Design Overview for “Mike’s Virtual Amp”

Name: Dang Duc Minh
Student ID: 105559691

This is a Python program
Anaconda needs to be installed:
https://www.anaconda.com/download/success
After installed, a custom environment needs to be created.
Open command prompt and enter all of the following in order:
conda create -n guitar_amp_py38 python=3.8
conda activate guitar_amp_py38
pip install pyaudio
pip install numpy
The program needs to run by using terminal: *python amp.py*

Here are the lists of libraries that I will be using:

- **pyaudio**:
  Manages real-time audio input and output streams for recording and playback.
- **numpy**:
  Processes audio data efficiently using arrays for mathematical operations.
- **threading**:
  Runs the GUI and audio processing simultaneously in separate threads.
- **tkinter**:
  Provides a graphical user interface (GUI) toolkit for building applications in Python.
- **Pillow (PIL)**:
  Handles image processing to display amp and effect pedal images.

1. Overview

Purpose:

Simulates a real-time virtual guitar amp where users can connect a guitar, apply various audio effects like gain and overdrive, and output the processed sound through headphones or speakers.

Key Features:

- **Real-Time Processing**: Continuous audio processing with adjustable parameters like master volume, bass, mid, treble, gain, and overdrive.
- **Recording Capability**: Allows audio recording with saved output in .wav format.
- **Graphical Interface**: Includes amp and pedal models with interactive controls for an intuitive experience.

- **Alternative Use**: Can also serve as a voice amplifier with a plugged in microphone

2. Program Elements

Real-Time Virtual Guitar Amp

- Allows users to:
  - Adjust sound parameters dynamically.
  - Toggle standby to enable or disable audio processing.
  - Apply effects like overdrive gain with toggling and adjustments.

Audio Output

- Processes and sends audio to the speakers or headphones in real time.

Audio Recording

- Users can record audio while playing and save recordings as .wav files.

3. Graphical Interface Elements

Main Interface:

- **Knobs**: Interactive controls for master volume, gain, bass, mid, treble, and overdrive gain. Each knob adjusts the corresponding parameter in real time.
- **Standby Switch**: Toggles audio processing on or off. Includes a visual indicator with a red LED for active and dark red for standby mode.
- **Overdrive Pedal**: Simulates an overdrive pedal with toggle functionality and a dedicated overdrive gain knob.
- **Recording Button**: Allows starting and stopping recordings. Saves recordings to the specified directory.
- **Dynamic Output Label**: Displays current sound settings, updating dynamically as users adjust controls.

Images:

- Amp and pedal images visually enhance the interface, representing hardware counterparts.

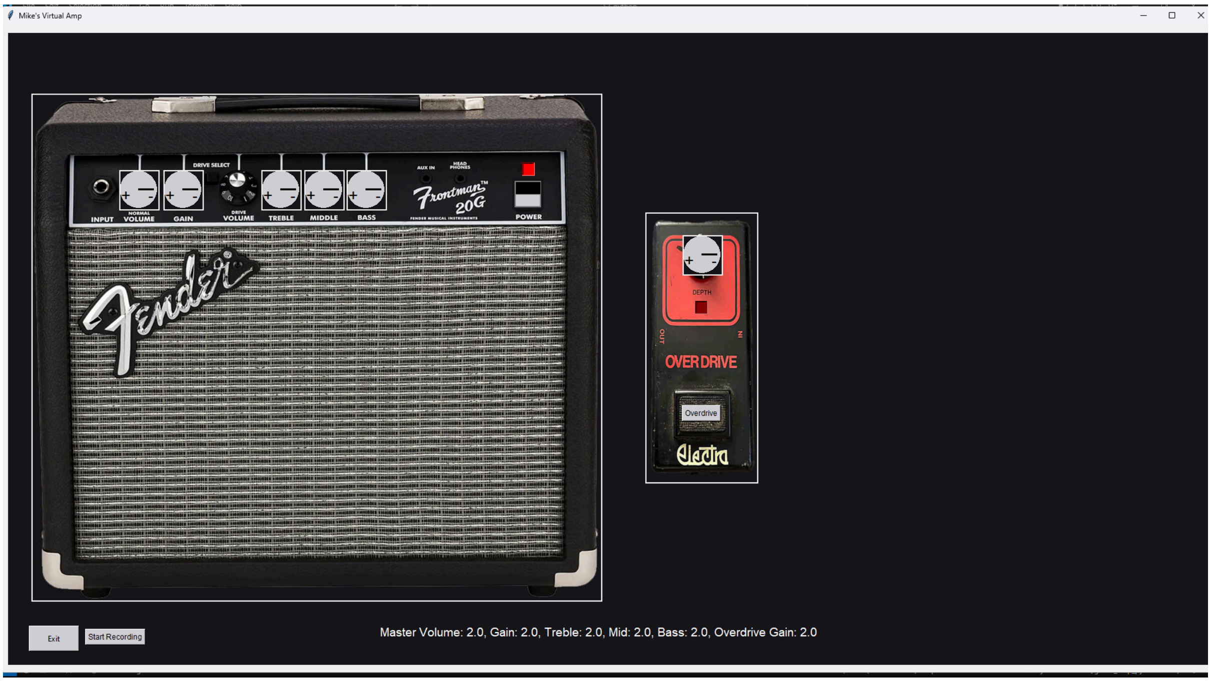
4. Event Handling

- **Knob Interaction**: Users rotate knobs by left- or right-clicking to increase or decrease parameter values.

