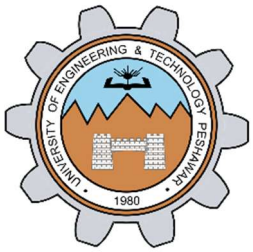
PROJECT REPORT &

OPEN-ENDED LAB REPORT



SPRING 2025

CSE-210L DATA STRUCTURES AND ALGORITHMS LAB

Submitted by:

Mian Muhammad Haziq (23PWCSE2235)

Other group members:

Muhammad Haseeb (23PWCSE2229)

Ahmed (23PWCSE2232)

Class Section: **B**

Submitted to:

Engr. Abdullah Hamid

July 4th, 2025

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Project title: Music Player

Project objectives:

The code implements a **graphical music player** using CSFML (C binding for SFML). Its main objectives are:

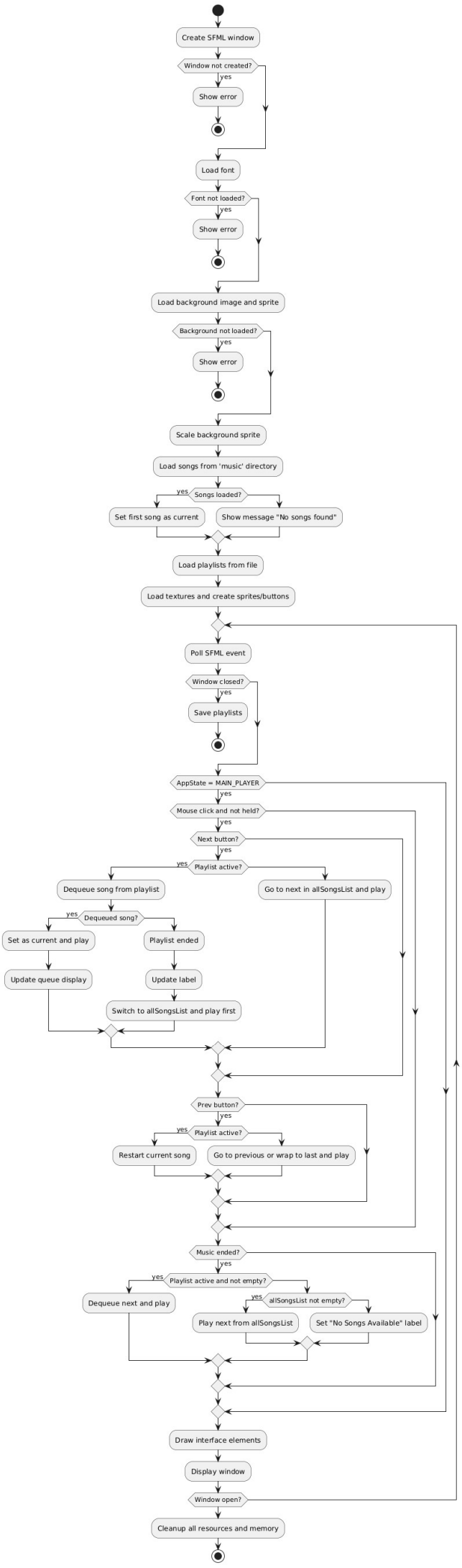
Core Functionalities:

1. **Play, Pause, Next, Previous:**
    - Plays `.ogg` music files.
    - Allows pausing, resuming, skipping to next, or restarting previous track.
2. **Dynamic Song Loading:**
    - Loads songs from the `music/` directory at startup.
3. **Playlist Management:**
    - User can create and select playlists from the GUI.
    - Stores playlists in a file and loads them at startup.
4. **UI Features:**
    - Custom background.
    - Button sprites for controls and playlists.
    - Labels for song names, recent songs, and current playlist.
5. **Recent History:**
    - Displays recently played songs using a stack structure.
6. **Memory Management:**
    - Frees all dynamically allocated memory on exit.

Open-ended lab objectives:

To demonstrate robust and efficient data management within the program, strategically **integrate appropriate data structure concepts** in your code. This integration should clearly illustrate how different data structures facilitate the effective **organization, storage, retrieval, and manipulation of data** for the specific problem at hand.

Project flowchart:



Project code:

```c
#include <SFML/Graphics.h>
#include <SFML/Audio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h> // For bool type

// Required for Windows API directory scanning
#include <windows.h>

// Define maximum songs that can be displayed for selection and max playlist name length
#define MAX_SELECTABLE_SONGS 100 // Adjust as needed
#define MAX_PLAYLIST_NAME_LENGTH 50
#define MAX_VISIBLE_LIST_ITEMS 10 // How many songs/playlists to show at once in lists
#define MAX_PATH_LENGTH 260 // Standard max path length on Windows (MAX_PATH is defined in windows.h)
#define PLAYLISTS_FILE "playlists.txt" // Name of the file to save/load playlists

// ------------------------- Song Structure -------------------------
typedef struct Song {
    char name[100]; // Stores just the song title
    char path[MAX_PATH_LENGTH]; // Stores the full path to the .ogg file
    struct Song* next;
    struct Song* prev;
} Song;

// ------------------------- Linked List (for Songs) -------------------------
void addSong(Song** list, const char* name, const char* path) {
    Song* temp = (Song*)malloc(sizeof(Song));
    if (!temp) {
        fprintf(stderr, "Memory allocation failed for Song.\n");
        return;
    }
    strncpy(temp->name, name, sizeof(temp->name) - 1);
    temp->name[sizeof(temp->name) - 1] = '\0'; // Ensure null-termination

    strncpy(temp->path, path, sizeof(temp->path) - 1);
    temp->path[sizeof(temp->path) - 1] = '\0'; // Ensure null-termination

    temp->next = NULL;
    temp->prev = NULL;

    if (*list == NULL) {
        *list = temp;
    } else {
        Song* last = *list;
        while (last->next != NULL)
            last = last->next;

        last->next = temp;
        temp->prev = last;
    }
}

// ------------------------- Recent Played Stack -------------------------
typedef struct StackNode {
    Song* song;
    struct StackNode* next;
} StackNode;

StackNode* recentStack = NULL;
sfText* recentText[5];

void pushRecent(Song* song) {
    // Prevent duplicates in recent stack if the last song is the same
    if (recentStack && recentStack->song == song) {
        return;
    }
    // Limit stack size (e.g., to 5)
    int count = 0;
    StackNode* temp = recentStack;
    while(temp) {
        count++;
        temp = temp->next;
    }
    if (count >= 5) {
        // Pop the oldest if stack is full
        StackNode* current = recentStack;
```

```
        StackNode* prev = NULL;
        while(current && current->next) {
            prev = current;
            current = current->next;
        }
        if (prev) {
            prev->next = NULL;
            free(current);
        } else { // Stack has only one element
            free(recentStack);
            recentStack = NULL;
        }
    }

    StackNode* node = (StackNode*)malloc(sizeof(StackNode));
    if (!node) {
        fprintf(stderr, "Memory allocation failed for StackNode.\n");
        return;
    }
    node->song = song;
    node->next = recentStack;
    recentStack = node;
}

Song* popRecent() {
    if (!recentStack) return NULL;
    StackNode* temp = recentStack;
    recentStack = recentStack->next;
    Song* song = temp->song;
    free(temp);
    return song;
}

// ------------------------- Globals (for main player) -------------------------
sfMusic* music = NULL;
Song* current = NULL;
Song* allSongsList = NULL; // Global list of all available songs

// Forward declaration for Playlist structure
typedef struct Playlist Playlist;
Playlist* currentPlaylist = NULL; // Currently active playlist

// Global textures and sprites for buttons (accessible by playNewSong)
sfTexture* playTexture = NULL;
sfTexture* pauseTexture = NULL;
sfSprite* globalPlaySprite = NULL;
sfText* globalSongLabel = NULL; // Global reference to the song display label
sfFont* globalFont = NULL; // Global reference to the font

// ------------------------- Playlist (Queue) -------------------------
typedef struct PlaylistNode {
    Song* song;
    struct PlaylistNode* next;
} PlaylistNode;

struct Playlist { // Definition for Playlist
    char name[100];
    PlaylistNode* front;
    PlaylistNode* rear;
    struct Playlist* next; // For linking multiple playlists (globally)
};

Playlist* playlists = NULL; // Global list of all playlists

Playlist* createPlaylist(const char* name) {
    Playlist* newPlaylist = (Playlist*)malloc(sizeof(Playlist));
    if (!newPlaylist) {
        fprintf(stderr, "Memory allocation failed for Playlist.\n");
        return NULL;
    }
    strncpy(newPlaylist->name, name, sizeof(newPlaylist->name) - 1);
    newPlaylist->name[sizeof(newPlaylist->name) - 1] = '\0'; // Ensure null-termination
    newPlaylist->front = newPlaylist->rear = NULL;
    newPlaylist->next = playlists; // Add to the global list of playlists (prepends)
    playlists = newPlaylist;
    return newPlaylist;
}

