PROJECTE FINAL APA

Editor d'àudio: SoundFix

Gisela León Pipó

Paula Puigdevall Tornero

ÍNDEX DE CONTINGUT

01 INTRODUCCIÓ

SOUND FIX
ENJOY YOUR EDIT

En aquest projecte hem desenvolupat un programa per dur a terme l'edició d'un fitxer d'àudio. El nostre objectiu era proporcionar una eina intuitiva i versàtil que permetés als usuaris manipular i millorar el contingut dels seus fitxers d'àudio de manera eficient i efectiva, així aplicant també el que hem après a l'assignatura.

Es una aplicació senzilla amb les funcionalitats més bàsiques de l'edició d'àudio:

- Convertidor de format
- Segmentació del fitxer
- Efectes d'àudio
- Reproducció de fitxers d'àudio
- Interficie per gestionar l'applicació

02 LLIBRERIES

Per dur a terme el projecte, hem utilitzat diferents llibreries de python enfocades al procesament d'àudio. Aquestes llibreries ens han permes realitzar funcions com la de lectura i escriptura de fitxers d'audio. D'aquesta manera hem pogut desenvolupar funcionalitats especifiques.

- Pydub
- Soundfile
- Sounddevice

Aquestes tres llibreries s'han utilitzat per al processat d'àudio. És a dir, hem utilitzat sounddevice i soundfile per manipular els fitxers d'audió tal que en poguessim fer una lectura i els poguessim reproduir. Per altra banda hem utilitzat pydub per fer els effectes i importar i exportar fitxers.

Per tal d'aconseguir la interficie gràfica d'usuari que hem creat per a la nostre aplicació, hem utilitzat la llibreria tkinter. A part també hem aprofitat la eina ffmpeg que ens ha permès realitzar tasques relacionades amb fitxers multimedia.

- Tkinter
- Ffmpeg

- Numpy
- PIL

Finalment per a les funcions en les que hem necessitat fer càlculs numèrics amb arrays, hem fet servir la llibreria Numpy ja que ens proporciona càlculs eficients. Per altra banda pel que fa a les imatges de la interficie hem utilitzat la llibreria PIL.

PROGRAMA PRINCIPAL

El programa principal del nostre projecte ens proporciona una classe Audio_processor que conté diferents mètodes per al processat d'àudio. Pel que fa a les diferentes funcionalitats del projecte també les hem declarat com a classes amb herencia a la classe audio_processor ja que totes havien d'utilitzar mètodes com la lectura o escriptura de fitxer.

EFECTES D'ÀUDIO

Els diferents efectes que hem creat en el programa són tots i cadascún d'ells una classe amb herencia a la classe audio_processor.

EFECTE ROBOT (APPLY_ROBOT_EFFECT):

El codi utilitza el senyal de modulació i tècniques de manipulació d'àudio per crear un efecte de robot en un fitxer d'àudio. Això s'aconsegueix mitjançant un ajustament de l'amplitud i el to del senyal d'àudio original. Aquests són els passos que segueix:

1. Obtenir el nombre de canals del fitxer d'àudio.
2. Generar el senyal de modulació fent servir un factor de modulació proporcionat.
3. Per cada canal d'àudio, ajustar el senyal de modulació al tamany del canal i es calculen els índexs de mostreig modulats.
4. Aplicar la interpolació lineal per assolir els valors d'àudio modulats.
5. Normalitza el senyal d'àudio modulat.
6. Assignar el senyal d'àudio modulat al canal corresponent del fitxer d'àudio.
7. Finalment, guardar l'arxiu d'àudio modulat amb l'efecte generat a un fitxer de sortida.

EFECTE ECO (APPLY_ECHO_EFFECT):

Aquesta funció utilitza un retard i un decaïment per generar un efecte eco en un fitxer d'àudio, generant repeticions atenuades i retardades del so original. Això afegeix profunditat i ambient a l'àudio, creant una experiència auditiva més interessant.

Els passos a seguir per generar aquest efecte han estat:

1. Calcular la quantitat de mostres de retard en funció del valor de retard i la freqüència de mostreig de l'àudio.
2. Creació d'un *array* buit del mateix tamany que l'àudio original per emmagatzemar l'àudio modificat.
3. Iterar sobre cada mostra de l'àudio original.
4. Si l'índex actual és major o igual al retard, s'aplica l'efecte eco sumant el valor actual de l'àudio original amb una versió atenuada del senyal obtingut retardat en el temps, multiplicat pel valor del decaïment. Això genera la repetició i l'esvaïment del so.
5. Si l'índex actual és menor al retard, es còpia el valor de l'àudio original directament l'àudio modificat sense aplicar-li l'efecte eco.
6. Finalment, es guarda el fitxer d'àudio modificat amb l'efecte eco a un fitxer de sortida.

