Keyboard_Pi_Conductor/PythonPiano/key_midi.py

```
import pygame
 2
    import PythonPiano.piano_lists as pl
    from pygame import mixer
    from music21 import note, stream, tempo
 4
    from threading import Thread
    import time
 6
 7
 8
 9
    class AudioRecorder:
        def __init__(self, save_path, bpm=90):
10
            print('initializing new audio recorder')
11
            self.save path = save path
12
            self.bpm = bpm
13
            self.metronome mark = tempo.MetronomeMark(number=self.bpm)
14
            self.left oct = 4
15
            self.right oct = 5
16
17
            self.white notes = pl.white notes
18
            self.black notes = pl.black notes
            self.black labels = pl.black labels
19
2.0
            # windowSurfaceObj = pygame.display.set_mode((64,48),1,16)
21
22
            self.initialize(bpm)
2.3
        def initialize(self, bpm=None):
24
25
            if bpm is not None:
                self.bpm = bpm
26
                self.metronome mark = tempo.MetronomeMark(number=self.bpm)
27
            self.score = stream.Stream()
28
            self.score.append(self.metronome_mark)
29
            pygame.mixer.init()
30
            pygame.mixer.set num channels(50)
31
32
33
            self.white_sounds = []
            self.black sounds = []
34
35
            for i in range(len(self.white notes)):
36
37
                self.white sounds.append(mixer.Sound(
38
    f'/home/pi/Documents/ECE5725 final proj/PythonPiano/assets/notes/{self.white notes[i
    ]}.wav'))
39
40
            for i in range(len(self.black notes)):
                self.black sounds.append(mixer.Sound(
41
```

```
42
    f'/home/pi/Documents/ECE5725 final proj/PythonPiano/assets/notes/{self.black notes[i
    ] } .wav'))
43
44
            self.note with time = []
            self.left dict = {'Z': f'C{self.left oct}', 'S': f'C#{self.left oct}', 'X':
45
    f'D{self.left_oct}', 'D': f'D#{self.left_oct}',
46
                               'C': f'E{self.left_oct}', 'V': f'F{self.left_oct}', 'G':
    f'F#{self.left_oct}', 'B': f'G{self.left_oct}',
                               'H': f'G#{self.left_oct}', 'N': f'A{self.left_oct}', 'J':
47
    f'A#{self.left_oct}', 'M': f'B{self.left oct}'}
48
49
            self.right dict = {
50
                'R': f'C{self.right oct}',
                '5': f'C#{self.right oct}',
51
                 'T': f'D{self.right oct}',
52
53
                 '6': f'D#{self.right oct}',
                 'Y': f'E{self.right oct}',
54
                'U': f'F{self.right oct}',
55
56
                '8': f'F#{self.right_oct}',
                'I': f'G{self.right_oct}',
57
                '9': f'G#{self.right oct}',
58
                'O': f'A{self.right_oct}',
59
                '0': f'A#{self.right oct}',
60
                 'P': f'B{self.right oct}'
61
62
            }
63
        def start recording(self):
64
65
            self.recording = True
            self.record()
66
67
68
        def process event(self, event):
69
            if event.type == pygame.QUIT:
70
                self.clean()
71
                return
72
            if event.type == pygame.TEXTINPUT:
73
                if event.text.upper() in self.left_dict:
74
                     keynote = self.left dict[event.text.upper()]
75
                     self.note_with_time.append((keynote, time.time()))
76
                     if self.left dict[event.text.upper()][1] == '#':
77
78
                         index = self.black labels.index(keynote)
79
80
                         self.black_sounds[index].play(0, 1000)
81
                     else:
                         index = self.white notes.index(keynote)
82
```

```
83
                          self.white sounds[index].play(0, 1000)
 84
                      # self.score.append(note.Note(keynote, type='quarter'))
 85
                 if event.text.upper() in self.right dict:
 86
                      keynote = self.right dict[event.text.upper()]
 87
                      self.note with time.append((keynote, time.time()))
 88
 89
 90
                      if self.right dict[event.text.upper()][1] == '#':
 91
                          index = self.black labels.index(keynote)
                          self.black sounds[index].play(0, 1000)
 92
 93
                      else:
 94
                          index = self.white_notes.index(keynote)
                          self.white_sounds[index].play(0, 1000)
 95
 96
                      # self.score.append(note.Note(keynote, type='quarter'))
 97
 98
             if event.type == pygame.KEYDOWN:
 99
                 if event.key == pygame.K_RETURN or event.key == pygame.K_KP_ENTER:
100
                      print('enter:stop')
                      self.save recording()
101
102
                      return
103
         def record(self):
104
             i = 0
105
             while self.recording:
106
107
                 for event in pygame.event.get():
108
                      if event.type == pygame.QUIT:
109
                          self.clean()
110
                          return
111
                      if event.type == pygame.TEXTINPUT:
112
                          if event.text.upper() in self.left dict:
                              keynote = self.left dict[event.text.upper()]
113
114
115
                              if self.left_dict[event.text.upper()][1] == '#':
116
                                  index = self.black labels.index(keynote)
117
118
                                  self.black sounds[index].play(0, 1000)
119
                              else:
120
                                  index = self.white notes.index(keynote)
121
                                  self.white sounds[index].play(0, 1000)
122
123
                              self.note with time.append((keynote, time.time()))
124
                              self.score.append(note.Note(keynote, type='quarter'))
125
                          if event.text.upper() in self.right_dict:
126
127
                              keynote = self.right dict[event.text.upper()]
128
```

```
129
                              if self.right dict[event.text.upper()][1] == '#':
                                  index = self.black labels.index(keynote)
130
                                  self.black_sounds[index].play(0, 1000)
131
132
                              else:
133
                                  index = self.white notes.index(keynote)
                                  self.white sounds[index].play(0, 1000)
134
135
136
                              self.note_with_time.append((keynote, time.time()))
137
                              # self.score.append(note.Note(keynote, type='quarter'))
                     if event.type == pygame.KEYDOWN:
138
139
                          if event.key == pygame.K RETURN or event.key ==
     pygame.K_KP_ENTER:
140
                              print('enter:stop')
141
                              self.save recording()
142
                              return
143
144
         def save recording(self):
             eighth timestep = 1 / (self.bpm * 2 / 60)
145
             for i in range(len(self.note with time) - 1):
146
147
                 curr_note, curr_time = self.note_with_time[i]
                 _, next_time = self.note_with_time[i+1]
148
149
                 n time steps = round((next time - curr time) / eighth timestep)
150
151
                 self.score.append(
                     note.Note(curr_note, quarterLength=n_time_steps/2))
152
153
154
             self.score.append(
155
                 note.Note(self.note with time[-1][0], quarterLength=2))
156
157
             print('stop record')
             self.recording = False
158
             self.score.write('midi', fp=self.save path)
159
             print(f"Recording stopped. MIDI file saved as {self.save_path}")
160
161
             pygame.mixer.quit()
162
163
164
     if __name__ == "__main__":
165
         recorder = AudioRecorder(save path='test recorder.mid')
         print('Hit Enter to Start Recording')
166
167
         input()
168
         recorder.start recording()
169
         print('Hit Enter to stop recording')
170
         input()
171
         recorder.stop_recording()
172
```

Keyboard_Pi_Conductor/PythonPiano/piano_lists.py

```
left hand = ['Z', 'S', 'X', 'D', 'C', 'V', 'G', 'B', 'H', 'N', 'J', 'M']
1
 2
    right hand = ['R', '5', 'T', '6', 'Y', 'U', '8', 'I', '9', 'O', '0', 'P']
 3
    piano notes = ['A0', 'A0#', 'B0', 'C1', 'C1#', 'D1', 'D1#', 'E1', 'F1", 'F1#', 'G1',
 4
    'G1#',
                    'A1', 'A1#', 'B1', 'C2', 'C2#', 'D2', 'D2#', 'E2', 'F2', 'F2#', 'G2',
    'G2#',
                   'A2', 'A2#', 'B2', 'C3', 'C3#', 'D3', 'D3#', 'E3', 'F3', 'F3#', 'G3',
 6
    'G3#',
7
                    'A3', 'A3#', 'B3', 'C4', 'C4#', 'D4', 'D4#', 'E4', 'F4', 'F4#', 'G4',
    'G4#',
                    'A4', 'A4#', 'B4', 'C5', 'C5#', 'D5', 'D5#', 'E5', 'F5', 'F5#', 'G5',
 8
    'G5#',
9
                    'A5', 'A5#', 'B5', 'C6', 'C6#', 'D6', 'D6#', 'E6', 'F6', 'F6#', 'G6',
    'G6#',
                    'A6', 'A6#', 'B6', 'C7', 'C7#', 'D7', 'D7#', 'E7', 'F7', 'F7#', 'G7',
10
    'G7#',
                    'A7', 'A7#', 'B7', 'C8']
11
12
    white notes = ['A0', 'B0', 'C1', 'D1', 'E1', 'F1', 'G1',
13
                    'A1', 'B1', 'C2', 'D2', 'E2', 'F2', 'G2',
14
                    'A2', 'B2', 'C3', 'D3', 'E3', 'F3', 'G3',
15
                    'A3', 'B3', 'C4', 'D4', 'E4', 'F4', 'G4',
16
                    'A4', 'B4', 'C5', 'D5', 'E5', 'F5', 'G5',
17
                    'A5', 'B5', 'C6', 'D6', 'E6', 'F6', 'G6',
18
                    'A6', 'B6', 'C7', 'D7', 'E7', 'F7', 'G7',
19
20
                    'A7', 'B7', 'C8']
21
22
    black notes = ['Bb0', 'Db1', 'Eb1', 'Gb1', 'Ab1',
23
                    'Bb1', 'Db2', 'Eb2', 'Gb2', 'Ab2',
24
                    'Bb2', 'Db3', 'Eb3', 'Gb3', 'Ab3',
                    'Bb3', 'Db4', 'Eb4', 'Gb4', 'Ab4',
25
26
                    'Bb4', 'Db5', 'Eb5', 'Gb5', 'Ab5',
27
                    'Bb5', 'Db6', 'Eb6', 'Gb6', 'Ab6',
28
                    'Bb6', 'Db7', 'Eb7', 'Gb7', 'Ab7',
29
                    'Bb7']
30
    black labels = ['A#0', 'C#1', 'D#1', 'F#1', 'G#1',
31
                     'A#1', 'C#2', 'D#2', 'F#2', 'G#2',
32
                     'A#2', 'C#3', 'D#3', 'F#3', 'G#3',
33
                     'A#3', 'C#4', 'D#4', 'F#4', 'G#4',
34
                     'A#4', 'C#5', 'D#5', 'F#5', 'G#5',
35
                     'A#5', 'C#6', 'D#6', 'F#6', 'G#6',
36
```

```
37 'A#6', 'C#7', 'D#7', 'F#7', 'G#7',
38 'A#7']
```