```

```c
// Function to create a playlist without adding to global 'playlists' list
// Used when creating a *copy* to play, to avoid modifying the master playlist
Playlist* createTemporaryPlaylist(const char* name) {
    Playlist* newPlaylist = (Playlist*)malloc(sizeof(Playlist));
    if (!newPlaylist) {
        fprintf(stderr, "Memory allocation failed for Playlist.\n");
        return NULL;
    }
    strncpy(newPlaylist->name, name, sizeof(newPlaylist->name) - 1);
    newPlaylist->name[sizeof(newPlaylist->name) - 1] = '\0';
    newPlaylist->front = newPlaylist->rear = NULL;
    newPlaylist->next = NULL; // IMPORTANT: Does not add to global 'playlists' list
    return newPlaylist;
}


void enqueueSong(Playlist* pl, Song* song) {
    if (!pl || !song) return; // Defensive check
    PlaylistNode* node = (PlaylistNode*)malloc(sizeof(PlaylistNode));
    if (!node) {
        fprintf(stderr, "Memory allocation failed for PlaylistNode.\n");
        return;
    }
    node->song = song;
    node->next = NULL;

    if (!pl->rear) {
        pl->front = pl->rear = node;
    } else {
        pl->rear->next = node;
        pl->rear = node;
    }
}

Song* dequeueSong(Playlist* pl) {
    if (!pl || !pl->front) return NULL;
    PlaylistNode* temp = pl->front;
    Song* song = temp->song;
    pl->front = pl->front->next;
    if (!pl->front) pl->rear = NULL;
    free(temp);
    return song;
}

sfText* queueText[5]; // Playlist queue UI (main screen)

void refreshQueueDisplay(sfFont* font, Playlist* pl) {
    PlaylistNode* temp = pl ? pl->front : NULL;
    for (int i = 0; i < 5; i++) {
        if (queueText[i]) {
            if (temp) {
                sfText_setString(queueText[i], temp->song->name);
                temp = temp->next;
            } else {
                sfText_setString(queueText[i], "");
            }
        }
    }
}

// ------------------------- App State Management -------------------------
typedef enum AppState {
    MAIN_PLAYER,
    CREATE_PLAYLIST_SCREEN,
    SELECT_PLAYLIST_SCREEN
} AppState;

// Global variable for current application state
AppState currentAppState = MAIN_PLAYER;

// ------------------------- Label Helper -------------------------
// Helper to create and return an sfText object
sfText* createLabel(sfFont* font, const char* text, float x, float y, int size) {
    sfText* label = sfText_create();
    if (!label) {
        fprintf(stderr, "Failed to create sfText.\n");
        return NULL;
    }
```

```
    sfText_setFont(label, font);
    sfText_setString(label, text);
    sfText_setCharacterSize(label, size);
    sfText_setPosition(label, (sfVector2f){x, y});
    sfText_setFillColor(label, sfWhite);
    return label;
}

// -------------------------- Display Recent -------------------------
void refreshRecentDisplay(sfFont* font) {
    StackNode* temp = recentStack;
    for (int i = 0; i < 5; i++) {
        if (recentText[i]) {
            if (temp) {
                sfText_setString(recentText[i], temp->song->name);
                temp = temp->next;
            } else {
                sfText_setString(recentText[i], "");
            }
        }
    }
}

// -------------------------- Music Control -------------------------
// Function to play a new song, uses global sprites/textures for consistency
void playNewSong(sfText* songLabel, sfFont* font, sfSprite* playSprite, sfTexture* pauseTex, sfTexture* playTex) {
    if (!current) {
        if (songLabel) sfText_setString(songLabel, "No Song Selected");
        if (playSprite && playTex) sfSprite_setTexture(playSprite, playTex, sfTrue);
        return;
    }

    if (music) {
        sfMusic_stop(music);
        sfMusic_destroy(music);
        music = NULL;
    }

    music = sfMusic_createFromFile(current->path);
    if (!music) {
        printf("Failed to load: %s\n", current->path);
        if (songLabel) sfText_setString(songLabel, "Error loading song!");
        if (playSprite && playTex) sfSprite_setTexture(playSprite, playTex, sfTrue);
        return;
    }

    sfMusic_play(music);
    if (songLabel) sfText_setString(songLabel, current->name);
    if (playSprite && pauseTex) sfSprite_setTexture(playSprite, pauseTex, sfTrue); // Set to pause icon when playing
    pushRecent(current);
    if (font) refreshRecentDisplay(font);
}

// -------------------------- Create Playlist Screen Variables & Functions -------------------------

// Input for playlist name
char createPlNameInput[MAX_PLAYLIST_NAME_LENGTH + 1] = "";
sfText* createPlNameLabel = NULL;
sfText* createPlNameTextInput = NULL;
sfRectangleShape* createPlNameInputRect = NULL; // New: Rectangle for input field

// UI elements for the Create Playlist Screen (static to persist across calls)
static sfText* createPlTitle_s = NULL;
static sfText* createPlSongsHeading_s = NULL;
static sfText* createPlCreateBtn_s = NULL;
static sfText* createPlCancelBtn_s = NULL;

int songSelected[MAX_SELECTABLE_SONGS]; // 0 for unselected, 1 for selected
sfText* selectableSongTexts[MAX_SELECTABLE_SONGS]; // Text objects for each song name

// New: Mouse click debounce flag
static bool mouseWasPressed = false; // For debouncing clicks in general for UI screens

// Function to handle the Create Playlist screen
// Returns true when the screen is finished (create or cancel)
bool handleCreatePlaylistScreen(sfRenderWindow* window, sfEvent* event, sfFont* font, Song* allSongs, Playlist** allPlaylists) {
    static bool uiInitialized = false;
    static sfClock* inputClock = NULL; // For input de-bouncing
```

```
312
313     // Initialize/Reset UI elements when entering the screen
314     if (!uiInitialized) {
315         // Clear previous selections and input
316         for (int i = 0; i < MAX_SELECTABLE_SONGS; i++) {
317             songSelected[i] = 0;
318             if (selectableSongTexts[i]) {
319                 sfText_destroy(selectableSongTexts[i]);
320                 selectableSongTexts[i] = NULL;
321             }
322         }
323         strcpy(createPlNameInput, ""); // Clear name input
324
325         // Destroy previous UI elements if re-initializing to prevent memory leaks
326         if (createPlTitle_s) sfText_destroy(createPlTitle_s);
327         if (createPlNameLabel) sfText_destroy(createPlNameLabel);
328         if (createPlNameTextInput) sfText_destroy(createPlNameTextInput);
329         if (createPlNameInputRect) sfRectangleShape_destroy(createPlNameInputRect); // Destroy the rectangle
330         if (createPlSongsHeading_s) sfText_destroy(createPlSongsHeading_s);
331         if (createPlCreateBtn_s) sfText_destroy(createPlCreateBtn_s);
332         if (createPlCancelBtn_s) sfText_destroy(createPlCancelBtn_s);
333
334         createPlTitle_s = createLabel(font, "Create New Playlist", 250, 20, 30);
335         if (createPlTitle_s) sfText_setFillColor(createPlTitle_s, sfGreen);
336
337         createPlNameLabel = createLabel(font, "Playlist Name:", 50, 80, 20);
338         createPlNameTextInput = createLabel(font, "", 200, 80, 20);
339         if (createPlNameTextInput) sfText_setFillColor(createPlNameTextInput, sfYellow);
340
341         // New: Create rectangle for input field
342         createPlNameInputRect = sfRectangleShape_create();
343         if (createPlNameInputRect) {
344             sfRectangleShape_setSize(createPlNameInputRect, (sfVector2f){300, 30}); // Adjust size as needed
345             sfRectangleShape_setPosition(createPlNameInputRect, (sfVector2f){195, 75}); // Position slightly behind text
346             sfRectangleShape_setFillColor(createPlNameInputRect, sfColor_fromRGBA(50, 50, 50, 150)); // Semi-transparent dark grey
347             sfRectangleShape_setOutlineThickness(createPlNameInputRect, 1);
348             sfRectangleShape_setOutlineColor(createPlNameInputRect, sfWhite);
349         }
350
351         createPlSongsHeading_s = createLabel(font, "Available Songs:", 50, 150, 20);
