Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 - Milestone 3 - Hangman
Student: Aditya D. (ad273)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 8/4/2025 2:12:03 PM
Updated: 8/4/2025 3:37:06 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-3-hangman/grading/ad273
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-3-hangman/view/ad273

Instructions

1. Refer to Milestone3 of Hangman / Word guess
    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%

Task #1 (0.50 pts.) - Connection/Details Panels

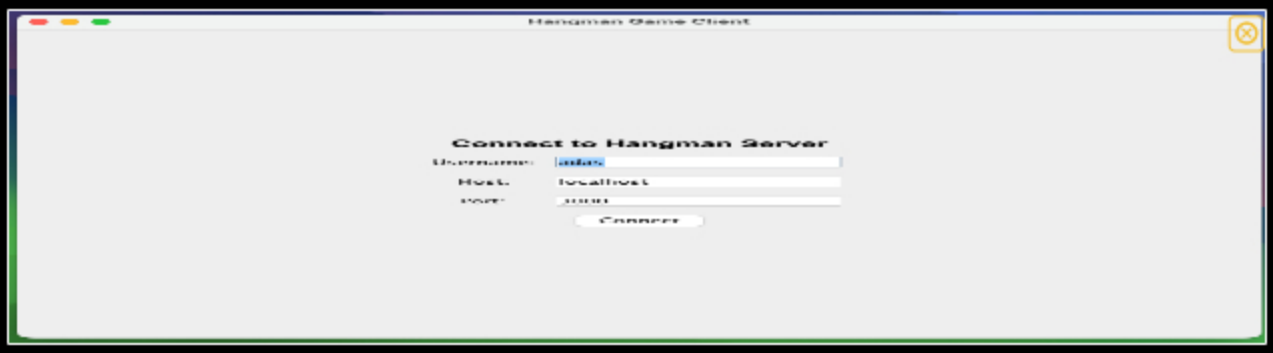
Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the connection panel with valid data
- Show the user details panel with valid data



UI of hangman: COnnect

Confirmation and ready button

Saved: 8/4/2025 2:56:14 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow from recording/capturing these details and passing them through the connection process

Your Response:

When the user launches the Hangman game, they are presented with a connection panel that includes input fields for the server address and their username. Once the user enters valid information and clicks the "Connect" button, the program captures these inputs from the GUI text fields. These values are then passed into the constructor of the GameClient class, which is responsible for managing the connection logic. Inside the GameClient, a socket connection is established using the provided server address and a predefined port. After a successful connection, the client constructs a Payload object of type CONNECT, assigns the entered username to it, and sends this payload to the server using an ObjectOutputStream. On the server side, the GameRoom class is constantly listening for incoming payloads. Once the CONNECT payload is received, the server processes it by registering the new client and preparing the game state. This establishes a persistent communication link between the client and server, enabling

the game to transition from the connection panel to the gameplay interface, where further interactions like chat and guesses will be handled.

Saved: 8/4/2025 2:56:14 PM

Task #2 (0.50 pts.) - Ready Panel

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the button used to mark ready
- Show a few variations of indicators of clients being ready (3+ clients)

3+ people joined

Mark Ready

Chris Ready

Saved: 8/4/2025 2:59:21 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for marking READY from the UI
- Briefly explain the code flow from receiving READY data and updating the UI

Your Response:

When a user clicks the "Ready" button in the ClientUI's ready panel, the button's ActionListener immediately calls the client.sendReady() method. This method creates a new Payload object with the PayloadType.READY enum and sends it through the Client's sendPayload method, which uses the ObjectOutputStream to transmit the data over the network to the server. On the server side, the ServerThread's message listening loop receives this payload and routes it through the handlePayload method, which identifies the READY type and calls the GameRoom's handleReady method with the sending client as a parameter. The GameRoom then toggles the client's ready status by adding or removing their ID from the readyPlayers HashSet, broadcasts a status message to all connected clients, and checks if all non-spectator players are ready. If the ready condition is met (minimum 2 players and all are ready), the GameRoom automatically calls startSession to begin the hangman game.

