**Universitatea Tehnică din Cluj-Napoca**

**Facultatea de Electronică, Telecomunicații și Tehnologia Informației**

**ASRSV**

**Sintetizator virtual implementat cu Python**

Profesor coordonator: Student:

prof. dr. ing. Eugen Lupu ing. Jimon Lucian-Daniel

2024

**Cuprins**

[Introducere 3](#_Toc156240922)

[Despre proiect 3](#_Toc156240923)

[Problema 3](#_Toc156240924)

[Soluția 3](#_Toc156240925)

[Tehnici de programare Python 3](#_Toc156240926)

[Fundamentare teoretică 4](#_Toc156240927)

[Sinteza digitală a sunetelor 4](#_Toc156240928)

[Generarea de forme de undă 4](#_Toc156240929)

[Notele Muzicale 4](#_Toc156240930)

[Funcționarea Sintetizatoarelor 4](#_Toc156240931)

[Imitarea Instrumentelor Acustice 5](#_Toc156240932)

[Tehnici Avansate de Sinteză 5](#_Toc156240933)

[Anvelope ADSR 5](#_Toc156240934)

[Aplicații Practice în Muzică și Design de Sunet 6](#_Toc156240935)

[Importanța Interfețelor Utilizator 6](#_Toc156240936)

[Viitorul Sintezei Sunetului 6](#_Toc156240937)

[Implementare 7](#_Toc156240938)

[Backend 7](#_Toc156240939)

[GUI 9](#_Toc156240940)

[Rezultate 11](#_Toc156240941)

[Generare de sunete pe bază de forme de undă 11](#_Toc156240942)

[Generare de sunete pe bază de mostre 11](#_Toc156240943)

[Efecte dinamice 11](#_Toc156240944)

[Efect de ecou 11](#_Toc156240945)

[FM 11](#_Toc156240946)

[Elemente GUI 12](#_Toc156240947)

[Concluzie și posibile îmbunătățiri 13](#_Toc156240948)

[Bibliografie 14](#_Toc156240949)

[Anexă 16](#_Toc156240950)

# Introducere

## Despre proiect

Prezentul proiect intitulat „Sintetizator virtual implementat cu Python” a fost structurat și dezvoltat ca parte a materiei „Analiza, sinteza și recunoașterea semnalului vocal”. Proiectul utilizează mai multe aspecte teoretice și de programare ce sunt utilizate în domeniul strâns legat al materiei, acela de analiză și recunoaștere vocală, însă diferă în motivul principal, care în cazul de față este generare de sunete muzicale, și nu vocale.

Scopul proiectului a fost dezvoltarea unui software care, bazat pe tehnici de procesare și generare de sunet, poate simula o octavă dintr-un sintetizator (orgă) digitală, sunetul căreia poate fi modificat de utilizator.

### Problema

Problema principală a obiectivului proiectului a fost găsirea unei soluții de a putea genera mai multe sunete în același timp, pentru a reda senzația de orgă reală, cât de mult se poate. Acest aspect a fost tratat prin strategia de multi-threading, astfel procesorul poate genera notele muzicale în mod paralel. O altă problemă a fost găsirea unor mostre de la instrumente muzicale, de calitate bună.

### Soluția

După cum sugerează titlul lucrării, limbajul de programare folosit a fost Python, care, printre alte beneficii extraordinare, vine cu o utilizare extinsă a diferitelor biblioteci și funcții predefinite sau open-source, în special în domeniul științei datelor. Astfel, Python a fost ales ca instrument principal al proiectului, și module specifice analizelor și sintezelor de date ca numpy. Proiectul combină în mod unic sinteza digitală a sunetului, anvelopa ADSR, efectele de întârziere și elementele interactive ale interfeței grafice pentru o experiență de utilizare plăcută.În cele din urmă, deoarece scripturile au fost destinate să fie utilizate de utilizatori umani, GUI-ul pentru proiect a fost realizat cu Tkinter, care este biblioteca standard GUI pentru Python.

Mostrele auditive cu diferite note muzicale a unor instrumente reale au fost găsite pe internet.

## Tehnici de programare Python

Python este unul dintre cele mai populare limbaje de programare, fiind un limbaj rapid, ușor de înțeles și accesibil, cu mii de biblioteci disponibile, făcându-l cel mai folosit limbaj de programare din câțiva ani (la ora actuală Python este încă cel mai folosit limbaj de programare și a câștigat premiul „Cel mai bun limbaj de programare 2021” [1] ).

Una dintre principalele utilizări ale Python este în știința datelor, iar acest lucru poate fi dovedit de multitudinea de biblioteci de învățare automată, simplitatea utilizării lor în scripturi și ușurința sintaxei limbajului, făcându-l perfect pentru scopuri precum acest proiect. .

# Fundamentare teoretică

## Sinteza digitală a sunetelor

Sinteza digitală a sunetelor este un proces complex și diversificat care implică crearea de sunete prin metode electronice, folosind algoritmi de calculator. Această tehnică, care a evoluat considerabil de la primele sale zile, atunci când sintetizatoarele analogice erau standardul, a cunoscut o evoluție remarcabilă în era digitală. Sintetizatoarele moderne pot fi fie hardware, fie software, aducând o nouă putere și flexibilitate în tehnologia digitală, care reflectă progresele semnificative în tehnologia audio și informatică [2]. Ele permit producătorilor de muzică și artiștilor să creeze sunete care altfel ar fi dificil sau imposibil de generat prin metode acustice tradiționale.

## Generarea de forme de undă

În centrul sintezei sunetului stă generarea formelor de undă. O formă de undă este o reprezentare grafică a variațiilor de presiune sonoră sau a semnalului electric care produce sunetul. Sintetizatoarele digitale încep procesul de sinteză cu un oscilator care generează o frecvență sonoră sub forma unei forme de undă. Există mai multe tipuri de forme de undă utilizate în sinteza digitală, inclusiv undele sinusoidale, pătrate și triunghiulare.

* Unda Sinusoidală: Este cea mai simplă formă de undă și produce un ton pur și clar. Datorită naturii sale netede, este adesea utilizată ca bază pentru alte forme de undă mai complexe.
* Unda Pătrată: Are un caracter distinct, producând un ton mai aspru și mai bogat în armonice decât unda sinusoidală. Este frecvent folosită în muzica electronică pentru a crea sunete de bas sau lead-uri care se remarcă în mix.
* Unda Triunghiulară: Ocupă un loc intermediar între unda sinusoidală și cea pătrată, având un caracter mai moale decât unda pătrată, dar mai bogat în armonice decât unda sinusoidală.

