

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Piano Follower

propusă de

Crivoi Andrei

Sesiunea: *iulie, 2019*

Coordonator științific

Pistol Ionuț

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI
FACULTATEA DE INFORMATICĂ

Piano Follower

Crivoi Andrei

Sesiunea: *iulie, 2019*

Coordonator științific

Pistol Ionuț

Cuprins

Capitolul I: Introducere	5
Descrierea problemei	6
Descrierea soluției.....	6
Capitolul II: Tehnologii folosite.....	8
Încărcarea datelor.....	9
Constant Q Transform	10
Estimarea atacului notelor.....	12
Estimarea Tempo-ului.....	13
Interfața grafică.....	13
Import și Export de fișiere	17
Reprezentări Grafice	17
Input/Output MIDI.....	19
Înregistrarea audio externă.....	21
Fișiere MXL.....	22
Fișiere WAV	23
Capitolul III: Descrierea aplicației	25
Interfața principală	26
Clasa DefaultToplevel și moștenitorii ei	31
Clasa WavManagerToplevel.....	34
Transcrierea Muzicală.....	37
Clasa MidiManagerToplevel și interacțiunea cu fișiere MIDI	39
Clasa ScoreManagerToplevel	41
Clasa RecorderToplevel și înregistrarea audio externă	42
Clasa PianoManagerToplevel	44
Capitolul IV: Concluzii	45
Bibliografie:	46

Capitolul I: Introducere

Lucrarea de față abordează un subiect deosebit, procesarea semnalului digital în vederea obținerii partiturii pornind de la o înregistrare sau clip audio, subiect care m-a atras și pe care am decis să-l tratez, luând în considerare profesia mea de viitor programator și pasiunea pe care o port muzicii, în special cea electronică. Tematica lucrării este una de actualitate, în contextul dezvoltării continue a pieței muzicale digitale și a creșterii uriașe a numărului producătorilor de muzică, înregistrată în ultimii ani la nivel global. Lucrarea are drept obiectiv crearea unui instrument electronic capabil de a citi informația dintr-un fișier audio și de a oferi posibilitatea de a scrie, la alegere, un fișier midi care să conțină notele muzicale din fișierul primit ca input sau chiar partitura echivalentă acestor fișiere, cu ajutorul formatului MXL ce urmează a fi descris în această teză.

Alte aplicații similare, capabile să lucreze cu fișiere de tip MXL sunt:

- MuseScore – aplicație creată cu scopul lucrului cu partituri, oferă posibilitatea de a crea, vizualiza și edita partituri și este aplicația pe care am folosit-o pentru afișarea partiturilor
- Myriad Melody Assistant
- MakeMusic Finale
- Avid Sibelius
- Steinberg Doricos
- flat.io – aplicație web folosită pentru crearea partiturilor

Primul capitol prezintă o scurtă introducere în temă, motivația alegerii acesteia, o descriere succintă a aplicației și a modului acesteia de funcționare. În al doilea capitol sunt menționate tehnologiile folosite în vederea realizării soluției, resurse și librării necesare implementării metodei alese pentru a rezolva problema transcrierii muzicale. Capitolul trei aduce prezentarea soluției, modul în care aplicația beneficiază de tehnologiile amintite în capitolul anterior și descrierea implementării acesteia.

Descrierea problemei

Marea majoritate a oamenilor nu dețin cunoștințe de teorie muzicală suficiente. Din acest motiv, mulți dintre aceștia nu sunt capabili să recunoască notele unei piese după ureche, cu atât mai mult, nu sunt capabili să remarce gama unei piese și nu sunt interesați să învețe să cânte la un instrument deoarece necesită mult timp și devotament. Aplicația propusă de mine dorește să facă acest lucru mai ușor pentru oricine dorește să învețe să folosească un instrument (în cazul de față, instrumentul preferat este pianul), să învețe teorie muzicală sau doar să genereze un clip MIDI ce conține notele muzicale ale piesei preferate pentru a realiza un edit sau remix, sau doar pentru pură curiozitate.

Pentru cei mai studioși, aplicația prezintă un modul în care userul poate compara două partituri la alegere, având astfel capacitatea de a vedea dacă înregistrarea proprie se apropie destul de partitura model și pentru a se putea autoevalua și a își corecta greșelile.

Descrierea soluției

Pentru realizarea acestei aplicații, limbajul de programare preferat este Python, datorită multitudinii de librării utile pe care acesta le pune la dispoziție cum ar fi:

- NumPy – librărie fundamentală pentru calcule științifice în Python, folosită în acest caz pentru pachetul FFT
- Matplotlib – librărie utilizată pentru reprezentarea 2D a tuturor graficelor necesare aplicației
- Wave – librărie ce pune la dispoziție interacțiunea cu fișiere de tip Wav
- PyAudio – librărie ce oferă posibilitatea de a executa operații I/O cu fișiere audio cross-platform
- TkInter – librărie utilizată pentru interfața grafică a aplicației
- Music21 – librărie implementată de studenții de la MIT ce pune la dispoziție lucrul cu fișiere de tip MusicXML
- Mido – librărie ce oferă posibilitatea de a citi sau scrie fișiere MIDI
- Librosa – pachet Python pentru analiza audio. Oferă resursele necesare pentru a crea sisteme de obținere a informațiilor din muzică

- PyGame – librărie utilizată pentru preluarea inputului MIDI

Pentru realizarea conversiei de la format audio la format MIDI sau MXL, proces cunoscut ca transcriere muzicală, sunt disponibile o multitudine de metode studiate de-a lungul timpului, toate având la bază un proces similar ce constă în:

- Estimarea frecvențelor
- Recunoașterea instrumentului
- Calcularea tempo-ului
- Detectarea atacului și a finalului notelor
- Structurarea modelului transcris

Pentru compararea partiturilor, este realizat un grafic frecvență – timp ce arată pentru fiecare partitură nota și momentul în care aceasta a fost introdusă, astfel, se pot observa diferențele dintre partituri și ce trebuie făcut pentru a reda mai bine partitura model.

Capitolul II: Tehnologii folosite

În melodiile scoase pe piață, plaja de instrumente și diferite sunete este mult prea mare, producătorii au în minte încă de la începutul proiectului faptul că, pentru un sunet curat și plin al înregistrării lor, trebuie să umple spectrul de frecvențe cu diferite elemente, de la bass (pentru frecvențele joase) până la pian și percuții (pentru frecvențele medii și înalte). Acest fapt face depistarea exactă a frecvenței și atacului unei note anume, de pe un instrument anume, foarte dificilă, poate chiar imposibilă.

Până în prezent, nu s-a găsit o metodă 100% precisă pentru realizarea transcrierii muzicale, datorită tuturor sunetelor ambientale ce apar în înregistrări, a aglomerării de instrumente, a neclarității, sau pur și simplu a unor erori de estimare a algoritmilor matematici folosiți. Din acest motiv, aplicația funcționează optim pentru înregistrări simple, clare, ce conțin un singur instrument și nu conțin alte sunete și elemente de percuție care să alterneze frecvența.

Cea mai folosită și studiată metodă pentru realizarea procesului de transcriere muzicală este *Transformarea Fourier*, mai precis versiunea îmbunătățită a acesteia, *Transformarea Fourier Rapidă*. Aceasta calculează Transformarea Discreta Fourier (DFT) a unei secvențe sau inversa acestei transformări (IDFT). Analiza Fourier convertește semnalul digital din domeniul original (în acest caz timpul) într-o reprezentare a frecvențelor și vice versa.

Din păcate, transformarea Fourier este limitată în precizie în situația de față. Pentru un rezultat mai corect, propun folosirea unui procedeu asemănător și anume *Transformarea Constant Q*, procedeu înrudit cu complexa transformare *Morlet wavelet*. Acesta se apropie mult mai mult de percepția umană, producând rezultate mult mai precise ce nu erau posibile folosind Transformarea Fourier. În figura de mai jos (Figura 1) se pot observa diferențele dintre transformarea Fourier (roșu) și cea Constant Q (albastru). Transformarea Fourier pe un domeniu frecvență timp se comportă liniar, în vreme ce transformarea Constant Q obține

rezultate logaritmice, similar percepției umane (verde)

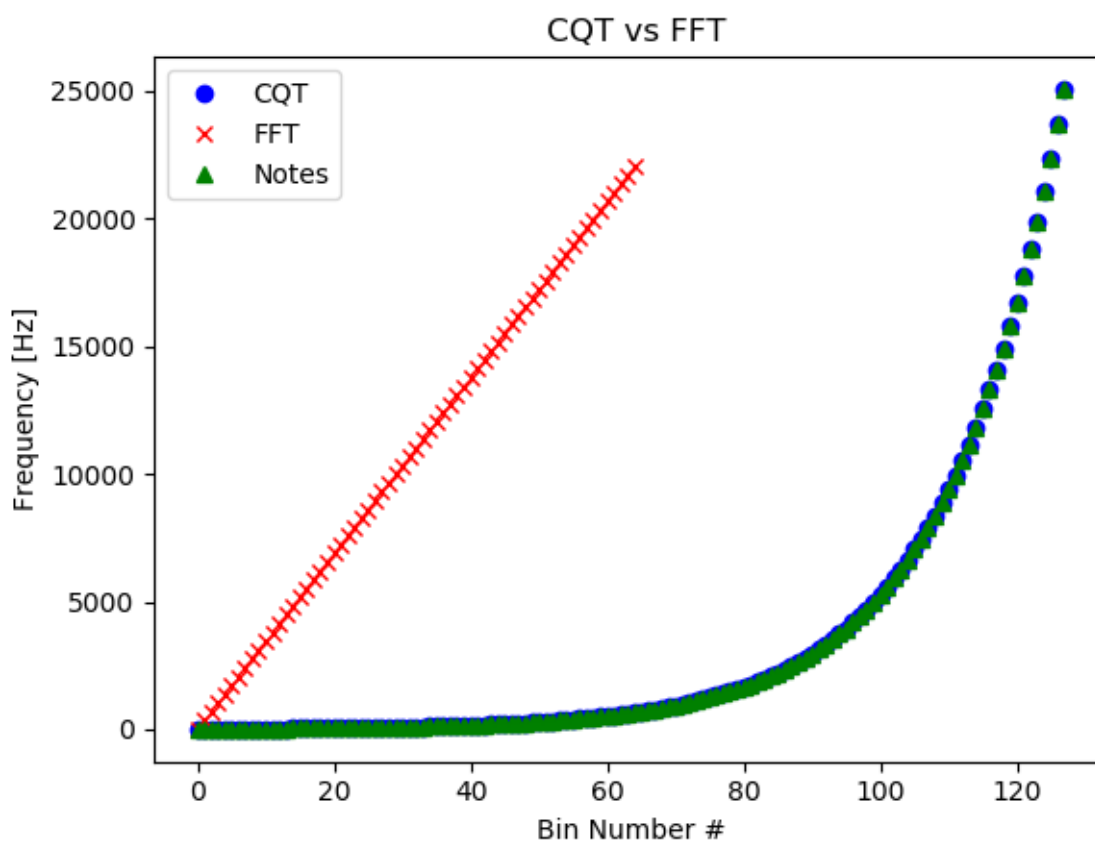


Figura 1: CQT comparat cu FFT

În urma transformării Constant Q, se obțin datele necesare, estimarea frecvențelor fiecărei note din înregistrare.

Încărcarea datelor

Pentru încărcarea datelor, librăria *librosa* propune în pachetul *librosa.core* metoda `load()`:

```
librosa.core.load(path, sr=22050, mono=True, offset=0.0, duration=None, dtype=<class 'numpy.float32'>, res_type='kaiser_best')
```

Ce prezintă parametrii:

- `path` – șir de caractere ce reprezintă calea către un fișier wav
- `sr` – rata de eșantionare („None” este folosit pentru a folosi rata de eșantionare nativă fișierului)
- `mono` – valoare booleană, dacă este True convertește semnalul primit în mono
- `offset` – reprezintă numărul de secunde după care să înceapă citirea
- `duration` – reprezintă durația maximă citită în secunde
- `dtype` – tipul de date (valori) pentru `y`
- `res_type` – tipul de eșantionare

Și returnează un tuplu format din `y`, lista cu date audio și `sr`, rata de eșantionare.

