Chord Identifier and Visualizer on Piano

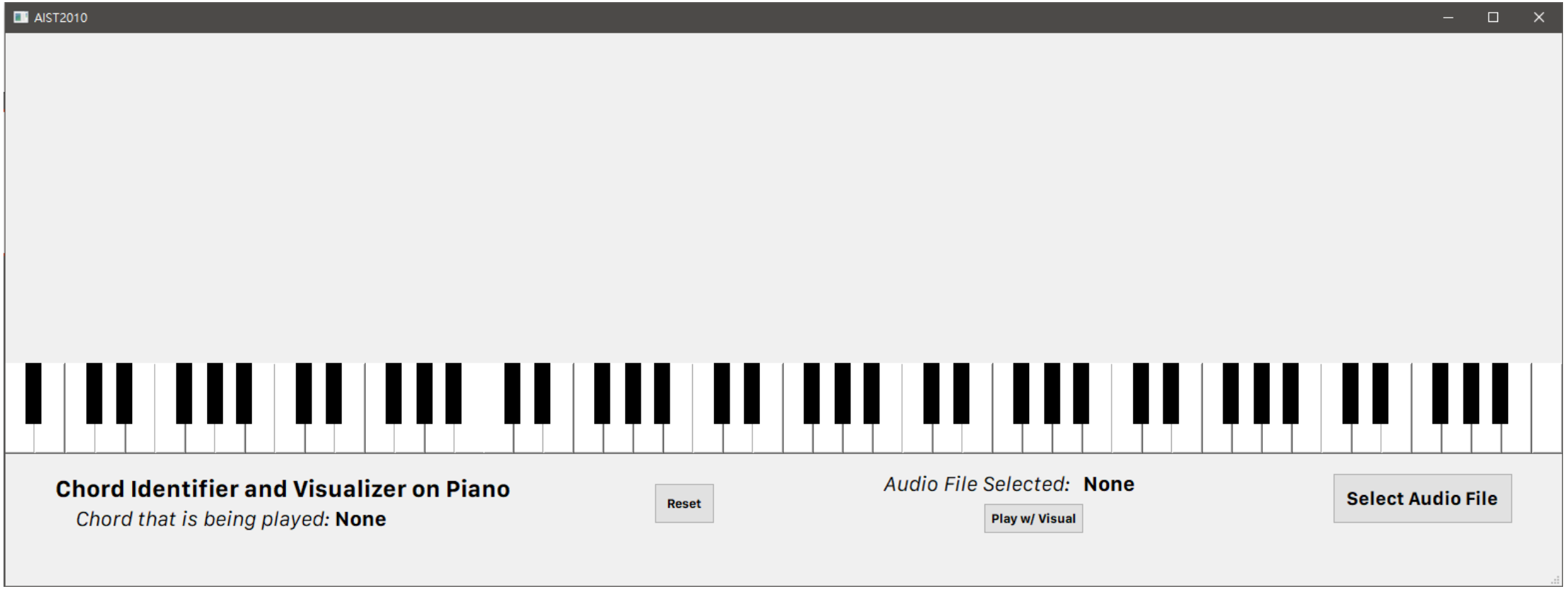
Purpose

- Piano visualization
- Chord Identifier
  - 1) MIDI keyboard input
  - 2) Audio file input
    - Restriction: only audio generated by pure sine wave works

