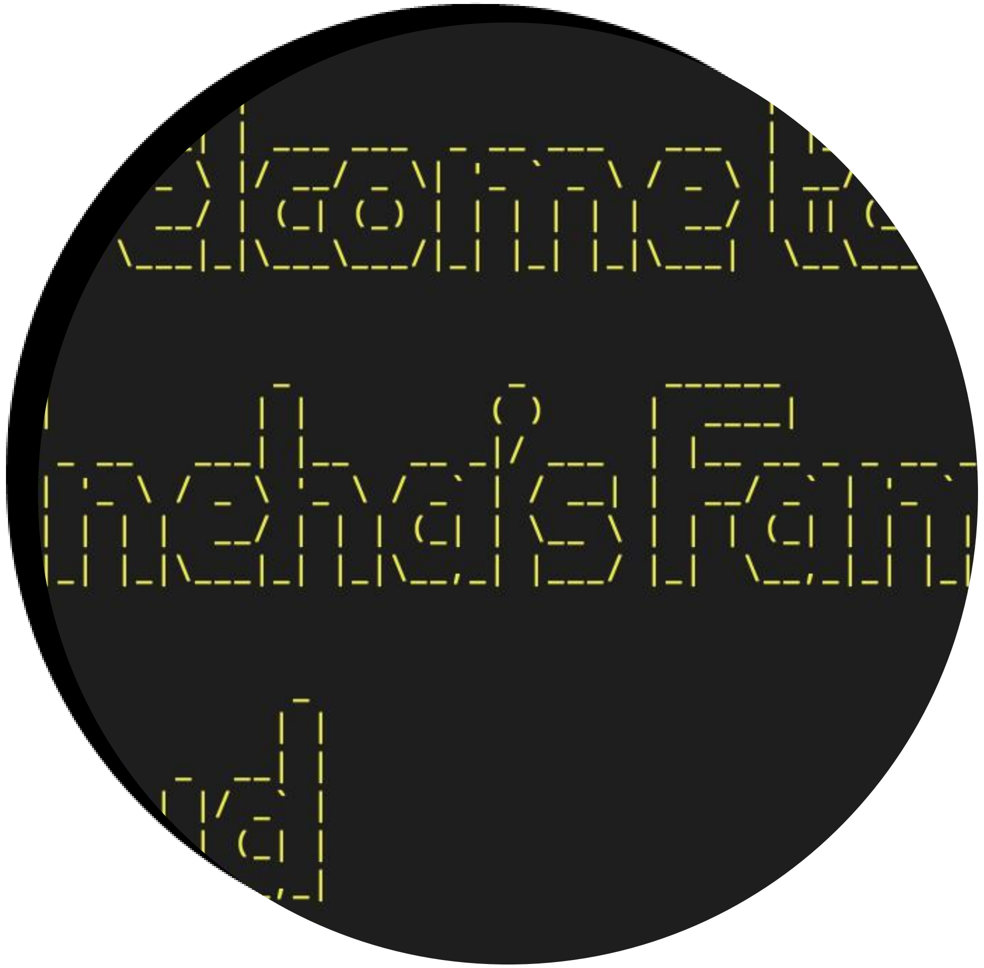
