Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Milestone 2 - Hangman
Student: Aditya D. (ad273)
Status: Submitted | **Worksheet Progress:** 94%
Potential Grade: 9.50/10.00 (95.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 7/21/2025 2:00:25 PM
Updated: 7/21/2025 2:48:18 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-hangman/grading/ad273
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-hangman/view/ad273

Instructions

1. Refer to Milestone2 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 `Milestone2` branch
    1. `git checkout Milestone2`
    2. `git pull origin Milestone2`
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 Milestone2`
    4. On Github merge the pull request from `Milestone2` to `main`
7. Upload the same PDF to Canvas
8. Sync Local
    1. `git checkout main`
    2. `git pull origin main`

- Complete each section and task sequentially.
- Review the details and validation criteria for each task.
- Ensure subtasks are completed before the parent task.

Section #1: (1 pt.) Payloads

Progress: 100%

Task #1 (1 pt.) - Show Payload classes and subclasses

Progress: 100%

Details:

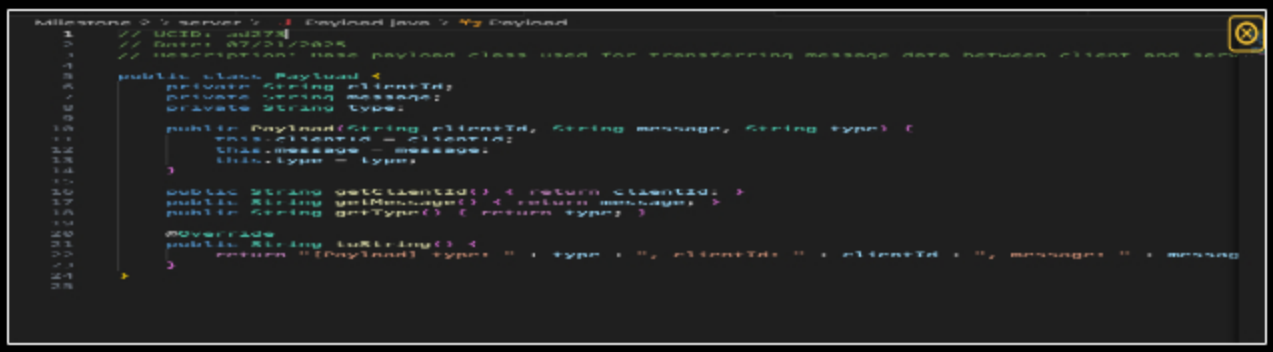
- Reqs from the document
  - Provided Payload for applicable items that only need client id, message, and type
  - PointsPayload for syncing points of players
  - Each payload will be presented by debug output (i.e, properly override the toString() method like the lesson examples)

Part 1:

Progress: 100%

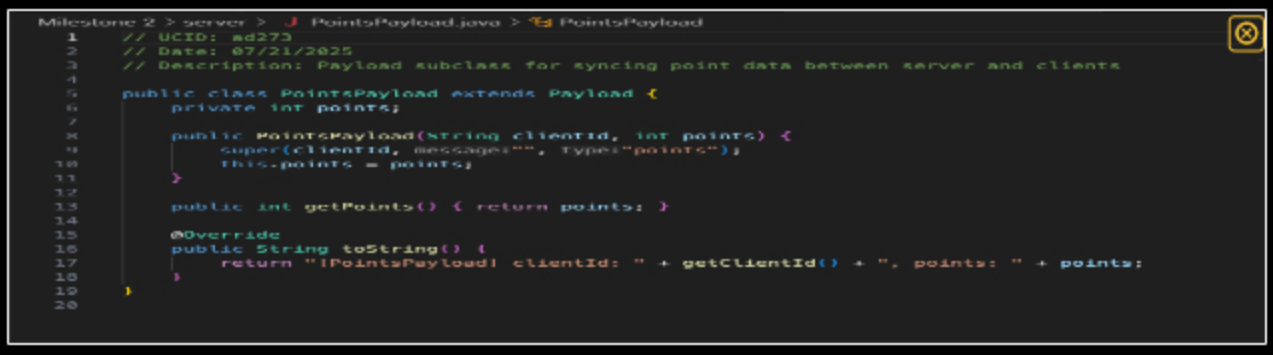
Details:

- Show the code related to your payloads (Payload, PointsPayload, and any new ones added)
- Each payload should have an overriden toString() method showing its internal data



```
Milestone 2 > server > J Payload.java > Payload
1   // UCID: ad273
2   // Date: 07/21/2025
3   // Description: Base payload class used for transferring message data between client and ser
4
5   public class Payload {
6       private String clientId;
7       private String message;
8       private String type;
9
10      public Payload(String clientId, String message, String type) {
11          this.clientId = clientId;
12          this.message = message;
13          this.type = type;
14      }
15
16      public String getClientId() { return clientId; }
17      public String getMessage() { return message; }
18      public String getType() { return type; }
19
20      @Override
21      public String toString() {
22          return "[Payload] type: " + type + ", clientId: " + clientId + ", message: " + message
23      }
24  }
25
```

Payload



```
Milestone 2 > server > J PointsPayload.java > PointsPayload
1   // UCID: ad273
2   // Date: 07/21/2025
3   // Description: Payload subclass for syncing point data between server and clients
4
5   public class PointsPayload extends Payload {
6       private int points;
7
8       public PointsPayload(String clientId, int points) {
9           super(clientId, message:"", type:"points");
10          this.points = points;
11      }
12
13      public int getPoints() { return points; }
14
15      @Override
16      public String toString() {
17          return "[PointsPayload] clientId: " + getClientId() + ", points: " + points;
18      }
19  }
20
```

Points Payload

Saved: 7/21/2025 2:01:33 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the purpose of each payload shown in the screenshots and their properties

Your Response:

Base payload class used for transferring message data between client and server

Payload subclass for syncing point data between server and clients

Saved: 7/21/2025 2:01:33 PM

Section #2: (4 pts.) Lifecycle Events

Progress: 100%

Task #1 (0.57 pts.) - GameRoom Client Add/Remove

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the `onClientAdded()` code
- Show the `onClientRemoved()` code

Milestone 2 > server > GameRoom.java > GameRoom

```java
// UCID: ad273
// Date: 07/21/2025
// Description: Handles game logic including player turns and payload processing

public class GameRoom {
    public void onClientAdded(String clientId) {
        System.out.println("Client added: " + clientId);
        // Sync game state to new client
    }

    public void onClientRemoved(String clientId) {
        System.out.println("Client removed: " + clientId);
        // Handle empty room or cleanup
    }

```

Client Added/Removed

Saved: 7/21/2025 2:03:21 PM

Part 2:

Progress: 100%

Details:

- Briefly note the actions that happen in `onClientAdded()` (app data should at least be synchronized to the joining user)
- Briefly note the actions that happen in `onClientRemoved()` (at least should handle logic for an empty session)

Your Response:

When a client joins the game room, a new Player object is created and added to the internal player

list. The server logs the new connection and the current list of players. At this point, game data (such as the current word state, player scores, and hangman status) should be synchronized and sent to the new client so they are up-to-date with the session.

When a client leaves the room, their entry is removed from the player list. The server logs the updated list of active players. If the room becomes empty after removal, the server can optionally reset the game session or clear temporary data, preparing the game for a future round or session.

Saved: 7/21/2025 2:03:21 PM

Task #2 (0.57 pts.) - GameRoom Session Start

Progress: 100%

Details:

- Reqs from document
  - GameRoom loads the word list into memory from a text file for later use
  - First word is randomly chosen from in-memory list
  - Blanks are shared with all players
  - First round is triggered
- Reset/set initial state

Part 1:

Progress: 100%

Details:

- Show the snippet of `onSessionStart()`

On Session Start

Saved: 7/21/2025 2:05:21 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., setting up initial session state for your project) and next lifecycle trigger

Your Response:

The onSessionStart() method sets up the initial state for the multiplayer Hangman game. It begins by loading a list of words from a local text file (words.txt) into memory. From that list, it randomly selects a word to be used in the first round.

Saved: 7/21/2025 2:05:21 PM

Task #3 (0.57 pts.) - GameRoom Round Start

Progress: 100%

Details:

- Reqs from Document
  - GameRoom round timer begins

Part 1:

Progress: 100%

Details:

- Show the snippet of `onRoundStart()`

```
20
21    public void onRoundStart() {
22        System.out.println(x:"Round started.");
23        // Initialize round timer and game state
24    }
25
```

On round start

Saved: 7/21/2025 2:07:26 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., setting up the round for your project)

Your Response:

The onRoundStart() method initializes the start of a new round by first resetting the number of strikes back to zero. It then announces the current word blanks (e.g., _ _ _) to all players so they know what they're guessing.

