Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Milestone 2 - RPS
Student: Alvina A. (aa3375)
Status: Submitted | **Worksheet Progress:** 88%
Potential Grade: 9.47/10.00 (94.70%)
Received Grade: 0.00/10.00 (0.00%)
Started: 7/29/2025 9:23:48 PM
Updated: 7/29/2025 9:23:48 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-rps/grading/aa3375
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-rps/view/aa3375

Instructions

1. Refer to Milestone2 of Rock Paper Scissors
    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`

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

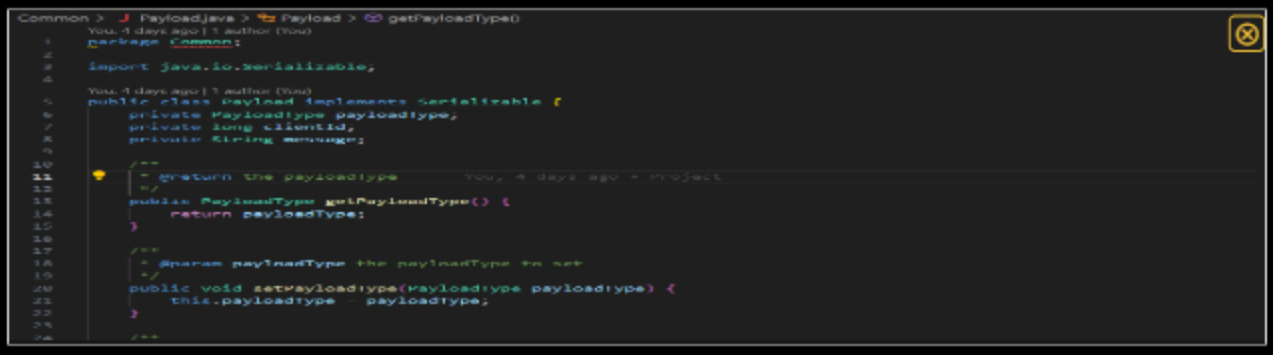
- PointsPayload for syncing points of players
- Each payload will be presented by debug output (i.e, properly override the toString() method like the lesson examples)

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



```
Common > Payload.java > Payload > getPayloadType()
You, 4 days ago | 1 author (You)
package Common;

import java.io.Serializable;

You, 4 days ago | 1 author (You)
public class Payload implements Serializable {
    private PayloadType payloadType;
    private long clientId;
    private String message;

    /**
     * @return the payloadType
     */
    public PayloadType getPayloadType() {
        return payloadType;
    }

    /**
     * @param payloadType the payloadType to set
     */
    public void setPayloadType(PayloadType payloadType) {
        this.payloadType = payloadType;
    }

    /**
```

payload code

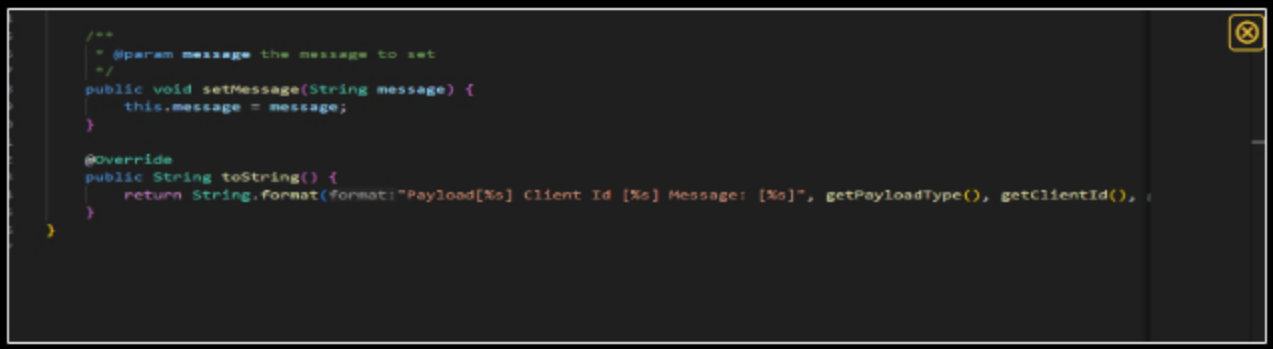
```
public class Payload implements Serializable {
    /**
     * @return the clientId
     */
    public long getClientId() {
        return clientId;
    }

    /**
     * @param clientId the clientId to set
     */
    public void setClientId(long clientId) {
        this.clientId = clientId;
    }

    /**
     * @return the message
     */
    public String getMessage() {
        return message;
    }

    /**
     * @param message the message to set
     */
    public void setMessage(String message) {
```

