COMP1752 Object-Oriented Programming: Jukebox Simulation Report

Title Page

- Course Title: COMP1752 Object-Oriented Programming
- Assignment: Jukebox Simulation
- Duong Thanh Dat ID: 001419280
- Submission Date 22/4/2025

Table of Contents

1. Introduction

This report documents the analysis, design, and implementation of a jukebox simulation application. The project extends an existing system by adding enhanced functionality to create a more comprehensive music management solution. The original application provided basic track viewing capabilities, allowing users to retrieve and display track information. This project expands the functionality with two key features: a playlist creation system and track rating management.
The development process follows a systematic approach, beginning with initial design considerations, progressing through implementation phases, and concluding with thorough testing and evaluation. This report details each phase chronologically, providing insight into the design decisions, technical implementation, and quality assurance processes applied throughout the project lifecycle.

2. Design and Development

2.1. Design:

The user interface design maintains visual consistency across all windows while incorporating new functionalities. A cohesive color scheme was implemented throughout the application, featuring a wheat-colored background, black text, dark orange button backgrounds, and white text on buttons.

The track player and view track interfaces maintain their original layouts with updated styling, as shown in Image 1 and Image 2.

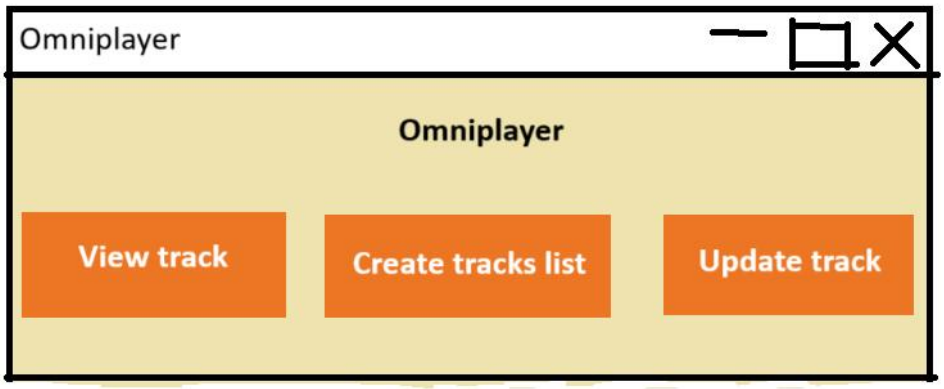


Image 1

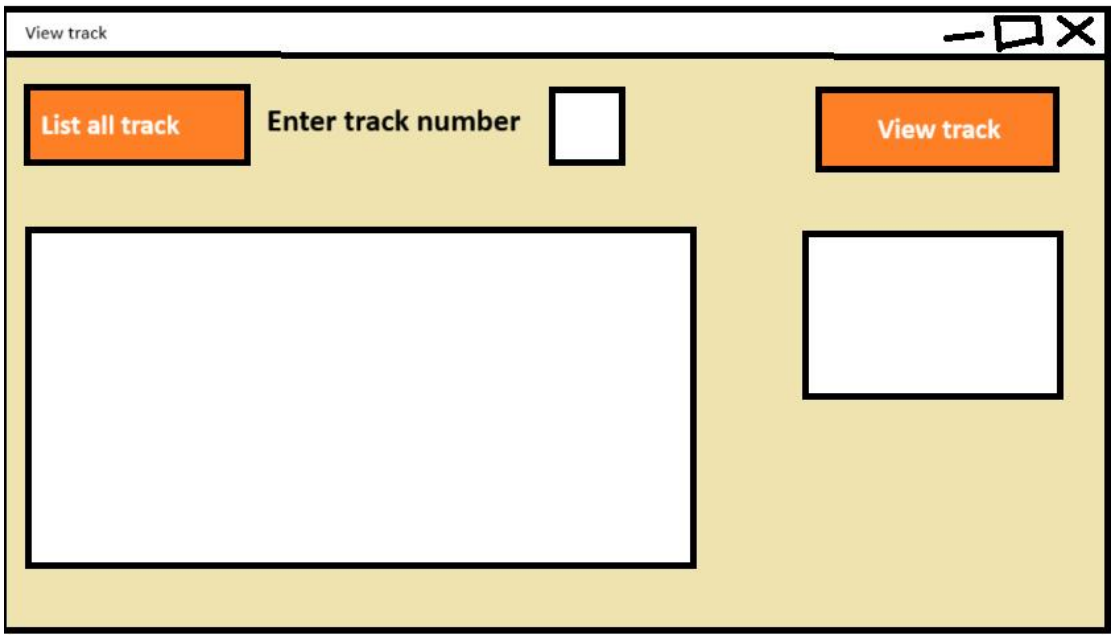


Image 2

The create tracks list interface introduces new functionality with a carefully designed layout. It features an input field for track numbers, a button to add tracks to the playlist, and a reset button with a red background to provide clear visual feedback for list deletion. A scrollable text area displays all tracks in the playlist, showing key information including name, artist, and play count. A play button allows users to simulate playing all tracks in the playlist, which increments their play count.

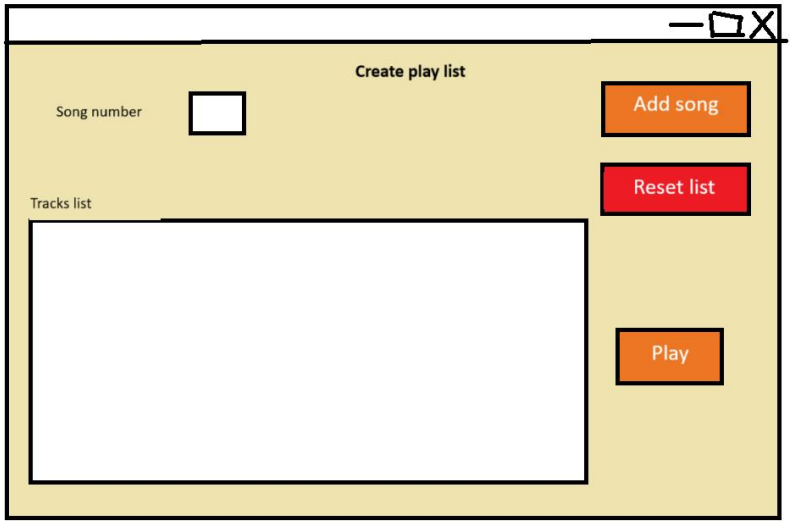


Image 3

The update tracks window provides a specialized interface for modifying track ratings. It contains two input fields for track number and new rating, along with an update button. When a valid update is performed, the window dynamically resizes to display the updated track information in a text box.

