Digital Piano

Ⅰ. Functions & Implementations

1. The GUI is designed according to the real piano keyboard.

   This function is implemented using the tkinter library.

2. Two types of input, by mouse and by keyboard, to play the sound of piano keys.

```
frame10 = Tk.Frame(frame12)
frame10.pack(side = Tk.TOP)
B_c4 = Tk.Button(frame10, command = fun_c4, textvariable = btn_text1, padx = 16, pady = 16, height = 8, bd
B_c4.pack(side = Tk.LEFT)
B_d4 = Tk.Button(frame10, command = fun_d4, textvariable = btn_text3, padx = 16, pady = 16, height = 8, bd
B_d4.pack(side = Tk.LEFT)
B_d4 = Tk.Button(frame10, command = fun_e4, textvariable = btn_text5, padx = 16, pady = 16, height = 8, bd
B_d4.pack(side = Tk.LEFT)
B_d4 = Tk.Button(frame10, command = fun_f4, textvariable = btn_text6, padx = 16, pady = 16, height = 8, bd
B_d4.pack(side = Tk.LEFT)
B_d4 = Tk.Button(frame10, command = fun_g4, textvariable = btn_text8, padx = 16, pady = 16, height = 8, bd
B_d4.pack(side = Tk.LEFT)
B_d4 = Tk.Button(frame10, command = fun_a4, textvariable = btn_text10, padx = 16, pady = 16, height = 8, b
B_d4.pack(side = Tk.LEFT)
B_d4 = Tk.Button(frame10, command = fun_b4, textvariable = btn_text12, padx = 16, pady = 16, height = 8, b
B_d4.pack(side = Tk.LEFT)
```

```
# Define TK root
root = Tk.Tk()
root.bind('q', fun_quitkey)
root.bind('a', fun_c4key)
root.bind('w', fun_c40key)
root.bind('s', fun_d4key)
root.bind('e', fun_d40key)
root.bind('d', fun_e4key)
root.bind('f', fun_f4key)
root.bind('t', fun_f40key)
root.bind('g', fun_g4key)
root.bind('y', fun_g40key)
root.bind('h', fun_a4key)
root.bind('u', fun_a40key)
root.bind('j', fun_b4key)
```

```
def fun_c4():
    # print('C4')
    global KEYPRESS
    KEYPRESS[0] = True
```

```
def fun_c4key(event):
    # print('C4')
    global KEYPRESS
    global KEY
    KEY = event.char
    KEYPRESS[0] = True
```

3. Multiple keys can be pressed at the same time to achieve chords.

   All the changeable parameters related to playing sounds should be an array whose length is 12, so that each key would not influence each other.

```
k = [0] * 12
f1 = [0.0] * 12
om1 = [0.0] * 12
a = [[0.0] * 3] * 12
b = [[0.0]] * 12
states = [np.zeros(ORDER)] * 12
x = [np.zeros(BLOCKLEN)] * 12
y = [np.zeros(BLOCKLEN)] * 12
KEY = ''
KEYPRESS = [False] * 12
CONTINUE = True
```

```
if TYPE == 0:
    ytotal = np.zeros(BLOCKLEN)
    for i in range(12):
        if KEYPRESS[i] and CONTINUE:
            x[i][0] = 10000.0

        [y[i], states[i]] = signal.lfilter(b[i], a[i], x[i], zi = states[i])

        x[i][0] = 0.0
        KEYPRESS[i] = False

        y[i] = np.clip(y[i].astype(int), -MAXVALUE, MAXVALUE)     # Clipping
        ytotal += y[i]
```

4. The musical scale can be changed by GUI slider.

```
key_range = Tk.IntVar()
# range.set(1)
slider = Tk.Scale(frame12, from_ = 0, to = 2, variable = key_range, orient = Tk.HORIZONTAL, showvalue = 0
slider.set(1)
slider.pack()
B_label15 = Tk.Label(frame12, text = 'left: C3-B3          center: C4-B4          right: C5-B5')
B_label15.pack()
```

```
def set_key(r):
    global f0
    if r == 0:
        f0 = 523 / 2
        btn_text1.set('C3')
        btn_text2.set('C3#')
        btn_text3.set('D3')
        btn_text4.set('D3#')
        btn_text5.set('E3')
        btn_text6.set('F3')
        btn_text7.set('F3#')
        btn_text8.set('G3')
        btn_text9.set('G3#')
        btn_text10.set('A3')
        btn_text11.set('A3#')
        btn_text12.set('B3')
```