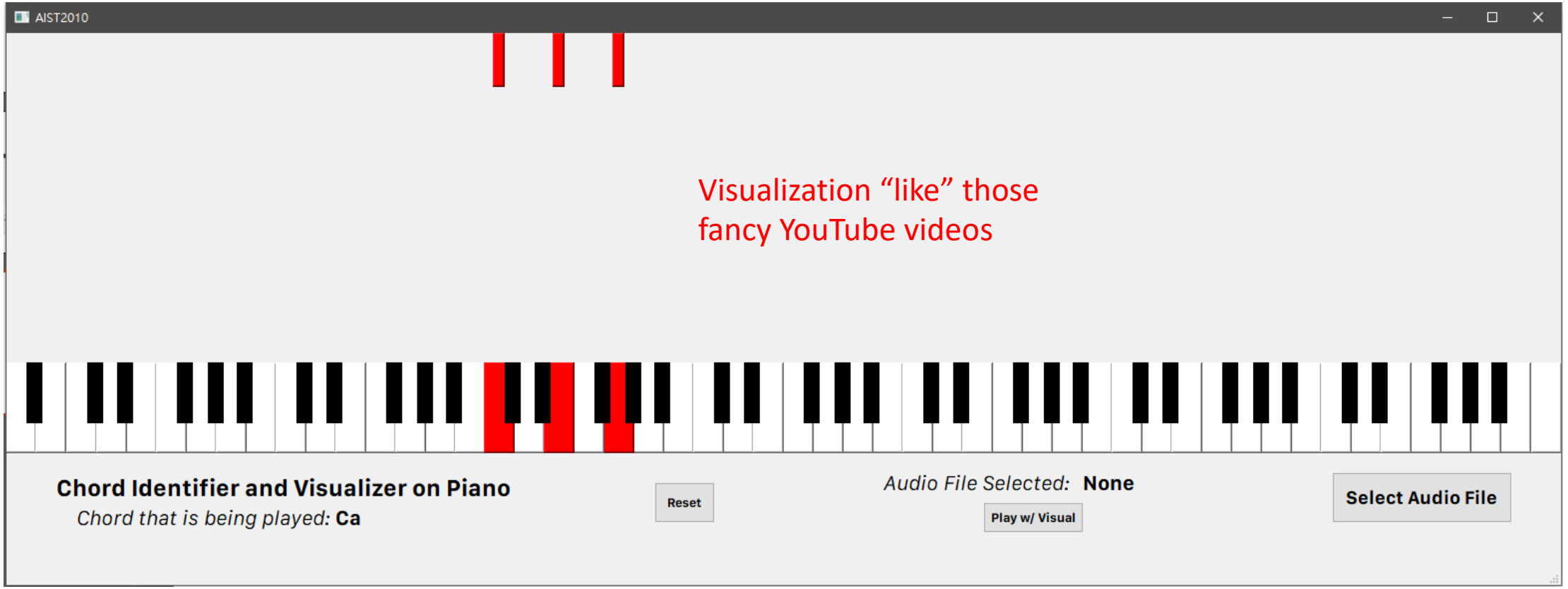
GUI

- By PyQt5 (https://doc.qt.io/qtforpython/)
- trigger and animation
  - Trigger: a signal to be sent when triggered
    - For transmitting notes played, signal to start animation and many more
  - Animation: smooth visualization
- Piano notes will be played by Pygame
  - Built-in MIDI module