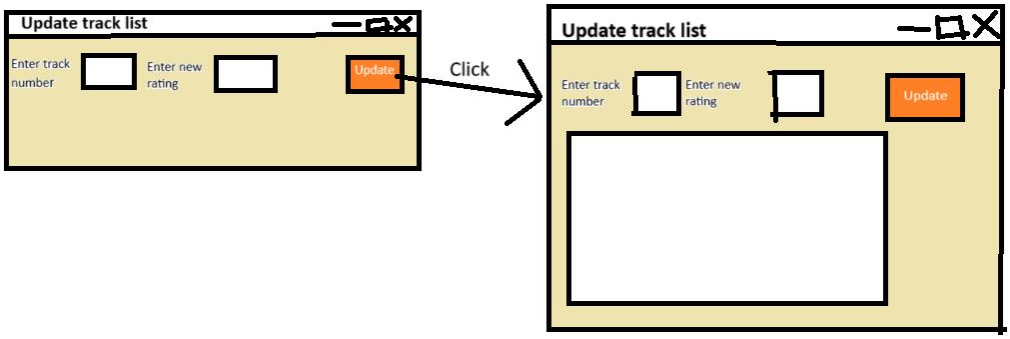


Image 4

2.2. Development

- **Planning**

The development approach was based on extending the original application while maintaining its core functionality. The process began with an analysis of the existing codebase to identify suitable extension points and integration strategies. Development was organized into two primary phases:

- Creating graphical user interfaces aligned with the design specifications
- Implementing the functional logic for each new feature

- Development

The first development step focused on constructing graphical interfaces that closely matched the design specifications. The track player and view tracks interfaces were updated with new styling while preserving their layouts (Image 5 and Image 6). The new interfaces for create tracks list and update tracks were built using inheritance

from tkinter.TK, providing consistent behavior and appearance (Image 7, Image 8 and Image 9).
Following the interface development, functional logic was implemented for each component:

The only function in update track is update track rating, if track number and rating are valid, the track with that key will change the rating to the rating in the rating entry, the window change size, text box appear and track information display on text box(Image 14). If any of the inputs is invalid, the text box appear and write invalid inputs depend on what is invalid.

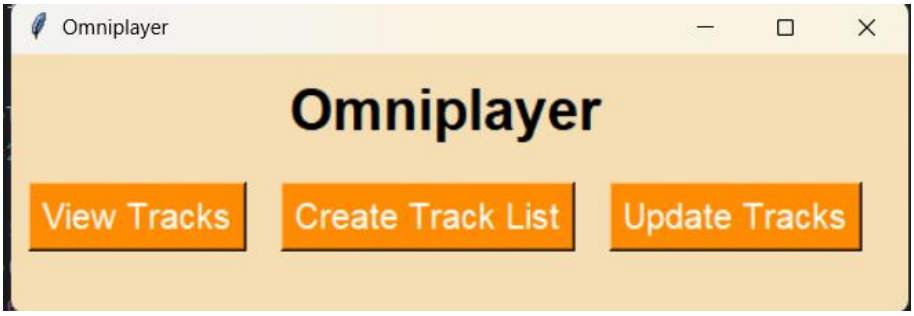


Image 5

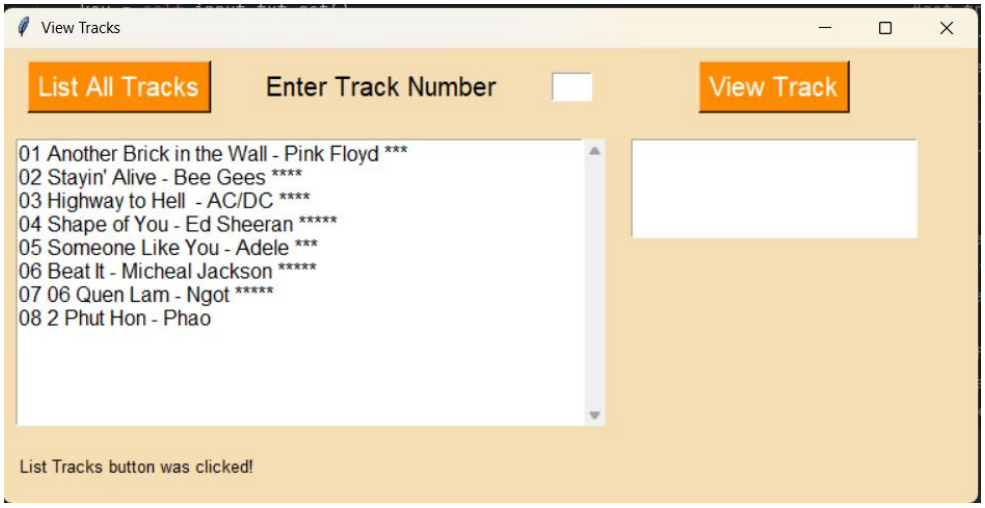


Image 6

Image 7

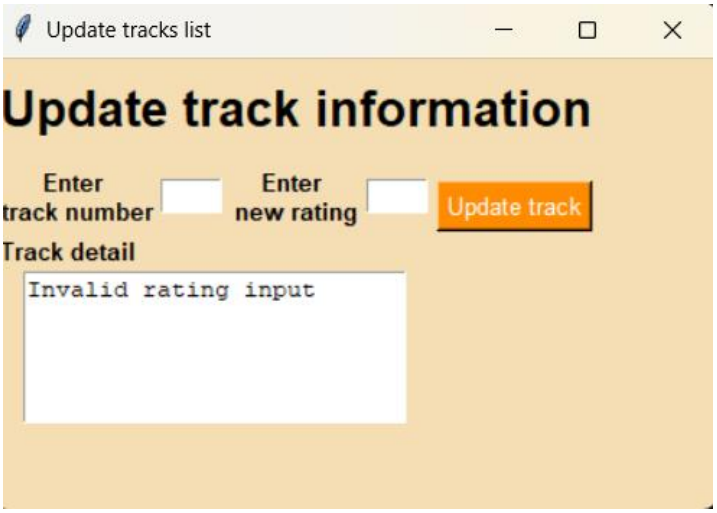


Image 8

Create Tracks List Functionality:

Add Song Function: This feature allows users to add tracks to the playlist by entering a track number. The system validates the input against the library and adds the track if it exists (Image 10). If the track does not exist, appropriate feedback is displayed (Image 12).

Reset List Function: This feature clears all tracks from the current playlist and displays a confirmation message (Image 13).

Play Tracks Function: This feature simulates playing all tracks in the playlist by incrementing their play count (Image 15).

Saving the playlist in external file: This feature keep the playlist in a file for further modification(More detail in Innovation section)