Constant Q Transform

Pentru implementarea CQT, librăria *librosa* pune la dispoziție în pachetul *librosa.core* metodele *cqt*, *magphase* și *amplitude_to_db*:

```
librosa.core.cqt(y, sr=22050, hop_length=512, fmin=None, n_bins=84, bins_per_octave=12,
tuning=0.0, filter_scale=1, norm=1, sparsity=0.01, window='hann', scale=True,
pad_mode='reflect', res_type=None)
```

Această metodă are următorii parametri:

- `y` – vector numpy ce reprezintă semnalul audio primit
- `sr` – rata de eșantionare a vectorului `y`
- `hop_length` – numărul de eșantioane între coloane succesive CQT
- `fmin` – frecvența minimă (valoarea implicită este Do1 \approx 32.70 Hz)
- `n_bins` – numărul de eșantioane de frecvență începând cu `fmin`
- `bins_per_octave` – numărul de eșantioane de frecvență pentru fiecare octavă
- `tuning` – reglează frecvența de bază
- `filter_scale` – factor de scalare al filtrului
- `norm` – tipul de normă utilizat pentru normalizarea funcției
- `sparsity` – gradul de rarefiere CQT

- `window` – specificație pentru filtru
- `scale` – valoare booleană ce este echivalentă cu `norm='ortho'` atunci când ia valoarea „True” și cu `norm=None` atunci când ia valoarea „False”
- `pad_mode` – modul de padding
- `res_type` – tipul de eșantionare

Și returnează o listă de numere reale ce reprezintă rezultatul transformării.

`librosa.core.magphase(D, power=1)`

Această metodă separă o spectrogramă cu valori complexe în componentele sale de magnitudine (S) și fază (P), astfel încat $D = S * P$ și prezintă următorii parametri:

- `D` – spectrogramă de valori complexe de tip `numpy.ndarray`
- `power` – valoare reală > 0 , reprezintă exponentul pentru spectrograma de magnitudine (exemplu: 1 pentru energie, 2 pentru putere, etc.)

Metoda *magphase* returnează componentele S și P în format `np.ndarray`

`librosa.core.amplitude_to_db(S, ref=1.0, amin=1e-05, top_db=80.0)`

Metoda *amplitude_to_db* convertește o spectrogramă de amplitudine într-una scalată de dB și primește patru parametri:

- `S` – spectrograma de amplitudine
- `ref` – poate fi o valoare scalar sau o metodă callback, în cazul unei valori scalare, amplitudinea este scalată relativ la `ref` după formula: $20 * \log_{10}(S / ref)$. Altfel, valoarea de referință este calculată ca $ref(S)$.
- `amin` – pragul minim pentru `S` și `ref`
- `top_db` – setează un prag în output la *top_db* sub valoarea maximă după formula:
 $max(20 * \log_{10}(S)) - top_db$

Estimarea atacului notelor

Pentru aflarea atacului fiecărei note, aceeași librărie propune în modulul *librosa.onset* metodele *onset_strength()* și *onset_detect()*:

`librosa.onset.onset_strength(y=None, sr=22050, S=None, lag=1, max_size=1, ref=None, detrend=False, center=True, feature=None, aggregate=None, centering=None, **kwargs)`

Cu parametrii:

- *y* – vector numpy ce reprezintă semnalul audio primit
- *sr* – rata de eșantionare a vectorului *y*
- *S* – spectrogramă logaritmică precalculată
- *lag* – timpul de întârziere pentru calcul
- *max_size* – mărimea exprimată în eșantioane de frecvență a filtrului local maxim
- *ref* – spectru de referință precalculat, opțional
- *detrend* – valoare booleană, decide dacă se execută filtrarea pentru a elimina componenta DC
- *center* – shiftază funcția de atac cu $n_fft / (2 * hop_length)$ frame-uri
- *feature* – funcție pentru calcularea timpului, folosește *librosa.feature.melspectrogram* implicit
- *aggregate* – funcție de agregare folosită atunci când se combina ‘onsets’ la diferite ‘freq bins’

Returnează un vector ce conține *onset strength envelope*.

`librosa.onset.onset_detect(y=None, sr=22050, onset_envelope=None, hop_length=512, backtrack=False, energy=None, units='frames', **kwargs)`

Cu parametrii:

- *y* – vector numpy ce reprezintă semnalul audio primit
- *sr* – rata de eșantionare a vectorului *y*
- *onset_envelope* – rezultatul funcției *onset_strength()* pentru același *y* și *sr*
- *hop_length* – numărul de eșantioane între coloane succesive CQT
- *units* – unitatea în care sunt returnate rezultatele finale
- *backtrack* – atacurile detectate sunt retrase la cel mai apropiat minim de *energie*

- `energy` – o funcție de energie ce este folosită pentru *backtrack*

Și returnează un vector cu valorile estimative a pozițiilor atacului notelor, în orice unitate este menționată în parametru *units*

Estimarea Tempo-ului

Pentru estimarea tempo-ului înregistrării, în pachetul *librosa.beat* este folosită metoda *beat_track()*:

```
librosa.beat.beat_track(y=None, sr=22050, onset_envelope=None, hop_length=512,
start_bpm=120.0, tightness=100, trim=True, bpm=None, prior=None, units='frames')
```

- `y` – vector numpy ce reprezintă semnalul audio primit
- `sr` – rata de eșantionare a vectorului `y`
- `onset_envelope` – rezultatul funcției *onset_strength()* pentru același `y` și `sr`
- `hop_length` – numărul de eșantioane între valori succesive ale „`onset_envelope`”
- `start_bpm` – punct de start pentru estimarea tempo-ului (măsurat în bătăi pe minut)
- `tightness` – nivel de precizie
- `trim` – elimină elementele cu atac slab din calcul
- `bpm` – opțional, setează tempo-ul la un număr fix și nu mai calculează estimativ
- `prior` – o distribuție apriori peste tempo, dacă este menționat, *start_bpm* va fi ignorat
- `units` – unitatea de măsură pentru bătăi, măsurată în cadre, eșantioane sau timp („frames”, „samples”, „time”)

Această metodă returnează un tuplu format dintr-un număr real *tempo* (estimarea tempo-ului în bpm) și un array *beats* (conține estimarea momentelor bătăilor în unitatea specificată prin parametrul *units*).

Interfața grafică

Interfața grafică a aplicației este implementată folosind librăria standard din Python, TkInter. Aceasta pune la dispoziție o multitudine de widgeturi utile în realizarea unui workflow simplist, user friendly.

Majoritatea obiectelor TkInter permit utilizarea unei palete de parametri opționali ce stilizează obiectul în cauză, spre exemplu:

- bd – grosimea marginilor
- bg – culoarea backgroundului
- fg – culoarea textului
- font – fontul textului
- height – înălțimea obiectului
- width – lățimea obiectului
- anchor – afectează punctul la care se raportează funcțiile place, pack sau grid atunci când așează obiectul în pagină
- etc.

Obiectele utilizate în aplicația de față sunt:

- Frame – folosit pentru gruparea și așezarea obiectelor mai ușor în interfață
- Button – adaugă butoane aplicației pentru diferite funcționalități
- Text – afișare de text pe mai multe linii în interfață
- Label – afișare de text sau imagini
- Canvas – ajută la implementarea graficelor matplotlib în interfața Tk
- Toplevel – deschide o nouă fereastră Tk pornind de la un părinte

Fiecare dintre acestea este creat cu ajutorul unui constructor cu parametri similari *master*, reprezentând fereastra părinte și *options*, o listă de opțiuni pentru stilizarea obiectului. Singurul diferit fiind Toplevel, care primește doar lista de opțiuni.

Opțiunile comune sunt:

- bg – culoarea backgroundului
- bd – grosimea marginilor
- height – înălțimea
- width – lățimea
- relief – poate fi unul dintre „sunken”, „raised”, „groove”, „ridge”, „flat” și reprezintă gradul de reliefare

În particular, fiecare obiect are și opțiuni proprii, ce pot fi folosite doar pentru anumite funcționalități, spre exemplu:

- Frame: cursor, highlightbackground, highlightcolor, highlightthickness
- Canvas: cursor, highlightcolor, scrollregion, xscrollincrement, xscrollcommand, yscrollincrement, yscrollcommand
- Label: anchor, bitmap, cursor, font, fg, image, justify, padx, pady, text, textvariable, underline, wraplength
- Button: activebackground, activeforeground, command, fg, font, highlightcolor, image, justify, padx, pady, state, text, underline, wraplength
- Text: cursor, exportselection, font, fg, highlightbackground, highlightcolor, highlightthickness, insertbackground, insertborderwidth, insertofftime, insertontime, insertwidth, padx, pady, selectbackground, spacing1, spacing2, spacing3, state, tabs, wrap, xscrollcommand, yscrollcommand
- Toplevel: cursor, class_, font, fg

Toate obiectele Tkinter beneficiază de un modul de geometrie prin care pot fi așezate și dimensionate în diferite moduri, spre exemplu:

- Metoda *pack()* – organizează widgeturile în blocuri înainte să le așeze în widgetul părinte, are următoarele opțiuni:
 - expand – valoare booleană, dacă este setată pe „True” face widgetul apelant să ocupe orice spațiu nefolosit în părinte
 - fill – determină dacă widgetul ocupă spațiu în plus sau păstrează dimensiuni minime. Poate fi: NONE (implicit), X (umple pe orizontală), Y (umple pe verticală), sau BOTH (umple în ambele direcții)
 - side – determină partea widgetului părinte în care obiectul este așezat. Poate fi: TOP (implicit), BOTTOM, LEFT, sau RIGHT
- Metoda *grid()* – organizează widgeturile într-o structură tip tabel în widgetul părinte, prezintă parametrii:
 - column – coloana în care este așezat widgetul, implicit 0 (coloana cea mai din stânga)
 - columnspan – numărul de coloane pe care widgetul le ocupă, implicit 1
 - ipadx – numărul de pixeli pentru padding orizontal în interiorul marginilor widgetului
 - ipady - numărul de pixeli pentru padding vertical în interiorul marginilor widgetului

- padx - numărul de pixeli pentru padding orizontal în exteriorul marginilor widgetului
- pady - numărul de pixeli pentru padding vertical în exteriorul marginilor widgetului
- row – rândul pe care este așezat widgetul, implicit este primul rând liber
- rowspan – numărul de rânduri pe care widgetul le ocupă, implicit 1
- sticky – dacă widgetul este mai mic decât celula sa din grid, acest parametru setează poziția widgetului în celulă. Implicit, widgetul va fi centrat, însă, acest parametru poate avea și valorile punctelor cardinale (N, S, E, W, NW, etc.)
- Metoda *place()* – așează widgeturile într-o poziție bazată pe coordonate în widgetul părinte, acceptă următorii parametri:
 - anchor – poziția în widgetul părinte față de care ceilalți parametri își calculează așezarea. Poate fi orice punct cardinal și are valoarea implicită NW (colțul din stânga sus al părintelui)
 - bordermode – poate fi *INSIDE* (implicit) indicând faptul că celelalte opțiuni se referă la interiorul părintelui (ignorând marginile) sau *OUTSIDE* altfel
 - height – înălțimea în pixeli
 - width – lățimea în pixeli
 - relheight – înălțimea ca un număr real între 0.0 și 1.0, reprezentând o fracțiune fixă din înălțimea părintelui
 - relwidth - lățimea ca un număr real între 0.0 și 1.0, reprezentând o fracțiune fixă din lățimea părintelui
 - x – poziționare orizontală în pixeli
 - y – poziționare verticală în pixeli
 - relx – poziționarea orizontală ca un număr real între 0.0 și 1.0, reprezentând o fracțiune fixă din lățimea părintelui
 - rely – poziționarea verticală ca un număr real între 0.0 și 1.0, reprezentând o fracțiune fixă din înălțimea părintelui
- Metoda *geometry()* (specifică ferestrelor) – poziționează și redimensionează fereastra după un șir de caractere de forma 'lățime x înălțime + poz_x + poz_y', dat ca parametru

Import și Export de fișiere

Tot din librăria Tkinter am folosit modulul `filedialog` pentru salvarea și deschiderea de fișiere externe aplicației (de exemplu, pentru importat fișiere wav sau salvat fișiere midi).