- **Effect Toggles**: Buttons and switches allow toggling effects like standby and overdrive.
- **Recording Control**: A dedicated button manages recording start/stop and saves recorded audio.
- **Mouse Events**: Buttons and switches respond to clicks, while dynamic text updates provide immediate feedback.

5. Update Loop

- **Continuous Audio Processing**: Audio input is processed in real-time to apply effects and amplify sound based on user input.
- **Recording Logic**: Captures processed audio frames when recording is enabled.
- **Dynamic GUI Updates**: The output label updates live to reflect changes to audio settings.



Master Volume: 2.0, Gain: 2.0, Treble: 2.0, Mid: 2.0, Bass: 2.0, Overdrive Gain: 2.0

Required Data Types

Field Name	Type	Notes
FORMAT	Integer	Audio Format (16-bit)

CHANNELS	Integer	Number of audio channels (1 for mono)
RATE	Integer	Sample rate (e.g. 22050 Hz)
CHUNK	Integer	Buffer size for audio processing (number of frames)
a	pyaudio.PyAudio	Instance of the Pyaudio
bass_gain	float	Gain factor for bass frequencies
mid_gain	float	Gain factor for mid frequencies
treble_gain	float	Gain factor for treble frequencies
master_gain	float	Overall gain factor for the audio output
master_volume	float	Volume control factor
audio_processing	bool	Indicating whether audio processing is active
data	bytes	Raw audio data received from the input stream
audio_data	numpy.ndarray	Numpy array holding processed audio data
img	tk.PhotoImage	Image object for the GUI background
root	tk.Tk	Main window instance for the Tkinter GUI
standby_button	tk.Button	Button widget to toggle standby state
led_label	tk.Label	Label widget to display the LED indicator
increment	float	Amount to adjust gains
param	string	Parameter name for gain adjustment
operation	string	Operation type for gain adjustment (increase or decrease)

Overview of Program Structure

Main Functions/Procedures

1. Initialization Functions

initialize_audio()

- **Purpose**: Sets up audio processing parameters and initializes the audio stream.
- **Key Tasks**:

    - Define FORMAT, CHANNELS, RATE, CHUNK.
    - Create and configure the PyAudio stream.
    - Set default gain values (bass_gain, mid_gain, etc.).

initialize_gui()

- **Purpose**: Sets up the graphical user interface (GUI) for the program.
- **Key Tasks**:
    - Create the main Tkinter window (root).
    - Add widgets like knobs, buttons, sliders, and labels.
    - Configure the standby button with an LED indicator.

2. Audio Input/Output Functions

audio_callback(in_data)

- **Purpose**: Processes incoming audio in real-time.
- **Key Tasks**:
    - Read audio data from the input stream.
    - Pass the data through the effects chain.
    - Write the processed audio to the output stream.

start_audio()

- **Purpose**: Begins audio processing and starts the PyAudio stream.
- **Key Tasks**:
    - Open the audio stream for reading/writing.
    - Toggle the standby button state and update the LED.

stop_audio()

- **Purpose**: Stops audio processing and closes the PyAudio stream.
- **Key Tasks**:
    - Stop the stream and reset its state.
    - Update the standby LED to indicate processing is off.

---

3. Effects Processing

apply_equalizer(audio_data)

- **Purpose**: Adjust the bass, mid, and treble frequencies of the audio signal.
- **Key Tasks**:
    - Apply frequency band filters using the gain values.
    - Modify the signal amplitude for each band.

apply_overdrive(audio_data)

- **Purpose**: Adds overdrive/distortion to the audio signal.

- **Key Tasks**:
    - Amplify the signal based on the gain knob.
    - Clip the signal to simulate distortion.

apply_delay(audio_data)

- **Purpose**: Adds a delay/echo effect to the audio signal.
- **Key Tasks**:
    - Mix delayed versions of the signal with the original.
    - Adjust delay speed and intensity using the delay pedal controls.

process_audio(audio_data)

- **Purpose**: Combines all active effects in the processing chain.
- **Key Tasks**:
    - Pass the signal through the equalizer, overdrive, and delay functions.
    - Return the fully processed audio data.

---

4. GUI Interaction

update_gain(param, operation)

- **Purpose**: Adjusts gain settings (e.g., bass, treble, or master) based on user input.
- **Key Tasks**:
    - Increase or decrease the gain for the selected parameter.
    - Update the corresponding label to reflect the new value.

toggle_standby()

- **Purpose**: Turns audio processing on or off.
- **Key Tasks**:
    - Start or stop the audio stream.
    - Change the LED indicator color (e.g., red for "on," dark red for "off").

update_display()

- **Purpose**: Dynamically updates GUI elements.
- **Key Tasks**:
    - Reflect changes in gain, effects, or amp settings on the GUI.