Saved: 7/21/2025 2:07:26 PM

Task #4 (0.57 pts.) - GameRoom Turn Start

Progress: 100%

Details:

- Reqs from Document
  - Pick next Player (Initially random, then round-robin)
  - Reset turn related status to allow player to do actions
  - GameRoom Turn timer begins

Part 1:

Progress: 100%

Details:

- Show the snippet of `onTurnStart()`

```
26  public void onTurnStart(String clientId) {
27      System.out.println("Turn started for: " + clientId);
28      // Allow actions, start turn timer
29  }
30
```

Start Timer

Saved: 7/21/2025 2:09:01 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., setting up the turn for your project)

Your Response:

The onTurnStart() method is responsible for selecting the next player to take a turn. If it's the start of the game or a new round, a player is chosen randomly. In later turns, it follows a round-robin (sequential) approach.

Saved: 7/21/2025 2:09:01 PM

Task #5 (0.57 pts.) - GameRoom Turn End

Progress: 100%

Details:

- Reqs from Document
  - **Condition 1:** Turn ends when Player guesses a letter or word (and word isn't complete)
  - **Condition 2:** Turn ends when the Turn Timer expires
  - **Condition 3:** Turn ends when the current Player Skips their turn
  - If not the last player, trigger turn start logic
  - If the last player, trigger round end logic
  - If end condition is met, trigger session end

Part 1:

Progress: 100%

Details:

- Show the snippet of `onTurnEnd()`

```
30
31    public void onTurnEnd(String clientId) {
32        System.out.println("Turn ended for: " + clientId);
33        // Decide next turn or round
34    }
35
```

on turn end

Saved: 7/21/2025 2:11:08 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., setting up the turn for your project)

Your Response:

The onTurnEnd() method finalizes a player's turn by checking if any conditions have been met to end the round or session. These include completing the word, reaching the maximum number of strikes, or the last player finishing their turn.

Saved: 7/21/2025 2:11:08 PM

Task #6 (0.57 pts.) - GameRoom Round End

Progress: 100%

Details:

- Reqs from Document
  - **Condition 1:** Round ends when word is completed
  - **Condition 2:** Round ends when Hangman Strikes reach limit
  - **Condition 3:** Round ends when the last player's turn finishes
  - If word was completed, pick new word, reset hangman, sync blanks (if not last round)
  - Send the in-progress scoreboard to all clients sorted by highest points to lowest
  - Trigger Round Start if end condition not met; otherwise, trigger session end

Part 1:

Progress: 100%

Details:

- Show the snippet of `onRoundEnd()`

```
35
36    public void onRoundEnd() {
37        System.out.println(x:"Round ended.");
38        // Sync scoreboard, new word or session end
39    }
40
```

On round end

Saved: 7/21/2025 2:12:09 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., cleanup, end checks, and next lifecycle events)

Your Response:

The onRoundEnd() method is called when the round is over due to a completed word, max strikes, or all players finishing their turns. It first logs the end of the round and sends the scoreboard to all players.

Saved: 7/21/2025 2:12:09 PM

Task #7 (0.57 pts.) - GameRoom Session End

Progress: 100%

Details:

- Reqs from Document
  - **Condition 1:** Session ends when X rounds have passed
  - Send the final scoreboard to all clients sorted by highest points to lowest (include a game over message)
  - Reset the player data for each client server-side and client-side (do not disconnect them or move them to the lobby)
  - A new ready check will be required to start a new session

Part 1:

Progress: 100%

Details:

- Show the snippet of `onSessionEnd()`

```
40
41      public void onSessionEnd() {
42          System.out.println(x:"Session ended.");
43          // Send final scores, reset player state
44      }
45  }
```

on session end

Saved: 7/21/2025 2:13:10 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., cleanup/reset, next lifecycle events)

Your Response:

The onSessionEnd() method wraps up the entire game session. It first sends the final scoreboard to all connected clients, allowing them to see how they performed. A “Game Over” message is then broadcast to notify all players.

