Submission Worksheet

Submission Data

Course: IT114-005-F2025
Assignment: IT114 - Milestone 3 - Trivia
Student: Nilkanth D. (nhd5)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 12/8/2025 10:59:08 PM
Updated: 12/9/2025 12:36:23 AM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-3-trivia/grading/nhd5
View Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-3-trivia/view/nhd5

Instructions

1. Refer to Milestone3 of Trivia
    1. Complete the features
2. Ensure all code snippets include your ucid, date, and a brief description of what the code does
3. Switch to the `Milestone3` branch
    1. `git checkout Milestone3`
    2. `git pull origin Milestone3`
4. Fill out the below worksheet as you test/demo with 3+ clients in the same session
5. Once finished, click "Submit and Export"
6. Locally add the generated PDF to a folder of your choosing inside your repository folder and move it to Github
    1. `git add .`
    2. `git commit -m "adding PDF"`
    3. `git push origin Milestone3`
    4. On Github merge the pull request from `Milestone3` to `main`
7. Upload the same PDF to Canvas
8. Sync Local
    1. `git checkout main`
    2. `git pull origin main`

Section #1: (1 pt.) Core Ui

Progress: 100%

--- Section Collapsed ---

Section #2: (2 pts.) Project Ui

Progress: 100%

--- Section Collapsed ---

Section #3: (4 pts.) Project Extra Features

Progress: 100%

--- Section Collapsed ---

Section #4: (2 pts.) Project General Requirements

Progress: 100%

≡ Task #1 (1 pt.) - Away Status

Progress: 100%

--- Task Collapsed ---

≡ Task #2 (1 pt.) - Spectators

Progress: 100%

Details:

- Spectators are users who didn't mark themselves ready
  - Optionally you can include a toggle on the Ready Check page
- The can see all chat but are ignored from turn/round actions and can't send messages
- Spectators will have a visual representation in the user list to distinguish them from other players
- A message should be relayed to the Game Events Panel that a spectator joined (i.e., during an in-progress session)

🖼 Part 1:

Progress: 100%

Details:

- Show the UI indicator of a spectator (visual and message)
- Show the related code flow from UI to server-side back to UI for showing the status
- Show the related code flow for sending the message to Game Events Panel
- Show various examples across 3+ clients of spectator status (including Game Events Panel messages)
- Show the code that ignores a spectator from turn/round logic
- Show the code that prevents spectators from sending messages (server-side)
- Show the spectator's view of the session
- Show the code related to the spectator seeing the session data (including things participants won't see)

````
**7. User List Display (Spectator Indicator)**

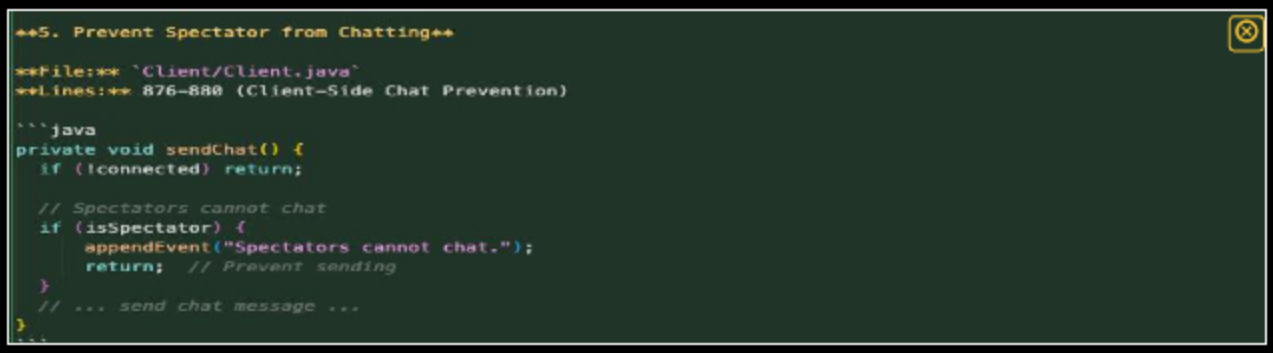
**File:** `Client/User.java`
**Line:** 32 (Spectator Indicator)

```java
````
````

```
if (spectator) sb.append(" [SPECTATOR]");  // Shows [SPECTATOR] in user list
```

