Course: IT114-010-S2025
Assignment: IT114 - Milestone 3 - Hangman
Student: Dhruvin R. (dbr2)
Status: Submitted | Worksheet Progress: 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-010-S2025/it114-milestone-3-hangman/grading/dbr2

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`

100%

Section #1: (1 pt.) Core Ui

100%

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

Combo Task:

Weight: *50%*
Objective: *Connection/Details Panels*

Image Prompt

Weight: *50%*

Details:

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

Connection panel with valid data

The user detail panel with vaild data

Saved: 5/9/2025 11:42:42 PM

Text Prompt

Weight: *50%*

Details:

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

Your Response:

The ConnectionPanel captures host and port details, validates them, and stores them in private fields, while the UserDetailsPanel captures and validates the username. When the user clicks "Connect" in

the UserDetailsPanel, the ClientUI.connect() method is called, which retrieves all three pieces of information (host, port, username) and passes them to Client.INSTANCE.connect(), which establishes a socket connection to the server and sends the username as part of the initial handshake.

Saved: 5/9/2025 11:42:42 PM

100%

Task #2 (0.50 pts.) - Ready Panel

Combo Task:

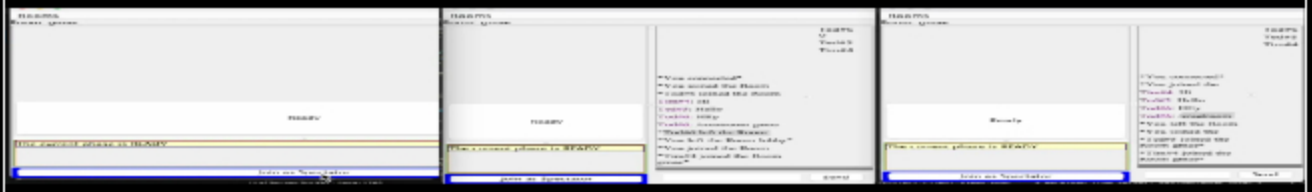
Weight: *50%*
Objective: *Ready Panel*

Image Prompt

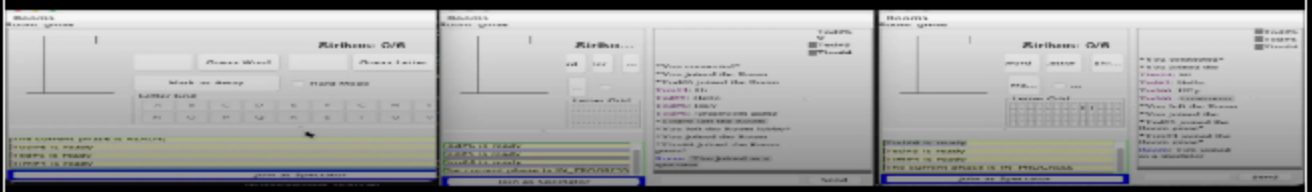
Weight: *50%*

Details:

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



shows the button used to mark ready



few variations of indicators of clients being ready

Saved: 5/9/2025 11:45:50 PM

Text Prompt

Weight: *50%*

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 marks themselves as ready in the UI, the handleReady method in BaseGameRoom is called, which updates the player's ready status and broadcasts it to all clients via sendReadyStatus. The server then checks if all players are ready to start the game. When receiving ready status updates, the client's processReadyStatus method processes the ReadyPayload and notifies all registered IReadyEvent listeners (like GameEventsPanel), which then updates the UI to show which players are ready/not ready.