## Notele Muzicale

Notele muzicale sunt esențiale în muzică, fiecare notă reprezentând o frecvență sonoră specifică. De exemplu, nota "La" standard (A4) are o frecvență de 440 Hz. În sintetizatoare, fiecare notă este generată prin ajustarea frecvenței oscilatorului pentru a produce frecvența corespunzătoare notei. Acest lucru permite sintetizatoarelor să imite sunetele instrumentelor acustice sau să creeze sunete complet noi și unice [3].

## Funcționarea Sintetizatoarelor

Sintetizatoarele muzicale funcționează prin generarea și manipularea electronică a undelor sonore, oferind o platformă versatilă pentru crearea unei game largi de sunete. Iată o prezentare generală a modului în care funcționează sintetizatoarele:

„Inima” unui sintetizator este oscilatorul. Acesta produce unde sonore primare, cum ar fi undele sonore în formă de ferăstrău, pătrate, sinusoidale, triunghiulare sau zgomot. În sintetizatoarele digitale, sunetele pot fi preluate din tabele de unde sau chiar mostre înregistrate. Tonalitatea acestor unde este adesea modificată de tensiuni de control, care sunt generate de o tastatură sau de un secvențiator.

După oscilator, sunetul trece prin o secțiune de filtrare (sau VCF), unde sunt modelate calitățile sale armonice. Există diferite tipuri de filtre, inclusiv filtre trece-jos, trece-înalt, trece-bandă și filtre crestătoare. Fiecare tip permite în mod selectiv trecerea anumitor frecvențe și blochează altele, modelând astfel timbrul sunetului [4].

Sunetul ajunge apoi la un amplificator (VCA), care îi determină volumul. O anvelopă atașată la amplificator controlează modificarea volumului în timp folosind patru etape: Attack, Decay, Sustain și Release (ADSR). Aceste etape definesc caracterul sunetului pe măsură ce evoluează din momentul în care este cântată o notă până când aceasta se estompează [4] [5].

## Imitarea Instrumentelor Acustice

Unul dintre cele mai fascinante aspecte ale sintetizatoarelor este capacitatea lor de a imita sunetele instrumentelor acustice tradiționale, cum ar fi pianul, vioara sau flautul. Acest lucru se realizează prin modelarea atentă a caracteristicilor specifice ale sunetelor acestor instrumente, cum ar fi atacul, decăderea, menținerea și eliberarea sunetului, precum și prin simularea rezonanței și altor proprietăți acustice.

## Tehnici Avansate de Sinteză

Pe lângă formele de undă de bază și imitarea instrumentelor acustice, sintetizatoarele digitale oferă și alte tehnici avansate de sinteză. Printre acestea se numără:

* Sinteza Aditivă: Această tehnică modelează un spectru sonor ca un set de linii discrete corespunzătoare sinusoidelor. Prin combinarea diferitelor frecvențe și amplitudini ale acestor componente, se poate crea o gamă largă de sunete
* Sinteza substractivă: Acesta este un tip comun de sinteză a sunetului în sintetizatoarele analogice, în care părțile parțiale ale unui semnal audio bogat în armonici sunt atenuate de un filtru pentru a modifica timbrul sunetului. Se pornește de la o formă de undă complexă, care este apoi modelată prin filtre și anvelope.
* Sinteza FM (Modulare în Frecvență): Utilizează două oscilatoare, unul pentru generarea unei frecvențe originale și celălalt pentru modularea primului prin introducerea armonicelor suplimentare. Această tehnică a fost popularizată de sintetizatorul Yamaha DX7 și este cunoscută pentru sunetele sale metalice și vibrante etc [6].

## Anvelope ADSR

Un aspect esențial al sintezei sunetului este utilizarea anvelopelor ADSR (Attack, Decay, Sustain, Release) pentru modelarea dinamicii sunetelor. Anvelopa ADSR definește modul în care intensitatea unui sunet evoluează de-a lungul timpului, de la momentul inițierii (atac) până la sfârșitul său (eliberare). Faza de „Atac” determină rapiditatea cu care sunetul atinge punctul său maxim după ce o notă este apăsată, urmată de „Decay” care descrie ritmul de scădere a sunetului la nivelul „Sustain”, care este intensitatea constantă a sunetului până la eliberarea notei. În cele din urmă, faza de „Release” descrie cum sunetul se estompează după ce nota nu mai este apăsată. Aceste faze se pot observa în Figura 1, preluată de la [7].  
Acest mecanism permite artiștilor să controleze expresivitatea și forma sunetului, de la sunete abrupte și percutive, la tonuri lungi și evolutive, fiind un instrument esențial în designul sonor și sinteza muzicală [8].

## Aplicații Practice în Muzică și Design de Sunet

Figură 1: Anvelopa ADSR

Sinteza digitală a sunetelor a devenit o componentă esențială în producția muzicală contemporană, designul de sunet pentru filme și jocuri video, precum și în crearea de efecte sonore speciale. Compozitorii și designerii de sunet se bazează pe flexibilitatea și diversitatea sunetelor sintetizate pentru a crea compoziții unice și medii imersive.

## Importanța Interfețelor Utilizator

Interfețele utilizator ale sintetizatoarelor, fie că sunt hardware sau software, joacă un rol crucial în modul în care artiștii interacționează cu tehnologia. Interfețele bine proiectate permit un acces rapid și intuitiv la parametrii sintetizatorului, facilitând experimentarea și crearea sunetelor. Aceasta include utilizarea de controllere MIDI, interfețe tactile și alte dispozitive care ajută la crearea unei conexiuni mai directe și expresive între artist și instrument.

## Viitorul Sintezei Sunetului

Pe măsură ce tehnologia avansează, noi posibilități apar în domeniul sintezei sunetului. Dezvoltările recente includ sinteza bazată pe inteligență artificială, care poate genera sunete și muzică în moduri noi și neașteptate, și tehnologiile emergente vor continua să transforme modul în care interacționăm și experimentăm muzica și sunetul [9].

# Implementare

Implementarea proiectului a fost realizată în două fișiere script „.py”.