Saved: 8/4/2025 2:59:21 PM

Section #2: (2 pts.) Project Ui

Progress: 100%

Task #1 (0.67 pts.) - User List Panel

Progress: 100%

Details:

- Show the username and id of each user
- Show the current points of each user
- Users should appear in turn order
- Show an indicator of whose turn it is

Part 1:

Progress: 100%

Details:

- Show various examples of points (3+ clients visible)
    - Include code snippets showing the code flow for this from server-side to UI
- Show that the sorting is maintained across clients

- Include code snippets showing the code that handles this
- Show various examples of the turn indicators
    - Include code snippets showing the code flow for this from server-side to UI

```
        }

        Run | Debug
        public static void main(String[] args) {
            int port = 3000;
            if (args.length > 0) {
                try {
                    port = Integer.parseInt(args[0]);
                } catch (NumberFormatException e) {
                    System.err.println(x:"Invalid port number. using default: 3000");
                }
            }

            Server server = new Server(port);
            try {
                server.start();
            } catch (IOException e) {
                System.err.println("Failed to start server: " + e.getMessage());
            }
        }
    }
```

Server

Room Manager

Saved: 8/4/2025 3:19:51 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for points updates from server-side to the UI
- Briefly explain the code flow for user list sorting
- Briefly explain the code flow for server-side to UI of turn indicators

Your Response:

The code flow for updating player points begins on the server-side, specifically within the GameRoom class, where game logic determines when a player should earn points—for example, after a correct letter or word guess. The server constructs a Payload object of type POINT_UPDATE, embedding the player's ID and updated point value. This payload is then broadcast to all connected clients. On the client side, the GameClient receives the payload, updates the local data structure that holds player information, and triggers a UI update—usually within a component like UserListPanel—to reflect the new points visually.

User list sorting is handled primarily by the server, which maintains a list of all active players and sorts them based on turn order or point values before sending the updated list to clients. This

sorted list is included in a payload of type USER_LIST and distributed to all clients. Once received, each client replaces its local version of the user list with the new, sorted one and refreshes the display to maintain consistent turn order and accurate point rankings across all participants.

Turn indicators are synchronized similarly. At the start of each new turn, the server identifies which player is next to act and sends a TURN_UPDATE payload containing that player's information to all clients. The client-side logic then updates its local state to reflect the current turn and updates the UI accordingly—often by highlighting the active player's name or showing a message like "It's Bob's turn!" in the Game Events panel. This ensures that every client remains in sync regarding turn order and game progression.

Saved: 8/4/2025 3:19:51 PM

Task #2 (0.67 pts.) - Game Events Panel

Progress: 100%

Details:

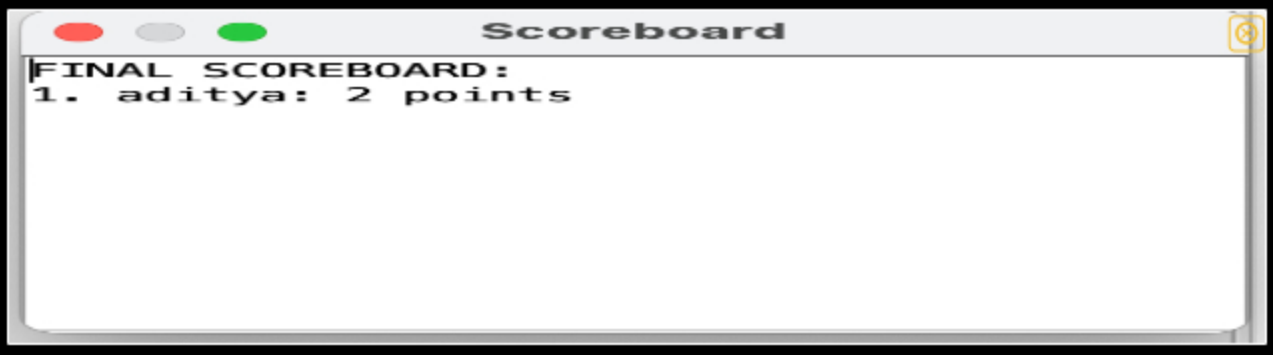
- Show the letter/guess history (including points)
- Show the scoreboard updates from Milestone 2
- Show a message of whose turn it is
- Show the countdown timer for the current turn

