WannaHang @ SIT

Project Presentation

Table of contents

01

Login

02

User Interface

03

Settings Page

04

Game Logic
(Hint)

05

Game Logic
(Health/Regen)

06

Game Logic
(Difficulty/Timer)

Table of contents

07

Game Logic (Validation)

08

Game Logic (Images)

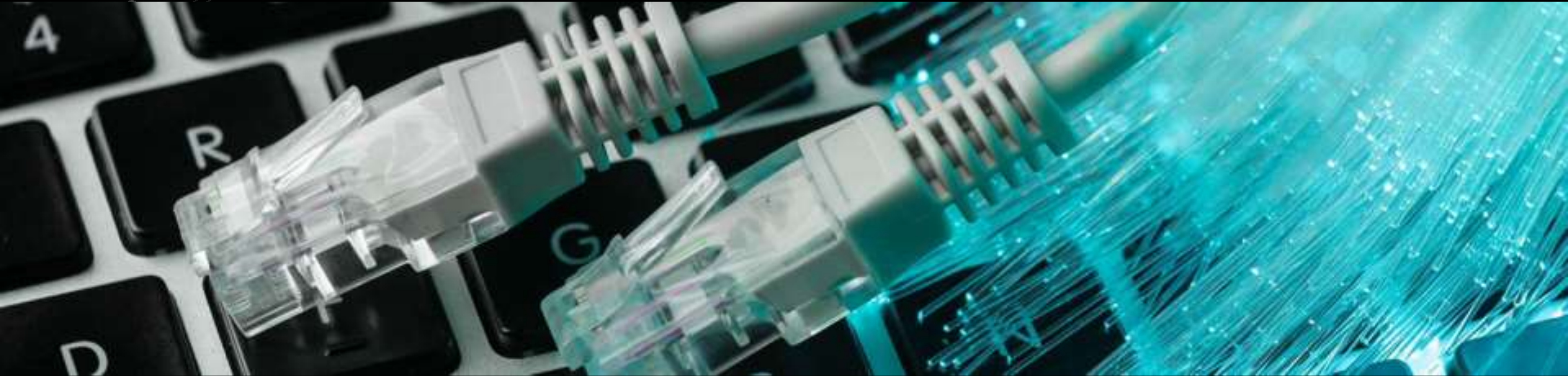
09

Music

10

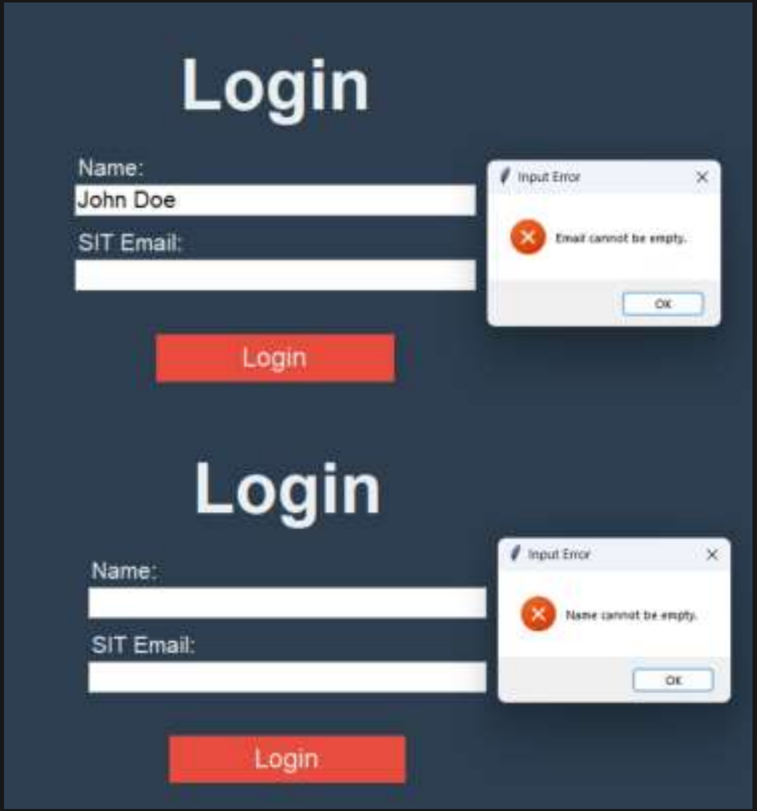
Game Logic (Completion)

01 Login



Login Validation

- If the user leaves either the name or email fields empty, an error will be shown, showing what they have done wrongly



Name: John Doe

SIT Email:

Input Error

Email cannot be empty.

Name:

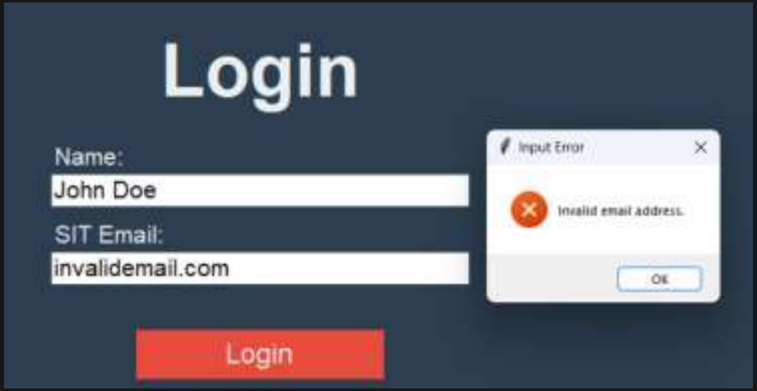
SIT Email:

Input Error

Name cannot be empty.

Login Validation

- If the email is not valid, error will be shown



Login

Name:

John Doe

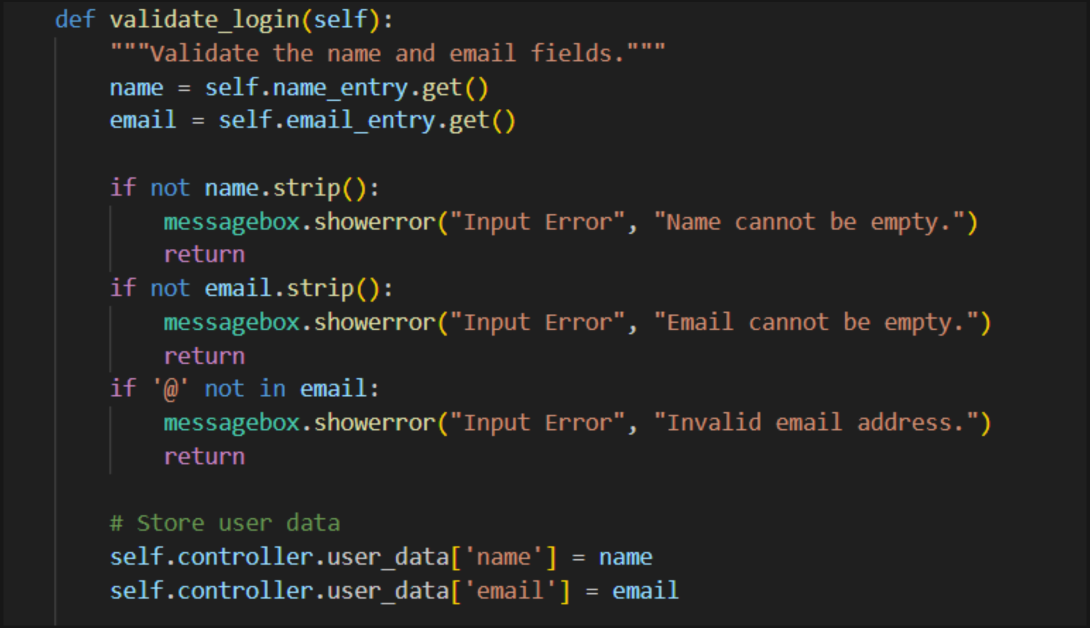
SIT Email:

invalidemail.com

Input Error

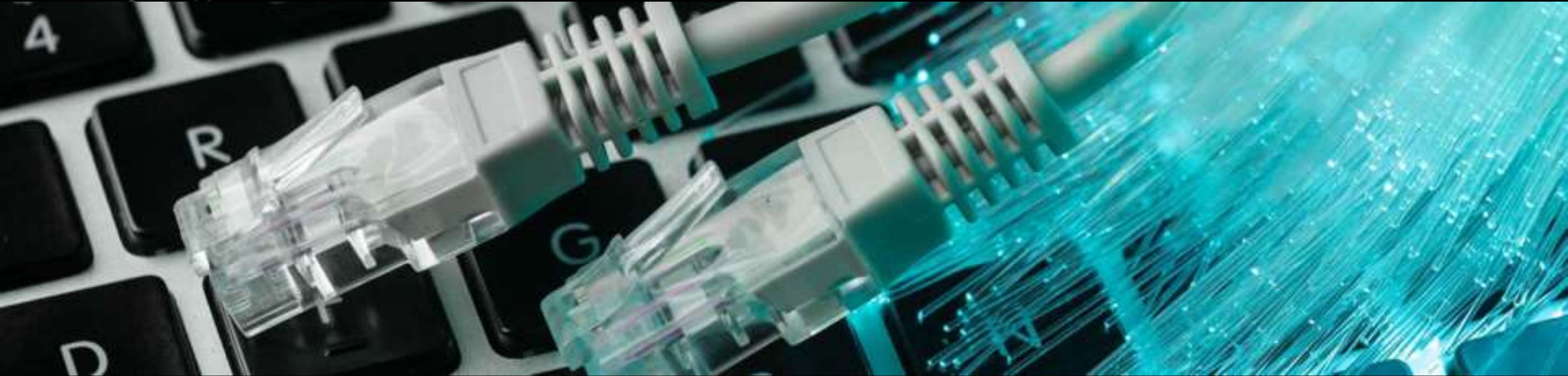
Invalid email address.

Login Code

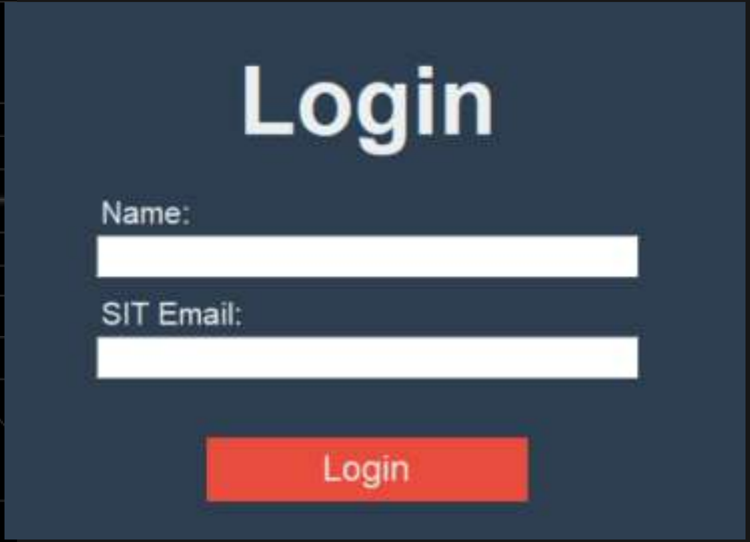


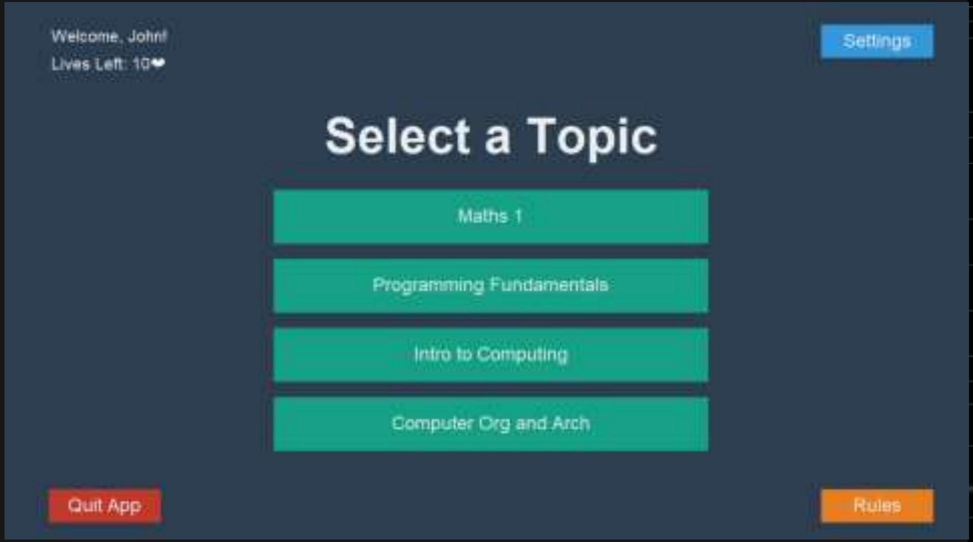
```python
def validate_login(self):
    """Validate the name and email fields."""
    name = self.name_entry.get()
    email = self.email_entry.get()

    if not name.strip():
        messagebox.showerror("Input Error", "Name cannot be empty.")
        return
    if not email.strip():
        messagebox.showerror("Input Error", "Email cannot be empty.")
        return
    if '@' not in email:
        messagebox.showerror("Input Error", "Invalid email address.")
        return

    # Store user data
    self.controller.user_data['name'] = name
    self.controller.user_data['email'] = email
```

02 User Interface



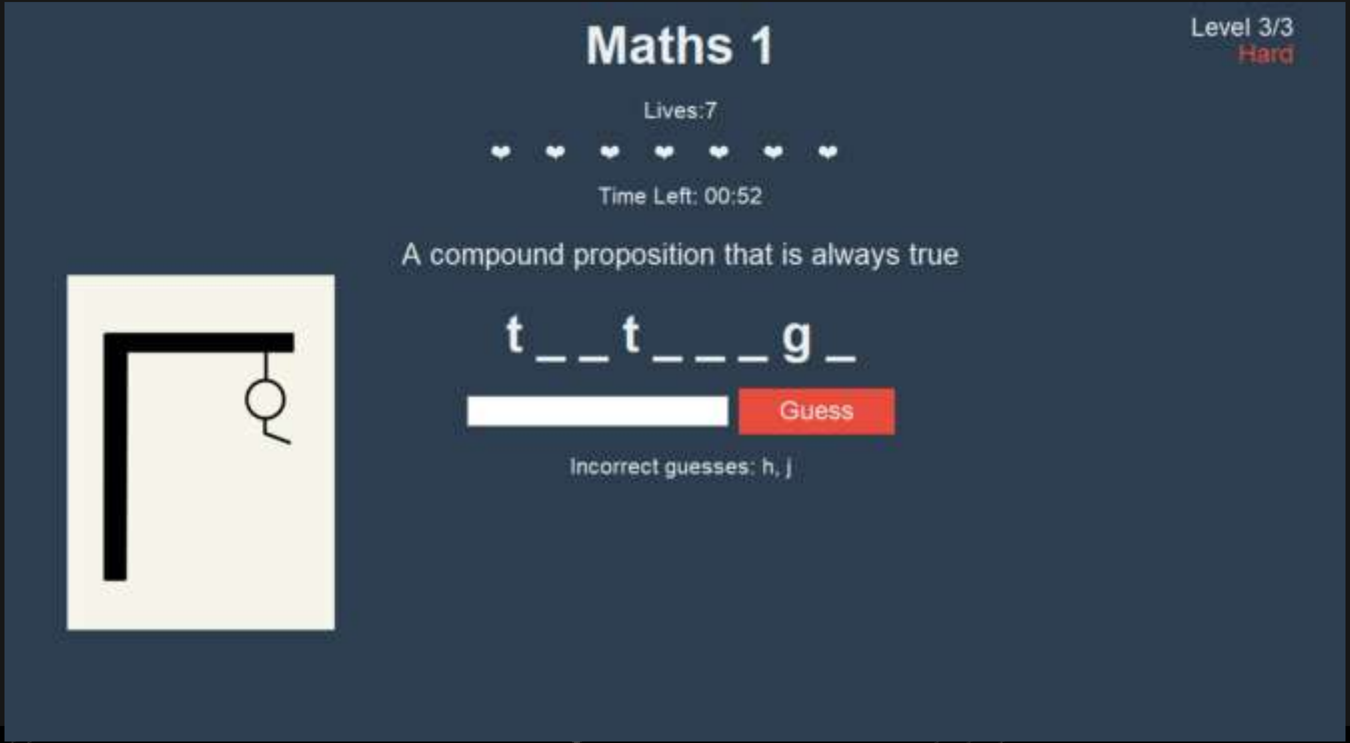
User Interface




Welcome, John!

