RELAZIONE PROGETTO MINIMOOG

INFORMATICA APPLICATA AL SUONO 2022-2023

Stighezza Valeria *976853*

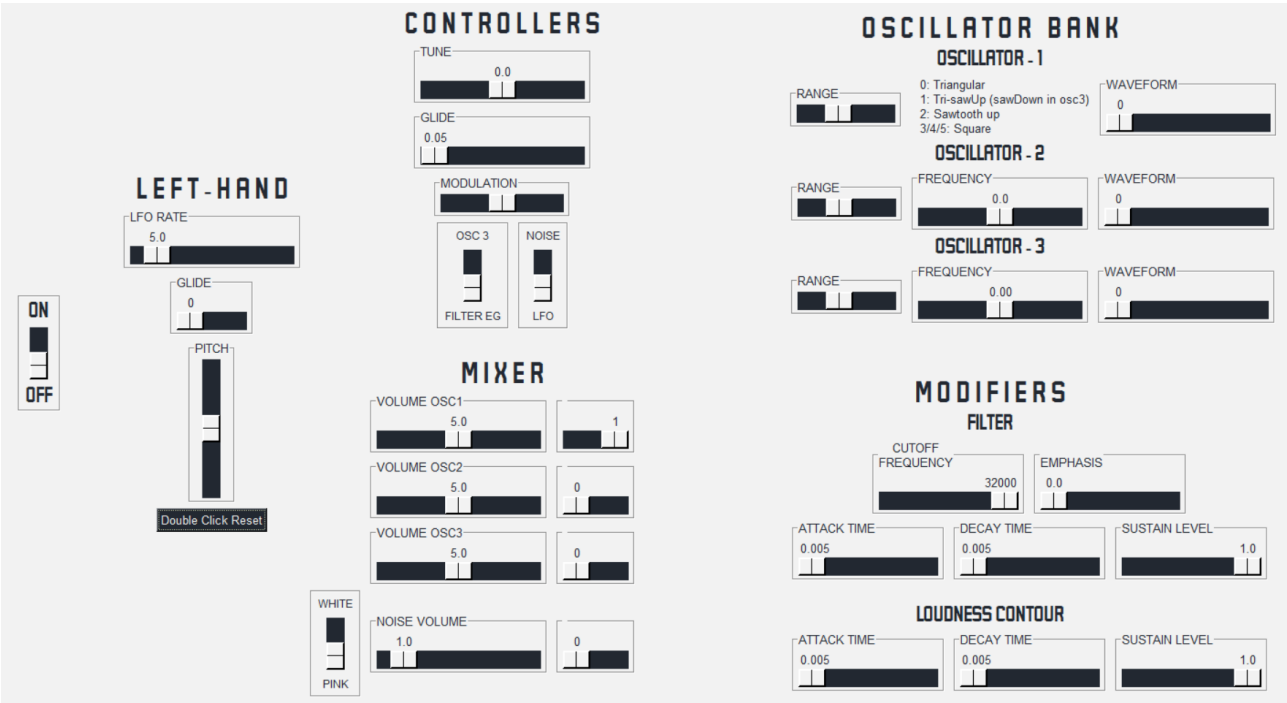
Massimo Davide *962719*

Il Minimoog è un sintetizzatore analogico monofonico: l'obiettivo di questo progetto è riprodurne una versione digitale sfruttando le funzionalità della libreria *Pyo* di Python.

Il cuore del codice, il main, si trova nel file *Minimoog.py*, il quale fa uso delle funzioni contenute in *functions.py* e delle classi da noi definite, appartenenti alla cartella *classes*.

La GUI

Il punto di partenza è stato creare una GUI apposita, così da poter prendere in input i vari parametri in maniera ordinata e riuscire a elaborarli in tempo reale.



La GUI è stata implementata come una classe iterabile chiamata *GuiLayout()* e poggia le sue basi su *PySimpleGui*, libreria che permette di creare interfacce grafiche Python.

È strutturata in 4 colonne, le quali contengono i seguenti moduli:

1) Tasto on/off
2) Left-hand
3) Controllers e mixer
4) Oscillator bank e modifiers

Ogni modulo è definito da una serie di slider binari o multivalore, ognuno dei quali ha abbinato un campo *key* che permette l'identificazione e l'utilizzo del singolo slider.