100%

Section #2: (2 pts.) Project Ui

100%

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

Combo Task:

Weight: *33.33%*
Objective: *User List Panel*

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

Image Prompt

Weight: *50%*

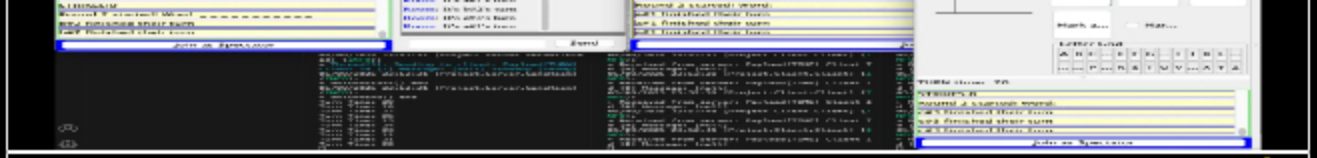
Details:

- 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

Shows various example of points

server-side of the code flow for points

Client side snippet of the code flow for point



examples of turn indicators

point sorting code flow

code snipped on the server side for turn indicator

100%

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

Combo Task:

Weight: *33.33%*
Objective: *Game Events Panel*

Details:

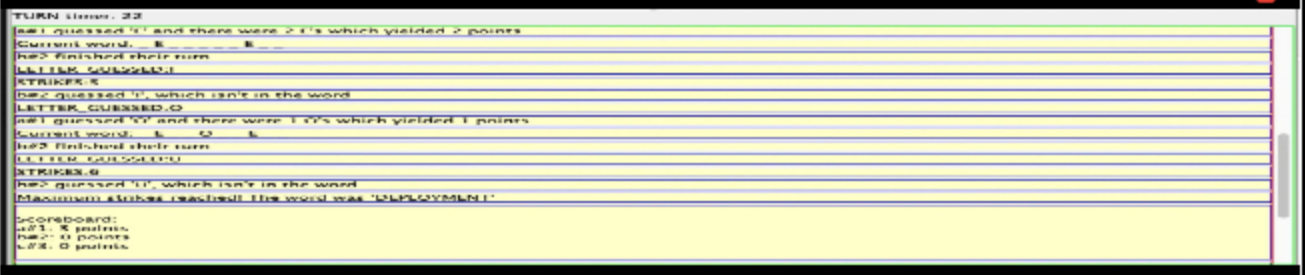
- Show the letter/guess history (including points)
- Show the scoreboard updates from Milestone 2
- Show a message of whose turn it is
- Show the countdown timer for the current turn

Image Prompt

Weight: *50%*

Details:

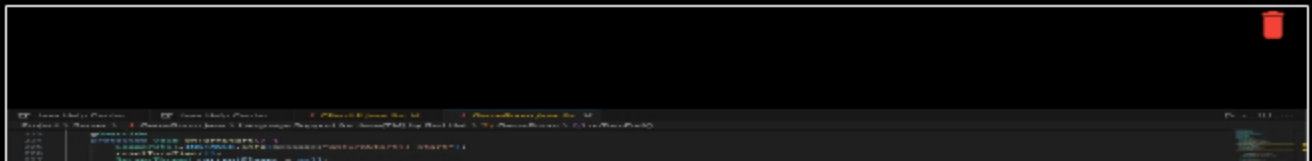
- Show various examples of each of the messages/visuals
- Show code snippets related to these messages from server-side to UI



Shows the snnipet for guessed letters, turn time, scoreboard, also states who's turn it is.

Letter/Guess History with Points

Scoreboard Updates



Turn Indicator Messages

Turn Countdown Timer

Turn Countdown Timer on the client side

100%

Task #3 (0.67 pts.) - Game Area

Combo Task:

Weight: *33.33%*
Objective: *Game Area*

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

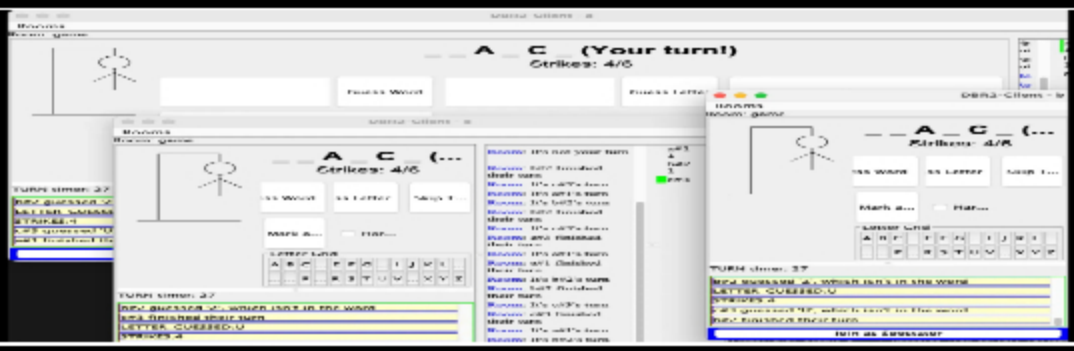
Image Prompt

Weight: *50%*

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)

