Informática Musical

Sintetizador y secuenciadores

Los ejercicios marcados con **(Evaluable)** son prácticas de laboratorio evaluables. Deben seleccionarse **al menos 2**, desarrollarlos y subirlos al CV. Todos los archivos entregados tienen que contener el nombre y los apellidos de los miembros del grupo (uno por línea).

Los ejercicios se apoyan en el Jupypter Notebook sobre síntesis y secuenciación del CV.

1. **(Evaluable)** Extender el instrumento FM presentado en clase de modo que permita seleccionar el tipo de onda (sinosuoidal, cuadrada, triangular, diente de sierra) tanto para el carrier como para la moduladora. Utilizar dos *ComboBox* de TkInter para mostrar las opciones para cada entrada.

2. **(Evaluable)** Extender el instrumento FM presentado en clase con un nuevo parámetro *bm* (beta modulation) que module dinámicamente el valor de *beta* en un pequeño rango de valores mediante un oscilador sinusoidal. Añadir los sliders correspondientes para la frecuencia y cantidad de modulación de este parámetro

3. **(Evaluable)** El *theremin* es un instrumento electrónico basado en osciladores, que consta de dos antenas metálicas que detectan la posición relativa de las manos del intérprete (*thereminista*). Esa posición determina la frecuencia y la amplitud del tono (la frecuencia se controla con una mano y la amplitud (volumen) con la otra). Puede verse un vídeo en `https://www.youtube.com/watch?v=K6KbEnGnymk`

   En este ejercicio se implementará un *theremin* con el modelo de síntesis *(lookup) wavetable* visto en clase (con este modelo es fácil evitar los pops debidos a los cambios de frecuencia/volumen). Las antenas se simularán con el ratón: la posición horizontal determina la frecuencia (en un rango prefijado, como [100,1500]) y la vertical, la amplitud en [0,1].

   El instrumento debe incluir un pequeño interfaz gráfico con TkInter, que permite detectar fácilmente la posición del ratón en en pantalla:

```
import tkinter as tk
root = tk.Tk()

def motion(event):
    x, y = event.x, event.y
    print('{}, {}'.format(x, y))

root.bind('<Motion>', motion)
root.mainloop()
```

   Extensiones:

   - Utilizar otros modelos de generación, como la síntesis FM. Deberán evitarse o suavizarse los pops al cambiar la frecuencia/volumen.
   - Incorporar un *autotune* para obtener solo frecuencias de un conjunto determinado (correspondientes a una escala) y utilice la posición del ratón para aproximar la nota más cercana de ese conjunto.

4. **(Evaluable)** En este ejercicio implementaremos un *sintetizador FM armónico*, i.e., un sintetizador que produce acordes (combinaciones de notas simultáneas) en vez de notas aisladas como explicaremos más adelante. Nuestro sintetizador utilizará la *escala diatónica* (las teclas blancas del piano: *c-d-e-f-g-a-b-c*) y podrá utilizar dos tipos de afinación, justa y temperada, según la siguiente tabla:

	A	B	C	D	E	F	G	a
Afinación justa	440	495	550	586.67	660	733.33	825	880
Relaciones	1	9/8	5/4	4/3	3/2	5/3	15/8	2
Afinación temperada	440	493.88	554.36	587.33	659.26	739.99	830.61	880
Relaciones	1	$2^{2/12}$	$2^{3/12}$	$2^{5/12}$	$2^{7/12}$	$2^{8/12}$	$2^{10/12}$	$2^{12/12}=2$

   Recordemos que con esta tabla podemos obtener todas las octavas: podemos subir/bajar una octava multiplicando/dividiendo su frecuencia por 2.

Nuestro sintetizador producirá las *tríadas (acordes) elementales de la escala diatónica* que se forman a partir de una nota cualquiera, añadiendo otras dos por encima *en saltos de 2*. Por ejemplo, a partir de la *nota C* obtenemos el *acorde C* con las notas *C-E-G*, y a partir de la *nota A* obtenemos el *acorde A* con las notas *A-C-E*. Como es habitual, utilizaremos las teclas *zxcvbnm* para una octava (baja) y *qwertyu* para la superior (alta). Cuando se pulse *z* sonará el acorde *C*.