352
353         // Populate selectable songs
354         Song* currentSongPtr = allSongs;
355         int i = 0;
356         float songY = 180;
357         while (currentSongPtr != NULL && i < MAX_SELECTABLE_SONGS) {
358             selectableSongTexts[i] = createLabel(font, currentSongPtr->name, 70, songY, 18);
359             if (selectableSongTexts[i]) sfText_setFillColor(selectableSongTexts[i], sfWhite); // Default color
360             songY += 25;
361             currentSongPtr = currentSongPtr->next;
362             i++;
363         }
364
365         createPlCreateBtn_s = createLabel(font, "CREATE", 200, 450, 24);
366         if (createPlCreateBtn_s) sfText_setFillColor(createPlCreateBtn_s, sfGreen);
367         createPlCancelBtn_s = createLabel(font, "CANCEL", 400, 450, 24);
368         if (createPlCancelBtn_s) sfText_setFillColor(createPlCancelBtn_s, sfRed);
369
370         inputClock = sfClock_create();
371         uiInitialized = true;
372         mouseWasPressed = false; // Reset mouse state for this screen
373     }
374
375     // --- Event Handling for Create Playlist Screen ---
376     if (event->type == sfEvtTextEntered) {
377         if (sfClock_getElapsedTime(inputClock).microseconds > 150000) {
378             if (event->text.unicode < 128) {
379                 if (event->text.unicode == '\b') {
380                     if (strlen(createPlNameInput) > 0) {
381                         createPlNameInput[strlen(createPlNameInput) - 1] = '\0';
382                     }
383                 } else if (event->text.unicode == '\n' || event->text.unicode == '\r') {
384                     // Ignore Enter key here, as we have a button
385                 } else if (strlen(createPlNameInput) < MAX_PLAYLIST_NAME_LENGTH) {
386                     char c = (char)event->text.unicode;
387                     strncat(createPlNameInput, &c, 1);
388                 }
389                 if (createPlNameTextInput) sfText_setString(createPlNameTextInput, createPlNameInput);
```

```
            }
            sfClock_restart(inputClock);
        }
    } else if (event->type == sfEvtMouseButtonPressed) {
        // Only process click if mouse wasn't pressed in previous frame to avoid multiple triggers
        if (!mouseWasPressed) {
            sfVector2i mouse = sfMouse_getPositionRenderWindow(window);

            // Click on song names to select/deselect
            Song* currentSongPtr = allSongs;
            for (int i = 0; i < MAX_SELECTABLE_SONGS && currentSongPtr != NULL; i++) {
                if (selectableSongTexts[i]) {
                    sfFloatRect textBounds = sfText_getGlobalBounds(selectableSongTexts[i]);
                    if (sfFloatRect_contains(&textBounds, (float)mouse.x, (float)mouse.y)) {
                        songSelected[i] = !songSelected[i]; // Toggle selection
                        if (songSelected[i]) {
                            sfText_setFillColor(selectableSongTexts[i], sfCyan); // Highlight selected
                        } else {
                            sfText_setFillColor(selectableSongTexts[i], sfWhite); // Back to normal
                        }
                        mouseWasPressed = true; // Mark as processed
                        break;
                    }
                }
                currentSongPtr = currentSongPtr->next;
            }

            // Click on CREATE button
            if (createPlCreateBtn_s) {
                sfFloatRect createBtnBounds = sfText_getGlobalBounds(createPlCreateBtn_s);
                if (sfFloatRect_contains(&createBtnBounds, (float)mouse.x, (float)mouse.y)) {
                    if (strlen(createPlNameInput) > 0) {
                        Playlist* newPl = createPlaylist(createPlNameInput);
                        if (newPl) {
                            Song* songToAddToPl = allSongs;
                            for (int i = 0; i < MAX_SELECTABLE_SONGS && songToAddToPl != NULL; i++) {
                                if (songSelected[i]) {
                                    enqueueSong(newPl, songToAddToPl);
                                }
                                songToAddToPl = songToAddToPl->next;
                            }
                            printf("Playlist '%s' created with selected songs.\n", createPlNameInput);
                        }
                    } else {
                        printf("Please enter a playlist name.\n");
                        if (createPlNameTextInput) sfText_setFillColor(createPlNameTextInput, sfRed);
                    }
                    uiInitialized = false;
                    if (inputClock) { sfClock_destroy(inputClock); inputClock = NULL; }
                    currentAppState = MAIN_PLAYER;
                    mouseWasPressed = true; // Mark as processed
                    return true;
                }
            }

            // Click on CANCEL button
            if (createPlCancelBtn_s) {
                sfFloatRect cancelBtnBounds = sfText_getGlobalBounds(createPlCancelBtn_s);
                if (sfFloatRect_contains(&cancelBtnBounds, (float)mouse.x, (float)mouse.y)) {
                    printf("Playlist creation canceled.\n");
                    uiInitialized = false;
                    if (inputClock) { sfClock_destroy(inputClock); inputClock = NULL; }
                    currentAppState = MAIN_PLAYER;
                    mouseWasPressed = true; // Mark as processed
                    return true;
                }
            }
        }
    } else if (event->type == sfEvtMouseButtonReleased) {
        mouseWasPressed = false; // Reset flag when mouse button is released
    }

    // --- Drawing for Create Playlist Screen ---
    sfRenderWindow_drawText(window, createPlTitle_s, NULL);
    sfRenderWindow_drawText(window, createPlNameLabel, NULL);
    if (createPlNameInputRect) sfRenderWindow_drawRectangleShape(window, createPlNameInputRect, NULL); // Draw the rectangle first
    sfRenderWindow_drawText(window, createPlNameTextInput, NULL); // Then draw the text on top
    sfRenderWindow_drawText(window, createPlSongsHeading_s, NULL);
```

```
    for (int i = 0; i < MAX_SELECTABLE_SONGS; i++) {
        if (selectableSongTexts[i]) {
            sfRenderWindow_drawText(window, selectableSongTexts[i], NULL);
        }
    }

    sfRenderWindow_drawText(window, createPlCreateBtn_s, NULL);
    sfRenderWindow_drawText(window, createPlCancelBtn_s, NULL);

    return false; // Screen is not finished yet
}

// -------------------------- SELECT PLAYLIST SCREEN --------------------------
// Declared static variables for UI elements for proper scope and cleanup handling
static sfText* selectPlTitle_s = NULL;
static sfText* playSelectedBtn_s = NULL;
static sfText* cancelSelectBtn_s = NULL;
static sfText* playlistText_s[MAX_VISIBLE_LIST_ITEMS]; // To display available playlists
static sfRectangleShape* playlistRect_s[MAX_VISIBLE_LIST_ITEMS]; // New: Rectangles for playlist names
static Playlist* selectedPlaylist_s = NULL; // To store the currently selected playlist
static int playlistSelectedIndex_s = -1; // Index of the selected playlist


bool handleSelectPlaylistScreen(sfRenderWindow* window, sfEvent* event, sfFont* font, Playlist* allPlaylists) {
    static bool uiInitialized = false;

    if (!uiInitialized) {
        // Cleanup existing elements if re-entering
        if (selectPlTitle_s) { sfText_destroy(selectPlTitle_s); selectPlTitle_s = NULL; }
        if (playSelectedBtn_s) { sfText_destroy(playSelectedBtn_s); playSelectedBtn_s = NULL; }
        if (cancelSelectBtn_s) { sfText_destroy(cancelSelectBtn_s); cancelSelectBtn_s = NULL; }
        for(int i = 0; i < MAX_VISIBLE_LIST_ITEMS; ++i) {
            if (playlistText_s[i]) { sfText_destroy(playlistText_s[i]); playlistText_s[i] = NULL; }
