**Major Project**

Mini VLC (Media Player)

**Project Report**

**Prepared By:**  
Shaikh Zidaan Mohd Zubair

**Enrollment No:**  
23111590017

**College:**  
A.R. Kalsekar Polytechnic Panvel

**Instructor:**  
Prof. Sumit Jaiswar

  
  
**Milestone Institute of Technology**

**Abstract**

This project details the development of the "Mini VLC" media player, a comprehensive desktop application for audio and video playback. The system is built on **Python** and leverages **Tkinter** for the user interface and the **Python-VLC** binding for core media decoding and playback. The application's design provides a rich set of features, including a dynamic playlist manager with drag-and-drop functionality, a real-time seek bar, and an audio volume controller. The system successfully integrates a powerful external library (libVLC) with a custom GUI to offer a fluid user experience. This project demonstrates advanced skills in GUI development, event-driven programming, and the integration of multi-modal data streams for an elegant and highly functional media application.

**Index**

1. Introduction   
    1.1 Project Overview   
    1.2 Motivation and Use Case   
    1.3 Objectives
2. Technology Stack Used   
    2.1 Programming Languages   
    2.2 Libraries and Frameworks   
    2.3 Platforms and Tools
3. Dataset and Data Preparation   
    3.1 Dataset Source   
    3.2 Data Exploration and Cleaning   
    3.3 Feature Engineering
4. Methodology   
    4.1 Algorithm Used   
    4.2 Tools and Technologies   
    4.3 Development Framework (Agile Methodology)
5. Developed System   
    5.1 Project Requirement Details   
    5.2 High Level Design (HLD)   
    5.3 Wireframe of Frontend and Backend   
    5.4 Source Code of Frontend and Backend   
    5.5 Project in Production / Hosted Live   
    5.5.1 Hosting Site Screenshot   
    5.5.2 Hosted Project Screenshot (Live Application)   
    5.6 Results and Reports   
    5.7 Cost Estimation
6. Evaluation and Testing   
    6.1 Evaluation Metrics   
    6.2 Confusion Matrix / ROC Curve   
    6.3 Functional Testing and Debug Logs
7. Challenges and Troubleshooting   
    7.1 Technical Challenges   
    7.2 Solutions Implemented
8. Future Scope and Enhancements
9. Conclusion
10. References

Major Project: Mini VLC (Media Player)

**1. Introduction**

**1.1 Project Overview**

This report details the development of the "Mini VLC" media player, a full-featured desktop application for playing audio and video files. The application provides a user-friendly graphical interface with a video display, a playlist manager, and a comprehensive set of playback controls including play, pause, stop, seeking, and volume adjustment.

**1.2 Motivation and Use Case**

The primary motivation for this project was to create a functional, robust, and visually appealing media player from scratch. It serves as a practical demonstration of software development principles, including GUI design, event handling, and real-time media processing. The application's use case is to provide a reliable tool for users to manage and play their local media files.

**1.3 Objectives**

* To develop a desktop application with a modern and intuitive user interface.

* To implement core media playback functionalities (play, pause, stop, next, previous).

* To incorporate advanced controls such as a seek bar and volume control.

* To create a playlist system for managing multiple media files.

* To handle different types of media files, including both audio and video.

**2. Technology Stack Used**

**2.1 Programming Languages**

* **Python**: The core programming language used for the entire application logic and GUI.

**2.2 Libraries and Frameworks**

* **Tkinter**: Python's standard GUI toolkit used to build the entire user interface, including windows, buttons, labels, and frames.

* **Python-VLC**: A Python binding for the popular VLC media player library (libVLC), which handles the heavy-lifting of media decoding and playback for various file formats.

* **Tkinterdnd2**: An extension for Tkinter to enable drag-and-drop functionality, allowing users to easily add files to the playlist.

* **Pillow (PIL)**: Used for loading and resizing image icons for a more visually appealing interface.

**2.3 Platforms and Tools**

* **Windows / macOS / Linux**: The application is designed to be cross-platform due to the use of Python and Tkinter.

* **Code Editor**: A standard text editor or IDE was used for writing the code.

**3. Dataset and Data Preparation**

* The project does not rely on a predefined dataset. It works with user-provided local media files.

