there is a function called `load_func()` that indicates when the audio files are done loading.

Playing the exact notes given:

```
var snippet = new MusicSnippet(["A3", "Bb4", "Cx5"]);
snippet.generate(load_func);
//Wait for files to finish loading
snippet.play();
```

Playing notes with random transpositions, inversions, octaves:

```
var snippet = new MusicSnippet(["C", "E", "G", "B"], "chord", "MM7", "7th");
snippet.generate(load_func);
//Wait for files to finish loading
snippet.play();
```

Playing notes, in a specific key and octave, broken and ascending:

```
var snippet = new MusicSnippet(["C", "E", "G", "B"], "chord", "MM7", "7th");
snippet.generate(load_func, "F", undefined, 3);
//Wait for files to finish loading
snippet.play(ASCENDING);
```

Using it in conjunction withs QuestionGenerator:

```
var dataArr = [0,5,13,27];      //Indexes in the data array generated from the csv file
var qg = new QuestionGenerator(dataArr);
var snippet = qg.getNextQuestion();
snippet.generate(load_func);
//Wait for files to finish loading
snippet.play();
```

6 Quizzing

6.1 index.php

This is the page where the user can design his/her own quiz (currently the home page). It contains a checkbox for every musical element that can be tested on. If something is checked and ”Start Training” is selected, the user is taken to takeQuiz.php.

6.1.1 Layout

Instructions
Checkboxes for tests
Deselect all button
Column of space | Chords column | Scales column | Column of space
Start Training button

Figure 1: Layout of index.php

6.1.2 Main Functionality

Select/Deselect Buttons Near the top of the page there is a main ‘Deselect All’ button that deselects all checkboxes when clicked. Within each type category (Chords and Scales), there are buttons that toggle

Test 1	**Test 2**	**Final**
Major Scale	Octatonic Scale	Hexatonic Scale
Natural Minor Scale	Diminished-Whole Tone Scale	Lydian-Mixolydian Scale
Harmonic Minor Scale	Fifth Mode Melodic Minor Scale	Phrygian-Dorian Scale
Melodic Minor Scale	Major 6/9 Chord	Rite of Spring Scale
Locrian Scale	Minor 6/9 Chord	Petrushka Chord
Dorian Scale	Major 9(♯5) Chord	Mystic Chord
Phrygian Scale	Minor-Major 9 Chord	Farben Chord
Lydian Scale	7(♯9) Chord	Split-Third chord
Mixolydian Scale	7(♭9) Chord	All elements in Test 1
Anhemitonic Pentatonic Scale	7(♯9/♯5) Chord	All elements in Test 2
Hirajoshi Pentatonic Scale	7(♭9/♯5) Chord	
Whole Tone Scale	All elements in Test 1	
MM7 Chord		
Mm7 Chord		
mm7 Chord		
dm7 Chord		
dd7 Chord		

Figure 2: List of all musical elements within their respective Test category

between Select All and Deselect All buttons. The Select All button shows when no checkboxes are checked within the Chords/Scales sections. When at least one checkbox is checked in a section, that corresponding button changes to Deselect All. Likewise, there are Select/Deselect buttons for each category within a type.

Relevant functions:

- `function checkBoxes(`*`checkbox_class, mainclass`*`); //Called when any Select All button is clicked`
- `function uncheckBoxes(`*`checkbox_class`*`); //Called when any Deselect All button is clicked`
- `function updateButtons(); //Called whenever something is checked`

Checkboxes for Tests There are three of these checkboxes: ‘Test 1’, ‘Test 2’, and ‘Final’. When one is checked, all musical elements that will be contained in that test is checked. Tests are cummulative so higher tier tests contain lower tier tests (‘Test 2’ contains ‘Test 1’, and ‘Final’ contains ‘Test 2’ and ‘Test 1). The list of what musical elements are contained in which tests are listed in figure 2.

Relevant functions:

- `function checkTest1(); //Called when Test 1 is checked`
- `function checkTest2(); //Called when Test 2 is checked`
- `function checkFinal(); //Called when Final is checked`

Start Training Clicking the Start Training button first checks to see if anything is checked. If nothing is checked, the page will display an alert stating to check something and not submit the form. If something is checked, the page submits a form and saves all the selected musical elements in PHP’s superglobal $_POST variable. The page then redirects to the Take Quiz page.