Selected letters and partially completed words



example of the hangman/strike indicator across all client

code for the strike/hangman indicator from server to ui

relevant code for the hangman indicator

code for selected letter and partially completed word. (server to ui)

code for selected letter and partially completed word. (ui to sserver)

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

Task #1 (2 pts.) - Restore Strikes

Combo Task:

Weight: *50%*
Objective: *Restore Strikes*

Details:

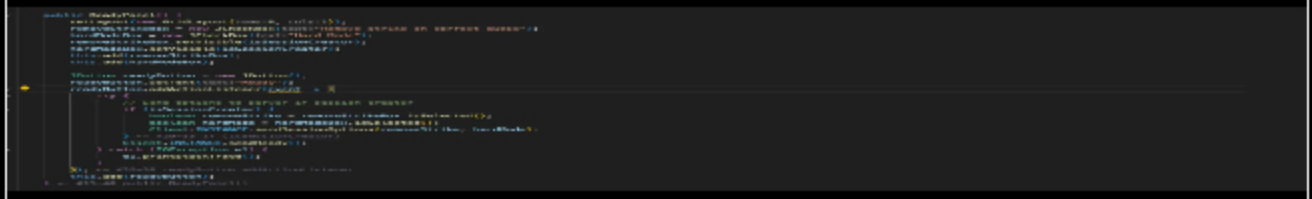
- Setting should be toggleable during Ready Check by session creator
- Allow toggling of the option to let correct guesses restore strikes

Image Prompt

Weight: *50%*

Details:

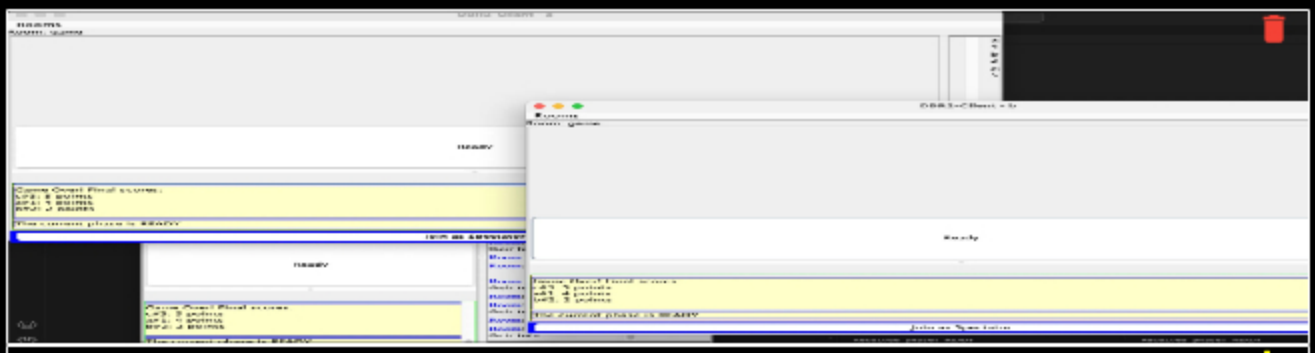
- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show a before and after of a strike being restored
  - Show the related code for the UI and handling this



code for the ready check.

code for before strike.

after



ready check screen

Saved: 5/10/2025 1:13:57 PM

Text Prompt

Weight: *50%*

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:

100%

Task #2 (2 pts.) - Hard Mode

Combo Task:

Weight: *50%*
Objective: *Hard Mode*

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

Image Prompt

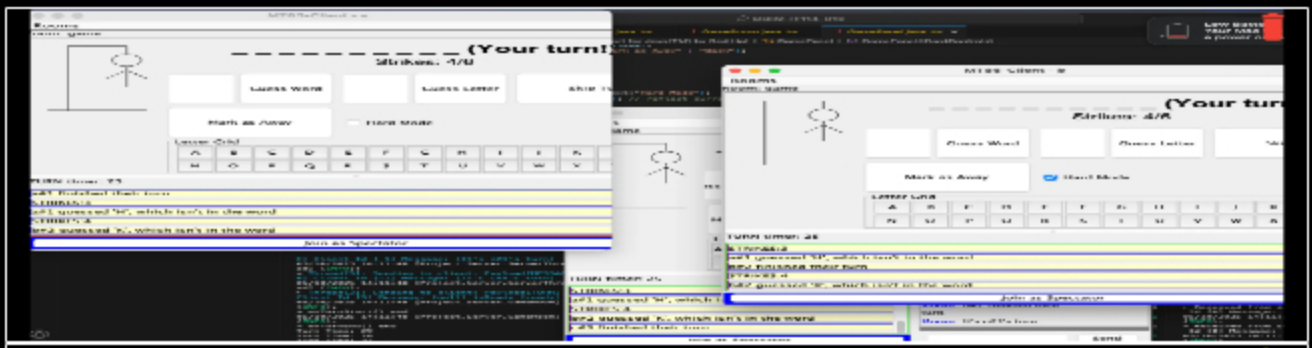
Weight: *50%*

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

options for the hardmode.

hard mode code



example of the hard mode

Saved: 5/10/2025 2:12:51 PM

Text Prompt

Weight: *50%*

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 code for the host's option to toggle the Ready Check feature includes a method that checks if the current client is the host and enables a button to start the game, while the handling code ensures that only the host can interact with game controls during turns by enabling or disabling input fields and buttons based on the host's statu

Saved: 5/10/2025 2:12:51 PM

100%

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

100%

Task #1 (1 pt.) - Away Status

Combo Task:

Weight: *50%*
Objective: *Away Status*

Details:

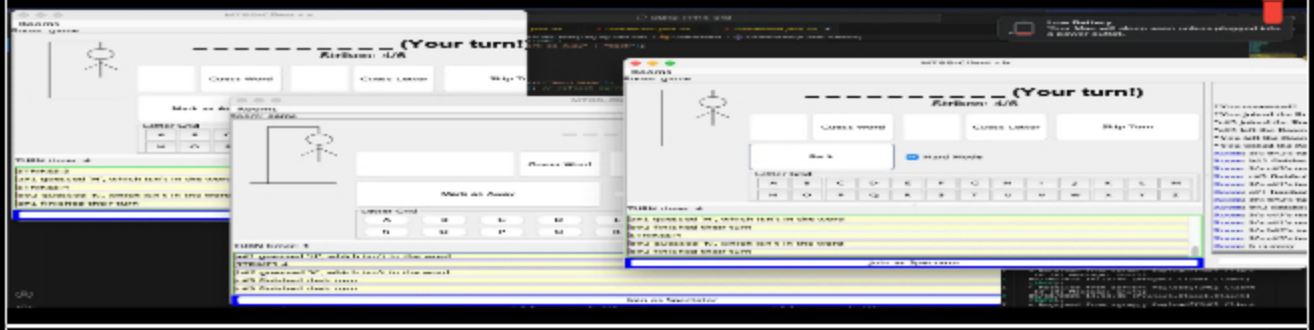
- Clients can mark themselves away and be skipped in turn flow but still part of the game
- The status should be visible to all participants
- A message should be relayed to the Game Events Panel (i.e., Bob is away or Bob is no longer away)
- The user list should have a visual representation (i.e., grayed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

Image Prompt

Weight: *50%*

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



The ui button to toggle away.

code for the away button

code for the iignore an away user

Saved: 5/10/2025 2:21:51 PM

Text Prompt

Weight: *50%*

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 action flow begins when the user toggles their status via the UI, sending an AwayStatusPayload to the server, which updates the user's status and broadcasts the change to all clients to update their UI. On the server-side, users marked as away are ignored in turn logic, preventing them from participating in the current round.