Part 1:

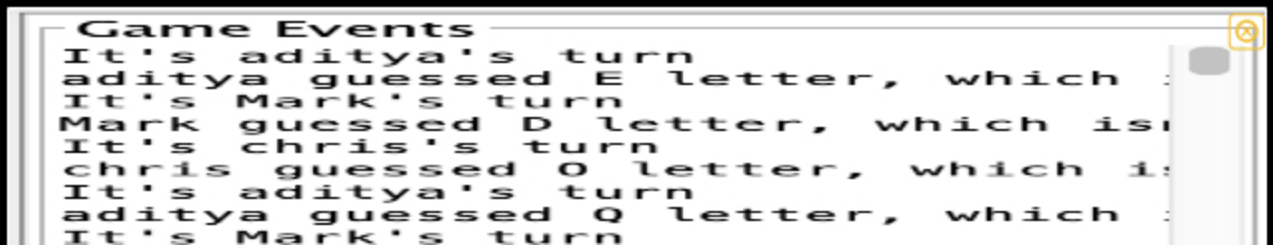
Progress: 100%

Details:

- Show various examples of each of the messages/visuals
- Show code snippets related to these messages from server-side to UI



Scoreboard



Letter Search History

Saved: 8/4/2025 3:22:03 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for generating these messages and getting them onto the UI

Your Response:

The code flow for generating messages—such as letter guesses, score updates, turn changes, and game events—and displaying them on the UI begins on the server-side, where key gameplay events are detected in classes like GameRoom. For example, when a player guesses a letter, the server checks if it is correct or not, updates the game state, and then creates a Payload with type GAME_EVENT or MESSAGE, attaching the appropriate message (e.g., "Alice guessed 'E' — Correct!" or "Bob's turn!").

This payload is then sent to all clients via the server's broadcasting method (e.g., client.send(payload)). On the client-side, the GameClient receives the incoming Payload and routes it based on its PayloadType. When a game event message is detected, it is forwarded to the UI class—typically GameUI or a specialized GameEventsPanel.

Within the UI panel, the message is added to a display component like a JTextArea or JList, often using a method such as appendMessage(String msg) or addToEventLog(). The UI is then refreshed to visually present the new message in real-time to the player, ensuring consistent and synchronized feedback across all connected clients.

Saved: 8/4/2025 3:22:03 PM

Task #3 (0.67 pts.) - Game Area

Progress: 100%

Details:

- Have some elements representing the Hangman or equivalent (can't just be a number presenting strikes)
- Have a grid of letters that the user will click to interact with
  - Chosen letters should be synced to disable the option on all clients (via server-side reply)
  - Re-enable all letters when a hangman resets
- Have a special input area for guessing the full word (separate from the chat input)
- Display blanks for the word

