Voicings Generator

ACTAM - CMRM

Francesco Piferi, Riccardo Rossi, Ferdinando Terminiello

Professors: **Francesco Bruschi, Vincenzo Rana**
Assistant: **Mattia Bianchi**

A.Y. 2021/2022

Contents

1 Introduction **3**
1.1 Features . . . 3
1.2 Tools . . . 3

2 An Introduction to Voicings **3**
2.1 Rootless . . . 3
2.2 Monk . . . 4
2.3 Powell . . . 5
2.4 Three Notes . . . 6
2.5 Four Notes . . . 7
2.6 Open Chord . . . 7

3 Coding Music **8**
3.1 Dynamic Scales . . . 8
3.2 Voicings Algorithm - Factory Pattern . . . 9
3.3 Dynamic Shifting . . . 10
3.4 The Instrument . . . 10

4 The Main Loop **12**
4.1 `setTimeout()` . . . 12
4.2 Loop Mode . . . 12
4.3 Legato Mode . . . 13

5 GUI **13**
5.1 A React-ive environment . . . 13
5.1.1 ChordsTable.js . . . 13
5.1.2 ChordsVisualizer.js . . . 14
5.1.3 VoicingsSelector.js . . . 14
5.2 Knobs . . . 14

6 Conclusions **15**
6.1 Results . . . 15
6.2 Possible future addictions . . . 15

1 Introduction

Voicings Generator is a web application tool which has the ability to create and play numerous different Voicings for a sequence of chords, given a **Voicings Type, Root Key and Modal Scale**.

1.1 Features

While the main purpose of the application is the creation of voicings for a sequence of chords to be played by a sampler, *Voicings Generator* also has other notable features:

- It can play sounds from an external MIDI controller so that the user is able to play a melody over the computed chords
- It can print the chords voicings so that they can be saved and played by the user whenever he pleases
- It can be used as a basic interface to explore the sound of different voicings
- It can be used as a learning tool to explore basic concepts of music theory: Voicings, Modal Scales, Grades...

1.2 Tools

The app was developed with **nodeJS** and the following modules:

- React
- Tone.js
- VexFlow
- react-piano

2 An Introduction to Voicings

In music, voicings are defined as **the vertical displacement of the notes in relation to one another**. There are many different types of voicings, each one with a different composition and tension. Let's explore some of them.

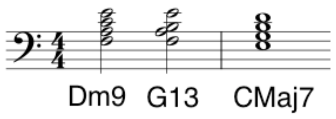
2.1 Rootless

A smooth and popular type of chord voicings are called (Bill Evans style) **Rootless Chord Voicings**. As the name suggests, these chord voicings **exclude the root note**. Instead of the root (and sometimes the 5th), the chord tension is played. This gives a jazzy sound (because of the tensions) while also not being too crowded. In rootless voicings, the top note should be between middle C and the C an octave above middle C on the keyboard, furthermore, the chord is entirely played by the left hand.

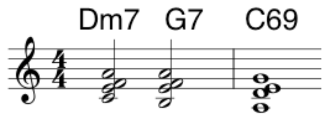
There are also two ‘types’ of rootless chord voicings, in particular **the first is the inversion of the second**. These voicing works really well for II-V-I progressions, because you only need to change a few notes (and by only a small interval) to move from one chord to the next.

Structure

- Type 1:
  - For **Major** and **minor** chords play: **3rd, 5th, 7th, 9th**
  - For **V7** chords play: **3rd, 7th, 9th, 13th**
- Type 2:
  - For **Major** and **minor** chords play: **7th, 9th, 3rd, 5th**
  - For **V7** chords play: **3rd, 13th, 7th, 9th**



Rootless - Type 1



Rootless - Type 2

2.2 Monk

Thelonious Monk is a Bebop Pianist known for his dissonant, quirky and asymmetrical style, rhythm and harmony. So, as could be expected, Thelonious Monk Chord Voicings **have a bit of ‘bite’** (read dissonance). They have a semitone interval at the bottom of the chord and a 3rd on top. And because there are only two semitones in a diatonic scale (both Major and melodic minor), then **there are only two possible chord shapes** – one for the tonic chord and one for every other chord in that key.