## Backend

Partea principală a programului este prezent în fișierul „synth.py” care este un fel de backend pentru proiect. În acest fișier cod sunt definite funcțiile de procesare de sunet, sinteză, generare și redare.

Funcția pentru generarea formelor de unde, generate\_waveform() este capabilă să creeze diferite tipuri de forme de undă, cum ar fi sinusoidală, pătrată, sinusoidală-pătrată și forme de undă bazate pe mostre sonore (pian, flaut, trompetă).

În cazul în care utilizatorul alege forme de unde generate, programul generează acestea cu ajutorul modului numpy, care pe baza capabilităților sale avansate de calcul numeric, permite crearea rapidă și eficientă a formelor de undă. De exemplu, în cadrul funcției generate\_waveform(), pentru a genera o undă sinusoidală, codul calculează un set de eșantioane pe parcursul duratei specificate prin formula: np.sin(2 \* np.pi \* freq \* t), unde freq reprezintă frecvența undei și t este un array de valori de timp, calculat pentru a se potrivi cu rata de eșantionare și durata. Aceasta produce o undă sinusoidală pură, reprezentativă pentru tonul respectiv. Pentru unda dreptunghiulară, codul folosește np.sign(np.sin(2 \* np.pi \* freq \* t)), care generează valori de +1 și -1, reprezentând formele abrupte ale unei unde dreptunghiulare. Semnalul numit „sine-square” este combinația celor două menționate anterior.

Pe de altă parte, dacă se alege utilizarea mostrelor de sunet de la instrumente muzicale reale, acestea sunt preluate din directoare specificate, cu locația lângă codul sursă, unde fiecare notă muzicală este stocată în câte un fișier de tip „.wav”, denumindu-se după frecvența sunetului. Astfel sunt câte 12 fișiere audio pentru fiecare instrument, anume:

* 261.63.wav (C4)
* 277.18.wav (C#4)
* 293.66.wav (D4)
* 311.13.wav (D#4)
* 329.63.wav (E4)
* 349.23.wav (F4)
* 369.99.wav (F#4)
* 392.00.wav (G4)
* 415.30.wav (G#4)
* 440.00.wav (A4)
* 466.16.wav (A#4)
* 493.88.wav (B4)

Cu funcția load\_instrument\_samples se deschide fișierul căutat pe bază de instrument și frecvență cu ajutorul modului wave, care apoi este delimitat în cadre, din care mai departe se extrag eșantioanele cu numpy, care se stochează într-un array separat pentru fiecare notă. În generate\_waveform, în cazul unei opțiuni de instrument real, se caută notele pentru instrumentul cutare, și fiecare formă de undă specifică frecvenței notei muzical va avea propria formă de undă generată din sunete înregistrate.

O altă funcționalitate implementată este cea de anvelopă ADSR (attack-decay-sustain-release), o tehnică de manipulare a sunetului foarte des folosită. Aceste patru efecte redau dinamica sunetului generat de către sintetizatorul virtual, și respectă parametri aleși de către utilizator. Dacă nu este ales niciun parametru, anvelopa nu este utilizată. Fiecare din cele patru efecte dinamice se poate trata ca un array de valori, definite cu ajutorul modului numpy. Attack este definit ca un vector de valori între 0 și 1, care are numărul de elemente egal cu valoarea de „attack” aleasă de către utilizator și rata de eșantionare care este setată la 44.1 kHz (np.linspace(0,1,attack\_length)). Decay este un vector de valori între 1 și valoare de sustain, care are numărul de elemente egal cu valoarea de „decay” aleasă de către utilizator și rata de eșantionare (np.linspace(1, sustain\_level, decay\_length)). Sustain este un vector de lungime sustain\_length (care este lungimea în timp a perioadei în care intensitatea sunetului se menține stabilă și este calculată ca fiind egal cu len(waveform) - attack\_length - decay\_length - release\_length), care are doar elemente de valoare sustain\_level. Release este reprezentat ca un vector de lungime release\_length, cu valori între sustain\_level și 0. La finalul funcției, forma de undă este înmulțită cu această anvelopă, astfel eșantionale vor avea valori diferite bazate pe parametri de timp și de valoare pentru fiecare etapă dinamică.

Pentru generarea de efect de ecou, se folosesc trei parametri luați de la GUI: feedback, delay\_time și mix. Primul este intensitatea ecoului, al doilea este timpul de delay acordat între sunetul principal și ecoul său, iar ultimul parametru controlează balanța între sunetul original și ecou. Initial, prin funcția de apply\_delay se definesc eșantionele de delay pe baza de delay\_time și rata de eșantionare, după care se crează un vector (care simulează o formă de undă) de lungime formă de undă de input + eșantionele de delay, prima parte a vectorului va fi locul eșantionelor din sunetul de bază iar restul valorilor vor fi 0 inițial, după care cu un ciclu for se modifică eșantionele astfel încât fiecărui eșantion cu index i+delay\_samples se adaugă valoarea sa înmulțită cu parametrul de feedback: delayed\_waveform[i + delay\_samples] += delayed\_waveform[i] \* feedback. După aceasta se generează outputul fiind o combinație între forma de undă originală și cea modificată cu ecou: output = delayed\_waveform[:len(waveform)] \* mix + waveform \* (1 - mix).

Implementarea modulației de frecvență (FM) este implementată în generate\_waveform și este bazată pe cele două argumente preluate de la interfața grafică, frecvența de modulare și indexul de modulare. Atunci când opțiunea de „fm” este aleasă, un sinusoid este generat cu numpy, fiind forma de undă modulatoare, iar outputul va fi un alt sinus în a cărei faze va fi integrat și forma de undă modulatoare, înmulțit cu indexul de modulare, cu linia de cod: waveform = np.sin(2 \* np.pi \* freq \* t + modulation\_index \* modulator), unde freq este frecvența notei muzicale actuale.

O altă funcționalitate importantă este paralelitatea, ceea ce ajută programul în a simula un pian real, pe care un muzician poate produce multiple sunete deodată (pentru acorduri de ex.). Aceasta este realizată în două faze. În prima fază, la pornirea programului, se pregenerează notele muzicale pentru tipurile de sunet simple (sinus, dreptunghiular și combinație), cu parametri de ADSR aleși impliciți și fără delay și feedback, pentru a optimiza programul inițial. După, la sfârșitul funcției de redare de sunet, se apelează funcția din backend de redare ca o funcție lamda, fiind un thread separat definit cu ajutotul modului the Thread threading.Thread(target=lambda: play\_waveform(waveform)).start(). Astfel fiecare sunet produs va fi controlat separat de către CPU, asigurând posibiliatea de a produce acorduri și multiple sunete.

