Submission Worksheet

Submission Data

Course: IT114-003-F2025
Assignment: IT114 - Milestone 3 - RPS
Student: Laura L. (lsl8)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 12/8/2025 11:25:16 PM
Updated: 12/11/2025 3:21:56 AM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-3-rps/grading/lsl8
View Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-3-rps/view/lsl8

Instructions

1. Refer to Milestone3 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 `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%

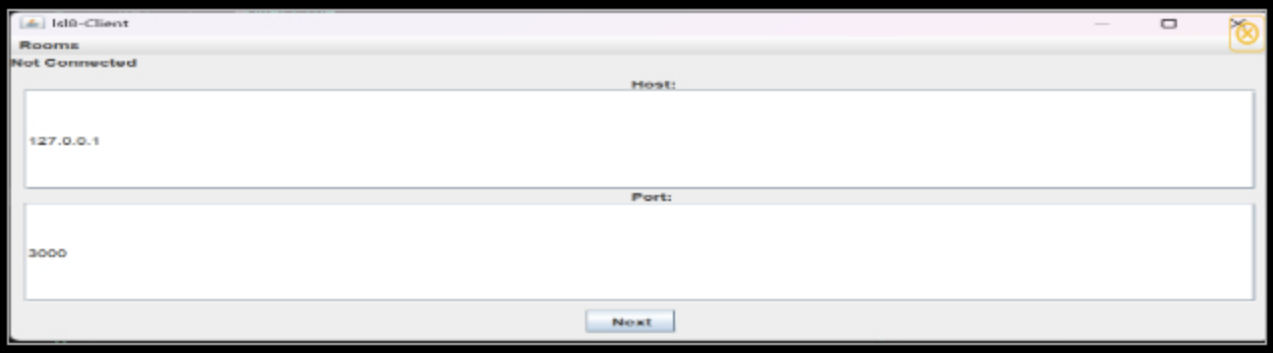
Part 1:

Progress: 100%

Details:

- Show the connection panel with valid data

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



Connection Panel

User panel

Saved: 12/8/2025 11:29:35 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:

To connect to the server, my program first shows two panels: the User Details panel and the Connection panel. On the User Details panel, I enter the username I want to use, and on the Connection panel I enter the port number. When I click the Connect button, the ClientUI.connect() method reads the values directly from those text fields and sends them into the Client.connect() method. The client stores the username, opens a socket to the host and port, starts its background listener thread, and immediately sends a CLIENT_CONNECT payload to the server with my chosen name. The server receives this message, saves the username, assigns me a unique client ID, and sends that back to the client. When the client receives the CLIENT_ID payload, it updates its internal user object and triggers a UI callback that switches the view from the connection panels to the main Chat/Game screen. So the flow is: user enters details - UI collects them - client connects with those details - server responds with an ID - UI transitions to the game screen

Saved: 12/8/2025 11:29:35 PM

Task #2 (0.50 pts.) - Ready Panel

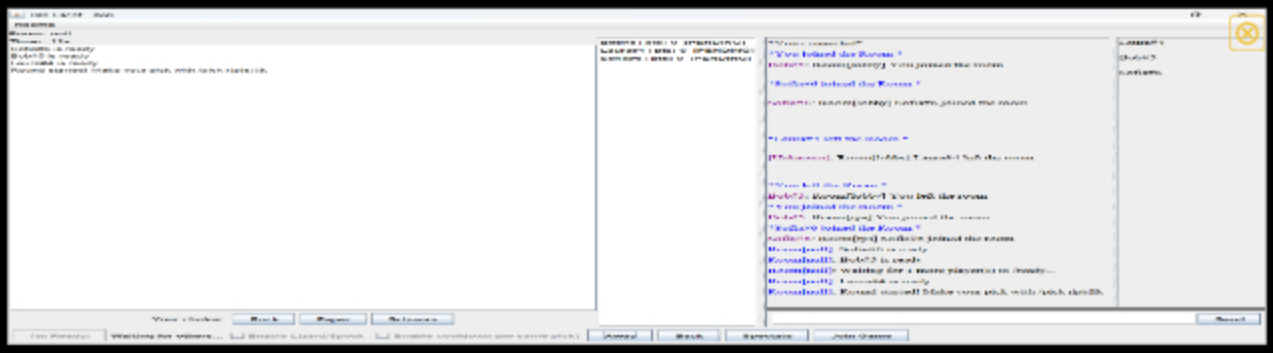
Progress: 100%

Part 1:

Progress: 100%

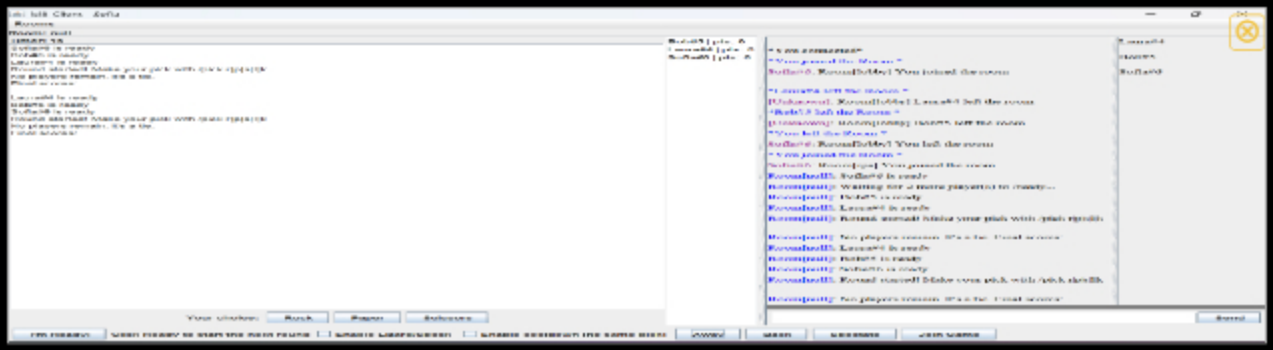
Details:

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



Ready button on the bottom left

All players ready



Ready button before players presses it

Saved: 12/8/2025 11:34:55 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:

Marking ready from the UI: When the user clicks the Ready button in the game UI, the button's action listener calls the sendReady() method inside the client code. This method creates a ReadyPayload object, sets its payload type to READY, marks the ready flag as true, and sends it to the server through the client's output stream. The server receives this payload inside ServerThread.processPayload(), recognizes the READY type, and passes it to GameRoom.handleReady(). Inside handleReady(), the server records that the specific player marked themselves as ready, broadcasts a message to all players saying "X is ready," and checks whether enough players are now ready to start the round. If not, the server responds back to the client with a message saying how many players are still needed

Receiving ready date and updating the UI: When the client receives a READY from the server, the message is processed inside Client.processPayload(), which detects the payload type and calls processReadyStatus(). In this method, the client updates its internal list of known players by setting that player's ready status to true. After updating the data model, the client triggers all UI callbacks registered under IReadyEvent.It prints a readable message in the Game Events panel such as "Laura is ready." This gives immediate visual feedback to the user that the READY state was received from the server and correctly reflected in the interface

Saved: 12/8/2025 11:34:55 PM

Section #2: (2 pts.) Project Ui

Progress: 100%

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

Progress: 100%

Details:

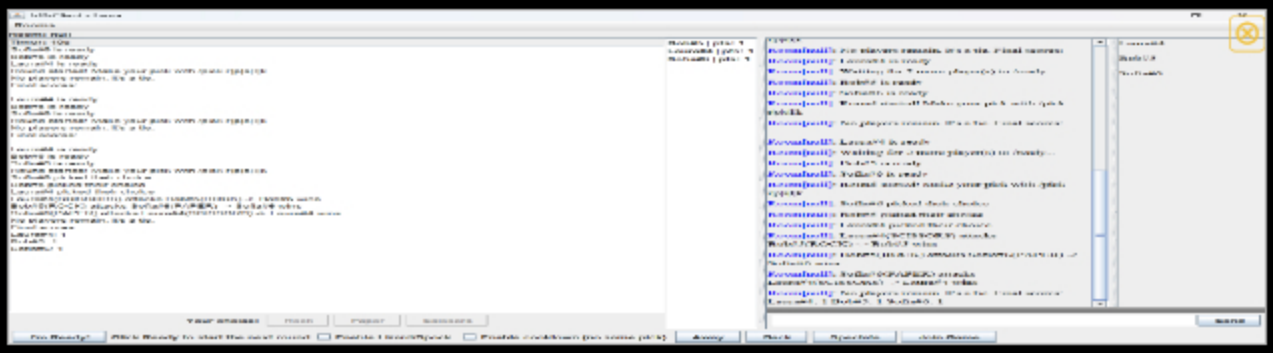
- Show the username and id of each user
- Show the current points of each user
- Users should appear in score order, sub-sort by name when ties occur
- Pending-to-pick users should be marked accordingly
- Eliminated users should be marked accordingly

Part 1:

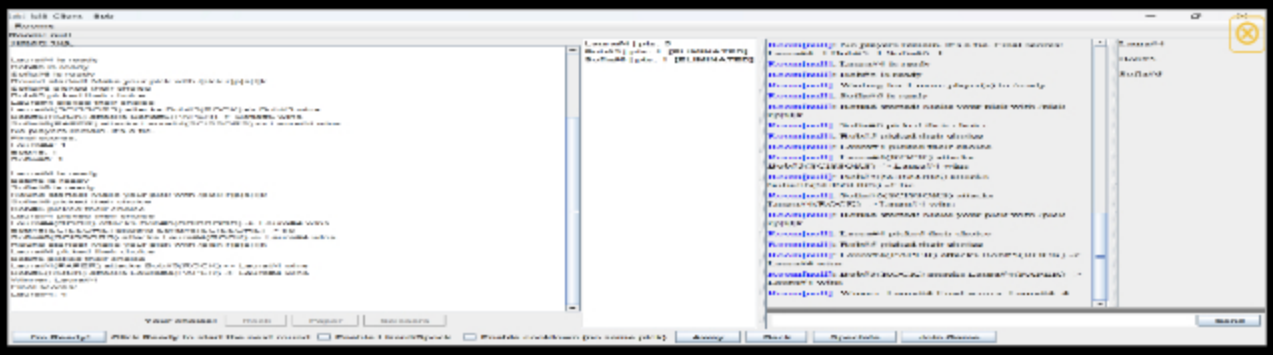
Progress: 100%

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 pending-to-pick indicators
  - Include code snippets showing the code flow for this from server-side to UI
- Show various examples of elimination indicators
  - Include code snippets showing the code flow for this from server-side to UI



Tie = everyone gets a point



Second round/ users sorted by points

```
//ls18 | 12/08/25        You, 1 second ago • Uncommitted changes
    private void syncAllPoints() {
        for (ServerThread st : clientsInRoom.values()) {
            long id = st.getClientId();

            PointsPayload p = new PointsPayload();
            p.setPayloadType(PayloadType.POINTS_SYNC);
            p.setClientId(Constants.DEFAULT_CLIENT_ID);
            p.setTargetClientId(id);
            p.setPoints(points.getOrDefault(id, 0));

            st.sendToClient(p);
        } <- #292-302 for (ServerThread st : clientsInRoom.values())
    } <- #291-303 private void syncAllPoints()
```

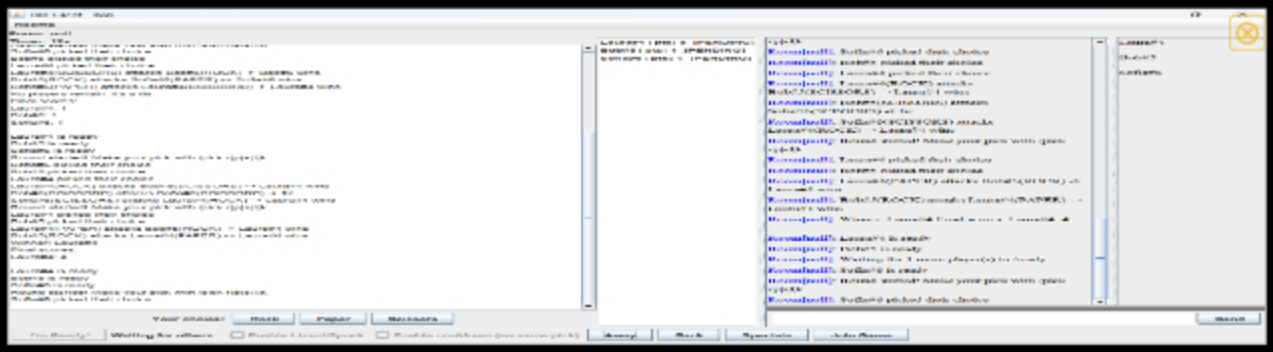
Syncing points

```
//ls10 | 12/08/25        You, 1 second ago • Uncommitted changes
    // Start processPoints() method
    private void processPoints(Payload payload) {
        if (!(payload instanceof PointsPayload)) {
            error("Invalid payload subclass for processCurdAdd");
            return;
        }
        PointsPayload pp = (PointsPayload) payload;
        long targetId   pp.getClientId();
        int points = pp.getPoints();
        if (targetId == Constants.DEFAULT_CLIENT_ID) {
            // reset all
            knownClients.values().forEach(cp   > cp.setPoints( 1));

            passToUICallback(IPointsEvent.class, e -> e.onPointsUpdate(Constants.DEFAULT_CLIENT_ID,
        } else if (knownClients.containsKey(targetId)) {
            knownClients.get(targetId).setPoints(points);

            passToUICallback(IPointsEvent.class, e -> e.onPointsUpdate(targetId, points));

        } <- #605 clo else if (knownClients.containsKey(targetId))
```

Updating + sorting

```java
//lsl8 12/08/25
    // ---------------- IPointsEvent ----------------
@Override
public void onPointsUpdate(long clientId, int points) {
    for (UserRow row : userRows) {
        if (row.clientId == clientId) {
            row.points = points;
            break;
        }
    } <- #423-428 for (UserRow row : userRows)
    // sort by points desc, then name asc
    userRows.sort(
        (a, b) -> {
            int cmp = Integer.compare(b.points, a.points);
            return (cmp != 0) ? cmp : a.name.compareToIgnoreCase(b.name);
        }
    ); <- #430-435 userRows.sort
    userTableModel.fireTableDataChanged();
} <- #422 437 public void onPointsUpdate(long clientId, int points)
```