            if (playlistRect_s[i]) { sfRectangleShape_destroy(playlistRect_s[i]); playlistRect_s[i] = NULL; } // Destroy rectangles
        }

        selectPlTitle_s = createLabel(font, "Select Playlist to Play", 200, 20, 30);
        if(selectPlTitle_s) sfText_setFillColor(selectPlTitle_s, sfWhite);

        // Populate playlist names and create their rectangles
        Playlist* tempPl = allPlaylists;
        float yPos = 60;
        int i = 0;
        while(tempPl && i < MAX_VISIBLE_LIST_ITEMS) {
            playlistText_s[i] = createLabel(font, tempPl->name, 50, yPos, 20);
            if(playlistText_s[i]) {
                sfText_setFillColor(playlistText_s[i], sfWhite); // Default color

                // Create rectangle for playlist name
                playlistRect_s[i] = sfRectangleShape_create();
                if (playlistRect_s[i]) {
                    sfFloatRect textBounds = sfText_getGlobalBounds(playlistText_s[i]);
                    // Add some padding to the rectangle
                    sfRectangleShape_setSize(playlistRect_s[i], (sfVector2f){textBounds.width + 20, textBounds.height + 10});
                    sfRectangleShape_setPosition(playlistRect_s[i], (sfVector2f){textBounds.left - 10, textBounds.top - 5});
                    sfRectangleShape_setFillColor(playlistRect_s[i], sfColor_fromRGBA(0, 0, 0, 100)); // Semi-transparent black
                    sfRectangleShape_setOutlineThickness(playlistRect_s[i], 1);
                    sfRectangleShape_setOutlineColor(playlistRect_s[i], sfWhite);
                }
            }
            tempPl = tempPl->next;
            yPos += 25;
            i++;
        }

        playSelectedBtn_s = createLabel(font, "PLAY SELECTED", 200, 450, 24);
        if(playSelectedBtn_s) sfText_setFillColor(playSelectedBtn_s, sfGreen);
        cancelSelectBtn_s = createLabel(font, "CANCEL", 450, 450, 24);
        if(cancelSelectBtn_s) sfText_setFillColor(cancelSelectBtn_s, sfRed);

        selectedPlaylist_s = NULL;
        playlistSelectedIndex_s = -1;
        uiInitialized = true;
        mouseWasPressed = false; // Reset mouse state for this screen
    }

    if (event->type == sfEvtMouseButtonPressed) {
```

```
        if (!mouseWasPressed) { // Only process click if mouse wasn't pressed in previous frame
            sfVector2i mouse = sfMouse_getPositionRenderWindow(window);

            // Check for clicks on playlist names
            Playlist* currentListPtr = allPlaylists; // Use a fresh pointer for iteration
            for (int i = 0; i < MAX_VISIBLE_LIST_ITEMS; ++i) {
                if (!playlistText_s[i]) break;

                // Use rectangle bounds for click detection for better hit area
                sfFloatRect plRectBounds = sfRectangleShape_getGlobalBounds(playlistRect_s[i]);
                if (sfFloatRect_contains(&plRectBounds, (float)mouse.x, (float)mouse.y)) {
                    // Deselect previous visual
                    if (playlistSelectedIndex_s != -1 && playlistRect_s[playlistSelectedIndex_s]) {
                        sfRectangleShape_setOutlineColor(playlistRect_s[playlistSelectedIndex_s], sfWhite);
                        sfRectangleShape_setFillColor(playlistRect_s[playlistSelectedIndex_s], sfColor_fromRGBA(0,0,0,100));
                    }

                    // Select new visual
                    playlistSelectedIndex_s = i;
                    sfRectangleShape_setOutlineColor(playlistRect_s[i], sfCyan); // Highlight selected outline
                    sfRectangleShape_setFillColor(playlistRect_s[i], sfColor_fromRGBA(0, 100, 100, 150)); // Darker cyan fill

                    // Find the actual playlist pointer based on its position in the list
                    selectedPlaylist_s = currentListPtr;
                    for(int j = 0; j < i; ++j) {
                        if (selectedPlaylist_s) selectedPlaylist_s = selectedPlaylist_s->next;
                    }
                    mouseWasPressed = true; // Mark as processed
                    break;
                }
                if (currentListPtr) currentListPtr = currentListPtr->next;
            }

            // Click on PLAY SELECTED button
            if (playSelectedBtn_s) {
                sfFloatRect playSelectedBtnBounds = sfText_getGlobalBounds(playSelectedBtn_s);
                if (sfFloatRect_contains(&playSelectedBtnBounds, (float)mouse.x, (float)mouse.y)) {
                    if (selectedPlaylist_s && selectedPlaylist_s->front) {
                        // Cleanup previous currentPlaylist if exists (to avoid memory leaks if switching playlists)
                        if (currentPlaylist) {
                            PlaylistNode* node = currentPlaylist->front;
                            while(node){
                                PlaylistNode* next = node->next;
                                free(node);
                                node = next;
                            }
                            free(currentPlaylist);
                            currentPlaylist = NULL;
                        }

                        // Create a *new temporary playlist* (copy) to be the current playing queue
                        currentPlaylist = createTemporaryPlaylist(selectedPlaylist_s->name);
                        PlaylistNode* tempNode = selectedPlaylist_s->front;
                        while(tempNode){
                            enqueueSong(currentPlaylist, tempNode->song);
                            tempNode = tempNode->next;
                        }

                        current = dequeueSong(currentPlaylist); // Get first song from the *copy*
                        if (current) {
                            playNewSong(globalSongLabel, globalFont, globalPlaySprite, pauseTexture, playTexture);
                        } else {
                            printf("Selected playlist is empty after copying.\n");
                            currentPlaylist = NULL; // No songs, clear playlist
                        }
                        refreshQueueDisplay(globalFont, currentPlaylist); // Refresh main queue display

                        uiInitialized = false;
                        currentAppState = MAIN_PLAYER;
                        mouseWasPressed = true; // Mark as processed
                        return true;
                    } else {
                        printf("No playlist selected or selected playlist is empty.\n");
                    }
                }
            }

            // Click on CANCEL button
```

```
            if (cancelSelectBtn_s) {
                sfFloatRect cancelBtnBounds = sfText_getGlobalBounds(cancelSelectBtn_s);
                if (sfFloatRect_contains(&cancelBtnBounds, (float)mouse.x, (float)mouse.y)) {
                    printf("Playlist selection canceled.\n");
                    uiInitialized = false;
                    currentAppState = MAIN_PLAYER;
                    mouseWasPressed = true; // Mark as processed
                    return true;
                }
            }
        }
    } else if (event->type == sfEvtMouseButtonReleased) {
        mouseWasPressed = false; // Reset flag when mouse button is released
    }


    sfRenderWindow_drawText(window, selectPlTitle_s, NULL);
    for(int i = 0; i < MAX_VISIBLE_LIST_ITEMS; ++i) {
        if(playlistRect_s[i]) sfRenderWindow_drawRectangleShape(window, playlistRect_s[i], NULL); // Draw rectangle first
        if(playlistText_s[i]) sfRenderWindow_drawText(window, playlistText_s[i], NULL); // Then draw text
    }
    sfRenderWindow_drawText(window, playSelectedBtn_s, NULL);
    sfRenderWindow_drawText(window, cancelSelectBtn_s, NULL);

    return false;
}

// -------------------------- Directory Scanning (Windows Specific) --------------------------
void loadSongsFromDirectory(Song** allSongsList, const char* directoryPath) {
