TabMaker

Group members: Riccardo Di Bella, Stefano Ravasi, Yan Zhuang

ACTM – CMRM Project

A.Y. 2022/23

MAIN GOAL

- Convert a chord sequence into a feasible and «musical» voicing sequence on guitar
  - Follow voicing principles
- Sequence is found algorithmically
  - More possibilities
  - Support alternate tunings

FEATURES

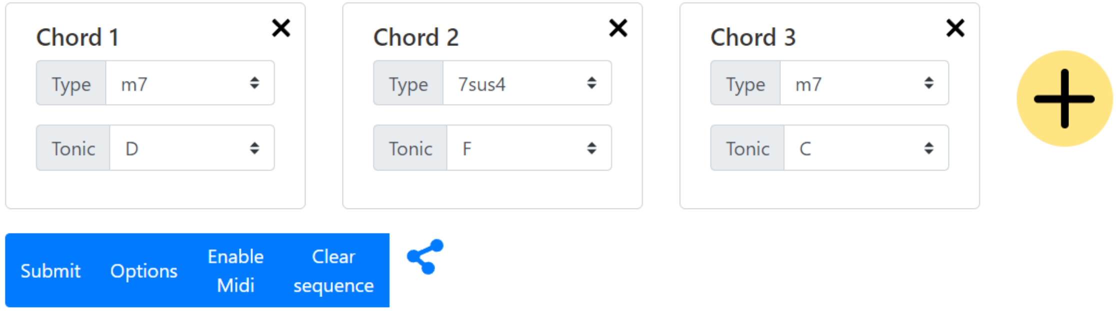
Chords selection: input the chord sequence

Sequence player: listen to the result

Fretboard/TAB visualization: view result on fretboard or TAB format

GUI

- Top part dedicated to the input
- Add or remove chords from the sequence
  - Specify **root** and **type**
- **Submit** button generates the voicing sequence
- **Options** button
- **Enable MIDI** button
- **Clear sequence** button
- Chord cards can be moved around