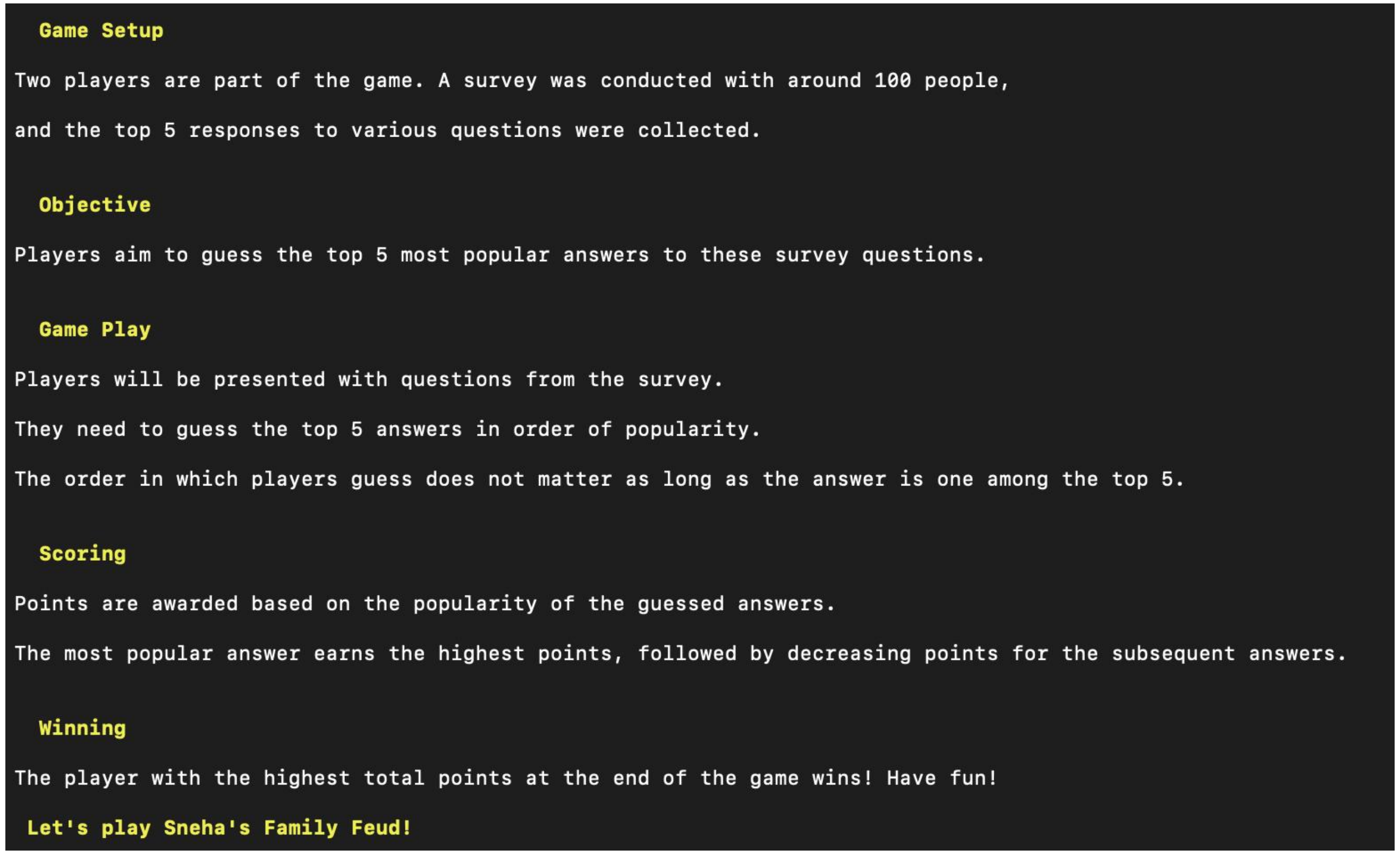


