



HIT137 SOFTWARE NOW

Assignment 3

Semester 2, 2024

Casuarina Campus

Professor: Dr. Thuseethan Selvarajah

Group Name: CAS 068

Group members	Student ID
Sangay Jamtsho	S374529
Thi Thuy Trang Tran	S375041
Cristine Casindac Mansinares	S377798
Shrey Kumar	S381119

TABLE OF CONTENT

QUESTION 1:	3
1.1 IDEA	3
1.2 KEY COMPONENTS	3
1.3 MAIN FEATURES	7
1.4 HOW IT WORKS	7
QUESTION 2:	9
2. 1 IDEA:	9
2.2 GAME ENVIRONMENT SET-UP	9
2.3 GAME VARIABLE	10
2.4 GAME MECHANICS	10
2.5 GAME LOOP	11
2.6 GAME TERMINATION	11
2.7 MAIN FEATURES	12
2.8 CONCLUSION	12
QUESTION 3:	13
3.1 TEAMWORK GANTT CHART	13
3.2 GITHUB'S LINK	14

QUESTION 1:

Create a Tkinter application using the concepts of object-oriented programming, such as, multiple inheritance, multiple decorators, encapsulation, polymorphism, and method overriding, etc.

Wherever you use these concepts in code, explain them using # (How they are linked to code).

1.1 IDEA

Creating the basic music player application with usage of libraries including:

- **Tkinter:** for creating of graphical user interface (GUI)
- **FileDialog (from tkinter):**
- **MessageBox (from tkinter):** for showing pop-up notifications, warning or information messages to user
- **PIL (Pillow):** for handling and displaying image (buttons)
- **Pygame:** for managing music playback (loading, playing, controlling audio file)

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import pygame
```

1.2 KEY COMPONENTS

- **The Pygame.mixer.init :** function initializes the Pygame mixer module, enabling the program to handle audio playback such as loading and playing sound files.

```
# Initialize pygame mixer for music playback
pygame.mixer.init()
```

- **MediaData Class:** Hold and manage list of media file that provide functionality to add and retrieve files from the list. Encapsulation was implemented here. The MediaData class secures the media list with a private attribute, media list. It offers controlled methods to add and access media files, ensuring the list remains intact.

```
class MediaData:
    def __init__(self):
        self.__media_list = [] # Private attribute for storing media files

    # Add a media file to the list
    def add_media_to_data(self, media):
        self.__media_list.append(media)

    # Get all media files
    def get_all_media(self):
        return self.__media_list
```

- **MediaPlayerBase Class:** define the method play_media and stop_media. The overriding method was implemented here. In MediaPlayerApp, the methods play_media and stop_media override the abstract definitions in MediaPlayerBase, delivering specific behaviors for starting and halting media playback.

```
class MediaPlayerBase:
    def play_media(self):
        raise NotImplementedError("You need to implement this method!")

    def stop_media(self):
        raise NotImplementedError("You need to implement this method!")
```

- **MediaPlayerApp Class:** this is main class, responsible for graphical interface and media player functionality and interact with users via buttons, listbox and volume control. The MediaPlayerApp class uses multiple inheritance, deriving from both MediaPlayerBase and MediaData. This allows it to manage media data effectively while also handling.

```

class MediaPlayerApp(MediaPlayerBase, MediaData):
    def __init__(self, master):
        super().__init__()
        self.master = master
        self.master.title("Music App")
        self.master.geometry("600x420")

        # Flag to stop playback
        self.stop_flag = False

        # Current playing index
        self.current_index = None

        # Layout setup
        self.setup_layout()

    # Setting up the background image
    def set_background(self, image_path):
        try:
            # Open image and resize to fit the window
            self.bg_image = Image.open(image_path)
            self.bg_image = self.bg_image.resize((400, 400), Image.LANCZOS)
            self.bg_photo = ImageTk.PhotoImage(self.bg_image)

