Quiz Game

Eliya Samary

Introduction

1. The project consists of 5 files:
   - **server.py** - implements a quiz game server (elaborated in the following sections).
   - **client.py** - defines a client for the quiz game (elaborated in the following sections).
   - **consts.py** – define constant variables that are shared across the application, ensuring consistency.
   - **quiz.py** - contains the questions of the quiz and the categories .
   - **quiz_photo.png** – an image utilized within the application's user interface.
2. Additionally, during execution, two new files, **server_log.log** and **client_log.log**, will be created to monitor server activities and errors. These logs will offer valuable insights for debugging and enhancing server performance.

Server.py

The `server.py` file implements a quiz game server that allows multiple clients to participate through a socket-based communication system. It manages game rooms, enabling users to create and join quizzes while handling commands for messaging, answering questions, and updating scores.

Explanation of Functions:

1. **send_message(m, c)**: Sends a message to the specified client connection (c). It encodes the message as bytes and logs the action.
2. **receive_message(c)**: Receives a message from the specified client connection (c). It decodes the received bytes into a string and logs the action. If an error occurs during reception, it logs the error and returns None.
3. **quiz_server(client)**: Main function to handle the quiz logic for a connected client. It listens for messages from the client, processes commands such as creating or joining rooms, and manages the game flow, including handling answers and participant updates.
4. **send_categories(client)**: Sends a list of available quiz categories to the specified client. It constructs a message containing the categories and calls send_message() to transmit it.
5. **close_connection(client)**: Closes the connection with the specified client and logs the action.
6. **generate_unique_room_pin()**: Generates a unique four-digit room PIN for new game rooms. It ensures that the PIN does not already exist in all_game_rooms and logs the generated PIN.

7. **create_room(message, client, room_pin)**: Creates a new game room based on the received message and assigns it a unique room PIN. It initializes the room with participants and scores, then informs the client of the room's creation.
8. **start_game(room_pin)**: Starts a new thread to initiate the game session for the specified room, allowing the game to proceed concurrently.
9. **handel_client_answer(room_pin, client_username, message)**: Processes a client's answer to a question, logging the action and delegating the task to handel_answer() to update scores based on the answer.
10. **update_participants(room_pin)**: Updates the list of participants in a game room and sends the updated list to all clients in that room.
11. **remove_participant_from_room(room_pin, client, room, client_username)**: Removes a participant from a game room, updating the participants list and scores accordingly.
12. **start_game_session(room_pin)**: Manages the overall game session for a specific room, asking questions to participants, handling answers, and updating scores until the game concludes.
13. **announce_final_scores(room_pin)**: Sends the final scores of all participants in the room to each client, logging the announcement.
14. **announce_question(room_pin, question, question_options)**: Sends the current quiz question and its options to all participants in the specified room.
15. **scores_handler(room_pin, correct_answer)**: Sends the current scores to each participant and informs them of the correct answer for the latest question.
16. **handel_answer(room_pin, client_username, answer, time_answer_took)**: Processes an individual participant's answer, calculating points based on correctness and response time, and updating the participant's score in the room.

17. **main()**: Initializes the server socket, listens for incoming connections, and starts a new thread for each connected client, allowing concurrent handling of multiple clients.

Client.py

The Client.py file defines a client application for a quiz game using a graphical user interface (GUI) built with Tkinter and socket programming for network communication. The QuizClient class manages the overall functionality of the client, including connecting to a server, handling user interactions, and processing quiz data. It provides an interface for players to join an existing quiz room or create a new one by entering a room pin. The client also supports real-time updates, such as displaying quiz questions, managing player scores, and notifying users of game status changes. Logging functionality is incorporated to track activities and errors, facilitating debugging and monitoring.

The **QuizClient class** initializes the client application, setting up the main user interface (UI) and establishing a socket connection to the quiz server. The UI components include labels, entry fields, and buttons for user interaction, such as joining a room or creating a new game.

Class QuizClient:

- **__init__(self)**: Initializes the main application window and establishes a connection to the quiz server. It sets up logging and displays the UI window for the client.
- **center_ui_window(self)**: Centers the application window on the screen by calculating the appropriate coordinates based on screen dimensions.