Per ogni slider è definibile il titolo, il range di valori su cui operare, la risoluzione di tali valori, la grandezza, l'orientamento (orizzontale o verticale), la possibilità (o non) di abilitare gli eventi e di vedere a stampa i valori selezionati

Il codice principale

Analizziamo lo script *Minimoog.py.*

Come prima cosa viene inizializzato il **server** di Pyo. La porta 99 in input del server di pyo indica che stiamo abilitando tutte le porte MIDI.

Il secondo passo è importare la definizione delle le **forme d'onda** (che verranno poi assegnate agli oscillatori) con la funzione *import_waves(),* la quale*:*

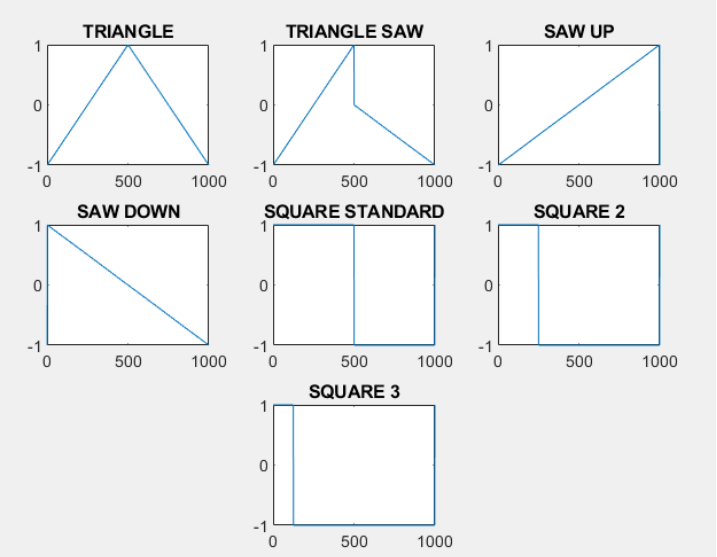
1. Trova la working directory, dalla quale parte un cammino all'interno dell'albero delle sotto-directory e dei file contenuti per cercare i file con estensione *.txt*
2. I file trovati contengono i campioni delle forme d'onda sotto forma di stringa: li apre e converte il testo in liste di numeri float, le quali vengono memorizzate in *wavetable*
3. Estrae i nomi dei file, cioè delle forme d'onda, dall'intero percorso (quindi escludendo il percorso prima del nome e l'estensione) così da poter usare questi nomi come campo chiave del dizionario *waves*
4. I valori memorizzati in *wavetable*, cioè le liste di float che saranno abbinate ai valori chiave di *waves*, vengono infine convertiti in DataTable, così che Pyo possa usarle per creare segnali periodici da assegnare come forma d'onda agli oscillatori

Da qui abbiamo costruito le liste *wavelist_12* e *wavelist_3* contenenti le tutte le forme d'onda richieste per gli oscillatori.

Le forme d'onda le sono state definite interamente su Matlab e importate su python col metodo decritto sopra.

Oltre alle classiche forme d'onda, abbiamo ricreato anche quella del Minimoog, ovvero la "*triangle saw*", la quale, sull'asse delle x:

- Si comporta come un'onda triangolare da 0 a 0.5 (metà sinistra)
- Si comporta come un'onda a dente di sega all'ingiù da 0.5 a 1 (metà destra)