Saved: 5/10/2025 2:21:51 PM

100%

Task #2 (1 pt.) - Spectators

Combo Task:

Weight: *50%*
Objective: *Spectators*

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)

Image Prompt

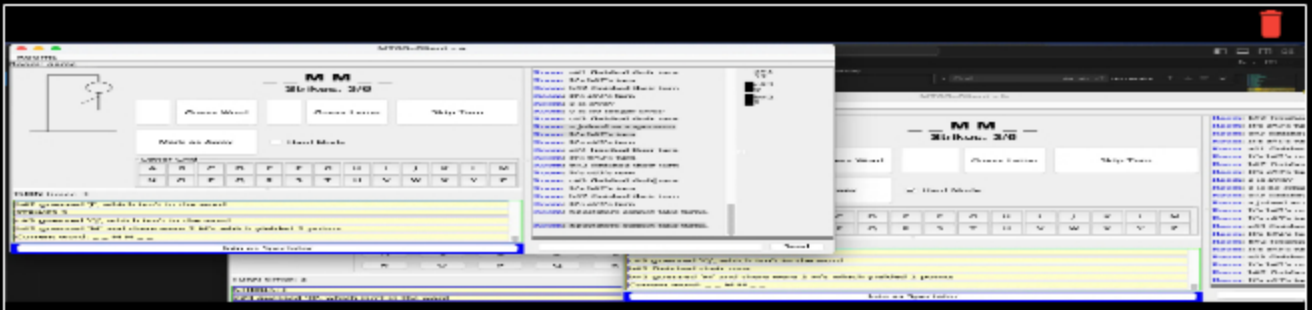
Weight: *50%*

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)

ui indicator for spectator.



example of spectator status and the view of the spectators

Saved: 5/10/2025 3:08:24 PM

Text Prompt

Weight: *50%*

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 on the server-side when a client requests to join as a spectator, which is processed by adding them to a special list and notifying all clients, including updating the UI to reflect their status; the server then ignores any turn-related actions from spectators during gameplay. Additionally, the server prevents spectators from sending messages by checking their status before processing any message payloads, and it shares extra details with spectators by selectively sending game state updates that include information not visible to regular participants, ensuring they remain informed without disrupting the game flow.

Saved: 5/10/2025 3:08:24 PM

100% Section #5: (1 pt.) Misc

100% Task #1 (0.33 pts.) - Github Details

Combo Task:

Weight: *33.33%*
Objective: *Github Details*

Image Prompt

Weight: *60%*

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

commit tab of the pull request

Saved: 5/10/2025 3:02:14 PM

Url Prompt

Weight: *40%*

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

URL #1
https://github.com/DhruvinRana/DBR2-IT11[illegible]/

URL
https://github.com/DhruvinRana/DB

Saved: 5/10/2025 3:02:14 PM

100%

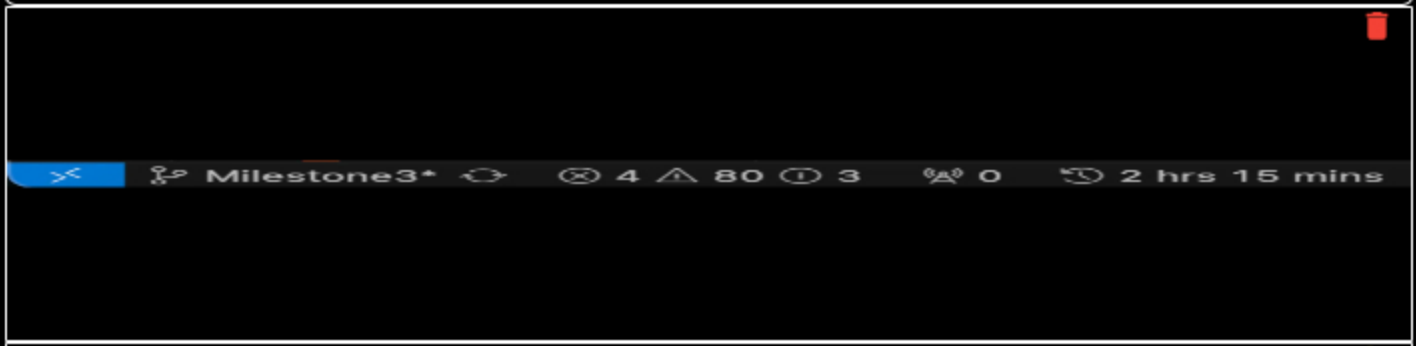
Task #2 (0.33 pts.) - WakaTime - Activity