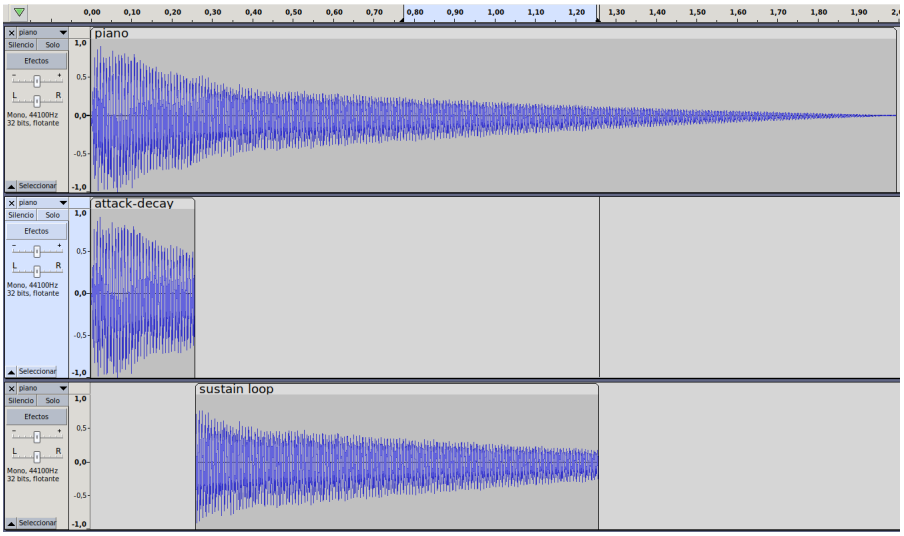
Discusión: ¿Qué afinación suena mejor, la justa o la temperada?

5. **(Evaluable)** Utilizando la filosofía de la clase `Instrument` vista en clase, implementar un instrumento **piano** com el modelo de síntesis de Karplus-Strong. Los chunks del método `next()` se obtendrán a partir del buffer circular que incorpora el algoritmo.

   Para investigar: la afinación se deriva de la longitud (en samples) del buffer de partida, lo que implica que pueden conseguirse frecuencias discretas. ¿Es posible conseguir una afinación más precisa, no restringida a ese número de samples?

6. **(Evaluable)** Exenter el secuenciador MIDI visto en clase de modo que al cargar el arvhivo MIDI cuente el número de canales y genere dinámicamente un instrumento (sintetizador FM) para cada uno de los canales. De este modo podremos configurar el timbre para cada uno de los canales. Será util poder definir presets para los instrumentos (parámetros para el generador FM y la envolvente ADSR) que se puedan guardar y cargar de archivo.

7. (**Evaluable**, difícil) Implementar un *sampler* que permita cargar muestras de audio *preparadas* para procesar en tiempo real. En particular, para una nota de piano como la de la primera pista de abajo, podemos utilizar dos fragmentos. El primero con la parte de ataque y decay, y el segundo con una zona *loopeable* para construir el sustain (la última parte de la muestra inicial la podemos descartar).



El fragmento de sustain debe elegirse adecuadamente para poder construir el loop. Sobre ese fragmento hay que linealizar el volumen y asegurarse de que suena bien cíclicamente.

Estos dos fragmentos serán los que cargue el sampler. El mensaje *noteOn* activará la generación de una nota: se reproduce la región de ataque-decay y luego se reproduce en loop la región de sustain hasta que llegue el mensaje *noteOff*. En ese momento se aplica un release a la región sustain (con tiempo parametrizable) hasta silenciar la nota.

Para definir el pitch de la nota, se tomará el pitch de la nota original y se variará la velocidad de reproducción para conseguir la nota deseada. Puede utilizarse la técnica de wavetable (recorrido a distintas velocidad variando el *stpep* de avance y utilizando interpolación), o bien, remuestreo de señal (función de numpy).