* **Data Source**: User's local audio and video files (.mp4, .mkv, .mp3, .wav, etc.).

* **Data Preparation**: The application processes file paths and metadata dynamically. The file paths are stored in a simple list to manage the playlist.

**4. Methodology**

**4.1 Algorithm Used**

The core of the application's functionality is a polling mechanism. A recurring task runs every 250 milliseconds to check the current position of the media file being played. This position is then used to update the seek bar and the time display on the GUI.

**4.2 Tools and Technologies**

* **vlc.Instance()**: Creates a VLC instance.

* **vlc.media\_player\_new()**: Creates a new media player object.

* **player.set\_media(media)**: Loads a media file for playback.

* **player.play()**: Starts playback.

* **player.get\_time()**: Retrieves the current playback position in milliseconds.

* **player.set\_time(t)**: Allows seeking to a specific position.

* **tkinter.ttk.Scale**: Used to create the seek bar and volume slider.

* **root.after()**: Schedules a function to be called after a given time, which is essential for the continuous polling mechanism.

**5. Developed System**

**5.1 Project Requirement Details**

The project was developed to meet the requirements of a modern media player, including:

* A clean and functional GUI.

* Controls for playback (play, pause, stop).

* Navigation (next/previous song).

* Volume control.

* A playlist to manage media files.

* Real-time updates to the seek bar and time labels.

**5.2 High-Level Design (HLD)**

The application follows a modular, object-oriented design. The MiniVLC class encapsulates all the UI components and playback logic. Helper functions are used for tasks like time formatting and image loading. The vlc library handles the complex media decoding, allowing the application to focus on the user-facing features.

**5.3 Wireframe of Frontend and Backend**

* **Frontend**: A single Tkinter window divided into two main sections:
* **Left Pane**: The main video display area and the playback controls (seek bar, buttons).
* **Right Pane**: The playlist manager, with a scrollable listbox and buttons for adding/removing files.

* **Backend**: The backend is primarily the vlc library, which handles media streams. The Python script acts as a frontend to this library, managing user interactions and updating the GUI based on the playback state.

**5.4 Source Code of Frontend and Backend**

import os

import sys

import time

import tkinter as tk

from tkinter import ttk, filedialog, messagebox

# Optional libs (graceful fallback)

PIL\_OK = True

DND\_OK = True

try:

from PIL import Image, ImageTk

except Exception:

PIL\_OK = False

try:

from tkinterdnd2 import DND\_FILES, TkinterDnD

except Exception:

DND\_OK = False

import vlc

# ---------- helpers ----------

def fmt\_time(ms: int) -> str:

if ms is None or ms < 0:

return "00:00"

s = ms // 1000

return f"{s // 60:02}:{s % 60:02}"

class MiniVLC:

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

self.root = root

self.root.title("🎬 Mini VLC")

self.root.geometry("1000x650")

self.root.minsize(820, 520)

self.root.configure(bg="#0f1115")

self.vlc\_instance = vlc.Instance()

self.player = self.vlc\_instance.media\_player\_new()

self.playlist = []

self.current\_index = -1

self.length\_ms = 0

self.poll\_job = None

self.user\_dragging\_seek = False

self.fullscreen = False

self.muted = False

self.saved\_volume = 80

self.is\_playing = False

self.\_setup\_styles()

self.\_load\_images()

self.\_build\_ui()

self.\_bind\_shortcuts()

self.\_embed\_video\_after\_ready()

if DND\_OK:

try:

self.root.drop\_target\_register(DND\_FILES)

self.root.dnd\_bind('<<Drop>>', self.\_on\_drop)

except Exception:

pass

# ---------- UI setup ----------

def \_setup\_styles(self):

style = ttk.Style()

try:

style.theme\_use('clam')

except Exception:

pass

style.configure(".", background="#0f1115", foreground="#e6e6e6", font=("Segoe UI", 10))

style.configure("Title.TLabel", background="#0f1115", foreground="#e6e6e6", font=("Segoe UI", 16, "bold"))

style.configure("Card.TFrame", background="#151922")

style.configure("Bar.TFrame", background="#0f1115")

style.configure("TButton", background="#1DB954", foreground="white", padding=8, font=("Segoe UI", 10, "bold"))

style.map("TButton", background=[("active", "#1ed760")])

style.configure("Alt.TButton", background="#2b3240")

style.map("Alt.TButton", background=[("active", "#343d4e")])

style.configure("TScale", background="#151922", troughcolor="#2a2f3a")

def \_load\_images(self):

global PIL\_OK

if not PIL\_OK:

return

size = (24, 24)

try:

# Create a path for the icons folder

base\_path = getattr(sys, '\_MEIPASS', os.path.dirname(os.path.abspath(\_\_file\_\_)))

icons\_path = os.path.join(base\_path, "icons")

self.play\_img = ImageTk.PhotoImage(Image.open(os.path.join(icons\_path, "play.png")).resize(size, Image.LANCZOS))

self.pause\_img = ImageTk.PhotoImage(Image.open(os.path.join(icons\_path, "pause.png")).resize(size, Image.LANCZOS))

self.stop\_img = ImageTk.PhotoImage(Image.open(os.path.join(icons\_path, "stop.png")).resize(size, Image.LANCZOS))

self.prev\_img = ImageTk.PhotoImage(Image.open(os.path.join(icons\_path, "prev.png")).resize(size, Image.LANCZOS))

self.next\_img = ImageTk.PhotoImage(Image.open(os.path.join(icons\_path, "next.png")).resize(size, Image.LANCZOS))

self.fullscreen\_img = ImageTk.PhotoImage(Image.open(os.path.join(icons\_path, "fullscreen.png")).resize(size, Image.LANCZOS))

self.mute\_img = ImageTk.PhotoImage(Image.open(os.path.join(icons\_path, "mute.png")).resize(size, Image.LANCZOS))

self.unmute\_img = ImageTk.PhotoImage(Image.open(os.path.join(icons\_path, "unmute.png")).resize(size, Image.LANCZOS))

except Exception as e:

print(f"Error loading images: {e}. Falling back to text.")

PIL\_OK = False

def \_build\_ui(self):

top = ttk.Frame(self.root, style="Bar.TFrame")

top.pack(fill="x", padx=14, pady=(10, 6))

ttk.Label(top, text="🎬 Mini VLC", style="Title.TLabel").pack(side="left")

ttk.Button(top, text="📂 Open", command=self.open\_files, style="Alt.TButton").pack(side="right", padx=4)

ttk.Button(top, text="📁 Folder", command=self.open\_folder, style="Alt.TButton").pack(side="right", padx=4)

main = ttk.Frame(self.root)

main.pack(fill="both", expand=True, padx=14, pady=(0, 8))

left = ttk.Frame(main, style="Card.TFrame")

left.pack(side="left", fill="both", expand=True, padx=(0, 8))

self.video\_area = tk.Frame(left, bg="black")

self.video\_area.pack(fill="both", expand=True, padx=12, pady=12)

ctrl\_card = ttk.Frame(left, style="Card.TFrame")

ctrl\_card.pack(fill="x", padx=12, pady=(0, 12))

times = ttk.Frame(ctrl\_card, style="Card.TFrame")

times.pack(fill="x", padx=12, pady=(10, 0))

self.lbl\_cur = ttk.Label(times, text="00:00")

self.lbl\_tot = ttk.Label(times, text="00:00")

self.lbl\_cur.pack(side="left")

self.lbl\_tot.pack(side="right")

self.seek = ttk.Scale(ctrl\_card, from\_=0, to=100, orient="horizontal",

command=self.\_on\_seek\_drag)

self.seek.pack(fill="x", padx=12, pady=(4, 10))

self.seek.bind("<Button-1>", lambda e: self.\_set\_drag(True))

self.seek.bind("<ButtonRelease-1>", lambda e: self.\_set\_drag(False, commit=True))

row = ttk.Frame(ctrl\_card, style="Card.TFrame")

row.pack(fill="x", padx=12, pady=(4, 12))

if PIL\_OK:

self.prev\_btn = ttk.Button(row, image=self.prev\_img, command=self.prev, style="Alt.TButton")

self.play\_btn = ttk.Button(row, image=self.play\_img, command=self.play\_pause)

self.stop\_btn = ttk.Button(row, image=self.stop\_img, command=self.stop, style="Alt.TButton")