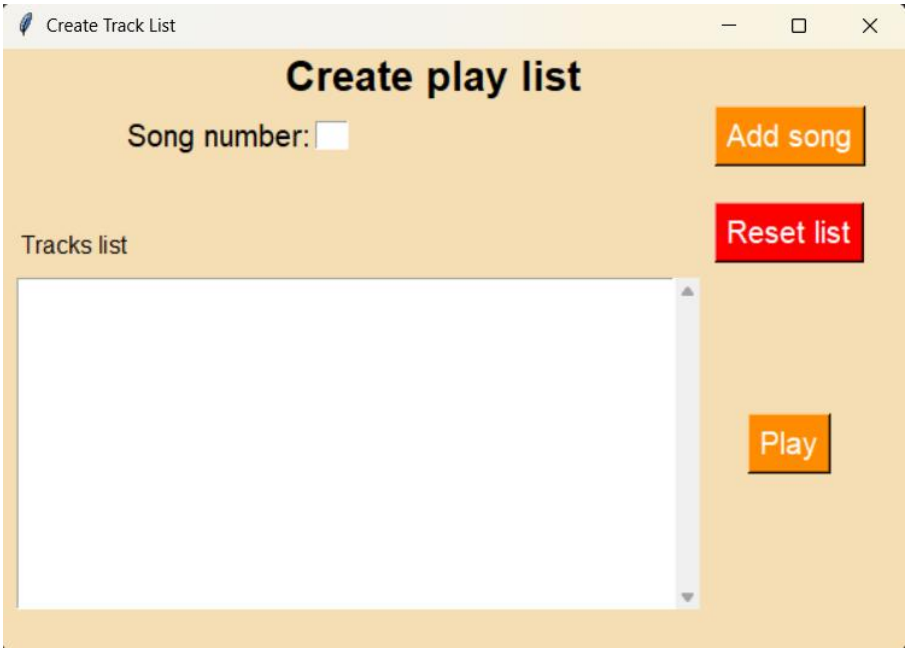


Image 9

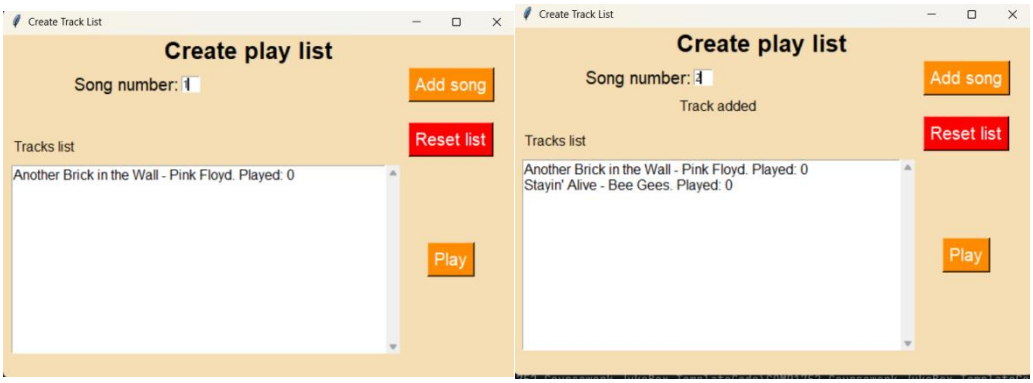


Image 10

```
def add_song(self):  1 usage   dry700
    #avoid the non-number input
    try :
        songnum = int(self.song_number.get())
        key = "%02d" % songnum # convert 1 digit number to 2 digits string key
        # check the validation of number and add song
        if key in list(library):  #check if track exist
            self.play_list.append(key) #add track key to playlist
            self.lbl7.configure(text="Track added") #write confirm text
            #self.print_detail()#write all track in playlist on screen
        else:
            self.lbl7.configure(text="Song doesn't exist")#write status on screen
        self.write_list_file()#write tracks key in playlist on save file
        self.print_detail()#write playlist on scrolltext
    except ValueError:#if non number input
        self.lbl7.configure(text="Invalid input")#write error text on screen
```

Image 11 Source code of the add song button

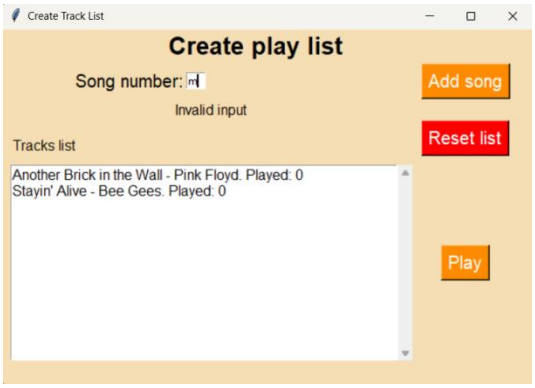


Image 12

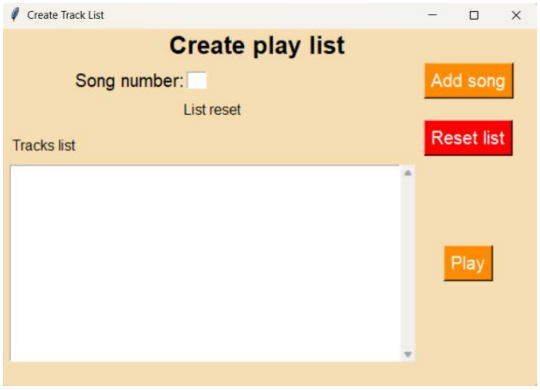


Image 13

```
def reset_list(self):  1 usage   dry700
    self.play_list = [] #reset the list
    self.detail = ""#reset the detail
    self.write_list_file()#write playlist on scrolltext
    self.print_detail()
    self.lbl7.configure(text="List reset")  # write error text on screen
```

Image 14 Source code of reset list button

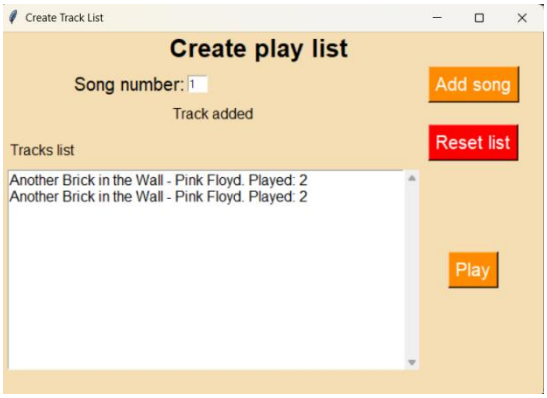


Image 15

```
def play_tracks_list(self):  1 usage   dry700
    #increase play count of tracks in playlist
    for key in self.play_list:
        lib.increment_play_count(key)
    self.print_detail()#write playlist on scrolltext
```

Image 16 Source code of play button

```
    def print_detail(self):  4 usages   dry700
        tracks_detail ='' #initiate write content
        for key in self.play_list: #get tracks detail
            name = lib.get_name(key) #get track name
            artist = lib.get_artist(key) #get track artist
            play_count = lib.get_play_count(key) #get play count
            tracks_detail += f"{name} - {artist}. Played: {play_count}\n" #add track detail to print content
        self.detail = tracks_detail #set writen content on scroll text
        set_text(self.list_txt, self.detail)#write content on scroll text
```

Image 17 Source code of print detail which use to print the playlist on text box

Update Track Functionality:

The update track feature provides a mechanism for modifying track ratings. When a valid track number and rating are provided, the system updates the rating and displays the modified track information in an expanded window (Image 17). The system includes comprehensive input validation to ensure both track numbers and ratings are valid before processing updates.