Lives Left: 10❤

User Interface



Maths 1

Level 3/3
Hard

Lives:7

❤ ❤ ❤ ❤ ❤ ❤ ❤

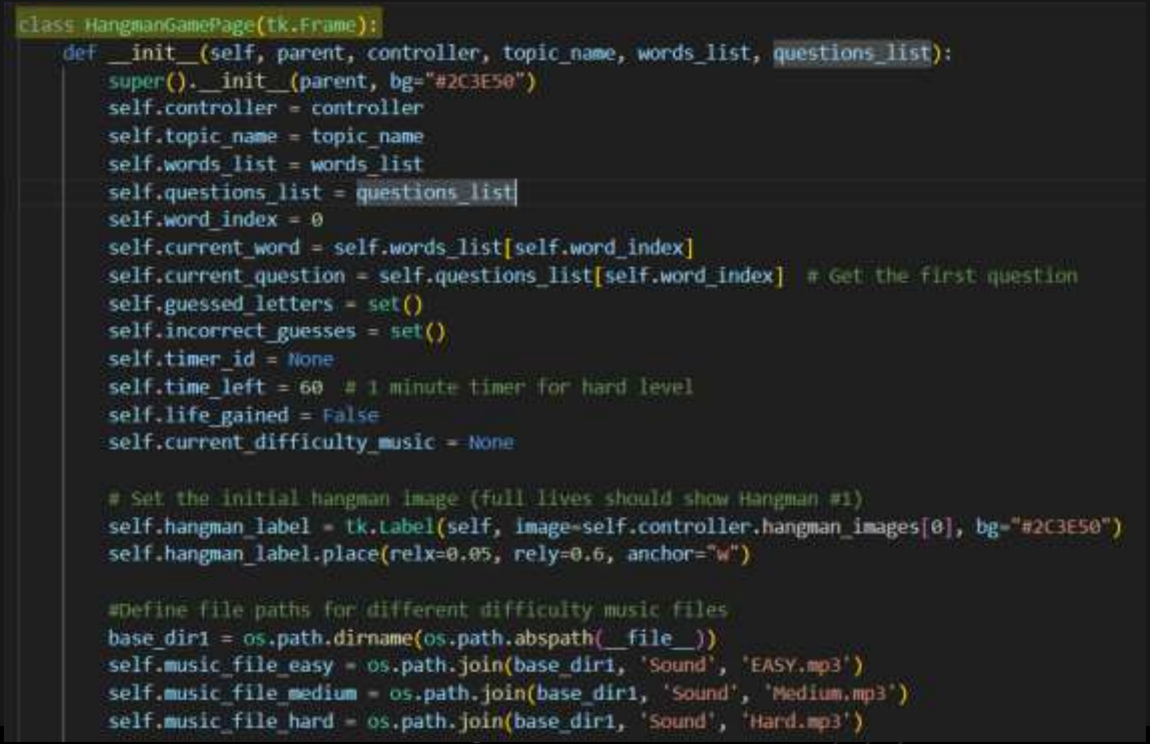
Time Left: 00:52

A compound proposition that is always true

t _ _ t _ _ _ g _

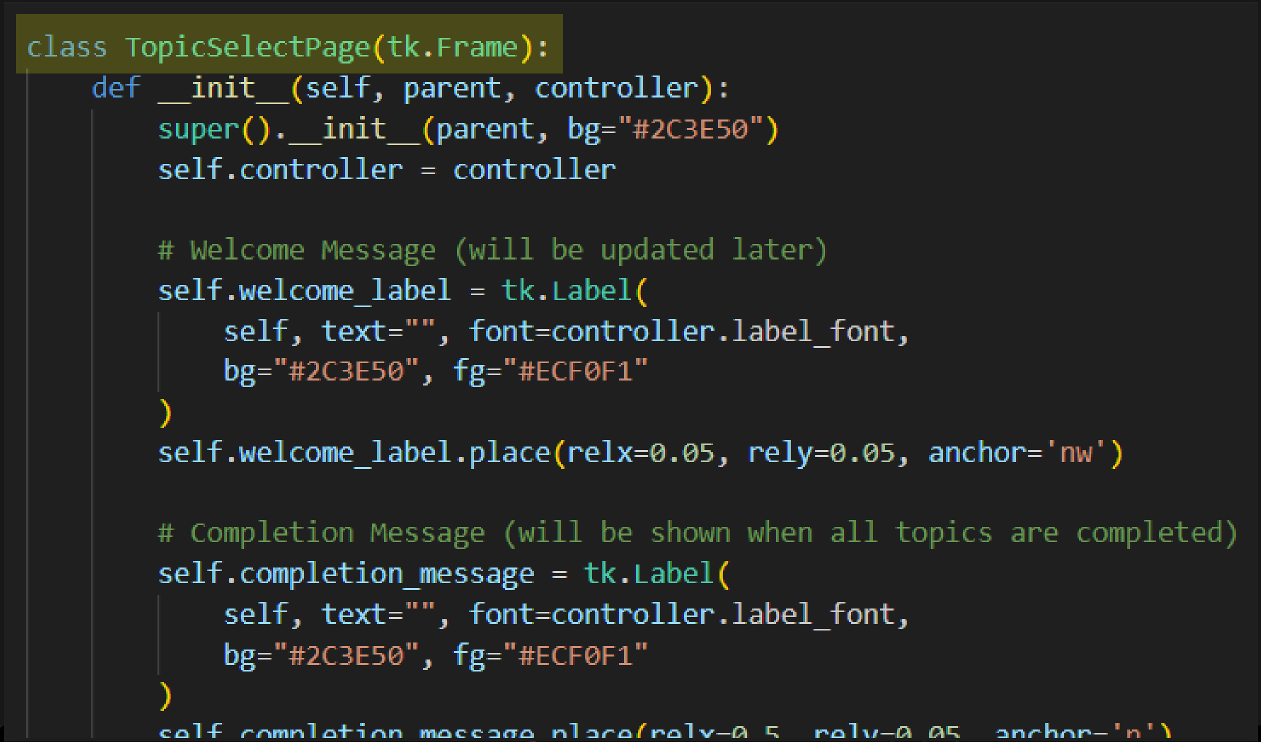
Incorrect guesses: h, j

User Interface Code



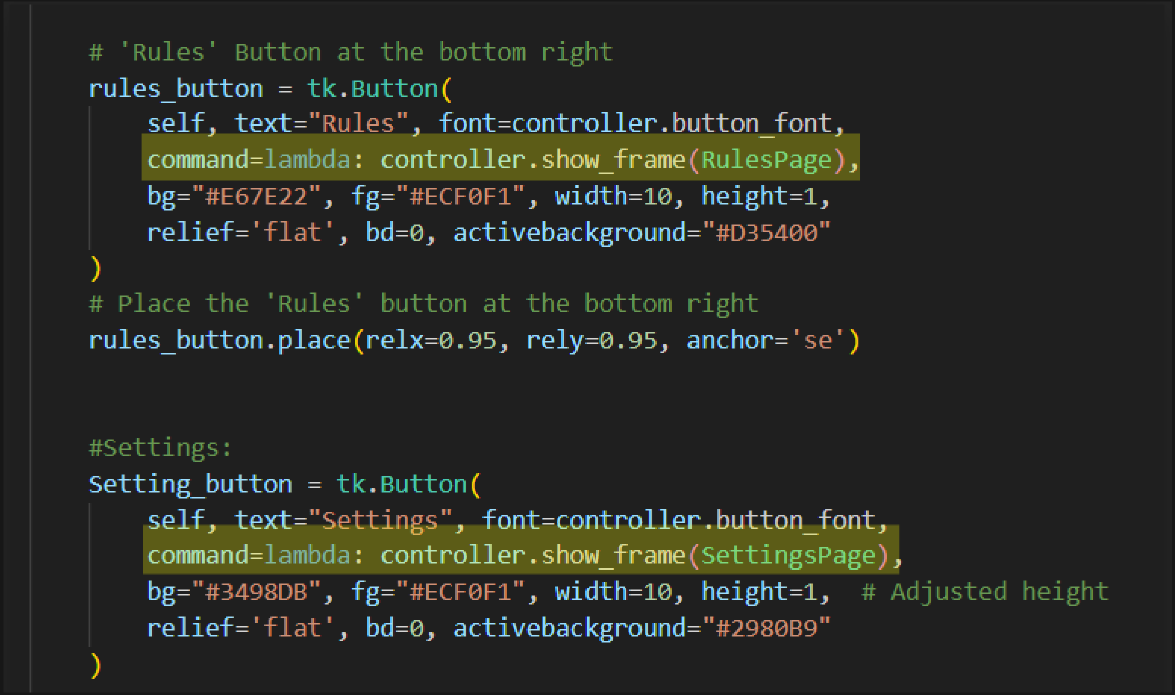
```python
class HangmanGamePage(tk.Frame):
    def __init__(self, parent, controller, topic_name, words_list, questions_list):
        super().__init__(parent, bg="#2C3E50")
        self.controller = controller
        self.topic_name = topic_name
        self.words_list = words_list
        self.questions_list = questions_list
        self.word_index = 0
        self.current_word = self.words_list[self.word_index]
        self.current_question = self.questions_list[self.word_index]  # Get the first question
        self.guessed_letters = set()
        self.incorrect_guesses = set()
        self.timer_id = None
        self.time_left = 60  # 1 minute timer for hard level
        self.life_gained = False
        self.current_difficulty_music = None

        # Set the initial hangman image (full lives should show Hangman #1)
        self.hangman_label = tk.Label(self, image=self.controller.hangman_images[0], bg="#2C3E50")
        self.hangman_label.place(relx=0.05, rely=0.6, anchor="w")

        #Define file paths for different difficulty music files
        base_dir1 = os.path.dirname(os.path.abspath(__file__))
        self.music_file_easy = os.path.join(base_dir1, 'Sound', 'EASY.mp3')
        self.music_file_medium = os.path.join(base_dir1, 'Sound', 'Medium.mp3')
        self.music_file_hard = os.path.join(base_dir1, 'Sound', 'Hard.mp3')
```

User Interface Code



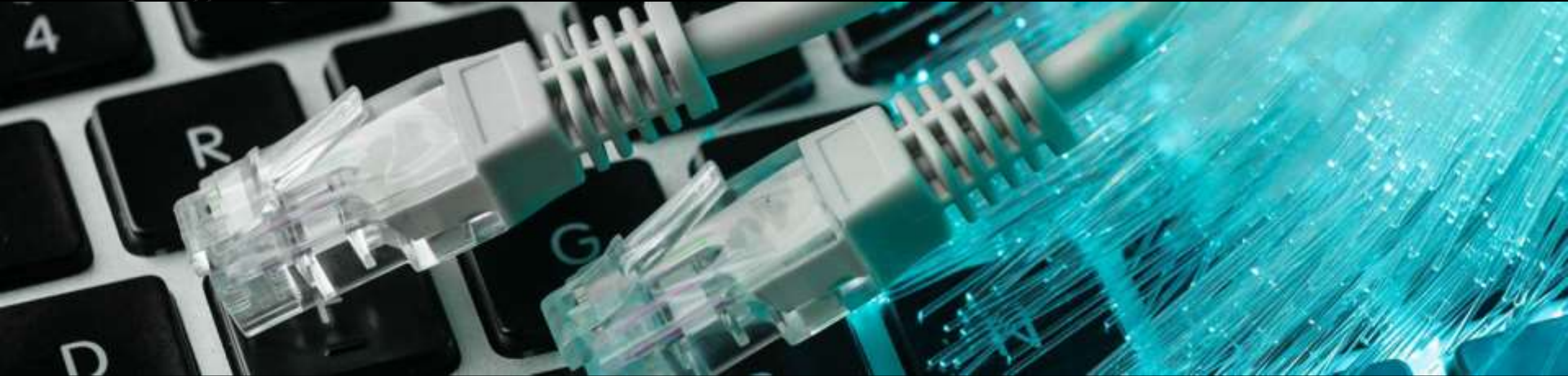
```
class TopicSelectPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent, bg="#2C3E50")
        self.controller = controller

        # Welcome Message (will be updated later)
        self.welcome_label = tk.Label(
            self, text="", font=controller.label_font,
            bg="#2C3E50", fg="#ECF0F1"
        )
        self.welcome_label.place(relx=0.05, rely=0.05, anchor='nw')

        # Completion Message (will be shown when all topics are completed)
        self.completion_message = tk.Label(
            self, text="", font=controller.label_font,
            bg="#2C3E50", fg="#ECF0F1"
        )
        self.completion_message.place(relx=0.5, rely=0.05, anchor='n')
```

User Interface Code



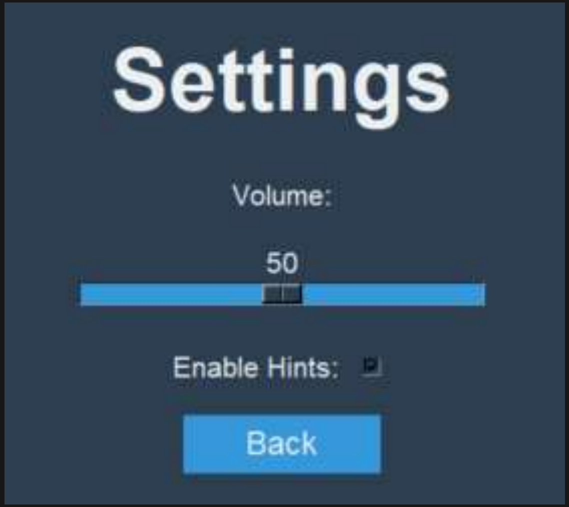
```
# 'Rules' Button at the bottom right
rules_button = tk.Button(
    self, text="Rules", font=controller.button_font,
    command=lambda: controller.show_frame(RulesPage),
    bg="#E67E22", fg="#ECF0F1", width=10, height=1,
    relief='flat', bd=0, activebackground="#D35400"
)
# Place the 'Rules' button at the bottom right
rules_button.place(relx=0.95, rely=0.95, anchor='se')


#Settings:
Setting_button = tk.Button(
    self, text="Settings", font=controller.button_font,
    command=lambda: controller.show_frame(SettingsPage),
    bg="#3498DB", fg="#ECF0F1", width=10, height=1,  # Adjusted height
    relief='flat', bd=0, activebackground="#2980B9"
)
```

03 Settings Page



Settings UI

- In our settings page, we have the two main functions
- Adjusting of the music volume
- Allowing user to enable hints
- User to go back to mainpage



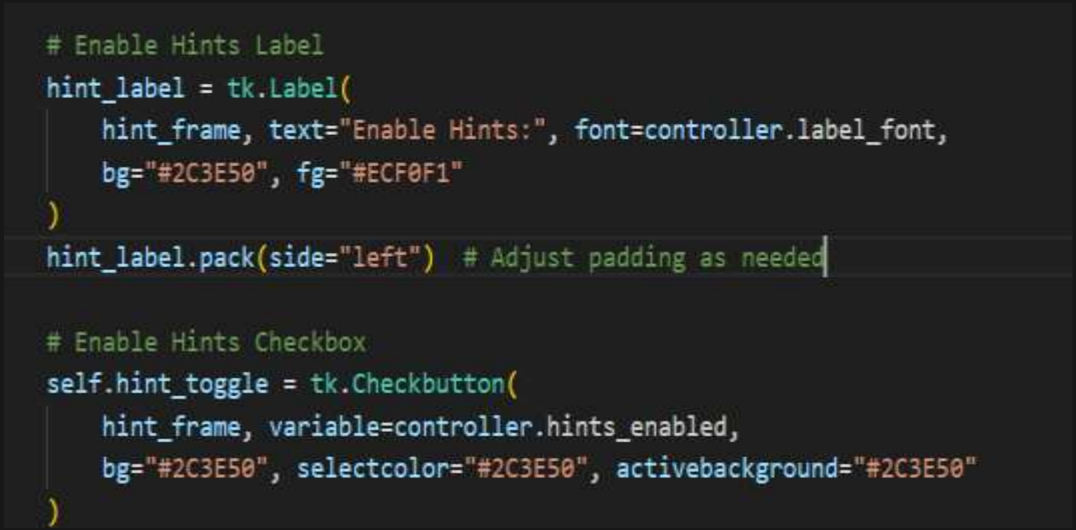
Settings

Volume:

50

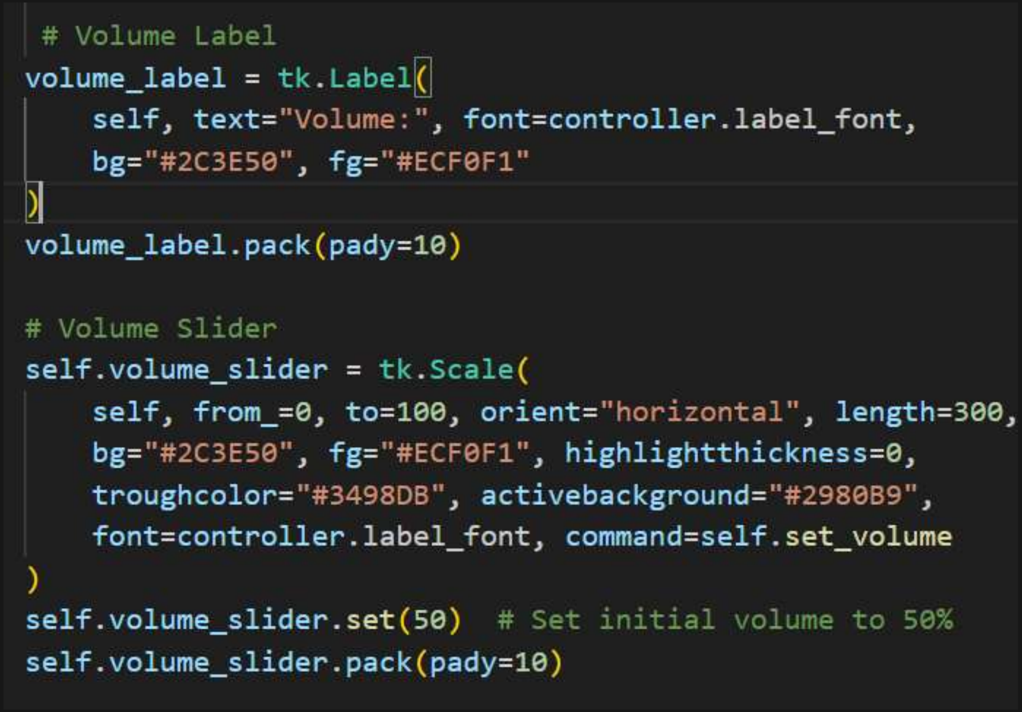
Enable Hints: □

Source code for hint UI



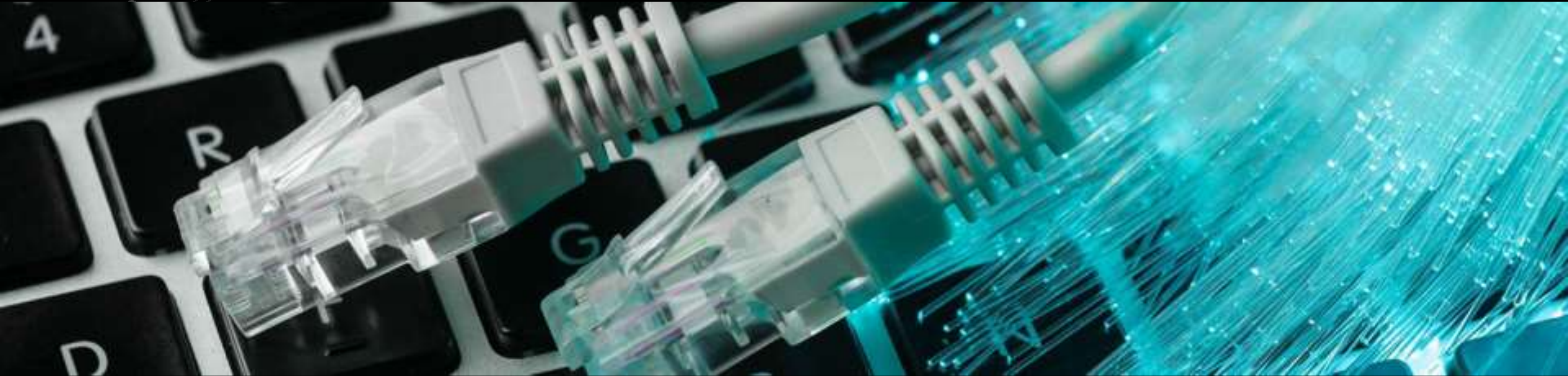
```
# Enable Hints Label
hint_label = tk.Label(
    hint_frame, text="Enable Hints:", font=controller.label_font,
    bg="#2C3E50", fg="#ECF0F1"
)
hint_label.pack(side="left")  # Adjust padding as needed

# Enable Hints Checkbox
self.hint_toggle = tk.Checkbutton(
    hint_frame, variable=controller.hints_enabled,
    bg="#2C3E50", selectcolor="#2C3E50", activebackground="#2C3E50"
)
```

Source code for Volume UI



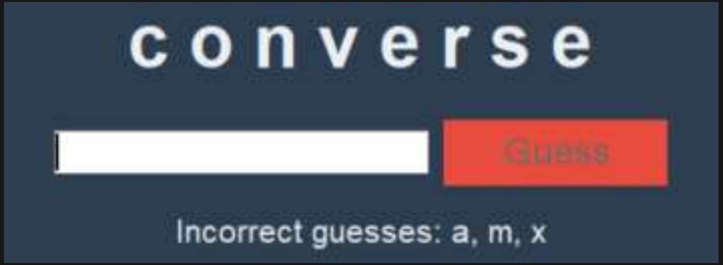
```
# Volume Label
volume_label = tk.Label(
    self, text="Volume:", font=controller.label_font,
    bg="#2C3E50", fg="#ECF0F1"
)
volume_label.pack(pady=10)

# Volume Slider
self.volume_slider = tk.Scale(
    self, from_=0, to=100, orient="horizontal", length=300,
    bg="#2C3E50", fg="#ECF0F1", highlightthickness=0,
    troughcolor="#3498DB", activebackground="#2980B9",
    font=controller.label_font, command=self.set_volume
)
self.volume_slider.set(50)  # Set initial volume to 50%
self.volume_slider.pack(pady=10)
```

04 Game Logic (Hint)



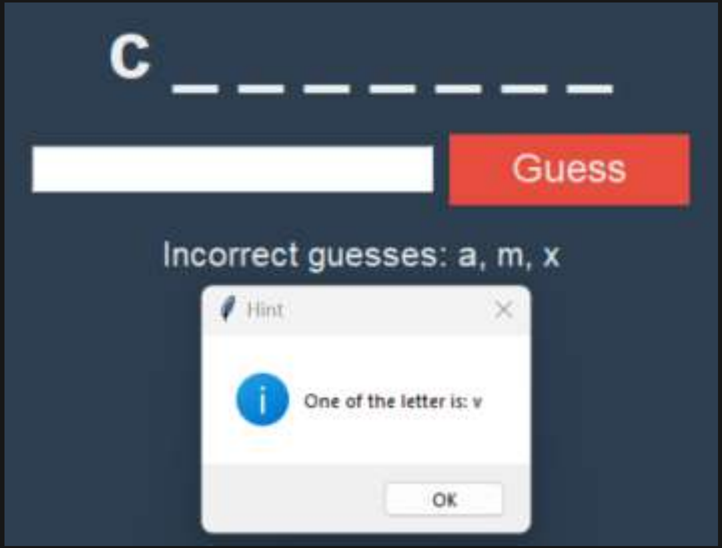
Hint feature

- In this test scenario, the user input 1 correct guess and 3 incorrect guesses
- The hint message box pop up would be shown for the answer.



c o n v e r s e

Incorrect guesses: a, m, x



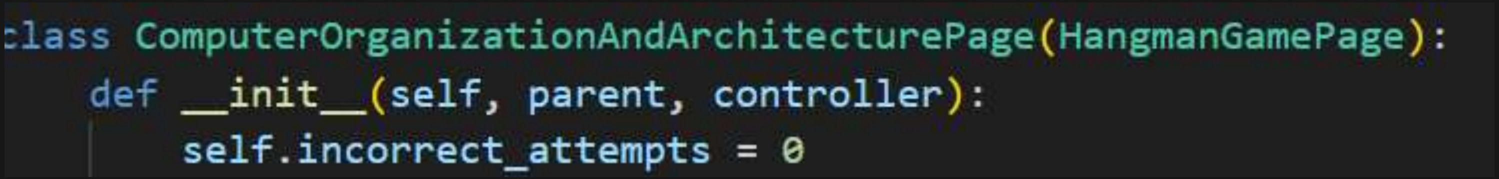
C _ _ _ _ _ _ _

Incorrect guesses: a, m, x

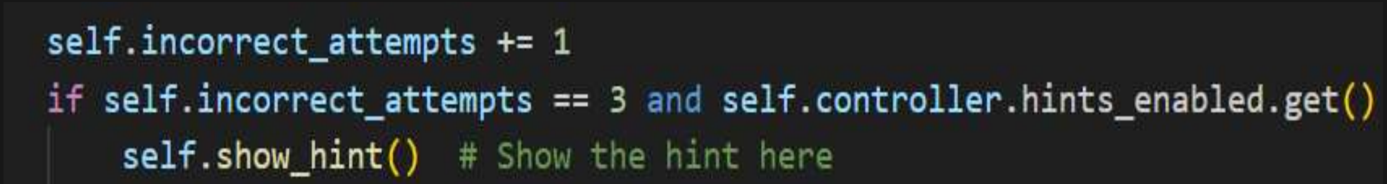
Hint

One of the letter is: v

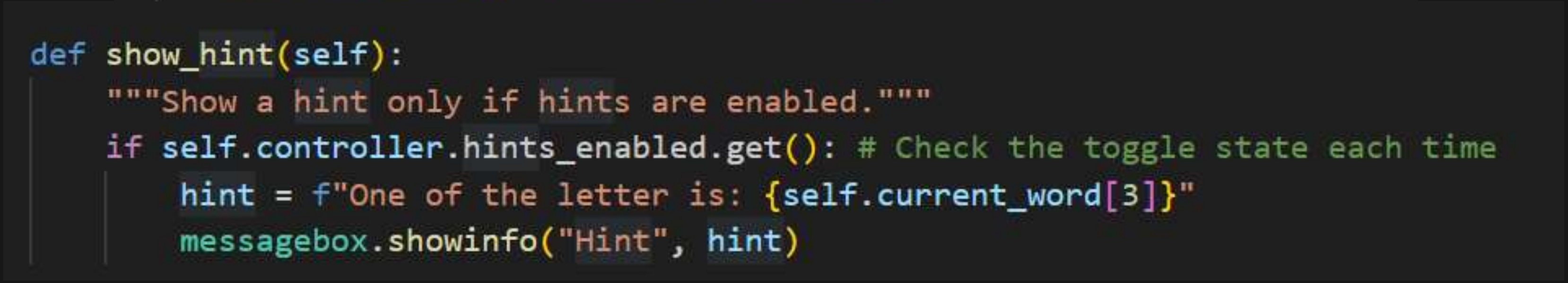
Source code for hint



```python
class ComputerOrganizationAndArchitecturePage(HangmanGamePage):
    def __init__(self, parent, controller):
        self.incorrect_attempts = 0
```

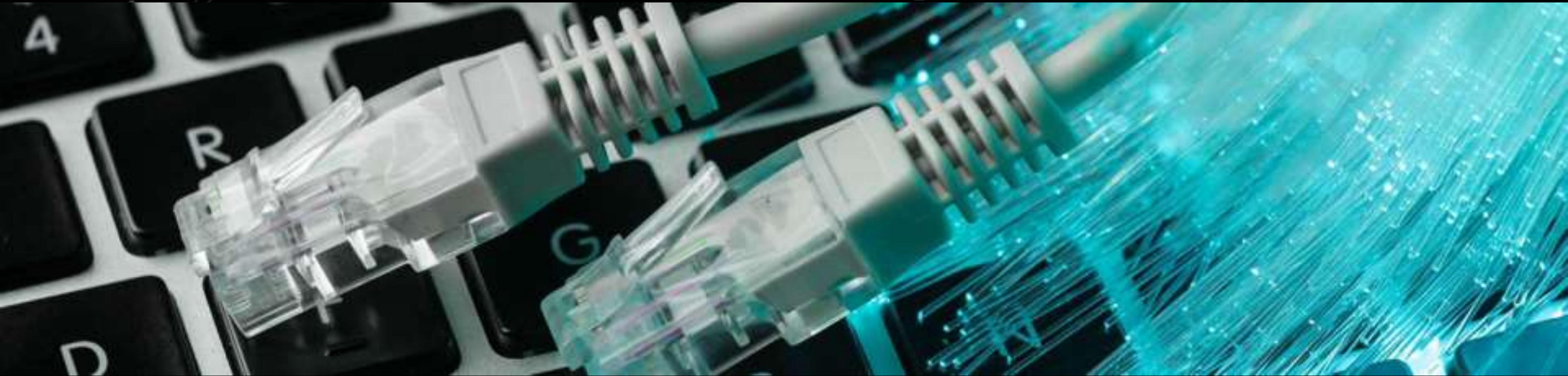


```python
self.incorrect_attempts += 1
if self.incorrect_attempts == 3 and self.controller.hints_enabled.get()
    self.show_hint()  # Show the hint here
```



```python
def show_hint(self):
    """Show a hint only if hints are enabled."""
    if self.controller.hints_enabled.get(): # Check the toggle state each time
        hint = f"One of the letter is: {self.current_word[3]}"
        messagebox.showinfo("Hint", hint)
```

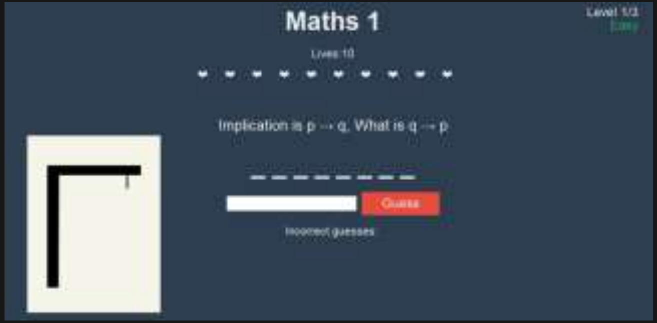
05 Game Logic (Difficulty)



Overview on difficulty

- Difficulty Levels: Easy/Medium/Hard

Easy & Medium:



Maths 1

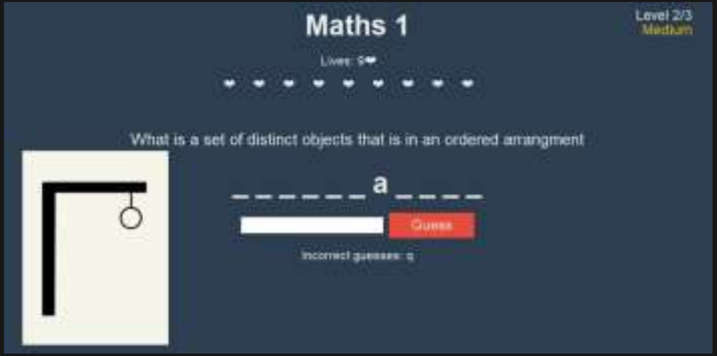
Level 1/3
Easy

Lives:10

Implication is p → q. What is q → p

_ _ _ _ _ _ _ _

Incorrect guesses:



Maths 1

Level 2/3
Medium

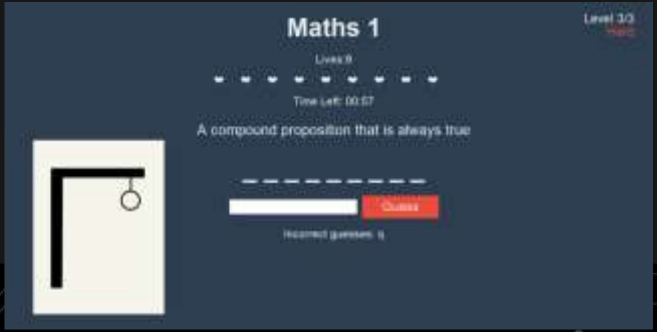
Lives: 9❤

What is a set of distinct objects that is in an ordered arrangment

_ _ _ _ _ _ a _ _ _ _

Incorrect guesses: q

Hard (Contains a timer of 60 seconds):



Maths 1

Level 3/3
Hard

Lives:9

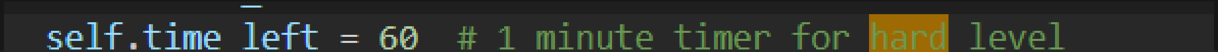
Time Left: 00:57

A compound proposition that is always true

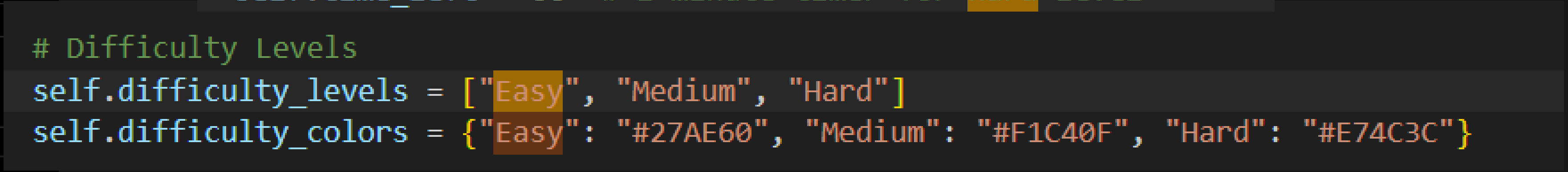
_ _ _ _ _ _ _ _ _

Incorrect guesses: q

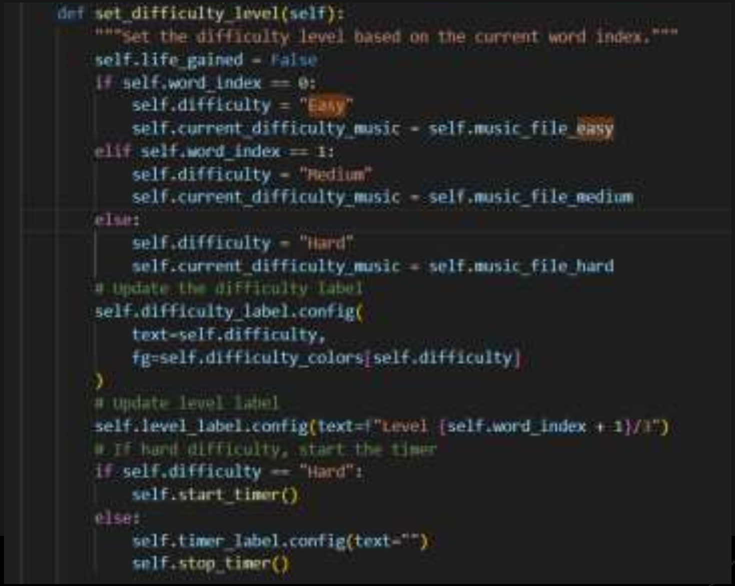
Source code for difficulty



```
self.time_left = 60  # 1 minute timer for hard level
```