Keyboard_Pi_Conductor/Audio_Midi/init.py

Keyboard_Pi_Conductor/Audio_Midi/audio_to_midi/integrate_main.py

```
import os
 1
 2
    import sys
    import logging
 3
 4
 5
    from Audio Midi.audio to midi import converter, progress bar
 6
 7
 8
    def _convert_beat_to_time(bpm, beat):
9
        try:
10
            parts = beat.split("/")
11
            if len(parts) > 2:
                 raise Exception()
12
13
14
            beat = [int(part) for part in parts]
            fraction = beat[0] / beat[1]
15
            bps = bpm / 60
16
17
            ms per beat = bps * 1000
            return fraction * ms_per_beat
18
        except Exception:
19
            raise RuntimeError("Invalid beat format: {}".format(beat))
20
21
22
    def audio_to_midi_conv(infile, outfile, beat='1/4', single_note=True):
23
24
        try:
25
            logging.basicConfig(level=logging.DEBUG, format="%(message)s")
26
27
            time_window = _convert_beat_to_time(bpm, beat)
28
29
            global bpm
30
            process = converter.Converter(
31
                 infile=infile,
                 outfile=outfile,
32
33
                 time_window=time_window,
                 activation_level=0.0,
34
35
                 condense=False,
                 condense max=False,
36
```

```
37
                 max note length=0,
                 note count=1 if single note else 0,
38
39
                 transpose=0,
40
                 pitch_set=[],
41
                 pitch_range=None,
                 progress=progress_bar.ProgressBar(),
42
                 bpm=bpm,
43
44
             )
45
             process.convert()
        except KeyboardInterrupt:
46
             sys.exit(1)
47
        except Exception as e:
48
49
             logging.exception(e)
50
             sys.exit(1)
51
```

Keyboard_Pi_Conductor/Audio_Midi/audio_to_midi/midi_writer.py

```
from collections import defaultdict
 1
 2
    import python3 midi as midi
 4
 5
    class NoteState:
 6
 7
        slots = ["is active", "event pos", "count"]
 8
 9
        def __init__(self, is_active=False, event_pos=None, count=0):
            self.is active = is active
10
            self.event pos = event pos
11
            self.count = count
12
13
14
15
    class MidiWriter:
        def __init__(
16
            self,
17
18
            outfile,
19
            channels,
            time_window,
20
21
            bpm=60,
            condense=False,
22
23
            condense max=False,
            max_note_length=0,
24
25
        ):
            self.outfile = outfile
26
27
            self.condense = condense
```

```
28
             self.condense max = condense max
             self.max note length = max note length
29
             self.channels = channels
30
             self.time window = time window
31
32
             self.bpm = bpm
             self.note state = [defaultdict(lambda: NoteState()) for in
33
    range(channels)]
34
35
            bps = self.bpm / 60
             self.ms per beat = int((1.0 / bps) * 1000)
36
            self.tick_increment = int(time_window)
37
            self.skip_count = 1
38
39
            self._need_increment = False
40
        def enter (self):
41
42
             self.stream = midi.FileStream(self.outfile)
43
            self.stream.start_pattern(
44
                 format=1,
                 tick_relative=False,
45
                 resolution=self.ms_per_beat,
46
                 tracks=[],
47
48
             )
            self.stream.start track(
49
50
                 events=[
                     midi.TimeSignatureEvent(
51
52
                         tick=0,
53
                         numerator=1,
54
                         denominator=4,
                         metronome=int(self.ms_per_beat / self.time_window),
55
56
                         thirtyseconds=32,
57
                     )
58
                 1,
                 tick_relative=False,
59
60
            return self
61
62
63
        def __exit__(self, type, value, traceback):
            self. terminate notes()
64
            self.stream.add event(midi.EndOfTrackEvent(tick=1))
65
            self.stream.end track()
66
            self.stream.end pattern()
67
            self.stream.close()
68
69
        def _skip(self):
70
71
            self.skip count += 1
72
```

```
73
         def reset skip(self):
              self.skip count = 1
 74
 75
 76
         @property
         def tick(self):
 77
             ret = 0
 78
             if self._need_increment:
 79
                  self._need_increment = False
 80
 81
                  ret = self.tick_increment * self.skip_count
                  self. reset skip()
 82
             return ret
 83
 84
 85
         def _note_on(self, channel, pitch, velocity):
 86
             pos = self.stream.add event(
                 midi.NoteOnEvent(
 87
                      tick=self.tick, channel=channel, pitch=pitch, velocity=60
 88
 89
                  )
 90
              self.note state[channel][pitch] = NoteState(
 91
 92
                  True,
 93
                  pos,
 94
                  1,
 95
              )
 96
 97
         def _note_off(self, channel, pitch):
              self.note state[channel][pitch] = NoteState()
 98
              self.stream.add event(
 99
                  midi.NoteOffEvent(
100
                      tick=self.tick,
101
102
                      channel=channel,
                      pitch=pitch,
103
104
                  )
105
              )
106
107
         def add notes(self, notes):
108
109
              notes is a list of midi notes to add at the current
110
                  time step.
111
112
             Adds each note in the list to the current time step
113
                  with the volume, track and channel specified.
114
115
              self. need increment = True
              if not self.condense:
116
117
                  self. terminate notes()
118
```

```
119
             for channel, notes in enumerate(notes):
120
                  new notes = set()
                  stale_notes = []
121
                  for note in notes:
122
123
                      note state = self.note state[channel][note.pitch]
                      new notes.add(note.pitch)
124
125
                      if (not self.condense) or (self.condense and not
     note state.is active):
126
                          self._note_on(channel, note.pitch, note.velocity)
                      elif self.condense and note state.is active:
127
                          event = self.stream.get event(
128
                              midi.NoteOnEvent, note_state.event_pos
129
130
                          )
131
                          old velocity = event.data[1]
132
                          if self.condense max:
133
                              new_velocity = max(note.velocity, old_velocity)
134
                          else:
                              count = note state.count
135
                              note state.count += 1
136
137
                              new_velocity = ((old_velocity * count) + note.velocity) // (
138
                                  note_state.count
139
                          if old velocity != event.data[1]:
140
141
                              event.data[1] = new velocity
142
                              self.stream.set_event(event, note_state.event_pos)
143
                  if self.condense:
144
                      active notes = [
145
146
                          note
147
                          for note in self.note state[channel]
                          if self.note state[channel][note].is active
148
149
150
                      for note in active_notes:
151
                          if (
152
                              note not in new notes
153
                              or self.note_state[channel][note].count >
     self.max_note_length
154
                          ):
155
                              stale_notes.append(note)
156
157
                      for note in stale notes:
158
                          self. note off(channel, note)
159
             if self._need_increment:
160
161
                  self. skip()
162
```

```
def _terminate_notes(self):

for channel in range(self.channels):

for note, note_state in self.note_state[channel].items():

if note_state.is_active:

self._note_off(channel, note)
```