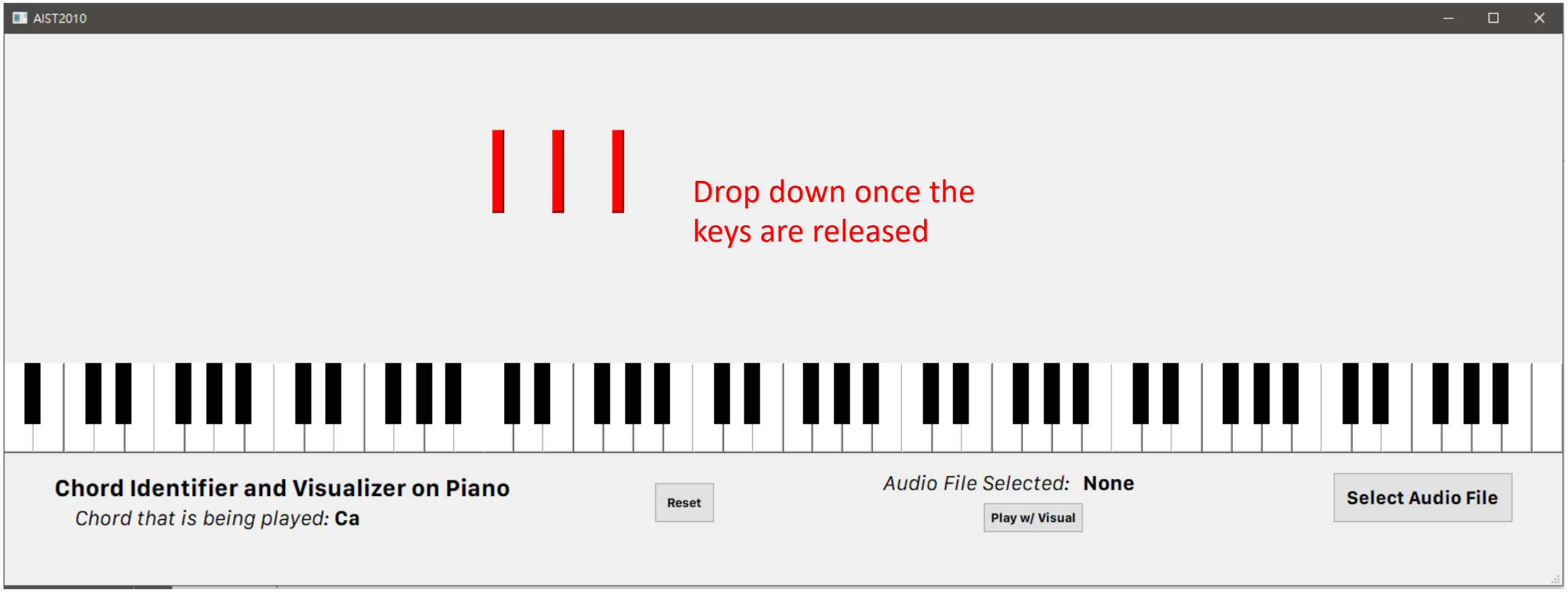
GUI


AIST2010
Chord Identifier and Visualizer on Piano
Chord that is being played: None
Reset
Audio File Selected: None
Play w/ Visual
Select Audio File

GUI


AIST2010
Visualization “like” those
fancy YouTube videos
Chord Identifier and Visualizer on Piano
Chord that is being played: Ca
Reset
Audio File Selected: None
Play w/ Visual
Select Audio File

GUI

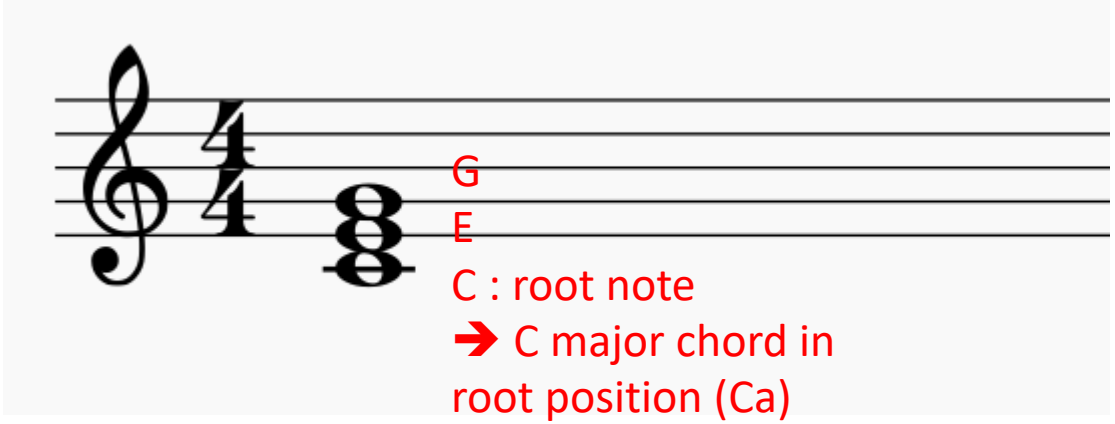

AIST2010
Drop down once the keys are released
Chord Identifier and Visualizer on Piano
Chord that is being played: Ca
Reset
Audio File Selected: None
Play w/ Visual
Select Audio File

How to identify chords?

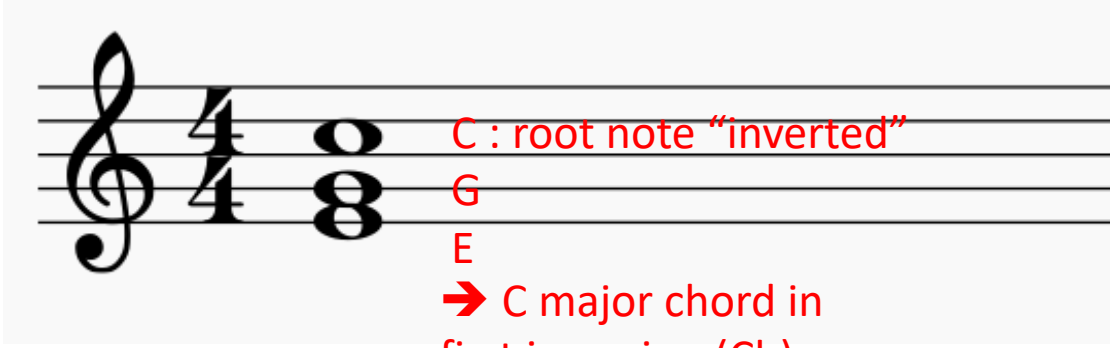
- By looking at the notes being played/in the audio file…