## GUI

În cele ce urmează, va fi prezentată GUI-ul aplicației, care este dezvoltat în cadrul „SynthApp.py”.

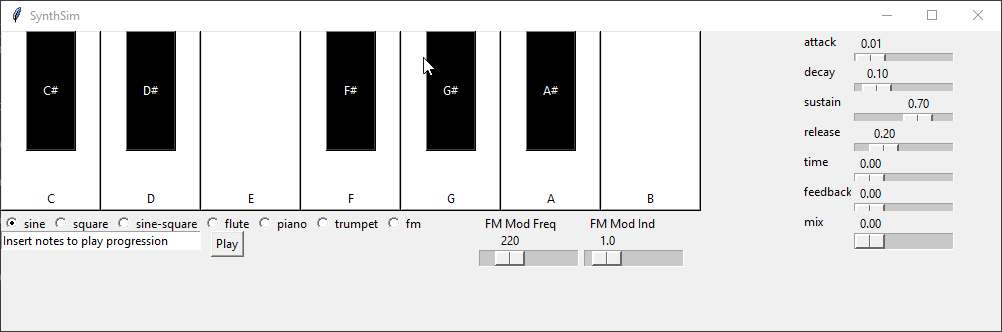


Figura 2: GUI

După cum arată Figura XX, interfața grafică proiectată are trei funcționalități principale:

1. Implementarea unei claviaturi virtuale, cu o singură octavă. Aceasta poate fi folosită atât apăsând cu cursorul pe clape, cât și folosind tastatura, notele fiind mapate în plaja tastelor A-J
2. Setarea sunetului dorit, care poate fi modificat privind sunetul (forme de unde ‚ca sinus, dreptunghi sau utilizarea mostrelor de sunet de la instrumente reale), effecte ADSR sau feedback și modulație de frecvență cu setarea parametrilor
3. Scrierea unei secvențe de note muzicale care după, apăsând pe butonul Play, se poate reda cu ajutotul sintatizoarului, care va prelua și setările de sunet

Întreaga interfață grafică a fost creată cu ajutorul modului grafic implicit al Python, Tkinter.

Claviatura virtuală este implementată cu ajutorul obiectelor de tip buton din modulul menționat, care sunt setate să fie sau albe (pentru note întregi) sau negre (pentru semi note), pentru a semăna cu un pian real cât de bine posibil. La apăsarea unui buton sau a unei taste mapate de pe tastatură, funcția de play\_sound implementată în backendul proiectului va prelua setările de sunet pentru note și va reda sunetul generat sau căutat din mostre, cu ajutorul modului simpleaudio.

Setările de efecte și timbre sunt realizate prin butoane de tip radio sau slider-uri pentru a alege valori, care au magnitudine dependentă de felul efectului (de ex. effectul „attack” poate avea valoare între 0 și 1, pe când frecvența de modulație poate fi între 0 și 1000).

A screenshot of a computer

Description automatically generatedRedarea unui șir de note muzicale se realizează cu un textbox, în care utilizatorul trebuie să introducă note muzicale reale, cu majuscule. Acestea vor fi redate utilizând setările de sunet alese, cu o pauză de 0.5s între ele pentru a generaliza funcționalitatea. În cazul în care utilizatorul introduce note muzicale incorecte, un mesaj de eroare apare pe ecran, așa cum prezintă figura următoare.

Figură 3: Mesaj de eroare în caz de note greșite

# Rezultate

## Generare de sunete pe bază de forme de undă

Generarea de note muzicale pe bază de sinus, undă dreptunghiulară și combinația acestora funcționează corect în cadrul aplicației. Sunetul de sinus este unul clar, care seamănă întocmai cu un sunet de orgă electrică cu un efect estompat. Sunetul „square” este unul foarte sintetic, care seamănă cu coloanele sonore a primelor jocuri video, de exemplu Super Mario Bros [10] sau Doom [11]. Combinația sinus-dreptunghiular produce un sunet așteptat, care este mai „sintetic” decât sinusoidul, dar mai natural și clar decât sunetul undei dreptunghiulare, și aduce a sunet de xilofon electric vechi.

## Generare de sunete pe bază de mostre

Mostrele folosite au fost preluate din baza de date a Universității din Iowa [12] și au fost tăiate și convertite în format .wav din .aiff. Din cauza procedurii de tăiere și export, la unele note se aude un sunet de tic (în special la flaut notele E și F#, pian notele B și F#, trompă notele C#, G și G#) și sunetul se aude nenatural, având o dinamică necorespunzătoare pe alocuri, iar la unele note, din cauza inexactității tăierii, sunetul are lungime mai scurtă (pian nota C). Eșantionarea de note muzicale prezintă atât avantajele (sunete reale) cât și dezavantajele (dificultate în găsirea și procesarea mostrelor) a procedeului.

## Efecte dinamice

Efectele anvelopei ADSR pot fi utilizate pentru oricare din semnalele sine, square sau sine-square. Acestea adaugă o dinamică variată notelor muzicale, și se poate observa toate 4 faza ale sunetului prin modificarea valorilor. Valoare de attack influențează începutul sunetului, o valoarea mare creează un efect care seamănă cu sunetul de orgă reală. Valoarea de decay în sine produce efecte subliniate dacă celelalte valori sunt setate la 0: o valoare mică produce un sunet estompat de xilofon simulat, o valoare mai mare alungește sunetul și îi atribuie adâncime. Sustain are singurul efect de a lungi sau scurta sunetul, așa cum era de așteptat. Valoarea de release în sine nu produce efect, dar cu celelalte valori setate și cu release nenul se produce un sunet aproape tăiat, cu un efect puternic descendent al sunetului. Combinația celor 4 poate produce diferite efecte, dar dacă toate sunt setate 0, nu se mai produce sunet, iar dacă mai mult de două valori sunt setate mari, se produce un sunet de tic la sfârșitul notei.

## Efect de ecou

Combinația de time-feedback produce efecte palpabile doar în cazul în care variabile de mix are o valoare mai mare, acordând formei de undă secundare o putere mai mare. Astfel un timp mare de delay va cauza neproducerea ecoului, căci sunetul va ieși din plaja de 0.5s predestinat, iar o valoare mică va produce un ecou aproape instant după sunetul principal.