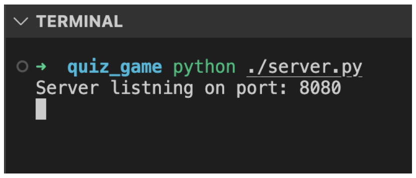
- **display_ui_window(self)**: Constructs and displays the initial UI components, including labels, buttons, and entry fields for room interaction. This includes options to join an existing room or create a new one.
- **create_new_room(self)**: Initiates the process of creating a new quiz room by starting a separate thread to handle the room creation.
- **send_message(self, m)**: Sends a message to the server, logging the action for debugging purposes.
- **receive_message(self)**: Receives messages from the server, decoding and logging them. It handles exceptions that may occur during message reception.
- **create_room(self)**: Sends a request to create a new quiz room and processes the server's response, prompting the user to choose a category if successful.
- **room_joining(self)**: Starts a thread for the process of joining an existing quiz room, allowing the user to enter a room pin.
- **show_error(self, m)**: Displays an error message in the UI, updating the error label with the specified message.
- **join_room(self)**: Validates the room pin entered by the user, sends a join request to the server, and handles the server's response, including error messages.
- **choose_category(self, all_categories)**: Creates a new window for the user to select a quiz category. It populates the window with radio buttons representing available categories.
- **start_after_choose_category(self, choose_category_frame)**: Sends the selected category to the server to create a new room, destroying the category selection window afterward.

- **enter_game(self)**: Initiates the game entry process, prompting the user to fill in their username and starting the quiz game.
- **fill_username(self)**: Opens a new window for the user to enter their username before starting the quiz.
- **submit_username(self)**: Sends the entered username to the server and closes the username entry window.
- **start_quiz_game(self, pin: str = None)**: Prepares the UI for the quiz game, displaying necessary information and initializing score tracking. It starts a separate thread to handle incoming messages from the server.
- **start_quiz_btn(self)**: Creates and displays a button to start the quiz, linking it to the start_quiz method.
- **start_quiz(self)**: Sends a message to the server to begin the quiz and hides the start button.
- **close_client(self)**: Logs out from the game and closes the client application, sending an exit message to the server.
- **center_window(self, window)**: Centers a given window relative to the main application window.
- **center_window_up(self, window, offset_n=50)**: Centers a window above the main application window, with an optional vertical offset.
- **client_handler(self)**: Continuously listens for messages from the server and processes them based on their type, such as displaying quiz questions, updating scores, and handling game results.
- **present_quiz_question(self, data)**: Displays the quiz question and its possible answers to the user, updating the UI elements accordingly.

Instructions and examples

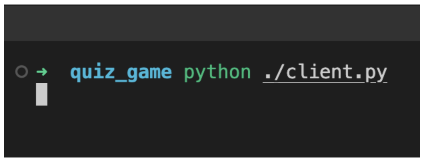
Game Instructions:

1. Ensure that Python is installed and updated on your machine, and that the required libraries are installed.
2. Open a terminal and navigate to the correct directory. First, run the `server.py` file.



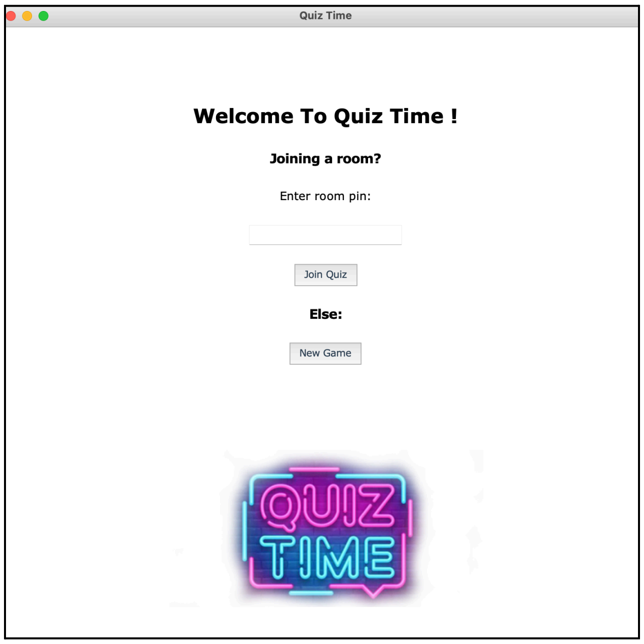
```
TERMINAL

○ → quiz_game python ./server.py
Server listning on port: 8080
```

3. Once you see the message indicating that the server is "listening on port: XXXX," you can distribute the `client.py` file to each participant in the game.



```
○ → quiz_game python ./client.py
```

4. It will open the game. On the main screen, you will have two options: create a new game "**New Game**" and join an existing game by room pin "**Join Quiz**".