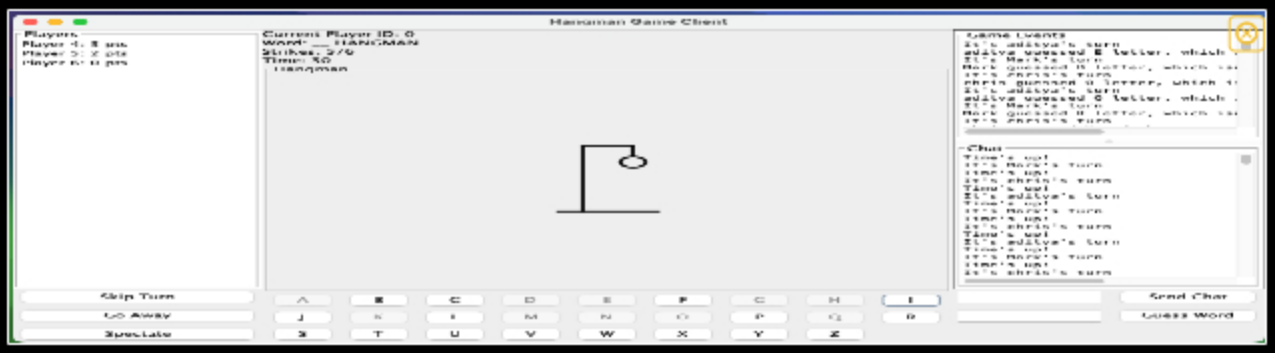
- These must update as letters are guessed correctly or word is fully guessed

Part 1:

Progress: 100%

Details:

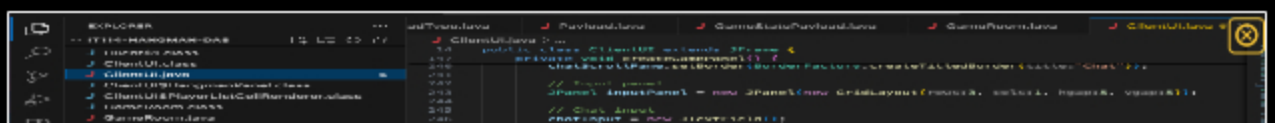
- Show various examples of the hangman/strikes indicator across 3+ clients
  - Show the related code from server-side to UI
- Show various examples of selected letters and partially completed words across 3+ clients
  - Show the related code from UI to server-side and server-side to UI for both selection and blanks
- Show the related code for a guess (UI to server-side)



Complete UI

Client UI

Client UI



Client UI

Saved: 8/4/2025 3:24:24 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for syncing the strikes and displaying them visually
- Briefly explain the code flow for letter selection and disabling (including updates to the blanks)
- Briefly explain the code for a full word guess

Your Response:

The code flow for syncing the strikes begins on the server-side in the GameRoom class. When a player guesses a letter incorrectly, the server increments the player's strike count and packages this update into a Payload with type UPDATE_STRIKES or similar. This payload is then broadcast to all clients using the server's broadcast() method. On the client-side, GameClient receives this payload and passes it to the appropriate UI handler (e.g., GameUI), which updates the strike visuals (like a hangman graphic or counter) by modifying the display components accordingly.

For letter selection and disabling, when a player clicks a letter button in the UI (within GameUI), a Payload with type LETTER_GUESS is created and sent to the server. The server checks if the letter is correct, updates the word state, and then sends back a response with the updated word blanks and the list of disabled letters. On the client side, GameClient processes the payload and updates the UI by disabling the guessed letter buttons and updating the blanks shown in the word display area.

Saved: 8/4/2025 3:24:24 PM

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

Progress: 100%

Task #1 (2 pts.) - Restore Strikes

Progress: 100%

Details:

- Setting should be toggleable during Ready Check by session creator
- Allow toggling of the option to let correct guesses restore strikes

Part 1:

Progress: 100%

Details:

- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show a before and after of a strike being restored
  - Show the related code for the UI and handling this

Togeglable button

Saved: 8/4/2025 3:25:21 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code for the host's option to toggle this feature
- Briefly explain the code related to handling the strike restoration (UI and server-side)

Your Response:

The host's option to toggle the "Restore Strikes" feature is typically implemented in the Ready Check panel (ReadyPanel.java or similar). A checkbox or toggle UI element is displayed only for the session creator, identified using a isHost flag. When the host toggles this option, an event listener creates a Payload with type TOGGLE_RESTORE_STRIKES and sends it to the server using GameClient.send().