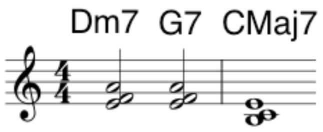
Since there are only two voicings in a key, always played with left hand, this kind of voicings are very easy to remember. This is also the reason why it could be monotonous and dull. If an entire song is played on one key, it can became boring, in fact this kind of voicings are usually played in songs where the key changes often.

These voicings are not really full chords. **They are proto-chords** that exist just to create a general ‘feel’ of a particular key and not to strictly play a particular chord.

Structure

– **Tonic Major Chord**: **7th, 1st, 3rd** (B, C, E in the key of C Major).

– **Every other chord** in the Major Key: **3rd, 4th, 6th of the key** (E, F, A in the key of C Major).



Monk

2.3 Powell

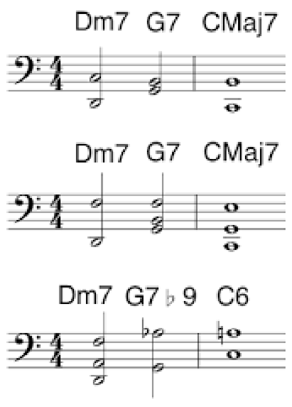
In Jazz, it's possible to omit the less important notes (root & 5th) to create a Shell Chord. **Bud Powell Chord Voicings are also Shell Chords (they contain only two or three notes), but they always include the root** (establishing the tonality of the chord). To facilitate this (and still keep it a Shell Chord) it is dropped either the 3rd or the 7th. Also these voicings are played by the left hand.

This style of chord voicing was popular with Early Bebop Pianists. It was simple, sparse and easy to play. **This was perfect for Bebop** because Bebop solos were fast, busy and harmonically complex. The simplicity of the shell chords contrasted nicely with the complexity of the solos. These voicings worked particularly well for Bebop because:

- They only contained two or three notes, which allowed the soloist to be more harmonically complex without clashing with the chord.
- They used large intervals (7ths & 10ths) which allowed the chord to be played in a low register without sounding muddy.
- They were easy to play at a fast tempo.

Structure

The structure of this kind of voicings can be constructed by taking the **fundamental note and one between the 3rd, 6th, 7th or 10th.**



Powell

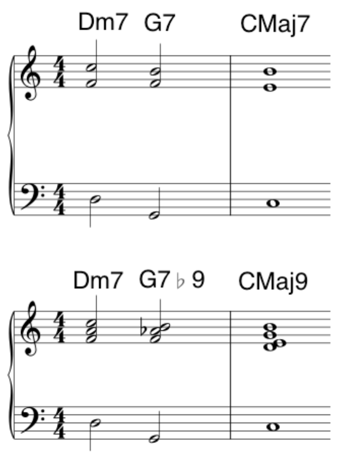
2.4 Three Notes

The basic Shell Chord consists of only the 3rd and 7th (Guide Tones) of the chord. The Guide Tones are the two notes that determine the quality of the chord (Maj, min, V7, etc). These two notes are the bare minimum to play a chord.

An important part of being a competent jazz pianists is being able to voice chords with two hands. Three Note Voicings are a good way of beginning to do this. The structure is very simple, in order to play a Three Notes Voicing it is necessary to **play the Shell Chord with the right hand and the root with the left hand.**

Structure

The root note is played by the left hand while the 3rd and the 7th with the right one.



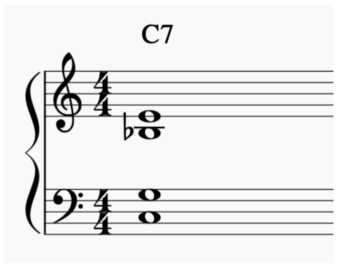
Three Note

2.5 Four Notes

Another good way to start playing voicings with both hands are Four Note Voicings. **Taking the basic Shell Chord, adding a note, usually the 5th, and playing it with right hand and then add a root note in left hand**, we have a 'Four Note Voicing'.