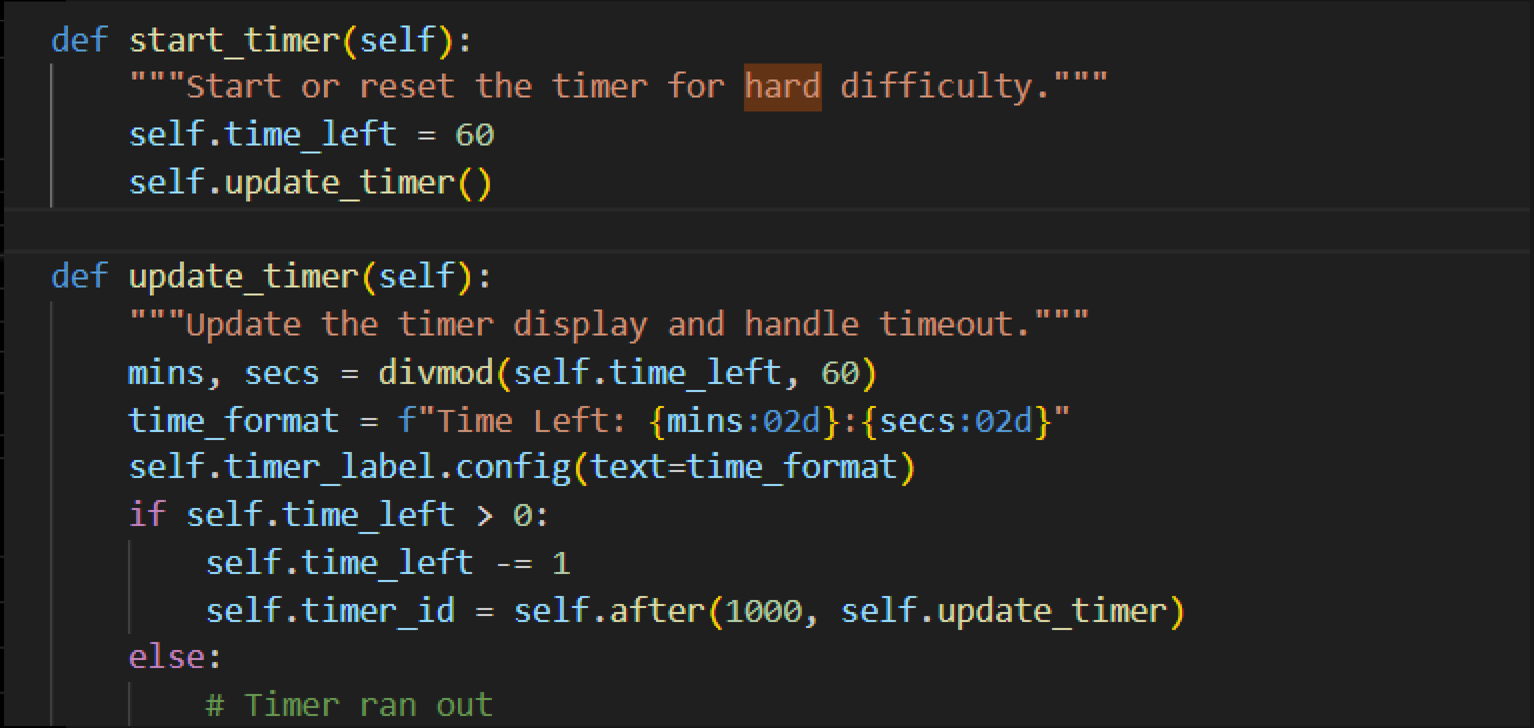


```
# Difficulty Levels
self.difficulty_levels = ["Easy", "Medium", "Hard"]
self.difficulty_colors = {"Easy": "#27AE60", "Medium": "#F1C40F", "Hard": "#E74C3C"}
```



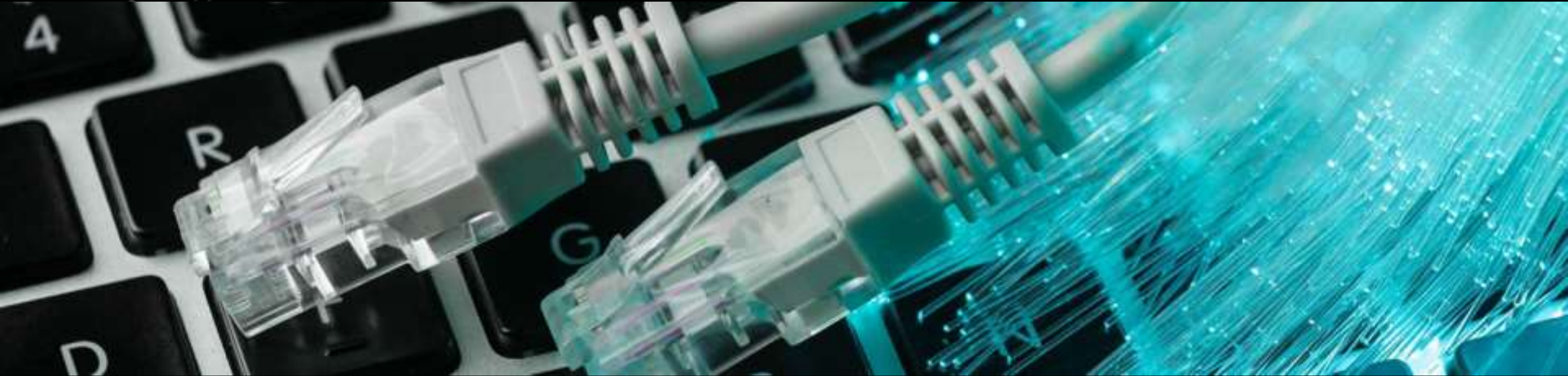
```
def set_difficulty_level(self):
    """Set the difficulty level based on the current word index."""
    self.life_gained = False
    if self.word_index == 0:
        self.difficulty = "Easy"
        self.current_difficulty_music = self.music_file_easy
    elif self.word_index == 1:
        self.difficulty = "Medium"
        self.current_difficulty_music = self.music_file_medium
    else:
        self.difficulty = "Hard"
        self.current_difficulty_music = self.music_file_hard
    # Update the difficulty label
    self.difficulty_label.config(
        text=self.difficulty,
        fg=self.difficulty_colors[self.difficulty]
    )
    # Update level label
    self.level_label.config(text=f"Level {self.word_index + 1}/3")
    # If hard difficulty, start the timer
    if self.difficulty == "Hard":
        self.start_timer()
    else:
        self.timer_label.config(text="")
        self.stop_timer()
```

Source code for difficulty



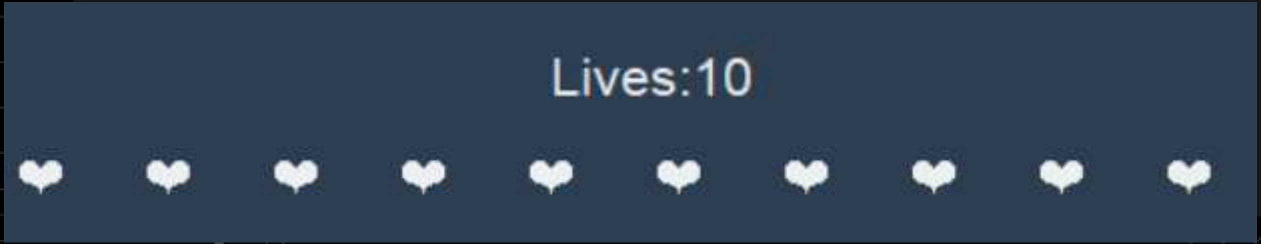
```python
def start_timer(self):
    """Start or reset the timer for hard difficulty."""
    self.time_left = 60
    self.update_timer()

def update_timer(self):
    """Update the timer display and handle timeout."""
    mins, secs = divmod(self.time_left, 60)
    time_format = f"Time Left: {mins:02d}:{secs:02d}"
    self.timer_label.config(text=time_format)
    if self.time_left > 0:
        self.time_left -= 1
        self.timer_id = self.after(1000, self.update_timer)
    else:
        # Timer ran out
```

06 Game Logic (Health & Regen)



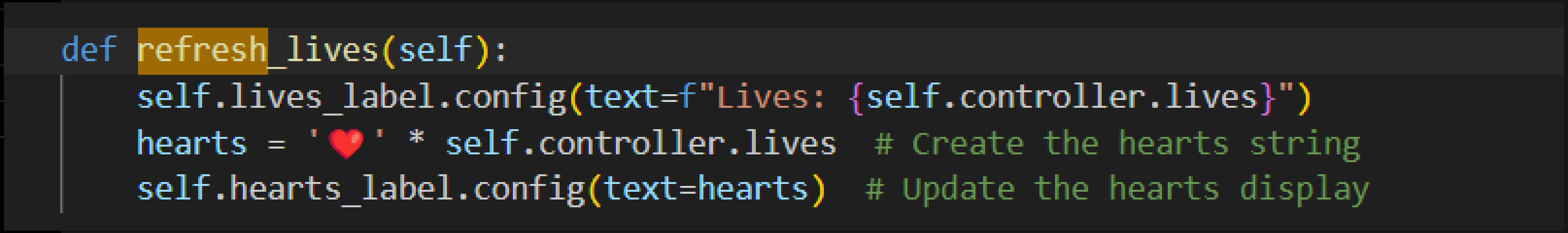
Overview on health & Regen

- User start off with a total of 10 lives
- For every wrongly guess letter, 1 lives will be minus off
- To gain the 1 live:
    - Easy → 3 correct letters
    - Medium & Hard → 4 correct letters
- If lives are gone, user will need to restart the whole game (go back to login screen)

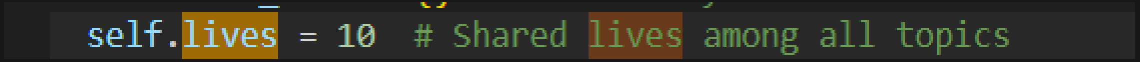


Lives:10

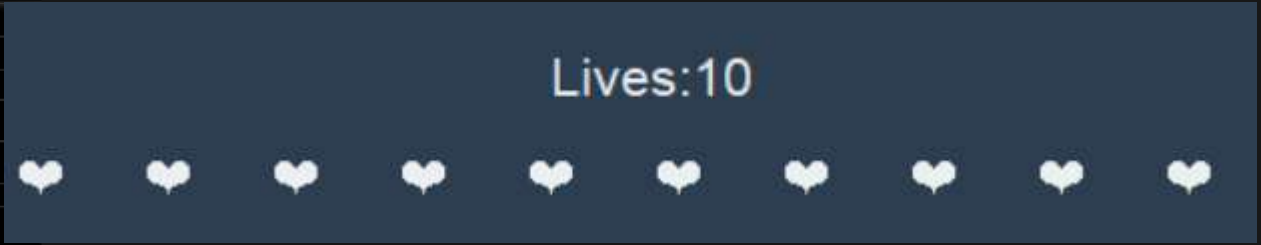
Source code for health



```
def refresh_lives(self):
    self.lives_label.config(text=f"Lives: {self.controller.lives}")
    hearts = '❤️' * self.controller.lives  # Create the hearts string
    self.hearts_label.config(text=hearts)  # Update the hearts display
```

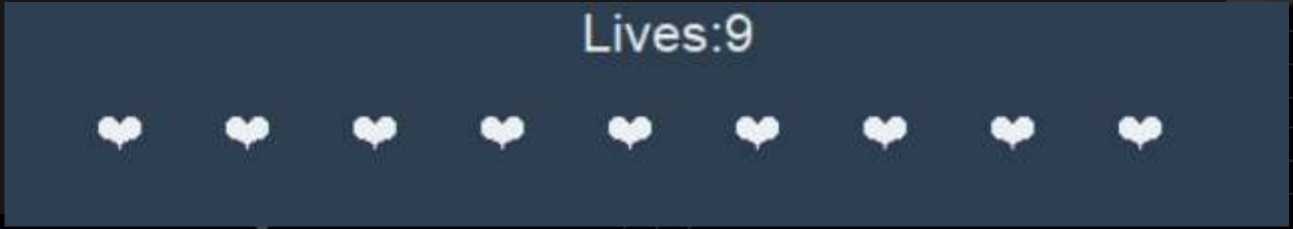


```
    self.lives = 10  # Shared lives among all topics
```



Lives:10

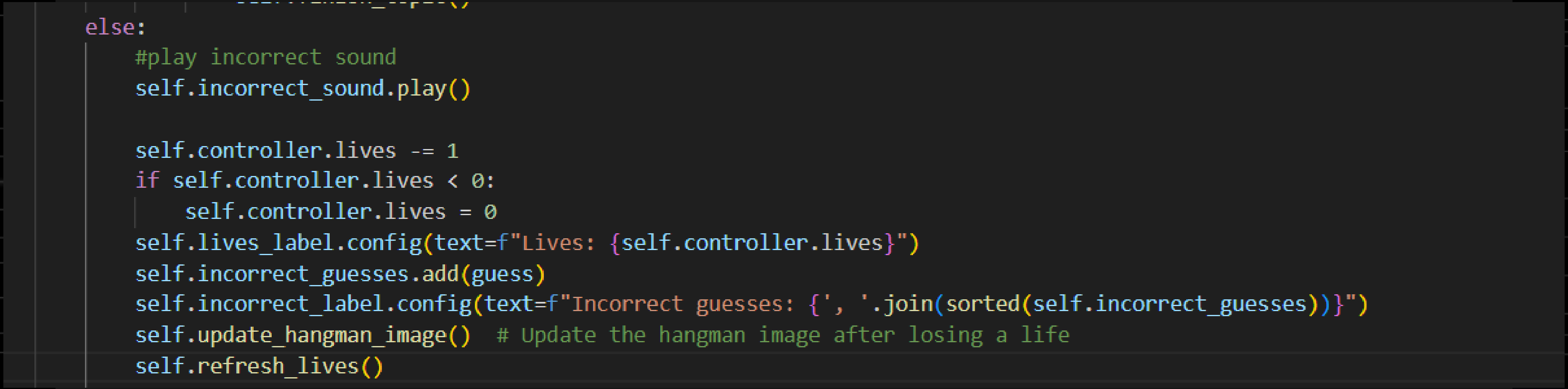
❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤



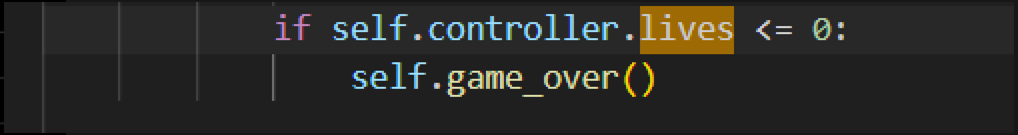
Lives:9

❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤

Source code for health (No timer)

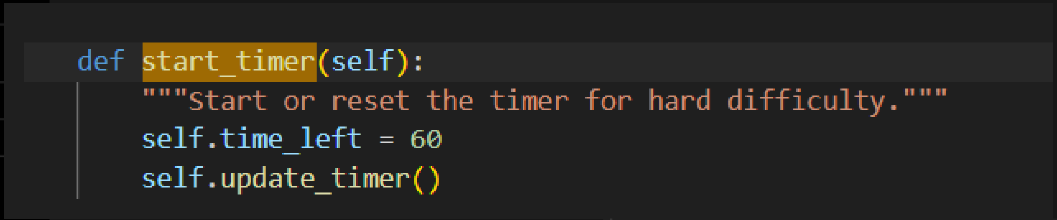


```
else:
    #play incorrect sound
    self.incorrect_sound.play()

    self.controller.lives -= 1
    if self.controller.lives < 0:
        self.controller.lives = 0
    self.lives_label.config(text=f"Lives: {self.controller.lives}")
    self.incorrect_guesses.add(guess)
    self.incorrect_label.config(text=f"Incorrect guesses: {', '.join(sorted(self.incorrect_guesses))}")
    self.update_hangman_image()  # Update the hangman image after losing a life
    self.refresh_lives()
```

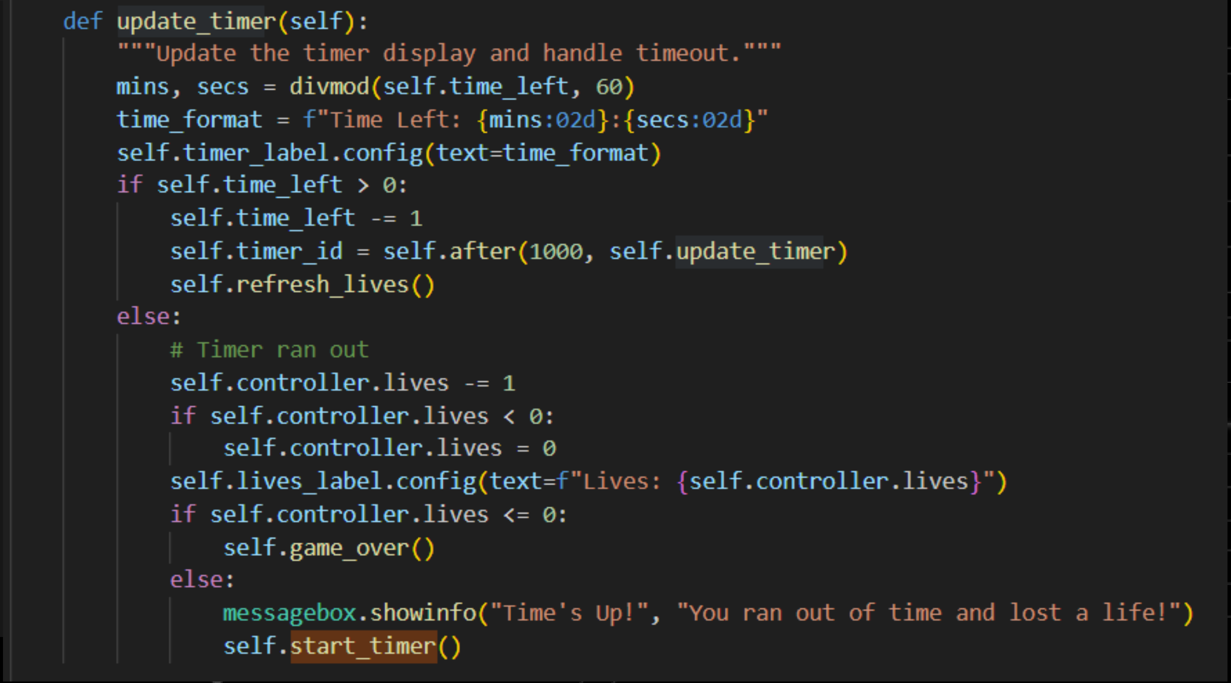


```
if self.controller.lives <= 0:
    self.game_over()
```