On the server-side, GameRoom listens for this payload type and updates a boolean variable (e.g., restoreStrikesEnabled). This setting is then broadcasted to all connected clients so the UI can reflect the active status of the feature (e.g., showing "Restore Strikes: ON" for all players). The toggle is disabled or hidden for non-host users to enforce that only the host can change this rule.

Saved: 8/4/2025 3:25:21 PM

Task #2 (2 pts.) - Hard Mode

Progress: 100%

Details:

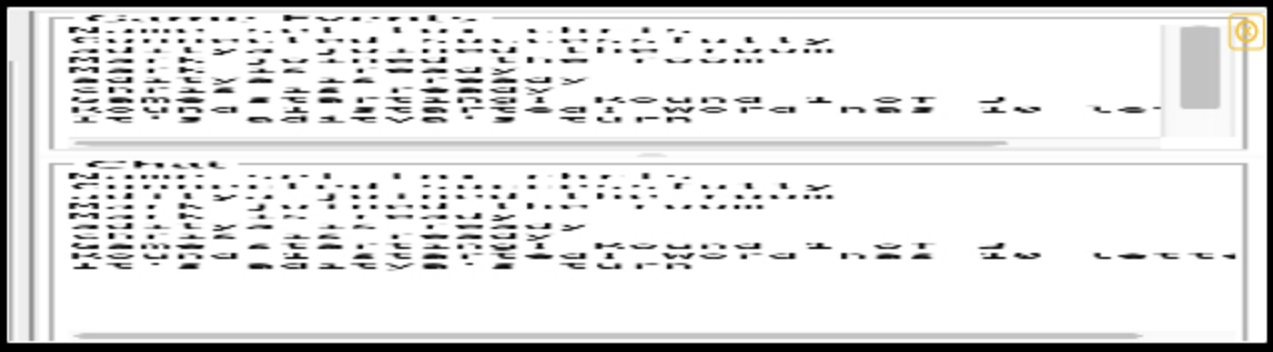
- Setting should be toggleable during Ready Check by session creator
- Hard mode prevents the letter buttons from disabling and won't show guessed letters in the Game Event Area

Part 1:

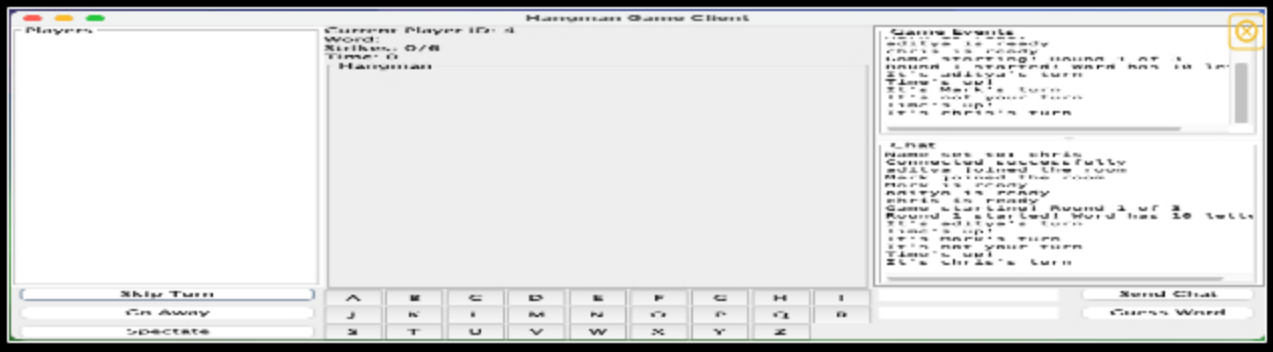
Progress: 100%

Details:

- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show a few examples of the play screen with a few turns to show these visuals have been disabled
  - Show the related code for the UI and handling the logic for hard mode



Everyone ready



Its togeable

Saved: 8/4/2025 3:14:35 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code for the host's option to toggle this feature
- Briefly explain the code related to handling and enforcing this feature

Your Response:

The host's option to toggle Hard Mode is implemented using a UI checkbox that is only active if the current user is identified as the session host. This is done by checking a condition like if (client.isHost()) when rendering the Ready Panel. The checkbox allows the host to enable or disable Hard Mode before the game starts.