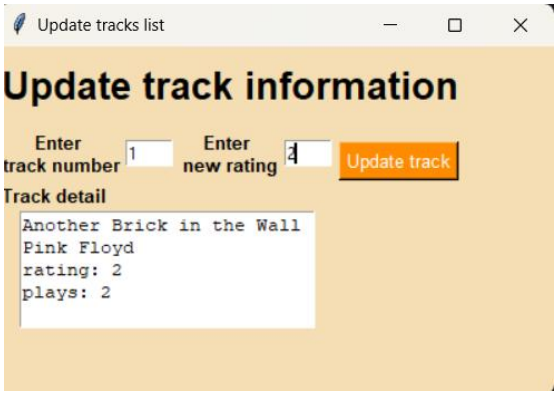


Image 18

```
def update_track(self):
    self.change_window() #change window size and add text block
    #avoid non number rating input
    try:
        new_rating = int(self.rating_entry.get()) #convert rating into integer
        if new_rating > 5 or new_rating < 0: #avoid invalid rating number
            raise Exception("Invalid rating input")#raise an exception

    except ValueError:#if non number input
        set_text(self.new_detail, "Invalid rating input")#write text on text box
        return
    except Exception as e:
        set_text(self.new_detail, e)#write text on text box
        return

    #avoid non number input
    try:
        track_number = int(self.number_entry.get())#get track number input
        key = "%02d" % track_number #convert int to str key
        name = lib.get_name(key) #get track name
        lib.set_rating(key,new_rating) #get track rating
        if name is not None:  # if track exist then
            artist = lib.get_artist(key)  # get track name
            rating = lib.get_rating(key)  # get track rating
            play_count = lib.get_play_count(key)  # get track play count
```

```
        track_details = f"{name}\n{artist}\nrating: {rating}\nplays: {play_count}"  # track detail text
        set_text(self.new_detail, track_details)  # track detail text display
    else:  # if track doesn't exist
        set_text(self.new_detail, f"Track {key} not found")  # display track not found
except ValueError: # if non number input
    set_text(self.new_detail, "Invalid number input")#write error text on screen
```

Table 1 Source code of update button

```
def change_window(self):  1 usage  dry700
    self.geometry("400x250") #change window size

    #Track detail label and text box
    self.lbl_current = tk.Label(self, text="Track detail", font=("Helvetica", 10, "bold"), bg="wheat")
    self.lbl_current.grid(row=3, column=0, sticky=tk.SW)
    self.new_detail = tk.Text(self, width=26, height=5, wrap="none")
    self.new_detail.grid(row=4, column=0, columnspan=5)
```

Image 19 change window source code

3. Testing and Faults

A systematic testing approach was employed to verify the application's functionality and identify potential issues. The testing methodology utilized a structured test case format with five key columns: Test Subject, Input, Expected Output, Actual Output, and Result. Testing covered three main categories:

GUI appearance and layout

Button functionality and behavior

Input validation and error handling

The test results, detailed in Table 1 (Appendix C: Test Table and Results), demonstrate that all core functionalities perform as expected. Several minor issues were identified and resolved during testing, including:

Input normalization: The system now correctly processes inputs such as "1" and "01" consistently, treating them as identical references to track 01.

Input validation: Comprehensive validation was implemented to handle invalid inputs gracefully and provide appropriate user feedback.

One unresolved issue remains: when reading library files with invalid rating values, the play count is reset to zero. This issue has been documented for future resolution.

Unit testing was also performed for the LibraryItem class using pytest. The testing suite includes three primary test functions:

Constructor test: Verifies that attributes are correctly initialized

Information test: Confirms that the information line returns correctly formatted data

Stars test: Validates that the star display mechanism accurately represents ratings

Pytest code and result in appendix section

4. Conclusions, Further Development, and Reflection

Conclusions

This project successfully delivered a functional jukebox simulation with enhanced capabilities for managing music collections. The application now supports viewing track details, creating customized playlists, and updating track ratings. The development process effectively applied object-oriented programming concepts, including class design, inheritance, and graphical user interface implementation.

The project demonstrates the practical application of file I/O operations, data structure management, and user interface design principles. The resulting application provides a cohesive and intuitive user experience while maintaining robust data management capabilities.

Further Development

With additional development time, several enhancements could be implemented to expand the application's functionality:

Advanced Search and Filtering: Development of a sophisticated search engine to locate tracks based on multiple criteria such as artist, genre, or release date.
Library Management System: Implementation of features to add new tracks to the library, improving the system's flexibility.
User Experience Enhancements: Refinement of the interface with advanced sorting options, visualization features, and keyboard shortcuts.

Reflection

This project represents a significant milestone in my application of object-oriented programming concepts to develop a practical software solution with multiple integrated functions. The jukebox simulation project provided a comprehensive platform to implement and demonstrate various programming techniques learned throughout the semester, from basic class design to more complex concepts such as inheritance, encapsulation, and polymorphism.

The development process was particularly illuminating in highlighting the importance of proper planning and design before implementation. Initially, I underestimated the complexity involved in integrating new components with existing code. This challenge required me to carefully analyze the original codebase to understand its structure and identify appropriate integration points. The experience reinforced the value of creating detailed design documentation and class diagrams before beginning implementation, a practice I will certainly adopt in future projects.

User interface design proved both challenging and rewarding. Balancing aesthetic considerations with functional requirements demanded careful attention to detail and a thorough understanding of the tkinter framework. I found that small details such as consistent color schemes, button placements, and feedback mechanisms significantly enhanced the overall user experience. The process of designing and implementing the interface components deepened my appreciation for user-centered design principles.

While the technical implementation leveraged my existing programming knowledge effectively, I encountered several unexpected challenges during development. Error handling and input validation, in particular, required more attention than initially anticipated. Ensuring robust operation under various input conditions highlighted the importance of comprehensive testing and defensive programming practices. The unresolved

issue with play count resetting when reading files with invalid ratings serves as a valuable reminder that edge cases must be thoroughly considered during development.

File I/O operations presented another area of learning. Implementing the persistence features for playlists and library data required careful consideration of file formats, parsing strategies, and error handling mechanisms. The experience provided practical insights into data serialization and storage techniques that will be valuable in future software development endeavors.

Documentation proved to be one of the most challenging aspects of the project. Creating comprehensive documentation that effectively communicates both technical details and design rationale required a different skill set than programming. I found it particularly difficult to articulate complex implementation details in a clear, concise manner that would be understandable to readers with varying levels of technical expertise. This experience has motivated me to focus more attention on developing my technical writing skills.

As I reflect on this project in its entirety, I recognize both its strengths and limitations. While the application successfully implements all required functionality, there are opportunities for improvement in areas such as code organization, error handling, and user experience refinement. These insights will inform my approach to future projects, where I plan to place greater emphasis on modular design, comprehensive testing, and user feedback collection throughout the development process.