Structure

The root note is played by the left hand while the right one plays the Guide Tones and the 5th.



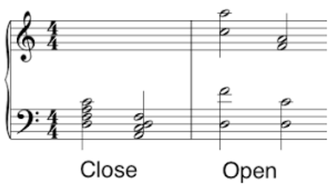
Four Note

2.6 Open Chord

To play in 'close harmony' means to play all the notes of the chord within the range of a single octave. While to play in 'open harmony' means to play all the same notes but over a **span larger than an octave**. This spreads the chord out over a wider range and in this way creates a **richer, more balanced and more 'open' sound**. This is called Voicing Tension.

Structure

- Type 1:
  - The fundamental and the 10th are played by the left hand and the 5th and the 7th are played by the right hand.
- Type 2:
  - The fundamental and the 7th are played by the left hand and the 3th and the 5th are played by the right hand.



Open Chord

3 Coding Music

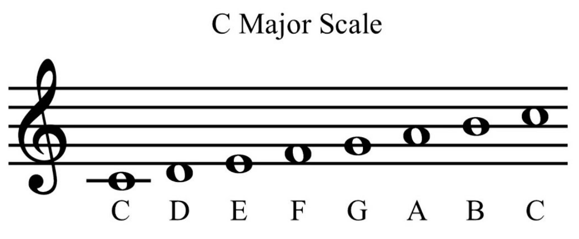
3.1 Dynamic Scales

Modern Western modes use the same set of notes as the major scale, in the same order, but starting from one of its seven degrees in turn as a tonic, and so present a different sequence of whole and half steps.[1]

It can be seen from this description that all modal scales are built basically in the same way, with a minor difference on the starting point from which to pick the first note of the scale. Following this principle, modal scales and their intervals are calculated iteratively with a simple, yet effective algorithm.

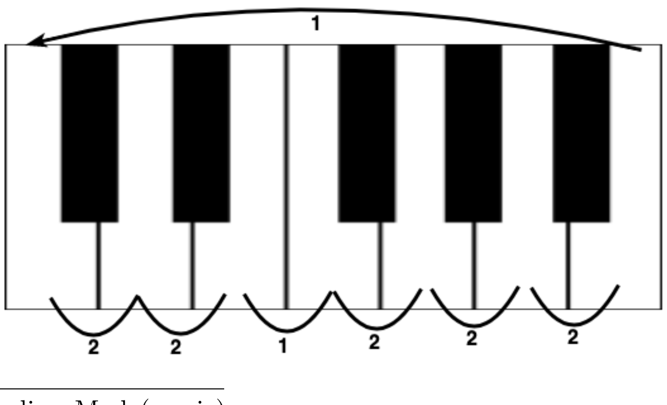
The Approach

Let's start from the most well known musical scale to the western world: the Major Scale.



This is none other than the **first mode: the Ionian Scale**, which is built starting from C and playing all white notes of the piano all the way up to the end of the octave. If we take the intervals between the notes of the ionian scale, counting one for each note (including sharps), we get an array that looks like this:

$$[2, 2, 1, 2, 2, 2, 1]$$



[1]Source: Wikipedia - Mode(music)

Looking at the image above, and knowing that the other modal scales are obtained by shifting the starting note of the scale, it starts to become clear that, to compute the intervals of the other modal scales, it comes down to shifting the array of intervals of the ionian scale. Namely, the other modal scales have intervals:

[2, 2, 1, 2, 2, 2, 1] $\longrightarrow$ [2, 1, 2, 2, 2, 1, 2] **dorian**

[2, 2, 1, 2, 2, 2, 1] $\longrightarrow$ [1, 2, 2, 2, 1, 2, 2] **phrygian**

[2, 2, 1, 2, 2, 2, 1] $\longrightarrow$ [2, 2, 2, 1, 2, 2, 1] **lydian**

[2, 2, 1, 2, 2, 2, 1] $\longrightarrow$ [2, 2, 1, 2, 2, 1, 2] **mixolydian**

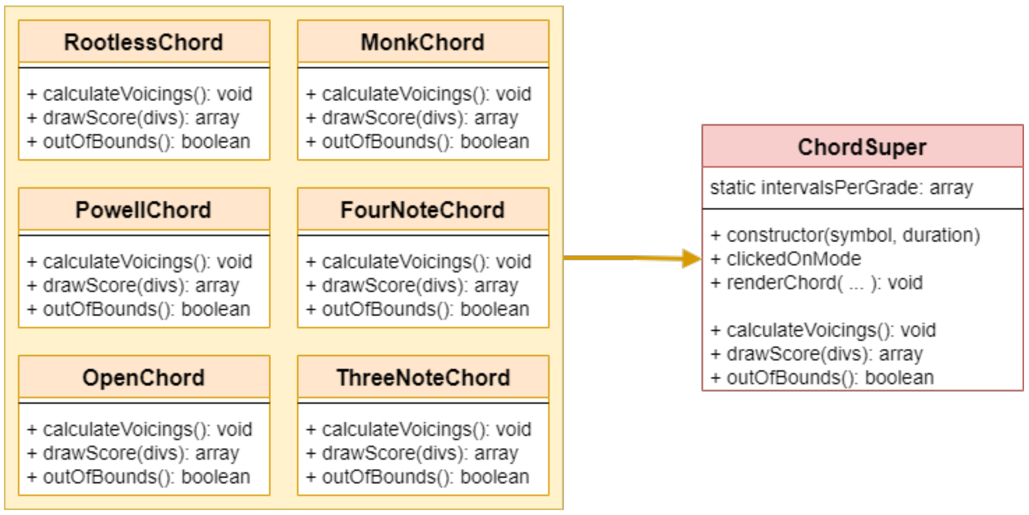
[2, 2, 1, 2, 2, 2, 1] $\longrightarrow$ [2, 1, 2, 2, 1, 2, 2] **aeolian**

[2, 2, 1, 2, 2, 2, 1] $\longrightarrow$ [1, 2, 2, 1, 2, 2, 2] **locrian**

This is also used to calculate the intervals of each grade of a modal scale, in the same exact way (Where the first grade intervals correspond to the intervals of the modal scale, the other are shifted).

3.2 Voicings Algorithm - Factory Pattern

In order to achieve a high flexibility in terms of adding new Voicings Types or removing existing ones, the Voicings Algorithm has been implemented following a simplified **Factory Pattern**.



Factory Pattern

ChordSuper class contains methods which are used by all sub-classes and has other abstract[2] methods which have to be implemented. Those are:

[2]abstract methods do not exist natively on JavaScript, but they can be achieved by throwing an error inside the super-method so that an implementation has to exist inside the sub-class

- **`calculateVoicings(type, intervals):`** constructs the array of voicings based on the sub-type of voicing (for example, Rootless Chords are of two types) and the array of intervals, which depend on the grade of the chord and on the current modal scale.

- **`drawScore(divs):`** draws the score of the chord inside two divs: one for the treble and one for the bass clefs. It returns an array of booleans which tells the GUI whether to show or hide the divs where the two staves are drawn.

- **`outOfBounds():`** checks whether the chord falls between certain integer values so that the voicings can be re-calculated (this will be explained later).

Since every Chord Class knows how to calculate its voicings and draw itself through these methods, the only thing to do to add a new Voicing Type is to create a new sub-class of ChordSuper which implements these three methods.

3.3 Dynamic Shifting

The range of chords that can be constructed and played with *Voicings Generator* is **huge** and could sometimes lead to kind-of-unpleasant chord progressions, where the notes would be counter intuitive to play on the piano due to their distance from one chord to the other.