Keyboard_Pi_Conductor/Audio_Midi/audio_to_midi/converter.py

```
import logging
 1
 2
    from collections import namedtuple
    from functools import lru cache
 5
    from operator import attrgetter
 6
 7
    import numpy
 8
    import soundfile
 9
10
    # import midi_writer, notes
    from Audio Midi.audio to midi import midi writer, notes
11
12
13
14
    class Note:
        __slots__ = ["pitch", "velocity", "count"]
15
16
        def __init__(self, pitch, velocity, count=0):
17
18
            self.pitch = pitch
            self.velocity = velocity
19
            self.count = count
20
21
22
    class Converter:
23
        def __init__(
24
            self,
25
            infile=None,
26
27
            outfile=None,
28
            time window=None,
            activation_level=None,
29
30
            condense=None,
            condense_max=False,
31
32
            max_note_length=0,
            transpose=0,
33
            pitch_set=None,
34
            pitch_range=None,
35
            note_count=None,
36
```

```
37
            progress=None,
38
            bpm=60,
39
        ):
40
41
            if infile:
                 self.info = soundfile.info(infile)
42
            else:
43
                 raise RuntimeError("No input provided.")
44
45
            self.infile = infile
46
            self.outfile = outfile
47
            self.time_window = time window
48
49
            self.condense = condense
50
            self.condense max = condense max
            self.max note_length = max_note_length
51
52
            self.transpose = transpose
53
            self.pitch_set = pitch_set
            self.pitch range = pitch range or [0, 127]
54
            self.note count = note count
55
56
            self.progress = progress
            self.bpm = bpm
57
58
            self.activation level = int(127 * activation level) or 1
59
            self.block size = self. time window to block size(
60
                 self.time_window, self.info.samplerate
61
62
63
            steps = self.info.frames // self.block_size
64
            self.total = steps
65
            self.current = 0
66
67
68
            self. determine ranges()
69
70
        def determine ranges(self):
71
            self.notes = notes.generate()
72
            self.max_freq = min(self.notes[127][-1], self.info.samplerate / 2)
73
            self.min_freq = max(self.notes[0][-1], 1000 / self.time_window)
74
            self.bins = self.block size // 2
            self.frequencies = numpy.fft.fftfreq(self.bins, 1 / self.info.samplerate)[
75
                 : self.bins // 2
76
77
            ]
78
79
            for i, f in enumerate(self.frequencies):
                 if f >= self.min freq:
80
                     self.min bin = i
81
                     break
82
```

```
83
             else:
                  self.min bin = 0
 84
             for i, f in enumerate(self.frequencies):
 85
                  if f >= self.max freq:
 86
 87
                      self.max bin = i
                      break
 88
             else:
 89
 90
                  self.max_bin = len(self.frequencies)
 91
         def increment progress(self):
 92
             if self.progress:
 93
                  self.current += 1
 94
 95
                  self.progress.update(self.current, self.total)
 96
         @staticmethod
 97
 98
         def _time_window_to_block_size(time_window, rate):
 99
             time window is the time in ms over which to compute fft's.
100
             rate is the audio sampling rate in samples/sec.
101
102
             Transforms the time window into an index step size and
103
104
                  returns the result.
              ......
105
106
             # rate/1000(samples/ms) * time_window(ms) = block_size(samples)
107
             rate per ms = rate / 1000
108
109
             block size = rate per ms * time window
110
111
             return int(block_size)
112
         def _freqs_to_midi(self, freqs):
113
114
             freq_list is a list of frequencies with normalized amplitudes.
115
116
117
             Takes a list of notes and transforms the amplitude to a
118
                 midi volume as well as adding track and channel info.
              .....
119
120
             notes = [None for in range(128)]
121
             for pitch, velocity in freqs:
122
123
                  if not (self.pitch range[0] <= pitch <= self.pitch range[1]):</pre>
124
                      continue
125
                  velocity = min(int(127 * (velocity / self.bins)), 127)
126
127
                  if velocity > self.activation level:
                      if not notes[pitch]:
128
```

```
129
                          notes[pitch] = Note(pitch, 60)
130
                      else:
                          notes[pitch].velocity = int(
131
                              ((notes[pitch].velocity * notes[pitch].count) + velocity)
132
133
                              / (notes[pitch].count + 1)
134
                          )
135
                          notes[pitch].count += 1
136
137
             notes = [note for note in notes if note]
138
             if self.note count > 0:
139
140
                 max_count = min(len(notes), self.note_count)
                  notes = sorted(notes, key=attrgetter("velocity"))[::-1][:max_count]
141
142
             return notes
143
144
145
         def _snap_to_key(self, pitch):
             if self.pitch set:
146
                 mod = pitch % 12
147
148
                 pitch = (12 * (pitch // 12)) + min(
                      self.pitch_set, key=lambda x: abs(x - mod)
149
150
                  )
151
             return pitch
152
         @lru_cache(None)
153
         def freq to pitch(self, freq):
154
             for pitch, freq range in self.notes.items():
155
                  # Find the freq's equivalence class, adding the amplitudes.
156
                  if freq_range[0] <= freq <= freq_range[2]:</pre>
157
158
                      return self._snap_to_key(pitch) + self.transpose
             raise RuntimeError("Unmappable frequency: {}".format(freq[0]))
159
160
161
         def _reduce_freqs(self, freqs):
              11 11 11
162
             freqs is a list of amplitudes produced by fft to frequencies().
163
164
165
             Reduces the list of frequencies to a list of notes and their
                  respective volumes by determining what note each frequency
166
                  is closest to. It then reduces the list of amplitudes for each
167
                  note to a single amplitude by summing them together.
168
              .. .. ..
169
170
171
             reduced freqs = []
172
             for freq in freqs:
                  reduced freqs.append((self. freq to pitch(freq[0]), freq[1]))
173
174
```

```
175
             return reduced freqs
176
         def _samples_to_freqs(self, samples):
177
              amplitudes = numpy.fft.fft(samples)
178
179
             freqs = []
180
              for index in range(self.min_bin, self.max_bin):
181
                  # frequency, amplitude
182
183
                  freqs.append(
184
                          self.frequencies[index],
185
186
                          numpy.sqrt(
187
                              numpy.float_power(amplitudes[index].real, 2)
188
                              + numpy.float power(amplitudes[index].imag, 2)
189
                          ),
190
                      ]
191
                  )
192
             # Transform the frequency info into midi compatible data.
193
194
             return self._reduce_freqs(freqs)
195
         def block to notes(self, block):
196
             channels = [[] for in range(self.info.channels)]
197
             notes = [None for _ in range(self.info.channels)]
198
199
200
             for sample in block:
                  for channel in range(self.info.channels):
201
202
                      channels[channel].append(sample[channel])
203
204
             for channel, samples in enumerate(channels):
                  freqs = self. samples to freqs(samples)
205
                  notes[channel] = self. freqs to midi(freqs)
206
207
208
             return notes
209
210
         def convert(self):
211
212
             Performs the fft for each time step and transforms the result
                  into midi compatible data. This data is then passed to a
213
                  midi file writer.
214
              .. .. ..
215
216
217
              logging.info(str(self.info))
             logging.info("window: {} ms".format(self.time_window))
218
              logging.info(
2.19
```

```
220
                  "frequencies: min = {} Hz, max = {} Hz".format(self.min freq,
     self.max freq)
221
             )
222
             with midi writer.MidiWriter(
223
                  outfile=self.outfile,
224
                  channels=self.info.channels,
225
                  time window=self.time window,
226
227
                  bpm=self.bpm,
                  condense=self.condense,
228
                  condense max=self.condense max,
2.2.9
                  max_note_length=self.max_note_length,
230
231
              ) as writer:
232
                  for block in soundfile.blocks(
233
                      self.infile,
234
                      blocksize=self.block size,
235
                      always_2d=True,
236
                  ):
                      if len(block) != self.block size:
237
238
                          filler = numpy.array(
239
                              [
                                  numpy.array([0.0 for _ in range(self.info.channels)])
240
                                  for in range(self.block size - len(block))
241
242
                              1
243
                          block = numpy.append(block, filler, axis=0)
244
                      notes = self. block to notes(block)
245
                      writer.add notes(notes)
246
                      self._increment_progress()
247
248
```

Keyboard_Pi_Conductor/Audio_Midi/audio_to_midi/init.py

```
1 |
```

Keyboard_Pi_Conductor/Audio_Midi/audio_to_midi/notes.py

```
import numpy

def generate():

"""

Generates a dict of midi note codes with their corresponding
    frequency ranges.

"""
```

```
9
        # C0
        base = [7.946362749, 8.1757989155, 8.4188780665]
10
11
        # 12th root of 2
12
        multiplier = numpy.float power(2.0, 1.0 / 12)
13
14
15
        notes = {0: base}
        for i in range(1, 128):
16
17
            mid = multiplier * notes[i - 1][1]
            low = (mid + notes[i - 1][1]) / 2.0
18
            high = (mid + (multiplier * mid)) / 2.0
19
            notes.update({i: [low, mid, high]})
20
21
22
        return notes
23
```

Keyboard_Pi_Conductor/Audio_Midi/audio_to_midi/progress_bar.py

```
1
    import time
 2
    import threading
    import progressbar
 3
 4
 5
    class ProgressBar:
 6
7
        def init (self, current=0, total=0):
            self.current = current
 8
 9
            self.total = total
            self.bar = progressbar.ProgressBar(max value=self.total)
10
11
        def update(self, current=0, total=0):
12
13
            current = min(current, total)
14
            self.bar.max value = total
            self.bar.update(current)
15
16
```