Quiz Time

Welcome To Quiz Time !

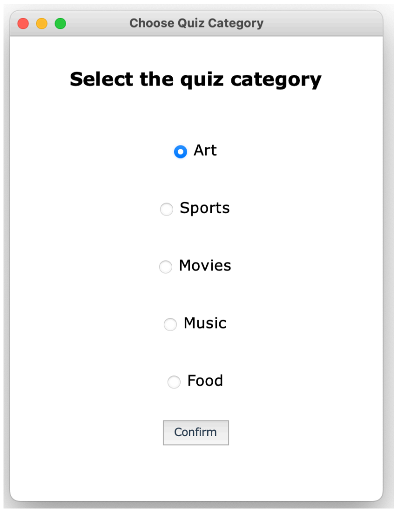
Joining a room?

Enter room pin:

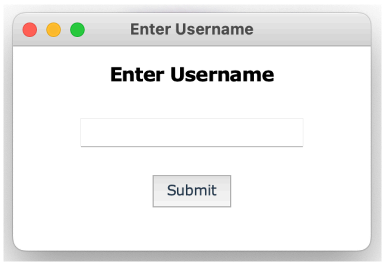
Else:

To Create a New Game- New Game Option:

- Select the quiz category.



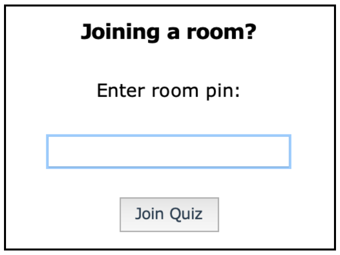
- Enter your username, which will be displayed on the participant list and the scoreboard.



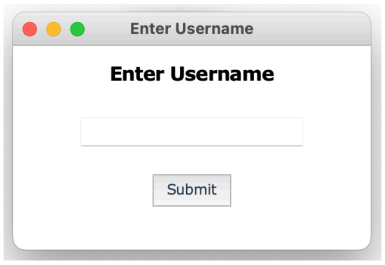
- Wait for other players to join.
- When you are ready to start the quiz, press the **Start** button.

To Join an Existing Game - Join Quiz Option:

- Enter the room PIN provided by the game host.

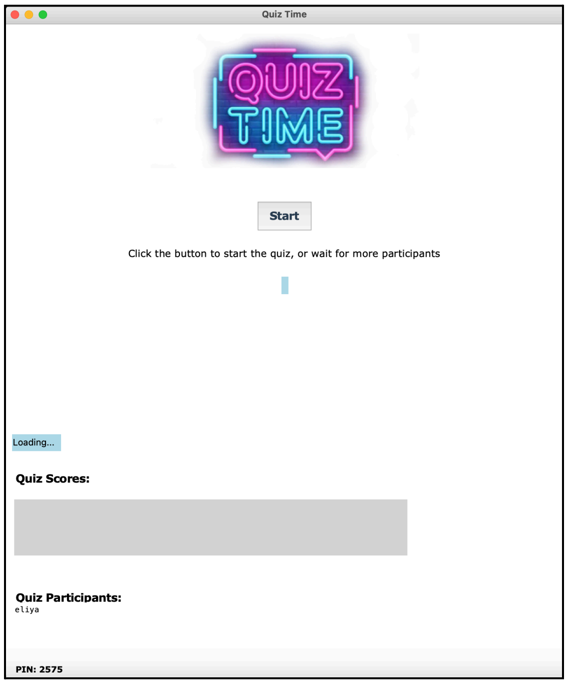


- Enter your username, which will be displayed on the participant list and the scoreboard.



- After joining, your name will be added to the participant list.
- When ready, press the **Start** button to begin the quiz.

UI display:



Click the button to start the quiz, or wait for more participants

Loading...