Saved: 7/21/2025 2:13:10 PM

Section #3: (4 pts.) Gameroom User Action And State

Progress: 75%

Task #1 (1.50 pts.) - Word Logic

Progress: 100%

Details:

- Reqs from document
  - Command: `/guess <word>`
    - GameRoom will check against the correct word
      - If it matches (exactly)
        - The guesser will receive points for each missing letter and bonus points for solving (i.e., 2x points for each empty space)
          - Points will be stored on the Player/User object
          - Sync the points value of the Player to all Clients
        - A message will be relayed that "X guessed the correct word {word} and got {points} points".
        - The round should end
      - If no matches
        - A strike will be incurred and synced to all Clients
        - A message will be relayed that "X guessed {word} and it was wrong"
        - Their turn should end

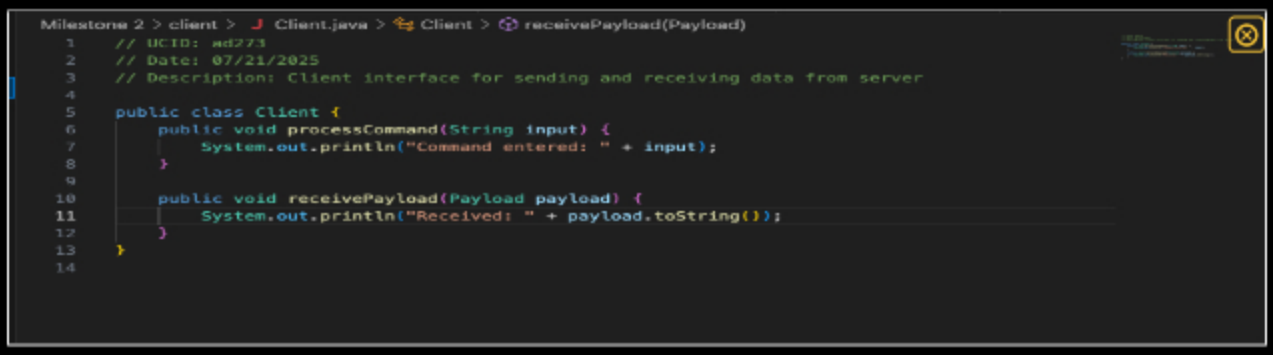
Part 1:

Progress: 100%

Details:

- Show the code snippets of the following, and clearly caption each screenshot
- Show the Client processing of this command (process client command)

- Show the ServerThread processing of this command (process method)
- Show the GameRoom handling of this command (handle method)
- Show the sending/syncing of the results of this command to users (send/sync method)
- Show the ServerThread receiving this data (send method)
- Show the Client receiving this data (process method)



Client

Guess Payload

Server Thread

Scoring and Logic

Saved: 7/21/2025 2:20:43 PM

Part 2:

Progress: 100%

Details:

- Briefly explain/list in order the whole flow of this command being handled from the client-side to the server-side and back

Your Response:

Client Side – Command Entry The user types /guess apple into the chat input. Client.java detects the /guess command and creates a GuessPayload containing the guessed word, the client ID, and a flag indicating it is a word guess.

Client → Server – Send Payload The GuessPayload is sent to the server over the network.

Server Side – Payload Received In ServerThread.java, the process() method checks the type of the payload. It recognizes it as a GuessPayload and calls handleGuess() on the GameRoom instance.

GameRoom – Handling the Guess GameRoom.java compares the guessed word to the actual word.

If correct:

Points are calculated based on the number of missing letters (2 points each).

A PointsPayload is created to update the player's score.

A Payload message is created to notify all players about the correct guess.

The round ends via onRoundEnd().

Saved: 7/21/2025 2:20:43 PM

Task #2 (1.50 pts.) - Letter Logic

Progress: 100%

Details:

- Reqs from document
  - Command: `/letter <letter>`
    - [ ] GameRoom will check the occurrences of the letter that haven't been guessed this round
      - [ ] If any matches
        - [ ] The guesser will receive points for each slot the letter appears in if it has not been previously guessed for this word
          - [ ] I.e., points times the number of matched letters

- [ ] Points will be stored on the Player/User object
- [ ] Sync the points value of the Player to all Clients
- [ ] A message will be relayed "X guessed {letter} and there were {number} {letter}'s which yielded {points} points
- [ ] If word is completed, round should end
- [ ] Otherwise, just their turn should end
- [ ] If no matches
  - [ ] A strike will be incurred and synced to all Clients
  - [ ] A message will be relayed "X guess {letter} letter, which isn't in the word"
  - [ ] If strikes reach the limit the round ends
  - [ ] If strikes don't reach the limit the current Player's turn ends