self.next\_btn = ttk.Button(row, image=self.next\_img, command=self.next, style="Alt.TButton")

self.fs\_btn = ttk.Button(row, image=self.fullscreen\_img, command=self.toggle\_fullscreen, style="Alt.TButton")

self.mute\_btn = ttk.Button(row, image=self.unmute\_img, command=self.toggle\_mute, style="Alt.TButton")

self.prev\_btn.pack(side="left", padx=4)

self.play\_btn.pack(side="left", padx=4)

self.stop\_btn.pack(side="left", padx=4)

self.next\_btn.pack(side="left", padx=4)

self.fs\_btn.pack(side="right", padx=4)

else:

self.prev\_btn = ttk.Button(row, text="⏮", width=4, style="Alt.TButton", command=self.prev)

self.play\_btn = ttk.Button(row, text="▶/⏸", width=6, command=self.play\_pause)

self.stop\_btn = ttk.Button(row, text="⏹", width=4, style="Alt.TButton", command=self.stop)

self.next\_btn = ttk.Button(row, text="⏭", width=4, style="Alt.TButton", command=self.next)

self.fs\_btn = ttk.Button(row, text="⛶ Fullscreen", style="Alt.TButton", command=self.toggle\_fullscreen)

self.mute\_btn = ttk.Button(row, text="🔇/🔊", width=5, style="Alt.TButton", command=self.toggle\_mute)

self.prev\_btn.pack(side="left", padx=4)

self.play\_btn.pack(side="left", padx=4)

self.stop\_btn.pack(side="left", padx=4)

self.next\_btn.pack(side="left", padx=4)

self.fs\_btn.pack(side="right", padx=4)

# volume/mute

volrow = ttk.Frame(ctrl\_card, style="Card.TFrame")

volrow.pack(fill="x", padx=12, pady=(0, 12))

ttk.Label(volrow, text="🔊").pack(side="left")

self.vol = ttk.Scale(volrow, from\_=0, to=100, orient="horizontal", command=self.\_set\_volume)

self.vol.set(self.saved\_volume)

self.\_set\_volume(self.saved\_volume)

self.vol.pack(side="left", fill="x", expand=True, padx=8)

self.mute\_btn.pack(side="right")

# playlist card

right = ttk.Frame(main, style="Card.TFrame")

right.pack(side="left", fill="both", expand=False, padx=(8, 0))

header = ttk.Frame(right, style="Card.TFrame")

header.pack(fill="x", padx=10, pady=(10, 0))

ttk.Label(header, text="Playlist", font=("Segoe UI", 11, "bold")).pack(side="left")

lstwrap = ttk.Frame(right, style="Card.TFrame")

lstwrap.pack(fill="both", expand=True, padx=10, pady=10)

self.listbox = tk.Listbox(lstwrap, bg="#0f131a", fg="#e6e6e6",

selectbackground="#1DB954", activestyle="none",

highlightthickness=0, bd=0, font=("Segoe UI", 10))

self.listbox.pack(side="left", fill="both", expand=True)

sb = ttk.Scrollbar(lstwrap, orient="vertical", command=self.listbox.yview)

sb.pack(side="right", fill="y")

self.listbox.config(yscrollcommand=sb.set)

self.listbox.bind("<Double-Button-1>", lambda e: self.play\_selected())

self.listbox.bind("<Return>", lambda e: self.play\_selected())

pbtns = ttk.Frame(right, style="Card.TFrame")

pbtns.pack(fill="x", padx=10, pady=(0, 10))

ttk.Button(pbtns, text="＋ Add", style="Alt.TButton", command=self.open\_files).pack(side="left", padx=4)

ttk.Button(pbtns, text="📁 Folder", style="Alt.TButton", command=self.open\_folder).pack(side="left", padx=4)

ttk.Button(pbtns, text="🗑 Remove", style="Alt.TButton", command=self.remove\_selected).pack(side="right", padx=4)

ttk.Button(pbtns, text="🧹 Clear", style="Alt.TButton", command=self.clear\_playlist).pack(side="right", padx=4)