5. Innovations

- Innovation 1:Save the playlist in an external file

The first innovation implements persistence for user-created playlists by storing them in an external file:

- **Functionality**: The system saves playlists created through the create tracks list interface to an external file, allowing users to retrieve their playlists across sessions.
- **Implementation**: This feature requires two new functions for reading from and writing to the playlist file. The read function parses the file content, splitting it by commas and converting the data to a playlist structure. The write function saves the current playlist data, overwriting previous content with updated information.
- **Benefits**: This innovation enhances user experience by preserving playlists between application sessions, eliminating the need to recreate playlists repeatedly.
- **Limitations**: The system is potentially vulnerable to file corruption if the playlist file is manually edited with incorrect formatting.

```
def write_list_file(self):
    f = open("_play_list.csv","w")#open and overwrite the file
    for track in self.play_list:
        f.write(f'{track},')#write keys in playlist on save file
    f.close()#close file

def read_list_file(self):
    f = open("_play_list.csv",'r')#open file in readonly mode
    contents = f.read()#read all tracks key as string
    if contents == '':#if string is empty
        f.close()#close file
        return [] #return empty list
```

```
    else:
        contents = contents.split(',') #convert string into list
        f.close()#close file
        if contents[len(contents)-1] == '':
            contents.pop(len(contents)-1)
        return contents #return key list
```

Table 2 Source code to read and write the playlist.csv file

- Innovation 2: Save the tracks library in a csv file

  The second innovation implements a CSV-based storage system for the track library:

  - **Functionality**: Track data is stored in a CSV file, allowing modifications to persist after the application is closed.
  - **Implementation**: The system creates and manages a CSV file where each track occupies one line with comma-separated attributes (name, artist, rating, play count). The read function parses this data to populate the library dictionary, while the write function updates the file with current library information whenever changes occur.
  - **Benefits**: This innovation ensures that track data modifications, including rating updates and play count increments, are preserved between application sessions.
  - **Limitations**: As with the playlist persistence feature, this system is susceptible to errors if the CSV file is incorrectly modified outside the application.

```
def update_library():
    f = open("_library.csv", "w")#open file to write
    for i in range(1, len(library) + 1): #write all track in library into file
        f.write(f'{library["%02d" % i].name},{library["%02d" % i].artist},{library["%02d" % i].rating},'
                f'{library["%02d" % i].play_count}\n')
    f.close()#close file

def read_library():
    library={} #initiate track library
    with open("_library.csv","r") as f: #open save file in readonly mode and auto close
        contents = f.readlines() #read all track in file save
        tracks_detail = [] #initiate detail list
        for content in contents:
            tracks_detail.append(content.strip().split(','))#split detail texts into lists
        i = 1 #start from first key in library
        for detail in tracks_detail:
            try: #avoid invalid rating and play count
                if int(detail[2]) > 5:#if rating greater than 5
                    library["%02d" % i] = LibraryItem(detail[0],detail[1],5)#add track with rating = 5
                elif int(detail[2]) < 0: #if rating lesser than 0
                    library["%02d" % i] = LibraryItem(detail[0],detail[1])#add track with rating = 0
                else:#if rating is valid
                    library["%02d" % i] = LibraryItem(detail[0],detail[1],int(detail[2]))#add track
                library["%02d" % i].play_count = int(detail[3])#set play count
                i+=1 #increase key index
            except IndexError: #if there is no play count
                library["%02d" % i] = LibraryItem(detail[0],detail[1],int(detail[2]))#add track playcount = 0
                i += 1 #increase key index
            except ValueError:#if there is no rating or play count
                library["%02d" % i] = LibraryItem(detail[0], detail[1])#add track with playcount = 0 and rating
```

```
= 0
            i += 1 #increase key index

    return library #return the track library
```

Table 3 Source code to read and write the library.csv file

Appendices

Appendix A: Commented Code (Stage 1)

```
import tkinter as tk #import tkinter
import tkinter.scrolledtext as tkst #import scroll text display

import track_library as lib #import library
import font_manager as fonts #import custom font


def set_text(text_area, content): #display on scroll text function
    text_area.delete("1.0", tk.END) #delete all previous text
    text_area.insert(1.0, content) #display new text


class TrackViewer(): #create view track class
    def __init__(self, window): #class initiate function
        window.geometry("750x350") #set window size
        window.title("View Tracks") #set title
        window.configure(bg="wheat") #set background colour

        #list track button
        list_tracks_btn = tk.Button(window, text="List All Tracks", command=self.list_tracks_clicked
                                    ,background="dark orange",foreground="white")
        list_tracks_btn.grid(row=0, column=0, padx=10, pady=10)

        #song number input label and entry
        enter_lbl = tk.Label(window, text="Enter Track Number",bg="wheat")
        enter_lbl.grid(row=0, column=1, padx=10, pady=10)
        self.input_txt = tk.Entry(window, width=3)
        self.input_txt.grid(row=0, column=2, padx=10, pady=10)

        #check track button
        check_track_btn = tk.Button(window, text="View Track", command=self.view_tracks_clicked
                                    ,background="dark orange",foreground="white")
        check_track_btn.grid(row=0, column=3, padx=10, pady=10)

        #scroll text to display tracks list
        self.list_txt = tkst.ScrolledText(window, width=48, height=12, wrap="none")
        self.list_txt.grid(row=1, column=0, columnspan=3, sticky="W", padx=10, pady=10)

        #track detail text display
        self.track_txt = tk.Text(window, width=24, height=4, wrap="none")
        self.track_txt.grid(row=1, column=3, sticky="NW", padx=10, pady=10)

        #button status text
        self.status_lbl = tk.Label(window, text="", font=("Helvetica", 10),bg="wheat")
        self.status_lbl.grid(row=2, column=0, columnspan=4, sticky="W", padx=10, pady=10)

        self.list_tracks_clicked() #tracks list display when initiate window

    def view_tracks_clicked(self):                                                  #display track function
        key = self.input_txt.get()                                                  #get track number from entry
        name = lib.get_name(key)                                                    #get track detail from key
        if name is not None:                                                        #if track exist then
            artist = lib.get_artist(key)                                            #get track name
            rating = lib.get_rating(key)                                            #get track rating
            play_count = lib.get_play_count(key)                                    #get track play count
            track_details = f"{name}
{artist}
rating: {rating}
plays: {play_count}"  #track detail text
            set_text(self.track_txt, track_details)                                 #track detail text display
        else:                                                                       #if track doesn't exist
            set_text(self.track_txt, f"Track {key} not found")                  #display track not found
        self.status_lbl.configure(text="View Track button was clicked!")            #display view track button was clicked

    def list_tracks_clicked(self):                                          #list all tracks function
        track_list = lib.list_all()                                         #get all tracks detail
        set_text(self.list_txt, track_list)                                 #display all tracks
        self.status_lbl.configure(text="List Tracks button was clicked!")   #display list track button was clicked

if __name__ == "__main__":  # only runs when this file is run as a standalone
    window = tk.Tk()        # create a TK object
    fonts.configure()       # configure the fonts
    TrackViewer(window)     # open the TrackViewer GUI
    window.mainloop()       # run the window main loop, reacting to button presses, etc
```

Image 20

Appendix B: Library_item test file

```
from library_item import LibraryItem

test_items = []
test_items.append({"name":"Beat It", "artist":"Micheal Jackson", "rating":5})
test_items.append({"name":"06 Quen Lam","artist":"Ngot","rating":4})
test_items.append({"name":"02 Mo Lam Ma","artist":"Ngot","rating":3})
test_items.append({"name":"Devourer of Gods","artist":"DM DOKURO","rating":4})

def test_init():
    for item in test_items:
        testing_item = LibraryItem(item["name"],item["artist"],item["rating"])
        assert testing_item.name == item["name"]
        assert testing_item.artist == item["artist"]
        assert testing_item.rating == item["rating"]
        assert testing_item.play_count ==0

def test_info():
    for item in test_items:
        testing_item = LibraryItem(item["name"],item["artist"],item["rating"])
        assert testing_item.info() == f"{item["name"]} - {item["artist"]} {testing_item.stars()}"

def test_stars():
    for item in test_items:
        testing_item = LibraryItem(item["name"],item["artist"],item["rating"])
        expected_star = ''
        for i in range(item['rating']):
            expected_star += '*'
        assert testing_item.stars() == expected_star

if __name__ == "__main__":
    test_init()
    test_info()
    test_stars()
```