Source code for health (With Timer)

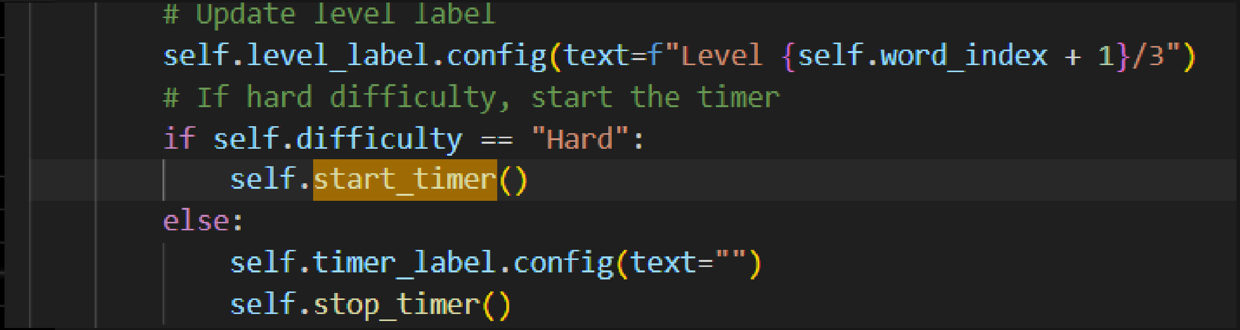


```python
    def start_timer(self):
        """Start or reset the timer for hard difficulty."""
        self.time_left = 60
        self.update_timer()
```

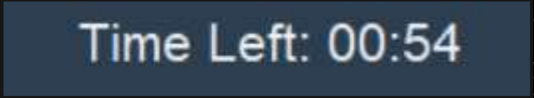


```python
    def update_timer(self):
        """Update the timer display and handle timeout."""
        mins, secs = divmod(self.time_left, 60)
        time_format = f"Time Left: {mins:02d}:{secs:02d}"
        self.timer_label.config(text=time_format)
        if self.time_left > 0:
            self.time_left -= 1
            self.timer_id = self.after(1000, self.update_timer)
            self.refresh_lives()
        else:
            # Timer ran out
            self.controller.lives -= 1
            if self.controller.lives < 0:
                self.controller.lives = 0
            self.lives_label.config(text=f"Lives: {self.controller.lives}")
            if self.controller.lives <= 0:
                self.game_over()
            else:
                messagebox.showinfo("Time's Up!", "You ran out of time and lost a life!")
                self.start_timer()
```

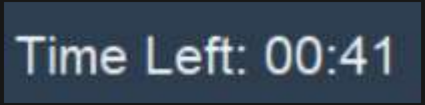
Source code for health (With Timer)



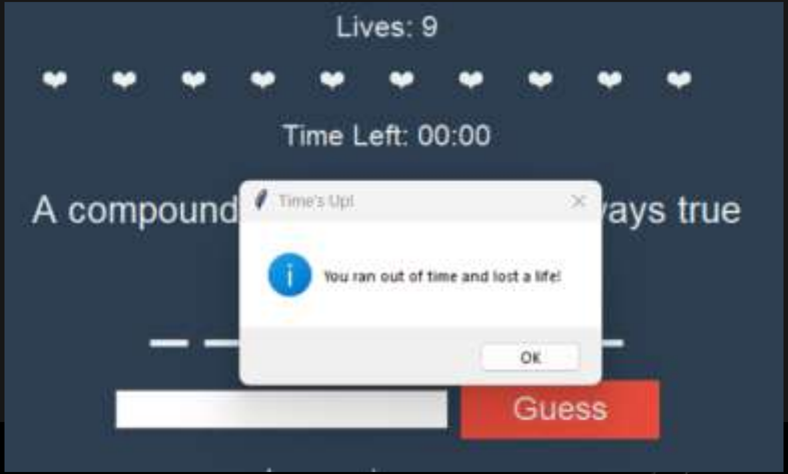
```
# Update level label
self.level_label.config(text=f"Level {self.word_index + 1}/3")
# If hard difficulty, start the timer
if self.difficulty == "Hard":
    self.start_timer()
else:
    self.timer_label.config(text="")
    self.stop_timer()
```



Time Left: 00:54



Time Left: 00:41



Lives: 9

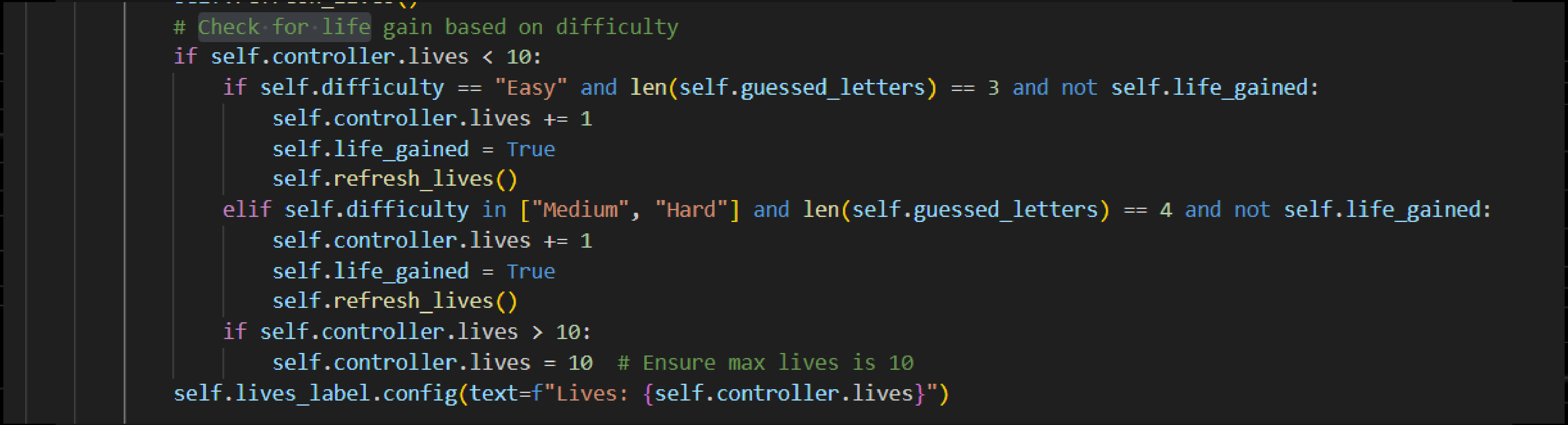
❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤

Time Left: 00:00

A compound ... ays true

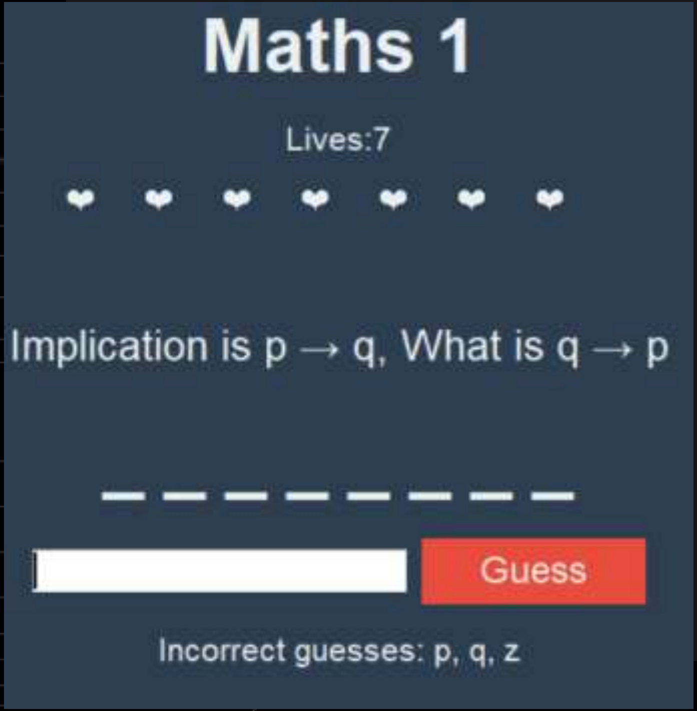
You ran out of time and lost a life!

Source code for Lives gain (Regen)



```
self.refresh_lives()
# Check for life gain based on difficulty
if self.controller.lives < 10:
    if self.difficulty == "Easy" and len(self.guessed_letters) == 3 and not self.life_gained:
        self.controller.lives += 1
        self.life_gained = True
        self.refresh_lives()
    elif self.difficulty in ["Medium", "Hard"] and len(self.guessed_letters) == 4 and not self.life_gained:
        self.controller.lives += 1
        self.life_gained = True
        self.refresh_lives()
    if self.controller.lives > 10:
        self.controller.lives = 10  # Ensure max lives is 10
self.lives_label.config(text=f"Lives: {self.controller.lives}")
```

Source code for Lives gain (Regen)



Maths 1

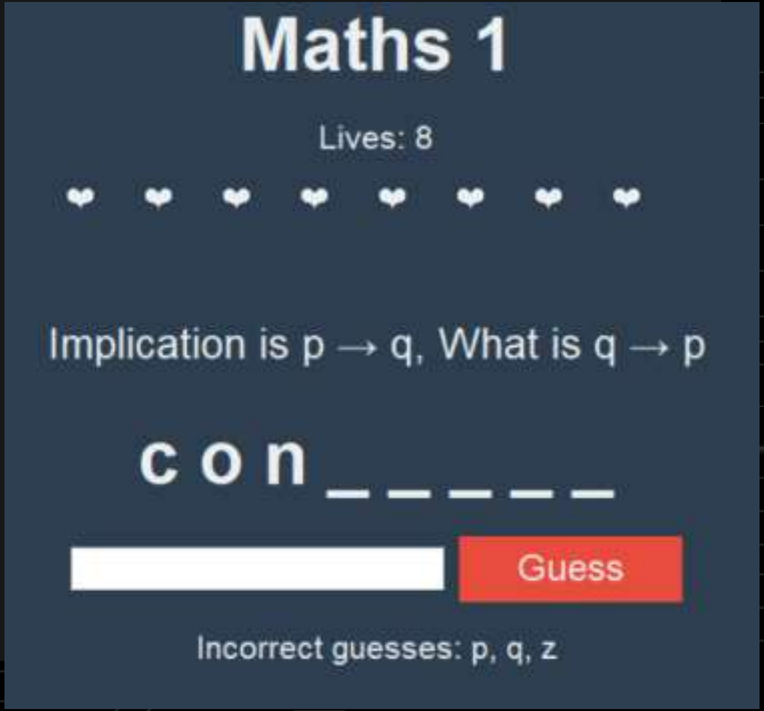
Lives:7

❤ ❤ ❤ ❤ ❤ ❤ ❤

Implication is p → q, What is q → p

_ _ _ _ _ _ _ _

Incorrect guesses: p, q, z



Maths 1

Lives: 8

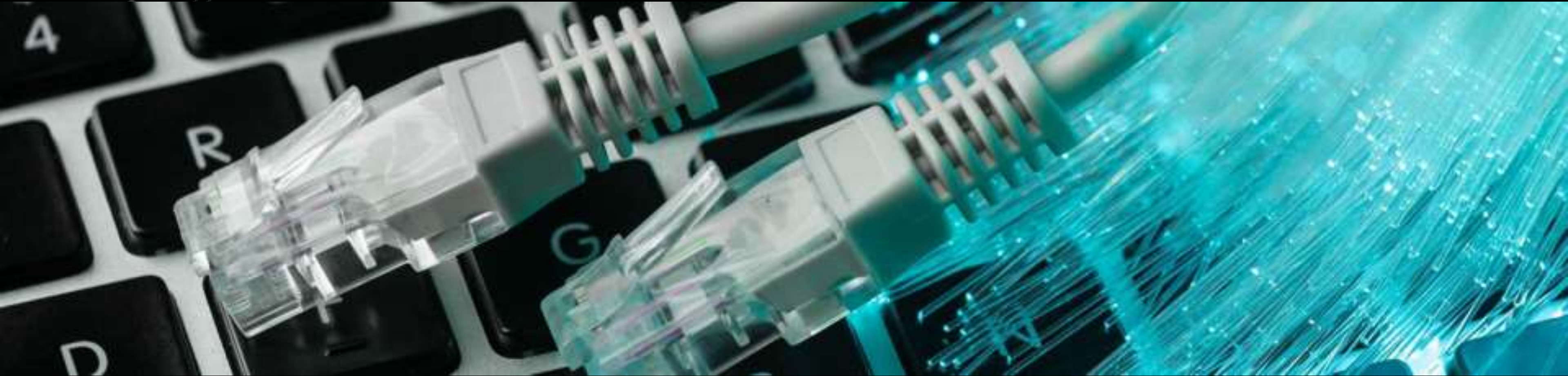
❤ ❤ ❤ ❤ ❤ ❤ ❤ ❤

Implication is p → q, What is q → p

c o n _ _ _ _ _

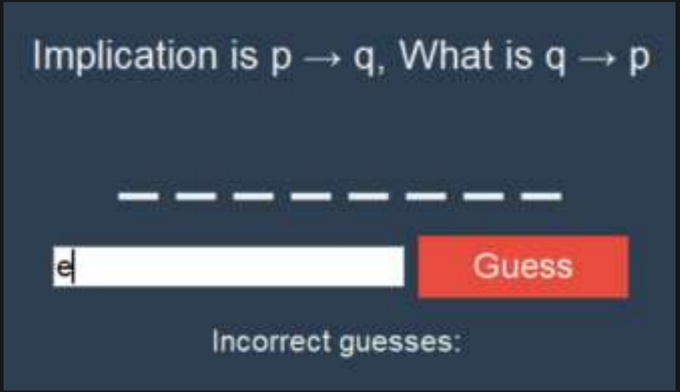
Incorrect guesses: p, q, z

07 Game Logic (Validation)



Input Validation
Valid input

When the user inputs an alphabet that is in the answer

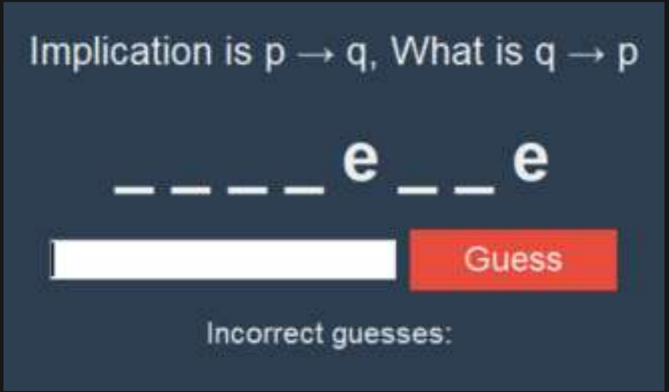


Implication is p → q, What is q → p

_ _ _ _ _ _ _ _

e

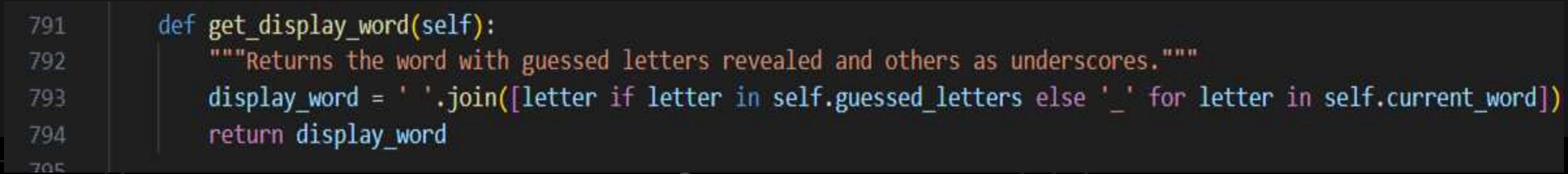
Incorrect guesses:



Implication is p → q, What is q → p

_ _ _ _ e _ _ e

Incorrect guesses:

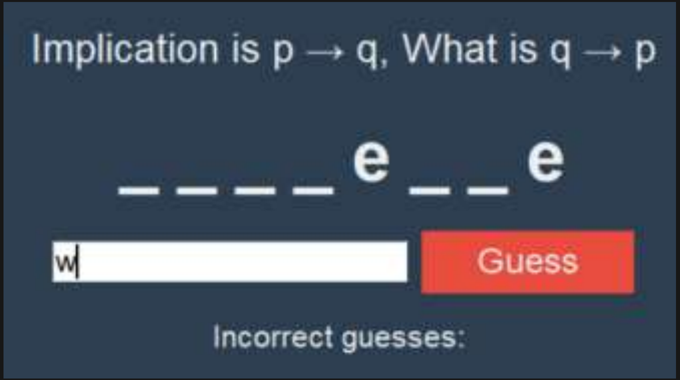


```python
    def get_display_word(self):
        """Returns the word with guessed letters revealed and others as underscores."""
        display_word = ' '.join([letter if letter in self.guessed_letters else '_' for letter in self.current_word])
        return display_word
```