    WIN32_FIND_DATAA findFileData;
    HANDLE hFind;
    char searchPath[MAX_PATH_LENGTH];
    char fullFilePath[MAX_PATH_LENGTH];

    snprintf(searchPath, sizeof(searchPath), "%s\\*.ogg", directoryPath);

    printf("Scanning directory: %s for .ogg files...\n", searchPath);

    hFind = FindFirstFileA(searchPath, &findFileData);
    if (hFind == INVALID_HANDLE_VALUE) {
        printf("Error opening directory or no .ogg files found: %s (Error Code: %lu)\n", directoryPath, GetLastError());
        return;
    }

    do {
        if (strcmp(findFileData.cFileName, ".") == 0 || strcmp(findFileData.cFileName, "..") == 0) {
            continue;
        }

        if (!(findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)) {
            snprintf(fullFilePath, sizeof(fullFilePath), "%s\\%s", directoryPath, findFileData.cFileName);
            addSong(allSongsList, findFileData.cFileName, fullFilePath);
            printf("Found and added: %s\n", fullFilePath);
        }
    } while (FindNextFileA(hFind, &findFileData) != 0);

    FindClose(hFind);
}

// -------------------------- Playlist Persistence --------------------------

// Function to save all playlists to a file
void savePlaylistsToFile(const char* filename, Playlist* allPlaylists) {
    FILE* fp = fopen(filename, "w");
    if (!fp) {
        fprintf(stderr, "Error: Could not open playlist file for writing: %s\n", filename);
        return;
    }

    Playlist* currentPl = allPlaylists;
    while (currentPl) {
        fprintf(fp, "#PLAYLIST_START:%s\n", currentPl->name);
        PlaylistNode* currentNode = currentPl->front;
        while (currentNode) {
            fprintf(fp, "%s\n", currentNode->song->path); // Save full path
            currentNode = currentNode->next;
        }
        fprintf(fp, "#PLAYLIST_END\n");
```

```
        currentPl = currentPl->next;
    }

    fclose(fp);
    printf("Playlists saved to %s\n", filename);
}

// Helper to find a Song* by its path from the allSongsList
Song* findSongByPath(const char* path) {
    Song* temp = allSongsList; // Assuming allSongsList is globally accessible
    while (temp) {
        if (strcmp(temp->path, path) == 0) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL; // Song not found
}

// Function to load playlists from a file
void loadPlaylistsFromFile(const char* filename, Playlist** allPlaylists) {
    FILE* fp = fopen(filename, "r");
    if (!fp) {
        printf("No existing playlist file found: %s. Starting fresh.\n", filename);
        return;
    }

    char line[MAX_PATH_LENGTH + 100]; // Buffer for reading lines
    Playlist* currentLoadingPlaylist = NULL;

    while (fgets(line, sizeof(line), fp)) {
        // Remove newline character if present
        line[strcspn(line, "\n")] = 0;

        if (strstr(line, "#PLAYLIST_START:") == line) { // Starts with #PLAYLIST_START:
            char playlistName[MAX_PLAYLIST_NAME_LENGTH + 1];
            sscanf(line, "#PLAYLIST_START:%[^\n]", playlistName); // Read name after prefix
            currentLoadingPlaylist = createPlaylist(playlistName); // Creates and adds to global 'playlists' list
            printf("Loading playlist: %s\n", playlistName);
        } else if (strcmp(line, "#PLAYLIST_END") == 0) {
            currentLoadingPlaylist = NULL; // End of current playlist
        } else {
            // This line is a song path
            if (currentLoadingPlaylist) {
                Song* foundSong = findSongByPath(line); // Find the Song* in your allSongsList
                if (foundSong) {
                    enqueueSong(currentLoadingPlaylist, foundSong);
                    printf("  Added song: %s\n", foundSong->name);
                } else {
                    printf("  Warning: Song path not found in library, skipping: %s\n", line);
                }
            }
        }
    }

    fclose(fp);
    printf("Playlists loaded from %s\n", filename);
}


// -------------------------- Main --------------------------
int main() {
    sfVideoMode mode = {800, 500, 32};
    sfRenderWindow* window = sfRenderWindow_create(mode, "Music Player", sfResize | sfClose, NULL);
    if (!window) {
        fprintf(stderr, "Failed to create SFML window.\n");
        return 1;
    }
    sfEvent event;

    globalFont = sfFont_createFromFile("Sansation_Bold.ttf"); // Assign to global font
    if (!globalFont) {
        fprintf(stderr, "Failed to load font: Sansation_Bold.ttf\n");
        return 1;
    }

    sfTexture* bgTexture = sfTexture_createFromFile("bg.png", NULL);
    if (!bgTexture) {
```

```
        fprintf(stderr, "Failed to load background image: bg.png\n");
        return 1;
    }
    sfSprite* bgSprite = sfSprite_create();
    if (!bgSprite) {
        fprintf(stderr, "Failed to create background sprite.\n");
        return 1;
    }
    sfSprite_setTexture(bgSprite, bgTexture, sfTrue);

    sfVector2u bgSize = sfTexture_getSize(bgTexture);
    sfVector2f scale;
    scale.x = (float)mode.width / bgSize.x;
    scale.y = (float)mode.height / bgSize.y;
    sfSprite_setScale(bgSprite, scale);

    // --- Load songs dynamically from 'music' directory ---
    char musicDirectory[] = "music";
    allSongsList = NULL;
    loadSongsFromDirectory(&allSongsList, musicDirectory);

    if (allSongsList == NULL) {
        printf("No .ogg songs found in the '%s' directory. Please add some music files.\n", musicDirectory);
    } else {
        current = allSongsList; // Set initial song for main player to the first found song
    }

    // --- Load Playlists from file AFTER songs are loaded ---
    loadPlaylistsFromFile(PLAYLISTS_FILE, &playlists);


    // ---------- Music Control Sprites ----------
    playTexture = sfTexture_createFromFile("play.png", NULL);
    pauseTexture = sfTexture_createFromFile("pause.png", NULL);
    sfTexture* nextTexture = sfTexture_createFromFile("next.png", NULL);
    sfTexture* prevTexture = sfTexture_createFromFile("prev.jpg", NULL);

    if (!playTexture || !pauseTexture || !nextTexture || !prevTexture) {
        fprintf(stderr, "Failed to load control button images.\n");
        return 1;
    }

    globalPlaySprite = sfSprite_create(); // Assign to global variable
    sfSprite* nextSprite = sfSprite_create();
    sfSprite* prevSprite = sfSprite_create();
    if (!globalPlaySprite || !nextSprite || !prevSprite) {
        fprintf(stderr, "Failed to create control sprites.\n");
        return 1;
    }

    sfSprite_setTexture(globalPlaySprite, playTexture, sfTrue);
    sfSprite_setTexture(nextSprite, nextTexture, sfTrue);
    sfSprite_setTexture(prevSprite, prevTexture, sfTrue);

    // Scaling for control buttons
    float control_scale = 0.15f;
    sfSprite_setScale(globalPlaySprite, (sfVector2f){control_scale, control_scale});
    sfSprite_setScale(nextSprite, (sfVector2f){control_scale, control_scale});
    sfSprite_setScale(prevSprite, (sfVector2f){control_scale, control_scale});

    // Positioning for control buttons (more central)
    sfVector2f playPos = {mode.width / 2.0f - sfSprite_getGlobalBounds(globalPlaySprite).width / 2.0f, 350};
    sfVector2f prevPos = {playPos.x - sfSprite_getGlobalBounds(prevSprite).width - 20, 350};
    sfVector2f nextPos = {playPos.x + sfSprite_getGlobalBounds(globalPlaySprite).width + 20, 350};

    sfSprite_setPosition(globalPlaySprite, playPos);
    sfSprite_setPosition(nextSprite, nextPos);
    sfSprite_setPosition(prevSprite, prevPos);

    // ---------- Playlist Button Sprites ----------
    sfTexture* createPlaylistTexture = sfTexture_createFromFile("create_playlist.png", NULL);
    sfTexture* playPlaylistTexture = sfTexture_createFromFile("play_playlist.png", NULL);

    if (!createPlaylistTexture || !playPlaylistTexture) {
        fprintf(stderr, "Failed to load playlist button images.\n");
        return 1;
    }