Table 4 Source code of Library test file

```
============================= test session starts ==============================
collecting ... collected 3 items

library_item_test.py::test_init PASSED                                   [ 33%]
library_item_test.py::test_info PASSED                                   [ 66%]
library_item_test.py::test_stars PASSED                                  [100%]

============================== 3 passed in 0.06s ===============================

Process finished with exit code 0
```

Image 21 Test result

Appendix C: Test Table and Results

Testing subject	Input	Expected output	Actual output	Test result
Track player GUI	None	GUI looks like Image 1	GUI looks like Image 1	Pass (Image 5)
Track viewer GUI	None	GUI looks like Image 2	GUI looks like Image 2	Pass (Image 6)
Create tracks list GUI	None	GUI looks like Image 3	GUI looks like Image 3	Pass (Image 9)
Update track GUI	None	GUI look like Image 4	GUI look like Image 4	Pass (Image 7 and Image 8)
Update track's update button	Any input or None in Track number input Non-number or none in Rating input	Update track GUI With invalid rating input on text box	Update track GUI With invalid rating input on text box	Pass (Image 22)
Update track's update button	Non-number or None in Track number input Number from 1 to 5 in rating input	Update track GUI with invalid number input on text box	Update track GUI with invalid number input on text box	Pass (Image 23)
Update track's update button	1 in Track number input 2 in rating input	Update track GUI with " Another Brick on the Wall Pink Floid Rating:2 Play:0" on text box	Update track GUI with " Another Brick on the Wall Pink Floid Rating:2 Play:0" on text box	Pass (Image 24)
Create track list' Add track button	Non-number input	Create track list GUI with invalid input label below entry	Create track list GUI with invalid input label below entry	Pass (Image 25)
Create track list' Add track button	Number lesser than 1 or greater than 8, in this case , 9	Create track list GUI with Song doesn't exist label below entry	Create track list GUI with Song doesn't exist label below entry	Pass (Image 26)
Create track list' Add track button	1 or 01	Create track list GUI with Track added label below entry and first track in library on scroll text box	Create track list GUI with Track added label below entry and first track in library on scroll text box	Pass (Image 27)
Create track list' Play button	None	Create track list GUI with first track in library on scroll text box and play count increase by one for each track in list	Create track list GUI with first track in library on scroll text box and play count increase by one for each track in list	Pass (Image 28)
Create track list' Reset list button	None	Create track list GUI with List reset label below entry and empty scroll text box	Create track list GUI with List reset label below entry and empty scroll text box	Pass (Image 29)
Play list save file edit and open Create track list GUI	01	Create track list GUI with first track in library on scroll text box	Create track list GUI with first track in library on scroll text box	Pass (Image 30, Image 31)

Library save file and open View tracks GUI	None	View tracks GUI with all the track in library file on scroll text box	View tracks GUI with all the track in library file on scroll text box	Pass (Image 32, Image 33)
Edit library save file and open View tracks, press view track button	Believer,Imagine Dragon,1 in library file 09 in track number input	View tracks GUI with all the track in library file and the new track with name, artist, rating : 1 on scroll text box and playcount:0 on text box	View tracks GUI with all the track in library file and the new track with name, artist, rating : 1 on scroll text box and playcount:0 on text box	Pass (Image 34, Image 35)
Edit library save file and open View tracks, press view track button	Thunder,Imagine Dragon,,4	View tracks GUI with all the track in library file and the new track with name, artist, rating : 0 on scroll text box and playcount:4 on text box	View tracks GUI with all the track in library file and the new track with name, artist, rating : 0 on scroll text box and playcount:0 on text box	Fail (Image 36, Image 37)