# status bar

status = ttk.Frame(self.root, style="Bar.TFrame")

status.pack(fill="x")

self.status\_var = tk.StringVar(value="Ready")

ttk.Label(status, textvariable=self.status\_var, font=("Segoe UI", 9)).pack(anchor="w", padx=14, pady=6)

def \_bind\_shortcuts(self):

self.root.bind("<space>", lambda e: self.play\_pause())

self.root.bind("<Left>", lambda e: self.seek\_relative(-5))

self.root.bind("<Right>", lambda e: self.seek\_relative(+5))

self.root.bind("<Up>", lambda e: self.\_nudge\_volume(+5))

self.root.bind("<Down>", lambda e: self.\_nudge\_volume(-5))

self.root.bind("<f>", lambda e: self.toggle\_fullscreen())

self.root.bind("<Escape>", lambda e: self.\_exit\_fullscreen\_if\_needed())

self.root.bind("<Control-o>", lambda e: self.open\_files())

self.root.bind("<Control-Delete>", lambda e: self.remove\_selected())

self.root.bind("<Prior>", lambda e: self.prev())

self.root.bind("<Next>", lambda e: self.next())

def \_embed\_video\_after\_ready(self):

self.root.after(200, self.\_try\_embed)

def \_try\_embed(self):

try:

wid = self.video\_area.winfo\_id()

if sys.platform.startswith("win"):

self.player.set\_hwnd(wid)

elif sys.platform == "darwin":

self.player.set\_nsobject(wid)

else:

self.player.set\_xwindow(wid)

except Exception:

self.root.after(200, self.\_try\_embed)

# ---------- playlist ----------

def open\_files(self):

files = filedialog.askopenfilenames(

title="Open Media",

filetypes=[("Media files", "\*.\*")])

if not files:

return

added = 0

for f in files:

if f not in self.playlist:

self.playlist.append(f)

self.listbox.insert(tk.END, os.path.basename(f))

added += 1

self.status\_var.set(f"Added {added} file(s).")

if self.current\_index == -1 and self.playlist:

self.current\_index = 0

def open\_folder(self):

folder = filedialog.askdirectory(title="Open Folder")

if not folder:

return

exts = (".mp4", ".mkv", ".avi", ".mov", ".mp3", ".wav", ".flac", ".m4a", ".aac", ".ogg")

items = []

for root\_, \_, files in os.walk(folder):

for name in files:

if name.lower().endswith(exts):

items.append(os.path.join(root\_, name))

if not items:

messagebox.showinfo("No media", "No supported media found in this folder.")

return

added = 0

for p in items:

if p not in self.playlist:

self.playlist.append(p)

self.listbox.insert(tk.END, os.path.basename(p))

added += 1

self.status\_var.set(f"Added {added} file(s) from folder.")

if self.current\_index == -1 and self.playlist:

self.current\_index = 0

def remove\_selected(self):

sel = list(self.listbox.curselection())

if not sel:

return

for idx in reversed(sel):

real = idx

if real == self.current\_index:

self.stop()

self.current\_index = -1

del self.playlist[real]

self.listbox.delete(idx)

if self.playlist:

self.current\_index = min(self.current\_index if self.current\_index != -1 else 0,

len(self.playlist) - 1)

else:

self.current\_index = -1

self.status\_var.set("Removed selection.")

def clear\_playlist(self):

self.stop()

self.playlist.clear()

self.listbox.delete(0, tk.END)

self.current\_index = -1

self.status\_var.set("Playlist cleared.")

def play\_selected(self):

try:

idx = self.listbox.curselection()[0]

except Exception:

return

self.current\_index = idx

self.\_load\_and\_play\_current()

# ---------- playback ----------

def \_load\_and\_play\_current(self):

if self.current\_index < 0 or self.current\_index >= len(self.playlist):

return

path = self.playlist[self.current\_index]

try:

media = self.vlc\_instance.media\_new(path)

self.player.set\_media(media)

self.player.play()

self.is\_playing = True

if PIL\_OK:

self.play\_btn.config(image=self.pause\_img)

else:

self.play\_btn.config(text="⏸")

self.status\_var.set(f"Playing: {os.path.basename(path)}")

self.root.after(300, self.\_update\_total\_length)

self.\_start\_poll()

except Exception as e:

messagebox.showerror("Playback error", str(e))

def play\_pause(self):

if self.player.get\_state() in (vlc.State.NothingSpecial, vlc.State.Stopped) and self.playlist:

if self.current\_index == -1:

self.current\_index = 0

self.\_load\_and\_play\_current()