Input Validation

Valid input

When the user inputs an alphabet that is NOT in the answer

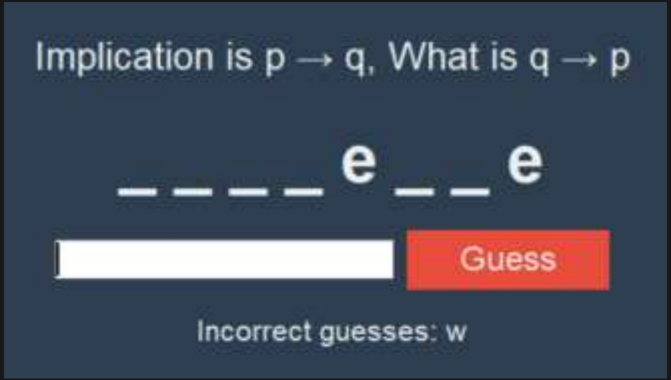


Implication is p → q, What is q → p

_ _ _ _ e _ _ e

w

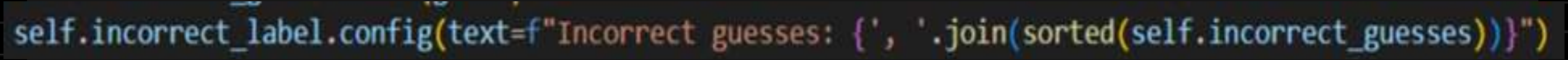
Incorrect guesses:



Implication is p → q, What is q → p

_ _ _ _ e _ _ e

Incorrect guesses: w

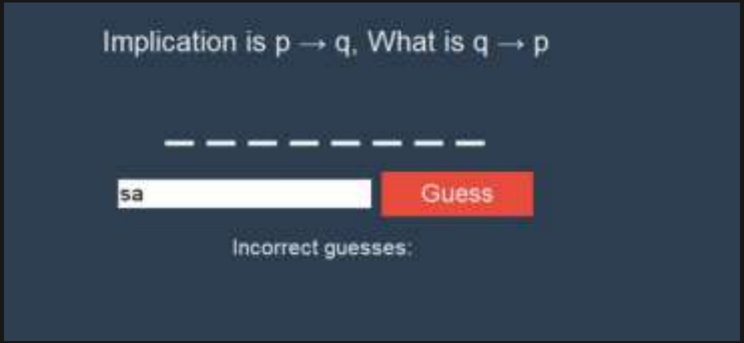


```
self.incorrect_label.config(text=f"Incorrect guesses: {', '.join(sorted(self.incorrect_guesses))}")
```

Input Validation

Invalid Input

When the user inputs more than one input

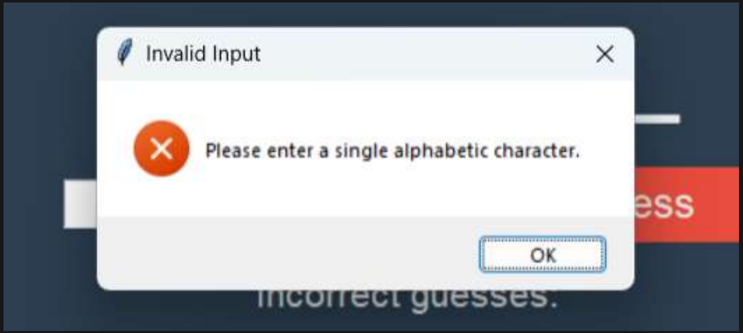


Implication is p → q. What is q → p

_ _ _ _ _ _ _ _

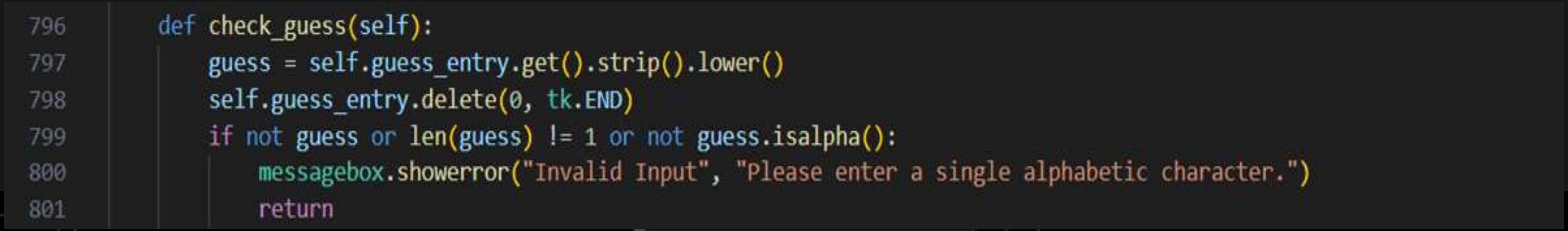
sa

Incorrect guesses:



Invalid Input

Please enter a single alphabetic character.

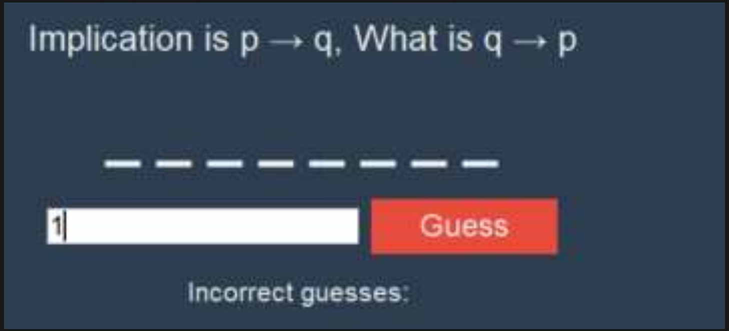


```
def check_guess(self):
    guess = self.guess_entry.get().strip().lower()
    self.guess_entry.delete(0, tk.END)
    if not guess or len(guess) != 1 or not guess.isalpha():
        messagebox.showerror("Invalid Input", "Please enter a single alphabetic character.")
        return
```

Input Validation

Invalid Input

When the user inputs an integer

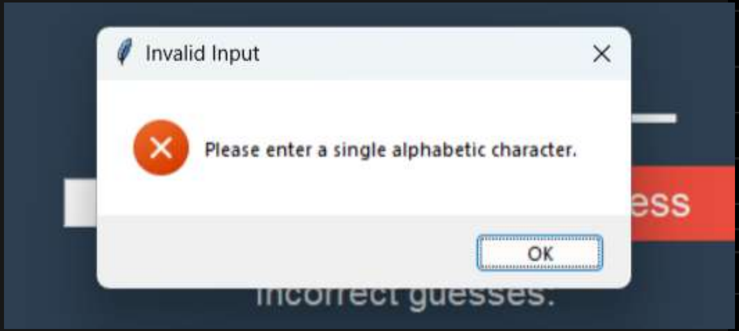


Implication is p → q, What is q → p

_ _ _ _ _ _ _ _

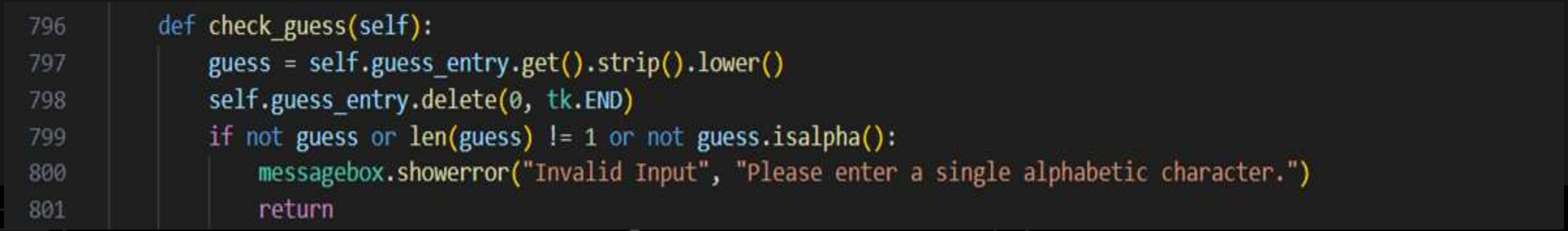
1

Incorrect guesses:



Invalid Input

Please enter a single alphabetic character.

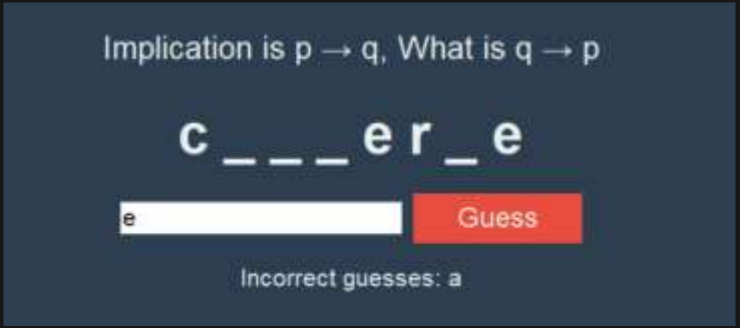


```
def check_guess(self):
    guess = self.guess_entry.get().strip().lower()
    self.guess_entry.delete(0, tk.END)
    if not guess or len(guess) != 1 or not guess.isalpha():
        messagebox.showerror("Invalid Input", "Please enter a single alphabetic character.")
        return
```

Input Validation

Invalid input

When the user inputs a word that has already been input before

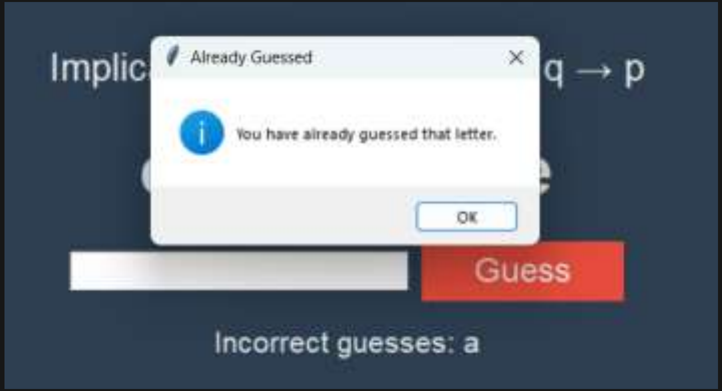


Implication is p → q, What is q → p

c _ _ _ e r _ e

e

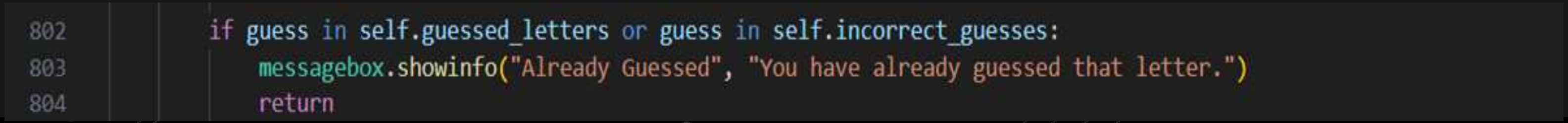
Incorrect guesses: a



Already Guessed

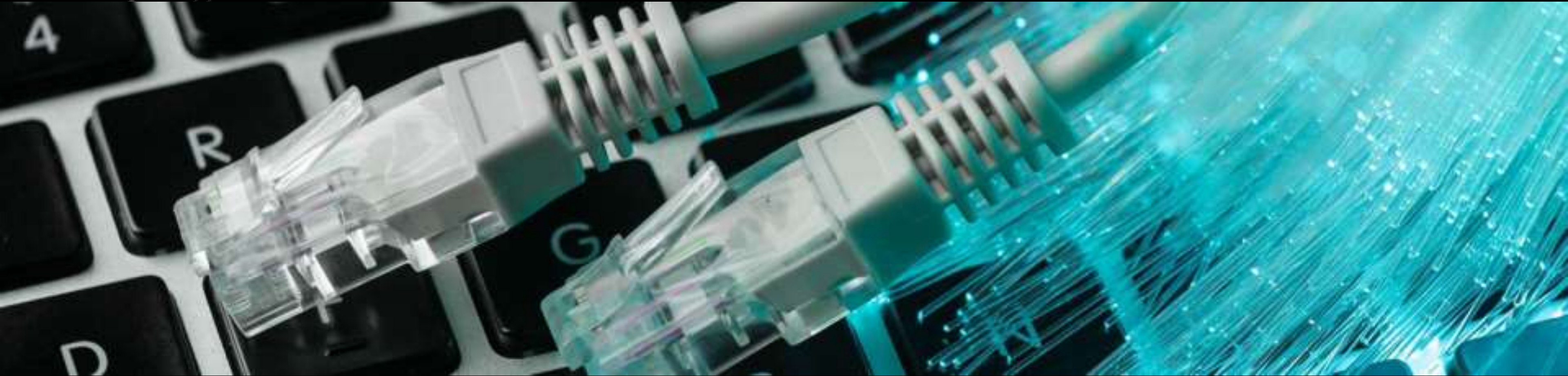
You have already guessed that letter.

Incorrect guesses: a

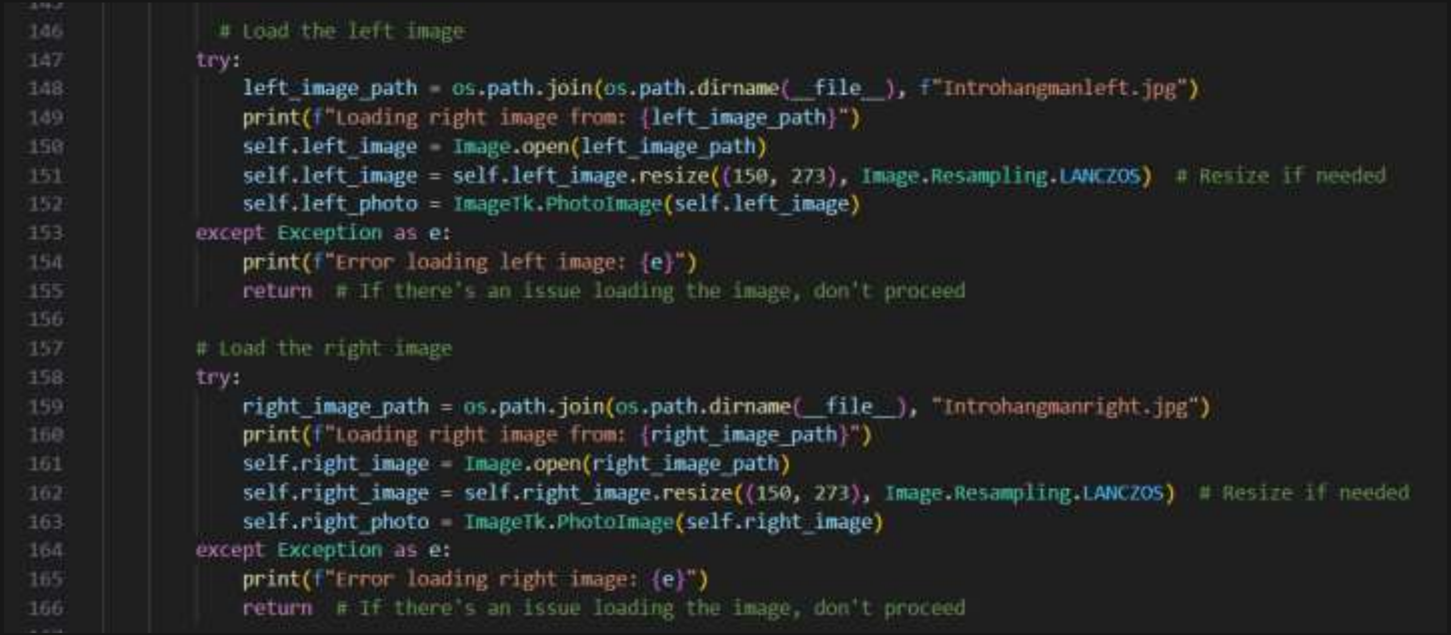


```
if guess in self.guessed_letters or guess in self.incorrect_guesses:
    messagebox.showinfo("Already Guessed", "You have already guessed that letter.")
    return
```