FAMILY FEUD GAME

SNEHA IRUKUVAJJULA

CSCI 5020



```
  Game Setup

Two players are part of the game. A survey was conducted with around 100 people,

and the top 5 responses to various questions were collected.

  Objective

Players aim to guess the top 5 most popular answers to these survey questions.

  Game Play

Players will be presented with questions from the survey.

They need to guess the top 5 answers in order of popularity.

The order in which players guess does not matter as long as the answer is one among the top 5.

  Scoring

Points are awarded based on the popularity of the guessed answers.

The most popular answer earns the highest points, followed by decreasing points for the subsequent answers.

  Winning

The player with the highest total points at the end of the game wins! Have fun!

 Let's play Sneha's Family Feud!
```

GAME PLAY

- The Family Feud Game Server is designed to facilitate a multiplayer Family Feud game where two players compete to guess the top answers to survey questions.

WHAT IS THE PROBLEM?

Software developers
like to solve problems.

If there are
no problems handily available

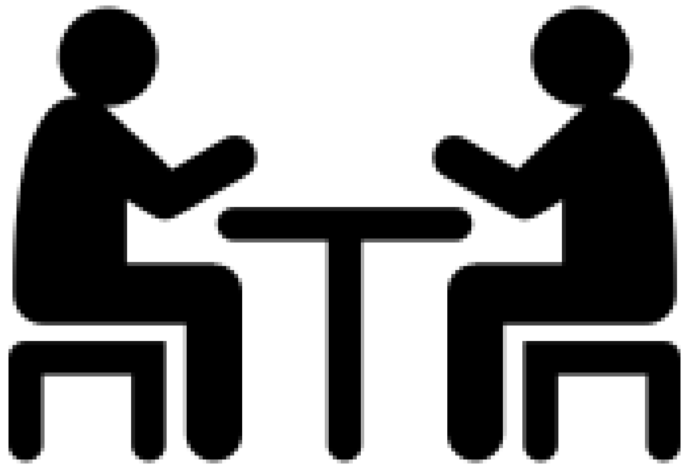
they will create **their own problems**!

- It's a TV show with Steve Harvey hosting the show. Can I play with my own family?
- Board game. But paid. Also, requires a minimum of 3 players (*1 host and 2 players*).
- Online version (*arkadium.com*) does not have the option to play with friends. Randomly chooses a player.

Can I automate this in a way where 2 players get a question at the exact same time?

THE CHALLENGE

- Two players need to get a question EXACTLY at the same time.
- Clients running a python code individually on their own machines? Possibility of human errors



CODE ARCHITECTURE

1. Server Setup

The server runs as a system service named "*family_feud.service*" on an AWS EC2 instance. It listens for incoming connections on port 5020.

2. Communication

Sockets are utilized for communication between the server and clients. Clients connect to the server via telnet.

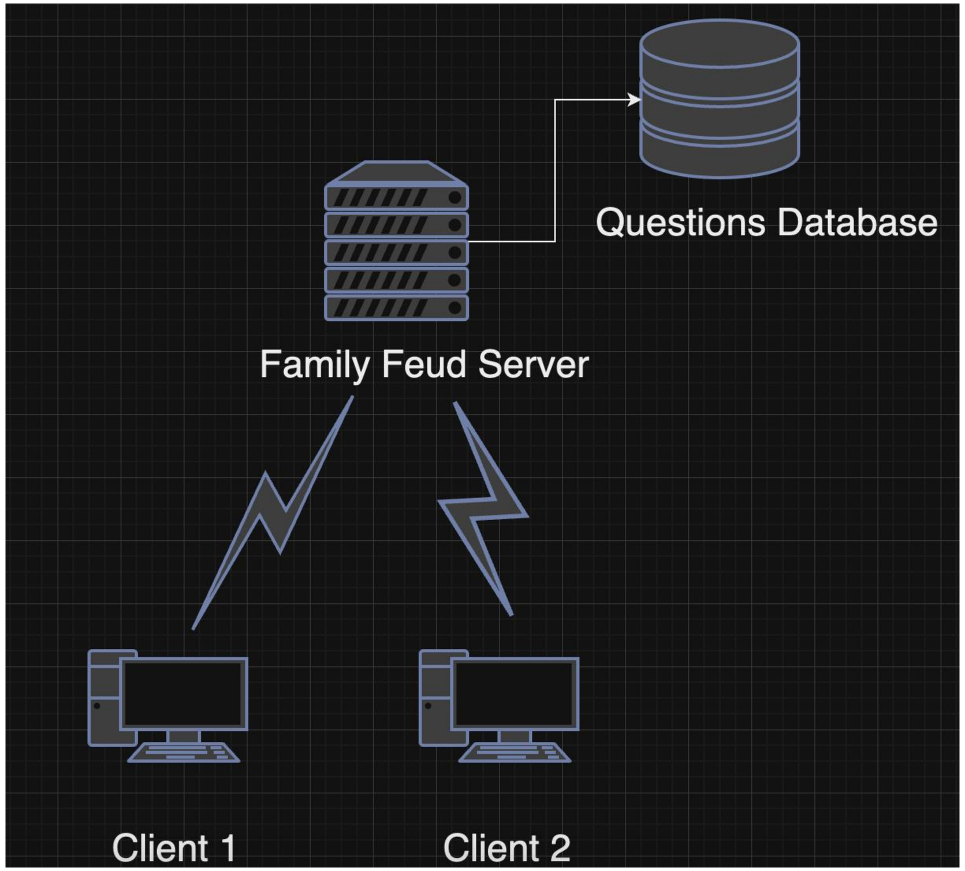
3. Database Integration

A MySQL database stores questions, answers, and corresponding point values.

