AI Assigment-2

Student: Arlan Kuralbayev

Mail: a.kuralbayev@innopolis.university

Group: Bs20-03

First of all, I don't know much about music theory. I wrote two solutions, one without mutation, I just choose random chord progressions, the second with.

First Solution

I added some popular progressions. Here I just created Midi files. In new_mid first solution, in new_mid2 second

```
mid = MidiFile(file)
score = music21.converter.parse(file)
key = score.analyze('key')

new_mid = MidiFile()
new_mid2 = MidiFile()
new_track = MidiTrack()
new_track2 = MidiTrack()
new_mid.ticks_per_beat = mid.ticks_per_beat
new_mid2.ticks_per_beat = mid.ticks_per_beat
new_mid.tracks.append(new_track)
new_mid2.tracks.append(new_track2)
```

I just choose the progression until the length of accompaniment will be equal to the melody. Also adding speed for progression. Then table_walk that will walk in Major/minor table. And add melody track to new_mid

```
def Solution():
    while new_mid.length < mid.length:
        progression = get_progression()
        speed = speed_of_progression[progressions.index(progression)]
        table_walk(progression, speed)
    new_mid.tracks.append(mid.tracks[1])
    new_mid.save("output.mid")
```

Run our progression and compare it to Minor/major table.

get_notes will get list of notes in midi format.

add_notes will add that notes to midi track with speed of each chords.

Also check if our lenght is not over melody

```
def table_walk(progression, speed):
    if key.mode == 'minor':
        for i in minor_table:
            if i[0] == (key.tonic.name + 'm'):
                for j in progression:
                    add_notes(get_notes(i[int(j[0]) - 1], j), speed[progression.index(j)])
                    if new_mid.length >= mid.length:
                        return
    elif key.mode == 'major':
        for i in major_table:
            if i[0] == (key.tonic.name):
                for j in progression:
                    add_notes(get_notes(i[int(j[0]) - 1], j), speed[progression.index(j)])
                    if new_mid.length >= mid.length:
                        return
```

Adding notes to track

We can change octave in change_octave also velocity

```
def add_notes(notes, speed):
    VELOCITY = 64
    notes = change_octave(notes)
    for i in notes:
        new_track.append(Message('note_on', channel=0, velocity=VELOCITY, note=i, time=0))
    for i in notes:
        if i == notes[0]:
            new_track.append(Message('note_off', channel=0, note=i, velocity=VELOCITY,  time=int(mid.ticks_per_beat*2/speed)))
        else:
            new_track.append(Message('note_off', channel=0, note=i, velocity=VELOCITY, time=0))
```

Get randomly progression from created ones. And get notes by using first note

```
def get_progression():
    return random.choice(progressions)

# Simple minor/major/sus2/sus4/diminished
def get_minor(first_note):
    notes = []
    notes.append(first_note)
    notes.append(first_note + 3)
    notes.append(first_note + 7)
    return notes
```

Second Solution

Just templates, in each individum I will contain 4 chords it means 12 notes. Top 10 individums I will crossing.

```
MAX_GENERATION = 20
MIN_SUBJECT = 200
MAX_SUBJECTS = 1000
TOP_INDIVID = 10
CNT_CHORDS = 4
individ = []
rate = []
scale = []
rank = []
```

Here my main Algorithm for mutations

First I calculate notes that can be played in our melody

Then I create first generation randomly

Then I run Max_generation times

Firstly I calculate score of each individum

Then according to average score delete useless

Then According ranking I crossing Top individums and create one for each crossing

And then I add the best Individums to track

```
def Solution2():
    calculate_scale()
    for i in range (0, MIN_SUBJECT):
        notes = []
        for j in range (0, CNT_CHORDS):
            symbol = random.randint(48, 59)
            type = random.randint(1, 5)
            note = get_notes2(symbol, type)
            for k in note:
                notes.append(k)
        individ.append(notes)
        rate.append(0)
        rank.append(0)
    for i in range (0, MAX_GENERATION):
        avg = math.ceil(calculate_rate())
        delete_lowest(avg)
        calculate_rank()
        crossing ()
    while mid.length > new_mid2.length:
        for i in range (0, len(individ)):
            if rank[i] <= 1:
                add_notes2(individ[i])
            if mid.length <= new_mid2.length:
                break
    new_mid2.tracks.append(mid.tracks[1])
    new_mid2.save("output.mid")
```

```
# Here I crossing two individums
# Choicing them by TOP_INDIVID
# then randomly choice which chord I will swap
# I will create the first individum with one part from second individum
def crossing():
    for i in range(0, len(individ)):
        if rank[i] <= TOP_INDIVID:
            for j in range(0, len(individ)):
                if rank[j] <= TOP_INDIVID:
                    if len(individ) >= MAX_SUBJECTS:
                        return
                    x = random.randint(0, CNT_CHORDS - 1)
                    y = random.randint(0, CNT_CHORDS - 1)
                    notes = individ[i]
                    notes[x * 3] = individ[j][y * 3]
                    notes[x * 3 + 1] = individ[j][y * 3 + 1]
                    notes[x * 3 + 2] = individ[j][y * 3 + 2]
                    individ.append(notes)
                    rate.append(0)
                    rank.append(0)
```