Quiz Scores:

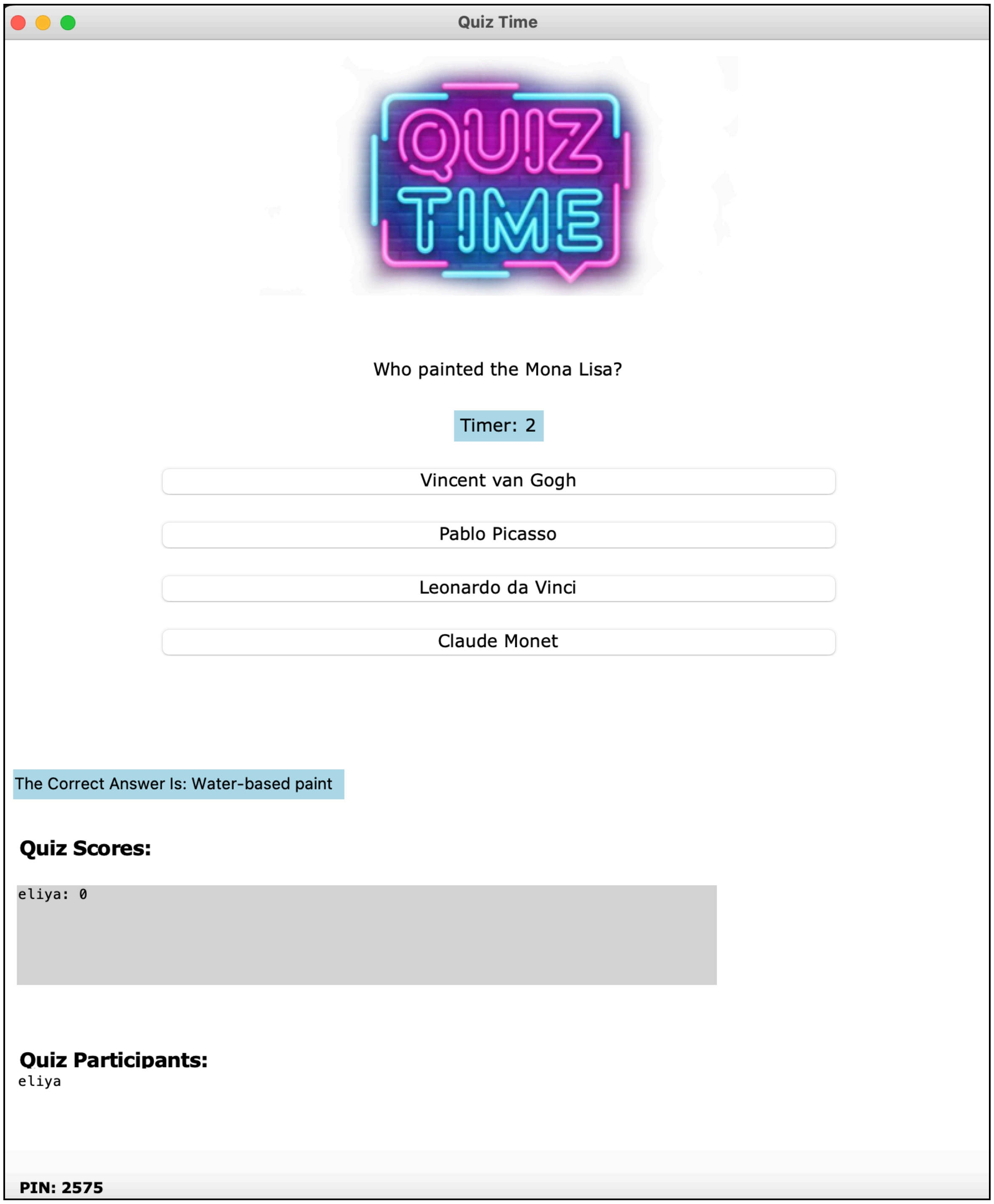
Quiz Participants:
eliya

PIN: 2575

Starting the Game:

Clicking the Start button initiates the game, and the following will be displayed on the board:

- A timer
- The current question
- Possible answers to the question
- A participant list
- The score board
- The room PIN



Who painted the Mona Lisa?