```

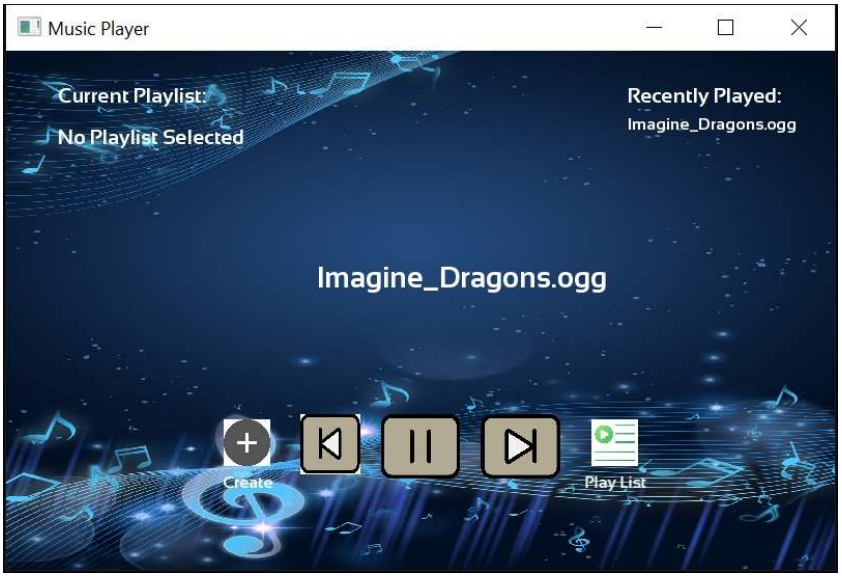
```
    sfSprite* createPlaylistSprite = sfSprite_create();
    sfSprite* playPlaylistSprite = sfSprite_create();
    if (!createPlaylistSprite || !playPlaylistSprite) {
        fprintf(stderr, "Failed to create playlist sprites.\n");
        return 1;
    }

    sfSprite_setTexture(createPlaylistSprite, createPlaylistTexture, sfTrue);
    sfSprite_setTexture(playPlaylistSprite, playPlaylistTexture, sfTrue);

    // Scaling for playlist buttons
    float playlist_btn_scale = 0.20f;
    sfSprite_setScale(createPlaylistSprite, (sfVector2f){playlist_btn_scale, playlist_btn_scale});
    sfSprite_setScale(playPlaylistSprite, (sfVector2f){playlist_btn_scale, playlist_btn_scale});

    // Positioning for playlist buttons
    sfSprite_setPosition(createPlaylistSprite, (sfVector2f){210, 355});
    sfSprite_setPosition(playPlaylistSprite, (sfVector2f){565, 355});

    // ---------- Labels (Main Player UI) ----------
    globalSongLabel = createLabel(globalFont, "No Song Playing", 300, 200, 28); // Assign to global
    sfText *recentHeading = createLabel(globalFont, "Recently Played:", 600, 30, 20);
    sfText *queueHeading = createLabel(globalFont, "Current Playlist:", 50, 30, 20);
    sfText *playlistNameLabel = createLabel(globalFont, "No Playlist Selected", 50, 70, 20);

    // Text labels *below* the new playlist sprites
    sfText* tempText = createLabel(globalFont, "Create", 0,0,16);
    float createLabelWidth = tempText ? sfText_getLocalBounds(tempText).width : 0;
    if (tempText) sfText_destroy(tempText);
    sfText *createPlaylistLabel = createLabel(globalFont, "Create", sfSprite_getPosition(createPlaylistSprite).x + (sfSprite_getGlobalBounds(createPlaylistSp

    tempText = createLabel(globalFont, "Play List", 0,0,16);
    float playLabelWidth = tempText ? sfText_getLocalBounds(tempText).width : 0;
    if (tempText) sfText_destroy(tempText);
    sfText *playPlaylistLabel = createLabel(globalFont, "Play List", sfSprite_getPosition(playPlaylistSprite).x + (sfSprite_getGlobalBounds(playPlaylistSprit


    // Initialize recent and queue display texts
    for (int i = 0; i < 5; i++) {
        recentText[i] = createLabel(globalFont, "", 600, 60 + i * 25, 16);
        queueText[i] = createLabel(globalFont, "", 400, 60 + i * 25, 16);
    }

    // Main application loop
    while (sfRenderWindow_isOpen(window)) {
        while (sfRenderWindow_pollEvent(window, &event)) {
            if (event.type == sfEvtClosed) {
                // Save playlists before closing!
                savePlaylistsToFile(PLAYLISTS_FILE, playlists);
                sfRenderWindow_close(window);
            }

            // Handle events based on current application state
            if (currentAppState == MAIN_PLAYER) {
                // Mouse event handling for main player only checks if the mouse button was just pressed
                if (event.type == sfEvtMouseButtonPressed && !mouseWasPressed) {
                    sfVector2i mouse = sfMouse_getPositionRenderWindow(window);

                    sfFloatRect playBounds = sfSprite_getGlobalBounds(globalPlaySprite);
                    sfFloatRect nextBounds = sfSprite_getGlobalBounds(nextSprite);
                    sfFloatRect prevBounds = sfSprite_getGlobalBounds(prevSprite);
                    sfFloatRect createPlaylistBtnBounds = sfSprite_getGlobalBounds(createPlaylistSprite);
                    sfFloatRect playPlaylistBtnBounds = sfSprite_getGlobalBounds(playPlaylistSprite);

                    // --- Play Button Logic ---
                    if (sfFloatRect_contains(&playBounds, (float)mouse.x, (float)mouse.y)) {
                        if (!music || sfMusic_getStatus(music) == sfStopped) {
                            playNewSong(globalSongLabel, globalFont, globalPlaySprite, pauseTexture, playTexture);
                        } else {
                            sfSoundStatus status = sfMusic_getStatus(music);
                            if (status == sfPlaying) {
                                sfMusic_pause(music);
                                sfSprite_setTexture(globalPlaySprite, playTexture, sfTrue); // Set to play icon when paused
                            } else {
                                sfMusic_play(music);
                                sfSprite_setTexture(globalPlaySprite, pauseTexture, sfTrue); // Set to pause icon when playing
                            }
                        }
```