Music Theory Time

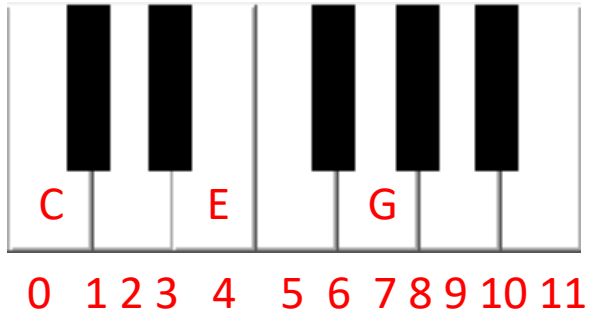
- Chords
  - Root note + Quality + Inversion
  - e.g.,



G
E
C : root note
➔ C major chord in root position (Ca)

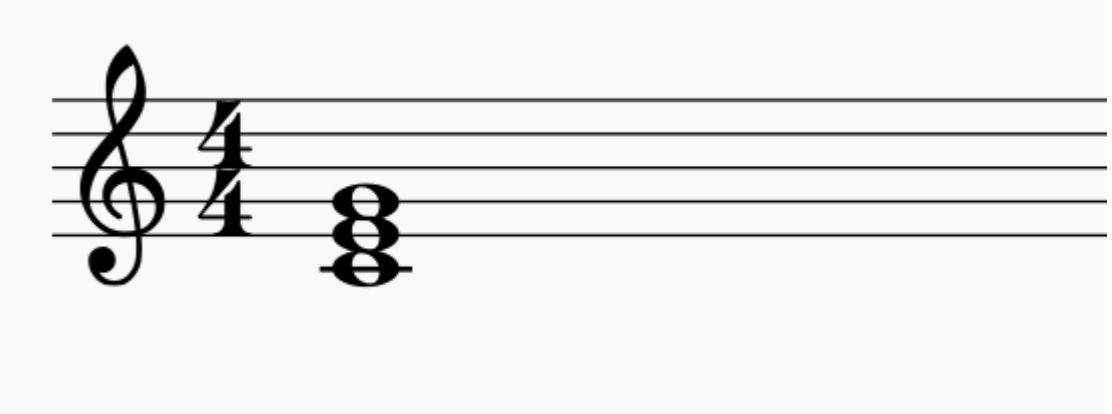


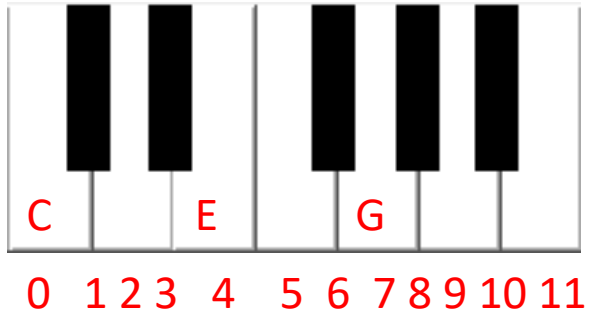
C : root note “inverted”
G
E
➔ C major chord in first inversion (Cb)



Music Theory Time

- Quality + inversion can be UNIQUELY determined by the distance between each consecutive notes of a chord!
- e.g.,





distance between C and E = 4 – 0 = 4

distance between E and G = 7 – 4

dis = [4,3] ➔ major chord in root position ➔ The first note is the root

Music Theory Time

```
QUALITY = {(4, 3): ["a", 0], (3, 5): ["b", 2], (5, 4): ["c", 1],
           (3, 4): ["ma", 0], (4, 5): ["mb", 2], (5, 3): ["mc", 1],
           (4, 3, 3): ["7a", 0], (3, 3, 2): ["7b", 3], (3, 2, 4): ["7c", 2], (2, 4, 3): ["7d", 1],
           (2, 2, 3, 3): ["9a", 0], (3, 3, 2, 2): ["9b", 3], (3, 2, 2, 2): ["9c", 2], (2, 2, 2, 3): ["9d", 1],
           (2, 3, 3, 2): ["9e", 4],
           (3, 3, 3): ["dim7", 0],
           (3, 3): ["dima", 0],
           (4, 4): ["+", 0],
           (2, 5): ["sus2", 0], (5, 2): ["sus4", 0],
           (4, 6): [" Italian 6th", 1]}
```