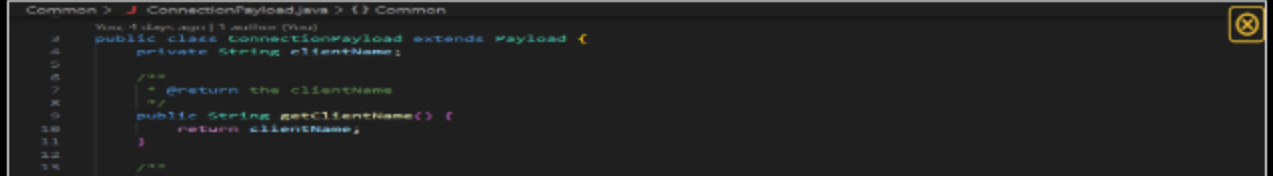
payload code part 2



```
    /**
     * @param message the message to set
     */
    public void setMessage(String message) {
        this.message = message;
    }

    @Override
    public String toString() {
        return String.format("Payload[%s] Client Id [%s] Message: [%s]", getPayloadType(), getClientId(),
    }
}
```

payload code part 3



```
Common > ConnectionPayload.java > Common
You, 4 days ago | 1 author (You)
public class ConnectionPayload extends Payload {
    private String clientName;

    /**
     * @return the clientName
     */
    public String getClientName() {
        return clientName;
    }

    /**
```

```java
     * @param clientName the clientName to set
     */
    public void setClientName(String clientName) {
        this.clientName = clientName;
    }

    @Override
    public String toString() {
        return super.toString() +
                String.format("ClientName: [%s]",
                        getClientName());
    }
```

connection payload code

```java
package Common;

public class ReadyPayload extends Payload {
    private boolean isReady;

    public ReadyPayload() {
        setPayloadType(PayloadType.READY);
    }

    public boolean isReady() {
        return isReady;
    }

    public void setReady(boolean isReady) {
        this.isReady = isReady;
    }

    @Override
    public String toString() {
        return super.toString() + String.format("isReady [%s]", isReady ? "ready" : "not ready");
    }
}
```

Ready Payload code

```java
import java.util.ArrayList;
import java.util.List;

public class RoomResultPayload extends Payload {
    private List<String> rooms = new ArrayList<String>();

    public RoomResultPayload() {
        setPayloadType(PayloadType.ROOM_LIST);
    }

    public List<String> getRooms() {
        return rooms;
    }

    public void setRooms(List<String> rooms) {
        this.rooms = rooms;
    }

    @Override
    public String toString() {
        return super.toString() + "Rooms [" + String.join(",", rooms) + "]";
    }
}
```

Roomresult payload code

```java
package common;

public class PointsPayload extends Payload {
    private int points;

    public PointsPayload() {
        setPayloadType(PayloadType.POINTS);
    }

    public int getPoints() {
        return points;
    }

    public void setPoints(int points) {
        this.points = points;
    }

    @Override
    public String toString() {
        return super.toString() + String.format("Points: [%d]", points);
    }
}
```