When toggled, this setting is captured and sent to the server using a Payload object with a custom type like PayloadType.SETTINGS or PayloadType.READY, containing a boolean flag such as hardMode = true. The server stores this value in the session's settings (usually in the GameRoom class) and distributes it to all connected clients once the game begins.

Saved: 8/4/2025 3:14:35 PM

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

Progress: 100%

Task #1 (1 pt.) - Away Status

Progress: 100%

Details:

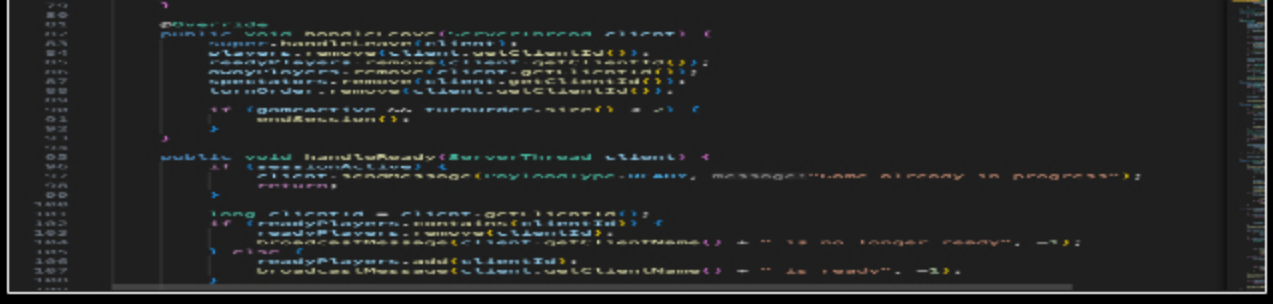
- Clients can mark themselves away and be skipped in turn flow but still part of the game
- The status should be visible to all participants
- A message should be relayed to the Game Events Panel (i.e., Bob is away or Bob is no longer away)
- The user list should have a visual representation (i.e., grayed out or similar)

Part 1:

Progress: 100%

Details:

- Show the UI button to toggle away
- 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 away status (including Game Events Panel messages)
- Show the code that ignores an away user from turn/round logic



Override

Saved: 8/4/2025 3:37:06 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for the away action from UI to server-side and back to UI
- Briefly explain how the server-side ignores the user from turn/round logic

Your Response:

The away status begins with a toggle button in the client's UI, typically found on the game screen. When the player clicks the "Away" button, a Payload is created with the type AWAY or similar, and sent to the server via the GameClient socket connection.

On the server-side, the ClientHandler receives this payload and updates that user's internal status to away = true. The server then broadcasts this status change to all connected clients using a STATUS_UPDATE payload (or similar), which includes a message like "Bob is away" or "Bob is no longer away."

When handling turn logic, the server checks if a user is marked as "away" before giving them a turn. This happens in the game loop, where it cycles through active players and skips any client with away == true. This ensures the game continues without waiting for or assigning turns to away users.

On the UI side, once the status update payload is received, each client updates the user list to visually show who is away — often by graying out the user or showing an "away" icon. A game event message is also appended to the event log area stating that the user is now away or has returned.