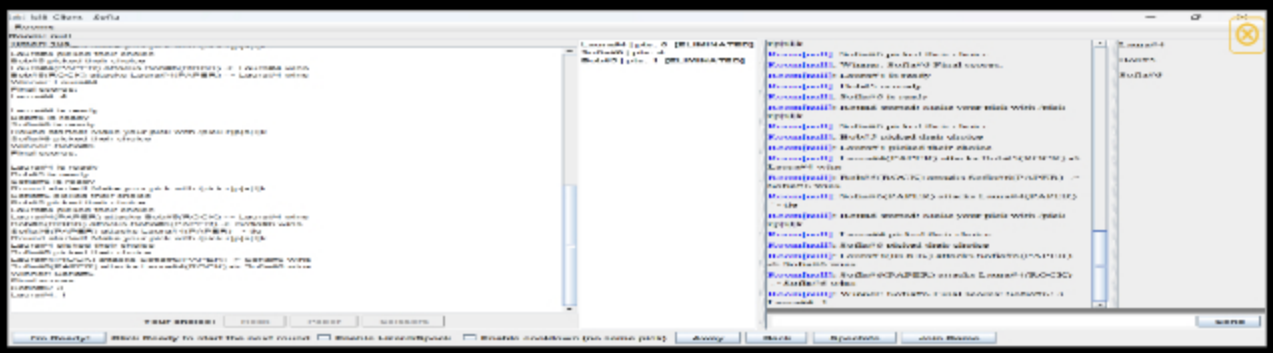
updating + sorting



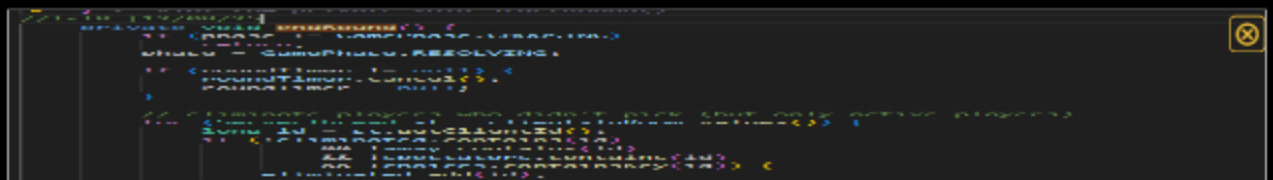
Game in session, players pending

```java
    } <- #231-250 private void refreshPlayerList()
//lsl8 | 12/08/25
    private String formatPlayerLine(PlayerInfo p) {
        StringBuilder sb = new StringBuilder();
        sb.append(p.name).append(" | pts: ").append(p.points);
        if (p.spectator) {
            sb.append("  [SPECTATOR]");
        } else if (p.eliminated) {
            sb.append("  [ELIMINATED]");
        } else if (p.away) {
            sb.append("  [AWAY]");
        } else if (inPickingPhase && !p.tookTurn) {
            sb.append("  [PENDING]");
        }
```

while picking = pending



two players eliminated after losing round



Server side endRound

Server attacks

client eliminated

Saved: 12/9/2025 12:05:34 AM

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 pending-to-pick indicators
- Briefly explain the code flow for server-side to UI of elimination indicators

Your Response:

1. In my project, each player's score is tracked on the server and then pushed out to all clients so the user list shows live points. On the server side, the GameRoom class keeps a Map<Long, Integer> points and updates it whenever someone wins an attack in resolveAttack(). At the end of each round, the server calls syncAllPoints(), which sends a POINTS_SYNC payload to each client containing the target player's id and current point total. On the client side, that payload is processed in Client.processPoints(), where I cast the incoming payload to a PointsPayload, update the corresponding User object in the

knownClients map, and then call the IPointsEvent callback. My GameView implements IPointsEvent.onPointsUpdate(long clientId, int points) and uses the updated knownClients data to rebuild the user list, sorting it by points (highest first) and then by name to break ties

2. For the pending-to-pick part, the server controls when a round starts and who has picked. On the server side, GameRoom.startRound() clears the choices map, sets the phase to CHOOSING, and sends a ROUND_START payload to everyone in the room. When each player calls /pick, the server handles it in handlePick(), where it parses the choice, saves it into the choices map, and then uses broadcastNotice() to send a PICKED_NOTICE payload with a message. On the client side, those special payloads are processed in Client.processPayload() and forwarded as game events by calling clientSideGameEvent(payload.getMessage())

3. Elimination is handled on the server using the eliminated set in GameRoom. When a round ends, endRound() walks through all the room's clients and automatically adds any active player who didn't pick to the eliminated set. In addition, inside resolveAttack(), if the attacker loses an attack, that attacker is also added to eliminated. After this, the server continues sending points and game events, but eliminated players are treated as "out" when counting active players and checking for game over. On the client side, even though there isn't a separate ELIMINATED payload, my GameView uses the information from the game events and the active-player logic to mark eliminated players visually. In the user list data model I keep a boolean flag like eliminated for each row; when I know a player has been eliminated for this session, I set that flag to true and render their row in gray with an "(Eliminated)" tag

Saved: 12/9/2025 12:05:34 AM

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

Progress: 100%

Details:

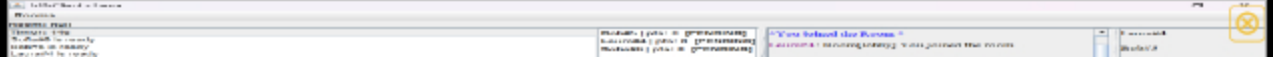
- Show the status of users picking choices
- Show the battle resolution messages from Milestone 2
    - Include messages about elimination
- Show the countdown timer for the round

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



Round Started!

```
//ls18 | 12/08/25 Game Starts        You, 1 second ago • Uncommitted changes
    private void startRound() {
        choices.clear();
        phase = GamePhase.CHOOSING;

        if (roundTimer != null)
            roundTimer.cancel();
        roundTimer = new Timer(true);

        roundTimer.schedule(new TimerTask() {
            @Override
            public void run() {
                synchronized (GameRoom.this) {
                    endRound(); // Condition 1: timer expires
                }
            } <- #196-200 public void run()
        }, 15000); // 15 second pick window <- #194-201 roundTimer.schedule

        clientsInRoom.values().forEach(ServerThread::sendRoundStart);
    } <- #187-204 private void startRound()
```

Round Start server side

```
    // ls18 | 12/08/25 | Round Start Message        You, 1 second ago • Uncommitted char
    protected boolean sendRoundStart() {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.ROUND_START);
        p.setClientId(Constants.DEFAULT_CLIENT_ID);
        p.setMessage("Round started! Make your pick with /pick r|p|s|l|k");
        return sendToClient(p);
    } <- #95-101 protected boolean sendRoundStart()
```

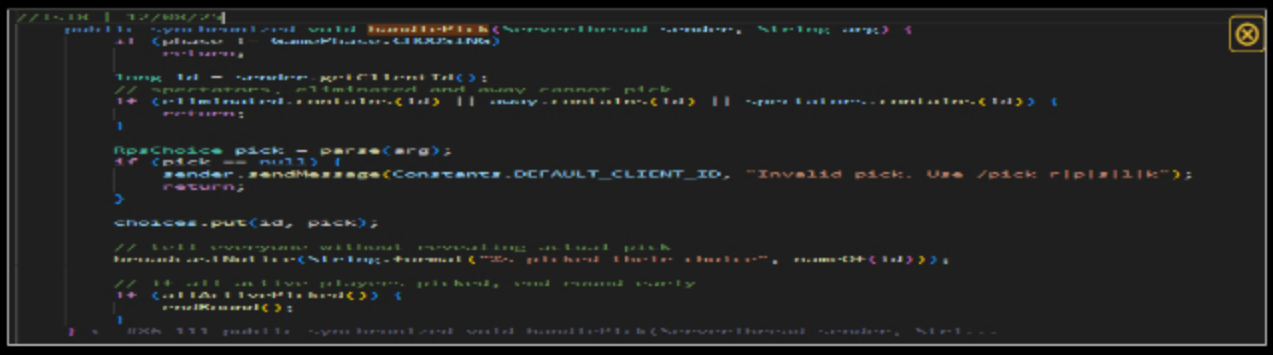
Round Start server side

```
    // ---------- M5 Milestone 2+3 specific payloads ----------
    //ls18 | 12/08/25 | Round Start : Picked : Result : Game Over messages        You,
    case ROUND_START:
        processMessage(payload);
        clientSideGameEvent(payload.getMessage());
    break;
    case PICKED_NOTICE:
        processMessage(payload);
        clientSideGameEvent(payload.getMessage());
    break;
    case BATTLE_RESULT:
        processMessage(payload);
        clientSideGameEvent(payload.getMessage());
    break;
    case GAME_OVER:
        processMessage(payload);
        clientSideGameEvent(payload.getMessage());
        processResetReady();
    break;
    case POINTS_SYNC:
        processPoints(payload);
```