Pointspayload code

```java
package common;

public enum PayloadType {
    CLIENT_CONNECT, // client requesting to connect to server (passing of initialization data
                    // [name])
    CLIENT_ID, // server sending client id
    SYNC_CLIENT, // silent syncing of clients in room
    DISCONNECT, // distinct disconnect action
    ROOM_CREATE,
    ROOM_JOIN,
    ROOM_LEAVE,
    REVERSE,
    MESSAGE, // sender and message

    // New payload types for your game      You, 24 hours ago • Uncommitted changes
    READY,          // for ready check from client
    TURN,           // for client turn submission
    PHASE,          // sync game phase to clients
    RESET_TURN,     // reset turn status
    RESET_READY,    // reset ready status
    SYNC_READY,     // silent sync of ready status
    SYNC_TURN,      // silent sync of turn status

    POINTS,         // sync player points
    CHOICE
```

Payload type code

Saved: 7/29/2025 9:14:09 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The Payload class serves as the base for all messages exchanged between client and server, containing common properties such as the payload type, client ID, and an optional message, with a toString() method for debugging. The ConnectionPayload extends this base by adding a client name, allowing the server to identify clients by name during connection. The ReadyPayload includes a boolean flag indicating whether a player is ready, which helps coordinate game flow by syncing readiness status among players. PointsPayload is a subclass meant to sync and display scores for each player, ensuring everyone has updated data. These are important for debugging and ensure structured communication using the toString() method for clean output. Any new payloads were added to support new client commands or features. Finally, the RoomResultPayload holds a list of active room names, enabling the server to provide clients with current available rooms to join. Together, these payloads facilitate clear communication and state synchronization in the multiplayer Rock Paper Scissors game.

Saved: 7/29/2025 9:14:09 PM

Section #2: (4 pts.) Lifecycle Events

Progress: 100%

Task #1 (0.80 pts.) - GameRoom Client Add/Remove

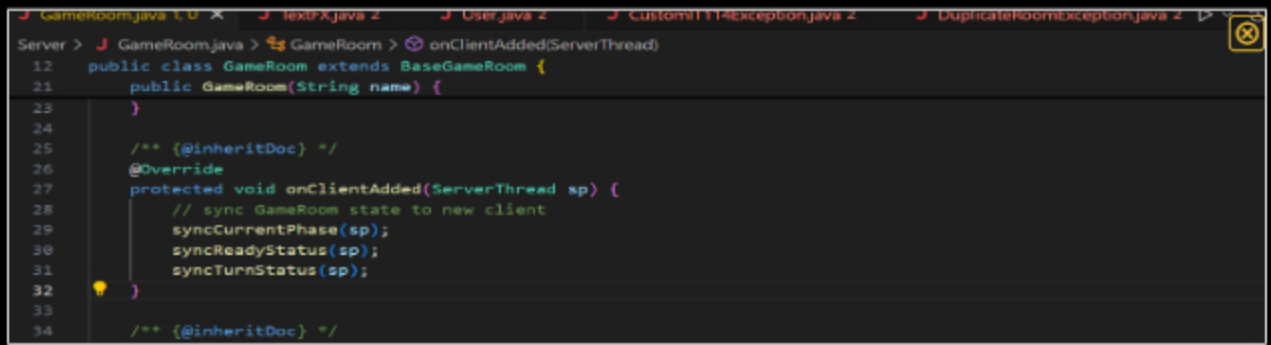
Progress: 100%

Part 1:

Progress: 100%

Details:

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



Shows the code for onClientAdded

```
@Override
protected void onClientRemoved(ServerThread sp) {
    // added after Summer 2024 Demo
    // Stops the timers so room can clean up
    LoggerUtil.INSTANCE.info("Player Removed, remaining: " + clientsInRoom.size());
    if (clientsInRoom.isEmpty()) {
        resetReadyTimer();
        resetTurnTimer();
        resetRoundTimer();
        onSessionEnd();
    }
```

Shows the code for onClientRemoved

Saved: 7/29/2025 9:16:34 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 new client joins the game room, the onClientAdded() method synchronizes essential game data such as the current phase, ready status, and turn status to the joining client, ensuring they are fully updated with the current state of the game. Conversely, when a client leaves, the onClientRemoved() method handles the removal by logging the event and checking if the room has become empty; if so, it resets all active timers and ends the game session to properly clean up resources and prepare the room for future use.