Keyboard_Pi_Conductor/Audio_Midi/audio_to_midi/main.py

```
#!/usr/bin/env python3

import argparse
import os
import sys
import logging

import converter, progress_bar
```

```
9
10
11
    def _convert_beat_to_time(bpm, beat):
12
        try:
13
            parts = beat.split("/")
            if len(parts) > 2:
14
                 raise Exception()
15
16
17
            beat = [int(part) for part in parts]
            fraction = beat[0] / beat[1]
18
            bps = bpm / 60
19
20
            ms_per_beat = bps * 1000
            return fraction * ms_per_beat
21
22
        except Exception:
            raise RuntimeError("Invalid beat format: {}".format(beat))
23
24
25
    def parse args():
26
27
        parser = argparse.ArgumentParser()
28
        parser.add_argument("infile", help="The sound file to process.")
        parser.add_argument(
29
             "--output", "-o", help="The MIDI file to output. Default: <infile>.mid"
30
31
        parser.add argument(
32
             "--time-window",
33
             "-t",
34
            default=5.0,
35
36
            type=float,
            help="The time span over which to compute the individual FFTs in
37
    milliseconds.",
39
        parser.add argument(
             "--activation-level",
40
             "-a",
41
            default=0.0,
42
43
            type=float,
44
            help="The amplitude threshold for notes to be added to the MIDI file. Must
    be between 0 and 1.",
45
        parser.add_argument(
46
             "--condense",
47
             "-c",
48
49
             action="store_true",
             help="Combine contiguous notes at their average amplitude.",
50
51
        )
52
        parser.add_argument(
```

```
53
             "--condense-max",
             "-m",
54
55
             action="store_true",
56
            help="Write the maximum velocity for a condensed note segment rather than
    the rolling average.",
57
        parser.add_argument(
58
             "--max-note-length",
59
             "-M",
60
61
            type=int,
            default=0,
62
            help="The max condensed note length in time window units.",
6.3
64
        )
65
        parser.add argument(
             "--single-note",
66
             "-s",
67
68
            action="store_true",
            help="Only add the loudest note to the MIDI file for a given time window.",
69
70
71
        parser.add_argument(
72
             "--note-count",
             "-C",
73
            type=int,
74
75
            default=0,
            help="Only add the loudest n notes to the MIDI file for a given time
76
    window.",
77
        )
78
        parser.add_argument(
             "--bpm", "-b", type=int, help="Beats per minute. Defaults: 60", default=60
79
80
81
        parser.add argument(
             "--beat",
82
             "-B",
83
84
            help="Time window in terms of beats (1/4, 1/8, etc.). Supercedes the time
    window parameter.",
85
86
        parser.add_argument(
             "--transpose",
87
             "-T",
88
            type=int,
89
            default=0,
90
91
            help="Transpose the MIDI pitches by a constant offset.",
92
        parser.add_argument(
93
             "--pitch-set",
94
             "-p",
95
```

```
96
             type=int,
 97
             nargs="+",
 98
             default=[],
             help="Map to a pitch set. Values must be in the range: [0, 11]. Ex: -p 0 2 4
 99
     5 7 9 11",
100
         parser.add_argument(
101
             "--pitch-range",
102
             "-P",
103
104
             nargs=2,
105
             type=int,
             help="The minimum and maximum allowed MIDI notes. These may be superseded by
106
     the calculated FFT range.",
107
108
         parser.add argument(
             "--no-progress", "-n", action="store_true", help="Don't print the progress
109
     bar."
110
111
         args = parser.parse args()
112
113
         args.output = (
             "{}.mid".format(os.path.basename(args.infile))
114
115
             if not args.output
116
             else args.output
117
         )
118
119
         if args.single note:
120
             args.note count = 1
121
122
         if args.pitch set:
             for key in args.pitch set:
123
124
                  if key not in range(12):
125
                      raise RuntimeError("Key values must be in the range: [0, 12)")
126
127
         if args.beat:
             args.time_window = _convert_beat_to_time(args.bpm, args.beat)
128
129
             print(args.time_window)
130
         if args.pitch range:
131
132
             if args.pitch_range[0] > args.pitch_range[1]:
                  raise RuntimeError("Invalid pitch range: {}".format(args.pitch range))
133
134
135
         if args.condense max:
136
             args.condense = True
137
138
         return args
```

```
139
140
     def main():
141
142
         try:
              logging.basicConfig(level=logging.DEBUG, format="%(message)s")
143
144
145
             args = parse_args()
146
147
             process = converter.Converter(
148
                  infile=args.infile,
                  outfile=args.output,
149
                  time_window=args.time_window,
150
                  activation level=args.activation level,
151
                  condense=args.condense,
152
153
                  condense max=args.condense max,
                  max_note_length=args.max_note_length,
154
155
                  note count=args.note count,
156
                  transpose=args.transpose,
157
                  pitch set=args.pitch set,
158
                  pitch_range=args.pitch_range,
159
                  progress=None if args.no_progress else progress_bar.ProgressBar(),
                  bpm=args.bpm,
160
161
              )
162
             process.convert()
163
         except KeyboardInterrupt:
164
              sys.exit(1)
         except Exception as e:
165
             logging.exception(e)
166
167
             sys.exit(1)
168
169
     if __name__ == "__main__":
170
171
         main()
172
```

Keyboard_Pi_Conductor/Audio_Midi/python_midi.py

```
# from midiutil.MidiFile import
import pyaudio
import wave

from Audio_Midi.audio_to_midi.converter import Converter
from threading import Thread

class AudioRecorder:
```

```
def init (self, save path):
10
            self.recording = False
11
            self.audio = pyaudio.PyAudio()
12
13
14
            self.usb_idx = 0
15
16
17
            # Outputs index of every audio-capable device on the Pi
            for ii in range(self.audio.get device count()):
18
                if "USB PnP Sound" in
19
    self.audio.get_device_info_by_index(ii).get('name'):
20
                     self.usb_idx = ii
21
                print(f"idx: {ii}, name:
    {self.audio.get device info by index(ii).get('name')}")
2.2
23
            # Set index of usb microphone
24
            # Set parameters for audio recording
25
            self.form_1 = pyaudio.paInt16 # 16-bit resolution
2.6
            self.chans = 1 # 1 channel
27
            self.samp rate = 44100 # 44.1kHz sampling rate
28
            self.chunk = 4096 # 2^12 samples for buffer
29
            self.save path = save path # name of .wav file
30
            self.frames = []
31
32
33
34
        def start recording(self):
            self.stream = self.audio.open(
35
36
                format=self.form 1,
37
                rate=self.samp rate,
                channels=self.chans,
38
                input_device_index=self.usb_idx,
39
40
                input=True,
                frames per buffer=self.chunk
41
42
43
            print("recording")
44
            self.recording = True
45
            self.record thread = Thread(target=self.record)
46
            self.record thread.start()
47
48
49
        def record(self):
            while self.recording:
50
51
                data = self.stream.read(self.chunk)
52
                self.frames.append(data)
```

```
53
54
        def stop recording(self):
            self.recording = False
55
56
57
            self.record thread.join()
            print('Recording Stopped, closing streams')
58
59
            self.stream.stop_stream()
60
61
            self.stream.close()
            self.audio.terminate()
62
63
            print("finished recording")
64
65
66
            # save the audio frames as .wav file
            wavefile = wave.open(self.save path, 'wb')
67
            wavefile.setnchannels(self.chans)
68
69
            wavefile.setsampwidth(self.audio.get_sample_size(self.form_1))
            wavefile.setframerate(self.samp rate)
70
            wavefile.writeframes(b''.join(self.frames))
71
72
            wavefile.close()
73
74
    def start recording(audio, form 1, chans, samp rate, chunk, usb idx):
75
        # create pyaudio stream
76
        stream = audio.open(format=form_1, rate=samp_rate, channels=chans,
77
                             input device index=usb idx, input=True,
78
                             frames per buffer=chunk)
79
        print("recording")
80
81
        return stream
82
83
84
    def record loop(stream, samp rate, chunk, record secs):
        # loop through stream and append audio chunks to frame array
85
86
        frames = []
        for ii in range(int((samp rate/chunk)*record secs)):
87
            data = stream.read(chunk)
88
89
            frames.append(data)
90
        return frames
91
92
    def stop recording(audio, stream, filename, chans, samp rate, frames, form 1):
93
        # stop the stream, close it, and terminate the pyaudio instantiation
94
95
        stream.stop stream()
        stream.close()
96
97
        audio.terminate()
98
```

```
99
         print("finished recording")
100
101
         # save the audio frames as .wav file
         wavefile = wave.open(filename, 'wb')
102
         wavefile.setnchannels(chans)
103
         wavefile.setsampwidth(audio.get sample size(form 1))
104
105
         wavefile.setframerate(samp_rate)
         wavefile.writeframes(b''.join(frames))
106
107
         wavefile.close()
108
109
     def audio to midi conv(input filename, output filename, time window=5.0,
110
     activ level=0.0, note count=0, bpm=60):
111
         process = Converter(
112
             infile=input filename,
113
             outfile=output filename,
114
             time window=time window,
115
             activation level=activ level,
             condense=None,
116
117
             condense_max=False,
118
             max_note_length=0,
             note count=note count,
119
             bpm=bpm,
120
121
         )
122
         process.convert()
123
124
     def record audio(audio length, save path):
125
         audio = pyaudio.PyAudio()
126
127
         # Outputs index of every audio-capable device on the Pi
128
129
         for ii in range(audio.get device count()):
130
             print(f"idx: {ii}, name: {audio.get_device_info_by_index(ii).get('name')}")
131
132
         # Set index of usb microphone
133
         usb idx = 1
134
135
         # Set parameters for audio recording
         form 1 = pyaudio.paInt16 # 16-bit resolution
136
         chans = 1 # 1 channel
137
         samp rate = 44100 # 44.1kHz sampling rate
138
         chunk = 4096  # 2^12 samples for buffer
139
         wav output filename = save path # name of .wav file
140
141
         frames = []
142
143
         stream = start recording(audio, form 1, chans, samp rate, chunk, usb idx)
```

```
144
         frames = record loop(stream, samp rate, chunk, audio length)
         stop recording(audio, stream, wav output filename,
145
                        chans, samp_rate, frames, form_1)
146
147
148
     if name == " main ":
149
150
         # record_audio(10, 'test_recorder.wav')
         recorder = AudioRecorder(save path='test recorder.wav')
151
152
         print('Hit Enter to Start Recording')
153
154
         input()
155
         recorder.start_recording()
156
157
         print('Hit Enter to Stop Recording')
158
         input()
159
         recorder.stop_recording()
160
```

Keyboard_Pi_Conductor/metadata.py

```
1
 2
   RAW_AUDIO_PATH = 'out/recorded_audio.wav'
   RECORDED_VIDEO_PATH = 'out/conducting_video.mp4'
 3
   RAW MIDI PATH = 'out/raw midi.mid'
 4
   MODIFIED MIDI PATH = 'out/modified midi.mid'
 5
   MODIFIED_AUDIO_PATH = 'out/modified_audio.wav'
 6
 7
   MIDI_BEAT = '1/4'
   MIDI BPM = 90
8
9
10
11
```