03 DESENVOLUPAMENT

SOUND FIX
ENJOY YOUR EDIT

EFECTE FLANGER (APPLY_FLANGER_EFFECT):

Aquest efecte utilitza el retard, la profunditat i la freqüència de modulació per generar un efecte *flanger* a un fitxer d'àudio. Això afegeix una oscil·lació o vibració del so, creant un efecte "*chorus*" que proporciona riquesa i textura a l'àudio.

Hem seguit els següents passos per aconseguir-lo:

1. Càlcul de mostres de retard i profunditat en funció dels valors de retard i profunditat i la freqüència de mostreig de l'àudio.
2. Creació d'un *array* buit del mateix tamany que l'àudio original per emmagatzemar l'àudio modificat.
3. Creació d'un modulador que genera un senyal sinusoidal que oscil·la entre *-depth_samples* i *+depth_samples* en funció del temps.
4. Iterar sobre cada mostra de l'àudio original.
5. Si l'índex actual és major o igual al retard, es calcula un índex modulat sumant l'índex actual amb el valor del modulador en aquella posició. Això produeix l'efecte d'oscil·lació.
6. Sumar el valor actual de l'àudio original. Això proporciona l'efecte *flanger*.
7. Si l'índex actual és menor al retardat, es còpia el valor de l'àudio original directament a l'àudio modificat sense aplicar l'efecte *flanger*.
8. Finalment, es guarda el fitxer d'àudio modificat amb l'efecte *flanger* en un fitxer de sortida.

EFECTE PITUFO (APPLY_PITUFO_EFFECT):

Aquest efecte altera la velocitat i el to de l'àudio, fent que soni més agut i ràpid, similar a la veu dels personatges del *pitufos*. Es crea mitjançant un factor de modulació i un factor tonal.

Els següents passos indiquen com fer-ho:

1. Obtenir el nombre de canals de l'àudio.
2. Calcular un modulador que representa un senyal que varia en funció del temps i del factor de modulació.
3. Iterar sobre cada canal de l'àudio.
4. Assolir el canal actual com un arranjament unidimensional.
5. Ajustar la mida del modulador al tamany del canal actual.
6. Calcular els índexs de mostreig modulats sumant els índexs originals amb el modulador multiplicat pel factor tonal i la freqüència de mostreig.
7. Aplicar la interpolació lineal per aconseguir els valors d'àudio modulats en funció dels índexs de mostreig modulats.
8. Normalitzar l'àudio modulat dividint-lo pel màxim absolut.
9. Assignar l'àudio modulat al canal actual.
10. Finalment, es guarda el fitxer d'àudio amb l'efecte generat a un fitxer de sortida.

EFECTE LOW (APPLY_LOW_EFFECT):

Aquest efecte altera la velocitat i el to de l'àudio, fent que soni més greu i lent. Es crea mitjançant un factor de modulació i un factor tonal.
Per aquesta funció hem seguit exactament el mateix procediment que en la del efecte pitufo. El que canvia és que a l'hora de calcular un modulador que representa un senyal que varía en funció del temps i del factor de modulació li hem afegit un signe (-) al càlcul per tal obtenir les mostres negatives tal i com es pot fer en el editor d'àudio audacity.

EFECTE LOWPASS (APPLY_LOWPASS_FILTER) I EFECTE HIGHPASS(APPLY_HIGHPASS_FILTER)

En les funcions Pas-baix i Pas-alt utilitzem la llibreria pydub per aplicar tant un filtre pas-baix com un filtre pas-alt, ja que ens proporciona una funció específica per crear aquest tipus de filtres. Quan apliquem aquest efecte, el que estem fent és restringir les freqüències superiors o inferiors, depenent de quin dels dos filtres sigui, a 1800 Hz. Això significa que les freqüències fora de la freqüència de tall seran atenuades.
En el cas del filtre pas-baix destaquen les freqüències baixes, donant una sensació d'estar escoltant la música des de fora d'un establiment. En canvi, en aplicar el filtre pas-alt, són les freqüències altes les que destaquen i la sensació és d'estar escoltant la música dels auriculars d'una altra persona.