08 Game Logic (Images)



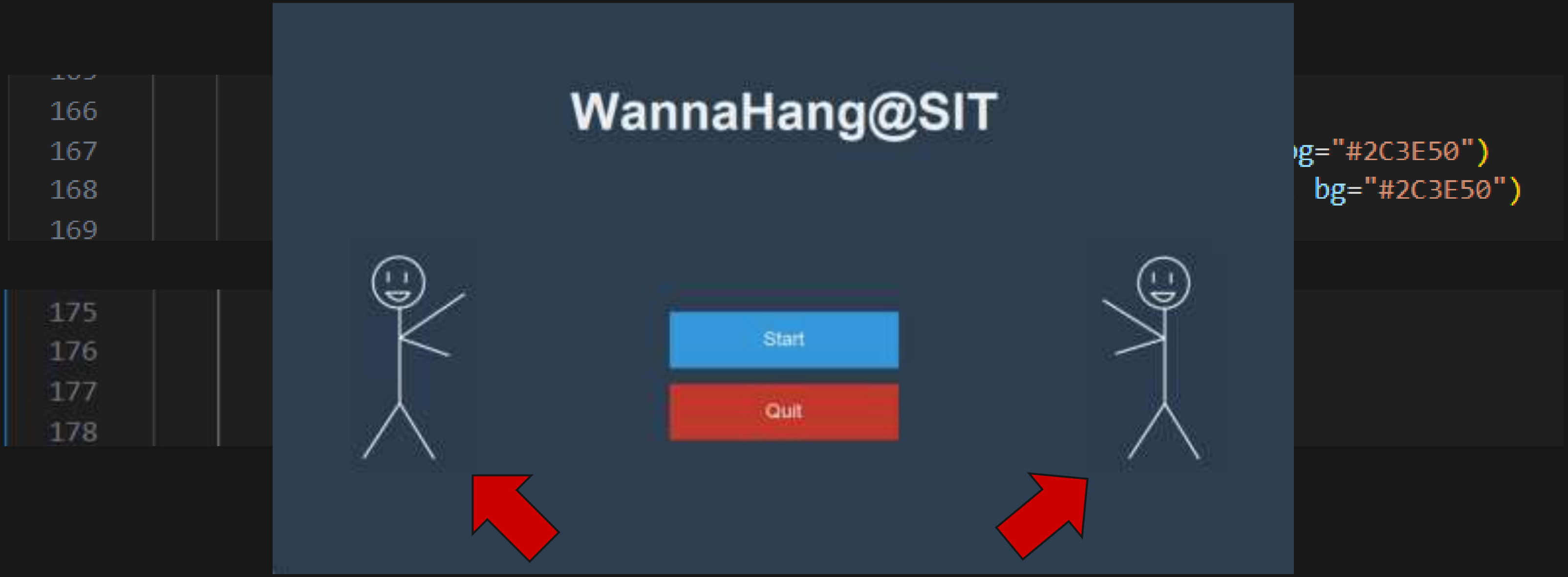
Loading of Images (Login hangman)



```
 # Load the left image
try:
    left_image_path = os.path.join(os.path.dirname(__file__), f"Introhangmanleft.jpg")
    print(f"Loading right image from: {left_image_path}")
    self.left_image = Image.open(left_image_path)
    self.left_image = self.left_image.resize((150, 273), Image.Resampling.LANCZOS)  # Resize if needed
    self.left_photo = ImageTk.PhotoImage(self.left_image)
except Exception as e:
    print(f"Error loading left image: {e}")
    return  # If there's an issue loading the image, don't proceed

# Load the right image
try:
    right_image_path = os.path.join(os.path.dirname(__file__), "Introhangmanright.jpg")
    print(f"Loading right image from: {right_image_path}")
    self.right_image = Image.open(right_image_path)
    self.right_image = self.right_image.resize((150, 273), Image.Resampling.LANCZOS)  # Resize if needed
    self.right_photo = ImageTk.PhotoImage(self.right_image)
except Exception as e:
    print(f"Error loading right image: {e}")
    return  # If there's an issue loading the image, don't proceed
```

g

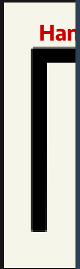
Arranging the Images in the application (Login hangman)

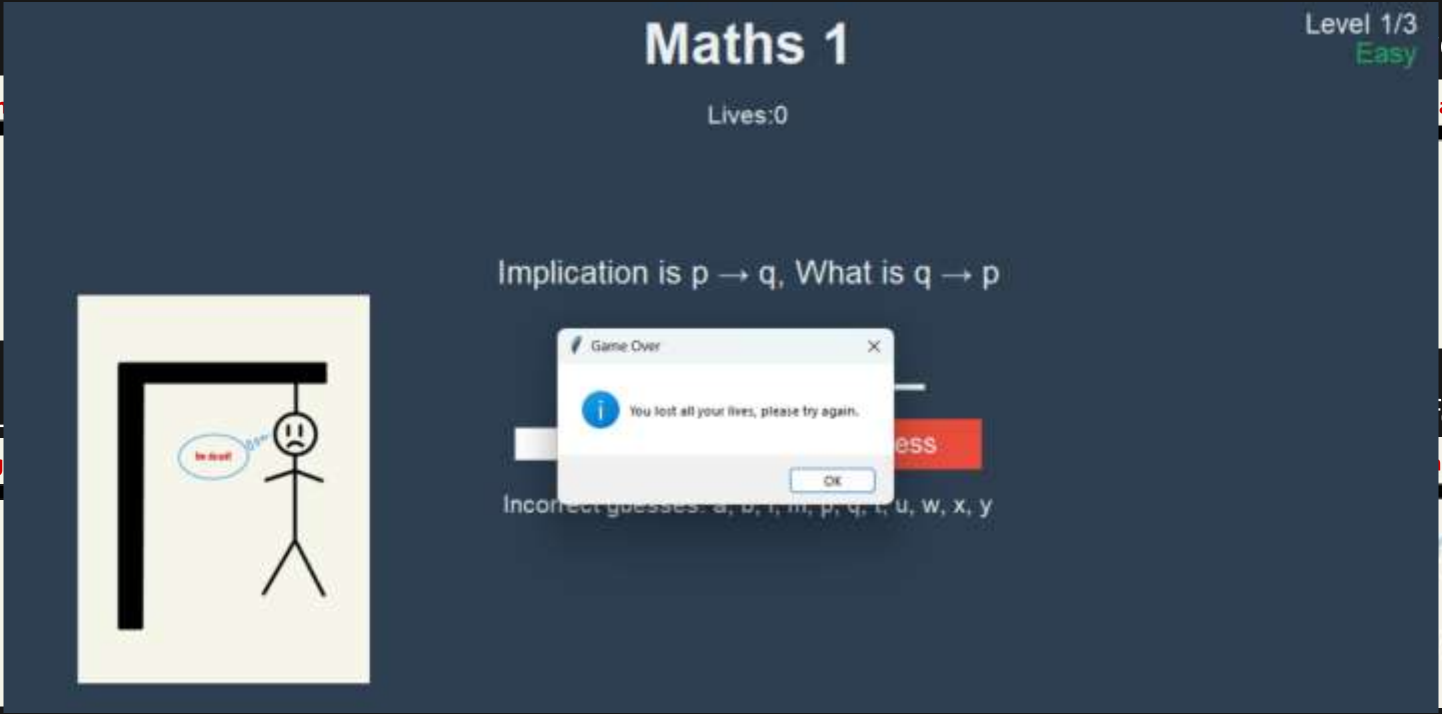


```
g="#2C3E50")
 bg="#2C3E50")
```

Hangman Life stages

10





Maths 1

Level 1/3
Easy

Lives:0

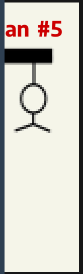
Implication is p → q, What is q → p

Game Over

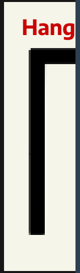
You lost all your lives, please try again.

Incorrect guesses: a, b, f, m, p, q, t, u, w, x, y

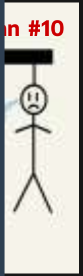
es



5 L

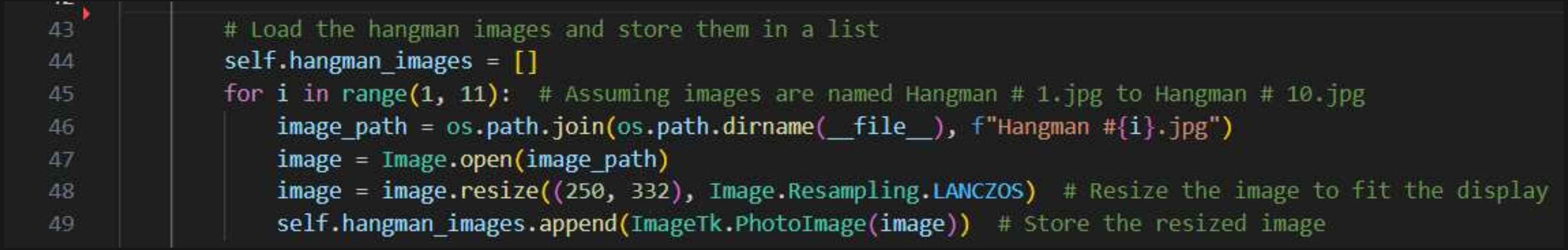


e



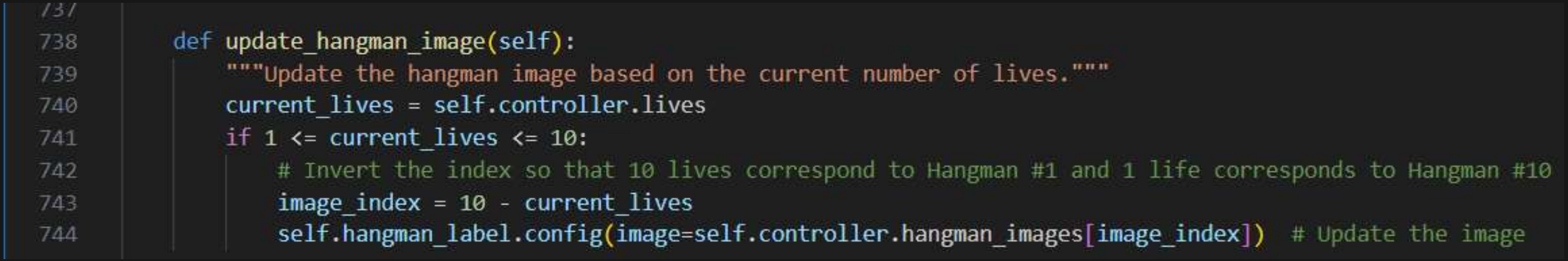
Loading of Images (Hangman stages)

Store the 10 images as list



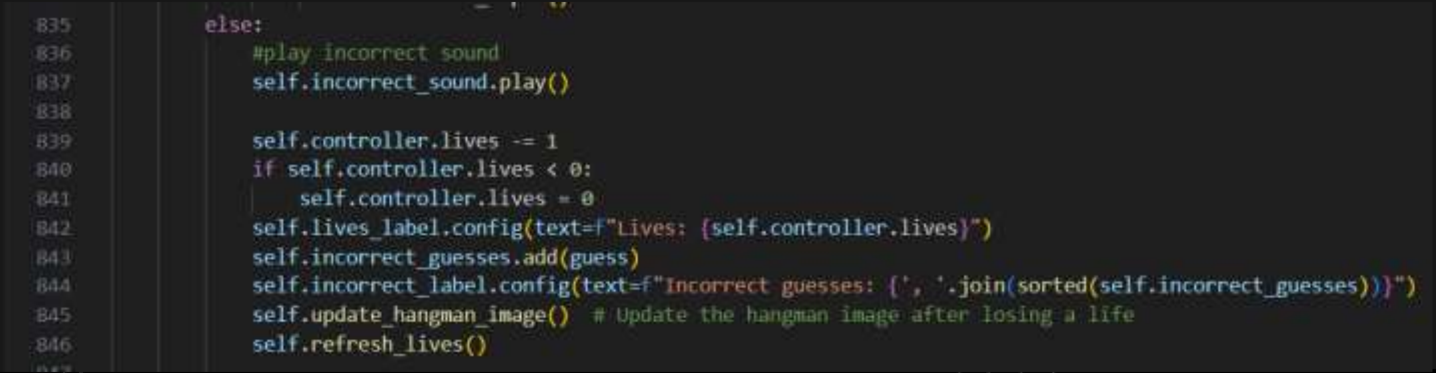
```python
        # Load the hangman images and store them in a list
        self.hangman_images = []
        for i in range(1, 11):  # Assuming images are named Hangman # 1.jpg to Hangman # 10.jpg
            image_path = os.path.join(os.path.dirname(__file__), f"Hangman #{i}.jpg")
            image = Image.open(image_path)
            image = image.resize((250, 332), Image.Resampling.LANCZOS)  # Resize the image to fit the display
            self.hangman_images.append(ImageTk.PhotoImage(image))  # Store the resized image
```

Function to update image



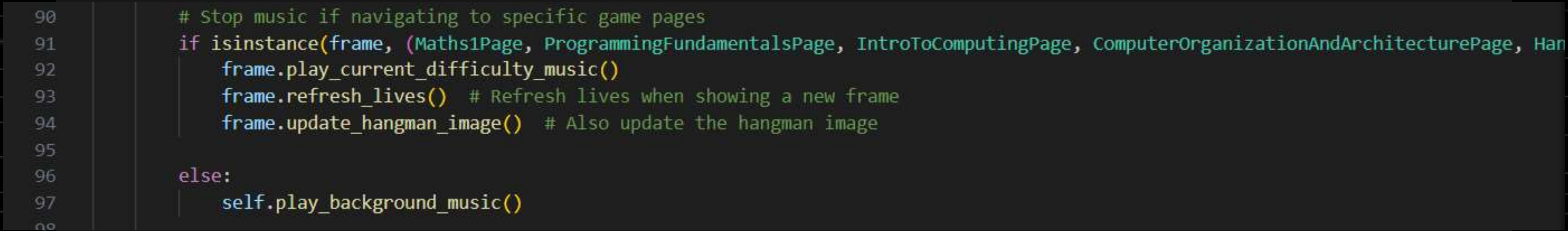
```python
    def update_hangman_image(self):
        """Update the hangman image based on the current number of lives."""
        current_lives = self.controller.lives
        if 1 <= current_lives <= 10:
            # Invert the index so that 10 lives correspond to Hangman #1 and 1 life corresponds to Hangman #10
            image_index = 10 - current_lives
            self.hangman_label.config(image=self.controller.hangman_images[image_index])  # Update the image
```

Calling the function



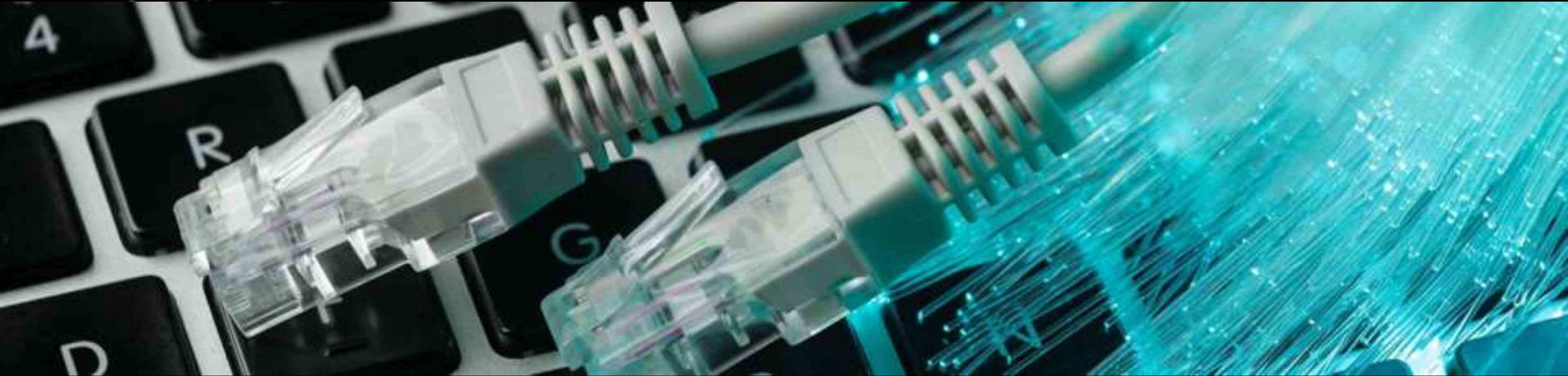
```python
        else:
            #play incorrect sound
            self.incorrect_sound.play()

            self.controller.lives -= 1
            if self.controller.lives < 0:
                self.controller.lives = 0
            self.lives_label.config(text=f"Lives: {self.controller.lives}")
            self.incorrect_guesses.add(guess)
            self.incorrect_label.config(text=f"Incorrect guesses: {', '.join(sorted(self.incorrect_guesses))}")
            self.update_hangman_image()  # Update the hangman image after losing a life
            self.refresh_lives()
```

Hangman image is always in line with number of lives left



```
        # Stop music if navigating to specific game pages
        if isinstance(frame, (Maths1Page, ProgrammingFundamentalsPage, IntroToComputingPage, ComputerOrganizationAndArchitecturePage, Han
            frame.play_current_difficulty_music()
            frame.refresh_lives()  # Refresh lives when showing a new frame
            frame.update_hangman_image()  # Also update the hangman image

        else:
            self.play_background_music()
```

09 Music



Music Implementation

Use music for

1. Background Music @ Start page
2. Correct/Incorrect Answer
3. Easy Level
4. Medium Level
5. Hard Level

Overview

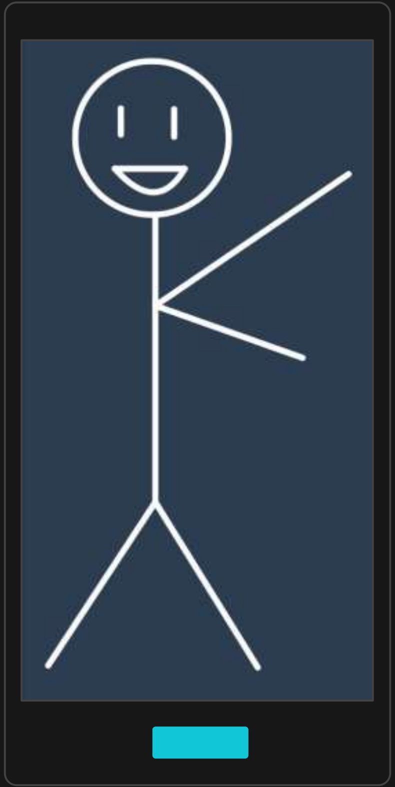
- Overview: The Hangman game incorporates background music that changes depending on the selected difficulty level (Easy, Medium, or Hard). This music enhances the gaming experience by providing an immersive atmosphere.

- How the Music Works:
  Loading Music: The game uses the pygame library to load and play music. The paths for the music files are set up when the game starts.

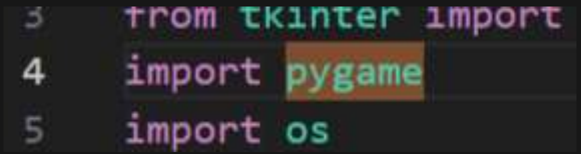
  Playing Music: Once the difficulty level is chosen, the corresponding music file is loaded and played in a continuous loop. This is done using pygame.mixer.music.play().

  Stopping Music: If the player switches levels or restarts the game, any currently playing music is stopped before the new track starts.