5. Special sound effects like chorus and echo are implemented besides of normal piano sounds.

   At first it sounds terrible after adding effects. Then we solved it by

changing all the changeable parameters related to the effects to an array whose length is 12, so that each key would not influence each other.

```
# Buffer to store past signal values. Initialize to zero.
BUFFER_LEN =  1024          # Set buffer length.  Must be more than N!
buffer = [BUFFER_LEN * [0]] * 12   # list of zeros

# Initialize buffer indices
kr = [0] * 12      # read index
kw = [0] * 12      # write index
```

```
def initFrequency(i):
    f1[i] = pow(2, k[i] / 12.0) * f0
    om1[i] = 2.0 * math.pi * float(f1[i]) / RATE
    a[i] = [1, -2 * r * math.cos(om1[i]), r**2]
    b[i] = [r * math.sin(om1[i])]


for i in range(12):
    k[i] = i
    # print('key ' + event.char + ' is ' + str(k))
    initFrequency(i)
    states[i] = np.zeros(ORDER)
    x[i] = np.zeros(BLOCKLEN)
    buffer[i] = BUFFER_LEN * [0]   # list of zeros
    kr[i] = 0
    kw[i] = 0

    KEYPRESS[i] = False
```

```
elif TYPE == 1:
    ytotal = np.zeros(BLOCKLEN)
    for i in range(12):
        if KEYPRESS[i] and CONTINUE:
            x[i][0] = 10000.0

        [y[i], states[i]] = signal.lfilter(b[i], a[i], x[i], zi = states[i])

        x[i][0] = 0.0
        KEYPRESS[i] = False

        tmp = np.zeros(BLOCKLEN)
        for j in range(0, BLOCKLEN):
            # Get previous and next buffer values (since kr is fractional)
            kr_prev = int(math.floor(kr[i]))
            frac = kr[i] - kr_prev     # 0 <= frac < 1
            kr_next = kr_prev + 1
            if kr_next == BUFFER_LEN:
                kr_next = 0

            # Compute output value using interpolation
            tmp[j] = (1 - frac) * buffer[i][kr_prev] + frac * buffer[i][kr_next]
```

```
        # Update buffer
        buffer[i][kw[i]] = y[i][j]

        # Increment read index
        kr[i] = kr[i] + 1 + W * math.sin( 2 * math.pi * f0 * j / RATE )
            # Note: kr is fractional (not integer!)

        # Ensure that 0 <= kr < BUFFER_LEN
        if kr[i] >= BUFFER_LEN:
            # End of buffer. Circle back to front.
            kr[i] = kr[i] - BUFFER_LEN

        # Increment write index
        kw[i] = kw[i] + 1
        if kw[i] == BUFFER_LEN:
            # End of buffer. Circle back to front.
            kw[i] = 0

    tmp = np.clip(tmp.astype(int), -MAXVALUE, MAXVALUE)     # Clipping
    ytotal += tmp
```

```
elif TYPE == 2:
    ytotal = np.zeros(BLOCKLEN)
    for i in range(12):
        if KEYPRESS[i] and CONTINUE:
            x[i][0] = 10000.0

        [y[i], states[i]] = signal.lfilter(b[i], a[i], x[i], zi = states[i])

        x[i][0] = 0.0
        KEYPRESS[i] = False

        tmp = np.zeros(BLOCKLEN)
        for j in range(0, BLOCKLEN):
            # Get previous and next buffer values (since kr is fractional)
            kr_prev = int(math.floor(kr[i]))
            frac = kr[i] - kr_prev     # 0 <= frac < 1
            kr_next = kr_prev + 1
            if kr_next == BUFFER_LEN:
                kr_next = 0

            # Compute output value using interpolation
            tmp[j] = y[i][j] + 0.5 * ((1-frac) * buffer[i][kr_prev] + frac * buffer[i][kr_next])

            # Update buffer
            buffer[i][kw[i]] = y[i][j]
```

```
        # Increment read index
        kr[i] = kr[i] + 1 + depth / 2.0 / 80 * math.sin(2 * math.pi * 2 * j / RATE)
            # Note: kr is fractional (not integer!)

        # Ensure that 0 <= kr < BUFFER_LEN
        if kr[i] >= BUFFER_LEN:
            # End of buffer. Circle back to front.
            kr[i] = kr[i] - BUFFER_LEN

        # Increment write index
        kw[i] = kw[i] + 1
        if kw[i] == BUFFER_LEN:
            # End of buffer. Circle back to front.
            kw[i] = 0

    tmp = np.clip(tmp.astype(int), -MAXVALUE, MAXVALUE)     # Clipping
    ytotal += tmp
```