Pentru import este utilă metoda `askopenfilename()` ce deschide o fereastră sub formă de explorer ce necesită alegerea unui fișier existent și returnează un șir de caractere ce reprezintă path-ul către fișierul ales. Această funcție primește ca parametrii:

- `Initialdir` – directorul din care se începe căutarea fișierului
- `Title` – titlul ferestrei create
- `Filetypes` – listă cu tipurile de fișiere permise

Pentru salvarea unui fișier am folosit metoda `asksavefilename()` ce deschide o fereastră similară celei anterioare, doar că, de data aceasta este necesară introducerea unui nume pentru fișierul nou creat și alegerea unui director în care acesta să fie salvat. Metoda primește aceeași parametri ca cea descrisă mai sus.

Reprezentări Grafice

Librăria Matplotlib este folosită pentru realizarea graficelor. Pentru afișarea acestora în GUI-ul Tkinter, trebuie creat un obiect `matplotlib.figure.Figure` căruia îi sunt adăugate grafice prin metoda `add_subplot()` ce sunt desenate apoi într-un obiect `FigureCanvasTkAgg` din modulul `matplotlib.backends.backend_tkagg`. `FigureCanvasTkAgg` este obiectul `matplotlib` ce face posibilă comunicarea cu `tkinter` și așezarea graficelor desenate în figură pe widgeturi `tkinter`. În acest sens, am utilizat metodele:

- `matplotlib.figure.Figure(figsize=None, dpi=None, facecolor=None, edgecolor=None, linewidth=0.0, frameon=None, subplotpars=None, tight_layout=None, constrained_layout=None)` – constructor implicit pentru o figură matplotlib ce primește parametrii:
 - `figsize` – tuplu de două numere reale (implicit: `rcParams[„figure.figsize”]` = [6.4, 4.8]) ce reglează dimensiunea figurii (lățime, înălțime) în inch
 - `dpi` – număr real (implicit: `rcParams[„figure.dpi”]` = 100.0) ce reprezintă numărul de puncte per inch (rezoluția)

- `facecolor` – culoarea fundalului figurii (implicit alb)
- `edgecolor` – culoarea marginilor figurii (implicit alb)
- `linewidth` – The linewidth of the frame
- `frameon` – valoare booleană renunță la desenarea patchului de fundal al figurii dacă primește valoarea „False”, implicit primește valoarea „True”
- `subplotpars` – parametrii subgraficelor, dacă nu este menționat, sunt folosiți parametrii de bază (implicit: `rcParams[„figure.subplot.*”]` = None)
- `tight_layout` – valoare booleană sau dicționar. Dacă este „False” folosește *subplotpars*, altfel ajustează parametrii subgraficelor folosind `tight_layout` cu padding implicit. Dacă inputul primit este un dicționar care conține cheile *pad*, *w_pad*, *h_pad* și *rect*, atunci paddingurile implicite din `tight_layout` vor fi suprascrise
- `constrained_layout` – asemănător `tight_layout`, dar construit pentru a fi mai flexibil, implicit fals
- `Figure.add_subplot(self, *args, **kwargs)` – adaugă un obiect *Axes* figurii ca parte a unui aranjament de subgrafice. Poate fi apelată în patru moduri:
 - `add_subplot(nrows, ncols, index, **kwargs)`
 - date trei numere naturale în ordinea număr_rânduri, număr_coloane, index, subgraficul va lua poziția „index” pe un grid cu *nrows* și *ncols* rânduri și coloane. Indexul începe de la 1 în colțul cel mai din stânga sus și este incrementat înspre dreapta
 - `add_subplot(pos, **kwargs)`
 - *pos* reprezintă un număr de trei cifre în care prima cifră reprezintă numărul de rânduri, a doua cifră reprezintă numărul de coloane iar a treia cifră indexul subgraficului. (astfel `add_subplots(2, 3, 4)` este același lucru cu `add_subplots(234)`)
 - `add_subplot(ax)`
 - în circumstanțe rare, o instanță a unui obiect *axes* deja creată în figura prezentă, dar care nu face parte din lista de axe ale figurii poate fi dată ca parametru
 - `add_subplot()`
 - dacă niciun argument nu este prezent, se folosește implicit (1, 1, 1)
- `FigureCanvasTkAgg(figure, master=None, resize_callback=None)`
 - *figure* – figura ce trebuie adăugată în Tkinter

- master – widgetul căruia îi este adăugată figura
- prezintă metoda *draw()* ce desenează figura „figure” în widgetul „master”
- este afișat în Tkinter prin metoda *get_tk_widget().pack()*

Input/Output MIDI

Aplicația poate primi input MIDI printr-un controller sau o clapă midi, comunicarea cu acestea realizându-se cu ajutorul modulului *midi* al librăriei *pygame*. Pentru a folosi acest modul, el trebuie inițializat. Acest lucru se realizează prin apelarea metodei *init()*:

- `pygame.init()`
- `pygame.midi.init()`

Semnalele MIDI sunt primite sub formă de mesaje de tipul:

`[[midi_event, midi_note, midi_velocity, 0], midi_time]`.

Acestea sunt citite cu pachetul *pygame.midi.Input*, prin metodele *poll()* și *read()*

- `pygame.midi.Input.poll()` – nu primește niciun parametru și returnează „True” dacă există date primite în bufferul MIDI și „False” altfel
- `pygame.midi.Input.read(num_events)` – primește un singur parametru ce reprezintă numărul de evenimente MIDI ce trebuie citite din buffer și returnează evenimentele citite

Fișierele MIDI sunt salvate cu ajutorul librăriei *mido* deoarece aceasta librărie deține un format similar de mesaje MIDI cu cel *pygame.midi* făcând astfel conversia foarte simplă. Este creat un obiect *mido.MidiFile* și unul *mido.MidiTrack* care este atașat primului. Toate mesajele MIDI primite ulterior sunt stocate în lista *MidiTrack* prin metoda implicită listelor din python *append* ce primește în acest caz instanțe *mido.Message* care să fie adăugate în *MidiTrack* și implicit în *MidiFile*. În final, fișierul midi poate fi salvat prin metoda *MidiFile.save()*.

- `mido.MidiFile(path)` – constructor pentru crearea unui obiect *mido.MidiFile*. Dacă parametrul „path” nu este precizat, obiectul oferă posibilitatea de a crea un nou fișier MIDI, altfel citește informația din fișierul de la pathul dat ca parametru.

- `mido.MidiTrack()` – constructor pentru crearea unui obiect `mido.MidiTrack`. Aceasta este o subclasă a listei, deci poate folosi toate metodele uzuale precum `append()`.
- `mido.Message(type, program, note, velocity, time)` – constructor pentru un mesaj MIDI ce poate avea tipul „program_change”, „note_on”, „note_off”, dat de parametrul *type*. Parametrul *program* reprezintă instrumentul de redare al clipului midi și este de obicei folosit într-un singur mesaj, la început, în orice fișier MIDI, împreună cu tipul mesajului „program_change”. Dacă tipul mesajului are legătură cu o notă, atunci parametrul *note* reprezintă numărul notei în standard MIDI, *velocity* reprezintă volumul de redare a notei. La parametrul *time* intervine diferența între mesajele *mido* și *pygame.midi*. Dacă în mesajele *pygame.midi* timpul era exprimat în milisecunde trecute de la startul clipului, în mesajele *mido*, timpul este exprimat în milisecunde trecute de la ultimul eveniment MIDI primit, știind asta însă, este ușor să convertim timpii între ei.
- `mido.MidiFile.save(path)` – salvează fișierul midi creat la calea respectivă

Din aceeași librărie, modulul *pygame.mixer* poate fi folosit pentru redarea unor sunete, acesta trebuie de asemenea inițializat printr-o metodă *init()*:

- `pygame.mixer.init()`

Se creează un obiect de tipul *pygame.mixer.Sound* printr-un constructor ce primește ca parametru calea către un fișier Wav și aplică asupra sa metodele *play()* și *fadeout()*

- `pygame.mixer.Sound.play(loops=0, maxtime=0, fade_ms=0)` – pornește redarea fișierului. Parametrul *loops* reprezintă numărul de dați în care sunetul va fi repetat după prima redare (ex: dacă *loops*=3, sunetul va fi auzit o dată si repetat de încă trei ori). Parametrul *maxtime* este folosit pentru limitarea duratei redării la un anumit număr de milisecunde. Parametrul *fade_ms* setează numărul de milisecunde în care sunetul crește în volum, pornind de la zero (ex: dacă *fade_ms*=0, sunetul va începe cu volum maxim, altfel va începe cu volum zero și va urca treptat un număr de milisecunde până va atinge maximul).
- `Pygame.mixer.Sound.fadeout(time)` – oprește treptat redarea prin scăderi succesive de volum timp de un număr de milisecunde

Același scop poate fi atins și prin intermediul librăriei *simpleaudio*, cu o abordare similară, dar rezultate și timp de răspuns mai bune în cazul redării mai multor sunete simultan

sau în succesiune rapidă (pian). Se creează un obiect de tipul *simpleaudio.WaveObject* în care este încărcat un fișier prin metoda *from_wave_file()* ce primește ca parametru calea către un fișier Wav. Sunetul este apoi redat prin metoda *simpleaudio.WaveObject.play()*, ce se comporta asemănător metodei *play()* descrise anterior.

Interacțiunea clapelor cu tastatura este realizată prin metoda *tk.bind()*. Aceasta primește doi parametri, tipul evenimentului și funcția ce trebuie executată atunci când se întâlnește evenimentul respectiv.

Înregistrarea inputului primit în clapele aplicației se face în fișiere text, unul pentru tastatură și unul pentru midi extern.

Înregistrarea audio externă

Pe lângă inputul prin clape sau fișiere, aplicația oferă și posibilitatea înregistrării unui clip audio prin microfon. Acest lucru este realizat de librăriile *pyaudio* și *wave*. Pentru înregistrare este creat un obiect *pyaudio.Pyaudio* asupra căruia este apelată metoda *open()* pentru a deschide un flux ce primește date de la microfon. Această metodă are următorii parametri:

- *PA_manager* – referință la instanța *pyaudio* de bază
- *rate* – rata de eșantionare
- *channels* – numărul de canale (mono sau stereo)
- *format* - mărimea și formatul unui eșantion
- *input* – specifică dacă fluxul este pentru input (implicit fals)
- *output* – specifică dacă fluxul este pentru output (implicit fals)
- *input_device_index* – specifică ce dispozitiv de intrare să fie folosit
- *output_device_index* – specifică ce dispozitiv de ieșire să fie folosit
- *frames_per_buffer* – numărul de cadre per buffer
- *start* – pornește fluxul imediat (implicit adevărat)
- *input_host_api_specific_stream_info* – specifică informația fluxului unui host API pentru intrare
- *output_host_api_specific_stream_info* – specifică informația fluxului unui host API pentru ieșire

- `stream_callback` – reprezintă o funcție callback pentru operații nonblocante. Pentru a folosi astfel de operații, funcția callback trebuie să respecte următoarea structură: `callback(in_data, frame_count, time_info, status_flag)` și trebuie să returneze un tuplu `(out_data, flag)`

Înregistrarea se face prin apelarea metodei `read()` a fluxului generat prin `open()` cu parametrii:

- `num_frames` – numărul de cadre de citit din flux
- `exception_on_overflow` – valoare booleană, implicit adevărată, specifică dacă o eroare IO ce apare în interiorul funcției datorată overflowului de date trebuie tratată sau ignorată

Metoda `read()` returnează un string ce reprezintă datele ce au fost citite.

