QuickQuizzer - Project Documentation

Tatiya Seehatrakul st124875

November, 2024

1 Project Objectives

The objective of this project is to create an interactive terminal-based quiz game application, QuickQuizzer, which supports two game modes: **Math Quiz** and **Hangman**. The application is designed to:

- Provide a fun, educational experience with real-time feedback for players.
- Allow users to choose between two distinct game modes, each with unique challenges.
- Measure user response time and accuracy in answering math questions or guessing letters.
- Provide a user-friendly interface with clear instructions and positive feedback for correct actions.

This application uses the **TCP** transport layer for reliable data transfer between the client and server. TCP ensures that each message (question, answer, score, etc.) is delivered in the correct order and without loss, which is essential for maintaining the flow of the quiz game. This reliability is critical for games where players expect accurate and sequential interaction with the server.

2 Source Code

The source code for this project can be found at the following link:
`https://github.com/werrnnnwerrrnnnnnn/Quick-Quizzer`

3 Application-Layer Protocol Design

The application-layer protocol is custom-designed for QuickQuizzer to handle communication between the client and server. It includes a set of request and response actions that manage user mode selection, question handling, answer evaluation, and score tracking.

3.1 Math Quiz Mode - Custom Protocol

Status Code	Status Phrase	Description
100	You've Got This!	Response when a level is selected, encouraging the player.
101	Score Updated	Sent after each answer to update the player's score.
200	Nailed It!	Sent for a correct answer.
300	Question Incoming	Sent before sending each new question.
400	Oops! Wrong Format	For unexpected characters (e.g., letters when a number is expected).
401	Only Numbers Allowed!	For alphabetic characters in numeric-only answers.
404	Try Again!	For incorrect answers.
410	Quiz Complete! Thanks for Playing!	Sent after the last question is answered or the quiz ends.
411	Goodbye! Thanks for playing!	Sent when the user exits the game.

3.2 Hangman Mode - Custom Protocol

Status Code	Status Phrase	Description
100	Ready, Set, Guess!	Initial message when the game starts.
200	Nice Choice!	For a correct guess.
202	Already Tried That!	When a letter has already been guessed.
400	No Digits Allowed	When a number is entered instead of a letter.
401	Special Characters Not Allowed	For special character inputs.
404	Wrong Guess	For an incorrect guess.
410	You Win! The Word Was "..."	For completing the word successfully.
411	Game Over - The Word Was "..."	For exhausting all attempts without guessing the word.

4 Implementation and Code Flow

The QuickQuizzer game consists of two main Python scripts: `quiz_server.py` and `quiz_client.py`.

4.1 Server Code Overview

The server:

- Listens for incoming connections from clients.
- Prompts the client to select a mode:"math" for Math Quiz mode or "hangman" for Hangman mode.
- Based on the selected mode, either starts the Math Quiz or Hangman game by sending relevant instructions and questions to the client.
- For Math Quiz, the server sends a series of math questions and evaluates each answer, providing feedback on accuracy and tracking the player's score and response time.
- For Hangman, the server randomly selects a word and interacts with the client as they guess letters, keeping track of attempts left and displaying partial progress.
- Upon completion or exit, sends a summary and final score, then closes the connection.

4.2 Client Code Overview

The client:

- Connects to the server and selects a game mode.
- Receives instructions and questions, then interacts with the server by sending answers or guesses.
- Receives feedback on each response, including scores, time taken per question, and whether the answer was correct.
- At the end of the game, displays a summary with the final score and exits the connection.

5 Example Code Explanation

Below is a simple breakdown of key code segments from the client and server files:

```
# Server code segment to handle client connection
def handle_client(client_socket, client_id):
    mode = client_socket.recv(1024).decode().strip().lower()
    if mode == "math":
        handle_math_quiz(client_socket, client_id)
    elif mode == "hangman":
        handle_hangman(client_socket, client_id)
    elif mode == "quit":
        client_socket.send("STATUS:411 Goodbye! Thanks for playing!\n".encode())
        client_socket.close()
    else:
        client_socket.send("STATUS:400 Oops! Wrong Format\n".encode())
        client_socket.close()

# Client code segment to start interaction
mode = input("Choose mode (math/hangman): ").strip().lower()
client_socket.send(f"{mode}\n".encode())
```

6 Transport Layer Choice: TCP

This project uses the **TCP** protocol because:

- **Reliability**: TCP ensures that each message is delivered accurately and in the correct sequence, which is essential for a game where questions must appear in order and each response needs to be acknowledged.
- **Error-checking and Retransmission**: TCP provides error-checking and retransmission, which helps in maintaining a smooth and uninterrupted game experience, as opposed to UDP, where packet loss could interrupt gameplay.
- **Connection-oriented Communication**: The game requires a constant connection between client and server to facilitate interactive gameplay, which TCP effectively supports.

7 Conclusion

The QuickQuizzer project successfully demonstrates a network application using custom protocols over a TCP connection. This project highlights key concepts in socket programming, including real-time communication, protocol design, and client-server interaction, making it an engaging and educational experience for players.