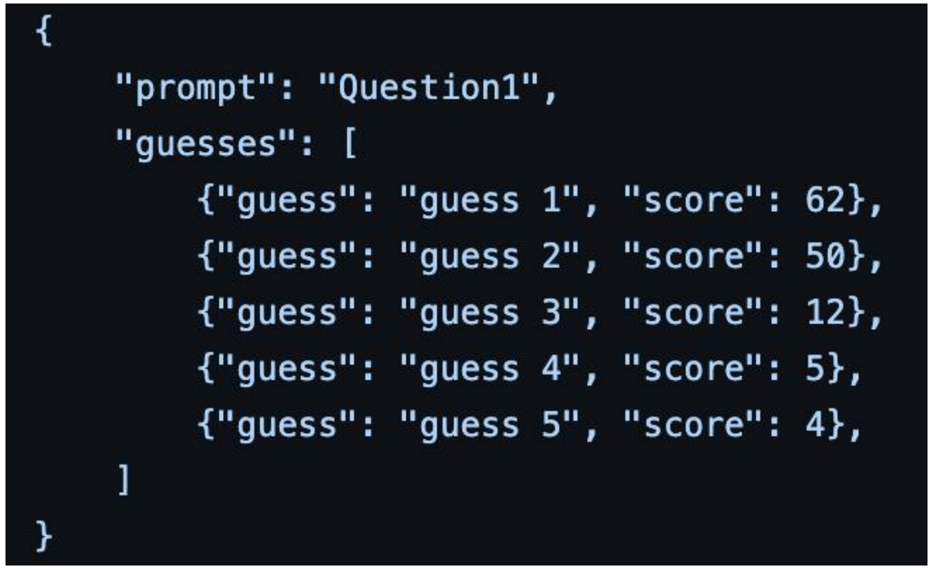
The server connects to the database to retrieve and manage questions during gameplay.

4. Multi-Threading

Each connected client is handled in a separate thread, allowing the server to manage multiple connections concurrently.



CODE ARCHITECTURE (CONTINUED..)



```
{
    "prompt": "Question1",
    "guesses": [
        {"guess": "guess 1", "score": 62},
        {"guess": "guess 2", "score": 50},
        {"guess": "guess 3", "score": 12},
        {"guess": "guess 4", "score": 5},
        {"guess": "guess 5", "score": 4},
    ]
}
```

5. Question Selection

Questions are selected from the database. The server ensures questions are not reused until the entire set has been exhausted.

6. Scoring Mechanism

The scoring mechanism is based on the player's guesses compared to the survey's top answers. Retrieves the points from the SQL database

