INFO0010: Introduction to computer networking
Projet 2

SICIM Merve S184346, EL MASRI Sam S190377

December 2024

Software Architecture

The Minesweeper server is designed with modularity and scalability in mind. The problem was broken down into the following major components:

1. Coordinate.java

Class: `Coordinate`

- Represents a cell on the grid.
- **Possible statuses for a cell:**
  - `UNREVEALED` (#)
  - `REVEALED` (displays a value, such as a number or bomb)
  - `FLAGGED` (F)

2. Grid.java

Class: `Grid`

- Represents the game board as a matrix of cells.
- **Attributes:**

3. MinesweeperServer.java

Class: `MinesweeperServer`

- Manages client connections and their interactions with the server.
- **Attributes:**
  - `playersClassement`: Player leaderboard (name, time).
  - `activeSessions`: Active sessions linked to cookies.

4. SessionInfo.java

Class: `SessionInfo`

- Represents a user session's data.
- **Attributes:**
  - `timestamp`: Timestamp of the last activity.
  - `currentGame`: `Grid` object representing the ongoing game.
  - `playerName`: Player's name (default: `"Anonymous"`).

5. WebSocket.java

Class: `WebSocket`

- Handles WebSocket communication between the server and client.
- **Key Methods:**
  - `receive()`: Reads data sent by the client (WebSocket frames).
  - `send(String message)`: Sends formatted messages to the client.

6. Worker.java

Class: `Worker`

- Represents a thread that processes a client's requests.
- **Functionality:**
  - Calls `processClientRequests()` from the server.
  - Releases the thread upon session completion.

7. leaderboard.html and play.html

HTML Files:

- `leaderboard.html`:
  - Displays the player leaderboard.
  - Connects to the server via WebSocket to fetch data.
- `play.html`:
  - Game interface using CSS for the grid display.
  - Includes JavaScript to handle interactions with the server via WebSocket.
  - Buttons for functionalities like `CHEAT` and leaderboard access.

Multi-thread Coordination

To ensure the server handles multiple simultaneous connections efficiently, we implemented the following mechanisms:

- **Thread Pool:** The server uses a fixed-size thread pool to handle a maximum number of concurrent requests. This approach prevents *DDOS*.
- **Synchronization:** Shared resources, such as the leaderboard or session data, are managed using thread-safe collections (e.g., `ConcurrentHashMap`) to avoid race conditions.
- **Timeouts:** Connection timeouts are enforced to prevent idle or malicious connections from consuming server resources indefinitely.
- **Error Handling:** Mechanisms are in place to gracefully handle unexpected errors or malformed requests without affecting the stability of other threads.

Limits

While the server is functional, it has the following limitations:

- **Scalability:** The server's performance is limited by the maximum number of threads in the pool. High traffic could result in delays or dropped connections.
- **Real-Time Updates:** Although WebSocket-based updates are implemented, they are not fully optimized for frequent leaderboard updates under heavy loads.
- **Browser Dependency:** Real-time features rely on JavaScript and WebSocket support. Users with disabled JavaScript experience limited functionality.
- **HTTP codes:** The project handles several HTTP status codes such as 400, 404, and 405. However, it could be extended to support additional codes.

Possible Improvements

The following improvements could enhance the server's functionality and user experience:

- **Persistent Storage:** Implement a database to store the leaderboard persistently across server restarts.
- **Advanced Real-Time Updates:** Optimize WebSocket communication to handle frequent updates for the leaderboard and game states efficiently.
- **Improved Security:** Add rate-limiting to prevent abuse from high-frequency requests and implement stricter validation for WebSocket and HTTP requests.
- **Enhanced Gameplay Features:** Allow users to customize game settings, such as grid size and difficulty level, for a more personalized experience.
- **Performance Optimization:** Use caching for static content and combine Gzip with Brotli compression for better performance.

Conclusion

This project successfully demonstrates the implementation of a Minesweeper server using Java Sockets and HTTP protocols. While functional, there is potential for improvement in scalability, robustness, and user experience. Addressing these areas could make the server more efficient and user-friendly.