Table 5

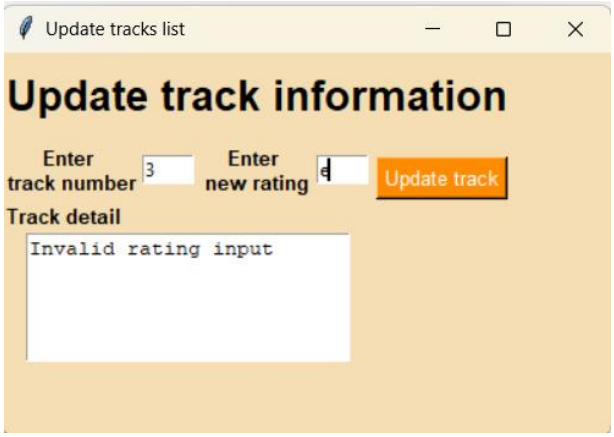


Image 22

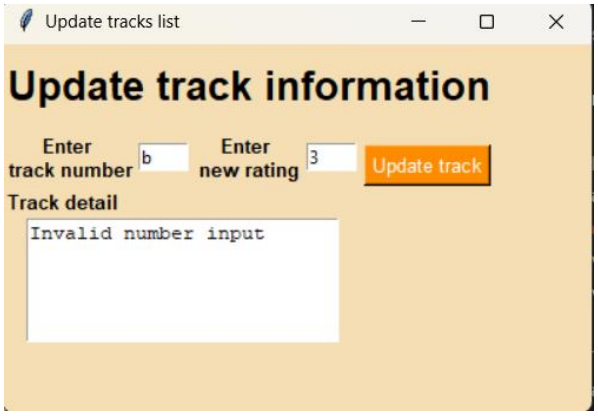


Image 23

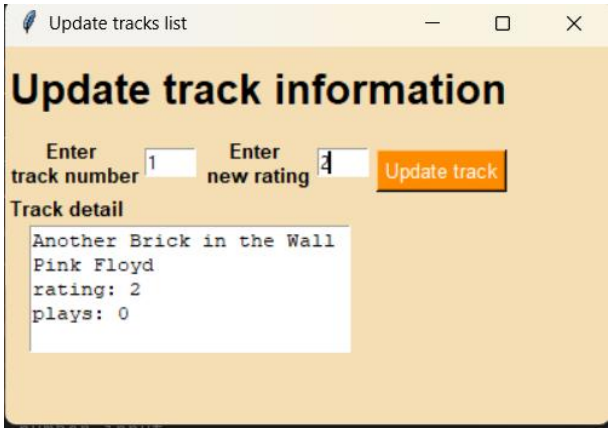


Image 24

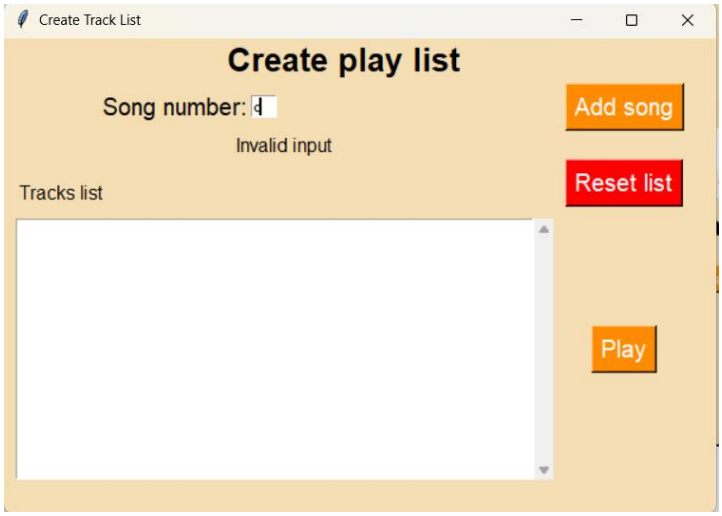


Image 25

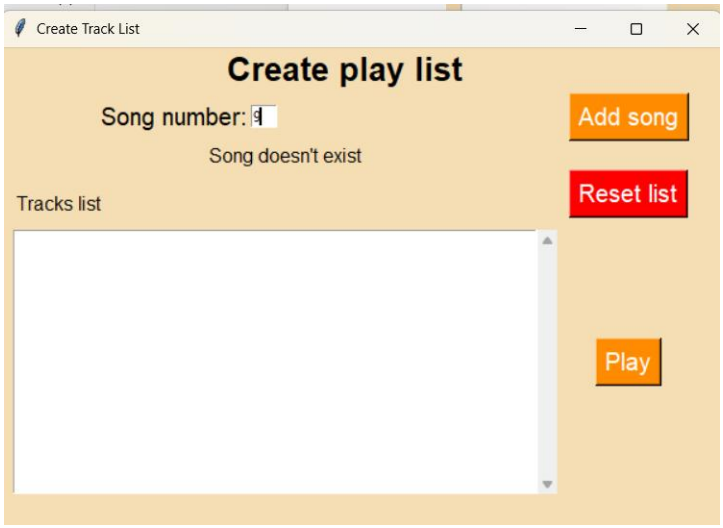


Image 26

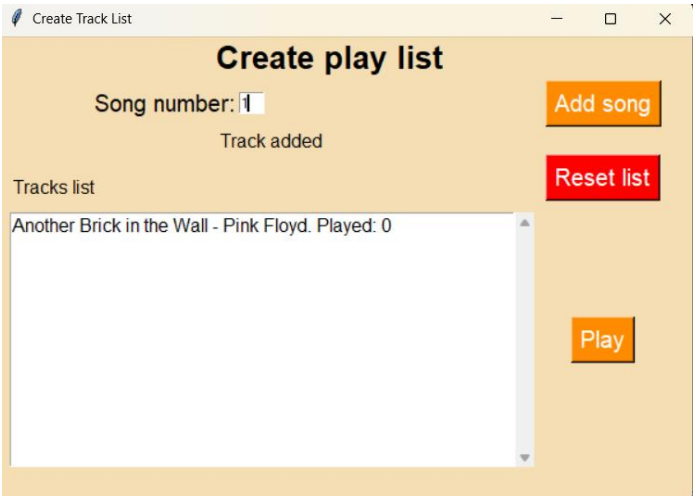


Image 27

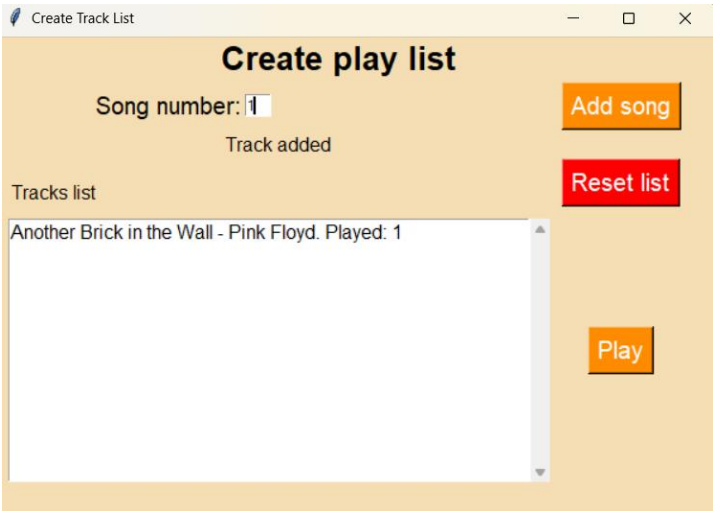


Image 28

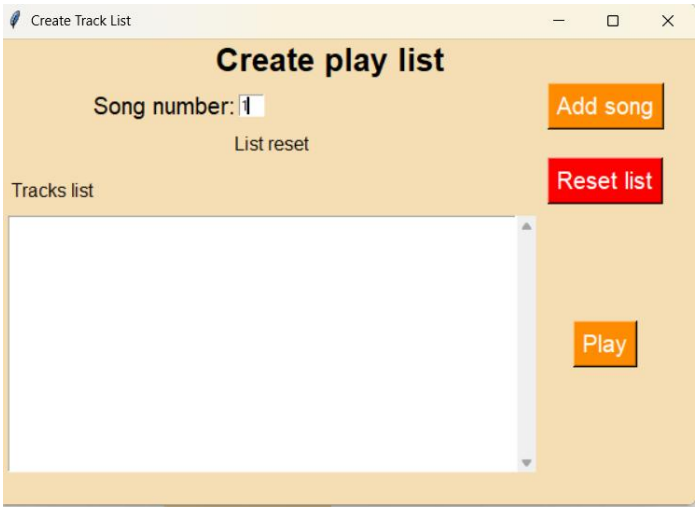


Image 29

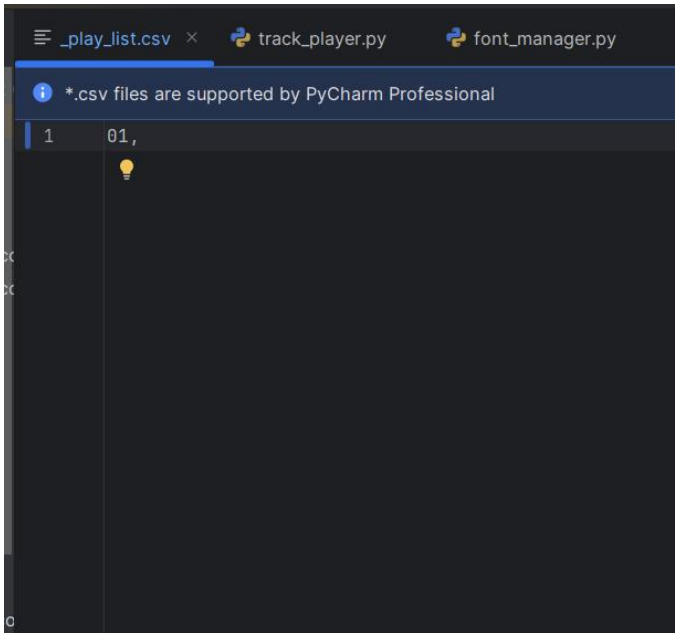


Image 30

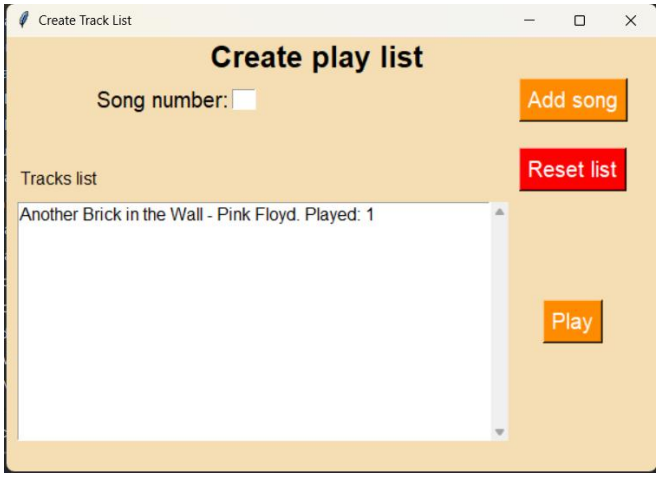


Image 31

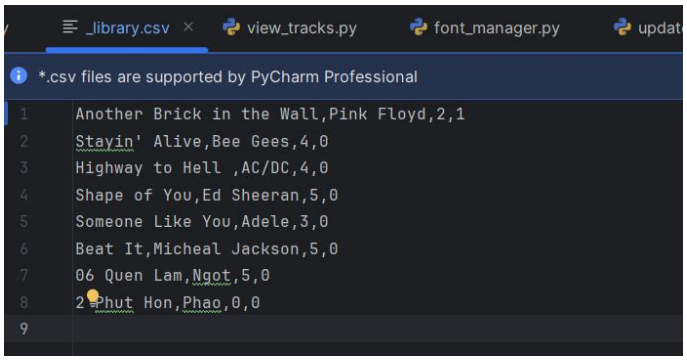


Image 32

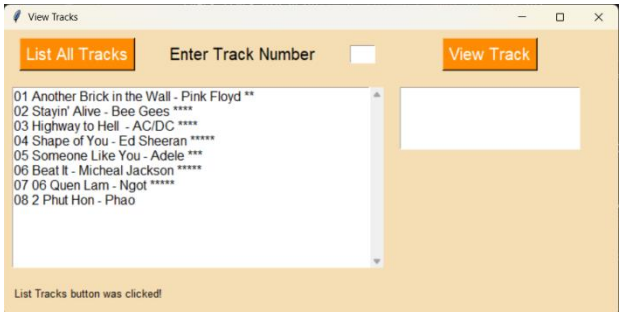


Image 33