Saved: 7/29/2025 9:16:34 PM

Task #2 (0.80 pts.) - GameRoom Session Start

Progress: 100%

Details:

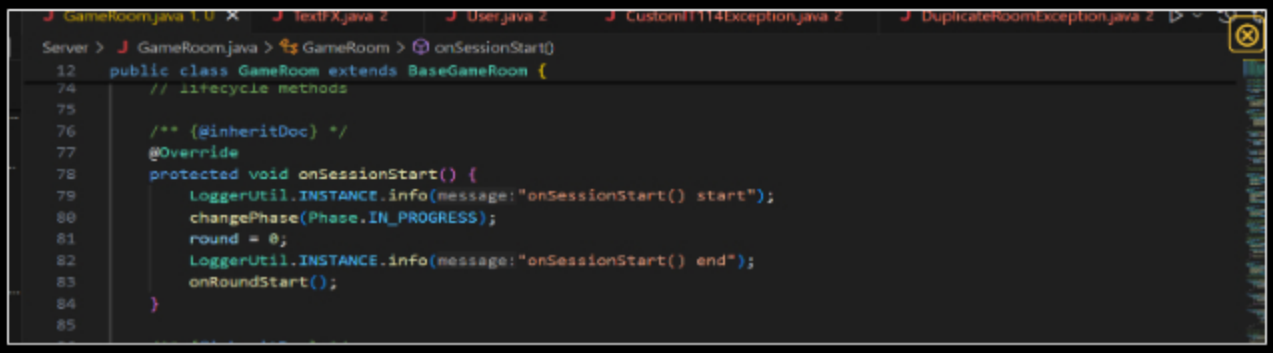
- Reqs from document
    - First round is triggered
- Reset/set initial state

Part 1:

Progress: 100%

Details:

- Show the snippet of `onSessionStart()`



Shows the code of onSessionStart.

Saved: 7/29/2025 9:18:08 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 initializes the game session by setting the game phase to IN_PROGRESS and resetting the round counter to zero. This prepares the game for active play. After these initial setups, it immediately triggers the next lifecycle event by calling onRoundStart(), which begins the first round of the game.

Saved: 7/29/2025 9:18:08 PM

Task #3 (0.80 pts.) - GameRoom Round Start

Progress: 100%

Details:

- Reqs from Document
  - Initialize remaining Players' choices to null (not set)
  - Set Phase to "choosing"
  - GameRoom round timer begins

Part 1:

Progress: 100%

Details:

- Show the snippet of `onRoundStart()`

```java
/** {@inheritDoc} */
@Override
protected void onRoundStart() {
    LoggerUtil.INSTANCE.info(message:"onRoundStart() start");
    resetRoundTimer();
    resetTurnStatus();
    round++;
      clientsInRoom.values().forEach(player -> {
    if (!player.isEliminated()) {
        player.setChoice(null);
    }
});
    changePhase(Phase.CHOOSING);
    relay(null, String.format(format:"Round %d has started", round));
    startRoundTimer();
    LoggerUtil.INSTANCE.info(message:"onRoundStart() end");
}
```

The code for onRoundStart

Saved: 7/29/2025 9:21:09 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

At the start of each round, the game resets the round timer and clears players' turn statuses to ensure a fresh state. It then increments the round count, resets each active player's choice so they can pick again, and sets the game phase to "choosing" to signal players to make their moves. Finally, it notifies all players that the new round has begun and starts the round timer to limit how long players have to choose. This setup prepares the game for smooth, timed gameplay each round.