To solve this problem, the application can automatically shift a chord's octave according to the position of the chord that came before. This helps the progression in keeping a certain coherence, but it also adds another major problem. Some progression could, in fact, make it so that consequent automatic shifts of octaves occur making the chords fall "out of range", in the sense that they become too grave or acute and, again, sound unpleasant.

This is why each Chord Class has the `outOfBounds` method: if a chord exceeds certain boundaries set by this method (unique for each voicing type), the application recalculates the voicings starting from scratch, effectively "resetting" the algorithm.

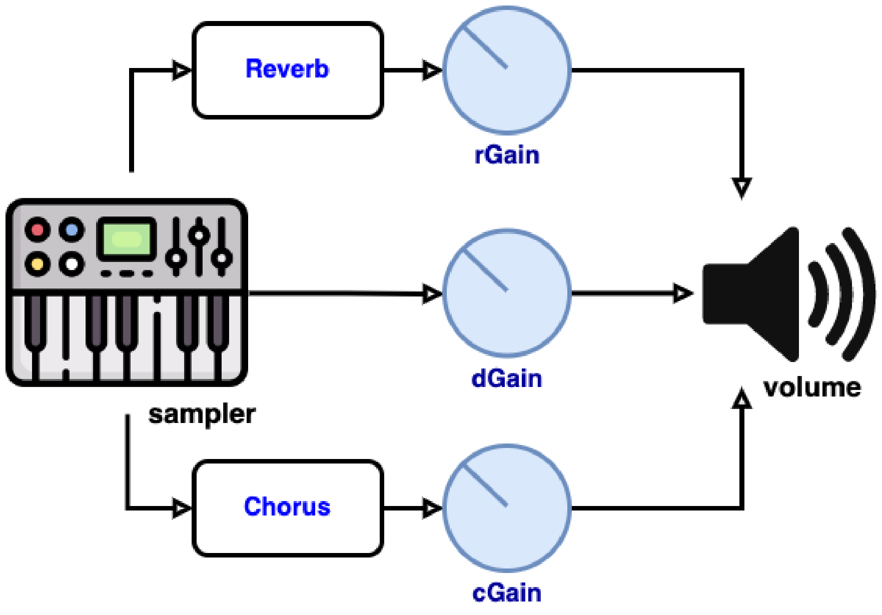
3.4 The Instrument

After having obtained the voicings of the chords sequence, the only thing left to do is to play them. This is done with the use of the `Instrument` class, which contains a sampler, to produce audio, and a couple of audio effects.

The Sampler

`Tone.js` has a sampler object which is able to automatically assign each note frequency to a sound, given a set of samples fetched on a base URL where files are stored. With those, `Tone`'s sampler is able to automatically fill the spaces between samples by shifting their pitch.

The Audio Chain



Instrument Audio Chain

The sampler is connected in parallel with two effects, **Reverb and Chorus**, each connected to a Gain node that controls their gain. The sampler is also connected to another Gain node which is used to deliver the dry output. All three nodes are connected to another Gain node which controls the overall volume of the instrument.

The instrument can then be connected (through the `connect` method) to another node, so that we can control the volume of all instruments simultaneously with a single master Gain node.

Playing Sounds

In order to deliver the actual sound, `Tone`'s sampler needs to convert the notes' numbers (which, like MIDI, range from 0 to 127) into frequencies. For this purpose, an array of 128 frequencies is pre-computed so that the right frequency of a note n can immediately be obtained by accessing the array in position n.