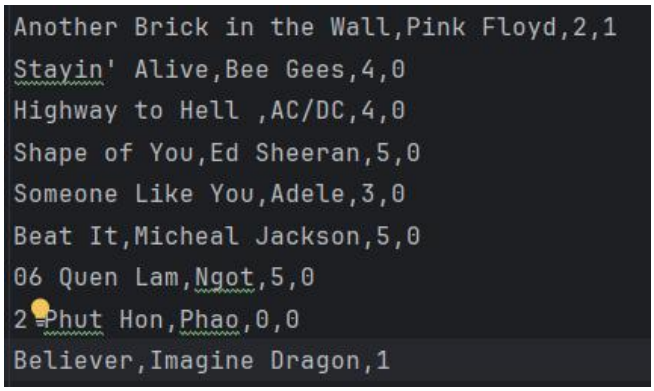


Image 34

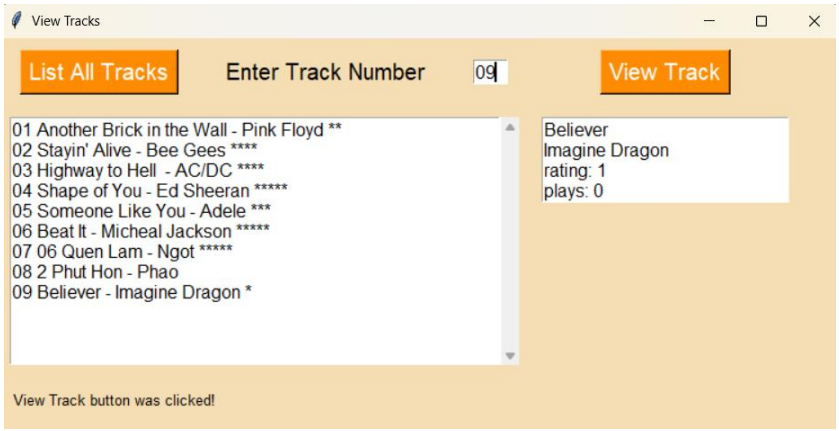


Image 35

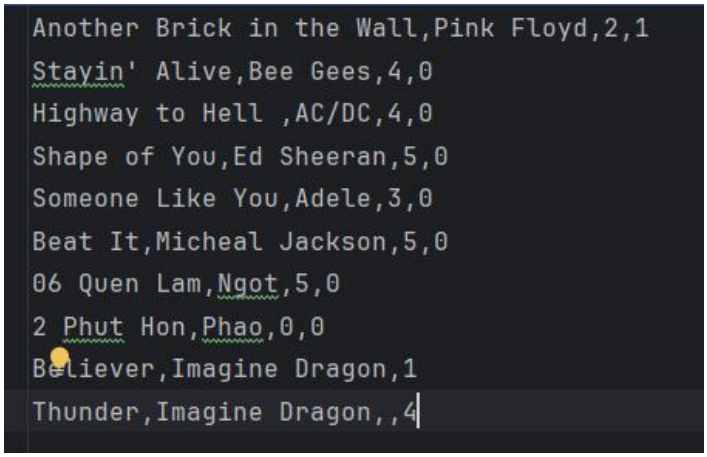
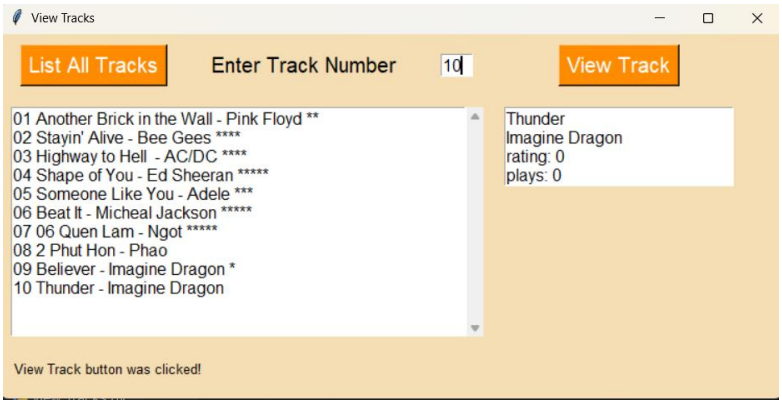


Image 36


View Tracks
List All Tracks
Enter Track Number
10
View Track
01 Another Brick in the Wall - Pink Floyd **
02 Stayin' Alive - Bee Gees ****
03 Highway to Hell - AC/DC ****
04 Shape of You - Ed Sheeran *****
05 Someone Like You - Adele ***
06 Beat It - Micheal Jackson *****
07 06 Quen Lam - Ngot *****
08 2 Phut Hon - Phao
09 Believer - Imagine Dragon *
10 Thunder - Imagine Dragon
Thunder
Imagine Dragon
rating: 0
plays: 0
View Track button was clicked!
Image 37