Per tant, el procediment que seguim en ambdós filtres es el següent:

1. Apliquem el filtre amb la funció específica de la llibreria pydub
2. Exportem l'àudio filtrat a un nou fitxer

FUNCIÓ CONVERT_TO_WAV

Aquesta funció proporciona d'una forma senzilla convertir fitxers d'àudio en format mp3 a format WAV utilitzant la llibreria ffmpeg. S'encarrega de gestionar situacions on es proporciona un fitxer d'entrada o quan apareix algún error durant el procés de conversió.

FUNCIÓ TRIM_AUDIO (RETALLAR ÀUDIO)

Aquesta funció permet retallar un fragment d'àudio específic d'un fitxer d'àudio utilitzant la informació proporcionada sobre el temps d'inici i la duració del segment. Aquest es guarda en un fitxer de sortida per el seu posterior ús.

FUNCIÓ PLAY I STOP

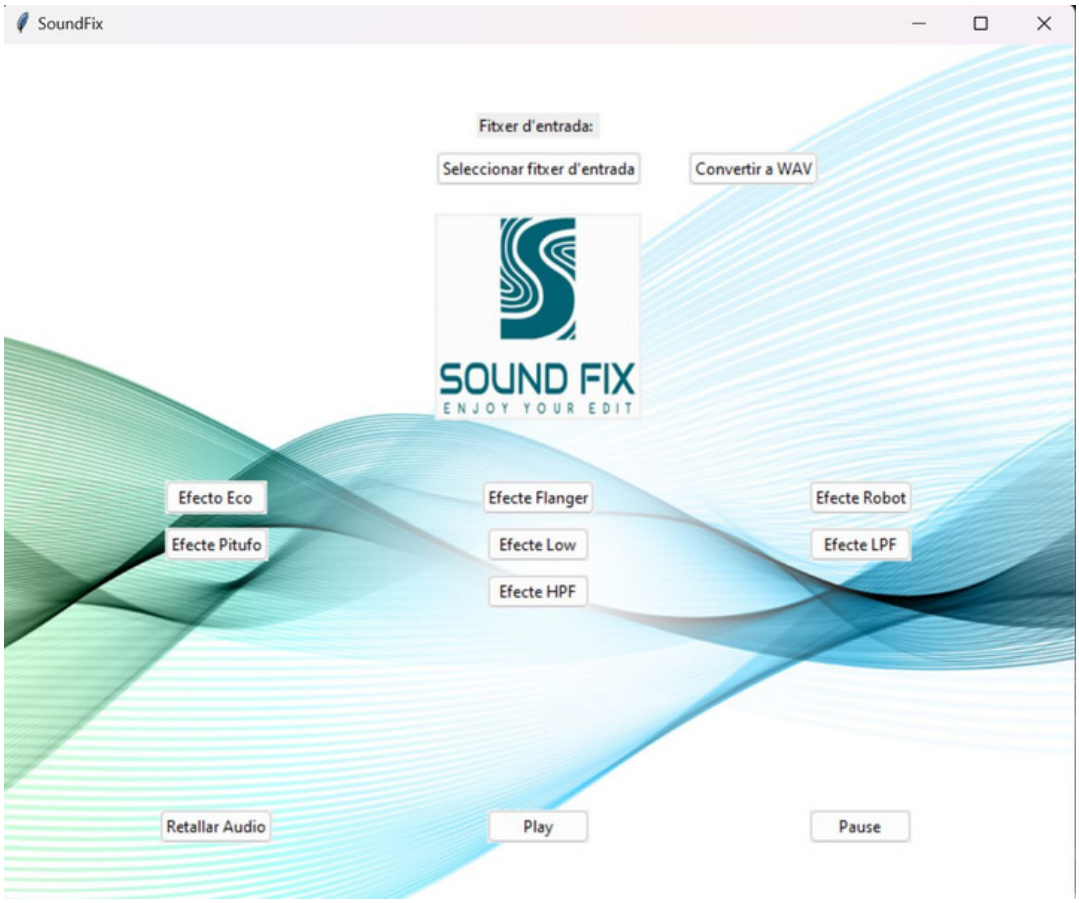
La funció "*play_audio*" reprodueix l'àudio emmagatzemat en el "*self.audio*" utilitzant la llibreria "*sounddevice*". L'àudio es reprodueix amb la taxa de mostreig especificada en el "self.*samplerate*".