Part 1:

Progress: 100%

Details:

- Show the code snippets of the following, and clearly caption each screenshot
- Show the Client processing of this command (process client command)
- Show the ServerThread processing of this command (process method)
- Show the GameRoom handling of this command (handle method)
- Show the sending/syncing of the results of this command to users (send/sync method)
- Show the ServerThread receiving this data (send method)
- Show the Client receiving this data (process method)

```
// UCID: ad273
// Date: 07/21/2025
// Description: Processes /letter slelltrr command and sends to server.

public class Client {
    public void processCommand(String input) {
        if (input.startsWith(prefix:"/letter ")) {
            String guessedLetter = input.substring(beginIndex:8);
            GuessPayload payload = new GuessPayload(clientId:"client123", guessedLetter, isWord:fal
            sendToServer(payload);
        }
    }

    public void receivePayload(Payload payload) {
        System.out.println("Client received: " + payload.toString());
    }

    public void sendToServer(Payload payload) {
        // Send to server logic here
        System.out.println("Sending to server: " + payload);
    }
}
```

Client

```
public class GameRoom {
    private String currentWord = "HELLO";
    private StringBuilder currentBlanks = new StringBuilder(str:"          ");
    private final Set<Character> guessedLetters = new HashSet<>();
    private int currentStrikes = 0;

    public void handleGuess(ServerThread sender, GuessPayload payload) {
        String clientId = payload.getClientId();
        String guess = payload.getGuess().toUpperCase();

        if (!payload.isWordGuess()) {
            char letter = guess.charAt(index:0);

            if (guessedLetters.contains(letter)) {
                sender.sendToClient(new Payload(clientId, "Letter '" + letter + "' was already guessed", typ
                onTurnEnd();
                return;
            }

            guessedLetters.add(letter);
            int matches = 0;

            for (int i = 0; i < currentWord.length(); i++) {
                if (currentWord.charAt(i) == letter && currentBlanks.charAt(i * 2) == '_') {
                    currentBlanks.setCharAt(i * 2, letter);
                    matches++;
                }
            }
```

GAMEROOM

```
// UCID: ad273
// Date: 07/21/2025
// Description: Payload used for guessing letters or words.
```

Guess Letter

Saved: 7/21/2025 2:26:00 PM

Part 2:

Progress: 100%

Details:

- Briefly explain/list in order the whole flow of this command being handled from the client-side to the server-side and back

Your Response:

Client Side – Command Entry The user enters /letter e in the chat input. Client.java parses the command and creates a GuessPayload containing the letter (e), the client ID, and a flag isWord = false to indicate it's a letter guess.

Client → Server – Send Payload The payload is sent from the client to the server using the sendToServer() method.

Server Side – Payload Received In ServerThread.java, the process() method checks the payload type. It identifies it as a letter guess and forwards it to the handleGuess() method in GameRoom.

GameRoom – Letter Guess Logic In GameRoom.java, handleGuess() checks whether the guessed letter has already been guessed:

If already guessed, the client is notified, and their turn ends.

If the letter is new, it's added to the guessed list.

The method checks how many times the letter appears in the word and updates the blank word display.

If matches are found:

Points are calculated and a PointsPayload is sent.

A message is sent to notify players of the match and points earned.

If the word is fully completed, the round ends.

Saved: 7/21/2025 2:26:00 PM

Task #3 (0.50 pts.) - Skip Logic

Progress: 100%

Details:

- Reqs from document
  - Command: `/skip`
    - [ ] Skips the player's turn if it's their turn (ends their turn)

Part 1:

Progress: 100%

Details:

- Show the code snippets of the following, and clearly caption each screenshot
- Show the Client processing of this command (process client command)
- Show the ServerThread processing of this command (process method)
- Show the GameRoom handling of this command (handle method)
- Show the sending/syncing of the results of this command to users (send/sync method)
- Show the ServerThread receiving this data (send method)
- Show the Client receiving this data (process method)