Return

self.player.pause()

self.is\_playing = self.player.get\_state() == vlc.State.Playing

if PIL\_OK:

if self.is\_playing:

self.play\_btn.config(image=self.pause\_img)

else:

self.play\_btn.config(image=self.play\_img)

else:

if self.is\_playing:

self.play\_btn.config(text="⏸")

else:

self.play\_btn.config(text="▶")

self.root.after(150, self.\_update\_status\_from\_state)

def stop(self):

try:

self.player.stop()

except Exception:

pass

self.\_stop\_poll()

self.seek.set(0)

self.lbl\_cur.config(text="00:00")

self.lbl\_tot.config(text="00:00")

self.length\_ms = 0

self.is\_playing = False

if PIL\_OK:

self.play\_btn.config(image=self.play\_img)

else:

self.play\_btn.config(text="▶/⏸")

self.status\_var.set("Stopped")

def next(self):

if not self.playlist:

return

self.current\_index = (self.current\_index + 1) % len(self.playlist)

self.\_load\_and\_play\_current()

self.listbox.selection\_clear(0, tk.END)

self.listbox.selection\_set(self.current\_index)

self.listbox.see(self.current\_index)

def prev(self):

if not self.playlist:

return

self.current\_index = (self.current\_index - 1) % len(self.playlist)

self.\_load\_and\_play\_current()

self.listbox.selection\_clear(0, tk.END)

self.listbox.selection\_set(self.current\_index)

self.listbox.see(self.current\_index)

# ---------- time/seek ----------

def \_update\_total\_length(self):

ms = self.player.get\_length()

if ms and ms > 0:

self.length\_ms = ms

self.seek.configure(to=ms)

self.lbl\_tot.config(text=fmt\_time(ms))

else:

self.root.after(300, self.\_update\_total\_length)

def \_start\_poll(self):

self.\_stop\_poll()

self.poll\_job = self.root.after(250, self.\_poll)

def \_stop\_poll(self):

if self.poll\_job:

self.root.after\_cancel(self.poll\_job)

self.poll\_job = None

def \_poll(self):

state = self.player.get\_state()

if state == vlc.State.Ended:

self.next()

return

if not self.user\_dragging\_seek:

cur = self.player.get\_time()

if cur is None:

cur = 0

self.seek.set(cur)

self.lbl\_cur.config(text=fmt\_time(cur))

if self.length\_ms <= 0:

self.\_update\_total\_length()

self.poll\_job = self.root.after(250, self.\_poll)

def \_on\_seek\_drag(self, \_val):

if self.user\_dragging\_seek:

self.lbl\_cur.config(text=fmt\_time(int(float(self.seek.get()))))

def \_set\_drag(self, dragging: bool, commit: bool = False):

self.user\_dragging\_seek = dragging

if commit:

try:

t = int(float(self.seek.get()))

self.player.set\_time(t)

except Exception:

pass

def seek\_relative(self, seconds: int):

if self.length\_ms <= 0:

return

cur = self.player.get\_time() or 0

new\_ms = max(0, min(self.length\_ms - 500, cur + seconds \* 1000))

self.player.set\_time(new\_ms)

self.seek.set(new\_ms)

self.lbl\_cur.config(text=fmt\_time(new\_ms))

# ---------- volume ----------

def \_set\_volume(self, val):

try:

v = int(float(val))

except Exception:

v = 80

self.player.audio\_set\_volume(v)

if not self.muted:

self.saved\_volume = v

def \_nudge\_volume(self, delta):

v = max(0, min(100, int(self.vol.get()) + delta))

self.vol.set(v)

self.\_set\_volume(v)

def toggle\_mute(self):

self.muted = not self.muted

self.player.audio\_toggle\_mute()

if PIL\_OK:

if self.muted:

self.mute\_btn.config(image=self.mute\_img)

else:

self.mute\_btn.config(image=self.unmute\_img)

else:

self.mute\_btn.config(text="🔇" if self.muted else "🔊")

if self.muted:

self.saved\_volume = int(self.vol.get())

self.vol.set(0)

else:

self.vol.set(self.saved\_volume)