## FM

În cazul modulației în frecvență, o frecvență joasă de semnal modulator produce un semnal asemănător notelor joase de la orga clasică, o frecvență medie generează un sunet sintetic care aduce cu muzica electronică industrială și o frecvență înaltă produce sunete de clopot, sau de sonerii ale telefoanelor mobile vechi. Acest ultim aspect este atât mai subliniat cu cât amplitudinea setată a semnalului modulator este mai mare. Amplitudinea în cazul sunetelor joase are poate cel mai impresionant efect: o amplitudine mică produce sunetul de orgă, una mijlocie aduce cu un claxon pentru tiruri iar o valoare mare face sunetul să fie ca unul de bas sintetic folosit în genul „drum and bass” sau „house”.

Modulația în frecvență implementat demonstrează vasta gamă de posibilități pentru sunete ce poate fi produs cu această tehnică și prin analiza și studiul diferitor combinații de valori și sunetele produse, se pot genera sunete inedite pentru orice stil de muzică.

## Elemente GUI

Partea de GUI prezentat în partea de Implementare funcționează corect, fiecare tastă din setul A,W, S, E, D, F, T, G, Y, H, U, J are mapat corect nota muzică cutare, iar atunci când utilizatorul dă click pe clapă sau apasă una din tastele menționate se produce sunetul ca atare.   
Funcționalitatea de redare a secvenței muzicale respectă cerințele menționate în lucrare, și dacă utilizatorul introduce o secvență care conține doar note muzicale, scrise cu majuscule, aceasta este redată cu setările rămase de la alegerea sunetului, efectelor dinamice și de ecou. În caz contrar, o fereastră cu un mesaj de eroare apare.

Setarea valorilor de ADSR și de ecou prin elemente de tip slider funcționează ca atare, și fiecare valoare este bine gestionată de către program; pe de altă parte, atunci când o valoare este schimbată de la valoarea implicită, sunetul inițial nu mai poate fi redat decât în cazul în care utilizatorul găsește exact valorile implicite, cu care începe programul.

# Concluzie și posibile îmbunătățiri

Proiectul „Sintetizator virtual implementat cu Python” a reușit să demonstreze eficient utilizarea limbajului Python și a bibliotecilor sale în crearea unui instrument muzical digital versatil. Prin combinarea tehnicilor de sinteză digitală a sunetului, generarea de forme de undă, implementarea anvelopelor ADSR și a efectelor de delay, proiectul a oferit o experiență de utilizare plăcută și interactivă. Interfața grafică dezvoltată cu Tkinter a facilitat interacțiunea utilizatorului cu sintetizatorul, permițând controlul ușor al parametrilor de sunet și redarea secvențelor muzicale. Implementarea paralelismului a permis generarea simultană a mai multor sunete, contribuind la realismul instrumentului.

Posibile îmbunătățiri pot include îmbunătățirea performanței: deși paralelismul a fost implementat, există potențial pentru optimizarea performanței, în special în gestionarea eficientă a resurselor în timpul generării simultane a mai multor sunete. Utilizarea unor tehnici avansate de programare asincronă ar putea îmbunătăți răspunsul și eficiența aplicației.

De asemenea, proiectul ar beneficia de o gamă mai largă de mostre audio și forme de undă, inclusiv sunete de instrumente mai diverse și efecte sonore unice, iar îmbunătățirea procesării acestora prin eliminarea artefactelor sonore și asigurarea unei tranziții netede între note, ar crește realismul și calitatea audio a sintetizatorului.

Nu în ultimul rând, interfața grafică ar putea fi îmbunătățită pentru a oferi o experiență mai intuitivă și vizual atractivă. Implementarea unui design mai modern și a unor elemente interactive mai avansate ar crește atractivitatea și ușurința de utilizare a sintetizatorului, iar corectarea unor detalii (oprirea scrierii în căsuța pentru secvența notelor după ce utilizatorul a dat click altundeva, posibilitatea de resetare a efectelor de ADSR și de ecou etc.) ar fi bine venită.

În continuare, introducerea unor funcții de editare mai avansate, cum ar fi un editor de secvențe muzicale, posibilitatea de a salva și încărca mostre muzicale, sau un mixer integrat pentru ajustarea nivelurilor de sunet și balans, ar adăuga valoare semnificativă proiectului, la fel ca posibilitatea de a conecta sintetizatorul la un controler MIDI extern, permițând utilizatorilor să folosească claviaturi fizice sau alte dispozitive MIDI pentru a controla sintetizatorul. În plus, implementarea altor forme de sinteză, cum ar fi sinteza granulară sau wavetable, ar putea oferi utilizatorilor mai multe opțiuni pentru crearea de sunete unice și personalizate, inclusiv de generare de muzică nouă bazată pe un input muzical.

În concluzie, proiectul a fost de succes, și au fost prezentate modalități diferite de generare de sunet muzical cu ajutorul limbajului Python și a unor mostre muzicale de la instrumente reale.