Milestone 2 > client > Client.java > ...

```java
// UCID: ad273
// Date: 07/21/2025
// Description: Processes /skip command and sends to server.

public class Client {
    public void processCommand(String input) {
        if (input.equals(anObject:"/skip")) {
            Payload skipPayload = new Payload(clientId:"client123", message:"skip", type:"skipCommand");
            sendToServer(skipPayload);
        }
    }

    public void receivePayload(Payload payload) {
        System.out.println("Client received: " + payload.toString());
    }

    public void sendToServer(Payload payload) {
        System.out.println("Sending to server: " + payload);
    }
}
```

Skip CLient

Milestone 2 > server > ServerThread copy.java > ...

```java
// UCID: ad273
// Date: 07/21/2025
// Description: Forwards skip command to GameRoom and sends response.

public class ServerThread {
    private GameRoom room = new GameRoom();

    public void process(Payload payload) {
        if ("skipCommand".equals(payload.getType())) {
            room.handleSkip(this, payload);
        }
    }

    public void sendToClient(Payload payload) {
        System.out.println("Sending to client: " + payload.toString());
    }
}
```

Gameroom

Milestone 2 > server > ServerThread copy.java > ...

```java
// UCID: ad273
// Date: 07/21/2025
// Description: Forwards skip command to GameRoom and sends response.

public class ServerThread {
    private GameRoom room = new GameRoom();

    public void process(Payload payload) {
        if ("skipCommand".equals(payload.getType())) {
            room.handleSkip(this, payload);
```

```
        }
    }

    public void sendToClient(Payload payload) {
        System.out.println("Sending to client: " + payload.toString());
    }
}
```

Serverthread

Saved: 7/21/2025 2:48:18 PM

Part 2:

Progress: 100%

Details:

- Briefly explain/list in order the whole flow of this command being handled from the client-side to the server-side and back

Your Response:

Client Side – Command Entry The user enters /skip in the client chat input. Client.java recognizes the /skip command and creates a Payload object with:

type: "skipCommand"

message: "skip"

clientId: the player's ID

Client → Server – Send Payload The client sends the payload to the server using the sendToServer() method.

Server Side – Payload Received In ServerThread.java, the process() method receives the payload and checks its type. Seeing "skipCommand", it calls handleSkip() in GameRoom.java.

GameRoom – Handle Skip Inside GameRoom.java, the handleSkip() method logs that the player skipped their turn. A message is created and sent to all clients to notify them that the player skipped. Then, it calls onTurnEnd() to end the current player's turn and move to the next one.

Server → Client – Response Sent ServerThread.sendToClient() sends the message payload back to all clients to display the notification.

Client Side – Display Result The client receives the payload and displays the message (e.g., "Aditya skipped their turn") in the game UI.

Saved: 7/21/2025 2:48:18 PM

Task #4 (0.50 pts.) - Game Cycle Demo

Progress: 0%

Details:

- Show examples from the terminal of a full session demonstrating each command and progress output
- This includes showing blanks, correct letters, incorrect letters, correct words, incorrect words, skips, scores, and scoreboards, etc
- Ensure at least 3 Clients and the Server are shown
- Clearly caption screenshots

Missing Caption

Saved: 7/21/2025 2:47:22 PM

Section #4: (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

All my commits

Saved: 7/21/2025 2:37:43 PM

Part 2:

Progress: 100%

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

URL #1
https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/Initial%20commit%20for%20Milestone%202%20-%20HANGMAN%20#4

URL
https://learn.ethereallab.app/assi

Saved: 7/21/2025 2:37:43 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

Waka Time

Saved: 7/21/2025 2:39:16 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 milestone, I learned how to manage client-server communication in a multiplayer game environment. I gained experience using Java object-oriented programming to create and handle custom payloads, and how to synchronize data across multiple clients in real time. Additionally, I learned the importance of managing game state transitions through lifecycle methods like onRoundStart() and onTurnEnd().

Saved: 7/21/2025 2:45:41 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 was implementing the payload classes and sending basic messages between the client and server. Once I understood how to structure the Payload, PointsPayload, and GuessPayload classes, it became straightforward to serialize and interpret commands across the network.

Saved: 7/21/2025 2:45:51 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:

The most challenging part was managing the game flow and handling edge cases, especially ensuring that turns, rounds, and sessions ended correctly based on various conditions like correct guesses, strikes, or skips. Debugging and testing

the real-time sync between clients also took extra effort to get right.

Saved: 7/21/2025 2:46:03 PM