6. Two rhythms are added so that we can play the piano along with the beats, and stop them at any time.

   When trying to play the piano (stream) and the rhythm (wav file) at the same time:

   1) Failed. Write to the stream at the same time. But the stream needs to set the play rate, but the frequency of piano sound and the rate of the rhythm are different.
   2) Succeed. Play the piano by the stream and play the rhythm by Pygame. Pygame Mixer has two classes: Sound and Music. Music is used to play the background music and Sound is used to play some sound effects. Specifically, Music just supports one music at a time, but Sound supports several sounds playing at the same time. Thus, I chose Sound. And I found using multithreading or not both work for Pygame.Mixer.Sound to play the piano sound and rhythm audio at the same time.

```
# Initialize pygame mixer
pygame.mixer.init(frequency = RHYTHM_RATE1, channels = 1)
```

```
def play_rhythm():
    global PLAY_RHYTHM1, PLAY_RHYTHM2
    # global rhythm1, rhythm2
    # print('play', wavfile)

    # playsound.playsound(wavfile)


    # pygame.mixer.pre_init(frequency = (int)(RHYTHM_RATE1), channels = 1)

    # pygame.mixer.music.load(wavfile)
    # pygame.mixer.music.play()
    my_sound = pygame.mixer.Sound('rhythm_cut1.wav')
    if PLAY_RHYTHM1:
        my_sound = pygame.mixer.Sound('rhythm_cut1.wav')
    elif PLAY_RHYTHM2:
        my_sound = pygame.mixer.Sound('rhythm2_cut2.wav')
    my_sound.set_volume(0.3)
    my_sound.play()
    # time.sleep(2.01)
    # pygame.mixer.quit()
```

```
    if ((not PLAY_RHYTHM1) and (not PLAY_RHYTHM2)):
        pygame.mixer.stop()
    if (not pygame.mixer.get_busy()) and (PLAY_RHYTHM1 or PLAY_RHYTHM2):
        play_rhythm()

    play_notes()
```

7. The melody that user plays could be recorded into a wave file.

```
is_recording = False
def record():
    global is_recording
    # global output_wf
    is_recording = True
```

```
def stopRecording():
    global is_recording
    is_recording = False
```

```
 # write signal to wave file
 if is_recording:
     output_wf.writeframes(binary_data)
```

8. The quit function is applied.

```
B_quit = Tk.Button(frame11, command = fun_quit, text = 'Quit', height = 2, bd = 2, fg = 'black', width
B_quit.pack()
```

```
def fun_quit():
  global CONTINUE
  print('Good bye')
  CONTINUE = False
```

II Something to improve

1. Change rhythm playback rate by GUI slide.
    1) Failed. Tried wave package, but there is only one output stream which only has one rate parameter, but the rate of keys and rate of the rhythm cannot be the same.
    2) Failed. Tried playsound package, but it can only play the sound but not change any parameter.
    3) Failed. Tried pygame package, but it can only change frequency,

which influences the pitch instead of playback rate.

In the end, give up changing playback rate.

2. The piano can be pressed by both Mouse or Keyboard, but when pressing keyboard, the GUI will not show the pressing action, which is a little confusing. It might be achieved by using something else instead of Tkinter.