# Bibliografie

|  |  |
| --- | --- |
| [1] | Tiobe, "TIOBE Index for January 2024," 2024. [Online]. Available: https://www.tiobe.com/tiobe-index/. |
| [2] | C. R. A. R. S. D. D. E. K. S. M. N. Jesse Engel, "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," in *Internation Conference of Machine Learning*, 2017. |
| [3] | M. N. M. Samridha Kumar, "Synthesizing musical notes of Harmonium using Spectral Domain Modeling," *International Journal of Scientific Engineering and Applied Science (IJSEAS),* vol. 1, 2015. |
| [4] | A. Miraglia, "VCF – VCA & Waveforms," [Online]. Available: https://www.alessiomiraglia.com/vcf-vca-waveforms/. [Accessed 15 01 2024]. |
| [5] | J. R. Jennifer Burg, "Digital Sound & Music," [Online]. Available: https://digitalsoundandmusic.com/. [Accessed 15 01 2024]. |
| [6] | M. Lavengood, "What Makes It Sound ’80s?: The Yamaha DX7 Electric Piano Sound," *Journal of Popular Music Studies,* vol. 31, no. 3, pp. 73-94, 2019. |
| [7] | Making Music, "Quick Guide to Envelopes," [Online]. Available: https://making-music.com/quick-guides/envelopes/. [Accessed 15 01 2024]. |
| [8] | T. T. VESA VÄLIMÄKI, "Virtual musical instruments — natural sound using physical models," *Organised Sound,* vol. 1, no. 2, pp. 75-86, 1996. |
| [9] | W. X. X. T. Q. L. Y. G. Zhen Ye, "NAS-FM: Neural Architecture Search for Tunable and Interpretable Sound Synthesis based on Frequency Modulation," *International Joint Conference on Artificial Intelligence,* 2023. |
| [10] | Wikipedia, "Super Mario Bros. theme," [Online]. Available: https://en.wikipedia.org/wiki/Super\_Mario\_Bros.\_theme. [Accessed 15 01 2024]. |
| [11] | Doom Wiki, "Doom (Original Game Soundtrack)," [Online]. Available: https://doomwiki.org/wiki/Doom\_(Original\_Game\_Soundtrack). [Accessed 15 01 2024]. |
| [12] | P. L. Fritts, "University of Iowa Electronic Music Studios," University of Iowa , [Online]. Available: https://theremin.music.uiowa.edu/index.html. [Accessed 15 01 2024]. |
| [13] | M. R. Shailesh Ramamurthy, "Filter Design for Synthesis of Musical Notes: A Multidimensional featurebased approach," in *IEEE ICSIPA 2013*, Melaka, Malaysia, 2013. |
| [14] | S. K. S. D. P. S. Avani Dharne, "OPUS-An Android Based Speech to Musical Notes Converter," *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT),* vol. 4, no. 4, 2015. |
| [15] | V. H. Valerii Kozlin, "Musical Projects in Guitar Pro 7.5 Sequencer," 12 2023. [Online]. Available: https://www.researchgate.net/publication/376744758\_Musical\_Projects\_in\_Guitar\_Pro\_75\_Sequencer. [Accessed 15 01 2024]. |
| [16] | G. Essl, "Topology in Sound Synthesis and Digital Signal Processing -- DAFx2022 Lecture Notes," in *DAFx2022*, Vienna, Austria, 2022. |
| [17] | hamiltron, "py-simple-audio," github, [Online]. Available: https://github.com/hamiltron/py-simple-audio. [Accessed 15 01 2024]. |
| [18] | Geeks for Geeks website, "https://www.geeksforgeeks.org/play-sound-in-python/," Geeks for Geeks, [Online]. Available: https://www.geeksforgeeks.org/play-sound-in-python/. [Accessed 15 01 2024]. |
| [19] | P. Kazarinoff, "Plotting sine and cosine with Matplotlib and Python," Python for Undergraduate Engineers, 05 02 2018. [Online]. Available: https://pythonforundergradengineers.com/plotting-sin-cos-with-matplotlib.html. [Accessed 15 01 2024]. |

# Anexă

Cod SynthApp.py:

# Main GUI Application

import tkinter as tk

import tkinter.messagebox

import threading

import time

from synth import pre\_generate\_waveforms, play\_waveform, generate\_waveform, load\_piano\_samples, DURATION, NOTE\_FREQUENCIES

# Define frequencies for one octave

WHITE\_KEYS = ["C", "D", "E", "F", "G", "A", "B"]

BLACK\_KEYS = ["C#", "D#", "F#", "G#", "A#"]

NOTE\_FREQUENCIES = { # Octave frequencies

"C": 261.63, "C#": 277.18, "D": 293.66, "D#": 311.13,

"E": 329.63, "F": 349.23, "F#": 369.99, "G": 392.00,

"G#": 415.30, "A": 440.00, "A#": 466.16, "B": 493.88

}

# GUI Application

class SynthApp(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("SynthSim")

self.geometry('1000x300')

# Pre-generate waveforms for each note and waveform type

self.waveforms = {waveform: pre\_generate\_waveforms(NOTE\_FREQUENCIES.values(), waveform)

for waveform in ["sine", "sine-square", "square"]}

# Create piano keys

self.create\_piano\_keys()

# Create sound effects options

self.create\_sound\_effects()

# Create a text entry for the note progression with a default message

self.note\_entry = tk.Entry(self)

self.note\_entry.insert(0, 'Insert notes to play progression')

self.note\_entry.place(x=0, y=200, width=200)

# Bind the focus\_in event to clear the default text

self.note\_entry.bind("<FocusIn>", self.clear\_text)

# Bind the focus\_out event to stop taking input

self.note\_entry.bind("<FocusOut>", self.stop\_input)

# Bind the keyboard to the note\_entry widget

self.bind\_keyboard(self.note\_entry)

# Create a button to play the note progression

self.play\_button = tk.Button(self, text="Play", command=self.play\_progression)

self.play\_button.place(x=210, y=200)

# FM Synthesis Parameters

self.fm\_mod\_freq = tk.DoubleVar(value=220) # Default modulator frequency

self.fm\_mod\_index = tk.DoubleVar(value=1) # Default modulation index

# FM Controls

fm\_mod\_freq\_slider = tk.Scale(self, from\_=0, to=1000, resolution=1, orient='horizontal', variable=self.fm\_mod\_freq, label="FM Mod Freq")

fm\_mod\_freq\_slider.place(x=475, y=180)

fm\_mod\_index\_slider = tk.Scale(self, from\_=0, to=10, resolution=0.1, orient='horizontal', variable=self.fm\_mod\_index, label="FM Mod Ind")

fm\_mod\_index\_slider.place(x=580, y=180)

def create\_piano\_keys(self):

white\_key\_width = 100 # Width of the white keys

black\_key\_width = white\_key\_width / 2 # Width of the black keys

white\_key\_index = 0

# Create white keys with labels positioned lower

for key in WHITE\_KEYS:

freq = NOTE\_FREQUENCIES[key]

btn = tk.Button(self, text=key, bg='white', fg='black', anchor='s',

command=lambda f=freq: self.play\_sound(f))

btn.place(x=white\_key\_index \* white\_key\_width, y=0,

width=white\_key\_width, height=180)

white\_key\_index += 1

# Create black keys with visible labels

black\_key\_index = 0

for key in BLACK\_KEYS:

freq = NOTE\_FREQUENCIES[key]

btn = tk.Button(self, text=key, bg='black', fg='white',

command=lambda f=freq: self.play\_sound(f))