Fișiere MXL

Lucrul cu fișierele MXL și toate obiectele legate de acestea sunt implementate cu ajutorul librăriei *music21* prin modulele:

- `music21.note.Note` – obiect ce conține toate caracteristicile unei note: nume, octavă, pitch, frecvență
- `music21.rest.Rest` – obiect ce conține toate caracteristicile unei pauze
- `music21.stream.Stream` – listă de note corelate cu timpul lor de atac, acesta este cel mai folosit obiect astfel încât reprezintă colecția de note similară partiturii în sine
- `music21.duration.Duration` - obiect ce caracterizează durata unei note sau a unei pauze
- `music21.converter` – folosit pentru încărcarea de fișiere mxl externe și convertirea lor la Stream.
- `music21.key` – pentru scrierea cheii într-o partitură
- `music21.midi.translate` – modul pentru conversia midi – partitură
 - `streamToMidiFile(inputM21)` – convertește un obiect Stream într-unul MidiFile
 - `midiFileToStream(mf, inputM21=None, quantizePost=True, **keywords)` – convertește un obiect MidiFile într-unul Stream

Fișierele MXL sunt deschise cu ajutorul unui software extern ce este necesar pentru rularea corectă a aplicației, numit MuseScore (<https://musescore.com/dashboard>) acest lucru este realizat intern de librăria *music21*.

Fișiere WAV

Pentru manevrarea fișierelor în format WAV am folosit librăria Wave datorită suportului pe care aceasta îl oferă pentru fișiere de tip mono și stereo.

Funcția *wave.open(file, mode)* deschide fișierul dat de parametrul „file”, dacă acesta este un string, și creează un obiect *wave_read* sau *wave_write* în funcție de parametrul *mode* care poate fi „rb” (read only) sau „wb” (write only). Dacă parametrul „file” este un obiect de tip fișier, modul folosit va fi *file.mode*

- Obiectele *wave_read* au următoarele metode:
 - *close()* – închide fluxul audio
 - *getnchannels()* – returnează numărul de canale audio (1 pentru mono, 2 pentru stereo)
 - *getsampwidth()* – returnează lățimea unui eșantion în bytes
 - *getframerate()* – returnează rata de eșantionare a cadrelor
 - *getnframes()* – returnează numărul de cadre audio
 - *getcomptype()* – returnează tipul de compresie („NONE” este singurul suportat)
 - *getcompname()* – versiune în limbaj natural a metodei *getcomptype* (returnează „not compressed”)
 - *getparams()* – returnează un tuplu (*nchannels*, *sampwidth*, *framerate*, *nframes*, *comptype*, *compname*) echivalent outputului tuturor metodelor *get()*
 - *readframes(n)* – citește și returnează cel mult *n* cadre audio în bytes
 - *rewind()* – duce pointerul fișierului la începutul fluxului audio
 - *getmarkers()* – returnează None
 - *getmark(id)* – ridică o eroare
 - *setpos(pos)* – setează pointerul fișierului la poziția specificată prin *pos*
 - *tell()* – returnează poziția curentă a pointerului în fișier
- Obiectele *wave_write* au următoarele metode:
 - *close()* – închide fluxul audio și verifică dacă *nframes* corespunde cu numărul de cadre scrise
 - *setnchannels(n)* – setează numărul de canale
 - *setsampwidth(n)* – setează lățimea unui eșantion la *n* bytes
 - *setframerate(n)* – setează rata de eșantionare a cadrelor

- `setnframes(n)` – setează numărul de cadre la `n`, însă va fi schimbat dacă numărul de cadre citite este diferit
- `setcomptype(n)` – setează tipul de compresie și descrierea, momentan doar „NONE” este suportat, însemnând nicio compresie
- `setparams(tuple)` – setează toți parametri printr-un tuplu de forma (`nchannels`, `sampwidth`, `framerate`, `nframes`, `comptype`, `compname`)
- `tell()` – returnează poziția pointerului în fișier
- `writeframesraw(data)` – scrie cadre audio fără să corecteze `nframes`
- `writeframes(data)` – scrie cadre audio și se asigură ca `nframes` este corect

Redarea sunetelor clapelor și a înregistrării audio se face într-un fir de execuție diferit prin metoda `start_new_thread()` din modulul `_thread`. Această metodă primește ca parametru o funcție pentru care creează un nou fir de execuție.

Capitolul III: Descrierea aplicației Piano Follower

În implementare vor fi folosite librăriile și metodele amintite în capitolul anterior împreună cu limbajul Python, după următoarea diagramă de clase:

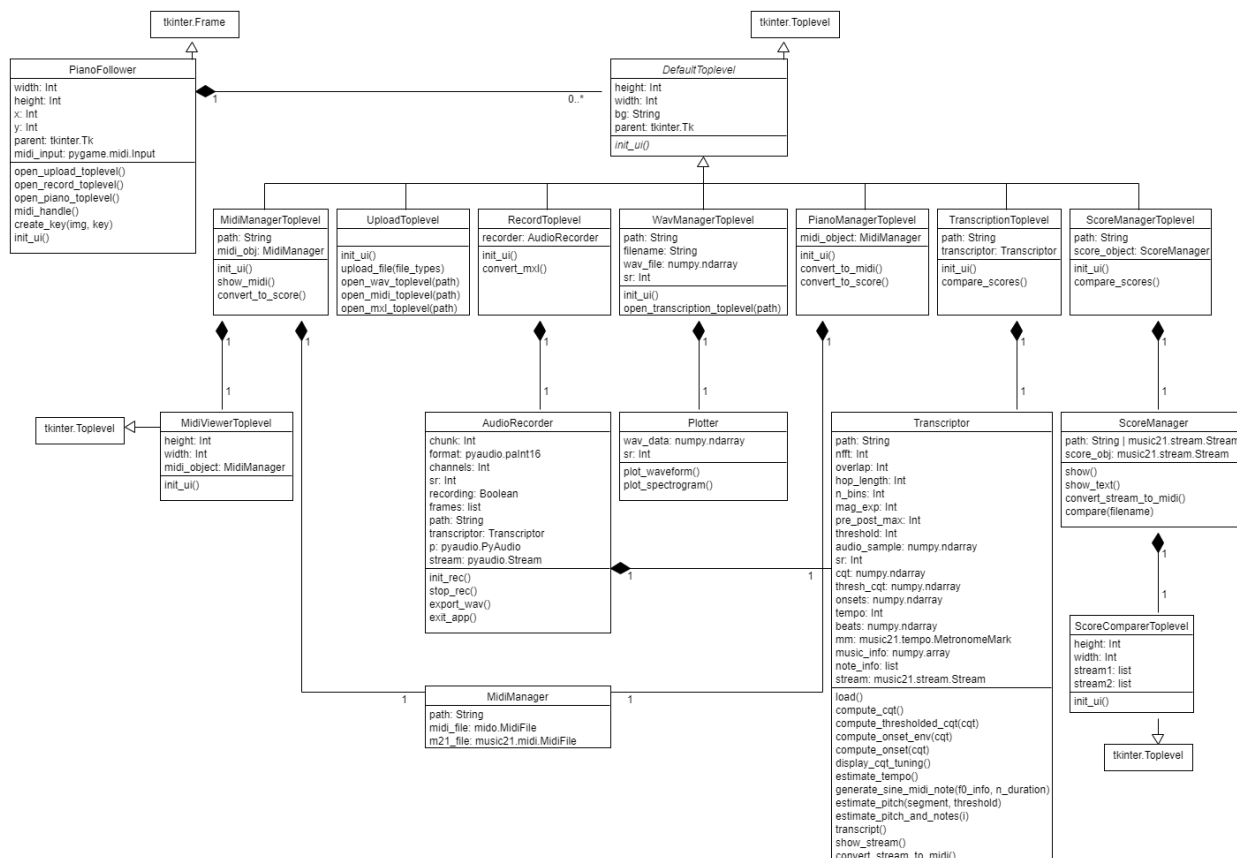


Figura 2: Arhitectura aplicației PianoFollower

După cum se poate observa în această figură, aplicația *PianoFollower* prezintă următoarele funcționalități:

- import de fișiere WAV, MIDI, MXL
- conversia de la WAV la MIDI sau MXL
- conversia de la MIDI la MXL și viceversa
- înregistrarea unui clip audio WAV prin microfon și supunerea acestuia la operațiile de conversie
- compararea a doua fișiere MXL (partituri)
- afișarea mesajelor MIDI aflate în interiorul unui fișier MIDI

Interfața principală

Fișierul „interface.py” reprezintă nucleul de pornire al aplicației. Acesta prezintă câteva instrucțiuni de inițializare, o multitudine de funcții utile interfeței și aplicației în general și clasa *PianoFollower*. Funcțiile implementate aici sunt:

- `increment_octave` – nu primește niciun parametru și incrementează valoarea variabilei globale *octave* ce reține valoarea octavei pentru clape
- `decrement_octave` – decrementează valoarea aceleiași variabile *octave*
- `play_note` – primește doi parametri *note* și *mode* și redă sunetul notei respective parametrului *note* folosind librăria respectivă parametrului *mode* (poate fi „sa” pentru *simpleaudio* sau „pg” pentru *pygame*)
- `play_midi` – comportament identic cu `play_note`, doar că nu adaugă octava în funcție de variabila globală *octave* deoarece notele midi vin cu tot cu octavă
- `label_press` – folosită pentru schimbarea imaginilor clapelor într-una mai închisă la culoare semnificând că sunt apăstate
- `label_release` – comportament asemănător `label_press`, dar schimbă imaginile înapoi în cele originale
- `find_label` – primește doi parametri *name* și *keys* și returnează valoarea din dicționarul *keys* a notei cu numele *name*
- `find_key` – primește un parametru *note* și returnează litera de pe tastatură ce reprezintă nota respectivă, printr-o convenție
- `find_midi_by_key` – primește un singur parametru *note* și returnează numărul midi corespunzător după următorul tabel

MIDI number		Note name	Keyboard	Frequency Hz		Period ms	
21	22	A0		27.500		36.36	
23		B0		30.868	29.135	32.40	34.32
24	25	C1		32.703		30.58	
26	27	D1		36.708	34.648	27.24	28.86
28		E1		41.203	38.891	24.27	25.71
29	30	F1		43.654		22.91	
31	32	G1		48.999	46.249	20.41	21.62
33	34	A1		55.000	51.913	18.18	19.26
35		B1		61.735	58.270	16.20	17.16
36	37	C2		65.406		15.29	
38	39	D2		73.416	69.296	13.62	14.29
40		E2		82.407	77.782	12.13	12.86
41	42	F2		87.307		11.45	
43	44	G2		97.999	92.499	10.20	10.81
45	46	A2		110.00	103.83	9.091	9.631
47		B2		123.47	116.54	8.099	8.581
48	49	C3		130.81		7.645	
50	51	D3		146.83	138.59	6.811	7.216
52		E3		164.81	155.56	6.068	6.428
53	54	F3		174.61		5.727	
55	56	G3		196.00	185.00	5.102	5.405
57	58	A3		220.00	207.65	4.545	4.816
59		B3		246.94	233.08	4.050	4.290
60	61	C4		261.63		3.822	
62	63	D4		293.67	277.18	3.405	3.608
64		E4		329.63	311.13	3.034	3.214
65	66	F4		349.23		2.863	
67	68	G4		392.00	369.99	2.551	2.703
69	70	A4		440.00	415.30	2.273	2.408
71		B4		493.88	466.16	2.025	2.145
72	73	C5		523.25		1.910	
74	75	D5		587.33	554.37	1.703	1.804
76		E5		659.26	622.25	1.517	1.607
77	78	F5		698.46		1.432	
79	80	G5		783.99	739.99	1.276	1.351
81	82	A5		880.00	830.61	1.136	1.204
83		B5		987.77	932.33	1.012	1.073
84	85	C6		1046.5		0.9556	
86	87	D6		1174.7	1108.7	0.8513	0.9020
88		E6		1318.5	1244.5	0.7584	0.8034
89	90	F6		1396.9		0.7159	
91	92	G6		1568.0	1480.0	0.6378	0.6757
93	94	A6		1760.0	1661.2	0.5682	0.6020
95		B6		1975.5	1864.7	0.5062	0.5363
96	97	C7		2093.0		0.4778	
98	99	D7		2349.3	2217.5	0.4257	0.4510
100		E7		2637.0	2489.0	0.3792	0.4018
101	102	F7		2793.0		0.3580	
103	104	G7		3136.0	2960.0	0.3189	0.3378
105	106	A7		3520.0	3322.4	0.2841	0.3010
107		B7		3951.1	3729.3	0.2531	0.2681
108		C8	J. Wolfe, UNSW	4186.0		0.2389	