  Volume Control: Players can adjust the music volume using an in-game slider. This is dynamically applied using pygame.mixer.music.set_volume().



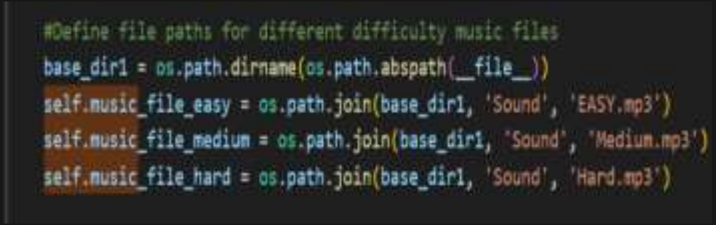
Code for Music Implementation



```
from tkinter import
import pygame
import os
```

Importing pygame module

To handle the audio in the game, the pygame library is used. This module includes functionality for loading and playing sound files.

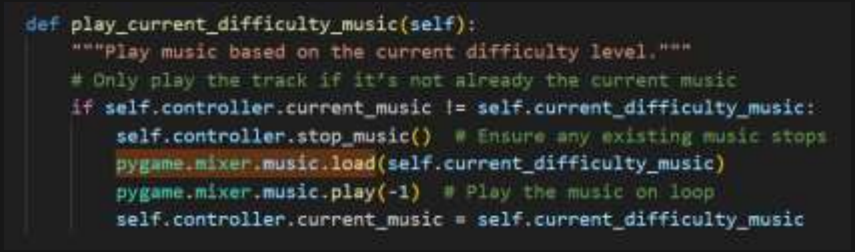


```
#Define file paths for different difficulty music files
base_dir1 = os.path.dirname(os.path.abspath(__file__))
self.music_file_easy = os.path.join(base_dir1, 'Sound', 'EASY.mp3')
self.music_file_medium = os.path.join(base_dir1, 'Sound', 'Medium.mp3')
self.music_file_hard = os.path.join(base_dir1, 'Sound', 'Hard.mp3')
```

Loading music files for different difficulty level

Different music files are loaded based on the difficulty level selected in the game. The paths for the music files are set as follows:

Code for Music Implementation



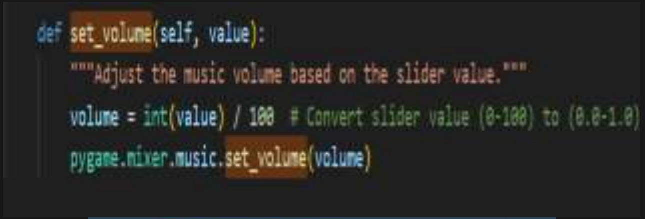
```python
def play_current_difficulty_music(self):
    """Play music based on the current difficulty level."""
    # Only play the track if it's not already the current music
    if self.controller.current_music != self.current_difficulty_music:
        self.controller.stop_music()  # Ensure any existing music stops
        pygame.mixer.music.load(self.current_difficulty_music)
        pygame.mixer.music.play(-1)  # Play the music on loop
        self.controller.current_music = self.current_difficulty_music
```

Playing music base on difficulty

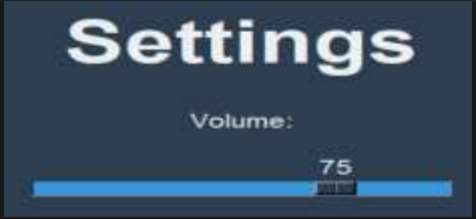
When the game starts, the music corresponding to the selected difficulty level is played. Here's how the music is played using pygame.mixer.music

The pygame.mixer.music.load() function loads the appropriate music file based on the difficulty level.

The pygame.mixer.music.play(-1) ensures that the music loops continuously during the game.



```python
def set_volume(self, value):
    """Adjust the music volume based on the slider value."""
    volume = int(value) / 100  # Convert slider value (0-100) to (0.0-1.0)
    pygame.mixer.music.set_volume(volume)
```



Settings

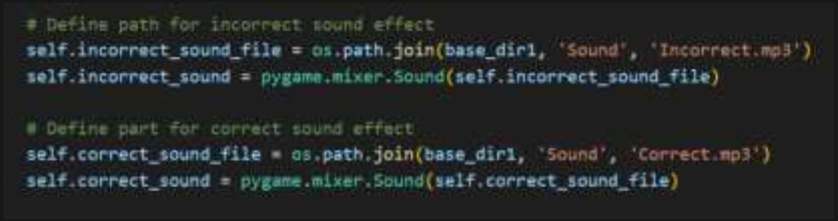
Volume:

75

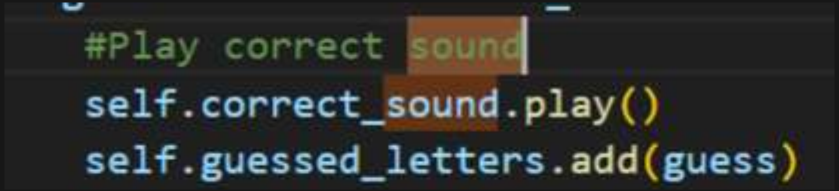
Adjusting of the volume

The game allows for volume adjustment through a slider, and the volume is set by converting the slider value (from 0 to 100) into a range from 0.0 to 1.0:

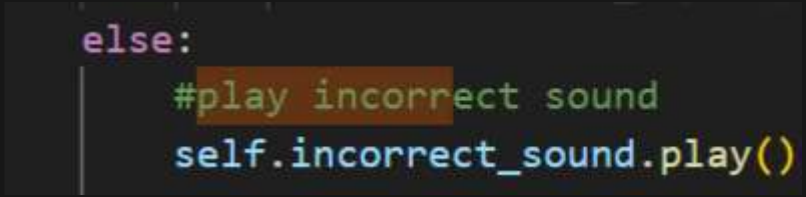
Code for Music Implementation



```
# Define path for incorrect sound effect
self.incorrect_sound_file = os.path.join(base_dir1, 'Sound', 'Incorrect.mp3')
self.incorrect_sound = pygame.mixer.Sound(self.incorrect_sound_file)

# Define part for correct sound effect
self.correct_sound_file = os.path.join(base_dir1, 'Sound', 'Correct.mp3')
self.correct_sound = pygame.mixer.Sound(self.correct_sound_file)
```



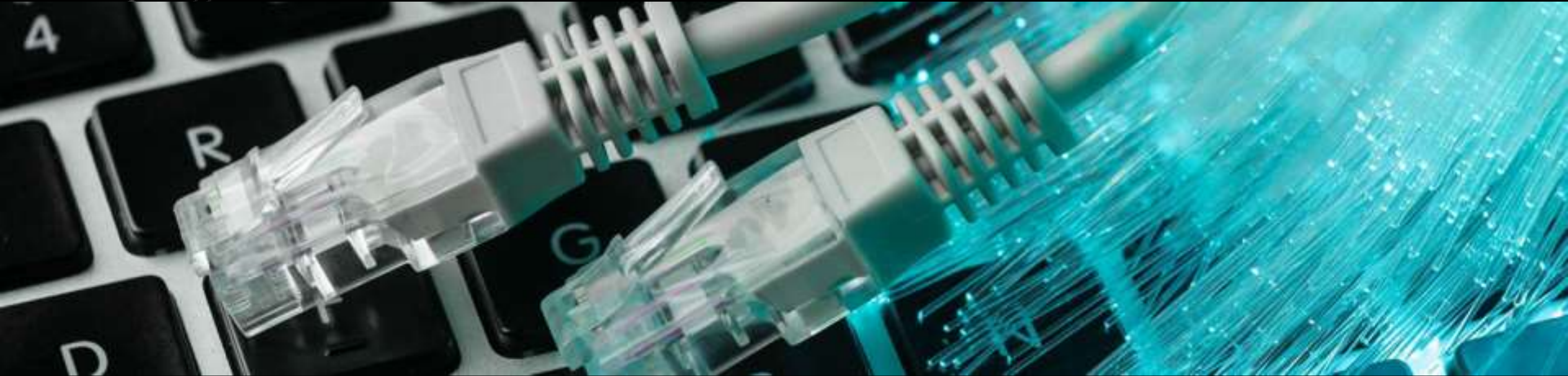
```
#Play correct sound
self.correct_sound.play()
self.guessed_letters.add(guess)
```



```
else:
    #play incorrect sound
    self.incorrect_sound.play()
```

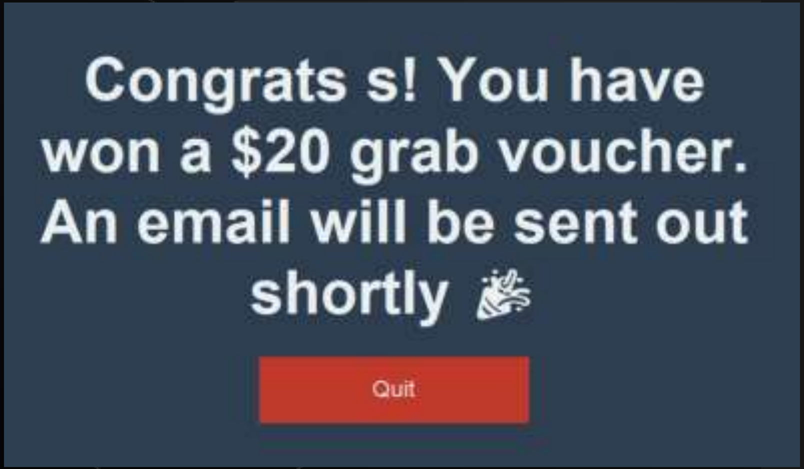
Playing music when correct/incorrect answer

10 Completion

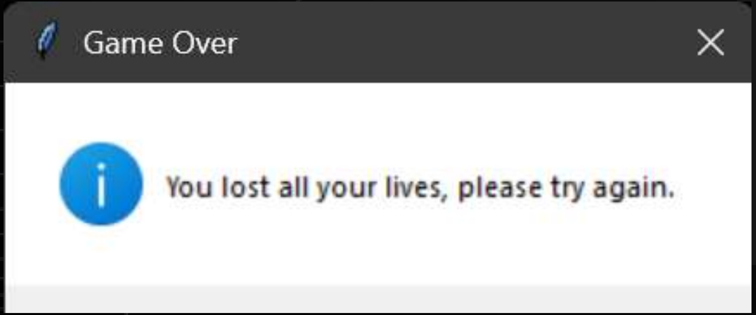


Completion Page

Player s



Congrats s! You have won a $20 grab voucher. An email will be sent out shortly 🎉



Game Over

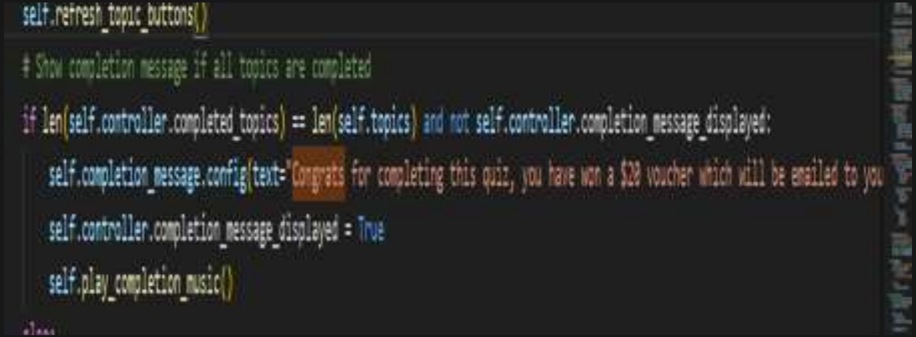
You lost all your lives, please try again.

1. Completion Message:
   When the player successfully completes all the topics, a message congratulates them, and rewards like a voucher may be offered (in this case, just part of the code).

2. Completion Music:
   Once the game is completed, special music plays to signify the player's success.

3. Game Over Logic:
   If the player loses, the game shows a "Game Over" message and plays the completion music, encouraging them to try again.

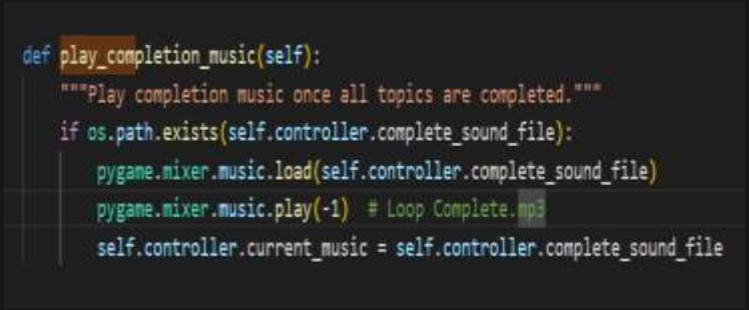
Code for Completion Page



```
self.refresh_topic_buttons()

# Show completion message if all topics are completed
if len(self.controller.completed_topics) == len(self.topics) and not self.controller.completion_message_displayed:
    self.completion_message.config(text="Congrats for completing this quiz, you have won a $20 voucher which will be emailed to you
    self.controller.completion_message_displayed = True
    self.play_completion_music()
else:
```

Completion Message Display

When all the topics in the game are completed, the system will display a message saying "Congrats for completing this quiz, you have won a $20 voucher...". This is likely the final message shown to the player upon successful completion of the game.

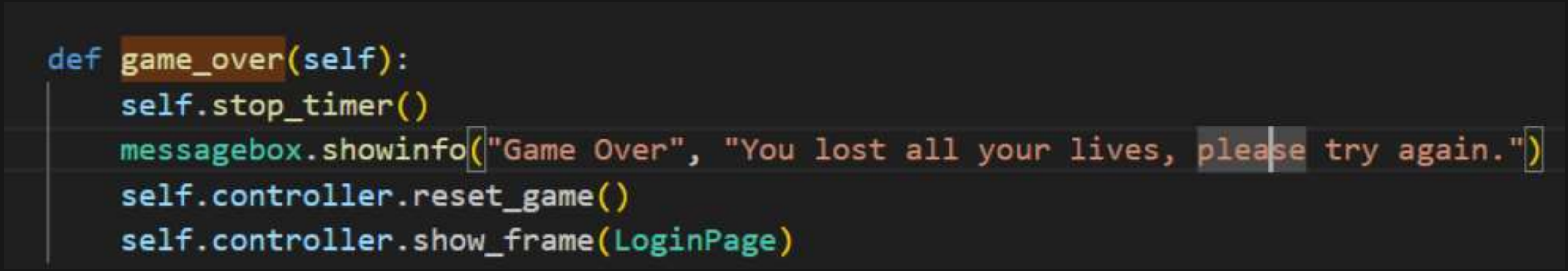


```
def play_completion_music(self):
    """Play completion music once all topics are completed."""
    if os.path.exists(self.controller.complete_sound_file):
        pygame.mixer.music.load(self.controller.complete_sound_file)
        pygame.mixer.music.play(-1)  # Loop Complete.mp3
        self.controller.current_music = self.controller.complete_sound_file
```

Completion Music

This function plays completion music when the game is finished. It loads the music file and starts playing it to enhance the feeling of victory.

Code for Completion Page



```python
def game_over(self):
    self.stop_timer()
    messagebox.showinfo("Game Over", "You lost all your lives, please try again.")
    self.controller.reset_game()
    self.controller.show_frame(LoginPage)
```

Game Over

If the player loses the game (i.e., they run out of lives), a message box pops up with a "Game Over" notification.

Conclusion

Learning Point 1

Sound and Music Integration: We successfully learned how to implement background music and sound effects using the pygame library, which enriched the game experience.

Learning Point 2

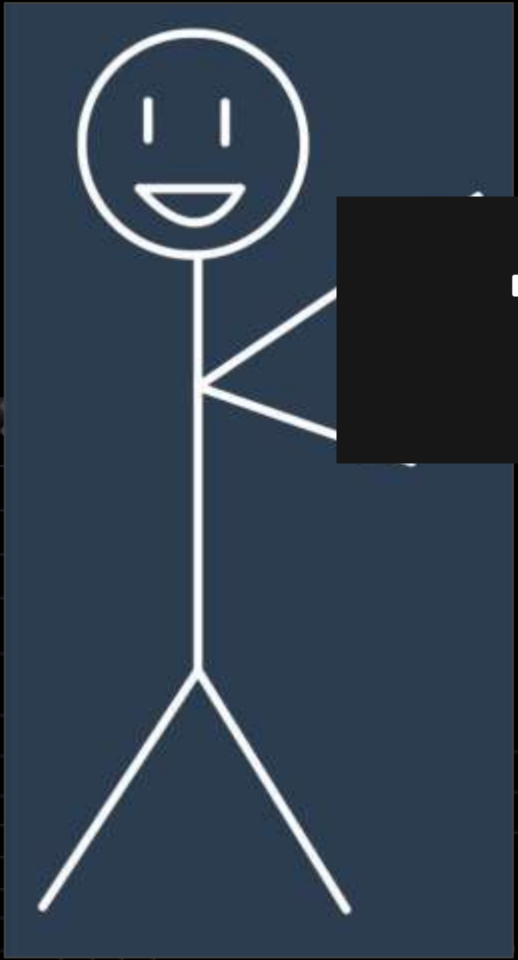
GUI Development with Tkinter: We gained hands-on experience in building a graphical user interface (GUI) using Tkinter, making the game more interactive and visually engaging.

Learning Point 3

Managing Game State: We learned how to manage different game states such as tracking player progress, handling win/loss conditions, and displaying completion messages.

Improvement

Leaderboard and Online Integration: Implement an online leaderboard where players can compare their scores globally. Adding online features, such as user profiles and global rankings, could make the game more competitive and engaging.



Thank You!