# Offset each black key to be between the white keys

offset = (white\_key\_width - black\_key\_width) / 2

btn.place(x=offset + black\_key\_index \* white\_key\_width, y=0,

width=black\_key\_width, height=120)

black\_key\_index += 1

if key == "D#": # Skip the gap between E and F

black\_key\_index += 1

def create\_sound\_effects(self):

self.effect\_var = tk.StringVar(value="sine")

effects\_frame = tk.Frame(self)

effects\_frame.place(x=0, y=180) # Position the effects below the keys

effects = ["sine", "square", "sine-square", "flute", "piano", "trumpet", "fm"]

for effect in effects:

rb = tk.Radiobutton(effects\_frame, text=effect, variable=self.effect\_var, value=effect)

rb.pack(side=tk.LEFT)

# ADSR controls

self.adsr\_values = {

'attack': tk.DoubleVar(value=0.01),

'decay': tk.DoubleVar(value=0.1),

'sustain': tk.DoubleVar(value=0.7),

'release': tk.DoubleVar(value=0.2)

}

for i, (name, var) in enumerate(self.adsr\_values.items()):

tk.Label(self, text=name).place(x=800, y=30\*i)

tk.Scale(self, from\_=0, to=1, resolution=0.01, orient='horizontal', variable=var).place(x=850, y=30\*i)

# Delay controls

self.delay\_values = {

'time': tk.DoubleVar(value=0),

'feedback': tk.DoubleVar(value=0),

'mix': tk.DoubleVar(value=0)

}

for i, (name, var) in enumerate(self.delay\_values.items()):

tk.Label(self, text=name).place(x=800, y=120+30\*i)

tk.Scale(self, from\_=0, to=1, resolution=0.01, orient='horizontal', variable=var).place(x=850, y=120+30\*i)

def play\_sound(self, freq):

# Getting values from the GUI

waveform\_type = self.effect\_var.get()

adsr\_params = (

self.adsr\_values['attack'].get(),

self.adsr\_values['decay'].get(),

self.adsr\_values['sustain'].get(),

self.adsr\_values['release'].get()

)

delay\_params = (

self.delay\_values['time'].get(),

self.delay\_values['feedback'].get(),

self.delay\_values['mix'].get()

)

duration = DURATION # Duration of the note in seconds

# Handle instrument samples separately

if waveform\_type == 'piano':

waveform = generate\_waveform(freq, waveform\_type, piano\_samples=load\_piano\_samples('samples\\piano'))

elif waveform\_type == 'flute':

waveform = generate\_waveform(freq, waveform\_type, piano\_samples=load\_piano\_samples('samples\\flute'))

elif waveform\_type == 'trumpet':

waveform = generate\_waveform(freq, waveform\_type, piano\_samples=load\_piano\_samples('samples\\trumpet'))

elif waveform\_type == 'fm':

fm\_params = (self.fm\_mod\_freq.get(), self.fm\_mod\_index.get())

waveform = generate\_waveform(freq, waveform\_type, duration=DURATION, adsr\_params=adsr\_params, fm\_params=fm\_params)

else:

waveform = generate\_waveform(freq, waveform\_type, duration=DURATION,

adsr\_params=adsr\_params, delay\_params=delay\_params)

# Play the waveform in a separate thread

threading.Thread(target=lambda: play\_waveform(waveform)).start()

def bind\_keyboard(self, widget):

# Bind the keyboard to the widget

keyboard\_to\_piano = {

'a': 'C', 'w': 'C#', 's': 'D', 'e': 'D#', 'd': 'E', 'f': 'F', 't': 'F#',

'g': 'G', 'y': 'G#', 'h': 'A', 'u': 'A#', 'j': 'B'

}

for key, note in keyboard\_to\_piano.items():

self.bind(f"<KeyPress-{key}>", lambda event, f=NOTE\_FREQUENCIES[note]: self.play\_sound(f))

def clear\_text(self, event):

# Clear the text in the entry when it receives focus

self.note\_entry.delete(0, 'end')

def stop\_input(self, event):

# Unbind the key press events when the entry loses focus

self.note\_entry.unbind("<KeyPress>")

def play\_progression(self):

# Get the note progression from the entry

progression = self.note\_entry.get().split()

# Validate all the notes before playing

for note in progression:

# Check if the note is valid

if note not in NOTE\_FREQUENCIES:

# If not, show an error message and return

tkinter.messagebox.showerror("Error", "Illegal note inserted. Please respect the musical notations")

return

# Play each note in the progression

for note in progression:

# If the note is valid, play it

self.play\_sound(NOTE\_FREQUENCIES[note])

# Pause for a short duration before playing the next note

time.sleep(0.5) # Adjusting this values changes the speed of the progression

if \_\_name\_\_ == "\_\_main\_\_":

app = SynthApp()

app.mainloop()

Cod synth.py:

# Backend for the synthesizer app

import simpleaudio as sa

import numpy as np

import wave

import os

import logging

# Constants

SAMPLE\_RATE = 44100 # Sampling rate in Hz

DURATION = 0.5 # Duration of notes in seconds

# Define frequencies for one octave

NOTE\_FREQUENCIES = {

"C": 261.63, "C#": 277.18, "D": 293.66, "D#": 311.13,

"E": 329.63, "F": 349.23, "F#": 369.99, "G": 392.00,

"G#": 415.30, "A": 440.00, "A#": 466.16, "B": 493.88

}

# Setup for logging, debugging purposes

logging.basicConfig(level=logging.DEBUG)

# Function to load piano samples

def load\_piano\_samples(sample\_directory):

try:

samples = {}

for note, freq in NOTE\_FREQUENCIES.items():

file\_path = os.path.join(sample\_directory, f"{freq:.2f}.wav")

with wave.open(file\_path, 'rb') as wave\_file:

frames = wave\_file.readframes(wave\_file.getnframes())

samples[freq] = np.frombuffer(frames, dtype=np.int16) / (2\*\*15)

return samples

except Exception as e:

print(f"Error loading piano samples: {e}")

return None

# Generate waveforms

def generate\_waveform(freq, waveform\_type='sine', duration=DURATION, adsr\_params=None, delay\_params=None, piano\_samples=None, fm\_params=None):

# Debug message

logging.debug(f"Generating waveform: freq={freq}, type={waveform\_type}")

# Initialize waveform to sine by default

t = np.linspace(0, duration, int(SAMPLE\_RATE \* duration), endpoint=False)

waveform = np.sin(2 \* np.pi \* freq \* t)