Figura 3: Standard de notare MIDI - frecvență

- `midi_press` – primește doi parametrii *event* și *keys*, unde *event* reprezintă un mesaj MIDI (în acest caz de tipul „note-on”), iar *keys* este dicționarul de clape al aplicației, folosit pentru schimbarea imaginii notei apăsată pe clapă. Redă nota din parametrul *event* și de asemenea înregistrează evenimentul într-un fișier text dacă înregistrarea este pornită din aplicație
- `midi_release` – comportament similar `midi_press`, doar că înregistrează evenimentele MIDI „note-off” (atunci când clapa apăsată este eliberată)

- `key_press` – primește un singur parametru *event* ce reprezintă un eveniment primit la tastatură (în acest caz apăsarea unei taste), redă nota respectivă și schimbă imaginea, similar `midi_press`
- `key_release` – analog `key_press`, doar că se ocupă de evenimentul de eliberare a tastei apăsate
- `button_press` – primește un singur parametru *event* ce se ocupă de evenimentele primite din clickul mouse-ului, redă sunetul necesar notei apăsate și apelează `label_press(event)` pentru schimbul de imagini
- `record_on_off` – schimbă valoarea variabilelor globale pentru înregistrare `midi_recording` și `recording`
- `record` – primește doi parametri *filename* și *note* și înregistrează nota în fișierul text de la calea *filename*
- `record_midi` – primește doi parametri *filename* și *msg* și scrie mesajul midi *msg* în fișierul text de la calea *filename*
- `play` – primește un singur parametru *filename* și redă înregistrarea din fișierul text *filename*
- `play_back` – apelează funcția *play* într-un nou fir de execuție prin metoda *start_new_thread* din modulul *_thread* amintit în capitolul anterior

În vederea realizării GUI-ului, am folosit o multitudine de funcții puse la dispoziție de librăria *Tkinter* și am implementat clasa *PianoFollower* ca subclasa a obiectului *Tk Frame* ce servește ca interfața principală a aplicației, în care avem la dispoziție 24 clape de pian (două octave întregi) ce răspund la clickul mouse-ului, la input midi (printr-un *midi controller*) și input la tastatură (pentru o tastatură QWERTY tastele de la ,a' la ,j' de pe rândul respectiv activează clapele albe, iar tastele ,w', ,e', ,t', ,y', ,u' activează clapele negre), cinci butoane pentru interacțiunea cu înregistrarea făcută pe aceste clape și anume un buton roșu pentru pornirea și oprirea înregistrării din clape, un buton verde pentru redarea înregistrării, două butoane pentru creșterea și respectiv scăderea octavei și un buton pentru prelucrarea înregistrării. În plus, sunt puse la dispoziție alte trei butoane pentru importarea unui fișier extern (audio, midi sau mxl), înregistrarea unui clip audio extern prin microfon și salvarea fișierului prelucrat (transcris) respectiv.

Pentru interacțiunea cu mesajele MIDI, clasa *PianoFollower* prezintă metoda *midi_handle*, fără parametri, ce verifică dacă există mesaje MIDI în buffer și apoi le citește și apelează funcția *midi_press* sau *midi_release* depinzând de tipul mesajului MIDI.

Pentru așezarea clapelor, clasa *PianoFollower* folosește metoda *create_key* ce primește ca parametru pathul către o imagine (clapă albă sau neagră) și un vector format din valoarea pe axa X a clapei în aplicație și denumirea notei muzicale respective clapei, în notație englezescă (C, D, E, F, G, A, B corespund notelor Do, Re, Mi, Fa, Sol, La, Si) și creează un obiect de tip *tk.Label* căruia îi atribuie imaginea dată ca parametru cu ajutorul unui obiect *tk.PhotoImage* și plasează în aplicație fiecare clapă prin funcția *place* din TkInter (alte metode de a așeza obiecte în aplicație sunt funcțiile *place* și *grid*, menționate în capitolul anterior). În metoda *init_ui* este definit vectorul *keys* ce conține valori utile funcției descrise mai sus. Acesta este parcurs, se verifică dacă clapa este albă sau neagră, se atribuie imaginea respectivă și se așează în aplicație. Ulterior sunt adăugate butoanele din meniul aplicației, așezate într-un *tk.Frame*

Exemplu:

```
menu_frame = tk.Frame(self, bd=3, bg='darkgray', height=50,
width=self.width)
menu_frame.place(x=0, y=self.height - 250)

file_btn = tk.Button(menu_frame, text='Import File', bd=1,
relief='raised',
                        command=lambda: self.open_upload_toplevel())
file_btn.name = 'file_btn'
file_btn.place(x=0, y=0, relwidth=1.0 / 3.0,
relheight=menu_frame.wininfo_height())
```

Marea majoritate a obiectelor *Tkinter* folosite sunt așezate în aplicație cu metoda *place()* descrisă în capitolul anterior deoarece aceasta prezintă, în opinia mea, cea mai precisă și ușor de înțeles și utilizat așezare.

Butoanele ce interacționează cu clapele sunt așezate în laterala acestora. Pentru cel de înregistrare și cel de redare am folosit imagini distinctive, încărcate similar clapelor:

```
img = tk.PhotoImage(file='../pictures/red_button.gif')
record_btn = tk.Label(self, image=img, bd=1)
record_btn.image = img
record_btn.name = 'rec_button'
record_btn.bind('<Button-1>', record_on_off)
record_btn.place(x=350, y=self.height - 200)
```

În schimb, datorită unei limitări a librăriei TkInter, butoanele de tip text nu pot fi dimensionate în pixeli, astfel ele trebuie așezate într-un container separat de tip *tk.Frame*:

```

octave_up_frame = tk.Frame(self, width=25, height=50, bg='darkgray')
octave_up_frame.pack_propagate(0)
octave_up_frame.place(x=350, y=self.height - 100)

img = tk.PhotoImage(file='../pictures/octave_up.gif')
octave_up_btn = tk.Button(octave_up_frame, text="Oct\n+", bd=1,
relief='raised',
                        height=octave_up_frame['height'])
octave_up_btn.image = img
octave_up_btn.name = 'octave_up'
octave_up_btn.bind('<Button-1>', lambda event: increment_octave())
octave_up_btn.pack()

```

Proprietatea *tk.Frame.pack_propagate(0)* specifică obiectului apelant faptul că acesta să nu poată fi redimensionat de copiii săi. Astfel, butonul poate fi așezat în interiorul frame-ului cu funcția *pack()*, iar frame-ul în sine poate fi plasat în aplicație și dimensionat după dorință.

În final, fereastra principală este așezată în centrul ecranului și este setată ca neredimensionabilă.

```

self.parent.title('Piano Follower')
self.parent.resizable(False, False)

sw = self.parent.winfo_screenwidth()
sh = self.parent.winfo_screenheight()
x = (sw - self.width) / 2
y = (sh - self.height) / 2
self.parent.geometry('%dx%d+%d+%d' % (self.width, self.height, x, y))

```

Astfel, se obține interfața principală (Figura 4):

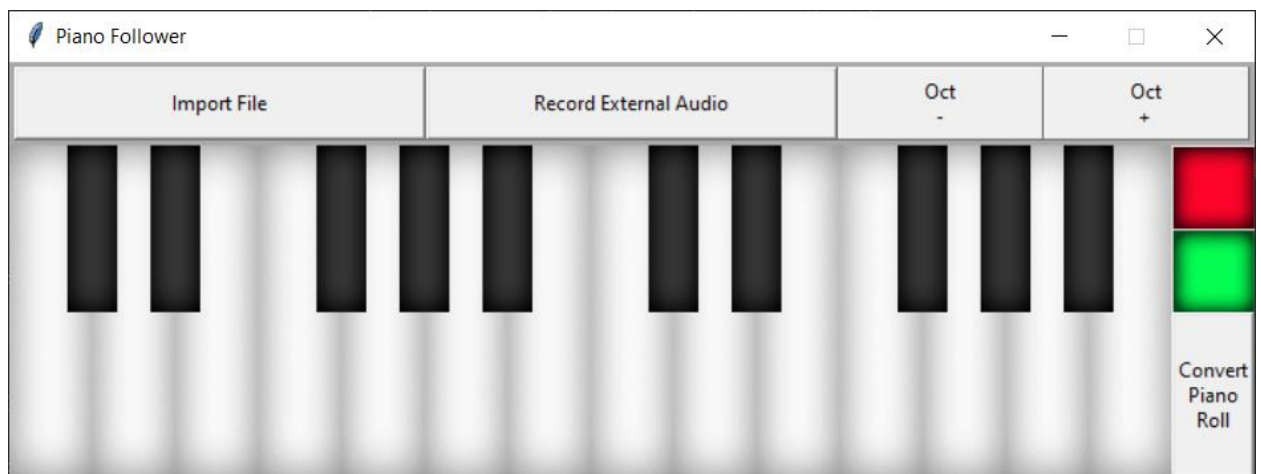


Figura 4: Interfața de pornire

Fiecare buton din meniul principal va deschide o nouă fereastră menită să ghideze userul în realizarea tuturor funcționalităților aplicației cu ușurință, astfel încât, prin parametrul *command* fiecare buton primește o metodă proprie ce creează obiectul necesar. Exemplu:

```
def open_upload_toplevel(self):
    return UploadToplevel(self)

def open_record_toplevel(self):
    return RecorderToplevel(self)

def open_piano_toplevel(self):
    return PianoManagerToplevel(self)
```

Clasa *DefaultToplevel* și moștenitorii ei

Aceste ferestre sunt implementate separat în clasele lor respective, toate pornind de la clasa abstractă *DefaultToplevel* ca subclasă a *tk.Toplevel*, ce prezintă o metodă abstractă *init_ui* care va fi suprascrisă în fiecare clasa moștenitoare și un constructor ce reglează lățimea, înălțimea și poziționarea widgetului față de părinte:

```
class DefaultToplevel(tk.Toplevel):
    def __init__(self, parent):
        self.height = 50
        self.width = parent.width
        self.bg = 'darkgrey'
        tk.Toplevel.__init__(self, height=self.height, width=self.width,
bg=self.bg)
        self.resizable(False, False)
        self.parent = parent
        self.x = int(parent.x)
        self.y = int(parent.y)
        self.geometry('{}x{}+{}+{}'.format(self.width, self.height, self.x,
self.y))
        self.init_ui()

    def init_ui(self):
        pass
```

Clasele moștenitoare *DefaultToplevel* sunt:

- *UploadToplevel*
- *WavManagerToplevel*
- *MidiManagerToplevel*
- *ScoreManagerToplevel*
- *RecorderToplevel*
- *PianoManagerToplevel*

- TranscriptionToplevel

Fiecare dintre acestea este implementată într-un fișier python separat cu același nume, pentru o organizare sugestivă a codului. De asemenea, fiecare constructor ale acestor clase apelează metoda *super().__init__()*, rulând astfel metoda *__init__()* a clasei părinte *DefaultToplevel*. Exemplu:

```
class UploadToplevel(DefaultToplevel):
    def __init__(self, parent):
        super().__init__(parent)
```