Keyboard_Pi_Conductor/integrate.py

```
import pygame
from pygame.time import Clock
import threading
from PythonPiano.key_midi import AudioRecorder
from Gesture_IR.record_video import VideoRecorder
from Gesture_IR.gesture_ir import gesture_ir_main
from IR_Midi.modify_input import modify_volume, convert_midi_to_audio,
play_audio_file

from GUI import init_system, GUI
from metadata import *
```

```
11
    import time
12
    from metronome import Metronome
13
14
15
    current frame = None
16
    class DataFrame:
17
        def __init__(self):
18
19
            self.audio_recorder = AudioRecorder(save_path=RAW_MIDI_PATH)
            self.bpm = MIDI BPM
20
             self.video recorder = VideoRecorder(RECORDED VIDEO PATH, bpm=self.bpm)
21
22
23
    df = DataFrame()
24
25
    def start audio playback recording and camera(frame, audio file, video path):
        df.video_recorder.initialize(df.bpm*2)
2.6
27
        df.video_recorder.spawn_camera()
        time.sleep(2.0)
28
29
30
        playback_thread = threading.Thread(target=play_audio_file, args=(audio_file,))
31
        frame.idle components = []
32
        frame.add text(
33
            text="START CONDUCTING TO MUSIC",
34
            center_frac_x=0.5,
35
            center frac y=0.25
36
37
        frame.event components = []
38
39
        frame.reset()
40
        frame.render()
41
42
        playback thread.start()
43
        df.video_recorder.start_recording()
44
        playback thread.join()
45
        df.video_recorder.stop_recording()
46
47
48
        frame.add text(
            text="MUSIC FINISHED",
49
            center_frac_x=0.5,
50
            center frac y=0.5
51
52
        )
53
        frame.event_components = []
        frame.reset()
54
55
        frame.render()
56
```

```
57
     def start production(display message):
58
         # Replace with the path to your audio file
59
         audio path = 'testrun1.wav'
60
         video path = 'testrun1 conduct.mp4'
61
         midi path = "mid.midi"
62
         out_midi_path = 'testrun1_final_midi.mid'
63
         out_audio_path = 'testrun1_final_audio.wav'
64
65
         df.bpm = 90
         beat = '1/4'
66
         single note = True
67
68
69
70
         # print("Hit enter to start recording audio: ")
71
         # input()
         # record_audio(20, save_path=audio_path)
72
73
74
         # print("Hit ENTER to convert the audio to midi file")
         # input()
75
76
         # audio to midi conv(audio path, midi path, beat=beat, df.bpm=df.bpm)
77
78
         exit()
79
         print("Hit ENTER to record conducting video")
80
81
         input()
         start audio playback recording and camera(
82
             audio path, video path
83
         )
84
85
         display message("Start production")
86
87
88
         print('Hit ENTER to convert gesture video to IR')
89
         input()
90
         # Convert gesture to IR
91
         ir data = gesture ir main(video path)
92
93
         # Modify MIDI file based on IR and convert to audio
94
         print('Hit ENTER to convert video IR to dynamics: ')
95
         input()
         volume changes = ir data
96
97
         modify volume(midi path, volume changes, out midi path, bpm=df.bpm)
98
99
         print('Hit ENTER to convert modified midi file to audio file')
100
         input()
101
         convert midi to audio(out midi path, out audio path)
102
```

```
103
         display message("Production finished, let's see the result")
104
105
     def show result():
106
         # Visualize MIDI files and play back the modified audio
107
         original midi = "converted midi.mid"
108
         modified_midi = "output_file.mid"
109
         # visualize notes(original midi, modified midi)
110
111
         play audio file("output audio.wav")
112
113
114
115
     def display_message(message):
116
         global current message
117
         current message = message
118
119
     def switch_frame(switch_frame):
120
         print('Switching Frame')
         global current frame
121
122
         current frame = switch frame
123
     def start recording_audio(frame, next_frame):
124
         if frame.get text()[0] != '':
125
             df.bpm = int(frame.get text()[0])
126
127
         metronome.reset_fps(df.bpm/30)
         print('recording with df.bpm: ', df.bpm)
128
129
         df.audio recorder.initialize(bpm=df.bpm)
130
131
         switch_frame(next_frame)
132
     def stop recording audio(next frame):
133
         print('clicked stop recording audio')
134
         df.audio_recorder.save_recording()
135
136
         switch frame(next frame)
137
138
     def save recording continue cb(next frame):
139
         convert_midi_to_audio(RAW_MIDI_PATH, RAW_AUDIO_PATH)
140
         switch frame(next frame)
141
142
     def start_conducting_cb(frame, next_frame):
143
         frame.add text(
             text="START CONDUCTING",
144
145
             center frac x=0.5,
             center_frac_y=0.25
146
147
         )
148
```

```
149
         frame.reset()
150
         frame.render()
151
152
         start audio playback recording and camera(frame, RAW AUDIO PATH,
     RECORDED VIDEO PATH)
153
154
         frame.idle_components = []
155
         frame.add_text(
156
             text="RECORDING DONE",
157
             center frac x=0.5,
             center frac y=0.25
158
159
         )
         frame.reset()
160
161
         frame.render()
162
163
         time.sleep(1)
164
         frame.idle_components = []
165
         frame.add button(
             "Start Conducting",
166
167
             0.4, 0.0625, 0.3, 0.4,
168
             callback=lambda: start_conducting_cb(start_record_video_frame,
     combine frame)
169
         )
170
171
         switch_frame(next_frame)
172
     def generate result cb(frame, next frame):
173
         txt, button rect, cb = frame.event components[0]
174
175
         frame.event_components[0] = 'Converting...', button_rect, None
         frame.reset()
176
         frame.render()
177
178
179
         # Convert gesture to IR
180
         ir info, ir data = gesture ir main(RECORDED VIDEO PATH)
181
182
         # Modify MIDI file based on IR and convert to audio
183
         volume_changes = ir_data
184
         modify volume (RAW MIDI PATH, volume changes, MODIFIED MIDI PATH, bpm=df.bpm)
185
186
         convert_midi_to_audio(MODIFIED_MIDI_PATH, MODIFIED_AUDIO_PATH)
187
188
         frame.event_components[0] = txt, button_rect, cb
189
         frame.reset()
190
         frame.render()
191
192
         switch frame(next frame)
```

```
193
     def play result cb(frame, button idx):
194
195
         txt, button_rect, cb = frame.event_components[button_idx]
         frame.event components[button idx] = 'Playing Audio...', button rect, cb
196
         frame.reset()
197
         frame.render()
198
199
200
         play audio file(MODIFIED AUDIO PATH)
201
202
         frame.event components[button idx] = txt, button rect, cb
203
         frame.reset()
204
         frame.render()
205
206
     def start over cb():
         start record video frame = GUI(window width, window height)
207
         start record video frame.add button(
208
209
             "Start Conducting",
210
             0.4, 0.0625, 0.3, 0.4,
211
             callback=lambda: start conducting cb(start record video frame,
     combine frame)
212
         )
213
214
         switch frame(start frame)
215
216
     def re_conduct_cb():
         start record video frame = GUI(window width, window height)
217
         start record video frame.add button(
218
219
             "Start Conducting",
             0.4, 0.0625, 0.3, 0.4,
220
             callback=lambda: start conducting_cb(start_record_video_frame,
221
     combine frame)
222
         )
223
224
         switch frame(start record video frame)
225
226
227
     init_system()
228
229
     window width = 800
230
     window height = 600
231
     # Combine done, playback results:
232
     playback frame = GUI(window width, window height)
233
     play idx = playback frame.add button(
234
         "Play Result",
235
         0.5, 0.0625, 0.25, 0.4,
236
```

```
237
         callback=lambda: play result cb(playback frame, play idx)
238
239
     playback frame.add button(
240
         "Produce Another Audio",
241
         0.5, 0.0625, 0.25, 0.6,
242
         callback=lambda: start_over_cb()
243
244
     )
245
     playback frame.add button(
246
         "Re-Record Conducting Video",
247
         0.5, 0.0625, 0.25, 0.8,
248
249
         callback=lambda: re_conduct_cb()
250
251
     # recording stops, click button to combine
252
253
     combine_frame = GUI(window_width, window_height)
     combine frame.add button(
254
         "Combine Video Dynamics with Audio",
255
256
         0.7, 0.0625, 0.15, 0.4,
257
         callback=lambda: generate_result_cb(combine_frame, playback_frame)
258
259
260
     # frame to start recording video then convert to intermediate representation
261
     start record video frame = GUI(window width, window height)
262
263
     start record video frame.add button(
         "Start Conducting",
264
         0.4, 0.0625, 0.3, 0.4,
265
266
         callback=lambda: start conducting cb(start record video frame, combine frame)
267
     )
268
     # Giving option to re-record audio or proceed:
269
270
     re record audio frame = GUI(window width, window height)
271
     go conducting button = re record audio frame.add button(
272
         "Continue",
273
         0.4, 0.0625, 0.3, 0.4,
274
         callback=lambda: save recording continue cb(start record video frame)
275
     )
276
277
     re record button = re record audio frame.add button(
         "Re-Record Audio",
278
279
         0.4, 0.0625, 0.3, 0.6,
         callback=lambda: switch frame(record audio frame)
280
281
282
```

```
283
     # frame when audio recording is in progress and waiting to stop
     audio recording frame = GUI(window width, window height)
284
     record_audio_button = audio_recording_frame.add_button(
285
         "Stop Recording",
286
         0.4, 0.0625, 0.3, 0.8,
287
         callback=lambda: stop recording audio(re record audio frame)
288
289
     )
290
291
     audio recording frame.add text(
         text="Hit ENTER to stop and save recording",
292
293
         center frac x=0.5,
         center frac y = 0.25
294
295
296
     audio_recording_frame.add_circle(radius=50, center_frac_x=0.5, center_frac_y=0.5)
     metronome = Metronome(frame=audio_recording_frame, fps=2)
297
298
299
     audio_recording_frame.add_per_frame(metronome.switch)
     audio recording frame.add event processor(df.audio recorder.process event)
300
301
302
303
     # frame for starting to record audio
     record audio frame = GUI(window width, window height)
304
     record audio button = record audio frame.add button(
305
         "Start Recording",
306
         0.4, 0.0625, 0.3, 0.4,
307
         callback=lambda: start_recording_audio(record_audio_frame,
308
     audio recording frame)
309
     record_audio_frame.add_textbox(
310
311
         0.4, 0.0625, 0.3, 0.7
312
313
     record_audio_frame.add_text(
314
315
         text="Click button to start recording audio",
316
         center frac x=0.5,
317
         center frac y=0.25
318
     )
319
320
     # starting screen
     start frame = GUI(window width, window height)
321
322
     start frame.add button(
323
         "Start Production",
324
         0.4, 0.0625, 0.3, 0.4,
         callback=lambda: switch_frame(record_audio_frame)
325
326
327
```

```
328 | clk = Clock()
     running = True
329
     current_frame = start_frame
330
331
     while running:
332
         for event in pygame.event.get():
333
334
             exit = current_frame.event_trigger(event)
             if exit:
335
336
                 running = False
337
338
         current frame.reset()
         current_frame.render()
339
340
         clk.tick(current_frame.fps)
341
342
     pygame.quit()
343
344
```