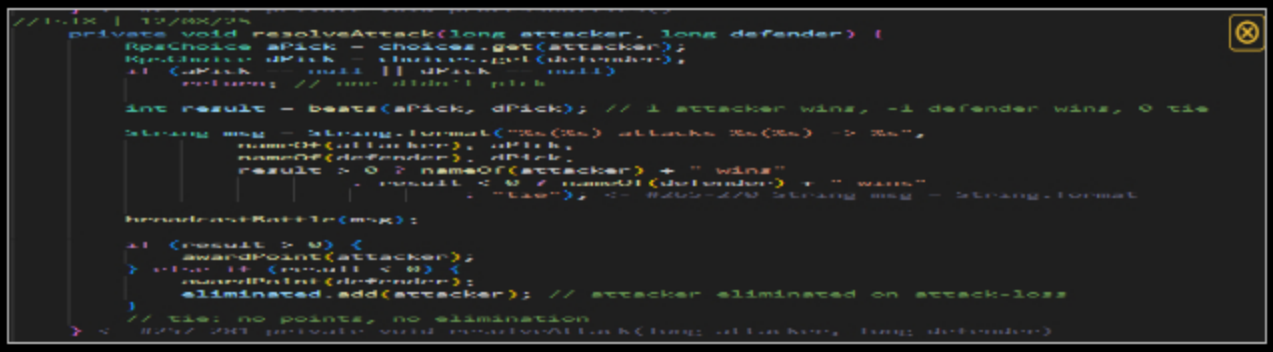
Round start client side

```
Room: null
Timer: 8s
Sofia#6 is ready
Bob#5 is ready
Laura#4 is ready
Round started! Make your pick with /pick r|p|s|l|k
Laura#4 picked their choice
Bob#5 picked their choice
Sofia#6 picked their choice
Laura#4(ROCK) attacks Bob#5(SCISSORS) -> Laura#4 wins
Bob#5(SCISSORS) attacks Sofia#6(ROCK) -> Sofia#6 wins
Sofia#6(ROCK) attacks Laura#4(ROCK) -> tie
Round started! Make your pick with /pick r|p|s|l|k
Sofia#6 picked their choice
Laura#4 picked their choice
Laura#4(ROCK) attacks Sofia#6(PAPER) -> Sofia#6 wins
Sofia#6(PAPER) attacks Laura#4(ROCK) -> Sofia#6 wins
Winner: Sofia#6
Final scores:
Sofia#6: 3
Laura#4: 1
```

Round Started + [User] picked their choice + Battle Log Messages + Winner UI Messages



"picked their choice" server side

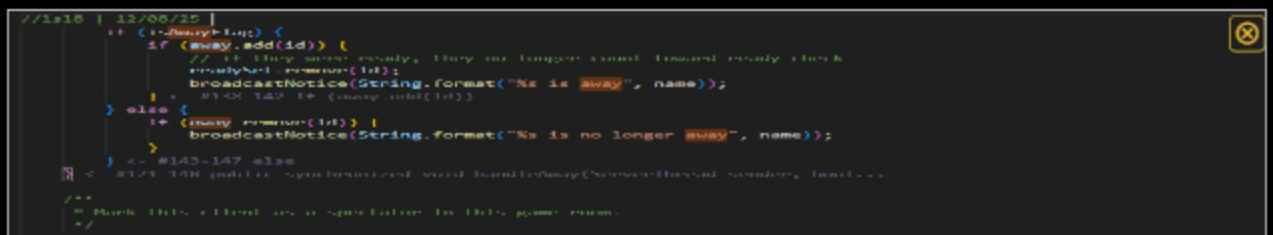


Battle Log messages server side

```
//ls18 | 12/08/25
    private void gameOver(String msg) {
        StringBuilder sb = new StringBuilder();
        sb.append(msg).append("\nFinal scores:\n");

        points.entrySet().stream()
                .sorted((a, b) -> Integer.compare(b.getValue(), a.getValue()))
                .forEach(e -> sb.append(String.format("%s: %d\n", nameOf(e.getKey()), e.getValue())));

        String text = sb.toString();
        clientsInRoom.values().forEach(st -> st.sendGameOver(text));
    } <- #305-315 private void gameOver(String msg)
```

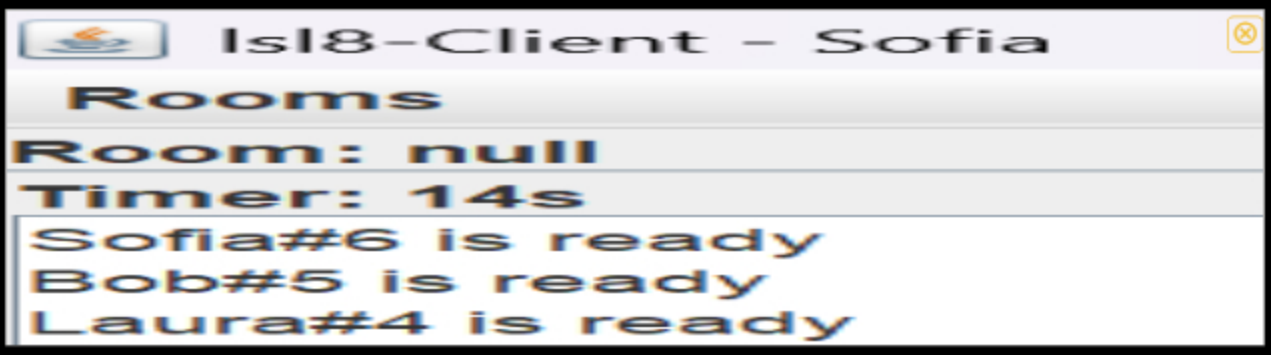
Final scores server side

Spectator + away messages



```
        long id = sender.getClientId();
        spectators.add(id);
        String name = nameOf(id);
        broadcastMsg(String.format("%s joined as a spectator", name));
        sender.sendMessage(Constants.DEFAULT_CLIENT_ID,
                "You joined as a spectator. You can see chat and game events but cannot ready or pick.");
    }
```

Spectator + Away server side



Timer

Saved: 12/11/2025 1:14:13 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

. All game-related messages start on the server and then travel through a defined payload system until they appear in the client's Game Events panel or user list. For example, when the game server wants to announce something such as a round starting, a player picking their choice, a battle result, a ready message, or away/spectator updates, it creates a message string and sends it inside a specific Payload type (like ROUND_START, PICKED_NOTICE, BATTLE_RESULT, or GAME_OVER). These messages are generated inside GameRoom methods such as startRound(), handlePick(), resolveAttack(), and also in spectator/away logic. Each message is sent by calling helper methods in ServerThread such as sendRoundStart(), sendPickedNotice(), or sendBattleResult(). On the client side, the incoming payload is received inside Client.listenToServer(), and then handled by Client.processPayload(). If the payload is a game-related message, the client passes the message into clientSideGameEvent(), which forwards it into the UI through the IMessageEvents callback

Saved: 12/11/2025 1:14:13 AM

Task #3 (0.67 pts.) - Game Area

Progress: 100%

Details:

- UI should have components to allow the user to select their choice

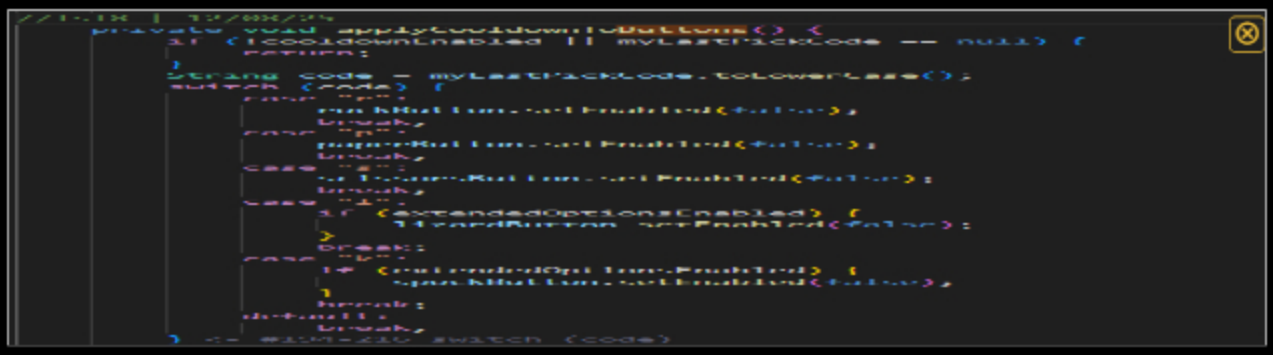
Part 1:

Progress: 100%

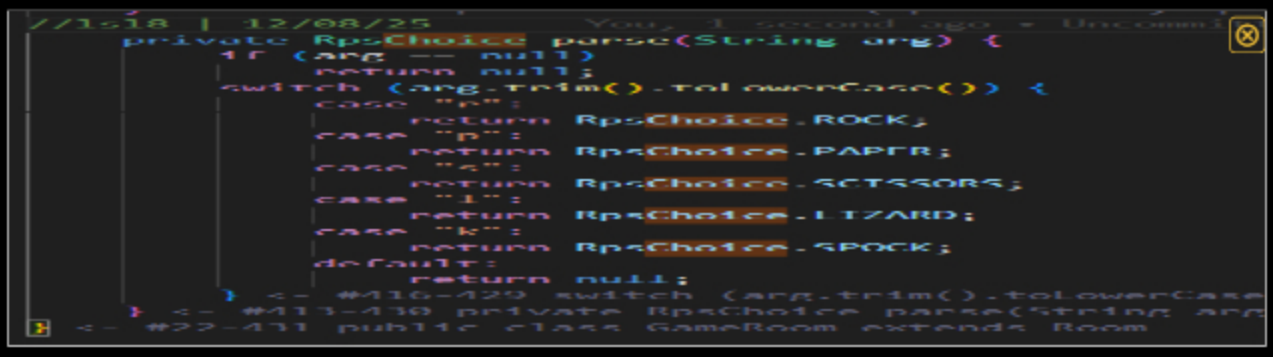
Details:

- Show various examples of selections across clients (3+ clients visible)
- Show the code related to sending choices upon selection
- Show the code related to showing visually what was selected

Game in process. Selection across clients [Laura, Bob, Sofia]



Client side buttons



Server side selection

```java
/**
 * RPS: send a choice (r/p/s/extended) to the server.
 *
 * @param choiceCode e.g. "r", "p", "s"
 * @throws IOException
 */
public void sendPick(String choiceCode) throws IOException {        You, 36 minutes
    Payload p = new Payload();
    p.setPayloadType(PayloadType.CHOICE_PICKED);
    p.setMessage(choiceCode);
```

```
    sendToServer(p);
} <- #328-333 public void sendPick(String choiceCode) throws IOException
```

From UI to Server

Saved: 12/11/2025 1:33:54 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for selecting a choice and having it reach the server-side
- Briefly explain the code flow for receiving the selection for the current player to update the UI

Your Response:

When the user clicks one of the choice buttons in the GameView (Rock, Paper, Scissors), the button's action listener calls my sendChoice() method, which then calls Client.INSTANCE.sendPick(code). Inside sendPick(), I create a Payload with type CHOICE_PICKED and attach the selected option (such as "r" or "s"). This payload is then sent over the socket to the server. On the server side, ServerThread.processPayload() receives the message and routes it to GameRoom.handlePick(). In this method, the server validates the pick (checking for spectators, eliminated players, away players, cooldown rules, and extended options), stores the selection in the choices map, and broadcasts a notice like "Laura picked their choice" to all clients. After the server logs a player's pick, it broadcasts a PICKED_NOTICE payload back to all connected clients. The client receives this in Client.processPayload(), which converts it into a UI event using clientSideGameEvent(). This forwards the message into GameView.onMessageReceive(), where I append the text into the Game Events panel.

Saved: 12/11/2025 1:33:54 AM

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

Progress: 100%

Task #1 (2 pts.) - Extra Choices

Progress: 100%

Details:

- Setting should be toggleable during Ready Check by session creator
  - (Option 1) Extra choices are available during the full session
  - (Option 2) Only activate extra options at different stages (i.e., last 3 players remaining)
- There should be at least 2 extra options for rps-5

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 the play screen with the extra options available
  - Show the related code for the UI and handling of these extra options (including battle logic)

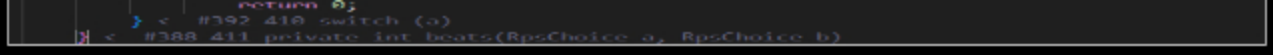
Lizard/Spock Toggle on

Game in session with extra feature

```java
private void initReadyPanel() {
    readyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

    readyButton = new JButton("I'm Ready!");
    readyStatusLabel = new JLabel("Not ready");
    extendedOptionsCheckbox = new JCheckBox("Enable Lizard/Spock");
    cooldownCheckbox = new JCheckBox("Enable cooldown (no same pick)");
    awayButton = new JButton("Away");
    backButton = new JButton("Back");
    spectateButton = new JButton("Spectate");
    joinGameButton = new JButton("Join Game");
```

checkbox set up

```java
//1518 | 12/08/25
        switch (a) {
            case ROCK:
                // Rock crushes Scissors & Lizard
                return (b == RpsChoice.SCISSORS || b == RpsChoice.LIZARD) ? 1 : -1;
            case PAPER:
                // Paper covers Rock & disproves Spock
                return (b == RpsChoice.ROCK || b == RpsChoice.SPOCK) ? 1 : -1;
            case SCISSORS:
                // Scissors cut Paper & decapitate Lizard
                return (b == RpsChoice.PAPER || b == RpsChoice.LIZARD) ? 1 : -1;
            case LIZARD:
                // Lizard poisons Spock & eats Paper
                return (b == RpsChoice.SPOCK || b == RpsChoice.PAPER) ? 1 : -1;
            case SPOCK:
                // Spock smashes Scissors & vaporizes Rock
                return (b == RpsChoice.SCISSORS || b == RpsChoice.ROCK) ? 1 : -1;
            default:
```



Battle logic

```
// Panel with all choice buttons
private JPanel buildPickButtonsPanel() {
    JPanel p = new JPanel(new FlowLayout(FlowLayout.CENTER));

    rockButton.addActionListener(e -> sendPickAndDisable("r"));
    paperButton.addActionListener(e -> sendPickAndDisable("p"));
    scissorsButton.addActionListener(e -> sendPickAndDisable("s"));
    // Extra options
    lizardButton.addActionListener(e -> sendPickAndDisable("l"));
    spockButton.addActionListener(e -> sendPickAndDisable("k"));

    p.add(new JLabel("Your choice: "));
    p.add(rockButton);
    p.add(paperButton);
    p.add(scissorsButton);
    p.add(lizardButton);
    p.add(spockButton);
```

Client to server

Saved: 12/11/2025 1:48:12 AM

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 these options including how it's handled during the battle logic
- Note which option you went with in terms of activating the choices

Your Response:

I chose option 1. The extra-options setting is controlled by a checkbox that the user can interact with during Ready Check. In the ChatGameView, the checkbox is created inside the Ready Panel and assigned an action listener. When the user clicks it, the listener calls into gameView.setExtendedOptionsEnabled(true or false) to enable or disable the additional RPS-5 buttons (Lizard and Spock) on all clients. Since Ready Check is the phase where game rules are configured, the toggle only appears and is interactable during Phase.READY. As soon as the round begins, the panel hides itself and the setting is locked in for that session.Once the user enables RPS-5 mode, the client UI exposes two additional pick buttons in GameView: Lizard and Spock. Each button sends a short code ("l" for Lizard, "k" for Spock) through Client.sendPick(), which creates a CHOICE_PICKED payload and sends it to the server. On the server side, the GameRoom.parse() method converts these characters into RpsChoice.LIZARD and RpsChoice.SPOCK. The battle resolution uses an expanded beats() method, where each of the 5 choices is compared using the official RPSLS rules (e.g., Lizard poisons Spock, Spock vaporizes Rock). The server uses this function inside the attack loop to determine round results, award points, and mark eliminated players.