related code flow from UI to server-side back to UI for showing the status

related code flow for sending the message to Game Events Panel



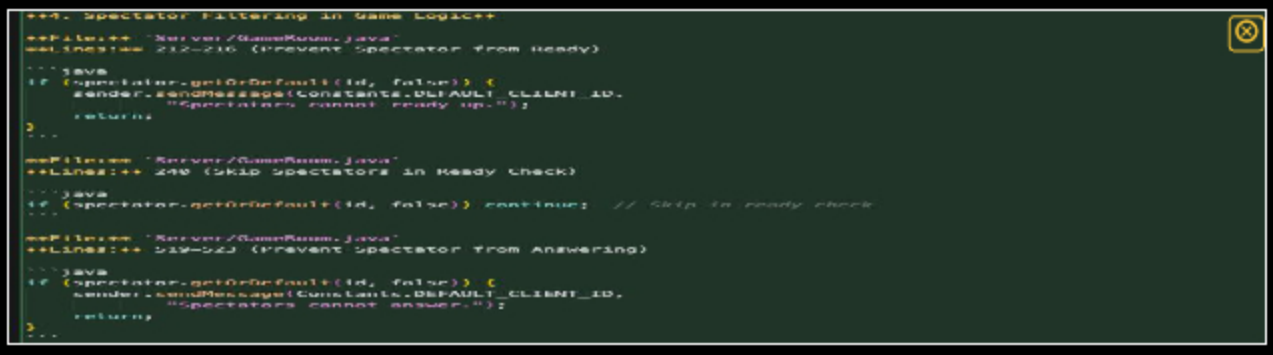
```
**5. Prevent Spectator from Chatting**

**File:** `Client/Client.java`
**Lines:** 876-880 (Client-Side Chat Prevention)

```java
private void sendChat() {
  if (!connected) return;

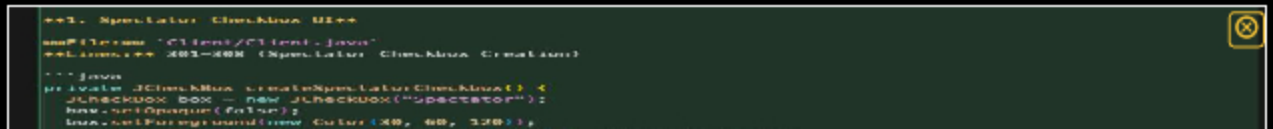
  // Spectators cannot chat
  if (isSpectator) {
      appendEvent("Spectators cannot chat.");
      return;  // Prevent sending
  }
  // ... send chat message ...
}
```
```

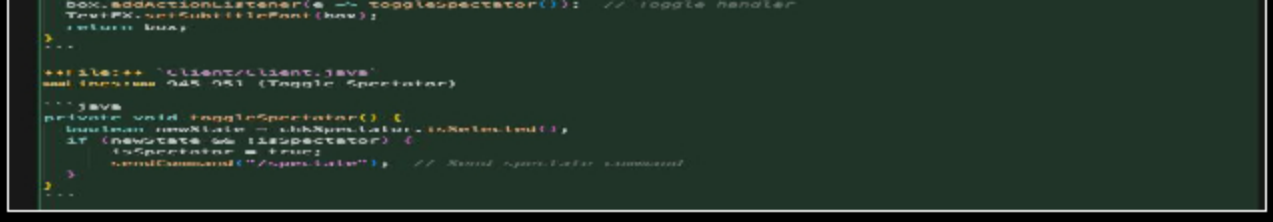
the code that ignores a spectator from turn/round logic



the code that prevents spectators from sending messages (server-side)

the code related to the spectator seeing the session data (including things participants won't see)





the spectator's view of the session

Saved: 12/9/2025 12:27:27 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for the spectator logic from server-side and to UI
- Briefly explain how the server-side ignores the user from turn/round logic
- Briefly explain the logic that prevents spectators from sending a message
- Briefly explain the logic that shares extra details to the spectator (information normal participants won't see such as the correct answer)

Your Response:

When a user selects Spectator mode, the client sends /spectate and the server sets their spectator flag, clears their ready state, broadcasts the update, and includes the spectator status in the next UserListPayload. Clients receive this payload, rebuild their user list, and show "[SPECTATOR]" in the UI. Server-side, spectators are ignored in all turn logic—ready checks, lock checks, and answer handling all skip users marked as spectators, and handleAnswer() rejects any attempt to answer. The client also prevents spectators from chatting; sendChat() checks isSpectator and blocks outgoing messages. Spectators still receive all game broadcasts, including questions, lock-in events, points, and the correct answer, allowing them full visibility while remaining non-participants.

Saved: 12/9/2025 12:27:27 AM

Section #5: (1 pt.) Misc

Progress: 100%

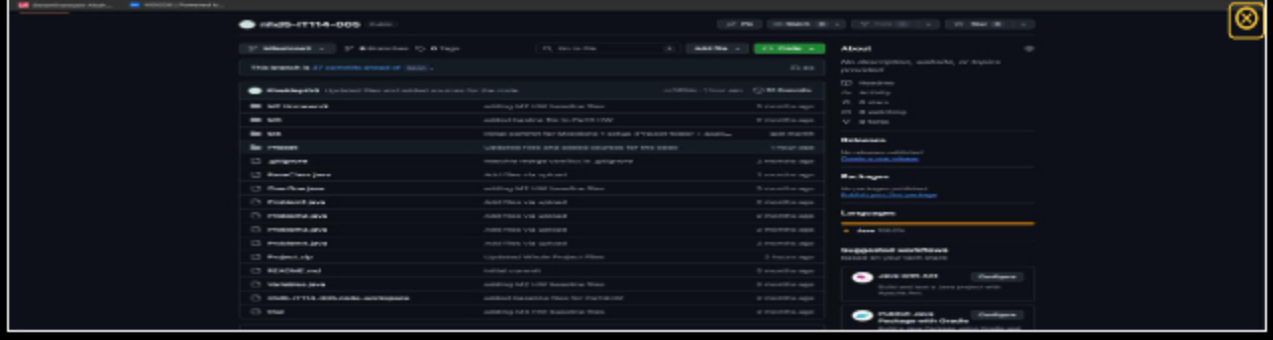
Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