- QUALITY: Python dictionary with

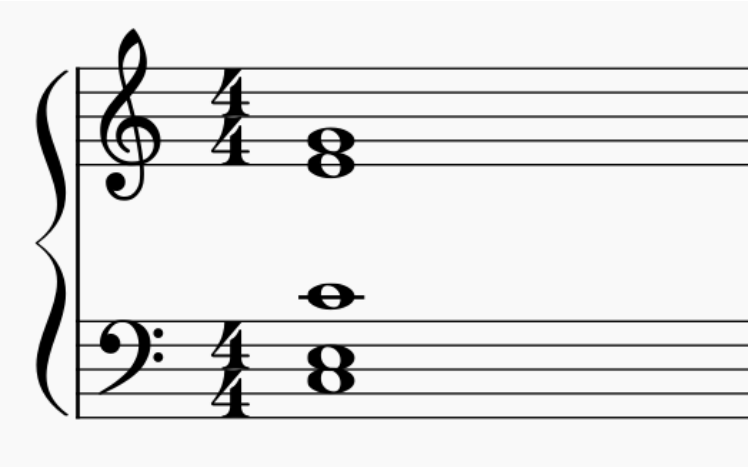
  key = tuple of distances, value = [quality + inversion, root]
- Can be easily extended to include many other types of chords
  - e.g., major seventh, major ninth, …

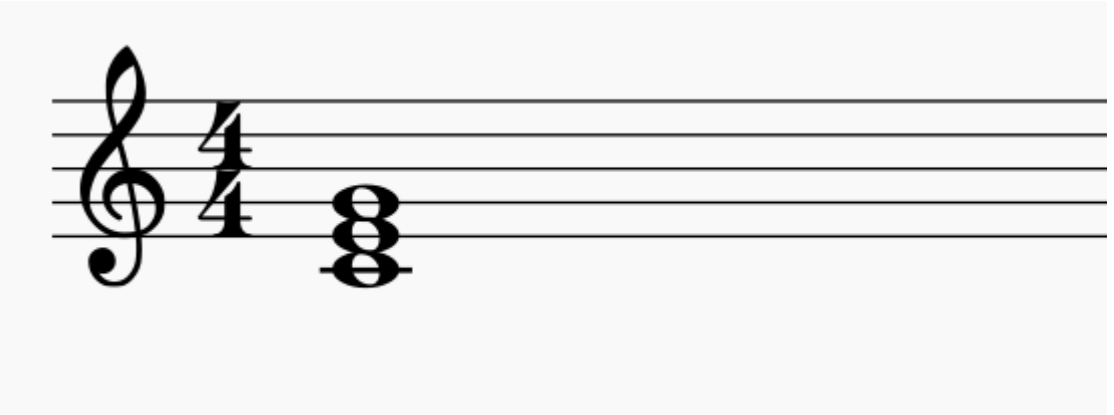
In other words…

- If we can find the notes being played either on the MIDI keyboard or in the audio file…
- we can find the chord by calculating the distance and then finding the value in the dictionary QUALITY
- However, chords can be played with different voicing

Implementation of Chord Identifier

- e.g.,





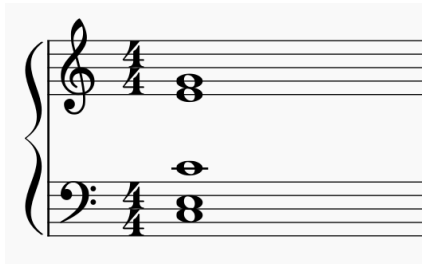
- They are both C major chord in root position
  - Only the unique notes and the lowest note matter
    - C, E and G, with the lowest note being C → Ca