Saved: 12/11/2025 1:48:12 AM

Task #2 (2 pts.) - Choice cooldown

Progress: 100%

Progress: 100%

Details:

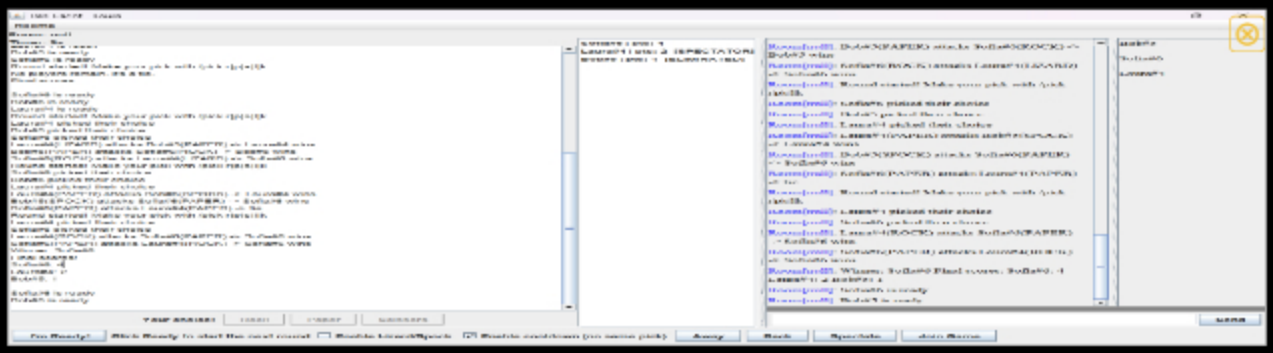
- Setting should be toggleable during Ready Check by session creator
- The choice on cooldown must be disable on the UI for the User

Part 1:

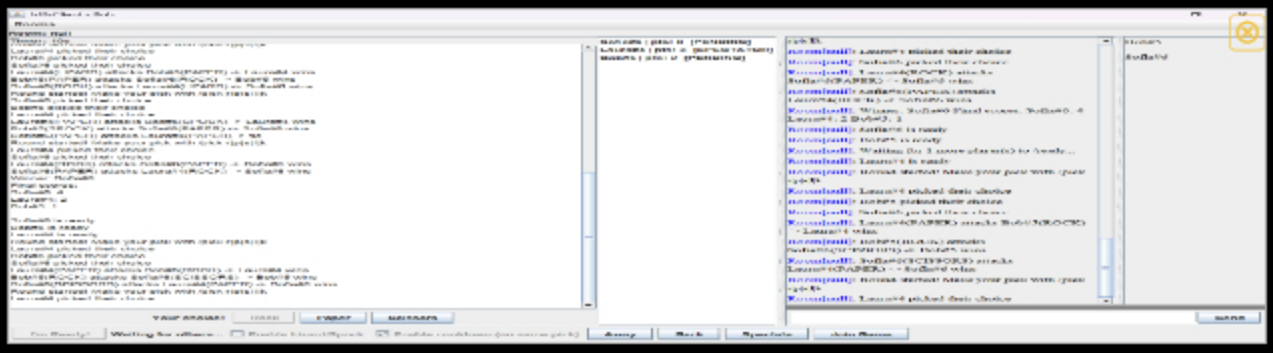
Progress: 100%

Details:

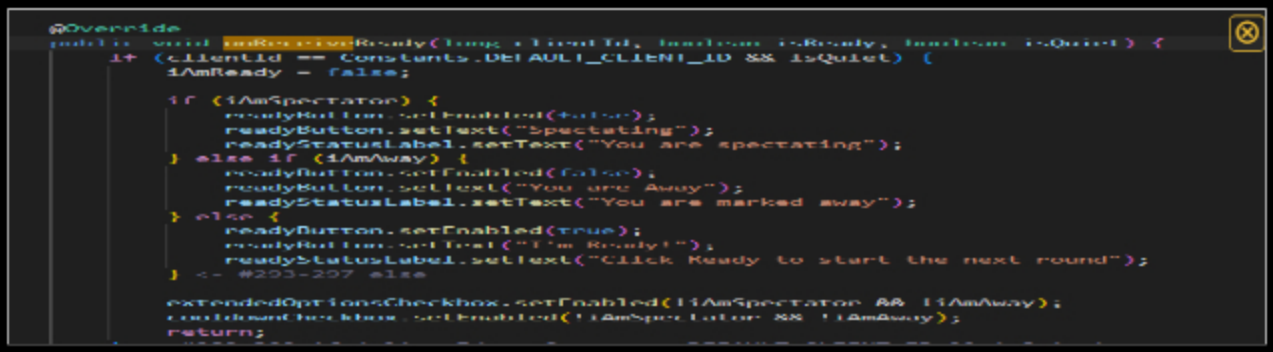
- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show a few examples of the play screen with the choice on cooldown
  - Show the related code for the UI and handling of the cooldown and server-side enforcing it



Enable cooldown



User can't pick rock due to picking it last round



cooldown checkbox visible

handling cooldown

Saved: 12/11/2025 2:17:42 AM

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 the cooldown period (include how this is recorded per user and reset when applicable)

Your Response:

In my implementation, the cooldown feature is controlled by a checkbox that the user can toggle on during the Ready Check phase. When the user toggles the checkbox, the UI calls gameView.setCooldownEnabled() to update the local client's UI behavior. Because Ready Check is the game's configuration phase, this checkbox is only interactable before the game begins, and once everyone presses Ready, the cooldown setting becomes locked for the entire session. When cooldown is enabled, the client keeps track of the player's last selected option using a variable such as lastPickCode. After each round, the UI calls resetPickButtonsForNewRound(), which disables only the button corresponding to the previous choice. This prevents the player from selecting the same option twice in a row during the next round. At the end of a session or game reset, both the UI and server clear their cooldown records so that each new session starts without restrictions

Saved: 12/11/2025 2:17:42 AM

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

Progress: 100%

Task #1 (1 pt.) - Away Status

Progress: 100%

Details:

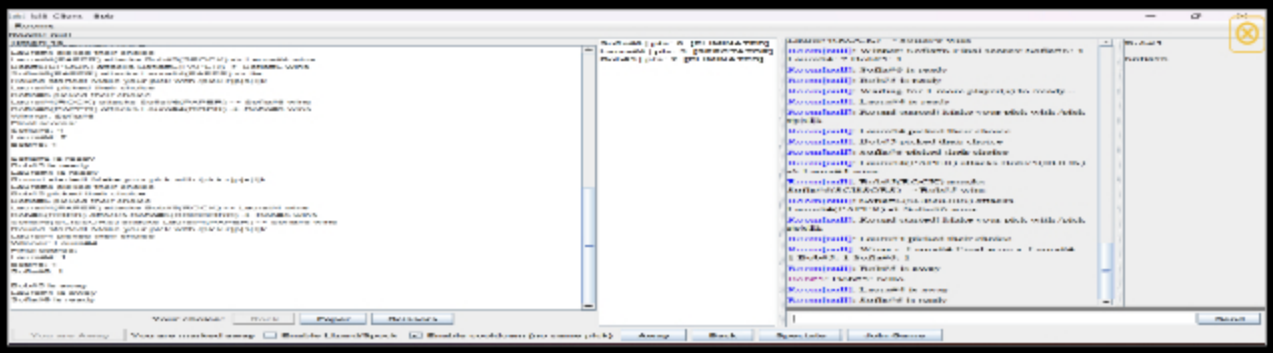
- Clients can mark themselves away and be skipped in turn flow but still part of the game
- The status should be visible to all participants
- A message should be relayed to the Game Events Panel (i.e., Bob is away or Bob is no longer away)
- The user list should have a visual representation (i.e., grayed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