La funció "*stop_audio*" detén la reproducció de l'àudio utilitzant la funció "*stop*" de la llibreria "*sounddevice*". L'àudio que es detén s'emmagatzema en el "*self.audio*"

Pel que fa a la part més visual del projecte, hem creat una interfície que permeti a l'usuari una edició d'àudio més intuïtiva. L'interfície inclou un botó per a cada funcionalitat del nostre programa:

- Càrrega de fitxers
- Convertir fitxers
- Efectes
- Segmentació de l'àudio
- Reproducció d'àudio
- Stop

Per al disseny de l'interfície hem utilitzat la llibreria tkinter, que és la que ens ha proporcionat totes les eines per definir tots els elements necessaris per crear-la.



05 CONCLUSIONS

Per concloure, hem aconseguit el nostre objectiu de desenvolupar una aplicació d'edició d'àudio que sigui intuitiva per a l'usuari, aplicant els coneixements adquirits a classe i realitzant recerca a través d'Internet. Malgrat haver enfrontat alguns reptes i haver superat diversos inconvenients, encara hi ha algunes limitacions pendents, com les següents:

- Millora de l'eficiència: Malgrat haver implementat classes i herències per augmentar l'eficiència, hem notat que el temps d'execució encara és una mica lent. Això significa que hi ha espai per a optimitzacions futures per millorar el rendiment general de l'aplicació.
- Altres funcionalitats: Considerem que hi ha una gran quantitat de funcionalitats addicionals que es podrien afegir a l'aplicació. Actualment, el programa és bastant senzill i només hem cobert alguns exemples bàsics de manipulació d'àudio. Seria interessant explorar altres possibilitats com l'afegit de transicions, efectes d'àudio avançats o eines d'edició més sofisticades.

En resum, tot i haver assolit l'objectiu principal de crear una aplicació d'edició d'àudio, reconeixem que encara hi ha àmbits en què es pot millorar. Aquesta experiència ens ha permès aplicar els nostres coneixements i habilitats, però també ens ha mostrat la complexitat i les possibilitats de millora en el camp de l'edició d'àudio.

06 BIBLIOGRAFIA

Aquestes son algunes de les web en les que hem fet recerca:

- https://docs.python.org/es/3/library/tkinter.html
- https://programacionpython80889555.wordpress.com/2020/02/25/manipulando-audios-en-python-con-pydub/

SOUND FIX
ENJOY YOUR EDIT

A continuació farem una breu explicació de com s'ha d'executar el programa per tal que funcioni de forma exitosa:

Pas 1:
Instal·lar totes les llibreries necessàries introduint a la terminal la següent comanda:
pip install nom_llibreria

Pel que fa a la instal·lació de la llibreria ffmpeg es fa d'una forma diferent. Hem d'entrar a la pàgina web oficial i descarregar el fitxer comprimit que conté l'executable.
Pàgina web: https://www.gyan.dev/ffmpeg/builds/
Seguidament, s'ha de descomprimir el fitxer i afegir la ruta del fitxer al path del teu ordinador. Per fer-ho, heu d'entrar a la configuració avançada de l'ordinador i entrar dins de les variables de l'entorn, després dins del path, afegir la ruta del fitxer ffmpeg.

Pas 2:
Un cop instal·lades totes les llibreries, es pot executar el fitxer de la interfície anomenat SoundFix.py.
Nota: Assegura't que el fitxer audio_processor.py estigui en el mateix directori que el de la interfície.

Pas 3:
Després d'executar el fitxer SoundFix.py i has realitzat els passos anteriors correctament, se t'obrirà automàticament una finestra amb la interfície de l'aplicació.

Funcionament de l'aplicació:
Hauràs de carregar el fitxer d'àudio que desitgis editar (ha d'estar en el mateix directori que el codi) amb el botó "Seleccionar fitxer d'entrada". L'aplicació no et deixarà utilitzar cap funcionalitat sense cap fitxer d'entrada, en cas de no seleccionar-ne cap, sortirà un missatge d'error a la pantalla indicant que es necessari carregar-ne un.

Nota: Les funcionalitats només serveixen per fitxers wav. És per això que també hem inclòs la funció "Convertir a WAV" per passar de mp3 a wav i per tant, també es pot carregar un fitxer mp3 i convertir-lo a wav abans de fer servir alguna funcionalitat.