Keyboard_Pi_Conductor/metronome.py

```
white = (255, 255, 255)
 2
    black = (0, 0, 0)
    red = (255, 0, 0)
 3
 4
 5
    import time
    from pygame import mixer
 6
 7
    class Metronome:
 8
        def init (self, frame, fps):
9
            self.frame = frame
10
            self.bit = False
11
12
13
            self.prev_switch = None
            self.fps = fps
14
            self.time_step = 1 / fps
15
16
17
            self.sound =
    mixer.Sound(f'/home/pi/Documents/ECE5725_final_proj/PythonPiano/Synth_Block_F_lo.wav'
    )
18
19
        def reset_fps(self, fps):
            self.fps = fps
20
            self.time_step = 1/fps
21
22
23
        def switch(self):
```

```
24
25
            if self.prev switch is not None and time.time() - self.prev switch <
    self.time_step:
26
                return
27
            self.prev_switch = time.time()
28
            self.bit = not self.bit
29
            center, radius, color = self.frame.circles[0]
30
31
32
            if self.bit:
33
                 self.frame.circles[0] = (center, radius, red)
34
35
                self.sound.play()
36
            else:
37
                 self.frame.circles[0] = (center, radius, white)
38
```

Keyboard_Pi_Conductor/mid_to_wav.py

```
from IR_Midi.modify_input import convert_midi_to_audio
midi_path = 'out/raw_midi.mid'
audio_path = 'recorded_audio.wav'
convert_midi_to_audio(midi_path, audio_path)
```

Keyboard_Pi_Conductor/Gesture_IR/init.py

```
1 |
```

Keyboard_Pi_Conductor/Gesture_IR/record_video.py

```
import cv2
 1
 2
    import numpy as np
    from threading import Thread
 4
    import time
 5
    # This is a script that records a video using the camera
 6
 7
8
    class VideoRecorder:
        def __init__(self, video_path, bpm):
9
10
11
            self.video_path = video_path
            self.initialize
12
13
```

```
14
        def initialize(self, bpm):
15
            # Create a VideoCapture object
            self.camera = cv2.VideoCapture(0)
16
17
            self.camera_open = False
18
            # Check if the camera is opened
19
            if not self.camera.isOpened():
20
                print("Cannot open camera")
2.1
22
                exit()
23
            self.bpm = bpm
2.4
            self.fps = self.bpm / 60
25
26
            # Define the codec and create a VideoWriter object
27
            fourcc = cv2.VideoWriter_fourcc(*'mp4v')
            self.video writer = cv2.VideoWriter(self.video path, fourcc, self.fps, (640,
28
    480))
29
        def spawn camera(self):
30
            self.recording = False
31
32
            self.camera_open = True
            self.record_thread = Thread(target=self.record)
33
            self.record_thread.start()
34
            self.timestamp = time.time()
35
36
        def start_recording(self):
37
            while not self.camera.isOpened():
38
39
                 time.sleep(0.01)
            self.recording = True
40
            self.timestamp = time.time()
41
42
43
        def stop recording(self):
44
            # Release everything if job is finished
45
46
            self.recording = False
            self.camera open = False
47
48
49
            self.record_thread.join()
50
            self.camera.release()
            self.video_writer.release()
51
            cv2.destroyAllWindows()
52
53
54
        def record(self):
55
            frames = []
56
57
            timestep = 1 / self.fps
            while self.camera_open and self.camera.isOpened():
58
```

```
59
                 ret, frame = self.camera.read()
60
                 if not ret:
61
                     print("Can't receive frame (stream end?). Exiting ...")
62
63
                     break
64
                # Flip the frame horizontally
65
                 # frame = cv2.flip(frame, 1)
66
67
                 frame = cv2.flip(frame, 0)
68
                 if self.recording and time.time() - self.timestamp > timestep:
69
                     self.timestamp = time.time()
70
                     print(f'Appending Frame at {self.timestamp}')
71
72
                     frames.append(frame)
73
                # Display the resulting frame
74
75
                cv2.imshow('frame', frame)
76
77
                k = cv2.waitKey(1) & 0xFF
78
                 if cv2.waitKey(1) & 0xFF == ord('q'):
79
                     break
80
            for f in frames:
81
                 self.video_writer.write(f)
82
83
84
    if name == " main ":
85
        recorder = VideoRecorder(video_path='output.mp4', bpm=90)
86
87
        input()
88
        recorder.spawn camera()
        print('camera spawned')
89
90
        input()
        recorder.start_recording()
91
92
        print('starting recording')
93
        input()
94
        recorder.stop_recording()
95
```

Keyboard_Pi_Conductor/Gesture_IR/gesture_ir.py

```
import cv2
import copy
import mediapipe as mp
import numpy as np
from tqdm import tqdm
```

```
import matplotlib.pyplot as plt
 7
 8
    from argparse import ArgumentParser
9
    def calc bounding rect(image, landmarks):
        image width, image height = image.shape[1], image.shape[0]
10
11
        landmark_array = np.empty((0, 2), int)
12
13
14
        for _, landmark in enumerate(landmarks.landmark):
            landmark_x = min(int(landmark.x * image_width), image_width - 1)
15
            landmark y = min(int(landmark.y * image height), image height - 1)
16
17
18
            landmark_point = [np.array((landmark_x, landmark_y))]
19
            landmark array = np.append(landmark array, landmark point, axis=0)
20
2.1
22
        x, y, w, h = cv2.boundingRect(landmark_array)
23
        return [x, y, x + w, y + h]
24
25
    def get_bounding_box(image, results):
26
        for hand_landmarks, handedness in zip(results.multi_hand_landmarks,
27
                                                 results.multi handedness):
28
            # Bounding box calculation
29
            brect = calc_bounding_rect(image, hand_landmarks)
30
            return brect
31
32
33
    def video_to_box_info(vid_path: str) -> list:
34
35
36
        video = cv2.VideoCapture(vid path)
37
38
        frame_count = video.get(cv2.CAP_PROP_FRAME_COUNT)
39
        fps = video.get(cv2.CAP PROP FPS)
        duration = frame count/fps
40
41
        info = {'fps': fps, 'duration': duration, 'frame_count': frame_count}
42
43
        # Check if video opened successfully
        if not video.isOpened():
44
            print("Error opening video file")
45
46
        mp hands = mp.solutions.hands
47
48
        hands = mp hands.Hands(
            static image mode=False,
49
            max num hands=1,
50
51
            min detection confidence=0.7,
```

```
52
           min tracking confidence=0.5,
53
       )
54
55
56
       progress bar = tqdm(total=frame count, desc="Processing frames", ncols=0)
57
       bounding_boxes = []
58
       # Read until video is completed
59
60
       curr frame = 1
       while video.isOpened():
61
62
63
           ret, image = video.read()
           if not ret:
64
65
               break
           image = cv2.flip(image, 1) # Mirror display
66
67
68
           # Detection implementation
   image = cv2.cvtColor(image, cv2.COLOR BGR2RGB)
69
70
71
           image.flags.writeable = False
           results = hands.process(image)
72
73
74
           image.flags.writeable = True
75
           76
           if results.multi_hand_landmarks is not None:
77
78
               # brect = get bounding box(image, results)
79
               thumb = results.multi_hand_landmarks[0].landmark[4]
80
               middle = results.multi hand landmarks[0].landmark[12]
               brect = (thumb.x - middle.x) ** 2 + (thumb.y - middle.y)**2 + (thumb.z -
81
   middle.z) **2
82
               brect = np.sqrt(brect)
83
               timestamp = curr frame / fps
84
85
86
87
               bounding boxes.append((timestamp, brect))
           curr frame += 1
88
           progress_bar.update(1)
89
90
91
       progress bar.close()
92
       # When everything done, release the video capture object
93
       video.release()
94
95
       # Closes all the frames
```

```
96
         cv2.destroyAllWindows()
 97
         return info, bounding_boxes
 98
 99
100
101
     def converter(timestamp, open_scale):
102
         assert len(timestamp) == len(open_scale)
103
104
         output = []
105
         for i in range(len(open_scale) - 1):
106
             output.append((timestamp[i], timestamp[i+1], (open scale[i+1] -
     open_scale[i])*100))
107
108
         return output
109
110
     def gesture_ir_main(video_path):
111
         info, bounding_boxes = video_to_box_info(video_path)
         diag_len = bounding_boxes
112
         # diag len = [(time, np.sqrt((box[2] - box[0])**2 + (box[3] - box[1])**2)) for
113
     time, box in bounding_boxes]
114
115
116
         timestamps, diag len = zip(*diag len)
117
         plt.plot(timestamps, diag_len)
118
         plt.savefig('plot.png')
119
120
         box sizes = np.array(diag len)
121
         max_expand, min_expand = box_sizes.max(), box_sizes.min()
122
123
         open_scale = (box_sizes - min_expand) / (max_expand - min_expand)
124
         ir = converter(timestamps, open scale)
125
         return info, ir
126
127
     if __name__ == "__main__":
128
129
         parser = ArgumentParser()
130
         parser.add_argument('--vid_path', type=str)
131
         args = parser.parse_args()
132
133
         print(gesture_ir_main(args.vid_path))
134
```