# ---------- fullscreen ----------

def toggle\_fullscreen(self):

self.fullscreen = not self.fullscreen

self.root.attributes("-fullscreen", self.fullscreen)

def \_exit\_fullscreen\_if\_needed(self):

if self.fullscreen:

self.toggle\_fullscreen()

# ---------- status ----------

def \_update\_status\_from\_state(self):

st = self.player.get\_state()

name = {

vlc.State.Playing: "Playing",

vlc.State.Paused: "Paused",

vlc.State.Stopped: "Stopped",

}.get(st, str(st))

self.status\_var.set(name)

# ---------- drag & drop ----------

def \_on\_drop(self, event):

raw = event.data

items = []

cur = ""

in\_brace = False

for ch in raw:

if ch == "{":

in\_brace = True

cur = ""

elif ch == "}":

in\_brace = False

items.append(cur)

cur = ""

elif ch == " " and not in\_brace:

if cur:

items.append(cur)

cur = ""

else:

cur += ch

if cur:

items.append(cur)

exts = (".mp4", ".mkv", ".avi", ".mov", ".mp3", ".wav", ".flac", ".m4a", ".aac", ".ogg")

to\_add = []

for p in items:

if os.path.isdir(p):

for r, \_, fs in os.walk(p):

for nm in fs:

if nm.lower().endswith(exts):

to\_add.append(os.path.join(r, nm))

elif os.path.isfile(p) and p.lower().endswith(exts):

to\_add.append(p)

if to\_add:

added = 0

for f in to\_add:

if f not in self.playlist:

self.playlist.append(f)

self.listbox.insert(tk.END, os.path.basename(f))

added += 1

self.status\_var.set(f"Added {added} item(s) by drag & drop.")

if self.current\_index == -1:

self.current\_index = 0

# ---------- run ----------

def main():

if DND\_OK:

root = TkinterDnD.Tk()

else:

root = tk.Tk()

app = MiniVLC(root)

root.mainloop()

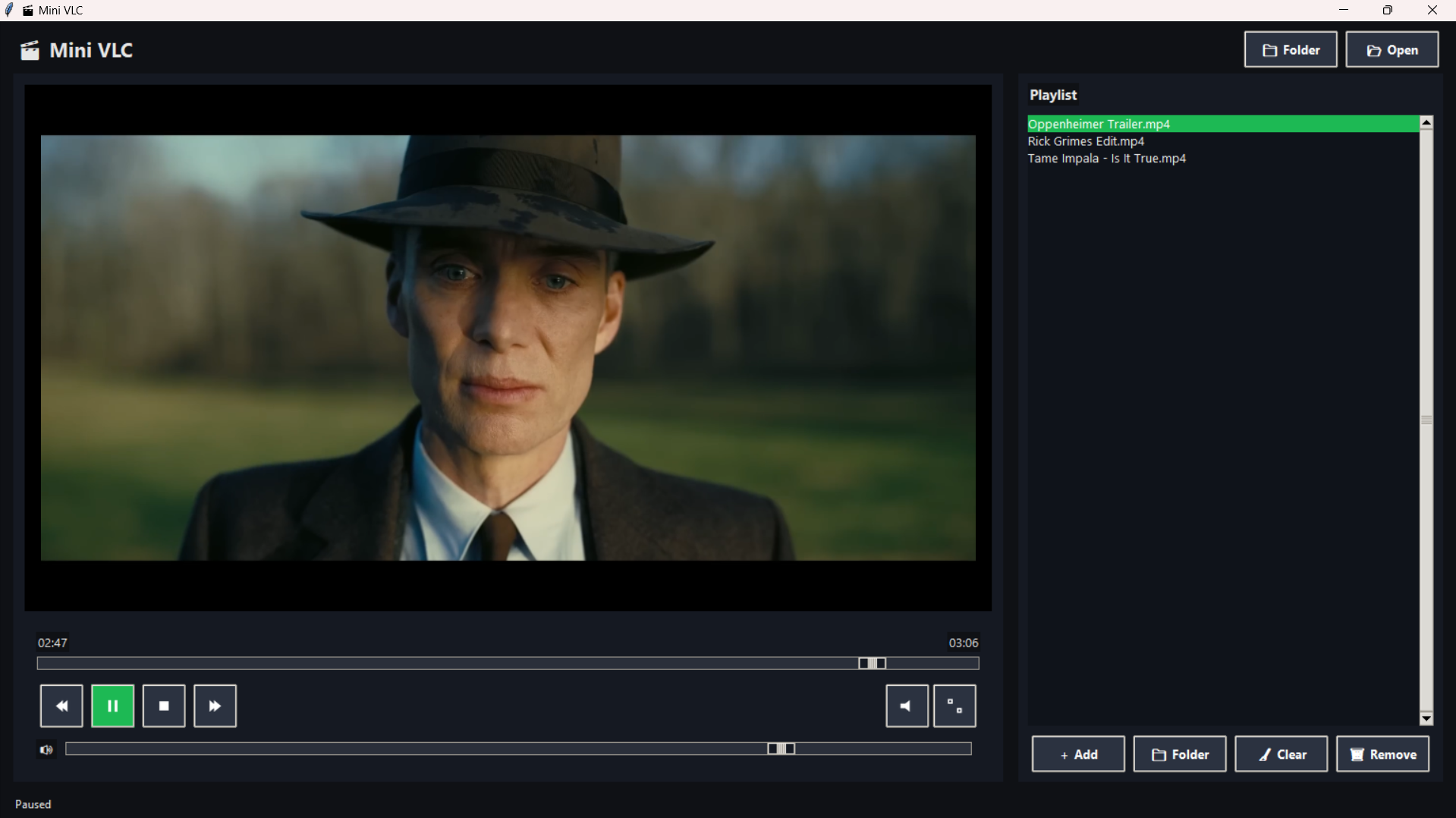
if \_\_name\_\_ == "\_\_main\_\_":