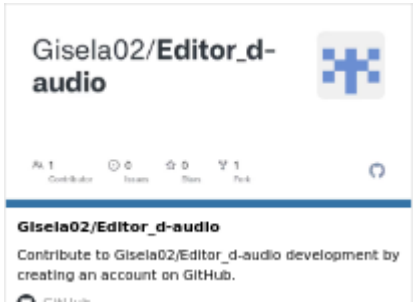
A partir d'aquí només caldrà que premis el botó amb l'efecte o filtre que vulguis posar i automàticament apareixerà un missatge a la pantalla dient que s'ha exportat correctament.
Per reproduir-lo hauràs de carregar el nou fitxer creat que apareixerà a la carpeta on són els fitxers de codi i clicar el botó de reproducció "Play". En cas que vulguis parar la reproducció hauràs de seleccionar el botó "Stop". Cal esmentar que no es podrà reprendre la reproducció sinó que haurà de tornar a començar en cas que així es desitgi.

Pel que fa a la segmentació de l'àudio apareixerà primer una finestra perquè li especifiquis des de quin segon vols començar el segment. Tot seguit hauràs de prémer el botó de "ok" i llavors perquè et surti la finestra en la qual s'ha de posar la duració del segment hauràs de clicar a qualsevol lloc de la interfície. És llavors quan s'exportarà l'àudio segmentat a la carpeta on hi ha el codi i l'àudio..

08 ANNEX 2

En el link de la dreta hi trobareu tot el projecte. Tant el codi de l'aplicació com el de la interfície. Per altra banda també hi trobareu alguns exemples en àudios.