Keyboard_Pi_Conductor/GUI.py

```
1 import pygame
```

```
white = (255, 255, 255)
 4
    black = (0, 0, 0)
    red = (255, 0, 0)
 5
 6
 7
    def init system():
 8
        pygame.init()
 9
10
    class GUI:
        def init (self, width, height, fps=30):
11
12
            self.window_width = width
13
14
            self.window_height = height
            self.window = pygame.display.set_mode((self.window_width,
15
    self.window height))
16
            pygame.display.set_caption("Music Production")
17
18
            self.event components = []
            self.event processors = []
19
20
            self.idle_components = []
            self.circles = []
21
            self.per frame call = []
22
            self.text boxes = []
23
            self.active text box = -1
24
            self.running = False
25
26
            self.font = pygame.font.Font(None, 36)
27
28
            self.fps = fps
29
30
        def add button(self, text, width frac, height frac, center frac x,
31
    center_frac_y, callback=None):
32
            rect = pygame.Rect(
                center_frac_x * self.window_width,
33
                center frac y * self.window height,
34
35
                width_frac * self.window_width,
36
                height_frac * self.window_height
37
            )
38
            self.event_components.append((text, rect, callback))
39
            return len(self.event components) - 1
40
41
42
        def event_trigger(self, event):
43
            if event.type == pygame.QUIT:
44
45
                self.running = False
```

```
46
                 return True
47
            for text, component, cb in self.event_components:
48
                 if not event.type == pygame.MOUSEBUTTONDOWN:
49
50
                     continue
                 if component.collidepoint(event.pos) and cb is not None:
51
52
                     cb()
53
54
            self.active\_text\_box = -1
            for i in range(len(self.text boxes)):
55
                 text, component, active = self.text boxes[i]
56
                 if not event.type == pygame.MOUSEBUTTONDOWN:
57
58
                     continue
59
                 self.text boxes[i] = text, component, component.collidepoint(event.pos)
60
61
                 if component.collidepoint(event.pos):
62
                     self.active_text_box = i
63
            for i in range(len(self.text boxes)):
64
65
                 text, component, active = self.text boxes[i]
                 if not active:
66
                     continue
67
                 if event.type == pygame.KEYDOWN:
68
                     if event.key == pygame.K RETURN:
69
                         active = False
70
71
                         self.active text box = -1
72
                     elif event.key == pygame.K BACKSPACE:
73
                         text = text[:-1]
                     else:
74
75
                         text += event.unicode
                     self.text boxes[i] = text, component, active
76
77
78
79
            for processor in self.event processors:
                 processor(event)
80
81
82
            return False
83
84
        def add_text(self, text, center_frac_x, center_frac_y):
85
            text = self.font.render(text, True, black)
86
            text_rect = text.get_rect(center=(center_frac_x * self.window_width,
87
    center_frac_y * self.window_height))
88
            self.idle components.append((text, text rect))
89
90
```

```
91
         def add circle(self, radius, center frac x, center frac y):
             self.circles.append(((center frac x * self.window width, center frac y *
 92
     self.window_height), radius, black))
 93
 94
         def add textbox(self, width frac, height frac, center frac x, center frac y):
             rect = pygame.Rect(
 95
                 center_frac_x * self.window_width,
 96
                 center_frac_y * self.window_height,
 97
 98
                 width_frac * self.window_width,
                 height_frac * self.window_height
 99
100
             )
101
102
             self.text_boxes.append(('', rect, False))
103
104
         def get text(self):
105
             ret = []
106
             for text, _, _ in self.text_boxes:
                 ret.append(text)
107
108
109
             return ret
110
         def reset(self):
111
             self.window.fill(white)
112
113
114
115
         def start(self):
116
             self.running = True
117
         def add_per_frame(self, func):
118
119
             self.per frame call.append(func)
120
121
         def add event processor(self, func):
122
             self.event_processors.append(func)
123
         def render(self):
124
125
126
             for func in self.per_frame_call:
127
                 func()
128
             for text, component, cb in self.event components:
129
130
                 pygame.draw.rect(self.window, black, component, 2)
131
                 render text = self.font.render(text, True, black)
132
                 text rect = render text.get rect(center=component.center)
                 self.window.blit(render_text, text_rect)
133
134
135
```

```
136
             for text, component, active in self.text boxes:
137
                 color = (red if active else black)
                 pygame.draw.rect(self.window, color, component, 3)
138
                 if text == '' and not active:
139
                     text = 'Enter Text Here'
140
                 render text = self.font.render(text, True, black)
141
                 text_rect = render_text.get_rect(center=component.center)
142
143
                 self.window.blit(render_text, text_rect)
144
             for text, text rect in self.idle components:
145
                 self.window.blit(text, text rect)
146
147
148
             for center, radius, color in self.circles:
149
                 pygame.draw.circle(self.window, color, center, radius, width=0)
150
151
152
             pygame.display.flip()
153
154
155
     # pygame.init()
156
     # gui = GUI(width=800, height=600)
157
158
     # gui.add button(
           "Start Production",
159
           0.4, 0.0625, 0.3, 0.4,
160
           callback=lambda: print("START PRODUCTION")
161
     #)
162
163
     # gui.add_button(
164
165
           "See Result",
           0.4, 0.0625, 0.3, 0.65,
166
           callback=lambda: print("See Results")
167
     #)
168
169
170
     # gui.add text(
           text="Are you ready to produce your music?",
171
172
           center_frac_x=0.5,
173
           center frac y = 0.25
174
     #)
     # running = True
175
176
177
178
     # gui2 = GUI(width=800, height=600)
179
     # gui2.add button(
180
181
           "Start Production 2",
```

```
182 # 0.4, 0.0625, 0.3, 0.4,
     #
          callback=lambda: print("START PRODUCTION 2")
183
184 # )
185
     # gui2.add button(
186
          "See Result 2",
187
           0.4, 0.0625, 0.3, 0.65,
188
           callback=lambda: print("See Results 2")
189
190
     #)
191
     # gui2.add text(
192
          text="Are you ready to produce your music? 2",
193
          center_frac_x=0.5,
194
     #
          center_frac_y = 0.25
195
     #)
196
     # running = True
197
198
199
200 \# i = 0
201
     # while running:
202 #
         if i < 100:
              g = gui
203
         else:
204
205
              g = gui2
206
          for event in pygame.event.get():
207
              g.event trigger(event)
208
209
210
         g.reset()
211 #
         g.render()
          i += 1
212 #
213
214
215
216
217
218
219
220
```

Keyboard_Pi_Conductor/get_code.py

```
import os
from pathlib import Path
```

```
4
 5
    def write code(file):
 6
        with open('code.md', 'a') as f:
            # remove root path from file string
 7
            f.write(f' {file} \n\n')
 8
            f.write('```\n')
 9
            with open(file, 'r') as code_file:
10
                 lines = code_file.readlines()
11
12
                 f.writelines(lines)
            f.write('\n``')
13
            f.write('\n\n')
14
15
    def main(path):
16
17
        for f in os.listdir(path):
18
            f = os.path.join(path, f)
            if os.path.isdir(f):
19
20
                main(f)
21
            elif f.endswith('.py'):
                write code(f)
22
23
24
    if __name__ == "__main ":
        root_path = Path(__file__).parent
25
        print(f'{root path=}')
26
        main(root_path)
27
28
29
30
```