Seguono poi l'inizializzazione

- di **variabili** che verranno usate successivamente;
- della **GUI** come oggetto della classe *GuiLayout()*, contenuta nel file *guiClass* di cui abbiamo parlato precedentemente;
- del lato **MIDI**
    - *poly=1* indica che il synth è monofonico,
    - *scale=1* indica che la nota MIDI verrà trattata con il suo valore in hertz,
    - il metodo *setStealing()* permette di implementare la monofonia con la modalità *stealing*, in modo tale che la nota che viene premuta sulla tastiera sia quella che viene suonata, anche nel caso in cui venga premuta sovrapponendosi a un'altra nota che si stava già premendo in precedenza;
- degli **ADSR** degli oscillatori, che tengono conto della *velocity*, ovvero dell'intensità con cui la nota viene premuta;
- degli **oscillatori**: sono 3, chiamati rispettivamente *osc1, osc2* e *osc3* e sono della classe *ModOsc()* , figlia di *Osc(),* creata appositamente per ampliare le funzionalità della classe padre (aggiunta dei metodi *getFreq()* e *getMul()*). Prendono come frequenza in input la variabile *Glide*, che grazie a *port()* permette di andare da una nota all'altra in modo continuo, facendo uno slide delle frequenze dalla frequenza della prima nota fino alla frequenza della seconda nota;
- dei **generatori di rumore**: *whiteN* per il rumore bianco e *pinkN* per quello rosa;
- delle variabili **mix** che mixano i flussi audio degli oscillatori in una singola voce (*mix2filter*), la quale entra in input a un **filtro** passa-basso risonante di 4° ordine (*LPfilt).* il filtro è un oggetto della classe *MMoogLP()*, figlia di *MoogLP()* ma con il metodo *getMul()* da noi aggiunto;
- della **LFO**, utilizzabile come modulante.

Finita la fase di inizializzazione comincia il **ciclo principale**, che termina solamente una volta che il sintetizzatore viene spento. Ad ogni iterazione processa gli eventi che si verificano e in base ai valori in input (i click del mouse) modifica il suo comportamento e agisce di conseguenza, vediamo come.

Innanzitutto controlla che il sintetizzatore sia acceso, altrimenti non sarà disponibile nessuna funzionalità.

Se è accesso, definisce i parametri da assegnare agli oggetti, partendo dal **Glide**: controlla che lo switch on/off del glide sul ***left-hand*** sia settato su 1 (on) e, se è così, si può usare lo slider "GLIDE" dei ***controllers*** per impostare *risetime* e *falltime*.

Se invece lo switch on/off è settato su 0 (off), vengono impostati *risetime* e *falltime* uguali a 0. Nel codice, ciò è gestito dalla funzione *setGlide()* e coinvolge tutti e 3 gli oscillatori.

Rimanendo sul left-hand, passiamo alla rotella (slider) del **pitch**: a ogni iterazione del ciclo principale viene salvato il valore della rotella in quell'istante nella variabile *tmp*, cosicchè possa essere usata nell'iterazione successiva per calcolare la differenza dello spostamento dello slider tra le due iterazioni.

Questa differenza infatti ci è comoda per adattare la frequenza degli oscillatori a ogni movimento della rotella: si moltiplica/divide la frequenza dell'oscillatore per $\mathbf{2}^{\pm differenza}$**.**

Utilizziamo la base 2 perché vogliamo che con la rotella si possa salire/scendere al massimo di un'ottava, cioè moltiplicare/dividere per 2. Ponendo [-1, 1] come range dello slider, raggiungiamo questo obiettivo (la nostra frequenza raggiungerà al massimo/minimo i valori $2^1 = 2 \;\; e \;\; 2^{-1} = \frac{1}{2}$)

Una volta usata la rotella del pitch, possiamo riportarne il valore a 0 facendo doppio click sul bottone sotto.

Ci spostiamo sull' ***oscillator bank***, dove vengono gestiti i 3 oscillatori. In particolare, vengono impostati:

- I **range** dei singoli oscillatori: con la funzione *setRange()* si sfrutta ancora la differenza dei valori degli slider tra le iterazioni, la quale viene usata come esponente al 2, indicando quante volte si debba moltiplicare/dividere per 2, cioè quanti cambi di ottava eseguire. Con la funzione *upadetOctave()* aggiorniamo i numeri stampati a schermo, semplicemente per mostrare sulla gui i valori sottoforma di potenze di 2, cioè [64, 32, 16, 8, 4, 2] come nel vero Minimoog anziché [6, 5, 4, 3, 2, 1] come nello slider da noi definito