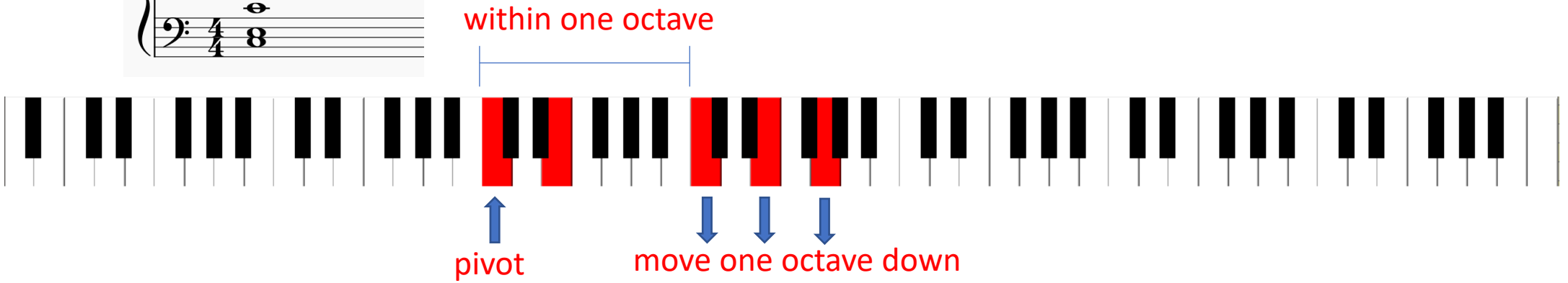
Implementation of Chord Identifier

- Idea:
  - Use the lowest note as the pivot
  - move each of the notes above down an octave at a time, until all notes are within one octave of the lowest note

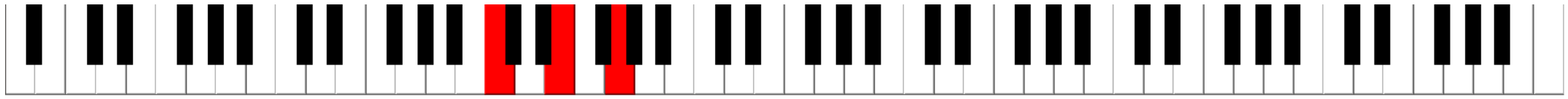
Implementation of Chord Identifier

- e.g.,





➔



- Q: How to “move one octave down”?
- A: use MIDI number

MIDI number

Octave	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-2	0	1	2	3	4	5	6	7	8	9	10	11
-1	12	13	14	15	16	17	18	19	20	21	22	23
0	24	25	26	27	28	29	30	31	32	33	34	35
1	36		38	39	40	41	42	43	44	45	46	47
2	48	49	50	51	52	53	54	55	56	57	58	59
3	60	61	62	63	64	65	66	67	68	69	70	71
4	72	73	74	75	76	77	78	79	80	81	82	83
5	84		86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107
7	108	109	110	111	112	113	114	115	116	117	118	119
8	120	121	122	123	124	125	126	127				

Same notes in successive octaves have MIDI numbers differ by 12

- Source: https://djip.co/blog/logic-studio-9-midi-note-numbers

Implementation of Chord Identifier

```python
# function for detecting the qualities of chords
# input: a list containing notes
# output: a string of full name of chord (bass note, quality, inversion)
def full_chord(noteMIDI):
    new_noteMIDI = noteMIDI
    # sort the noteMIDI list
    new_noteMIDI.sort()
    # the lowest note is the pivot, move other note down octave(s), so that
    # all notes are within one octave from the lowest note
    for i in range(len(new_noteMIDI) - 1):
        while new_noteMIDI[i + 1] >= new_noteMIDI[0] + 12:
            new_noteMIDI[i + 1] -= 12  # 12 notes in one octave
    # now that all notes are with in one octave of each other, remove duplicate notes, then sort again
    new_noteMIDI = list(set(new_noteMIDI))
    new_noteMIDI.sort()
    # find distance between each note
    dis = []
    for i in range(len(new_noteMIDI) - 1):
        dis.append(new_noteMIDI[i + 1] - new_noteMIDI[i])
    # this gives us the quality and the inversion of the chord
    result = QUALITY.get(tuple(dis))
    if result is None:
        return "No chord is found!"
    # find the root note of the chord
    root = librosa.midi_to_note(new_noteMIDI[result[1]], octave=False)

    return root + result[0]
```

Move each notes one octave downward at a time

Find the distances

Search the dict QUALITY

Use librosa.midi_to_note to change the MIDI number to char

How about audio file?