Details:
From the Commits tab of the Pull Request screenshot the commit history



commit histroy

Saved: 12/9/2025 12:29:14 AM

Part 2:

Progress: 100%

Details:
Include the link to the Pull Request for Milestone3 to main (should end in `/pull/#`)

URL #1
https://github.com/Pixeldepth5/nhd5-IT114-005/commits/393dcfb67c3fe23a9d3e9f96700a0e28d72c0630

URL
https://github.com/Pixeldepth5/n

Saved: 12/9/2025 12:29:14 AM

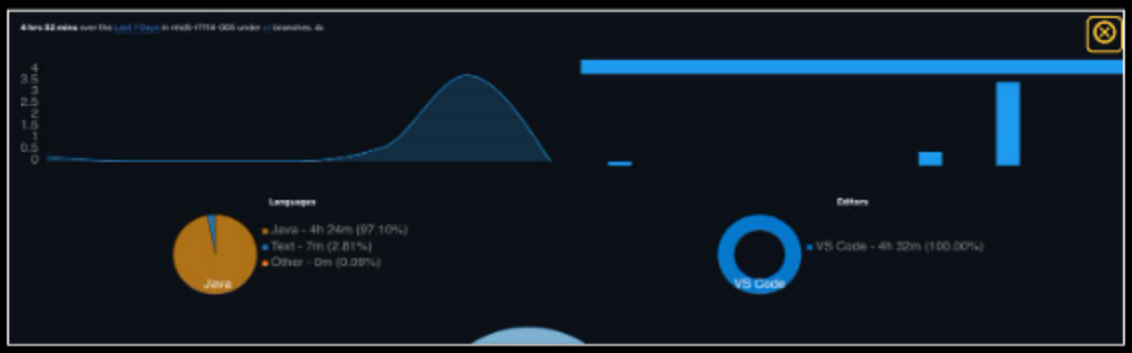
Task #2 (0.33 pts.) - WakaTime - Activity

Progress: 100%

Details:

- Visit the WakaTime.com Dashboard
- Click `Projects` and find your repository
- Capture the overall time at the top that includes the repository name
- Capture the individual time at the bottom that includes the file time
- Note: The duration isn't relevant for the grade and the visual graphs aren't necessary

Wakatime



Wakatime

Saved: 12/9/2025 12:30:23 AM

Task #3 (0.33 pts.) - Reflection

Progress: 100%

Task #1 (0.33 pts.) - What did you learn?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

In the project, I learned how to develop a multi-threaded client-server application using Java sockets and Swing GUI. I got hands-on experience with real-time communication among multiple clients and a server, handling concurrent connections, and synchronizing game state across participants. This project taught me how to design a messaging system based on payloads, how to manage game logic on the server side, and how to create responsive user interfaces dynamically from the server events. I also learned how to manage the edge cases like spectator handling, away status, and category filtering. Above all, I learned how to structure a networked application with proper separation of concerns between client UI logic and server-side game management, enabling data consistency across all the connected clients.

Saved: 12/9/2025 12:32:53 AM

Task #2 (0.33 pts.) - What was the easiest part of the assignment?

Progress: 100%

Details:

Briefly answer the question (at least a few decent sentences)

Your Response:

The easiest part was working with the data structures and payload classes. Creating the User class, Payload classes, and UserListPayload was straightforward since they were simple data containers with getters and setters. The toDisplayString() method for formatting user information was easy to implement. Additionally, parsing and formatting question data using pipe-delimited strings was simple and didn't require complex parsing logic. These foundational data structures provided a solid base for the more complex networking and game logic features.

Saved: 12/9/2025 12:34:30 AM

Task #3 (0.33 pts.) - What was the hardest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

The hardest part was implementing the game logic flow, especially managing the ready check system and ensuring games started only when appropriate conditions were met. Determining when all active players (excluding spectators and away users) were ready required careful filtering logic. The round management, including timer synchronization, answer validation, and points calculation based on answer order, was complex. Handling edge cases like players joining mid-game, becoming spectators, or marking themselves away during an active round required extensive conditional logic throughout the codebase. Ensuring the game state reset properly between rounds while maintaining player points was also challenging.

Saved: 12/9/2025 12:36:23 AM