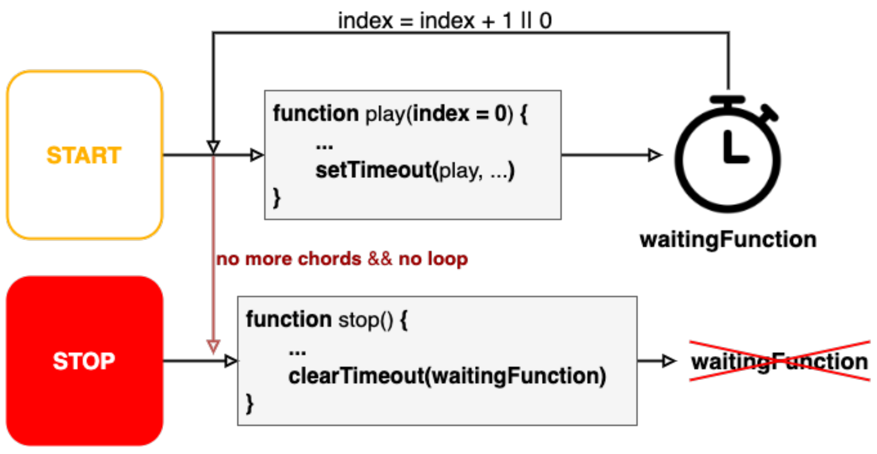
Attack and release phases of the instrument are handled separately because of the way the application works.

4 The Main Loop

Voicings Generator's core comes down to a single loop which starts when the user presses the start button and stops either when the chords sequence is finished (if the loop mode is off) or when the user clicks the stop button. This loop is not an actual `while` or `for` loop, but rather a recursive function which continuously calls itself after a certain time has passed.

4.1 `setTimeout()`

The global `setTimeout()` method sets a timer which executes a function or specified piece of code once the timer expires.[3]



```
function play(index = 0) {
    ...
    setTimeout(play, ...)
}
```

```
function stop() {
    ...
    clearTimeout(waitingFunction)
}
```

When the start button is pressed, the first chord of the sequence is played and a new timeout is set containing the same function used to play the first chord, but with a different index pointing to the next chord of the sequence. Same happens when the second chord is played and so on. This effectively creates a loop and, since the timeout time is decided by the bpm of the application, when the user changes said bpm the time is updated right after the current loop iteration making the application responsive to user input.

When the user presses stop or there are no more chords to play in the sequence, any waiting function is deleted and the instrument's release is immediately invoked. That is, if the loop option is set to off. If, otherwise, the loop functionality is on, when the chord sequence is finished a new timeout is set with the same play function but with an index pointing to the first chord of the sequence, resetting the loop to initial conditions.

4.2 Loop Mode

By clicking the Loop button (*two arrows forming a circle inside a square*), the user can activate the **Loop Mode**. This makes it so that when the chords progression is over, instead of stopping the execution of chords the application

[3]Source: MDN web docs

restarts the loop, returning to the same conditions as when the user first clicked the Start button.

4.3 Legato Mode

By clicking the Legato button (*two notes joined by an arch below them*), the user can activate the **Legato Mode**. When the application is in Legato Mode, chords which are equal in grade and subsequent will be played as a single chord with a duration which corresponds to the number of subsequent chords before the end of the sequence or another chord is played.

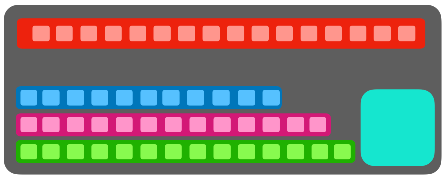
For example, if the chords sequence is `| I | II | II | III |`, the second chord (`II`) will be played once with a duration of two beats.

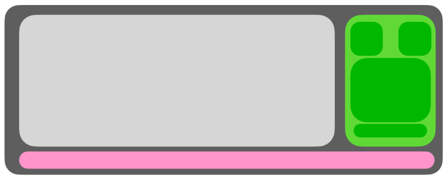
5 GUI

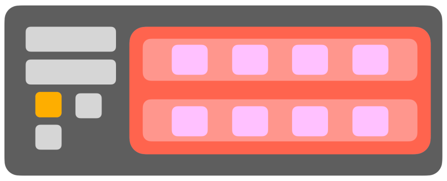
5.1 A React-ive environment

The entirety of the user interface has been developed using `React`, a well known and supported module which introduces the concept of components. This approach proves to be really useful as it increases the usability and clarity of the code and facilitates its implementation.

Three main components have been developed, in the following images a schematics of their structure and inner components.