- Audio files are very complicated to analyze…
- Harmonics ➔ usual algorithm learnt in lecture (e.g., STFT) does not work
- Missing harmonics: practically, notes played on different instruments will have different harmonics being omitted
- The fundamental frequency can also be missing…

Look at an easier case

- So I first look at how to extract notes from audio generated by pure sine waves
- SuperCollider:

```
(
{SinOsc.ar(440,0,0.2) + SinOsc.ar(523.25,0,0.2) + SinOsc.ar(1318.51,0,0.2)}.scope;
s.record(duration: 3);
)
```

Analyzing audio file

- Turns out librosa has a built-in function *librosa.piptrack:*

librosa.piptrack

librosa.piptrack(y=None, sr=22050, S=None, n_fft=2048, hop_length=None, fmin=150.0, fmax=4000.0, threshold=0.1, win_length=None, window='hann', center=True, pad_mode='reflect', ref=None) [source]

Pitch tracking on thresholded parabolically-interpolated STFT.

This implementation uses the parabolic interpolation method described by [1].

[1] : https://ccrma.stanford.edu/~jos/sasp/Sinusoidal_Peak_Interpolation.html

- *https://librosa.org/doc/main/generated/librosa.piptrack.html*

Analyzing audio file

- The output of *librosa.piptrack* is two Numpy arrays, storing the instantaneous frequencies and the corresponding magnitudes at any given bin and time

Analyzing audio file

- If a frequency is present at an instant
  - the entry corresponding to the bin and the time can be regarded as the frequency
  - Otherwise, the entry will be zero
- Finding nonzero entries = finding the frequencies present in the audio file

```
def detect_pitch(p, t):
    index = numpy.nonzero(p[:, t])
    pitch = p[index, t]
    return pitch
```

Analyzing audio file

- Harmonics will still be present, but we know that harmonics are multiples of the fundamental frequency

```python
# boolean: True if is harmonics, False if is not
def checkHarmonics(freq1, freq2):
    # freq 2 is larger
    result = freq2 / freq1
    if result.is_integer():
        return True
    else:
        return False
```

Analyzing audio file

```python
# return a list containing chord notes from the audio input
# https://librosa.org/doc/main/generated/librosa.piptrack.html
def audioAnalyze(y, sr):
    # librosa.piptrack to extract pitch
    pitches, magnitudes = librosa.piptrack(y=y, sr=sr)
    # the first ~10 columns are often meaningless
    pitches_candidate = []
    for time in range(10, len(pitches[0])):
        pitches_candidate.append(detect_pitch(pitches, time))
    # onset_detection: assumption: chord usually happen just after onset
    onset_frames = librosa.onset.onset_detect(y, sr)
    chord_note_candidate = []
    for x in onset_frames:
        if x <= len(pitches_candidate):
            # so that the frequencies can be conveniently rounded to exact frequencies for excluding the harmonics
            chord_note_candidate.append(librosa.note_to_hz(librosa.hz_to_note(pitches_candidate[x])))
    chord_note_no_harmonics = []

    for x in chord_note_candidate:
        # x is a list of list (...), so x[0] corresponding to freq in an onset
        temp = []
        for i in range(0, len(x[0])):
            if i == 0:
                temp.append(x[0][i])
            for j in range(0, i):
                if j == i - 1:
                    temp.append(x[0][i])
                    break
                elif checkHarmonics(x[0][j], x[0][i]):
                    break
        chord_note_no_harmonics.append(temp)

    # https://stackoverflow.com/questions/3724551/python-uniqueness-for-list-of-lists
    chord_note_no_harmonics = [list(x) for x in set(tuple(x) for x in chord_note_no_harmonics)]

    return chord_note_no_harmonics
```

When I plot the chromagram, the start of the chromagram is often noisy → disregard the first 10 columns

Chord changes often happen at the start of an onset
→ Look at the start of the onset to find chord note candidates

Use the checkHarmonics function defined on the last slide to eliminate the harmonics

Limitation

- Only analyzing pure sine waves is possible
  - Neural network trained by chromagram of different chords can be used to classify the audio
  - Separate the audio by the onsets (Lab4), then fit each subsection into the neural network
- PyQt5
  - Everything must be programmed by using trigger
    - Lag ➔ bugs in GUI because of timing issues