https://github.com/Gisela02/Editor_d-audio.git



AUDIO_PROCESSOR.PY

```
import numpy as np
import soundfile as sf
import sounddevice as sd
from pydub import AudioSegment
from pydub.playback import play

class AudioProcessor:
  def __init__(self, input_file):
    self.input_file = input_file
    self.audio, self.samplerate = sf.read(input_file)

  def save_audio(self, output_file):
    sf.write(output_file, self.audio, self.samplerate)

  def play_audio(self):
    sd.play(self.audio, self.samplerate)

  def stop_audio(self):
    sd.stop(self.audio)

  def trim_audio(self, output_file, start_time, duration):
    start_sample = int(start_time * self.samplerate)
    end_sample = int((start_time + duration) * self.samplerate)
    trimmed_audio = self.audio[start_sample:end_sample]
    sf.write(output_file, trimmed_audio, self.samplerate)
```

```
class RobotEffect(AudioProcessor):
  def apply_robot_effect(self, output_file, modulation_factor=0.1, pitch_factor=0.9):
    num_channels = self.audio.shape[1] # Obtenir el número de canals
    mod = np.sin(2 * np.pi * np.arange(0, len(self.audio)) * (1.0 / self.samplerate) * modulation_factor)

    for channel in range(num_channels):
      audio_1d = self.audio[:, channel] # Obtenir el canal actual com una millora unidimensional
      mod_channel = mod[:len(audio_1d)] # Ajustar el tamañ de mod al tamañ del canal actual
      # Calcular els índexs de mostreig modulats
      indices = np.arange(len(audio_1d)) + mod_channel * self.samplerate * pitch_factor
      # Aplicar interpolació lineal per obtenir els valors d'àudio modulats
      modulated_audio = np.interp(indices, np.arange(len(audio_1d)), audio_1d)
      # Normalitzar l'àudio modulat
      max_value = np.max(np.abs(modulated_audio))
      modulated_audio /= max_value
      # Assignar l'àudio modulat al canal actual
      self.audio[:, channel] = modulated_audio
    # Guardar l'àudio modulat en el fitxer de sortida
    sf.write(output_file, self.audio, self.samplerate)


class EchoEffect(AudioProcessor):
  def apply_echo_effect(self, output_file, delay=0.7, decay=0.5):
    delay_samples = int(delay * self.samplerate)
    output_audio = np.zeros_like(self.audio)
    for i in range(len(self.audio)):
      if i >= delay_samples:
        output_audio[i] = self.audio[i] + decay * output_audio[i - delay_samples]
      else:
        output_audio[i] = self.audio[i]
    sf.write(output_file, output_audio, self.samplerate)

class FlangerEffect(AudioProcessor):
  def apply_flanger_effect(self, output_file, delay=0.003, depth=0.002, rate=0.2):
    delay_samples = int(delay * self.samplerate)
    depth_samples = int(depth * self.samplerate)
    modulator = depth_samples * np.sin(2 * np.pi * rate * np.arange(len(self.audio)) / self.samplerate)
    flanger_audio = np.zeros_like(self.audio)
    for i in range(len(self.audio)):
      if i >= delay_samples:
        index = int(i - delay_samples + modulator[i])
        flanger_audio[i] = self.audio[i] + self.audio[index]
      else:
        flanger_audio[i] = self.audio[i]
    sf.write(output_file, flanger_audio, self.samplerate)
```


```
class PitufoEffect(AudioProcessor):
  def apply_pitufo_effect(self, output_file, modulation_factor=0.1, pitch_factor=0.9):
    num_channels = self.audio.shape[1]
    mod = (2 * np.pi * np.arange(0, len(self.audio)) * (1.0 / self.samplerate) * modulation_factor)

    for channel in range(num_channels):
      audio_1d = self.audio[:, channel]
      mod_channel = mod[:len(audio_1d)]
      indices = np.arange(len(audio_1d)) + mod_channel * self.samplerate * pitch_factor
      modulated_audio = np.interp(indices, np.arange(len(audio_1d)), audio_1d)
      max_value = np.max(np.abs(modulated_audio))
      modulated_audio /= max_value
      self.audio[:, channel] = modulated_audio
    sf.write(output_file, self.audio, self.samplerate)

class LowEffect(AudioProcessor):
  def apply_low_effect(self, output_file, modulation_factor=0.1, pitch_factor=0.9):
    num_channels = self.audio.shape[1]
    mod = -(2 * np.pi * np.arange(0, len(self.audio)) * (1.0 / self.samplerate) * modulation_factor)

    for channel in range(num_channels):
      audio_1d = self.audio[:, channel]
      mod_channel = mod[:len(audio_1d)]
      indices = np.arange(len(audio_1d)) + mod_channel * self.samplerate * pitch_factor
      modulated_audio = np.interp(indices, np.arange(len(audio_1d)), audio_1d)
      max_value = np.max(np.abs(modulated_audio))
      modulated_audio /= max_value
      self.audio[:, channel] = modulated_audio
    sf.write(output_file, self.audio, self.samplerate)

class LowPassFilter(AudioProcessor):
  def apply_lowpass_filter(self, output_file, cutoff_frequency=1800):
    audio = AudioSegment.from_file(self.input_file)
    filtered_audio = audio.low_pass_filter(cutoff_frequency)
    filtered_audio.export(output_file, format='wav')

class HighPassFilter(AudioProcessor):
  def apply_highpass_filter(self, output_file, cutoff_frequency=1800):
    audio = AudioSegment.from_file(self.input_file)
    filtered_audio = audio.high_pass_filter(cutoff_frequency)
    filtered_audio.export(output_file, format="wav")
```