Chord 1
Type: m7
Tonic: D

Chord 2
Type: 7sus4
Tonic: F

Chord 3
Type: m7
Tonic: C

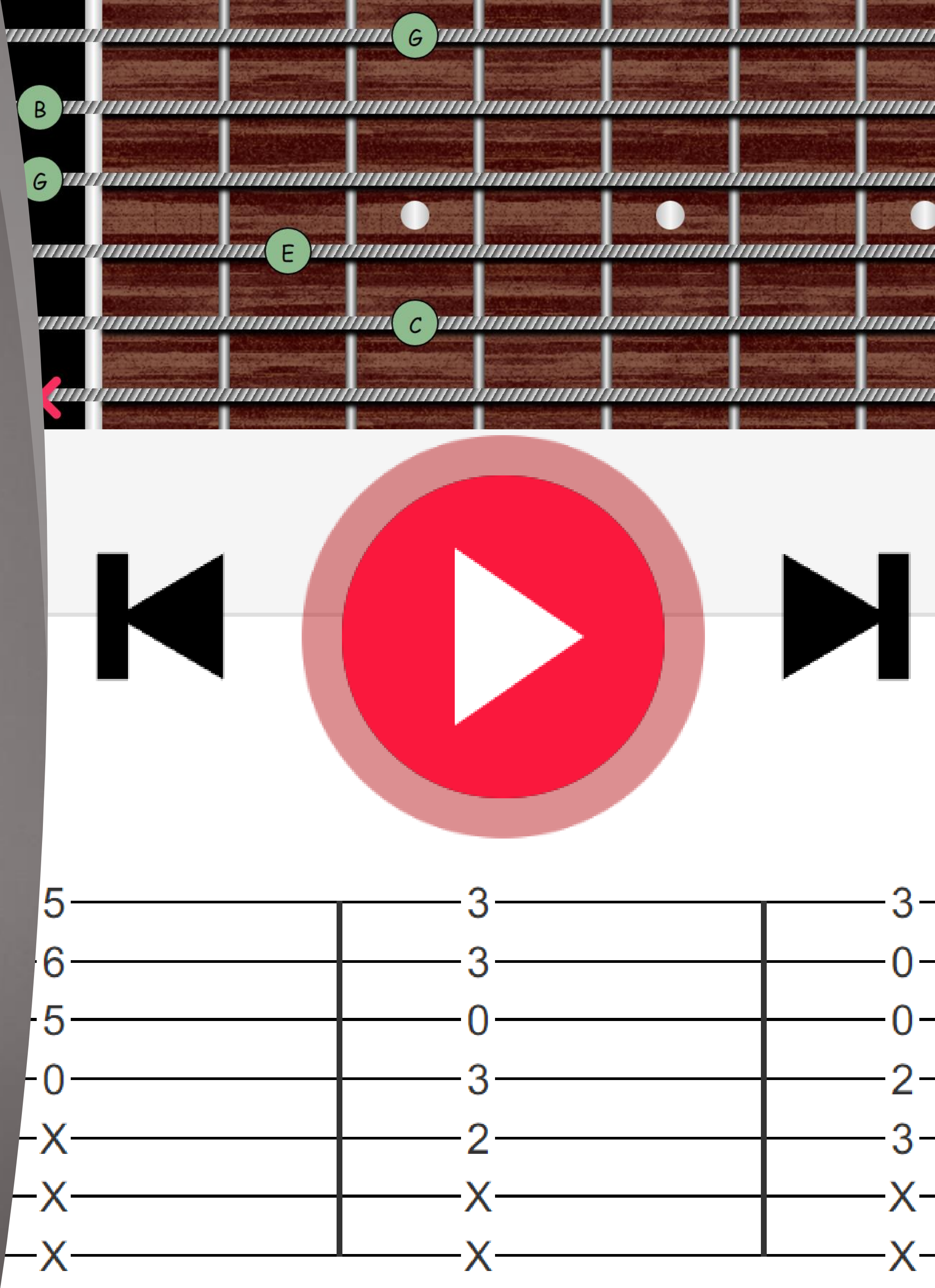
GUI

- The lower section of the GUI is devoted to chord and sequence playing/visualization:

- **Play/forward/backward** buttons to listen to the sequence

- Change view between **fretboard** or **TAB**

- **Tuning** menu

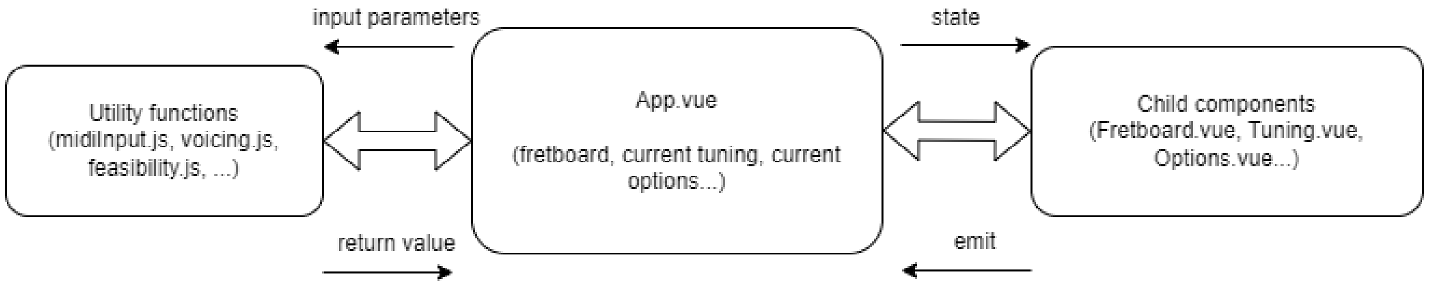


Frameworks and Technologies Used

- Entire application is developed in Javascript/HTML/CSS
- Vue.js framework used for the GUI
- Web MIDI API for MIDI interaction
- Tonal js music theory library
  - Chords notes
  - Musical intervals
- AudioSynth
  - Generating guitar sound for chord player

Code Structure

- Main component: **App.vue**
- Child components:
  - Input: **Tuning.vue, Options.vue, ToggleButton.vue**
  - Output: **Fretboard.vue, Tablature.vue**
- Utility functions:
  - **voicing.js**
  - **feasibility.js**
  - **freatbordModel.js**
  - **midiInput.js**
  - **note.js**
  - **sound.js**


input parameters
state
Utility functions (midiInput.js, voicing.js, feasibility.js, ...)
App.vue
(fretboard, current tuning, current options...)
Child components (Fretboard.vue, Tuning.vue, Options.vue...)
return value
emit