Away button + [User] is away and cannot ready up

handling away server side

```
// Detect away status messages
if (message.endsWith(" is away")) {
    String name = message.substring(0, message.length() - " is away".length()).trim();
    setAwayByName(name, true);
    return;
} <- #319-323 if (message.endsWith(" is away"))
if (message.endsWith(" is no longer away")) {
    String name = message.substring(0, message.length() - " is no longer away".length()).trim();
    setAwayByName(name, false);
    return;
} <- #324-328 if (message.endsWith(" is no longer away"))
```

send away message to game events panel

```
    private String formatPlayerLine(PlayerInfo p) {
        StringBuilder sb = new StringBuilder();
        sb.append(p.name).append("  | pts: ").append(p.points);
        if (p.spectator) {
            sb.append("  |SPECTATOR|");
        } else if (p.eliminated) {
            sb.append("  [ELIMINATED]");
        } else if (p.away) {
            sb.append("  |AWAY|");
        } else if (inPickingPhase && !p.tookTurn) {
            sb.append("  [PENDING]");
        }
        return sb.toString();
    } <- #253-266 private String formatPlayerLine(PlayerInfo p)
```

visual on client side

```
//ls18 | 12/08/25
    private int countActivePlayers() {
        int count = 0;
        for (ServerThread st : clientsInRoom.values()) {
            long id = st.getClientId();
            if (!eliminated.contains(id)
                    && !away.contains(id)
                    && !spectators.contains(id)) {
                count++;
            }
        } <- #343-350 for (ServerThread st : clientsInRoom.values())
        return count;
    } <- #341-352 private int countActivePlayers()
```

skips user when isAway

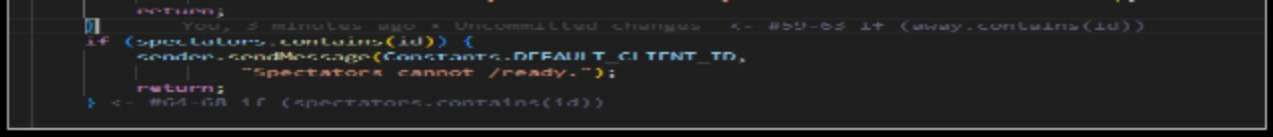
```
    private boolean allActivePicked() {        You, 2 weeks ago • Mi
        for (ServerThread st : clientsInRoom.values()) {
            long id = st.getClientId();
            if (!eliminated.contains(id)
                    && !away.contains(id)
                    && !spectators.contains(id)
                    && !choices.containsKey(id)) {
                return false;
            }
        } <- #329-337 for (ServerThread st : clientsInRoom.values())
        return countActivePlayers() > 0;
    } <- #328-339 private boolean allActivePicked()
```

Skip away users when checking if everyone has picked

```
//ls18 | 12/08/25
    public synchronized void handlePick(ServerThread sender, String arg) {
        if (phase != GamePhase.CHOOSING)
            return;

        long id = sender.getClientId();
        // spectators, eliminated and away cannot pick
        if (eliminated.contains(id) || away.contains(id) || spectators.contains(id)) {
            return;
        }       You, 2 minutes ago • Uncommitted changes
```

Ignore picks from Away players

```
    /**
     * /ready handler
     */
    public synchronized void handleReady(ServerThread sender) {
        long id = sender.getClientId();

        if (eliminated.contains(id)) {
            sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "Spectators cannot /ready.");
            return;
        }
        if (away.contains(id)) {
            sender.sendMessage(Constants.DEFAULT_CLIENT_ID,
                    "You are marked away and cannot /ready. Use /back to return first.");
```



cannot /ready

Saved: 12/11/2025 2:48:05 AM

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:

When the player clicks the Away / Back button in the game UI, the button toggles its label and calls the client method isAway. This creates a payload that contains the player's updated away status and sends it to the server through the socket. On the server side, this payload is received in ServerThread.processPayload() and passed into GameRoom.handleAway(), which adds or removes that player from the server's away set. As soon as the away state changes, the server broadcasts a message to all players such as "Laura is away" or "Laura is no longer away." Each client receives this broadcast in Client.processPayload(), and the message is forwarded into the UI through clientSideGameEvent(), where GameView.onMessageReceive() displays it in the Game Events Panel. Whenever the game checks whether a round can start, whether all players have made their picks, or how many active players remain, the server always excludes IDs that appear in the away set. For example, in countActivePlayers() and allActivePicked(), the server skips any user who is marked away, meaning they are never required to /ready and never hold up the round. Additionally, handlePick() refuses picks coming from Away users so they cannot accidentally rejoin the round until they press the Back button. This ensures Away players stay connected and can watch the game, but they are completely ignored by the turn logic, allowing the remaining players to continue uninterrupted

Saved: 12/11/2025 2:48:05 AM

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

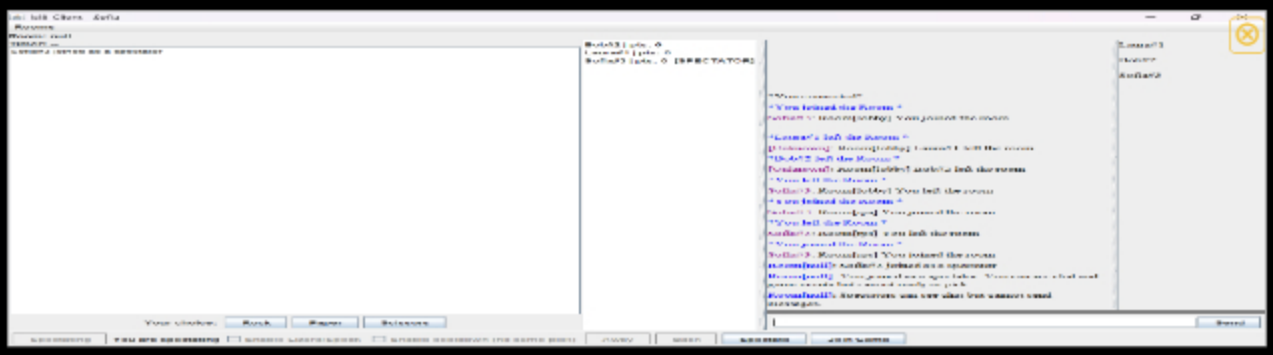
an in-progress session)

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)



UI indicator of spectator

```
private String formatPlayerLine(PlayerInfo p) {
    StringBuilder sb = new StringBuilder();
    sb.append(p.name).append("  | pts: ").append(p.points)
    if (p.spectator) {
        sb.append("  [SPECTATOR]");
    } else if (p.eliminated) {
        sb.append("  [ELIMINATED]");
    } else if (p.away) {
        sb.append("  [AWAY]");
    } else if (inPickingPhase && !p.tookTurn) {
        sb.append("  [PENDING]");
    }
    return sb.toString();
} <- #253-266 private String formatPlayerLine(PlayerInfo
```

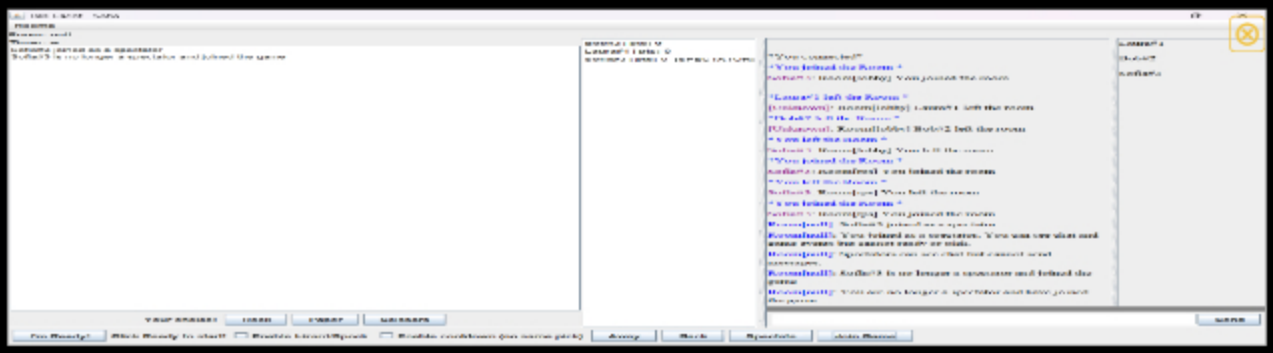
visual tag

handle spectate

handle spectate

```java
/**
 * Mark this client as a spectator in this game room.      You, 17 minutes ago • Uncommitted changes
 */
public synchronized void handleSpectatorJoin(ServerThread sender) {
    long id = sender.getClientId();
    spectators.add(id);
    String name = nameOf(id);
    broadcastNotice(String.format("%s joined as a spectator", name));
    sender.sendMessage(Constants.DEFAULT_CLIENT_ID,
            "You joined as a spectator. You can see chat and game events but cannot ready or pick.");
} <- #153-160 public synchronized void handleSpectatorJoin(ServerThread sen...
```

