Contribution Statements- Team 3

Brandon: Implementation of synchronization for the hash map, ensuring critical sections are properly managed using read and write locks.Implemented heartbeat functionality in servers to monitor client disconnections and manage game data accordingly. Refined methods to make them idempotent, ensuring smooth operation and data consistency.

Nimrat: Implemented idempotence by modifying methods/functions and introducing sequence numbers for consistent operation. Added a duplicator function to the client for testing idempotence, contributing to the robustness of the system. Modified client to take server as input

Rutu: Updated code documentation, ensuring it meets standards and provides comprehensive insights into the assignment implementation. Created communication protocols designs as well as this document explaining design choices and task allocations. Quality testing was also completed. Modified client to take server as input.

We all agree that we contributed equally towards this project. Communication was always kept in between all team members via Discord and Google Drive, and where everyone would join the meetings as decided upon. Questions were thoroughly asked whenever needed and the entire team would help to the best of their ability. A respectful demeanor was always kept within the team and communication was kept always constant about the project.

We all worked in the areas where we had our strengths. This project was completed by decomposing big tasks, and assigning each smaller task to one another. We all chose a deadline together of when to complete each task and would individually and responsibly complete our own parts.

All in all, we are all happy with our contributions as a team and would give each other full marks in terms of contribution! No issues to report.

Tasks

Task Set 1: Improvements

Tasks

1. Synchronization for Hash Map

Task Set 2: Implement Communication Changes

Tasks

1. Implement Heart Beat functionality
2. Make methods idempotent
3. Modify client to take server as input
4. Add a duplicator in the client

Task Set 3: Error Handling and Testing

Tasks

1. Advanced Exception Handling

Task Set 4: Refinement and Additional Features

Tasks

1. Code Optimization / Refactoring
2. Advanced Criss Cross Puzzle
3. Documentation

Communication Protocol Design Descriptions (Failure Detection):

- **Client Registration and Heartbeats**:
    - When a client connects to the server, it registers itself by invoking the keepMyNameWhileAlive method with its ID.
    - Periodically, each client sends a heartbeat signal to the server by invoking the heartBeat method with its ID. This indicates that the client is still active and functioning.
- **Server-Side Failure Detection**:
    - The server periodically checks for disconnected clients. It does so by maintaining a record of client states, including their last heartbeat timestamps.
    - If a client fails to send a heartbeat within a certain time window (e.g., 10 seconds), the server considers the client as disconnected.
- **Inactive Client Removal**:
    - When the server detects that a client has become inactive (i.e., failed to send a heartbeat), it marks the client as inactive in its records.
    - After marking inactive clients, the server removes their data from its records to prevent stale entries, also helping with optimizing resource usage.
- **Idempotent Methods**:
    - Some server methods, such as checkWord, checkScore, checkUser, checkWin, checkLoss, are designed to be idempotent. This means that invoking these methods multiple times with the same parameters produces the same result. Idempotent methods are great for fault tolerance and can simplify failure recovery.

Communication Protocol Specifics - Failure Detection

Server:

1. Initialization:
    - main(String[] args): Initializes the server, sets up RMI registry, binds server object, and awaits client connections.
2. Heartbeat Mechanism:
    - keepMyNameWhileAlive(int user_id): Registers a client in the ClientStateRecord to monitor its liveliness.
    - heartBeat(int user_id): Turns the client status to alive in the ClientStateRecord to indicate active communication.
3. Client Management and Failure Detection:
    - ClientStateRecord: Manages client state including ID, liveliness, and registration timestamp.
    - removeClientRecord(int user_id): Removes a client record from the ClientStateRecord if it is inactive or disconnected.

Client Component:

1. Initialization and Connection:
    - main(String[] args): Initializes the client, prompts for login information, establishes RMI connection with the server, and starts the heartbeat thread.
    - loginPrompt(Scanner scan): Prompts the user to enter their login ID for accessing the crossword puzzle game.
2. Game Interaction:
    - primaryHandler(Scanner scan): Orchestrates user interactions after login, such as adding/removing words, checking scores, starting/ending games, and guessing letters/words.
    - promptBeforeStartingGame(Scanner scan): Displays a menu of options before starting the game and performs actions based on user input.

- gameHandler(Scanner scan): Manages the game after the user selects to start the game, including setup, interaction, and handling game-ending conditions.
- gameMenu(Scanner scan): Displays the game menu options, reads user input, and performs corresponding actions based on the choice.