# Generate waveform based on type

if waveform\_type == 'piano':

print("Piano samples in play\_sound:", piano\_samples)

if freq in piano\_samples:

waveform = piano\_samples[freq]

sample\_length = len(waveform) / SAMPLE\_RATE

if sample\_length < duration:

repeats = int(np.ceil(duration / sample\_length))

waveform = np.tile(waveform, repeats)[:int(SAMPLE\_RATE \* duration)]

else:

waveform = waveform[:int(SAMPLE\_RATE \* duration)]

else:

raise ValueError(f"No piano sample for frequency {freq}")

elif waveform\_type == 'flute':

print("Flute samples in play\_sound:", piano\_samples)

if freq in piano\_samples:

waveform = piano\_samples[freq]

sample\_length = len(waveform) / SAMPLE\_RATE

if sample\_length < duration:

repeats = int(np.ceil(duration / sample\_length))

waveform = np.tile(waveform, repeats)[:int(SAMPLE\_RATE \* duration)]

else:

waveform = waveform[:int(SAMPLE\_RATE \* duration)]

else:

raise ValueError(f"No flute sample for frequency {freq}")

elif waveform\_type == 'trumpet':

print("Trumpet samples in play\_sound:", piano\_samples)

if freq in piano\_samples:

waveform = piano\_samples[freq]

sample\_length = len(waveform) / SAMPLE\_RATE

if sample\_length < duration:

repeats = int(np.ceil(duration / sample\_length))

waveform = np.tile(waveform, repeats)[:int(SAMPLE\_RATE \* duration)]

else:

waveform = waveform[:int(SAMPLE\_RATE \* duration)]

else:

raise ValueError(f"No trumpet sample for frequency {freq}")

elif waveform\_type == 'fm':

carrier\_freq = freq

if waveform\_type == 'fm':

if fm\_params is None:

fm\_params = (220, 1) # Default values for FM modulation

modulator\_freq, modulation\_index = fm\_params

modulator = np.sin(2 \* np.pi \* modulator\_freq \* t)

waveform = np.sin(2 \* np.pi \* freq \* t + modulation\_index \* modulator)

else:

t = np.linspace(0, duration, int(SAMPLE\_RATE \* duration), endpoint=False)

if waveform\_type == 'sine':

waveform = np.sin(2 \* np.pi \* freq \* t)

elif waveform\_type == 'square':

waveform = np.sign(np.sin(2 \* np.pi \* freq \* t))

elif waveform\_type == 'sine-square':

# Placeholder for a combination of sine and square waves

waveform = np.sin(2 \* np.pi \* freq \* t) \* (np.sign(np.sin(2 \* np.pi \* freq \* t)) + 1)

# Apply ADSR envelope if parameters are provided

if adsr\_params and waveform\_type != 'piano':

attack, decay, sustain, release = adsr\_params

waveform = apply\_adsr\_envelope(waveform, attack, decay, sustain, release)

# Apply delay effect if parameters are provided

if delay\_params and waveform\_type != 'piano':

delay\_time, feedback, mix = delay\_params

waveform = apply\_delay(waveform, delay\_time, feedback, mix)

if waveform is None:

logging.warning(f"Waveform type '{waveform\_type}' not recognized. Returning silence.")

waveform = np.zeros(int(SAMPLE\_RATE \* duration)) # Return silence if type not recognized

return waveform

# ADSR Envelope creation

def apply\_adsr\_envelope(waveform, attack, decay, sustain\_level, release):

# Ensure attack, decay, and release are non-zero to avoid divide-by-zero issues

attack = max(attack, 1.0 / SAMPLE\_RATE)

decay = max(decay, 1.0 / SAMPLE\_RATE)

release = max(release, 1.0 / SAMPLE\_RATE)

# Calculate lengths of attack, decay, and release in samples

attack\_length = int(attack \* SAMPLE\_RATE)

decay\_length = int(decay \* SAMPLE\_RATE)

release\_length = int(release \* SAMPLE\_RATE)

# Calculate the sustain length to make sure the envelope fits the waveform length

sustain\_length = len(waveform) - attack\_length - decay\_length - release\_length

if sustain\_length < 0:

sustain\_length = 0

release\_length = len(waveform) - attack\_length - decay\_length

if release\_length < 0:

decay\_length = len(waveform) - attack\_length

if decay\_length < 0:

attack\_length = len(waveform)

decay\_length = 0

release\_length = 0

# Create the envelope

envelope = np.concatenate([

np.linspace(0, 1, attack\_length), # Attack

np.linspace(1, sustain\_level, decay\_length), # Decay

np.ones(sustain\_length) \* sustain\_level, # Sustain

np.linspace(sustain\_level, 0, release\_length) # Release

])

# Ensure the envelope is not longer than the waveform

envelope = envelope[:len(waveform)]

# Apply envelope to the waveform

return waveform \* envelope

# Delay Effect Function

def apply\_delay(waveform, delay\_time, feedback, mix):

delay\_samples = int(delay\_time \* SAMPLE\_RATE)

delayed\_waveform = np.zeros(len(waveform) + delay\_samples)

delayed\_waveform[:len(waveform)] = waveform

for i in range(len(waveform)):

delayed\_waveform[i + delay\_samples] += delayed\_waveform[i] \* feedback

output = delayed\_waveform[:len(waveform)] \* mix + waveform \* (1 - mix)

return output

# Play waveform

def play\_waveform(waveform):

max\_val = np.max(np.abs(waveform))

if max\_val > 0:

audio = waveform \* (2\*\*15 - 1) / max\_val

audio = audio.astype(np.int16)

play\_obj = sa.play\_buffer(audio, 1, 2, SAMPLE\_RATE)

play\_obj.wait\_done()

else:

# Handle the case where there is no sound to play

print("No audio to play: waveform is silent.")

# Pre-generate waveforms for a set of frequencies

def pre\_generate\_waveforms(frequencies, waveform\_type='sine', adsr=(0.01, 0.1, 0.7, 0.2), delay=(0, 0, 0)):

waveforms = {}

for freq in frequencies:

waveform = generate\_waveform(freq, waveform\_type)

waveform = apply\_adsr\_envelope(waveform, \*adsr)

waveform = apply\_delay(waveform, \*delay)

waveforms[freq] = waveform

return waveforms