            # Create a label to hold the background image
            self.bg_label = tk.Label(self.master, image=self.bg_photo)
            self.bg_label.place(x=0, y=0, relwidth=1, relheight=1)
        except Exception as e:
            print(f"Error loading image: {e}")

    # Setting up the UI layout
    def setup_layout(self):
        # Left Frame - Media List
        self.left_frame = tk.Frame(self.master, bg="lightgray", width=200, padx=10, pady=10)
        self.left_frame.pack(side=tk.LEFT, fill=tk.Y)

        # Right Frame - Player Controls
        self.right_frame = tk.Frame(self.master, bg="white", padx=10, pady=10)
        self.right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        # Listbox for media files
        self.media_listbox = tk.Listbox(self.left_frame, selectmode=tk.SINGLE, width=30, height=20)
        self.media_listbox.pack()

        # Add media button
        self.add_media_button = tk.Button(self.left_frame, text="Add Media", command=self.add_media)
        self.add_media_button.pack(pady=10)

        # Load images for buttons
        self.play_img = ImageTk.PhotoImage(Image.open("./play.png").resize((40, 40), Image.LANCZOS))
        self.stop_img = ImageTk.PhotoImage(Image.open("./stop.png").resize((40, 40), Image.LANCZOS))
        self.previous_img = ImageTk.PhotoImage(Image.open("./previous.png").resize((40, 40), Image.LANCZOS))
        self.next_img = ImageTk.PhotoImage(Image.open("./next.png").resize((40, 40), Image.LANCZOS))

        # Control buttons with images
        self.play_button = tk.Button(self.right_frame, image=self.play_img, command=self.play_media)
        self.play_button.pack(pady=10)

        self.stop_button = tk.Button(self.right_frame, image=self.stop_img, command=self.stop_media)
        self.stop_button.pack(pady=10)

        self.previous_button = tk.Button(self.right_frame, image=self.previous_img, command=self.play_previous_media)
        self.previous_button.pack(pady=10)

        self.next_button = tk.Button(self.right_frame, image=self.next_img, command=self.play_next_media)
        self.next_button.pack(pady=10)

```

```

# Play the selected media based on index
def play_selected_media(self, index):
    all_media = self.get_all_media()
    if index is not None and 0 <= index < len(all_media):
        selected_media = all_media[index]
        self.media_info_label.config(text=f"Now Playing: {selected_media.split('/')[-1]}")
        print(f"Playing media: {selected_media}")

        # Highlight the current selection in the listbox
        self.media_listbox.select_clear(0, tk.END)
        self.media_listbox.select_set(index)

        # Load and play the selected media
        pygame.mixer.music.load(selected_media)
        pygame.mixer.music.set_volume(self.volume_slider.get()) # Set volume
        pygame.mixer.music.play()

# Method to play next media
def play_next_media(self):
    all_media = self.get_all_media()
    if self.current_index is not None and self.current_index + 1 < len(all_media):
        self.current_index += 1
        self.play_selected_media(self.current_index)
    else:
        messagebox.showinfo("End of Playlist", "This is the last track.")

# Method to play previous media
def play_previous_media(self):
    if self.current_index is not None and self.current_index > 0:
        self.current_index -= 1
        self.play_selected_media(self.current_index)
    else:
        messagebox.showinfo("Start of Playlist", "This is the first track.")

```

- Main functions to run the tkinter app **def main ()**: : creating the main window with tk.Tk(), starting the media player app, keeping the window open with root.mainloop(), and running the program only when the script is run directly with if __name__ == "__main__":.