3. Remote Method Invocation:
    - Remote methods such as addWord, removeWord, checkWord, guessLetter, guessWord, checkWin, checkLoss, updateUserScore, endGame, etc., are invoked on the server to perform game-related actions and queries.
    - Heartbeat signals are sent periodically to the server using the heartBeat(int user_id) method to maintain liveliness.
4. Error Handling:
    - Exceptions are handled to ensure graceful termination and appropriate user feedback in case of communication failures or errors.

Duplicator Component (Duplicator):

1. Function Invocation:
    - runInterface(CrissCrossPuzzleServer duplicate, ConnectionFunction function): Invokes a function on the CrissCrossPuzzleServer interface object to test server idempotence of functions.

<u>Design Choices and Reasonings as to the Why behinds decisions?</u>

1. **Separation of Workflows:**
    - The code separates between client and server components. The client handles user interactions and input/output, while the server manages game logic and interacts with microservices.
    - **Reason:** This separation enhances code modularity, maintainability, and scalability. It allows for easier debugging, and testing too.
2. **User Authentication/Login Prompt:**
    - The code prompts users to enter their login IDs to ensure unique identifiers for each participant in a number form.

- **Reason:** This ensures user identification and prevents multiple instances of the same user from starting multiple games concurrently, maintaining fair gameplay and server resources. This was largely improved via assignment 2.

3. **Error Handling Mechanisms:**
    - The code implements error handling mechanisms to catch exceptions like RemoteException and MalformedURLException.
    - **Reason:** Error handling increases the validity of the communication process, ensuring that exceptions are handled gracefully and preventing abrupt termination of the program. It should increase reliability and user experience.
4. **Heartbeat Functionality:**
    - The client periodically sends heartbeat signals to the server to indicate its active status.
    - **Reason:** Heartbeat functionality enables failure detection and ensures that the server can track the activity status of connected clients. It helps maintain the integrity of the game session and identifies inactive clients for appropriate handling.
5. **Remote Method Invocation (RMI) Protocol:**
    - RMI is used for communication between client and server, using method calls with defined interfaces and parameters.
    - **Reason:** RMI simplifies communication by abstracting away the underlying networking details and providing a familiar method-based interaction model especially for us developers. It increased code readability for us, reduced complexity, and promoted interoperability between components. This was a part of the specifications for assignment 2 but I added it here under design choices because the way communication is handled is a big choice that can impact the communication channels.

Instructions for Running the Project **Criss-Cross Word Puzzle Game**:

1. Move into the RMI folder
2. Complete a "javac *.java" in the terminal
3. Open 3 terminals + the number of terminals wanted for clients
4. In terminal 1 enter the RMI folder and run "java WordRepoRPC"
5. In terminal 2 enter the RMI folder and run "java UserAccountsRPC"
6. In terminal 3 enter the RMI folder and run "java ServerRPC "
7. Enter the clients into the RMI folder
8. Have the clients run "java ClientRPC :1099/CrissCrossPuzzleServer"
9. Login and enjoy playing

Game Workflow (technical insight):

1. **Client Startup:**
    - The client application starts, and the user is prompted to enter their login ID.
    - The login ID ensures unique identification of each player.
2. **Connection to Server:**
    - The client establishes a connection to the server using Remote Method Invocation (RMI).
    - The server-side component is responsible for managing game logic and interaction with microservices.
3. **Heartbeat Mechanism:**
    - After establishing the connection, the client starts a separate thread to send periodic heartbeat signals to the server.
    - Heartbeat signals indicate the client's active status to the server, facilitating failure detection and session management.
4. **Primary Game Handling:**
    - Upon successful connection and login, the client enters the primary game handling phase.

    - The client checks whether the user is already active or proceeds with the game menu options.
5. **Game Menu Display:**
    - The client displays a menu of options to the user, allowing them to perform various game-related actions.
    - Options include adding/removing words from the repository, checking the score, and starting a game.
6. **Game Initialization:**
    - If the user chooses to start a game, they specify the game level (difficulty) and the number of failed attempts allowed.
    - The client sends the game initialization request to the server, which sets up the game with the specified parameters.
7. **Game Interaction Loop:**
    - The client enters a loop where it displays the game menu to the user and handles user inputs until the game is completed or the user quits.
    - During the loop, the user can guess letters/words to solve the puzzle, check the score, or perform other game-related actions.
8. **Game Completion:**
    - If the user successfully solves the puzzle, they are notified of their victory, and the game ends.
    - If the user exhausts their allowed attempts without solving the puzzle, they are notified of their loss, and the game ends.
9. **Session Termination:**
    - Upon quitting the game or completing a session, the client notifies the server and closes the connection.
    - The server handles session termination and updates user activity status accordingly.
10. **Error Handling:**
    - Throughout the workflow, error handling mechanisms are in place to catch and handle exceptions such as RemoteException.