Timer: 2

Vincent van Gogh

Pablo Picasso

Leonardo da Vinci

Claude Monet

The Correct Answer Is: Water-based paint

Quiz Scores:

```
eliya: 0
```

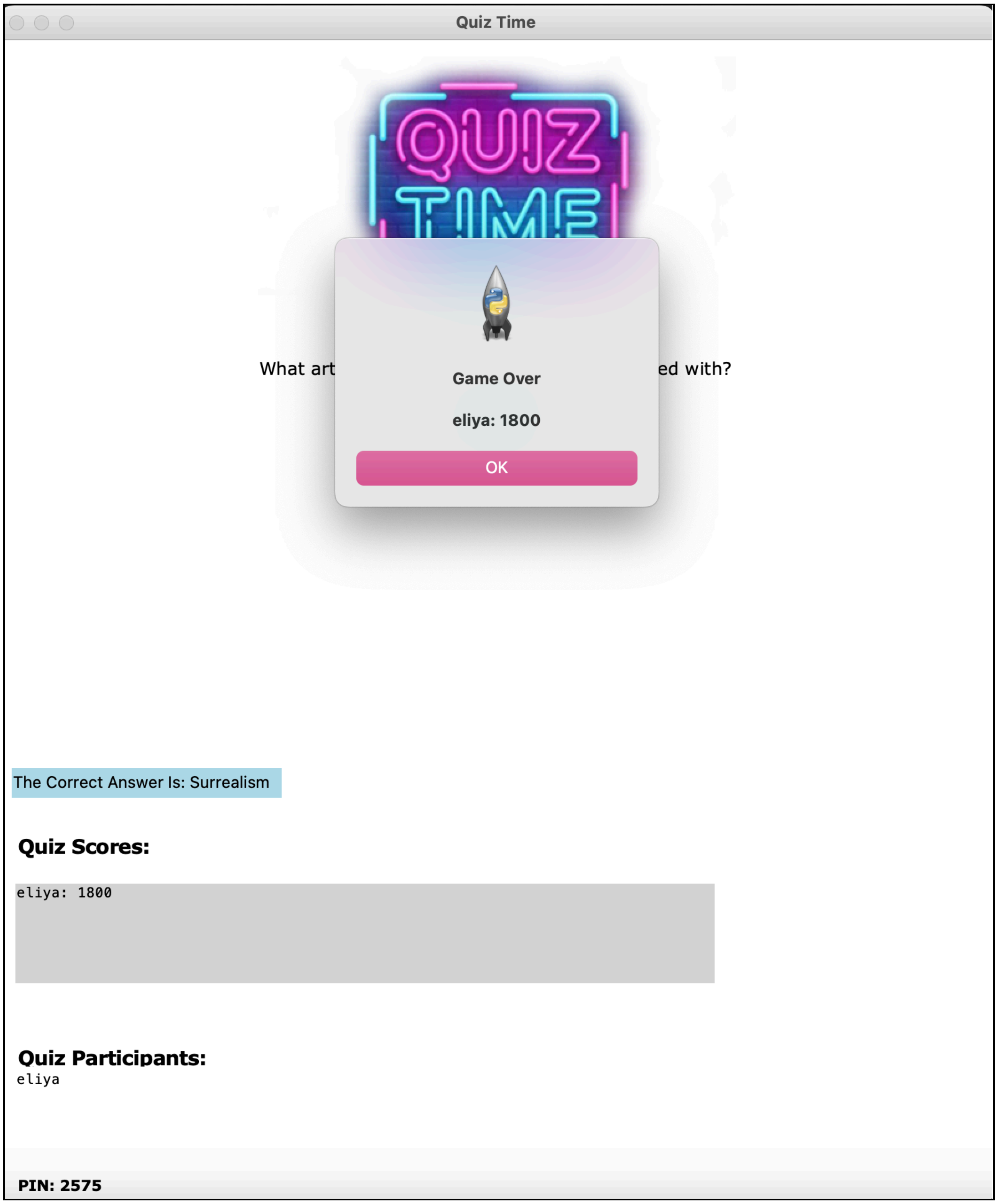
Quiz Participants:

```
eliya
```

PIN: 2575

All elements will be dynamically updated throughout the game.

The game will conclude after **five rounds**. At the end, the **final scoreboard** will be displayed, showcasing the participants' scores:



What art

ed with?

Game Over

eliya: 1800

The Correct Answer Is: Surrealism

Quiz Scores:

```
eliya: 1800
```

Quiz Participants:

```
eliya
```

PIN: 2575