```

# Main function to run the Tkinter app
def main():
    root = tk.Tk()
    app = MediaPlayerApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()

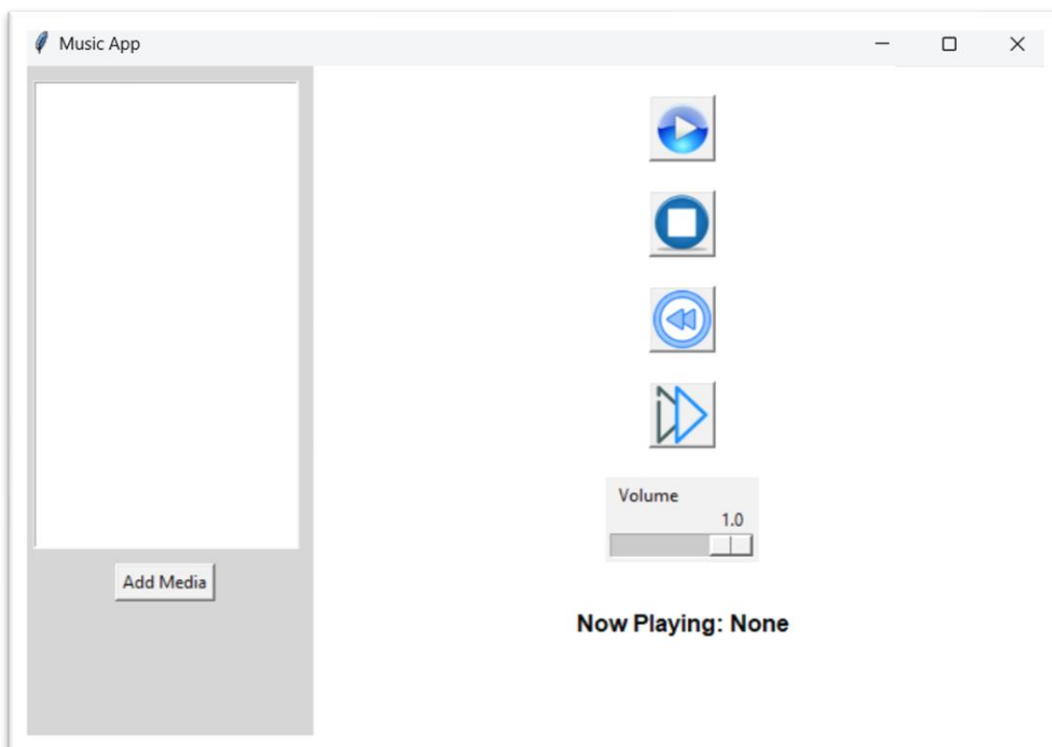
```

1.3 MAIN FEATURES

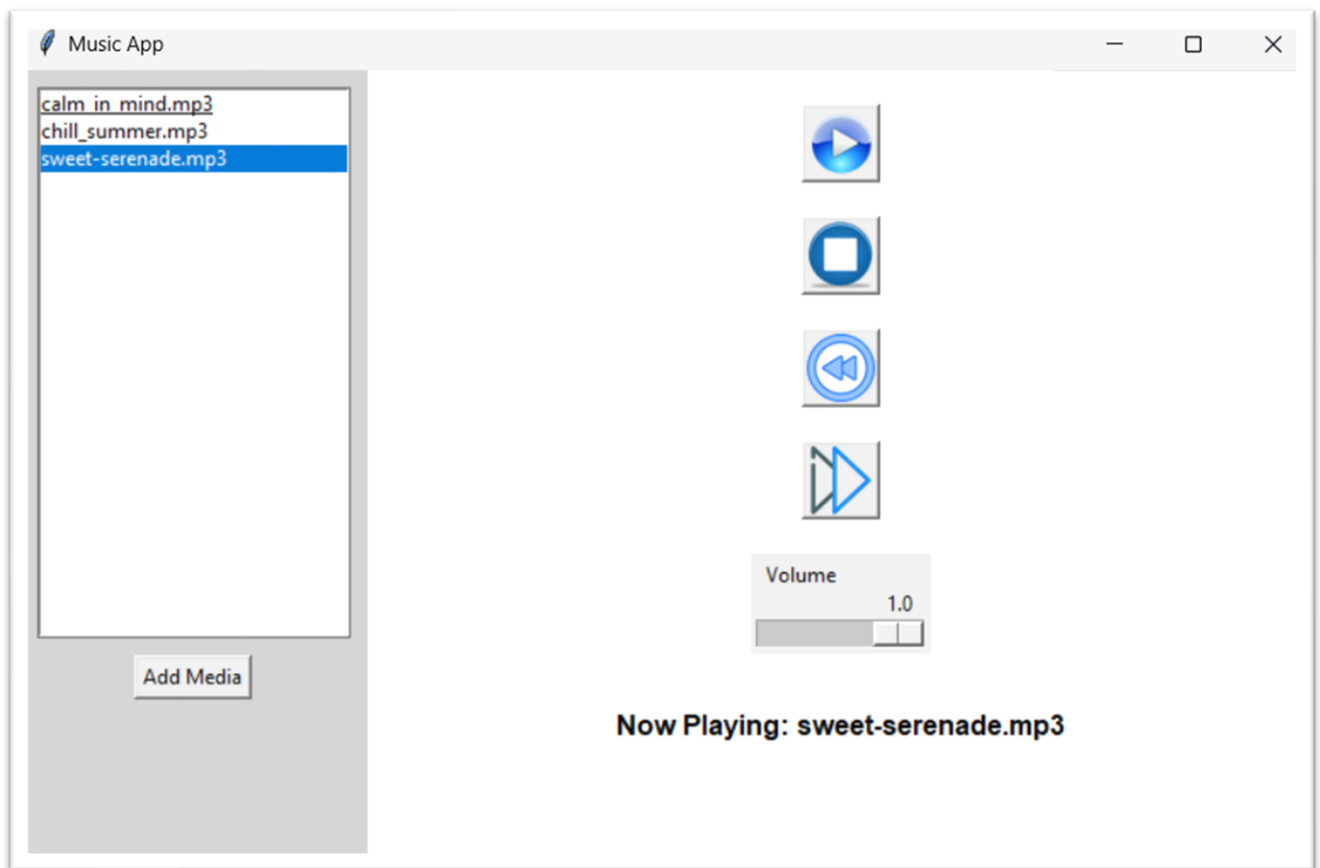
- **Media list:** The user can add and manage MP3 file or playback
- **Play, stop and navigation functions:** users can play, stop and move between media file easily
- **Volume control:** Users can control the volume during playback
- **Notifications:** MessageBox inform users when important actions occur or information messages.





1.4 HOW IT WORKS

- Load the mp3 file and image button into the same folder before running the python code.
- Run the python code and simple, user-friendly interface appear like below:



- **UI Layout:** The application is split into two main sections:
 - A left panel that shows a list of media files the user adds.
 - A right panel containing the playback controls (play, stop, next, previous), a volume controller and display the current track
- **Adding Media:** User can click the “Add Media” button to browse and select MP3 file and the file name will be listed on listbox for playback.



- **Playing Media:** After selecting a file from the list and clicking the “play” button , the application use “pygame” to play the chosen media and the file is displayed as well as playback begin.
- **Stopping Media:** When the users click “stop” button , the music stops and display “Now playing: None”
- **Navigating between Media:** User can use the “Next” button  or “Previous” button  to navigate between the file in the mp3 list.
- **Volume control:** The user can adjust the volume of the playback using the slider to update the volume.
- **MessageBox Notifications:** will show some important notifications:
 - Warning: when users try to play music without file
 - Information messages: when users reach the first or the last track in the playlist

QUESTION 2:**2.1 IDEA:**

"Zod Runner" is a 2D side-scroller game developed using the pygame library. The player navigates through multiple levels, avoiding enemies and collecting coins. The game includes an escalating difficulty level and ends when the player loses all lives or successfully completes all levels. Below is a detailed report of the game's structure, mechanics, and functionality based on the provided code.

2.2 GAME ENVIRONMENT SET-UP**• Library Imports:**

- ✓ **pygame**: Core library for handling game graphics, sounds, and event loops.
- ✓ **Configuration** (as **config**): A custom module that holds configuration values like screen width and height.
- ✓ Custom imports (**Enemy**, **Collectible**, **engine**, **context**): These handle enemy behaviors, collectible objects, game mechanics, and context-specific elements (e.g., images, surfaces).
- ✓ System imports (**exit**, **choice**, **randint**): Used for basic operations like exiting the game and randomizing in-game elements.

