Game Development

1. Concept Development

- Game Idea: The initial idea is to create a multiplayer Rock-Paper-Scissors game where two players can compete over a network.
- Core Mechanics: The game will follow standard Rock-Paper-Scissors rules with added elements like health points and win streaks affecting damage.

2. Design

- Game Design:
    - Rules: Players choose Rock, Paper, or Scissors. The winner is determined based on standard rules (Rock beats Scissors, Scissors beat Paper, Paper beats Rock). Players have health points (HP) which decrease based on the game outcome.
    - Health and Damage: Each player starts with 100 HP. Winning a round deals 10 damage to the opponent. Winning three rounds in a row doubles the damage dealt.
    - User Interface: Simple text-based interface where players input their choices and see the results.
- System Design:
    - Server-Client Architecture: The game uses a server-client model where the server handles game logic and state, and the clients handle user inputs and display game status.
    - Networking: Utilizes sockets for communication between server and clients. The server listens for connections and processes game rounds, while clients send choices and receive results.

3. Implementation

- Programming Language: C, chosen for its performance and control over system resources.
- Libraries: Standard C libraries for I/O and socket programming (<stdio.h>, <stdlib.h>, <string.h>, <sys/types.h>, <sys/socket.h>, <netinet/in.h>, <unistd.h>, <arpa/inet.h>).
- Code Structure:
    - Server (server.c):

        - Sets up a listening socket and accepts connections from two clients.
        - Manages game rounds, processes player choices, and updates health points.
        - Sends game results to clients.
    - Client (client.c):
        - Connects to the server.
        - Sends player choices and receives game results.
        - Displays health bars and round outcomes to the player.

4. Testing

- Unit Testing: Test individual functions like game_round to ensure they return correct results.
- Integration Testing: Test the entire server-client interaction to ensure smooth communication and correct game logic.
- User Testing: Have real users play the game to identify any usability issues or bugs.

Detailed Code Explanation

Server (server.c):

- Initialization:
    - Creates a socket using socket().
    - Binds the socket to a port using bind().
    - Listens for incoming connections using listen().
- Game Loop:
    - Accepts connections from two clients using accept().
    - Initializes health points for both players.
    - Enters a loop where it:
        - Receives choices from both clients.
        - Determines the winner using game_round().
        - Updates health points based on the round outcome.
        - Sends updated health points and round results back to the clients.
    - Ends the game when one player's health reaches zero.

Client (client.c):

- Initialization:
    - Creates a socket and connects to the server using connect().

- Game Loop:
    - Prompts the player to input their choice.
    - Sends the choice to the server.
    - Receives and displays the round results and updated health points from the server.
    - Continues until the game is over (either player's health reaches zero).

Key Points in Game Development

1. Networking:
    - Understanding of TCP/IP protocols and socket programming is crucial.
    - Ensuring reliable communication between server and clients to maintain game state consistency.
2. Game Logic:
    - Implementing clear and fair rules for determining round winners.
    - Managing player health and incorporating additional features like streak-based damage.
3. User Interaction:
    - Providing clear instructions and feedback to players.
    - Ensuring the game loop is intuitive and engaging.
4. Error Handling:
    - Handling network errors and ensuring the game can recover or exit gracefully.
5. Testing and Debugging:
    - Rigorous testing to ensure all edge cases are handled.
    - Debugging network-related issues which can be challenging due to asynchronous communication.