(p) ChordsTable.js


(q) ChordsVisualizer.js


(r) VoicingsSelector.js

5.1.1 ChordsTable.js

In this section the user selects the key signature, the modal scale and the grade that he wants to play. With a drag and drop it is possible to change the order of the grades, or with a double click it is possible to delete it. Once the user is satisfied, he can press the `StartButton` and listen to the voicings.

5.1.2 ChordsVisualizer.js

This component is divided in three parts:

- ChordScore: This section displays three chords on the musical scope, hilighting the once that's playing.
- GlobalSettings: This component contains two components himself.
  - CheckButtons: Two buttons for legato and loop.
  - OctaveSelector: This is active only for few voicings. With this button the user can change the octave *Voicings Generator* is playing.
- Piano: This is React-piano, hilights the keys that has to be pressed in order to play the chord and, if there is a MIDI plugged in, hilights also the keys the user is pressing on his physical keyboard.

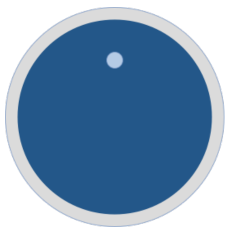
5.1.3 VoicingsSelector.js

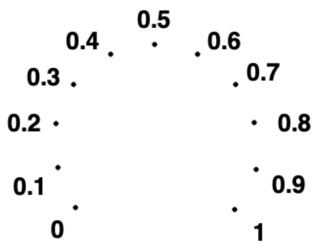
This component is composed by 2 `Selector`, 1 `SettingsButton`, 1 `CheckButton` and 1 `PrinterButton`

By default *Voicings Generator* will show the name and the description of the voicing the user is using. By clicking on a button, the page will dynamically change in order to display the knobs for the effects, the knobs for the volume control or the prompt of the print.

5.2 Knobs

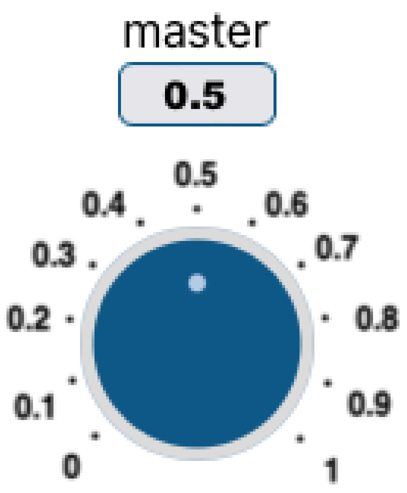
To make the application as understandable as possible, knobs has been developed to manage all the modifications of the audio parameters. Each knob consists of the image of the knob, the graduated crown around it to indicate the values and make the modification more accurate and a text box that shows the instant value of the knob.


(s) Knob



(t) Graduated crown

To manage the rotation of these objects is been developed a function called `rotation()`, which, understanding the position of the mouse, rotates, via `CSS`, the knob image. The same function has the task of extrapolating the value of the instant rotation and display it in the text box above the knob. The value of each for better accuracy can be written directly in the text box, the knob will rotate accordingly.



Complete image of a knob

6 Conclusions

Voicings Generator is many things: a powerful, yet simple tool to explore and study jazz voicings, an innovative way to write short pieces of harmonies, a fun musical playground to experiment and hear interesting chords sequences.

6.1 Results

The application behaves well in finding the right voicings for any chord sequence, regardless of root key and modal scale. We did some research and found videos online where jazz standards are played by a person with different types of voicings and the scores match those generated by the application the majority of times.

Voicings Generator is far from being perfect though, as it sometimes behaves in unexpected ways, making way for future improvements.

6.2 Possible future addictions

There is a couple of things to add which would extend the application's functionality and make for an overall better user experience. The most notable are:

- A server implementation with a user login system to store and fetch previously made chords sequences
- A metronome to help when playing a melody through MIDI input
- More instruments to chose from instead of the classic piano
- More voicings types
- A chord recognition system (which would require artificial intelligence to work properly)