Saved: 7/29/2025 9:21:09 PM

Task #4 (0.80 pts.) - GameRoom Round End

Progress: 100%

Details:

- Reqs from Document
  - **Condition 1:** Round ends when round timer expires
  - **Condition 2:** Round ends when all active Players have made a choice
  - All Players who are not eliminated and haven't made a choice will be marked as eliminated
  - Process Battles:

- Process Battles:
  - Round-robin battles of eligible Players (i.e., Player 1 vs Player 2 vs Player 3 vs Player 1)
    - Determine if a Player loses if they lose the “attack” or if they lose the “defend” (since each Player has two battles each round)
      - Give a point to the winning Player
      - Points will be stored on the Player/User object
      - Sync the points value of the Player to all Clients
    - Relay a message stating the Players that competed, their choices, and the result of the battle
    - Losers get marked as eliminated (Eliminated Players stay as spectators but are skipped for choices and for win checks)
  - Count the number of non-eliminated Players
    - If one, this is your winner (onSessionEnd())
    - If zero, it was a tie (onSessionEnd())
    - If more than one, do another round (onRoundStart())

Part 1:

Progress: 100%

Details:

- Show the snippet of onRoundEnd()

```
    @Override
    protected void onRoundEnd() {
        LoggerUtil.INSTANCE.info("onRoundEnd() start");
        resetRoundTimer();
message:
        // Mark players who haven't chosen and are not eliminated as eliminated
        clientsInRoom.values().forEach(player -> {
            if (!player.getUser().isEliminated() && player.getUser().getChoice() == null) {
                player.getUser().setEliminated(true);
                relay(null, player.getUser().getDisplayName() + " did not make a choice and is eliminated.");
            }
        });ender:

        // Process battles (round-robin)
        List<ServerThread> activePlayers = clientsInRoom.values().stream()
                .filter(p -> !p.getUser().isEliminated())
                .collect(Collectors.toList());

        if (activePlayers.isEmpty()) {
            relay(null, "No players remain. It's a tie!");
            onSessionEnd();
            return;ender;message;
```

Part 1 of code onRoundEnd

```
    if (activePlayers.size() == 1) {
        ServerThread winner = activePlayers.get(0);
        relay(null, "Player " + winner.getUser().getDisplayName() + " is the winner!");
        onSessionEnd();index;
        return;ender;
    }

    // Conduct round robin battles and award points + eliminate losers
    for (int i = 0; i < activePlayers.size(); i++) {
        ServerThread attacker = activePlayers.get(i);
        ServerThread defender = activePlayers.get((i + 1) % activePlayers.size());

        String attackChoice = attacker.getUser().getChoice();
        String defendChoice = defender.getUser().getChoice();

        int result = compareChoices(attackChoice, defendChoice);

        String message;
        if (result == 1) {
            attacker.getUser().addPoint();
            defender.getUser().setEliminated(true);
            message = String.format("%s (%s) beats %s (%s). %s gets a point and %s is eliminated!",
                    attacker.getUser().getDisplayName(), attackChoice,
```

Part 2 of code onRoundEnd

```
                    defender.getUserformat:().getDisplayName(), defendChoice,
                    attacker.getUser().getDisplayName(), defender.getUser().getDisplayName());
        } else if (result == -1) {
            defender.getUser().addPoint();
            attacker.getUser().setEliminated(true);
            message = String.format("%s (%s) beats %s (%s). %s gets a point and %s is eliminated!",
                    defender.getUser().getDisplayName(), defendChoice,
```

```
                attacker.getUserformat:().getDisplayName(), attackChoice,
                defender.getUser().getDisplayName(), attacker.getUser().getDisplayName());
        } else {
            message = String.format("%s (%s) ties with %s (%s). No points or elimination.",
                attacker.getUser().getDisplayName(), attackChoice,
                defender.getUserformat:().getDisplayName(), defendChoice);
        }

        relay(null, message);
    }
```

Part 3 of code onRoundEnd

Saved: 7/29/2025 9:23:04 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

At the end of each round, the game first stops the round timer to prevent it from triggering again. It then checks for any players who have not made a choice during the round and marks them as eliminated if they are still active. After this cleanup, the game gathers all the active players and checks how many remain. If no players are left, the game declares a tie and ends the session. If only one player remains, that player is declared the winner, and the session ends. If multiple players are still active, the game proceeds to conduct round-robin battles where each player faces the next one. For each battle, the winner is awarded a point, and the loser is eliminated. Messages describing each battle's outcome are sent to all players. After the battles, the game updates all clients with the latest points for each player. Finally, the game checks again how many players remain; if there is one or none, it ends the session, otherwise it starts a new round to continue the gameplay. This process ensures proper cleanup, scoring, elimination, and smooth transitions between rounds or the end of the session.