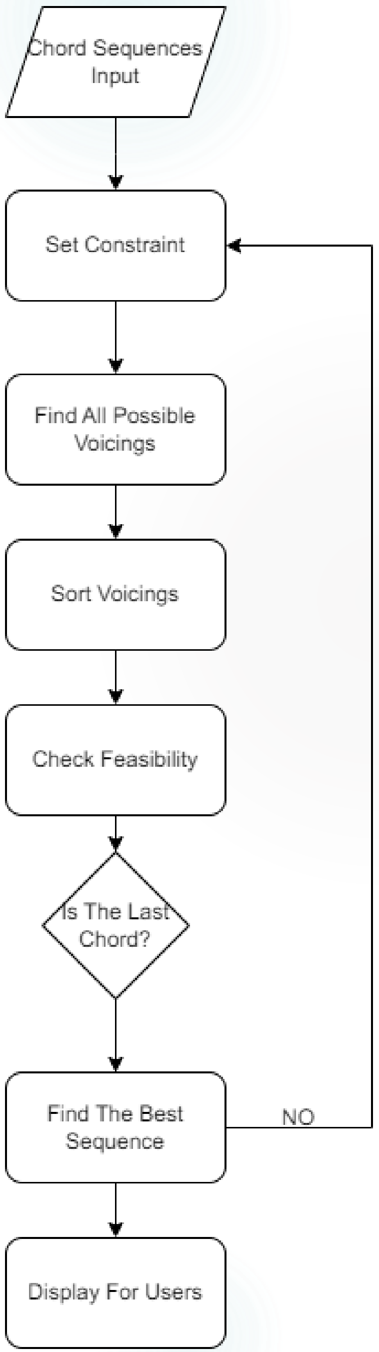
fretboardModel.js

- Data model to represent the guitar fretboard as a matrix of Note objects (note.js)
  - Note: pitch class and octave
  - Equality supports enharmonic representations
- Functions to analyze the fretboard:
  - getNote(position)
  - findPositions(fretboard, note, ignoreOctave)

feasibility.js – Checking feasibility

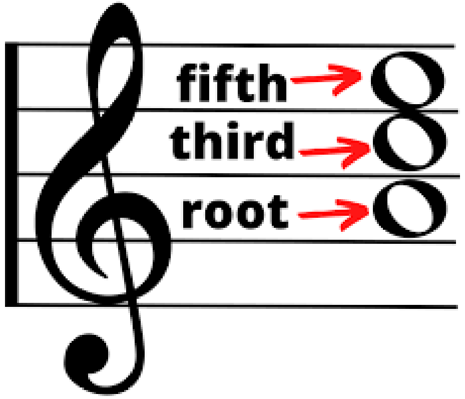
- First check: only search within limited range
- Second check:
  - More complex rules
  - Assign an integer to each finger → **fingers list = [0, 1, 2, 3]**
  - Assign a finger to each pressed fret
  - Easy to implement checks
    - E.g. finger crossing check → ***(finger-usedFinger)*(currentFret-usedFret) < 0***
  - Find a finger-frets assignment that doesn't violate constraints

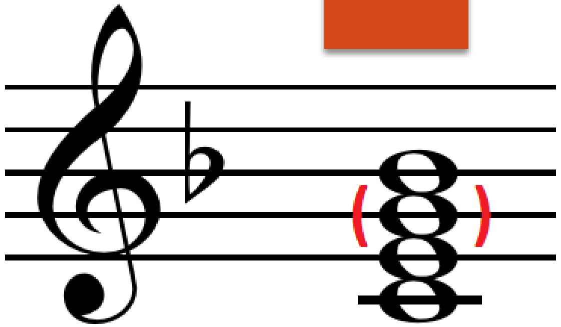
voicing.js - Voicing Algorithm Overview



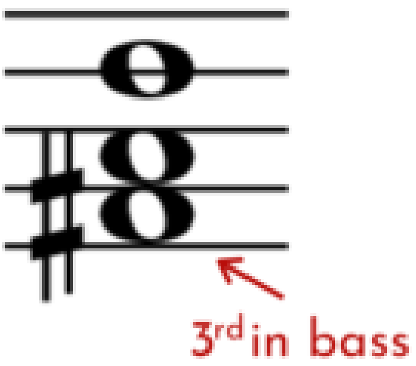
voicing.js – positions search

- ***recursivePositionSearch*** finds all possible positions for a chord
  - Consider all possible «paths» from the top to the bottom string
    - ***findNextPositions*** *find all valid positions on the next string*
  - Stops as soon as it violates constraints

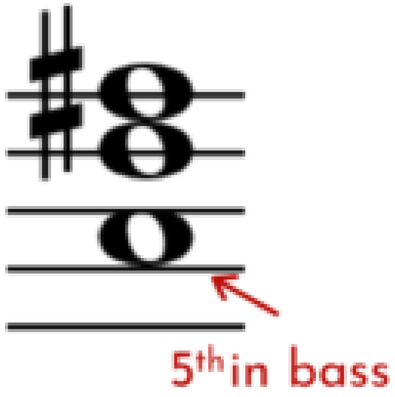




1st inversion



2nd inversion



voicing.js – Voicing Sequence Building

- Find “optimal” sequence:
  - **Distance** between consecutive chords is minimized
  - Number of **tritone resolutions** between consecutive chords is maximized
  - Only a local **short “window”** of chords is considered (4 chords)

midiInput.js – Handle MIDI input

- Chord recognition:
  - Consider notes within a 200 ms window
  - When we have at least 3 notes
    - Recognize chord using ***Chord.detect(notes)*** *from tonal.js library*
    - *Use **callback** to App.vue to append chord to sequence*

CONCLUSIONS AND FURTHER DEVELOPMENTS

- Chord sequences are playable and enjoyable from a musical perspective
- Simple but effective GUI
- Implement **more interesting voicing rules**
- Add more user options (e.g. **interacting** with the resulting sequence)
- Add **musical staff notation** for chord input/visualization