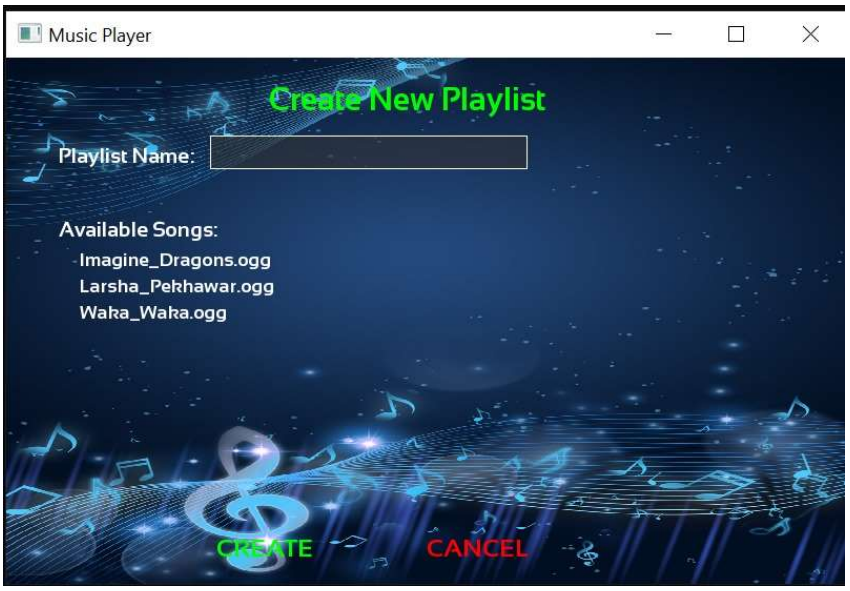
```
                    mouseWasPressed = true;
                }

                // --- Next / Prev (Playlist-aware) ---
                if (sfFloatRect_contains(&nextBounds, (float)mouse.x, (float)mouse.y)) {
                    if (currentPlaylist) { // If a playlist is active
                        current = dequeueSong(currentPlaylist);
                        if (current) {
                            playNewSong(globalSongLabel, globalFont, globalPlaySprite, pauseTexture, playTexture);
                            refreshQueueDisplay(globalFont, currentPlaylist); // Update queue display
                        } else {
                            printf("Playlist ended. Switching to main song list.\n");
                            sfText_setString(globalSongLabel, "Playlist Ended");
                            if (globalPlaySprite && playTexture) sfSprite_setTexture(globalPlaySprite, playTexture, sfTrue);
                            currentPlaylist = NULL; // Playlist finished
                            refreshQueueDisplay(globalFont, currentPlaylist); // Clear queue display
                            // Optionally, auto-play first song from allSongsList here
                            if (allSongsList) {
                                current = allSongsList;
                                playNewSong(globalSongLabel, globalFont, globalPlaySprite, pauseTexture, playTexture);
                            }
                        }
                    } else if (current) { // No playlist, cycle through all songs
                        current = current->next ? current->next : allSongsList;
                        playNewSong(globalSongLabel, globalFont, globalPlaySprite, pauseTexture, playTexture);
                    }
                    mouseWasPressed = true;
                }

                if (sfFloatRect_contains(&prevBounds, (float)mouse.x, (float)mouse.y)) {
                    if (currentPlaylist) {
                        // Simplified for queue: just restart current song if within a playlist
                        // A full 'previous' in a queue requires re-enqueueing or a different list structure.
                        printf("Restarting current song in playlist (Prev button).\n");
                        if (current) { // Only restart if there's a song playing
                            sfMusic_stop(music);
                            sfMusic_play(music);
                        }
                    } else if (current) { // No playlist, cycle through all songs
                         if (current->prev) {
                             current = current->prev;
                         } else {
                             Song* tempLast = allSongsList;
                             while(tempLast && tempLast->next) {
                                 tempLast = tempLast->next;
                             }
                             current = tempLast ? tempLast : allSongsList;
                         }
                         playNewSong(globalSongLabel, globalFont, globalPlaySprite, pauseTexture, playTexture);
                     }
                     mouseWasPressed = true;
                }

                // --- Playlist Buttons (Transition to other screens) ---
                if (sfFloatRect_contains(&createPlaylistBtnBounds, (float)mouse.x, (float)mouse.y)) {
                    currentAppState = CREATE_PLAYLIST_SCREEN;
                    mouseWasPressed = true;
                }

                if (sfFloatRect_contains(&playPlaylistBtnBounds, (float)mouse.x, (float)mouse.y)) {
                    currentAppState = SELECT_PLAYLIST_SCREEN;
                    mouseWasPressed = true;
                }
            }
            else if (event.type == sfEvtMouseButtonReleased) {
                mouseWasPressed = false; // Reset flag when mouse button is released
            }
        } else if (currentAppState == CREATE_PLAYLIST_SCREEN) {
            handleCreatePlaylistScreen(window, &event, globalFont, allSongsList, &playlists);
        } else if (currentAppState == SELECT_PLAYLIST_SCREEN) {
            handleSelectPlaylistScreen(window, &event, globalFont, playlists);
        }
    }

    // --- Drawing based on current application state ---
    sfRenderWindow_clear(window, sfBlack);
    sfRenderWindow_drawSprite(window, bgSprite, NULL);