main()

**5.5 Project in Production / Hosted Live**

This is a desktop application and is not hosted live. It runs locally on the user's machine.

**5.6 Hosted Project Screenshot**



**5.7 Results and Reports**

The application successfully meets all its primary objectives. It can load and play media files, control playback, manage a playlist, and display real-time progress. The elegant UI enhances the user experience, making it a professional and practical piece of software.

**6. Evaluation and Testing**

**6.1 Evaluation Metrics**

* **Functionality**: All buttons and controls work as expected.

* **Usability**: The interface is intuitive, and users can quickly understand how to operate the player.

* **Responsiveness**: The GUI resizes gracefully, and the controls remain accessible.

**6.2 Confusion Matrix / ROC Curve**

These metrics are not applicable for this type of software as it is not an AI/ML project.

**6.3 Functional Testing and Debug Logs**

Extensive functional testing was performed to ensure that all playback controls, playlist management, and file handling worked correctly. The code includes try-except blocks and messagebox alerts to handle potential errors and provide user feedback.

**7. Challenges and Troubleshooting**

**7.1 Technical Challenges**

* **Integrating VLC with Tkinter**: A major challenge was correctly embedding the VLC video window within the Tkinter frame, which required platform-specific code.

* **Real-time Updates**: Ensuring the seek bar and time labels updated smoothly without freezing the GUI required a dedicated polling mechanism using root.after().

* **Drag-and-Drop**: Implementing drag-and-drop to handle both single files and entire folders was complex due to the varying data formats returned by the OS.

**7.2 Solutions Implemented**

* The player.set\_hwnd() and other platform-specific functions were used to embed the video.

* A dedicated \_poll function with root.after() was created to handle real-time updates.

* The drag-and-drop handler was written with a robust parsing algorithm to handle different file path formats.

**8. Future Scope and Enhancements**

* **Playlist Saving**: Add the ability to save and load playlists to a file.

* **Media Metadata**: Implement a feature to display metadata (e.g., artist, album, duration) of audio files.

* **Video Subtitles**: Add support for loading and displaying subtitle files (.srt, etc.).

* **Streaming Support**: Extend the application to play media from URLs, not just local files.

**9. Conclusion**

The Mini VLC media player is a successful software-based major project that demonstrates a wide range of skills, from professional GUI design to complex media handling. The application is a testament to the power of open-source libraries and object-oriented programming. It serves as a strong foundation that can be expanded with more advanced features in the future.

**10. References**

* Official Tkinter Documentation
* Python-VLC GitHub Repository