Relevant functions:

- `function validateForm(); //Called when Start Training is clicked`

6.1.3 Background Functionality

Grabbing Data When the page first begins loading, the page will first parse the CSV file using `scripts/ParseCSV.js` and store all its data in the `data` variable. The page will then go through the `data` variable and populate the webpage with the categories and their elements.

Displaying Categories Because of the two column approach in displaying categories within each type, it would look odd if the second column was longer than the first. In order to prevent this, the page sorts the categories by the number of occurrences in the csv file. Categories are then displayed from smallest to largest.

Relevant functions:

- `function bubbleSort(a); //Called immediately after the categories and their occurences are obtained`

Cookies When Start Training is selected, the page will also save a cookie (using Javascript) containing information on what was selected. The next time the page is accessed, the cookie will be read and the corresponding checkboxes will be checked.

Relevant functions:

- `function makeCookie(); //Called in validateForm()`

Displaying Sharps and Flats When the page pulls the data from the 'data' variable, it parses the elements' quality for flats and sharps, which are represented by b's and x's, and converts them to their corresponding HTML codes (&#9837; for flats and &#9839; for sharps).

Relevant functions:

- `function displayQuality(quality); //Called on each quality grabbed from the data array`

6.2 takeQuiz.php

The Take Quiz page takes the information submitted from the Index page and quizzes the user on the selections by hiding the answer and playing the corresponding audio for each musical element.

6.2.1 Layout

Play/Stop/Loading Button
Reveal Answer Button/Answer
Next Question Button
Show List Button
List of User Chosen Musical Elements

Figure 3: Layout of takeQuiz.php

6.2.2 Main Functionality

Play/Stop/Loading Buttons The Loading button displays when the audio is loading and changes to the Play button when the audio finishes loading. The button also is disabled. By default, it is the first button to display. The Play button plays the audio in quiz style and changes to the Stop button once clicked. The Stop button fades out all currently playing audio and disables itself during the fade. Once the fade finishes, the Stop button changes back to the play Button.

Relevant functions:

- `function loadFunc(); //Called whenever snippet.generate() is called`
- `function play(); //Called when the Play button is clicked`
- `function stop(); //Called when the Stop button is clicked`

Reveal Answer Button Clicking the Reveal Answer Button reveals what the currently loaded audio is. In other words, it displays the type and quality of the musical element that would be played if the Play button is clicked. It also reveals the Next Question Button.

Relevant functions:

- `function reveal(); //Called when the Reveal Answer button is clicked`

Next Question Button Clicking the Next Question Button hides the answer and itself, and displays the Reveal Answer Button. The button also fades out all current sound playing and loads the next audio to be played by randomly selecting from the list of user chosen musical elements.

Relevant functions:

- `function nextQuestion(); //Called when the Next Question button is clicked`
- `function hide(); //Called inside nextQuestion()`

Show List Button The show list button displays/hides the list of the musical elements that the user chose in the Index page.

Relevant functions:

- `function showList(); //Called when the Show List button is clicked`

6.2.3 Background Functionality

Processing POST Data The page's first task is to process all data in PHP's superglobal $_POST variable which will contain the information pertaining to what the user selected in the Index page. Using Question-Generator (`scripts/QuestionGenerator.js`), the page generates a list of MusicSnippets (`scripts/MusicSnippet.js`) which can then be used to generate and play audio.

The page will also process the data in order to generate the list of user chosen elements.

Displaying Inversions 7th chords are given a random inversion. MusicSnippet represents inversion notation with a pipe (i.e. dd4|2). Take Quiz will parse the answer given by MusicSnippet for a pipe and display the proper fraction representation of inversions.

Relevant functions:

- `applyInversion(`*`answerElement`*`); //Called in loadFunc()`

6.3 review.php

6.3.1 Selection Options

The user is presented with five dropdowns and one slider/input. On load, only the first dropdown, "Type", has selectable items. When the user has made a non-null selection, the "Quality" dropdown is filled with data from the csv file, and the "Direction/Inversion" dropdown is filled with the appropriate options. When the user selects a quality, the "Key" dropdown is filled with valid keys. When the user selects a key or direction/inversion option, the "Octave" dropdown is filled with valid octave numbers. The "Tempo" slider and text input are independent of the dropdowns.