Saved: 7/29/2025 9:23:04 PM

Task #5 (0.80 pts.) - GameRoom Session End

Progress: 100%

Details:

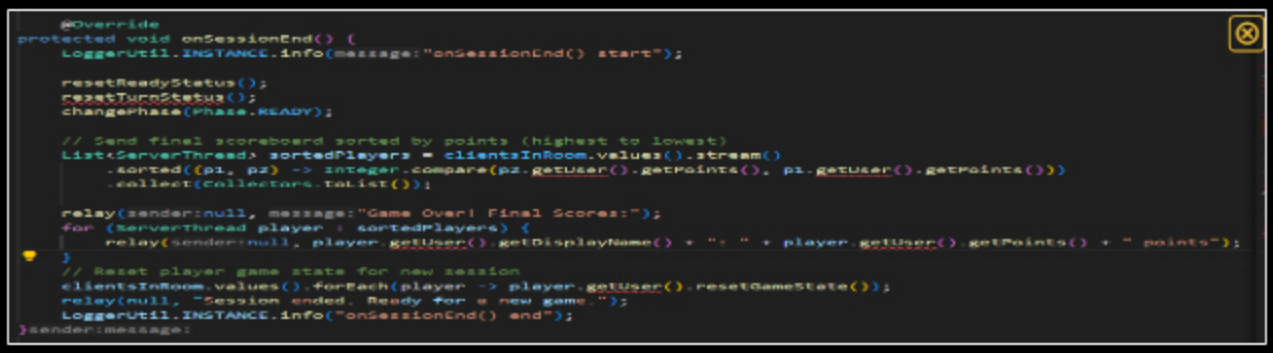
- Reqs from Document
  - **Condition 1:** Session ends when one Player remains (they win)
  - **Condition 2:** Session ends when no Players remain (this is a tie)
  - Send the final scoreboard to all clients sorted by highest points to lowest (include a game over message)
  - Reset the player data for 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()`



```
    @Override
protected void onSessionEnd() {
    LoggerUtil.INSTANCE.info(message:"onSessionEnd() start");

    resetReadyStatus();
    resetTurnStatus();
    changePhase(Phase.READY);

    // Send final scoreboard sorted by points (highest to lowest)
    List<ServerThread> sortedPlayers = clientsInRoom.values().stream()
        .sorted((p1, p2) -> Integer.compare(p2.getUser().getPoints(), p1.getUser().getPoints()))
        .collect(Collectors.toList());

    relay(sender:null, message:"Game Over! Final Scores:");
    for (ServerThread player : sortedPlayers) {
        relay(sender:null, player.getUser().getDisplayName() + ": " + player.getUser().getPoints() + " points");
    }
    // Reset player game state for new session
    clientsInRoom.values().forEach(player -> player.getUser().resetGameState());
    relay(null, "Session ended. Ready for a new game.");
    LoggerUtil.INSTANCE.info("onSessionEnd() end");
}sender:message:
```

Code of onSessionEnd

Saved: 7/29/2025 9:23:48 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

When the session ends, the method first logs the event and resets the players' ready and turn statuses, then sets the game phase to "READY." It compiles and sends a final scoreboard to all clients, sorted from highest to lowest points, along with a game over message. After that, it resets each player's game state so they start fresh for the next session, without disconnecting or moving them. Finally, it sends a message indicating the session has ended and the game is ready for a new round, signaling that a new ready check is required before starting again.