```
import pygame
import Configuration as config

import context
from Enemy import Enemy
from Collectible import Collectible

import engine
from sys import exit
from random import choice, randint
```

• Display Setup:

- ✓ A Pygame display is created with dimensions from the configuration file (**config.WIDTH**, **config.HEIGHT**).
- ✓ The game window is titled "Zod Runner."
- ✓ A **pygame.time.Clock()** object regulates the game's frame rate.

```
pygame.init()
screen = pygame.display.set_mode((config.WIDTH, config.HEIGHT))
pygame.display.set_caption("Zod Runner")
clock = pygame.time.Clock()
```

2.3 GAME VARIABLE

- **Game State:**
 - **game_over:** Boolean flag to track if the game is over.
 - **start_time:** Tracks the start of the game to compute scores.
 - **score, hit_score:** Score variables to track player performance.
 - **life:** The number of lives the player has (initially 2).
 - **current_level:** Tracks the player's progress through the levels (starts at 1).
 - **current_enemies:** Tracks the number of enemies spawned in each level.
 - **enemies_level:** A list defining the number of enemies per level.
 - **obstacle_timer:** A custom event that triggers the creation of obstacles and coins at regular intervals (800 ms).
-

2.4 GAME MECHANICS

Rendering Functions:

- **game_over_render(life=0):**
 - Displays the game-over screen when the player's lives reach 0.
 - Blits the background, game over text, and optionally a "player dead" image.
 - Calls the **engine.render_life()** function to display the player's remaining lives.
 - Shows the player's score and life count on screen.
- **game_render(start_time, life, hit_score):**
 - The primary game rendering function.
 - Draws and updates the player, obstacles, and collectible coins.
 - Checks for collisions:
 - **Obstacle collisions:** Reduces the player's life upon collision.
 - **Coin collisions:** Increases the player's score when coins are collected.
 - Updates the score based on time elapsed and coins collected.
 - Displays the score, remaining lives, and level on the screen.
 - Handles game-over logic when the player's life reaches 0.

- `render_levels(levels):`
 - Calls the `engine.render_levels()` function to display the current game level on screen.
-

Collision Detection:

- Obstacle Collision: When the player collides with an enemy (detected via `engine.obstacle_collision_sprite()`), they lose a life. If life drops to 0, the game ends.
 - Coin Collection: Collectibles (coins) are detected via `engine.coin_collision_sprite()`. Each collected coin increases the player's hit score, which is added to the total score.
-

2.5 GAME LOOP

The game operates within a while loop that continues as long as the current level is less than or equal to the maximum level (4 levels defined). Here's a breakdown of the loop:

- **Event Handling:**
 - The game listens for events like quitting the game, keypresses, and mouse clicks.
 - If the game is over, pressing the space bar or clicking the mouse restarts the game with default values (resetting lives, score, and levels).
 - **Obstacle and Coin Spawning:**
 - The custom `obstacle_timer` event triggers every 800 ms. When this event fires, the game spawns enemies or coins based on the current state.
 - Enemies spawn until a set limit is reached per level. After the level's enemy quota is filled, the game progresses to the next level.
 - Coins are randomly spawned in the game using a random number of types and positions.
 - **Rendering and Updates:**
 - If the game is not over, the `game_render()` function updates the game elements (player, obstacles, coins) and checks for collisions. The game screen is updated at a rate of 60 frames per second (`clock.tick(60)`).