FUNCTION updateQuality()
Called by the ”type” dropdown on event ”onchange”. Clears ”quality”, ”key”, and ”opt” dropdowns, then fills ”quality” and ”opt”.

FUNCTION updateKey()
Called by the ”quality” dropdown on event ”onchange”. If the quality was deselected, it clears the ”key” dropdown. Otherwise, it fills ”key” with note names, selecting the correct enharmonics for scales.

FUNCTION updateOctave()
Called by the ”key” and ”opt” dropdowns on event ”onchange”. If the key has not been selected, it clears the ”octave” dropdown. Otherwise, it fills ”octave” with valid octave numbers if they exist, and alerts if there are not.

FUNCTION updateTempoSlider() and updateTempoDisplay()
Called by the ”tempo” slider on event ”oninput” and ”tempoDisplay” text input on event ”onchange”, respectively. Each function simply updates the value of the input which didn’t call it with the value of the input that did.

6.3.2 Playing the Selected Item

Pressing the play button triggers the function playSelected(). First, the function ensures that all arguments have been defined by checking if a key has been chosen, as that will always be the last dropdown filled. Then, using the values of the dropdowns, a MusicSnippet object is created and placed in a global variable, so that we can stop it later if necessary. The onload function passed to the MusicSnippet through generate() contains the call to play().

7 Note Object

All of the functionality discussed in previous sections stores notes in String format. **Note.js** is a way of representing notes that will be extremely important as new functionality is added to the website. Currently, only successive melodic intervals uses it. This object is useful because Notes can keep track of their letter-name, accidental, and octave number, and it contain functions can be performed on Notes. A few notable functions are:

- **this.compareTo(otherNotes)** Compares two notes and returns an integer representing the distance in half steps.
- **this.getInterval(otherNote)** Calulates the distance from this Note to the other Note and returns the interval in String format. (e.g. m2, M2, P5, etc.) This does not provide any information about which Note is higher than the other, and will not yet work for compound intervals.
- **this.getNextNote(interval, direction)** Calculates the Note a particular distance away from this Note and returns a new Note object. The number for interval must be provided in fifths.

8 Successive Melodic Intervals

Successive Melodic Intervals (SMIs) are a different style of quizzing students. 4 random sounding notes are played in succession, and the user must identify the intervals between each pair of consecutive notes (3-part answer). SMIs can be created by making a new SuccessiveMelodicIntervals object. It will automatically generate a sequence of 4 notes, which can be obtained using the getNotes() function.

```
var smi = new SuccessiveMelodicIntervals();
var notes = smi.getNotes();
var answers = smi.getAnswers();
```

Two scripts make SMIs possible. They must be included in your web page for them to work.

- SuccessiveMelodicsIntervals.js
- ValidateSMI.js
- Note.js

8.1 Backtracking Algorithm

Backtracking is contained within SuccessiveMelodicIntervals.js. It is an algorithm for solving constraint problems. A set of validation rules must be established to make sure the generated solution so far meets the criteria.

The backtracking algorithm works by creating an array of notes to try, and randomly picking one from that list. Index *i* increments upon successful placement of a Note object in the pattern. Success is determined by checking the note pattern so far with the validate() function in ValidateSMI.js. If the most recent Note placed is unsuccessful, it removes it from the note pattern and tries a different Note from the list. If the algorithm iterates through the entire list of notes to try and none work, then index is decremented and the next most current Note is also removed from the note pattern list. More Notes from the list of Notes to try and placed and tested until it finds something valid.

Randomness is needed to prevent the algorithm from generating the same Note pattern every time. A seeded shuffler is used to shuffle Notes within the list of Notes to try. The seed is necessary so that when backtracking happens, the list of Notes will be created in the same order, and continuing iterating from where it left off. A random number is created every time a new SuccessiveMelodicIntervals is created, so the seed will be different every time. The seed is (g * o) where g is the random number and o is the ordinal of the previous Note

8.2 Validation Rules

Validation rules may be found within ValidateSMI.js. Validation is a series of boolean-returning functions. The rules that have been implemented are:

- Do not repeat notes.
- Do not repeat intervals in the same direction.
- Do not allow all three intervals to occur in the same direction.
- Do not exceed the span of the octave.
- Do not outline a triad or seventh chord.