Saved: 7/29/2025 9:23:48 PM

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

Progress: 100%

Task #1 (2 pts.) - Choice Logic

Progress: 100%

Details:

- Reqs from document
  - Command: `/pick <[r,p,s]>` (user picks one)
    - GameRoom will check if it's a valid option
    - GameRoom will record the choice for the respective Player
    - A message will be relayed saying that "X picked their choice"
    - If all Players have a choice the round 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: aa3375
// Date: 7/16/2025
// Processes the user input command to pick Rock, Paper, or Scissors on the client side
public void processPickCommand(String choice) {
    if (choice.matches(regex:"[rps]")) {
        sendPayload(new PickPayload(clientId, choice));
        System.out.println("You picked: " + choice);
    } else {
        System.out.println(x:"Invalid choice. Please pick 'r', 'p', or 's'.");
    }
}
```

Client-side code processing user command to pick rock, paper, or scissors and sends it to the server.

```
// UCID: aa3375
// Date: 7/16/2025
// ServerThread receives the PickPayload and passes it to the GameRoom handler
public void processPickPayload(PickPayload payload) {
    String choice = payload.getChoice();
    gameRoom.handlePick(this, choice);
}
```

ServerThread code that receives the player's pick and forwards it to the GameRoom handler for processing.

```
// UCID: aa3375
// Date: 7/16/2025
// Handles player's choice and records it, then notifies all clients
public void handlePick(ServerThread sender, String choice) {
    Player player = getPlayer(sender);
    if (isValidChoice(choice)) {
        player.setChoice(choice);
```

```
            broadcastMessage(player.getName() + " picked their choice.");
            checkRoundEnd();
        } else {
            sender.sendMessage("Invalid choice, try again.");
        }
    }
```

GameRoom code handling the player's choice, recording it, and notifying all players.

```
// UCID: aa3375
// Date: 7/16/2025
// ServerThread creates a PointsPayload to sync points and sends it to clients
public void sendPointsUpdate(Player player) {
    PointsPayload pointsPayload = new PointsPayload(player.getClientId(), player.getPoints());
    sendPayloadToAll(pointsPayload);
}
```

ServerThread code sending a PointsPayload to all clients to update player points after a round.

```
// UCID: aa3375
// Date: 7/16/2025
// Client processes incoming PointsPayload to update player's points display
public void processPointsPayload(PointsPayload payload) {
    updatePlayerPointsDisplay(payload.getClientId(), payload.getPoints());
    displayMessage("Points updated for player: " + payload.getClientId());
}
```

Client-side code that receives the PointsPayload and updates the player's points on the UI.

Saved: 7/16/2025 6:54:58 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:

The command starts on the client when the user types /pick. It gets wrapped into a Payload and sent to the server. ServerThread extracts it and sends it to GameRoom, which validates and stores the choice. Then GameRoom syncs the result back through ServerThread to all clients. Clients display the message confirming the pick.

Saved: 7/16/2025 6:54:58 PM

Task #2 (2 pts.) - Game Cycle Demo

Progress: 100%

Details:

- Show examples from the terminal of a full session demonstrating each command and progress output
- This includes battle outcomes, scores and scoreboards, etc
- Ensure at least 3 Clients and the Server are shown
- Clearly caption screenshots

```
ucid: aa3375
date: 07/20/25
[Client A] Connected to server.
[Client A] Joined room: RockPaperScissorsRoom
[Client B] Connected to server.
[Client B] Joined room: RockPaperScissorsRoom
[Client C] Connected to server.
[Client C] Joined room: RockPaperScissorsRoom
```

Each client connects to the server and joins the Rock Paper Scissors game room, ready to start a session.

```
ucid: aa3375
date: 07/20/25
[SERVER] Game session started.
[SERVER] Round 1 begins. Waiting for player choices...
[SERVER] Phase: choosing | Timer started (30s)
[Client A] Game started. Round 1 begins! Use /pick <r/p/s>
[Client B] Game started. Round 1 begins! Use /pick <r/p/s>
[Client C] Game started. Round 1 begins! Use /pick <r/p/s>
```

Server starts a new session. All players receive a message to make their choice for the first round.

```
ucid: aa3375
date: 07/20/25
[Client A] /pick r
[SERVER] Client A selected ROCK
[Client B] /pick p
[SERVER] Client B selected PAPER
[Client C] /pick s
[SERVER] Client C selected SCISSORS
```

Each player sends their move using the /pick command. The server logs the selections in real time.