Saved: 8/4/2025 3:37:06 PM

Task #2 (1 pt.) - Spectators

Progress: 100%

Details:

- Spectators are users who didn't mark themselves ready

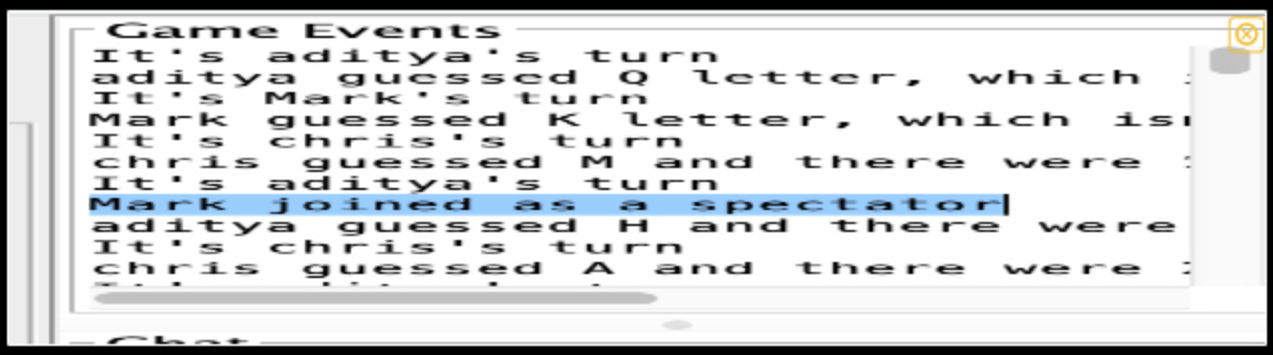
    - Optionally you can include a toggle on the Ready Check page
- The can see all chat but are ignored from turn/round actions and can't send messages
- Spectators will have a visual representation in the user list to distinguish them from other players
- A message should be relayed to the Game Events 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 such as the correct word)



With the press of the button, spectator can be given

Saved: 8/4/2025 3:27:22 PM

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)

Your Response:

The spectator logic begins during the ready check phase. When a client joins but does not click the "Ready" button, they are marked as a spectator. On the server-side, this is typically handled by tracking ready states in a ClientHandler or similar class. If a client does not send a READY-type payload, the server adds them to the game session with a spectator flag set to true.

When it's time to assign turns, the server skips over any clients marked as spectators, ensuring they are not selected to play or guessed letters. This is enforced in the game loop or turn manager logic by checking the isSpectator() flag before proceeding to the next player.

To prevent spectators from sending messages or interacting, the server checks this flag when processing incoming payloads (especially GUESS or MESSAGE types) and ignores those payloads if the sender is a spectator.

To enhance the spectator experience, the server can send additional payloads (e.g., with the correct word or strike count) only to clients marked as spectators. These payloads are tagged with a unique payload type such as REVEAL and processed on the UI to display data that is hidden from regular players, allowing the spectator to follow the game without influencing it.

Saved: 8/4/2025 3:27:22 PM

Section #5: (1 pt.) Misc

Progress: 100%

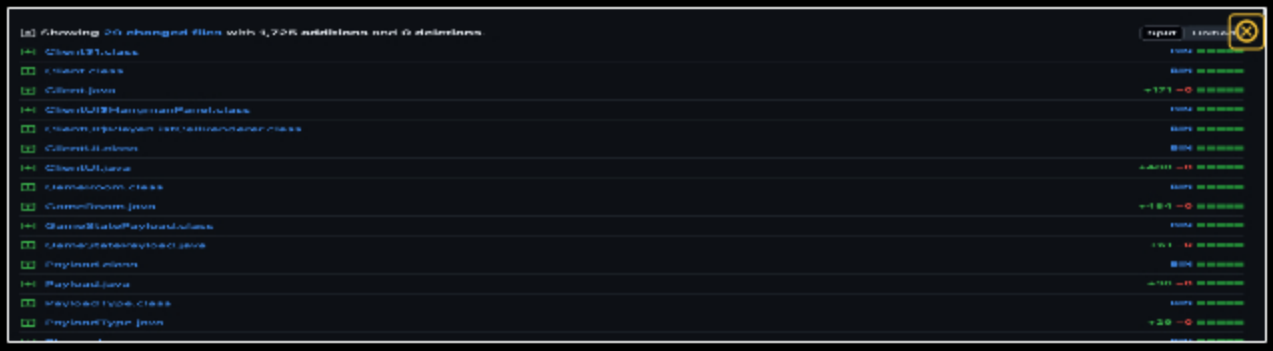
Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