-

2.6 GAME TERMINATION

- **Game Over State:**
 - When the player loses all lives (`life == 0`), the game-over screen is displayed, and the game halts.

- The game waits for player input to either restart the game or quit.
 - **Exit Conditions:**
 - The player can quit the game by closing the window or by reaching the last level, where the game ends after a brief delay.
-

2.7 MAIN FEATURES

- **Player Controls:**
 - The player can control the game using the keyboard or mouse to restart when the game ends.
 - **Obstacles and Collectibles:**
 - Enemies (obstacles) and coins (collectibles) spawn periodically based on random choices.
 - **Game Levels:**
 - The game includes 4 progressively harder levels, each with an increasing number of enemies.
 - **Score Tracking:**
 - The score is calculated based on both coin collection and time survived. The hit score (from collected coins) is combined with the elapsed time to create the total score.
-

2.8 CONCLUSION

"Zod Runner" provides a simple yet engaging side-scrolling experience with basic player mechanics, obstacle dodging, and score collection. The increasing difficulty with each level and life-based game-over system adds an element of challenge. The modular structure of the game (with separate **engine**, **Enemy**, and **Collectible** modules) suggests flexibility for future expansion, such as adding more levels, enemy types, or collectibles.

QUESTION 3:

Welcome to the final task of this assignment. You are required to create a GitHub repository and add all your group mates to it (make sure to keep it public, not private). You should do this before you start the assignment.

All the answers and contributions should be recorded in GitHub till you submit the assignment.

3.1 TEAMWORK GANTT CHART

GANTT CHART							
HIT137 SOFTWARE NOW							
Task No.	Task Name	Start date	End date	Meeting 1 26/09/2024	Meeting 2 06/10/2024	Meeting 3 13/10/2024	Final meeting 17/10/2025
1	Contribution for Assignment 3 and individual tasks	26/09/2024	06/10/2024	All members			
2	Details of Individual contribution	06/10/2024	13/10/2024		All members		
	Music app			Jane, Cristine	Jane, Cristine	Jane, Cristine	
	Pygame			Sangay, Shrey Kurma	Sangay, Shrey Kurma	Sangay, Shrey Kurma	
3	Reporting	13/10/2024	17/10/2024			All members	
4	Review	17/10/2024	18/10/2024				All members

Sharing information in teamwork on the Assignment 3

C

HIT137 SOFTWARE NOW

Chat

Shared

Meet now

4

W

X

P

Find Word, Excel, PowerPoint, and other files shared in this chat here!

Recent

Files

Links

Filter by keyword

Upload

<input type="checkbox"/>	Name	Date shared ↓	Shared by
	MusicApp 1.zip	10/7/2024	Cristine Casindac Mansinares
	pygame2.zip	10/7/2024	Sangay Jamtsho
	zod-runner.zip	10/7/2024	Shrey Kumar
	pygame.rar	10/6/2024	Thi Thuy Trang Tran
	pygame_sample_week10.zip	10/6/2024	Thi Thuy Trang Tran
	Image_Classifier.py	10/6/2024	Sangay Jamtsho
	test3.py	10/6/2024	Thi Thuy Trang Tran
	calm_in_mind.mp3	10/6/2024	Thi Thuy Trang Tran
	chill_summer.mp3	10/6/2024	Thi Thuy Trang Tran

3.2 GITHUB'S LINK

Link: [Janetran91bio/HIT137_Assignment3_Group_Cas068 \(github.com\)](https://github.com/Janetran91bio/HIT137_Assignment3_Group_Cas068)

The screenshot shows the GitHub interface for the repository `HIT137_Assignment3_Group_Cas068` by user `Janetran91bio`. The repository is public and has 14 commits. The main branch is selected. The file list shows:

File/Folder	Description	Commit Time
MusicApp	Question1_Tkinter Music App Implementation	2 days ago
pygame	Delete pygame/_MACOSX/COST	yesterday
.gitignore	Initial commit	3 days ago
README.md	Initial commit	3 days ago