Butonul cu textul „Import File” din *Figura 4* apelează metoda *open_upload_toplevel* ce returnează o nouă instanță a unui obiect *UploadToplevel* ce arată astfel:

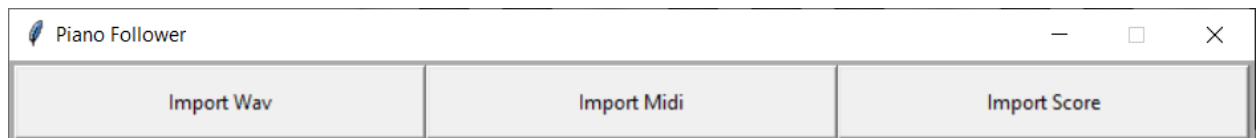


Figura 5: Upload Toplevel

Metoda *init_ui* suprascrisă în clasa *UploadToplevel* creează un obiect *Tk.Frame* în care adaugă trei butoane de dimensiuni egale (o treime din lățimea widgetului), printr-un procedeu identic cu cel al adăugării butoanelor în interfața principală, fiecare având rolul de a încărca în aplicație unul din tipurile de fișiere cu care aceasta lucrează: wav, midi și mxl.

```
menu_frame = tk.Frame(self, bd=3, bg='darkgray', height=self.height,
width=self.width)
menu_frame.place(x=0, y=0)

wav_btn = tk.Button(menu_frame, text='Import Wav', bd=2, relief='raised',
                    command=lambda: self.upload_file(file_types= (('wav
files', ('*.wav', '*.wave')),
                                                                    ('all
files', '*..*'))))

wav_btn.name = 'wav_btn'
wav_btn.place(x=0, y=0, relwidth=1.0 / 3.0,
relheight=menu_frame.winfo_height())
```

De asemenea comanda butoanelor este metoda *upload_file* ce primește un singur parametru *file_types* ce reprezintă un tuplu de valori, similar celui descris pentru funcțiile *tkinter.filedialog.askopenfilename* și *tkinter.filedialog.asksavefilename* descrise în capitolul anterior.


```

def upload_file(self, file_types=('all files', '*..*')):
    file_path = filedialog.askopenfilename(initialdir="/", title="Select
file",
filetypes=file_types)
    extension = file_path.split('/')[-1].split('.')[1]
    if extension == 'wav' or extension == 'wave':
        self.open_wav_toplevel(file_path)
    elif extension == 'mid' or extension == 'midi':
        self.open_midi_toplevel(file_path)
    elif extension == 'mxl':
        self.open_mxl_toplevel(file_path)

```

Acest parametru există tocmai pentru că în corpul metodei este apelată funcția *askopenfilename* pentru a deschide un fișier de tipul *file_types*. Ulterior se verifică tipul fișierului și se deschide un nou *Tk.Toplevel* prin una din celelalte trei metode interne clasei: *open_wav_toplevel*, *open_midi_toplevel*, *open_mxl_toplevel*:

```

def open_wav_toplevel(self, path):
    self.destroy()
    return WavManagerToplevel(self.parent, path)

def open_midi_toplevel(self, path):
    self.destroy()
    return MidiManagerToplevel(self.parent, path)

def open_mxl_toplevel(self, path):
    self.destroy()
    return ScoreManagerToplevel(self.parent, path)

```

Obiectul apelant este închis cu ajutorul metodei *destroy* a obiectelor Tkinter, pentru a reduce numărul de ferestre și a nu aglomera ecranul.

Fiecare din aceste trei obiecte primește în constructor și o cale către un fișier pe lângă parametrul *parent* ce este obligatoriu prezent prin moștenire de la *DefaultToplevel*.

Clasa WavManagerToplevel

Clasa *WavManagerToplevel* se ocupă de fișierele Wav ce sunt încărcate prin metoda *UploadToplevel.upload_file*.

```
class WavManagerToplevel(DefaultToplevel):
    def __init__(self, parent, path):
        self.path = path
        self.filename = self.path.split('/')[-1].split('.')[0]
        self.wav_file, self.sr = librosa.load(self.path, sr=None, mono=True)
        super().__init__(parent)
```

Constructorul încarcă fișierul din calea dată ca parametru cu ajutorul funcției *librosa.load* și salvează rezultatele în membrii *wav_file* și *sr*

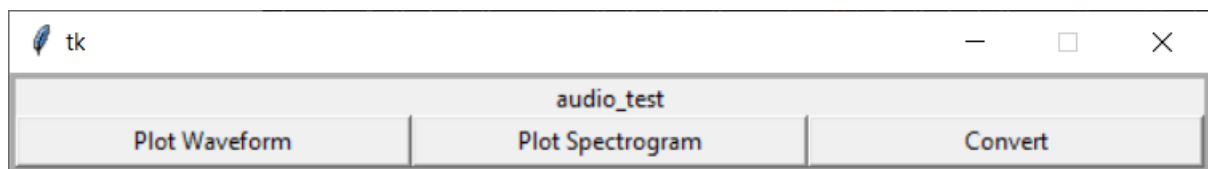


Figura 6: *WavManagerToplevel*

Metoda *init_ui* a acestei clase creează un *Tk.Frame* căruia îi adaugă un *Tk.Label* ce conține numele fișierului deschis, fără extensie dat de membrul *filename* și alte trei butoane, primele două pentru calcularea și afișarea waveformului și a spectrogramei respectiv, iar cel de-al treilea pentru a deschide un meniu ce servește la conversia în midi sau partitură. Exemplu:

```
menu_frame = tk.Frame(self, bd=3, bg='darkgray', height=self.height,
width=self.width)
menu_frame.place(x=0, y=0)
name_label = tk.Label(menu_frame, text=self.filename)
name_label.place(x=0, y=0, relwidth=1.0, relheight=2.0 / 5.0)
waveform_btn = tk.Button(menu_frame, text='Plot Waveform', relief='raised',
                        command=lambda: plot_waveform(self.wav_file,
self.sr))
waveform_btn.place(relx=0.0, rely=2.0 / 5.0, relwidth=1.0 / 3.0,
relheight=3.0 / 5.0)
```

Primele două butoane folosesc drept comandă două funcții statice *plot_waveform* și respectiv *plot_spectrogram*, ambele primind aceeași parametrii:

- *wav_data* – reprezentând y-ul returnat de *librosa.load*

- *sr* – rata de eşantionare

Aceste funcții creează un obiect de tip *Plotter*, clasă ce moștenește *Tk.Toplevel*. Primește ca parametrii în constructor *wav_data* și *sr*, similar celor descriși anterior și pune la dispoziție două metode: *plot_waveform* și *plot_spectrogram* ce folosesc o figură și un canvas *matplotlib* pentru a desena waveformul și respectiv spectrograma inputului.

```
class Plotter(tk.Toplevel):
    def __init__(self, wav_data, sr):
        tk.Toplevel.__init__(self)
        self.wav_data = wav_data
        self.sr = sr
    def plot_waveform(self):
        f = fig.Figure()
        f.add_subplot(111).plot(self.wav_data)
        canvas = FigureCanvasTkAgg(f, master=self)
        canvas.draw()
        canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
    def plot_spectrogram(self):
        f = fig.Figure()
        f.add_subplot(111).specgram(self.wav_data, NFFT=1024, noverlap=900,
Fs=self.sr)
        canvas = FigureCanvasTkAgg(f, master=self)
        canvas.draw()
        canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
```

Așadar, în interfața din *Figura 6* apăsarea butonului „Plot Waveform” va aduce un rezultat de forma:

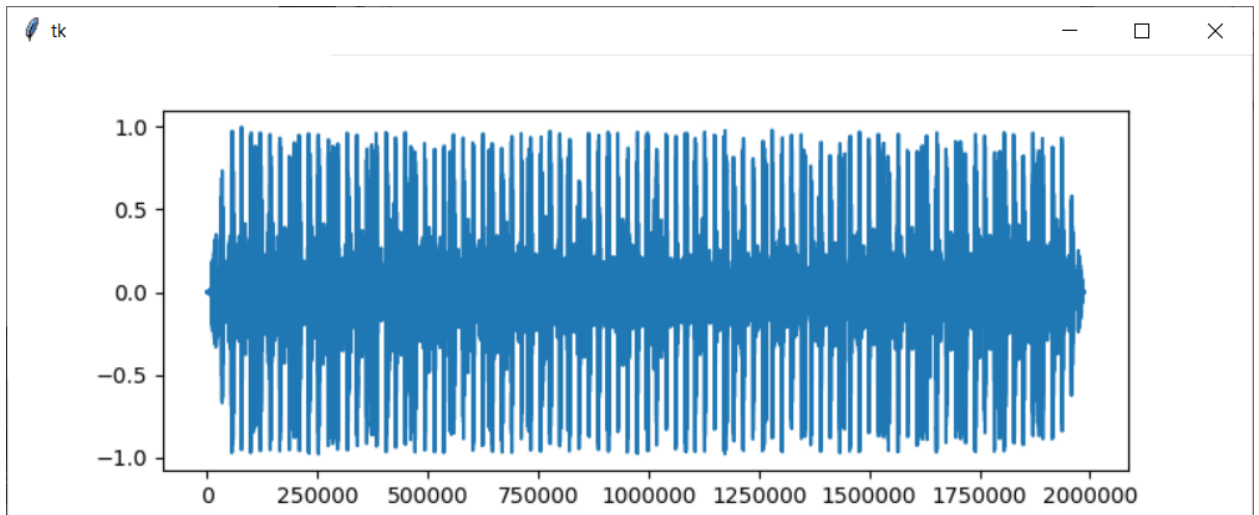


Figura 7: Exemplu de waveform generat de Plotter

În timp ce, apăsarea butonului „Plot Spectrogram” va aduce un rezultat de forma:

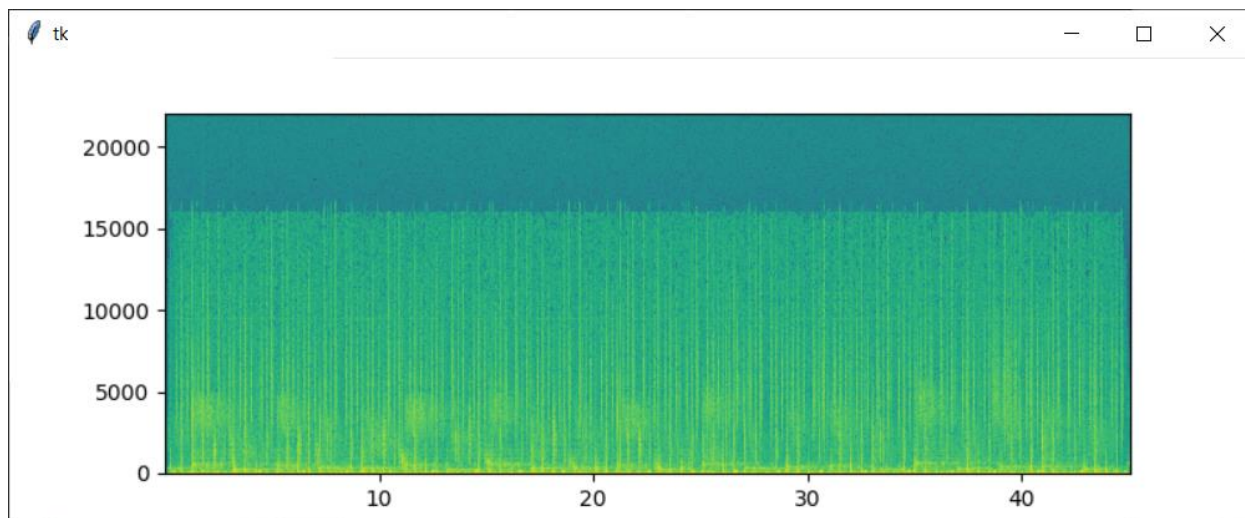


Figura 8: Exemplu de spectrogramă generată de Plotter

Butonul „Convert” apelează metoda *open_transcription_toplevel* ce creează un obiect de tipul *TranscriptionToplevel*, iar obiectul *WavManagerToplevel* apelant este închis prin metoda *destroy*.

```
def open_transcription_toplevel(self, path):
    self.destroy()
    return TranscriptionToplevel(self.parent, path)
```

Obiectul *TranscriptionToplevel* generează un *Transcriptor* responsabil de transcrierea muzicală a fișierului Wav găsit la calea dată ca parametru:

```
class TranscriptionToplevel(DefaultToplevel):
    def __init__(self, parent, path):
        self.path = path
        self.transcriptor = Transcriptor(self.path)
        self.transcriptor.transcript()
        super().__init__(parent)
```

Metoda *init_ui* a clasei *TranscriptionToplevel* generează un *Tk.Frame* căruia îi adaugă trei butoane responsabile pentru afișarea partiturii fișierului din *path*, afișarea rezultatului grafic a funcției *cqt* și compararea partiturii rezultate cu o altă partitură respectiv. Exemplu:

```
def init_ui(self):
    menu_frame = tk.Frame(self, bd=3, bg='darkgray', height=self.height,
width=self.width)
    menu_frame.place(x=0, y=0)

    transcript_btn = tk.Button(menu_frame, text='Show Musical Score',
relief='raised',
                                command=lambda:
self.transcriptor.show_stream())
    transcript_btn.place(relx=0, rely=0, relheight=1.0, relwidth=1.0 / 3.0)
```

Ultima metodă a acestei clase este *compare_scores* ce creează un obiect *ScoreManager*, caută un fișier mxl nou cu ajutorul metodei *tkinter.filedialog.askopenfilename* și compară cele două partituri.