7. Continuous Integration and Deployment

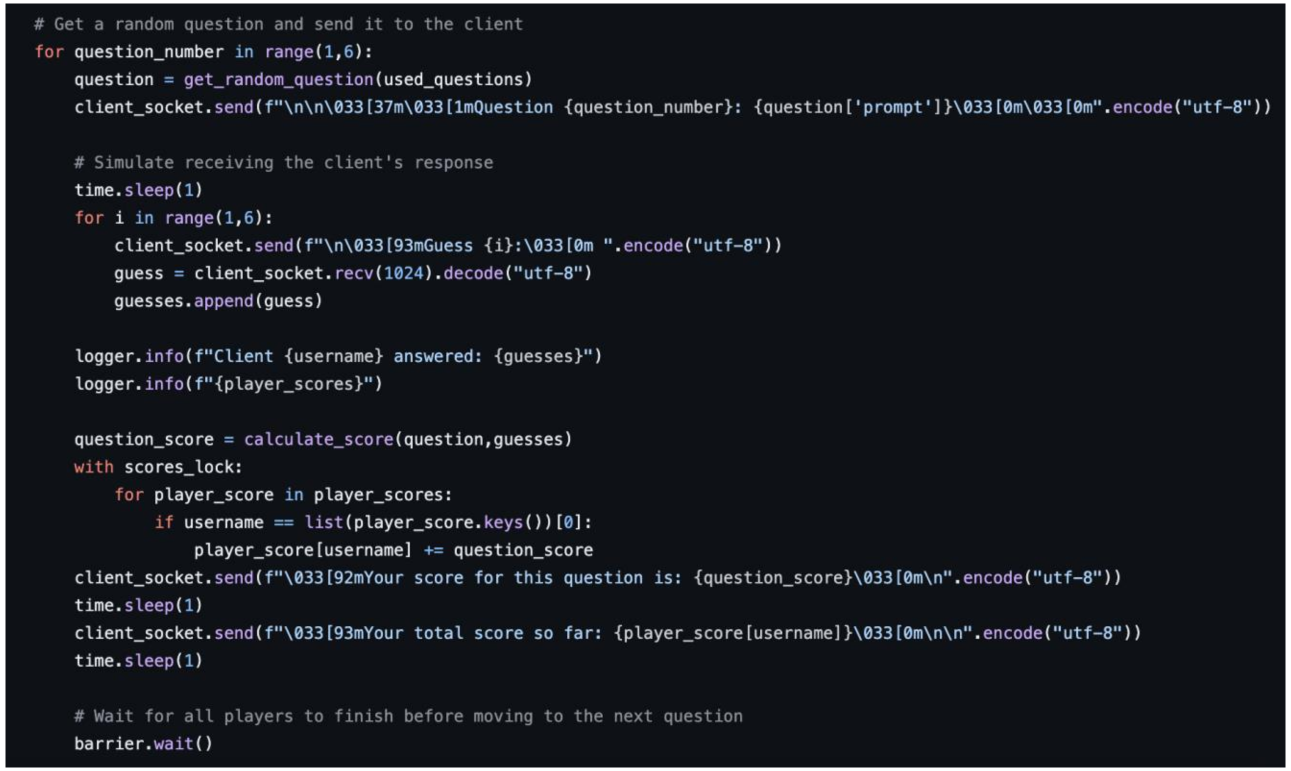
The server code is set up as a service, ensuring it remains "always-on". Automatic updates are pulled from my Git repository every minute using a cron job, keeping the game up-to-date with the latest changes.

CORE FUNCTIONALITY

- The question variable holds a dictionary representing the selected question, the top 5 most popular answers and points associated to each of these answers.

- The client is prompted to provide five guesses (Guess 1 to Guess 5), and each guess is collected in the guesses list.

- player_scores is a list of dictionaries with player usernames and scores.



```python
# Get a random question and send it to the client
for question_number in range(1,6):
    question = get_random_question(used_questions)
    client_socket.send(f"\n\n\033[37m\033[1mQuestion {question_number}: {question['prompt']}\033[0m\033[0m".encode("utf-8"))

    # Simulate receiving the client's response
    time.sleep(1)
    for i in range(1,6):
        client_socket.send(f"\n\033[93mGuess {i}:\033[0m ".encode("utf-8"))
        guess = client_socket.recv(1024).decode("utf-8")
        guesses.append(guess)

    logger.info(f"Client {username} answered: {guesses}")
    logger.info(f"{player_scores}")

    question_score = calculate_score(question,guesses)
    with scores_lock:
        for player_score in player_scores:
            if username == list(player_score.keys())[0]:
                player_score[username] += question_score
    client_socket.send(f"\033[92mYour score for this question is: {question_score}\033[0m\n".encode("utf-8"))
    time.sleep(1)
    client_socket.send(f"\033[93mYour total score so far: {player_score[username]}\033[0m\n\n".encode("utf-8"))
    time.sleep(1)

    # Wait for all players to finish before moving to the next question
    barrier.wait()
```

LIVE DEMO

Enough about code. Let's play!

I need 2 volunteers with laptops and telnet installed.

QUESTIONS?