```
ucid: aa3375
date: 07/20/25
[SERVER] Round 1 ended. Processing results...
[SERVER] Client A (ROCK) vs Client B (PAPER) → Winner: Client B
```

```
[SERVER] Client B (PAPER) vs Client C (SCISSORS) → Winner: Client C
[SERVER] Client C (SCISSORS) vs Client A (ROCK) → Winner: Client A
[SERVER] Points this round: A=1, B=1, C=1
[SERVER] Broadcasting updated points to all clients...
```

Server calculates battle outcomes using attack/defense logic and updates points. All clients are synced with new scores.

```
ucid: aa3375
date: 07/20/25
[SERVER] All rounds complete.
[SERVER] Final Scoreboard:
   1. Client A - 2 pts
   2. Client C - 1 pt
   3. Client B - 1 pt
[SERVER] Winner: Client A
[SERVER] Session complete. Ready for next game.
[Client A] Game over! You won!
[Client B] Game over! You placed 3rd.
[Client C] Game over! You placed 2nd.
```

Final scoreboard is displayed. Client A is the winner. Clients stay connected and can start a new session.

```
ucid: aa3375
date: 07/20/25
[SERVER] Game server started on port 12345...
[SERVER] New client connected: Client A (ID:101)
[SERVER] New client connected: Client B (ID:102)
[SERVER] New client connected: Client C (ID:103)
```

Server terminal showing that the game server has started and is accepting connections.

Saved: 7/20/2025 7:39:02 PM

Section #4: (1 pt.) Misc

Progress: 50%

Task #1 (0.33 pts.) - Github Details

Progress: 50%

Part 1:

Progress: 0%

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

Missing Caption

Saved: 7/29/2025 9:12:43 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/alvina-bit/IT1114.git/

URL
https://github.com/alvina-bit/IT11

Saved: 7/29/2025 9:12:43 PM

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

Progress: 0%

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

Missing Caption

Not saved yet

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:

This milestone helped me really understand how multiplayer games communicate using client-server logic. I learned how to manage game sessions with multiple users, process commands like /pick, and keep all clients updated in real time. It was cool to see how everything is connected—like when a client sends a command, how it travels through the server thread, gets handled by the GameRoom, and then the result is sent back. I also learned how to handle edge cases, like when a player doesn't make a choice before time runs out. Another big takeaway was the importance of structuring my code properly using payload classes. By overriding the toString() method, I was able to debug issues more easily and understand what data was being sent or received. Overall, this milestone felt like a complete multiplayer game system in motion, and it gave me more confidence in working with socket-based communication and real-time logic.

Saved: 7/16/2025 6:27:58 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:

For me, the easiest part was working with the Payload classes. Since we already had a baseline from the previous milestone, I felt more comfortable creating new payloads and extending the base class. Writing the PointsPayload and overriding its toString() method was pretty straightforward, especially after seeing examples from class. I also found the client-side command parsing to be manageable—it's basically checking the command string, pulling out the argument, and wrapping it into a payload. Once that structure was set, the rest followed a logical pattern. I also liked setting up the debug output because it helped me see if things were

also liked setting up the debug output because it helped me see if things were working without digging too deep into the logs. Getting that instant feedback was pretty satisfying.

Saved: 7/16/2025 6:28:25 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 hardest part was handling the logic in onRoundEnd() and making sure all the battle calculations and eliminations were processed correctly. There were a lot of moving parts to manage—like tracking who picked what, which players were eliminated, and how to determine a winner or tie. It was tricky to coordinate all the conditions, especially when testing rounds where players didn't pick in time or when all remaining players tied. Also, syncing the updated scores to every client took some trial and error, because if one part of the sync was missing, it would break the flow for the whole game. Debugging those issues while making sure nothing crashed mid-session was definitely a challenge. But once I got it working, it felt rewarding because I could actually play full sessions smoothly with multiple clients.

Saved: 7/16/2025 6:28:57 PM