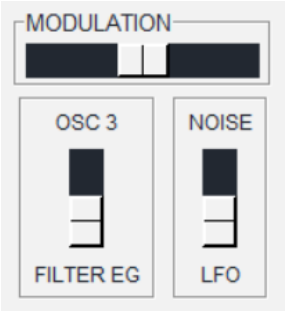
- la **frequenza** degli oscillatori 2 e 3, che con la funzione *setFrequency()* si comporta in maniera molto simile a quella del range, solo che questa volta non dobbiamo salire/scendere di una o più ottave, ma di un certo numero di semitoni. Dato che 1 semitono corrisponde a $\sqrt[12]{2} = 1.059$, nel calcolo da fare per adattare la frequenza degli oscillatori allo spostamento degli slider bisogna mettere come base 1.059. Quindi moltiplichiamo/dividiamo la frequenza per $1.059^{\pm differenza}$

- Il **tune** è gestito allo stesso modo della frequenza, con la differenza che coinvolge tutti e 3 gli oscillatori allo stesso tempo

- Le **forme d'onda** di tutti e 3 gli oscillatori, dove ogni forma d'onda è un elemento di *wavelist_12* o *wavelist_3*

Nel ***mixer*** vengono gestiti i **volumi** dei 3 oscillatori e dei generatori di rumore bianco/rosa . Lo abbiamo fatto con la funzione *setVolume(),* la quale agisce sul *mul* degli elementi, alzandolo o abbassandolo. Se un oscillatore/generatore di rumore viene spento, il suo *mul* viene azzerato.

La parte della **modulazione** è gestita nei ***controllers*** e coinvolge tutte le altre parti dell'interfacia grafica.



Lo slider "*modulation"* può assumere solo i valori {0, 1, 2} e a seconda di ciò cambia la sorgente modulante:

- se il valore è **1**, cioè lo slider è posizionato in centro, allora NON è richiesto alcun tipo di modulazione
- se il valore è **0,** cioè lo slider è posizionato a sinistra, possiamo modulare in due possibili modi: con l'oscillatore 3 o con il filter EG
    - la modulazione con l'oscillatore 3 è stata implementata moltiplicando l'oscillatore 3 e il filtro applicato al mix degli oscillatori 1 e 2
    - nella modulzione con il filtro EG, la forma definita dagli slider *ATTACK TIME, DECAY TIME e SUSTAIN LEVEL* di *Filter Contour* sono usati come sorgente di modulazione
- se il valore è **2**, cioè lo slider è posizionato a destra, possiamo anche qui modulare in due possibili modi: con il rumore o con una LFO
    - Se decidiamo di modulare con l'LFO , questa viene appositamente creata e moltiplicata per *LPfilt*, cioè il filtro applicato al mix (degli oscillatori 1,2 e 3 e dei generatori di rumore). Quando la modulazione viene fermata tramite GUI, l'oscillatore dell'LFO viene messo in pausa col metodo *stop().*
      La frequenza dell'LFO può essere modificata tramite lo slider *LFO RATE* nel *left-hand*
    - Per modulare con il rumore selezionato (bianco o rosa), lo moltiplichiamo per il filtro passabasso applicato al mix dei 3 oscillatori.

Parte importante è stata infine gestire i valori degli oggetti dopo la modulazione, in modo tale che cambiando sorgente modulante o togliendola si abbia l'effetto desiderato senza interferenze causate da vecchie modulazioni.

Infine, i ***modifiers*****,** composti da 2 parti

- **Filters :** agiscono sul filtro che può essere usato come sorgente modulante. Il filtro ha un suo ***ADSR*** che viene anche applicato alla ***frequenza di cutoff***, così da rendersi "dinamico": quando siamo nel punto di attacco, la frequenza raggiungerà il range massimo nell'inviluppo.
  Il filtro ha inoltre uno slider per l'***emphasis***, che prende una parte dell'uscita del filtro e lo rimanda all'ingresso del filtro, creando un picco di risonanza che si verifica alla frequenza di taglio.
- **Loudness contour**: serve a impostare l'ADSR dell'output generale. Tutti gli ADSR sono stati settati con la funzione *setADSR().*

Quando il sintetizzatore viene spento si esce dal ciclo *while* e la finestra della la gui si chiude.