Figura 9: Transcription Toplevel

Transcrierea Muzicală

Fișierul *Transcriptor.py* prezintă funcționalitățile principale pentru realizarea transcrierii muzicale prin următoarele mijloace:

- Funcții statice:
 - *second_to_quarter* – primește doi parametri *duration* și *tempo* și calculează numărul de pătrimi conținute în numărul de secunde dat de parametrul *duration* în funcție de ritm, dat de parametrul *tempo*
 - *remap* – primește cinci parametri: *x*, *in_min*, *in_max*, *out_min*, *out_max*, numare reale ce reprezintă respectiv: numărul de bază și capetele minim și maxim ale intervalelor input și output. Returnează o valoare cuprinsă între *out_min* și *out_max* raportată la poziția lui *x* în intervalul input [*in_min*, *in_max*]
- Clasa *Transcriptor*, ce prezintă metodele:
 - *__init__* – constructorul clasei ce primește un parametru, *path*, calea către un fișier Wav
 - *load* – încarcă fișierul din calea *path* cu ajutorul *librosa.load*

- `compute_cqt` – aplică o transformare CQT pe semnalul încărcat de metoda `load` cu ajutorul metodelor `librosa.cqt`, `librosa.magphase`, `librosa.amplitude_to_db`
- `compute_thresholded_cqt` – primește un singur parametru `cqt`, rezultatul metodei `compute_cqt`, căruia îi aplică un prag inferior, astfel încercă să elimine valorile ce corespund sunetelor ambientale, de volum scăzut care reduc precizia detectării atacului notelor
- `compute_onset_env` – primește un singur parametru `cqt`, pentru care este folosit rezultatul metodei `compute_thresholded_cqt` și returnează rezultatul metodei `librosa.onset.onset_strength` aplicat pentru acel `cqt`
- `compute_onset` – primește același parametru `cqt` și calculează estimativ atacul notelor cu ajutorul metodei `librosa.onset.onset_detect` în „cadre” și returnează un tuplu format din timpii atacului în milisecunde, timpii în cadre și `onset_envelope`
- `display_cqt_tuning` – afișează grafic rezultatul funcției `cqt`
- `estimate_tempo` – estimează tempo-ul clipului cu ajutorul metodei `librosa.beat.beat_track` și returnează un tuplu format din trei valori: `tempo` și `beats` (rezultatele returnate de metoda `beat_track`) și un obiect `MetronomeMark` creat pentru tempo-ul respectiv
- `generate_note` – primește trei parametri `f0_info`, `n_duration` și `round_to_sixteenth`, ce reprezintă respectiv un tuplu (frecvență, amplitudine), durata în cadre a notei și o valoare booleană care, dacă este setată pe „True”, rotunjește durata notei midi la șaisprăzecimi. Această metodă ia informația din parametrul `f0_info` și calculează amplitudinea semnalului ca valoare reală între 0 și 1 (cu ajutorul funcției `remap`) și durata în secunde. Generează apoi un obiect `Note` sau `Rest` ce reprezintă nota cu frecvența `f0_info[0]`, o listă `midi_info` ce conține informațiile midi corespunzătoare notei, duratei și volumului midi și o undă sinus cu frecvența și durata respectivă pentru redarea MIDI.
- `estimate_pitch` – primește doi parametri `segment` și `threshold` și returnează un tuplu format din frecvența predominantă pe segmentul audio respectiv și valoarea amplitudinii (înainte ca aceasta să fie „remaped” între 0 și 1)
- `estimate_pitch_and_notes` – primește un singur parametru `i`, număr natural folosit ca indice, apelează metoda `estimate_pitch` pentru un segment `cqt` mărginit de doi timpi de atac consecutivi (rezultați din metoda `compute_onset`)

- `transcript` – populează membrul intern *stream* al clasei cu notele *music21.note.Note* găsite cu ajutorul ultimelor trei metode descrise
- `show_stream` – afișează partitura din membrul intern *stream* cu ajutorul *MuseScore*
- `convert_stream_to_midi` – apelează metoda *streamToMidiFile* și obține un fișier

Clasa *MidiManagerToplevel* și interacțiunea cu fișiere MIDI

Dacă, în schimb, se dorește importul unui fișier MIDI, în interfața din *Figura 5* poate fi apăsat butonul „Import Midi”. Acesta va crea un obiect *MidiManagerToplevel* responsabil pentru lucrul cu acest tip de fișiere.

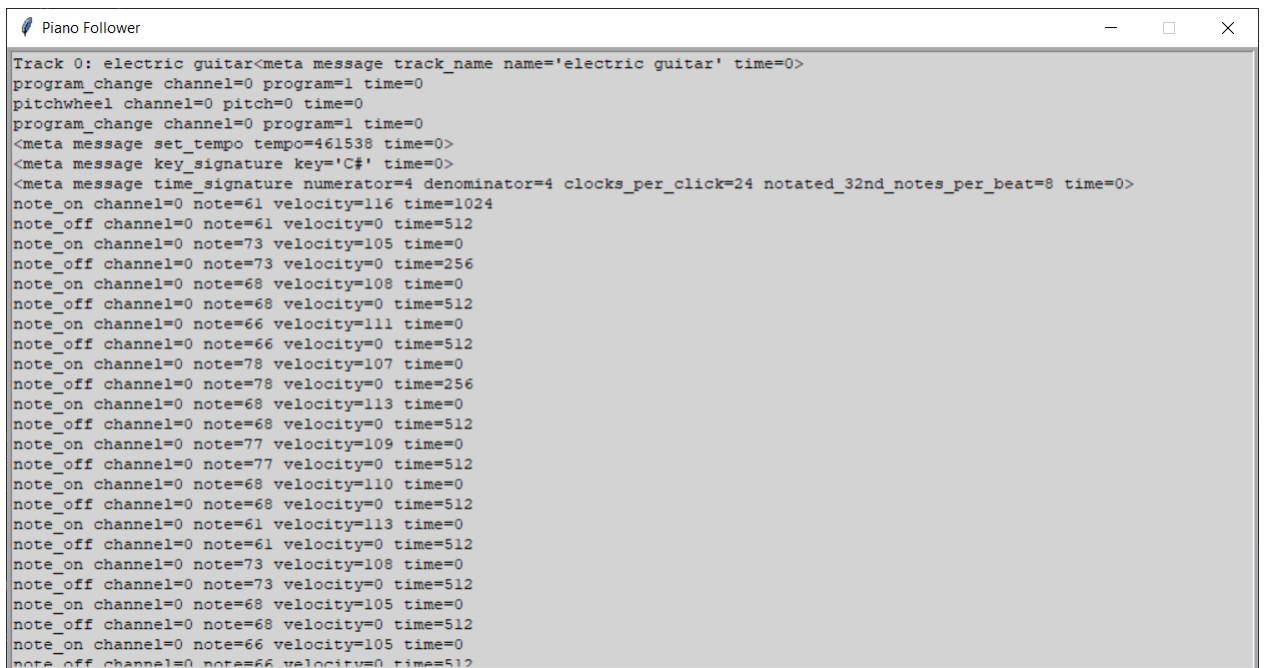


Figura 10: Midi Manager Toplevel

Constructorul clasei *MidiManagerToplevel* primește calea către un fișier midi și salvează în membrul *midi_object* un obiect *MidiManager*. Pe lângă constructor, clasa *MidiManagerToplevel* prezintă încă două metode: *show_midi* și *convert_to_score*, apelate de butoanele interfeței din *Figura 9*.

- `show_midi` – nu primește niciun parametru și returnează un nou obiect *MidiViewerToplevel*
- `convert_to_score` – de asemenea, nu primește niciun parametru și convertește fișierul midi importat într-unul mxl și îl afișează

Clasa *MidiViewerToplevel* se ocupă cu afișarea în mod text a fișierelor midi, cu ajutorul obiectelor *tk.Frame* și *tk.Text*, extrăgând mesajele midi din fișier (în formatul propus de librăria *mido*) și afișându-le într-un *tk.Toplevel* similar următoarei figuri:

The image shows a window titled "Piano Follower" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a text area displaying a list of MIDI messages. The messages include track information, program changes, tempo settings, key signatures, time signatures, and a series of note-on and note-off events for an electric guitar track. The notes are specified with channel, note number, velocity, and time in ticks.

```
Track 0: electric guitar<meta message track_name name='electric guitar' time=0>
program_change channel=0 program=1 time=0
pitchwheel channel=0 pitch=0 time=0
program_change channel=0 program=1 time=0
<meta message set_tempo tempo=461538 time=0>
<meta message key_signature key='C#' time=0>
<meta message time_signature numerator=4 denominator=4 clocks_per_click=24 notated_32nd_notes_per_beat=8 time=0>
note_on channel=0 note=61 velocity=116 time=1024
note_off channel=0 note=61 velocity=0 time=512
note_on channel=0 note=73 velocity=105 time=0
note_off channel=0 note=73 velocity=0 time=256
note_on channel=0 note=68 velocity=108 time=0
note_off channel=0 note=68 velocity=0 time=512
note_on channel=0 note=66 velocity=111 time=0
note_off channel=0 note=66 velocity=0 time=512
note_on channel=0 note=78 velocity=107 time=0
note_off channel=0 note=78 velocity=0 time=256
note_on channel=0 note=68 velocity=113 time=0
note_off channel=0 note=68 velocity=0 time=512
note_on channel=0 note=77 velocity=109 time=0
note_off channel=0 note=77 velocity=0 time=512
note_on channel=0 note=68 velocity=110 time=0
note_off channel=0 note=68 velocity=0 time=512
note_on channel=0 note=61 velocity=113 time=0
note_off channel=0 note=61 velocity=0 time=512
note_on channel=0 note=73 velocity=108 time=0
note_off channel=0 note=73 velocity=0 time=512
note_on channel=0 note=68 velocity=105 time=0
note_off channel=0 note=68 velocity=0 time=512
note_on channel=0 note=66 velocity=105 time=0
note_off channel=0 note=66 velocity=0 time=512
```

Figura 11: MidiViewerToplevel

Clasa *MidiManager* se ocupă de interacțiunea cu fișierele MIDI. Constructorul admite doi parametri *path* și *midi_file* reprezentând calea către un fișier și o valoare booleană ce definește dacă fișierul din *path* este unul MIDI sau nu. (în cazul în care nu este unul MIDI, se folosește fișierul text în care este salvat inputul din clapele interfeței principale și se crează un obiect *mido.MidiFile* nou ce urmează a fi scris cu informația din acel fișier, altfel se citește informația din fișierul MIDI într-un obiect *mido.MidiFile*)

Această clasă mai prezintă alte trei metode:

- *midi_msg_to_mido_msg* – metodă ce convertește mesajele midi salvate în fișierul text în mesaje mido
- *convert_midi_to_stream* – convertește obiectul *mido.MidiFile* într-un *music21.stream.Stream*
- *save_midi* – salvează informația midi într-un fișier MIDI a cărui nume este setat de parametrul *filename*

Clasa ScoreManagerToplevel

Pentru importul unui fișier MXL, interfața din *Figura 5* prezintă butonul „Import Score”, ce creează un obiect *ScoreManagerToplevel*



Figura 12: Score Manager Toplevel

Această interfață pune la dispoziție trei butoate pentru afișarea partiturii, conversia la MIDI și compararea cu o altă partitură, respectiv.