```

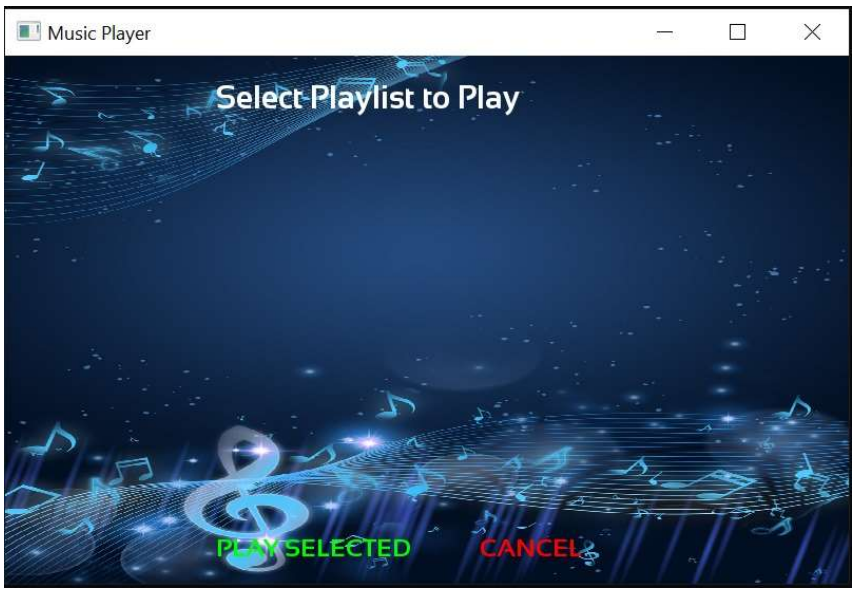
```
        if (currentAppState == MAIN_PLAYER) {
            // Auto-play next song if current one stops AND there's an active playlist
            if (music && sfMusic_getStatus(music) == sfStopped) {
                if (currentPlaylist && currentPlaylist->front) {
                    current = dequeueSong(currentPlaylist);
                    if (current) {
                        playNewSong(globalSongLabel, globalFont, globalPlaySprite, pauseTexture, playTexture);
                    } else {
                        sfText_setString(globalSongLabel, "Playlist Ended");
                        if (globalPlaySprite && playTexture) sfSprite_setTexture(globalPlaySprite, playTexture, sfTrue);

                        // Clean up the temporary currentPlaylist when it's empty
                        PlaylistNode* node = currentPlaylist->front;
                        while(node){
                            PlaylistNode* next = node->next;
                            free(node);
                            node = next;
                        }
                        free(currentPlaylist);
                        currentPlaylist = NULL;

                        refreshQueueDisplay(globalFont, currentPlaylist); // Clear queue display
                    }
                } else { // Fallback to main song list auto-play if no playlist or playlist is empty (and current song ended)
                    if (allSongsList && current) { // Ensure current is not NULL before trying to find next
                        current = current->next ? current->next : allSongsList; // Cycle back to start of all songs
                        playNewSong(globalSongLabel, globalFont, globalPlaySprite, pauseTexture, playTexture);
                    } else {
                        sfText_setString(globalSongLabel, "No Songs Available");
                        if (globalPlaySprite && playTexture) sfSprite_setTexture(globalPlaySprite, playTexture, sfTrue);
                    }
                }
            }

            // Draw all main player UI elements
            sfRenderWindow_drawSprite(window, globalPlaySprite, NULL);
            sfRenderWindow_drawSprite(window, nextSprite, NULL);
            sfRenderWindow_drawSprite(window, prevSprite, NULL);
            sfRenderWindow_drawSprite(window, createPlaylistSprite, NULL);
              sfRenderWindow_drawSprite(window, playPlaylistSprite, NULL);

              sfRenderWindow_drawText(window, globalSongLabel, NULL);
              sfRenderWindow_drawText(window, recentHeading, NULL);
              sfRenderWindow_drawText(window, queueHeading, NULL);
              sfRenderWindow_drawText(window, createPlaylistLabel, NULL);
              sfRenderWindow_drawText(window, playPlaylistLabel, NULL);

              // Update and display current playlist name
              if (currentPlaylist) {
                  sfText_setString(playlistNameLabel, currentPlaylist->name);
              } else {
                  sfText_setString(playlistNameLabel, "No Playlist Selected");
              }
              sfRenderWindow_drawText(window, playlistNameLabel, NULL);


              for (int i = 0; i < 5; i++) {
                  sfRenderWindow_drawText(window, recentText[i], NULL);
                  sfRenderWindow_drawText(window, queueText[i], NULL);
              }

          } else if (currentAppState == CREATE_PLAYLIST_SCREEN) {
              handleCreatePlaylistScreen(window, &event, globalFont, allSongsList, &playlists);
          } else if (currentAppState == SELECT_PLAYLIST_SCREEN) {
              handleSelectPlaylistScreen(window, &event, globalFont, playlists);
          }

          sfRenderWindow_display(window);
      }

      // --- Cleanup ---
      if (music) { sfMusic_stop(music); sfMusic_destroy(music); }
      if (globalFont) sfFont_destroy(globalFont);
      if (bgSprite) sfSprite_destroy(bgSprite);
      if (bgTexture) sfTexture_destroy(bgTexture);

      if (globalPlaySprite) sfSprite_destroy(globalPlaySprite);
      if (nextSprite) sfSprite_destroy(nextSprite);
```

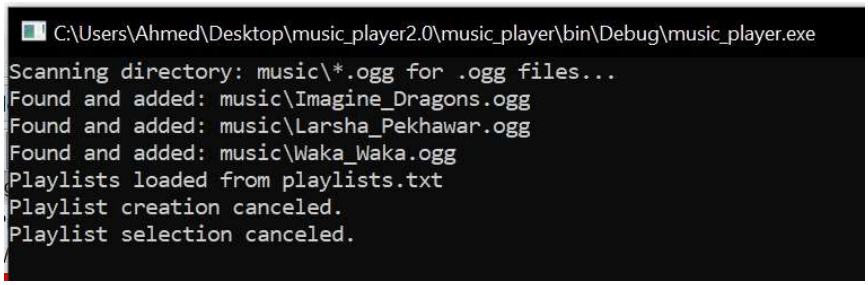
```
    if (prevSprite) sfSprite_destroy(prevSprite);
    if (playTexture) sfTexture_destroy(playTexture);
    if (pauseTexture) sfTexture_destroy(pauseTexture);
    if (nextTexture) sfTexture_destroy(nextTexture);
    if (prevTexture) sfTexture_destroy(prevTexture);

    if (createPlaylistSprite) sfSprite_destroy(createPlaylistSprite);
    if (playPlaylistSprite) sfSprite_destroy(playPlaylistSprite);
    if (createPlaylistTexture) sfTexture_destroy(createPlaylistTexture);
    if (playPlaylistTexture) sfTexture_destroy(playPlaylistTexture);

    // Main player UI texts
    if (globalSongLabel) sfText_destroy(globalSongLabel);
    if (recentHeading) sfText_destroy(recentHeading);
    if (queueHeading) sfText_destroy(queueHeading);
    if (createPlaylistLabel) sfText_destroy(createPlaylistLabel);
    if (playPlaylistLabel) sfText_destroy(playPlaylistLabel);
    if (playlistNameLabel) sfText_destroy(playlistNameLabel);


    for (int i = 0; i < 5; i++) {
        if (recentText[i]) sfText_destroy(recentText[i]);
        if (queueText[i]) sfText_destroy(queueText[i]);
    }

    // Cleanup for Create Playlist UI elements
    if (createPlTitle_s) sfText_destroy(createPlTitle_s);
    if (createPlNameLabel) sfText_destroy(createPlNameLabel);
    if (createPlNameTextInput) sfText_destroy(createPlNameTextInput);
    if (createPlNameInputRect) sfRectangleShape_destroy(createPlNameInputRect);
    if (createPlSongsHeading_s) sfText_destroy(createPlSongsHeading_s);
    if (createPlCreateBtn_s) sfText_destroy(createPlCreateBtn_s);
    if (createPlCancelBtn_s) sfText_destroy(createPlCancelBtn_s);
    for (int i = 0; i < MAX_SELECTABLE_SONGS; i++) {
        if (selectableSongTexts[i]) sfText_destroy(selectableSongTexts[i]);
    }

    // Cleanup for Select Playlist UI elements
    if (selectPlTitle_s) sfText_destroy(selectPlTitle_s);
    if (playSelectedBtn_s) sfText_destroy(playSelectedBtn_s);
    if (cancelSelectBtn_s) sfText_destroy(cancelSelectBtn_s);
    for(int i = 0; i < MAX_VISIBLE_LIST_ITEMS; ++i) {
        if (playlistText_s[i]) sfText_destroy(playlistText_s[i]);
        if (playlistRect_s[i]) sfRectangleShape_destroy(playlistRect_s[i]);
    }


    if (window) sfRenderWindow_destroy(window);

    // Clean up song linked list (allSongsList)
    Song* tempSong;
    while (allSongsList) {
        tempSong = allSongsList;
        allSongsList = allSongsList->next;
        free(tempSong);
    }

    // Clean up playlists and their nodes
    Playlist* currentPl = playlists;
    while (currentPl) {
        PlaylistNode* currentNode = currentPl->front;
        while (currentNode) {
            PlaylistNode* nextNode = currentNode->next;
            free(currentNode);
            currentNode = nextNode;
        }
        Playlist* nextPl = currentPl->next;
        free(currentPl);
        currentPl = nextPl;
    }

    // Clean up recent stack nodes
    StackNode* currentStackNode = recentStack;
    while (currentStackNode) {
        StackNode* nextStackNode = currentStackNode->next;
        free(currentStackNode);
        currentStackNode = nextStackNode;
    }

    return 0;
}
```

Project output:








C:\Users\Ahmed\Desktop\music_player2.0\music_player\bin\Debug\music_player.exe
Scanning directory: music*.ogg for .ogg files...
Found and added: music\Imagine_Dragons.ogg
Found and added: music\Larsha_Pekhawar.ogg
Found and added: music\Waka_Waka.ogg
Playlists loaded from playlists.txt
Playlist creation canceled.
Playlist selection canceled.