---

5. Utility Functions

create_knob(image_path, callback_function)

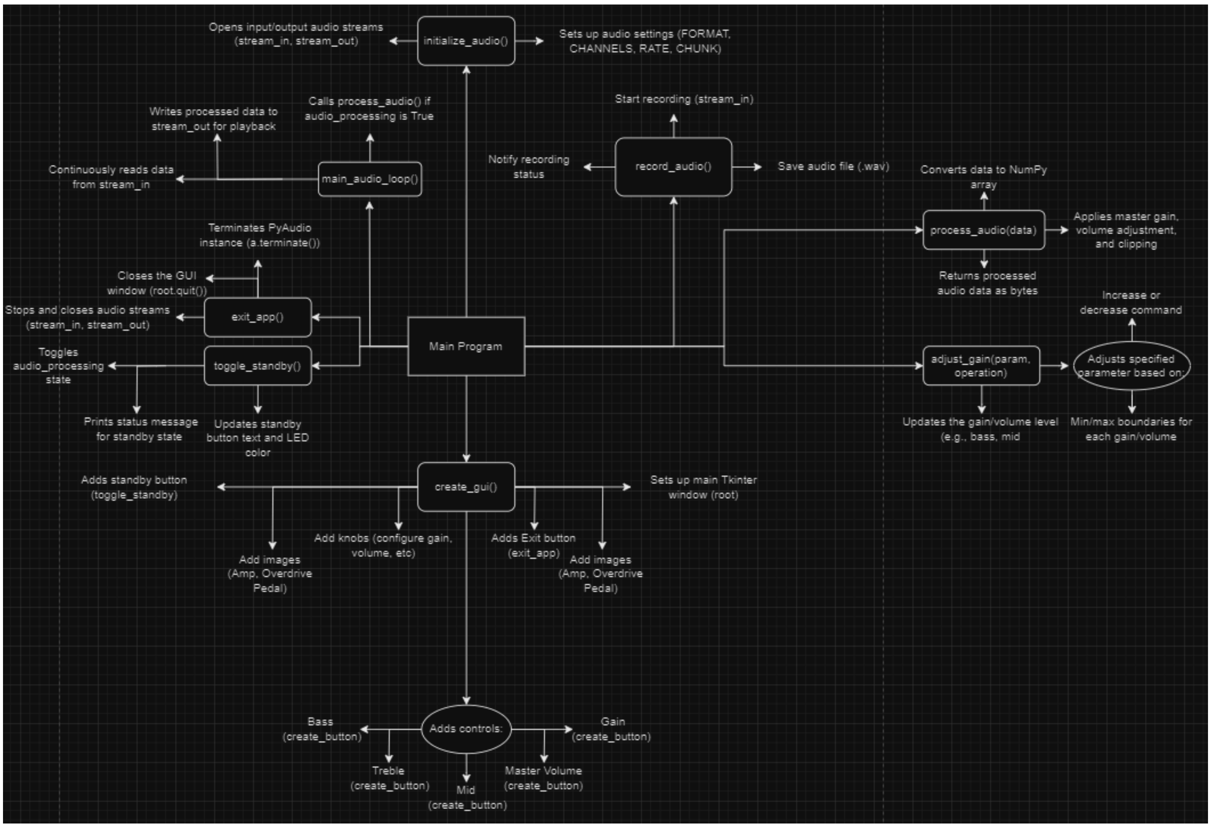
- **Purpose**: Creates a custom knob widget for adjusting audio parameters.
- **Key Tasks**:

    - Load knob image with transparent background.
    - Bind the rotation of the knob to the callback function.

main_loop()

- **Purpose**: Runs the main event loop for the Tkinter GUI.
- **Key Tasks**:
    - Starts the GUI and keeps it responsive to user actions.

Program Structure Chart


initialize_audio()
Opens input/output audio streams (stream_in, stream_out)
Sets up audio settings (FORMAT, CHANNELS, RATE, CHUNK)
Writes processed data to stream_out for playback
Calls process_audio() if audio_processing is True
Continuously reads data from stream_in
main_audio_loop()
Start recording (stream_in)
Notify recording status
record_audio()
Save audio file (.wav)
Converts data to NumPy array
process_audio(data)
Applies master gain, volume adjustment, and clipping
Returns processed audio data as bytes
Terminates PyAudio instance (a.terminate())
Closes the GUI window (root.quit())
Stops and closes audio streams (stream_in, stream_out)
exit_app()
Main Program
Toggles audio_processing state
toggle_standby()
Prints status message for standby state
Updates standby button text and LED color
Increase or decrease command
adjust_gain(param, operation)
Adjusts specified parameter based on:
Updates the gain/volume level (e.g., bass, mid
Min/max boundaries for each gain/volume
Adds standby button (toggle_standby)
create_gui()
Sets up main Tkinter window (root)
Add knobs (configure gain, volume, etc)
Adds Exit button (exit_app)
Add images (Amp, Overdrive Pedal)
Add images (Amp, Overdrive Pedal)
Adds controls:
Bass (create_button)
Gain (create_button)
Treble (create_button)
Master Volume (create_button)
Mid (create_button)