În constructorul acestei clase este generat un obiect *ScoreManager* responsabil pentru realizarea funcționalităților celor trei butoane.

Clasa *ScoreManager* poate primi ca parametru calea către un fișier MXL pe care să-l citească și să-l transforme într-un *music21.stream.Stream* prin metoda *music21.converter.parse* sau chiar un *music21.stream.Stream*.

Metodele clasei *ScoreManager* sunt:

- `show` – afișează partitura în MuseScore
- `show_text` – afișează partitura în format text în consolă
- `convert_stream_to_midi` – convertește partitura într-un fișier midi
- `compare` – primește un singur parametru *filename* ce reprezintă calea către alt fișier MXL cu care să fie comparat cel intern și returnează un nou obiect *ScoreComparerToplevel*, responsabil pentru afișarea ferestrei de comparare a două partituri

Clasa *ScoreComparerToplevel* moștenește *tkinter.Toplevel*. Constructorul acestei clase primește doi parametri *stream1* și *stream2* ce reprezintă două obiecte *music21.stream.Stream* și apelează metoda *init_ui*. Această metodă generează două cadre *tk.Frame*, două zone de text *tk.Text* plasate în cadrele respective și un al treilea cadru *tk.Frame* pentru afișarea graficului de comparare ca în *Figura 13*.

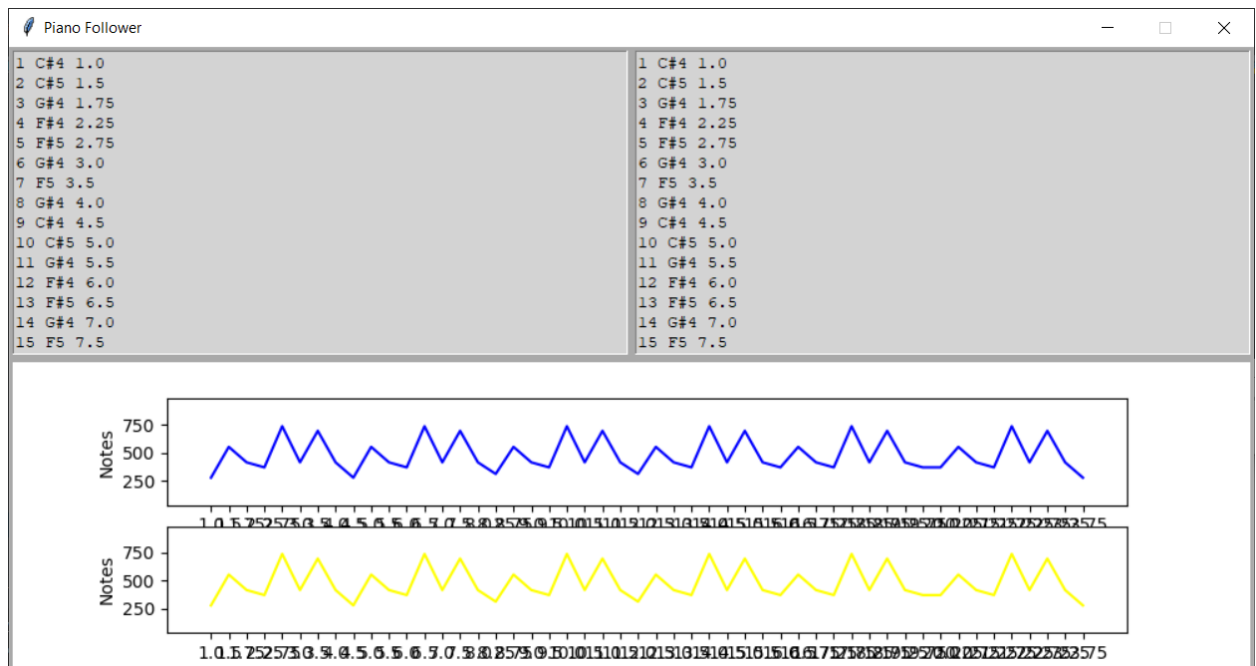


Figura 13: Interfața de comparare pentru două fișiere mxl identice

Clasa RecorderToplevel și înregistrarea audio externă

Interfața pentru înregistrarea externă este implementată în clasa *RecorderToplevel* ce moștenește *DefaultToplevel* și conține cinci butoane cu nume sugestive pentru înregistrare, oprire, export în format Wav, conversie la format midi și respectiv mxl.

Constructorul *RecorderToplevel* primește ca parametru un obiect „parent” ce reprezintă widget-ul apelant și creează un obiect *AudioRecorder* responsabil pentru funcționalitățile înregistrării.

```
class RecorderToplevel(DefaultToplevel):
    def __init__(self, parent):
        self.recorder = AudioRecorder('output.wav')
        super().__init__(parent)
```

Clasa *AudioRecorder* pune la dispoziție funcționalitățile de bază în vederea capturării semnalului primit în microfon, cu ajutorul librăriei PyAudio, și o metodă de salvare a clipului înregistrat în format Wav, cu ajutorul librăriei Wave.

Pentru înregistrare, trebuie creată o instanță a unui obiect *pyaudio.PyAudio* care va deschide un flux de date pentru citirea informației primite de la microfon.

```
self.p = pyaudio.PyAudio()
self.stream = self.p.open(
    format=self.format,
    channels=self.channels,
    rate=self.rate,
    input=True,
    output=True,
    frames_per_buffer=self.chunk
)
```

Datele citite sunt stocate într-o listă internă clasei numită *frames*:

```
def init_rec(self):
    print('stream started at time {}'.format(time.time()))
    self.recording = True
    while self.recording:
        data = self.stream.read(self.chunk)
        self.frames.append(data)

def stop_rec(self):
    print('stream stopped at time {}'.format(time.time()))
    self.recording = False
    self.stream.stop_stream()

def exit_app(self):
    print('stream closed')
    self.p.close(self.stream)
```

Conținutul acestei liste este apoi scris într-un obiect Wave și salvat în format Wav și este apelată metoda *exit_app* ce închide fluxul de citire.

```
def export_wav(self):
    with wave.open(self.filename, 'wb') as wf:
        wf.setnchannels(self.channels)
        wf.setsampwidth(self.p.get_sample_size(self.format))
        wf.setframerate(self.rate)
        wf.writeframes(b''.join(self.frames))
    self.exit_app()
```

În cazul de față, pentru scrierea fișierului înregistrat prin microfon am folosit modul „wb”. Acesta creează un obiect *wave_write* ce dispune de o multitudine de metode cu nume sugestive care au ca scop atribuirea valorilor necesare scrierii unui fișier WAV (mai multe detalii sunt descrise în capitolul anterior).

Funcția *writeframes* primește ca parametru datele ce trebuie scrise în fișier în format binar. În exemplul curent datele respective sunt reprezentate în lista *frames* internă clasei, ce sunt convertite în binar cu ajutorul metodei *join()* a clasei *String* din python.

Clasa PianoManagerToplevel

Pentru transcrierea inputului primit în clape sau salvarea acestuia în format midi am implementat clasa *PianoManagerToplevel* ca subclasă a *DefaultToplevel*. Constructorul generează un obiect *MidiManager* ce este încărcat cu fișierul text în care este salvat inputul MIDI.

Această clasă prezintă metodele:

- *convert_to_midi()* – convertește înregistrarea text în clip midi
- *convert_to_score()* – convertește înregistrarea text în partitură

```
def convert_to_midi(self):
    self.midi_object.midi_msg_to_mido_msg()
    file = filedialog.asksaveasfile(initialdir="/", title="Save file",
                                     filetypes=(('midi files', ('*.mid',
                                                                '*.midi')),
                                                ('all files', '*.*')))

    self.midi_object.save_midi(file.name)

def convert_to_score(self):
    file_path = filedialog.askopenfilename(initialdir='/', title='Open
Midi',
                                           filetypes=(('midi files',
                                                       ('*.mid', '*.midi')),
                                                       ('all files', '*.*')))
    self.midi_object.convert_midi_to_stream(file_path)
```

Capitolul IV: Concluzii

În concluzie, aplicația *Piano Follower* aduce utilizatorilor posibilitatea de a manevra fișiere audio WAV, fișiere MIDI și fișiere MXL în scopul realizării transcrierii muzicale sau a comparării a două partituri.

În urma documentației și a implementării aplicației, am descoperit dificultatea realizării procesului de transcriere muzicală. Pentru marea majoritate a melodiilor existente, acest proces este, în prezent, imposibil datorată multitudinii de instrumente și voci prezente în aceste piese. Rezultatele optime pot fi găsite pe înregistrări de pian relativ simple (sau alte instrumente ce nu prezintă vibrato puternic), însă, chiar și atunci, algoritmi de estimare a atacului nu funcționează mereu ideal datorită variației de volum. În practică, aplicația ar putea atinge cele mai bune rezultate în fiecare caz particular dacă pentru fiecare înregistrare ar fi încercate diferite valori pentru membrii clasei *Transcriptor* și anume *mag_exp*, *pre_post_max* și *threshold* pentru a se potrivi mai bine cu amplitudinea și distanța dintre notele din înregistrare până când se obține un rezultat vizual mai bun la metoda *display_cqt_tuning* (liniile coloană ce reprezintă atacurile să fie cât mai apropiate de începutul fiecărui segment roșu ce reprezintă frecvența dominantă pe acel pasaj).

Inovația aplicației este compararea de partituri în mod grafic, lucru care nu a mai fost abordat până acum oficial, după cunoștințele mele.

Bibliografie:

1. ISMIR. (2000). *MIR Software*. <http://www.ismir.net/resources/software-tools/>
2. Aliaksandr Paradzinets, Hadi Harb, Liming Chen. *Use of continuous Wavelet-like Transformation in automated music transcription*.
https://www.researchgate.net/publication/200688645_Use_of_Continuous_Wavelet-like_Transform_in_automated_music_transcription
3. Renato Profeta, Guitars.AI, Technische Universität Ilmenau
<https://github.com/GuitarsAI/BasicAutoTranscriptionRepo>
4. <https://librosa.github.io/librosa/>
5. <https://mido.readthedocs.io/en/latest/>
6. <https://musescore.com/dashboard>
7. <https://wiki.python.org/moin/PythonInMusic>
8. <http://web.mit.edu/music21/>
9. <http://www.music-software-development.com/>
10. <http://zulko.github.io/blog/2014/02/12/transcribing-piano-rolls/>
11. <https://newt.phys.unsw.edu.au/jw/notes.html>
12. <http://zulko.github.io/blog/2014/03/29/soundstretching-and-pitch-shifting-in-python/>
13. <https://www.pygame.org/docs/>
14. <http://www.music-software-development.com/midi-tutorial.html>
15. <https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message>
16. https://docs.python.org/3/library/_thread.html
17. <https://www.tcl.tk/man/tcl8.4/TkCmd/bind.htm>
18. https://www.researchgate.net/publication/200688645_Use_of_Continuous_Wavelet-like_Transform_in_automated_music_transcription