Image Prompt

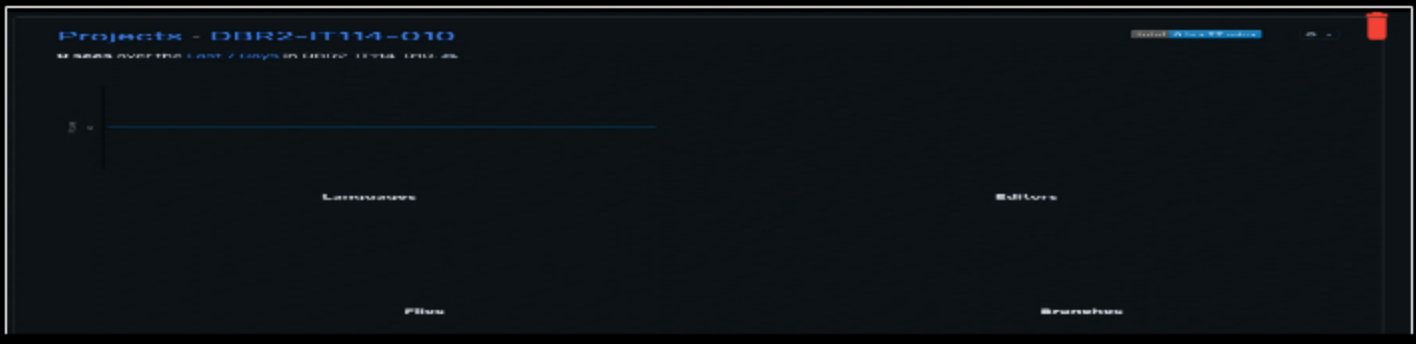
Weight: *33.33%*
Objective: *WakaTime - Activity*

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 is broken on the site.



100%

Task #3 (0.33 pts.) - Reflection

Weight: *33.33%*
Objective: *Reflection*

Sub-Tasks:

100%

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

Text Prompt

Weight: *33.33%*
Objective: *What did you learn?*

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

Your Response:

I learned about the implementation of spectator functionality in a multiplayer game application, including how the server manages spectator status, prevents them from participating in turns, and restricts their ability to send messages. Additionally, I gained insight into how the server selectively shares game information with spectators, enhancing their viewing experience without affecting the gameplay for active participants.

Saved: 5/10/2025 3:06:43 PM

100%

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

Text Prompt

Weight: *33.33%*
Objective: *What was the easiest part of the assignment?*

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

Your Response:

The easiest part of the assignment was understanding the basic structure of the client-server

communication, particularly how messages are sent and received between the client and server. Additionally, implementing the spectator functionality was straightforward due to the clear separation of roles and responsibilities in the codebase.

Saved: 5/10/2025 3:07:11 PM

100%

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

Text Prompt

Weight: *33.33%*
Objective: *What was the hardest part of the assignment?*

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

Your Response:

The hardest part of the assignment was ensuring that the spectator functionality was seamlessly integrated without disrupting the existing game logic, particularly in managing how spectators receive updates while preventing them from interfering with gameplay. Additionally, debugging issues related to message handling and ensuring proper synchronization between the server and multiple clients added complexity to the implementation.

Saved: 5/10/2025 3:07:57 PM