Commits

Saved: 8/4/2025 3:36:25 PM

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/sadaytida24/it114-450-m2025/pull/5

URL
https://github.com/sadaytida24/it

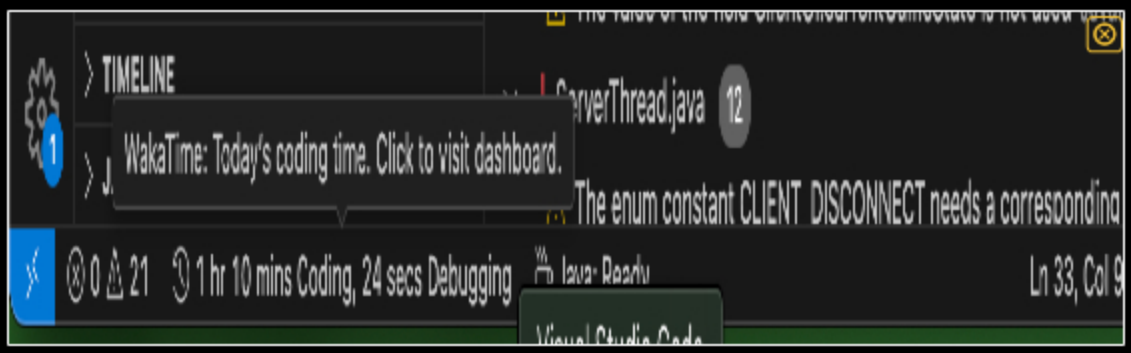
Saved: 8/4/2025 3:36:25 PM

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

Progress: 100%

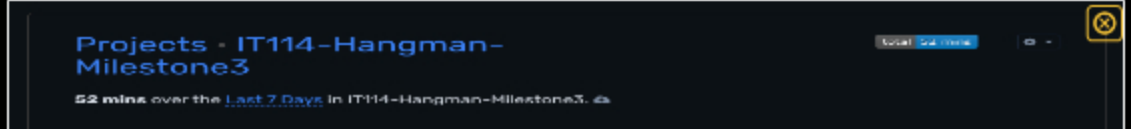
Details:

- 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



Bottom code time

Individual file time



Project and repo time

Saved: 8/4/2025 3:09:59 PM

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:

Through this assignment, I learned how to develop a real-time multiplayer game using Java with a client-server architecture. I gained hands-on experience with socket programming, GUI development using Swing, and managing state synchronization between multiple clients. I also learned how to break down a large project into modular components like UI panels, networking classes, and shared data models. Most importantly, I developed a better understanding of how to handle turn-based logic, user status (ready, away, spectator), and dynamically update the user interface based on server responses.

Saved: 8/4/2025 3:07:56 PM

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 of the assignment was setting up the initial structure of the project and organizing the different components into client, server, and shared packages. Creating the basic GUI elements like text fields, buttons, and labels

using Swing was also straightforward, especially since these were primarily reused from previous milestones. Implementing the connection logic using sockets and sending initial payloads for usernames and connection setup felt familiar and manageable due to the clear separation of responsibilities in the codebase. This foundational setup helped streamline the later stages of development.

Saved: 8/4/2025 3:08:02 PM

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:

ThTe hardest part of the assignment was getting all components—client, server, shared logic, and UI panels—to communicate seamlessly in real time. Debugging synchronization issues, especially when handling turn-based logic, ready status, and guessing inputs across multiple clients, was particularly challenging. It required carefully tracing how data flowed from the UI to the server and back, and ensuring that updates were consistently reflected on all connected clients. Additionally, resolving unexpected exceptions, missing payload handling, and integrating placeholder UI elements into a fully functional interface made the process more complex and time-consuming.

Saved: 8/4/2025 3:08:12 PM