SOUNDFIX.PY


```
import tkinter as tk
import numpy
from tkinter import filedialog
from tkinter import messagebox
from tkinter import ttk
from tkinter import simpledialog
from functools import partial
from pydub import AudioSegment
from audio_processor import AudioProcessor, EchoEffect, FlangerEffect, RobotEffect,
PitufoEffect, LowEffect, LowPassFilter, HighPassFilter
from PIL import Image
import ffmpeg
import os


class SoundFix:
  def __init__(self, root):
    self.root = root
    self.root.title("SoundFix")
    self.root.geometry("800x700")

    # Agregar un fons
    self.background_image = tk.PhotoImage(file="background.png")
    self.background_label = tk.Label(self.root, image=self.background_image)
    self.background_label.place(x=0, y=0, relwidth=1, relheight=1)

    # Agregar un logo
    self.logo_image = tk.PhotoImage(file="logo.png")
    self.logo_label = tk.Label(self.root, image=self.logo_image)
    self.logo_label.pack(pady=10)
    self.logo_label.place(relx=0.5, rely=0.4, anchor=tk.S)

    self.input_file = None
    self.output_file = None
    self.playing = False

    self.create_widgets()
```

```
  def create_widgets(self):
    # Text per mostrar el fitxer d'entrada seleccionat
    self.label_input = ttk.Label(self.root, text="Fitxer d'entrada: ")
    self.label_input.pack()
    self.label_input.place(relx=0.5, rely=0.1, anchor=tk.S)

    # Botó per seleccionar el fitxer d'entrada
    self.btn_browse_input = ttk.Button(self.root, text="Seleccionar fitxer d'entrada",
command=self.browse_input)
    self.btn_browse_input.pack()
    self.btn_browse_input.place(relx=0.5, rely=0.15, anchor=tk.S)

    # Botó per convertir un fitxer MP3 a WAV
    self.btn_convert_to_wav = ttk.Button(self.root, text="Convertir a WAV",
command=self.convert_to_wav)
    self.btn_convert_to_wav.pack()
    self.btn_convert_to_wav.place(relx=0.7, rely=0.15, anchor=tk.S)

    # Botó para retallar el fitxer d'àudio
    self.btn_trim_audio = ttk.Button(self.root, text="Retallar Audio", command=self.trim_audio)
    self.btn_trim_audio.pack()
    self.btn_trim_audio.place(relx=0.2, rely=0.85, anchor=tk.S)

    # Botón per reproducir l'àudio
    self.btn_load_and_play = ttk.Button(self.root, text="Play", command=self.load_and_play_audio)
    self.btn_load_and_play.pack()
    self.btn_load_and_play.place(relx=0.5, rely=0.85, anchor=tk.S)

    # Botó per parar la reproducció d'àudio
    self.btn_stop_audio = ttk.Button(self.root, text="Stop", command=self.stop_audio)
    self.btn_stop_audio.pack()
    self.btn_stop_audio.place(relx=0.8, rely=0.85, anchor=tk.S)

    # Botó per aplicar l'efecte robot
    self.btn_robot_effect = ttk.Button(self.root, text="Efecte Robot",
command=self.apply_robot_effect)
    self.btn_robot_effect.pack()
    self.btn_robot_effect.place(relx=0.8, rely=0.5, anchor=tk.S)

    # Botó per aplicar l'efecto eco
    self.btn_echo_effect = ttk.Button(self.root, text="Efecto Eco", command=self.apply_echo_effect)
    self.btn_echo_effect.pack()
    self.btn_echo_effect.place(relx=0.2, rely=0.5, anchor=tk.S)

    # Botó per aplicar l'efecte flanger
    self.btn_flanger_effect = ttk.Button(self.root, text="Efecte Flanger",
command=self.apply_flanger_effect)
    self.btn_flanger_effect.pack()
    self.btn_flanger_effect.place(relx=0.5, rely=0.5, anchor=tk.S)
```

```
    # Botó per aplicar l'efecte pitufo
 self.btn_pitufo_effect = ttk.Button(self.root, text="Efecte Pitufo",
command=self.apply_pitufo_effect)
    self.btn_pitufo_effect.pack()
    self.btn_pitufo_effect.place(relx=0.2, rely=0.55, anchor=tk.S)

    # Botó per aplicar l'efecte Low
    self.btn_low_effect = ttk.Button(self.root, text="Efecte Low", command=self.apply_low_effect)
    self.btn_low_effect.pack()
    self.btn_low_effect.place(relx=0.5, rely=0.55, anchor=tk.S)

    # Botó per aplicar l'efecte LPF
    self.btn_lowPass_effect = ttk.Button(self.root, text="Efecte LPF",
command=self.apply_lowpass_filter)
    self.btn_lowPass_effect.pack()
    self.btn_lowPass_effect.place(relx=0.8, rely=0.55, anchor=tk.S)

    # Botó per aplicar l'efecte HPF
    self.btn_highPass_effect = ttk.Button(self.root, text="Efecte HPF",
command=self.apply_highpass_filter)
    self.btn_highPass_effect.pack()
    self.btn_highPass_effect.place(relx=0.5, rely=0.6, anchor=tk.S)

  def browse_input(self):
    self.input_file = filedialog.askopenfilename(filetypes=[("Audio Files", "*.wav *.mp3")])
    self.label_input.config(text="Fitxer d'entrada: " + self.input_file)

  def convert_to_wav(self):
    if self.input_file:
      input_filename, input_extension = os.path.splitext(self.input_file)
      output_file_wav = input_filename + ".wav"
      try:
        ffmpeg.input(self.input_file).output(output_file_wav).run()
        messagebox.showinfo("Convertir a WAV", "La conversió s'ha realitzat correctament!")
      except ffmpeg.Error as e:
        messagebox.showerror("Error!", f"Hi ha hagut un error durant la conversió: {str(e)}")
    else:
      messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")
```

```
def trim_audio(self):
  if self.input_file:
    start_time = simpledialog.askfloat("Retallar Audio", "Inserti el punt d'inici (en segons):")
    duration = simpledialog.askfloat("Retallar Audio", "Inserti la duració del tall (en segons):")
    input_filename, input_extension = os.path.splitext(self.input_file)
    output_file_trimmed = input_filename + "_Tall.wav"

    audio_processor = AudioProcessor(self.input_file)
    audio_processor.trim_audio(output_file_trimmed, start_time, duration)

    messagebox.showinfo("Audio Retallat!", "L'àudio s'ha retallat correctament!")

  else:
    messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")

def load_and_play_audio(self):
  if self.input_file:
    audio_processor = AudioProcessor(self.input_file)
    audio_processor.play_audio()
  else:
    messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")

def stop_audio(self):
  if self.input_file:
    audio_processor = AudioProcessor(self.input_file)
    audio_processor.stop_audio()
  else:
    messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")

def apply_robot_effect(self):
  if self.input_file:
    robot_effect = RobotEffect(self.input_file)
    input_filename, input_extension = os.path.splitext(self.input_file)
    output_file_robot = input_filename + "_Robot.wav"
    robot_effect.apply_robot_effect(output_file_robot)
    messagebox.showinfo("Efecte Robot", "L'efecte Robot s'ha aplicat correctament!.")
  else:
    messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")
```

```
def apply_echo_effect(self):
  if self.input_file:
    echo_effect = EchoEffect(self.input_file)
    input_filename, input_extension = os.path.splitext(self.input_file)
    output_file_echo = input_filename + "_Echo.wav"
    echo_effect.apply_echo_effect(output_file_echo)
    messagebox.showinfo("Efecte Eco", "L'efecte Eco s'ha aplicat correctament!.")
  else:
    messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")

def apply_flanger_effect(self):
  if self.input_file:
    flanger_effect = FlangerEffect(self.input_file)
    input_filename, input_extension = os.path.splitext(self.input_file)
    output_file_flanger = input_filename + "_Flanger.wav"
    flanger_effect.apply_flanger_effect(output_file_flanger)
    messagebox.showinfo("Efecte Flanger", "L'efecte Flanger s'ha aplicat correctament!.")
  else:
    messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")

def apply_pitufo_effect(self):
  if self.input_file:
    pitufo_effect = PitufoEffect(self.input_file)
    input_filename, input_extension = os.path.splitext(self.input_file)
    output_file_pitufo = input_filename + "_Pitufo.wav"
    pitufo_effect.apply_pitufo_effect(output_file_pitufo)
    messagebox.showinfo("Efecte Pitufo", "L'efecte Pitufo s'ha aplicat correctament!.")
  else:
    messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")

def apply_low_effect(self):
  if self.input_file:
    low_effect = LowEffect(self.input_file)
    input_filename, input_extension = os.path.splitext(self.input_file)
    output_file_low = input_filename + "_Low.wav"
    low_effect.apply_low_effect(output_file_low)
    messagebox.showinfo("Efecte Low", "L'efecte Low s'ha aplicat correctament!.")
  else:
    messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")
```

```
  def apply_lowpass_filter(self):
    if self.input_file:
      lowPass_effect = LowPassFilter(self.input_file)
      input_filename, input_extension = os.path.splitext(self.input_file)
      output_file_lowPass = input_filename + "_LowPass.wav"
      lowPass_effect.apply_lowpass_filter(output_file_lowPass)
      messagebox.showinfo("Efecte LPF", "L'efecte Low Pass Filter s'ha aplicat correctament!.")
    else:
      messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")

  def apply_highpass_filter(self):
    if self.input_file:
      highPass_effect = HighPassFilter(self.input_file)
      input_filename, input_extension = os.path.splitext(self.input_file)
      output_file_highPass = input_filename + "_HighPass.wav"
      highPass_effect.apply_highpass_filter(output_file_highPass)
      messagebox.showinfo("Efecte HPF", "L'efecte High Pass Filter s'ha aplicat correctament!.")
    else:
      messagebox.showerror("Error!", "Si us plau, seleccioni un fitxer d'entrada.")

if __name__ == "__main__":
  root = tk.Tk()
  app = SoundFix(root)
  root.mainloop()
```