Broadcasting



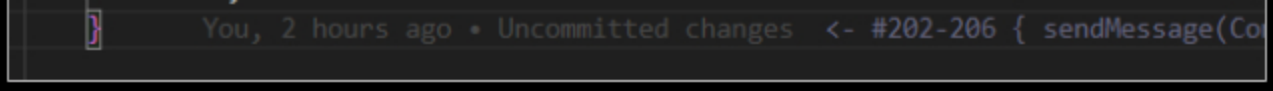
joined game

ignores spectator

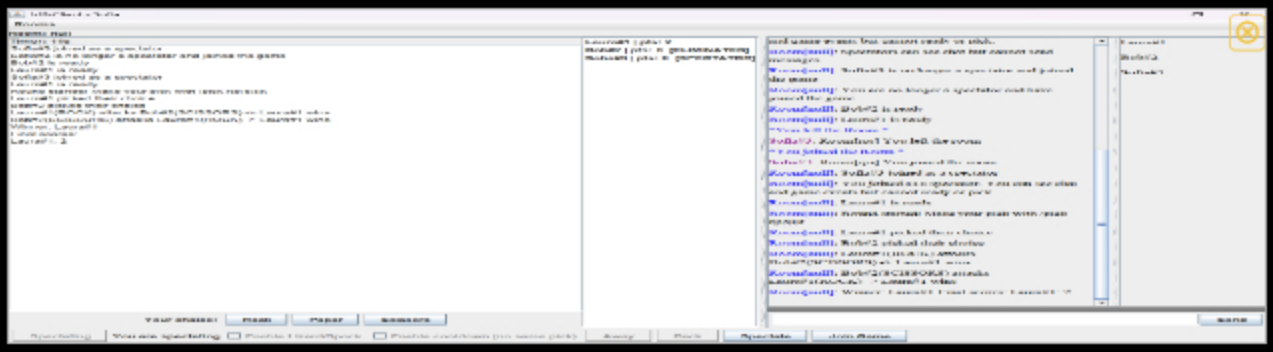
```java
//ls18 | 12/08/25
    public synchronized void handlePick(ServerThread sender, String arg) {      You, 2 weeks ago • Milestone
        if (phase != GamePhase.CHOOSING)
            return;

        long id = sender.getClientId();
        // spectators, eliminated and away cannot pick
        if (eliminated.contains(id) || away.contains(id) || spectators.contains(id)) {
            return;
        }
```

ignores spectator

```java
    // Block chat messages from spectators
    if (currentRoom instanceof GameRoom
            && ((GameRoom) currentRoom).isSpectator(getClientId())) {
        sendMessage(Constants.DEFAULT_CLIENT_ID,
                "Spectators can see chat but cannot send messages.");
        break;
```



can't send messages



spectator view of the session

Saved: 12/11/2025 3:13:45 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

Spectator status is mainly controlled on the server and then reflected in the UI through broadcast messages. When a client requests to become a spectator, the server receives a SPECTATE payload in ServerThread.processPayload() and forwards it to GameRoom.handleSpectatorJoin() or handleStopSpectate(). The server then calls broadcastNotice() with messages like "Bob joined as a spectator" or "Bob is no longer a spectator and joined the game", which are sent to all clients. On the client side, these notices are received in Client.processPayload() and passed to GameView.onMessageReceive(), where I both append them to the Game Events panel and update the user list model (for example, setting a spectator flag and showing [SPECTATOR] next to that player's name). On the server, spectators are treated as "non-players" for turn and round logic. In the GameRoom class, I track spectators in a spectators set and always check this set when deciding who counts as an active player. Methods like countActivePlayers() and allActivePicked() skip any client IDs that are in spectators, so they are never required to /ready and never block a round from starting or ending. Similarly, in handlePick(), I immediately return if the sender's ID is in the spectators set (or in away/eliminated), which means spectator picks are completely ignored by the game resolution logic. This ensures the game only considers real participants when running the round timers, checking if everyone has picked, awarding points, and determining when the game is over. To keep spectators from acting like regular players, I also gate chat and command messages on the server. When a text message comes in, the server's room logic (for example, GameRoom.handleMessage()) checks whether the sender's client ID is in the spectators

set. If it is, the server either ignores the message or sends a small response back such as "You are a spectator and cannot send messages." and then returns without broadcasting anything

Saved: 12/11/2025 3:13:45 AM

Section #5: (1 pt.) Misc

Progress: 100%

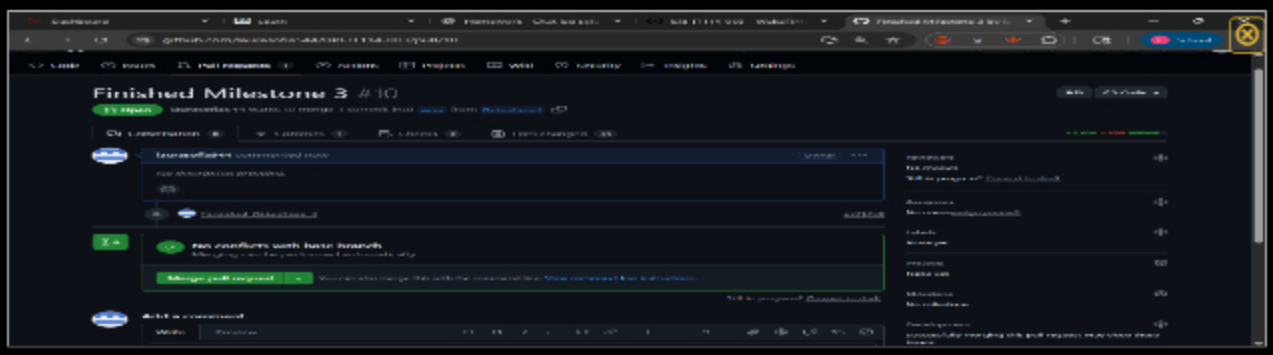
Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



commit history

Saved: 12/11/2025 3:16:32 AM

Part 2:

Progress: 100%

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

URL #1
https://github.com/laurasofia544/lsl8-IT114-003/0

URL
https://github.com/laurasofia544/

Saved: 12/11/2025 3:16:32 AM

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

Progress: 100%

Details:

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

overall time

individual time

Saved: 12/11/2025 3:17:34 AM

Task #3 (0.33 pts.) - Reflection

Progress: 100%

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

Progress: 100%

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

Your Response:

Throughout this project I learned how a real client–server works and how much coordination is required between the UI, the networking layer, and the server's game logic. Implementing features like Ready Check, Away, Spectator mode

cooldown logic, and RPS-5 options showed me how important it is to keep the server authoritative while letting the client reflect the state visually. I also gained experience structuring multi-step workflows and saw how vital it is to track state on both sides to keep the game consistent

Saved: 12/11/2025 3:20:09 AM

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

Progress: 100%

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

Your Response:

The easiest part for me was building the basic UI panels and wiring up simple controls like the Connect screen, Ready button, and basic Game Events display. These features followed a clear pattern: create the UI component, trigger a send method on click, and then reflect server messages in the UI. Once I understood how callbacks and payloads worked, adding new UI elements such as the cooldown checkbox or Away button became very manageable. Displaying server messages in the Game Events panel also felt straightforward because the groundwork had already been laid in the earlier milestones and baseline files

Saved: 12/11/2025 3:21:02 AM

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

Progress: 100%

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

Your Response:

The hardest part of the project was managing all of the game-state logic on the server and making sure it stayed synchronized across all clients. Features like spectator mode, away status, cooldown rules, elimination tracking, and extended RPS-5 options required careful coordination between multiple classes, especially since each change needed to update the UI for every connected client. Debugging why certain clients were starting the game early or why UI indicators were not updating correctly took a lot of time because the issue could be in the server, the

networking code, or the UI. Ensuring that the server always ignored spectators and away players, enforcing cooldown rules securely, and keeping the user list sorted and updated were all challenging parts because they involved both logic and communication between multiple parts of the system

Saved: 12/11/2025 3:21:56 AM