Keyboard_Pi_Conductor/IR_Midi/visualize_note.py

```
1
    from music21 import converter, stream, tempo, meter, chord, note
 2
 3
    import midi2audio
 4
    import pygame
    from IR Midi.modify input import modify volume
 5
 6
    from visual midi import Plotter, Preset
 7
    from pretty midi import PrettyMIDI
 8
 9
    def get_note_info(midi_file):
10
11
        score = converter.parse(midi_file)
12
        notes = []
        for element in score.flat.notesAndRests:
13
            if isinstance(element, stream. Voice):
14
15
                element = element.flat.notesAndRests
```

```
16
            if isinstance(element, chord.Chord):
17
                pitches = '.'.join(n.nameWithOctave for n in element.pitches)
                volume = element.volume.velocity
18
19
            elif isinstance(element, note.Note):
2.0
                pitches = element.pitch.nameWithOctave
                volume = element.volume.velocity
21
            else:
22
                pitches = ''
2.3
24
                volume = 0
25
26
            notes.append({
27
                 "pitch": pitches,
28
                 "start": element.offset,
29
                 "end": element.offset + element.duration.quarterLength,
                 "volume": volume
30
31
            })
32
33
        return notes
34
35
36
    def draw_note_volumes(screen, notes, y_offset, color):
37
        for note in notes:
            start x = int(note["start"] * 50)
38
            end x = int(note["end"] * 50)
39
            volume = int(note["volume"] / 127 * screen.get_height() / 2)
40
41
            pygame.draw.rect(screen, color, (start x, y offset -
                              volume, end x - start x, volume))
42
            pygame.draw.rect(screen, (0, 0, 0), (start x,
43
44
                              y_offset - volume, end_x - start_x, volume), 1)
45
46
            font = pygame.font.Font(None, 24)
47
            pitch text = font.render(note["pitch"], True, (0, 0, 0))
            volume_text = font.render(str(note["volume"]), True, (0, 0, 0))
48
49
            screen.blit(pitch text, (start x, y offset - volume - 20))
            screen.blit(volume text, (start x, y offset - volume - 40))
50
51
52
53
    def draw modified notes(screen, original notes, modified notes):
        for i in range(len(original notes)):
54
            if original notes[i]["volume"] != modified notes[i]["volume"]:
55
                start x = int(modified notes[i]["start"] * 50)
56
                end x = int(modified notes[i]["end"] * 50)
57
58
                volume = int(modified notes[i]["volume"] /
                              127 * screen.get height() / 2)
59
                pygame.draw.rect(screen, (255, 0, 0), (start x,
60
                                  screen.get height() - volume, end_x - start_x, volume))
61
```

```
62
                 pygame.draw.rect(screen, (0, 0, 0), (start x, screen.get height(
 63
                  ) - volume, end x - start x, volume), 1)
 64
 65
                 font = pygame.font.Font(None, 24)
 66
                 pitch text = font.render(
                     modified notes[i]["pitch"], True, (0, 0, 0))
 67
                 volume_text = font.render(
 68
                     str(modified_notes[i]["volume"]), True, (0, 0, 0))
 69
 70
                 screen.blit(
                     pitch text, (start x, screen.get height() - volume - 20))
 71
 72
                 screen.blit(
 73
                     volume_text, (start_x, screen.get_height() - volume - 40))
 74
 75
     def visualize notes(original file, modified file):
 76
 77
         original_notes = get_note_info(original_file)
 78
         modified_notes = get_note_info(modified_file)
 79
         pygame.init()
 80
 81
         screen_width, screen_height = 800, 600
         screen = pygame.display.set_mode((screen_width, screen_height))
 82
         pygame.display.set caption("MIDI Note Volume Visualization")
 83
 84
         running = True
 85
 86
         while running:
 87
             for event in pygame.event.get():
                 if event.type == pygame.QUIT:
 88
                     running = False
 89
 90
 91
             screen.fill((255, 255, 255))
 92
 93
             draw note volumes(screen, original notes,
 94
                                screen_height // 2, (0, 0, 255))
 95
             draw note volumes(screen, modified notes, screen height, (0, 0, 255))
             draw modified notes (screen, original notes, modified notes)
 96
 97
 98
             pygame.display.flip()
 99
100
         pygame.quit()
101
102
103
     def visualize midi(original file, modified file):
104
         original pm = PrettyMIDI(original file)
105
         modified pm = PrettyMIDI(modified file)
106
107
         preset = Preset(plot width=800, plot height=400, row height=50)
```

```
108
         plotter = Plotter(preset, plot max length bar=16)
109
         plotter.show(original_pm, "/tmp/original_midi.html")
110
         plotter.show(modified_pm, "/tmp/modified_midi.html")
111
112
113
     def test_visual():
114
115
         volume changes = [
116
             (3.0, 8.0, 130), # Louder at 7s, lasting for 3s
             (10.0, 380.0, -100) # Quieter at 15s, lasting for 3s
117
118
119
120
         modified_score = modify_volume('input.mid', volume_changes)
121
         modified_score.write('midi', 'output_file.mid')
122
         visualize_notes('input.mid', 'output_file.mid')
123
124
         visualize_midi('input.mid', 'output_file.mid')
125
126
127
     if __name__ == "__main__":
         test_visual()
128
129
```

Keyboard_Pi_Conductor/IR_Midi/merge_midi.py

```
from music21 import converter, instrument, note, chord, stream
 1
 2
 3
    def merge midi(path 1, path2, out path):
 4
 5
 6
        Merge two tracks into 1.
 7
 8
        Args:
9
            path_1, paht_2 (str): Path to the MIDI file.
10
            out_path (str): Path to the output MIDI file
11
12
        Returns:
            music21.stream.Stream: Modified MIDI score.
13
14
        11 11 11
        midi file1 = converter.parse(path 1)
15
        midi_file2 = converter.parse(path2)
16
17
18
        merged stream = stream.Stream()
19
20
        for element in midi_file1.flat:
```

```
21
            merged stream.append(element)
22
23
        for element in midi_file2.flat:
24
            merged_stream.append(element)
25
        merged stream.write("midi", fp=out path)
26
27
28
    if __name__ == "__main__":
29
        path 1 = "/home/pi/Documents/ECE5725 final proj/testrun1 final midi.mid"
30
        path 2 = "/home/pi/Documents/ECE5725 final proj/IR Midi/input.mid"
31
        out = "/home/pi/Documents/ECE5725_final_proj/IR_Midi/merged_output.mid"
32
33
        merge_midi(path_1, path_2, out)
34
```

Keyboard_Pi_Conductor/IR_Midi/modify_input.py

```
1
    from music21 import converter, stream, tempo, meter
 2
    import midi2audio
 3
    import pygame
 5
    import numpy as np
 6
 7
    def convert_midi_to_audio(midi_file, audio_file):
 8
 9
10
        Convert a MIDI file to an audio file using midi2audio.
11
12
        Args:
13
            midi_file (str): Path to the MIDI file.
14
             audio_file (str): Path to save the audio file.
        .....
15
16
        fs = midi2audio.FluidSynth(
             sound_font='/usr/share/sounds/sf2/FluidR3_GM.sf2')
17
        fs.midi_to_audio(midi_file, audio_file)
18
19
20
    def play_audio_file(audio_file):
21
        11 11 11
22
        Play an audio file using pygame.
23
24
25
        Args:
             audio_file (str): Path to the audio file.
2.6
27
28
        print("start play audio")
```

```
29
        pygame.mixer.init()
30
        pygame.mixer.music.load(audio file)
31
        pygame.mixer.music.play()
32
        while pygame.mixer.music.get busy():
33
            pygame.time.delay(100)
        pygame.mixer.quit()
34
35
36
37
    def modify_volume(midi_file, volume_changes, out_midi_file, bpm=60):
38
        Modify the volume of a MIDI file based on user input.
39
40
41
        Args:
42
            midi file (str): Path to the MIDI file.
            volume changes (list): List of tuples containing (start time, end time) and
43
    volume change.
44
45
        Returns:
            music21.stream.Stream: Modified MIDI score.
46
        .....
47
        # Load the MIDI file
48
49
        score = converter.parse(midi file)
50
51
        flat score = score.flat
52
        # bpm=score.flat.getElementsByClass(tempo.MetronomeMark)[0].number
53
        for note in flat score.notes:
54
55
            print(f'note offset: {note.offset}')
56
57
        volume = 0
        low, high = 0, 0
58
59
        for _, _, volume_change in volume_changes:
            volume += volume_change
60
            low = min(low, volume)
61
            high = max(high, volume)
62
63
64
        for i in range(len(volume_changes)):
            t1, t2, volume change = volume changes[i]
65
            volume_changes[i] = (t1, t2, volume_change / (high - low) * 87)
66
67
        i = 0
68
        volume = 0 - low + 40
69
70
71
        note_volume_list = []
72
        while flat score.notes[i].offset < volume changes[i][0]:</pre>
73
            note volume list.append(volume)
```

```
74
             i += 1
 75
 76
         buffer = []
 77
         for start time seconds, end time seconds, volume change in volume changes:
 78
             end offset = bpm * end time seconds / 60
 79
 80
             while i < len(flat score.notes) and flat score.notes[i].offset < end offset:
 81
 82
                 buffer.append(flat score.notes[i])
                 i += 1
 83
 84
             for note in buffer:
 85
 86
                 volume += volume_change / len(buffer)
 87
                 note volume list.append(volume)
 88
 89
             buffer = []
 90
         while i < len(flat score.notes):
 91
 92
             note volume list.append(volume)
 93
             i += 1
 94
         note volume list = np.array(note volume list)
 95
         note volume list = (note volume list - note volume list.min()) / \
 96
             (note_volume_list.max() - note_volume_list.min()) * 100 + 20
 97
 98
         print('note volume list: ', note volume list)
 99
         for i in range(len(note volume list)):
100
             flat score.notes[i].volume.velocity = note_volume_list[i]
101
102
103
         score.write('midi', out midi file)
104
105
     def test_volume_modification():
106
107
         volume changes = [
108
             (3.0, 8.0, 130), # Louder at 7s, lasting for 3s
109
             (10.0, 380.0, -100) # Quieter at 15s, lasting for 3s
110
         1
111
         modified score = modify volume(
112
              '/home/pi/Documents/ECE5725_final_proj/output_file.mid', volume_changes,
113
     'output file.mid')
114
115
         # Convert the modified MIDI file to an audio file
         audio file = 'output audio.wav'
116
117
         convert midi to audio('output file.mid', audio file)
118
```

```
# Play the audio file
play_audio_file(audio_file)

121

122

123 if __name__ == "__main__":
    test_volume_modification()

125  # import sys

126  # play_audio_file(sys.argv[1])

127
```

Keyboard_Pi_Conductor/IR_Midi/init.py

```
1 |
```