# Submission Worksheet

## Submission Data

**Course:** IT114-450-M2025
**Assignment:** IT114 - Milestone 3 - RPS
**Student:** Mukaddis I. (mi348)
**Status:** Submitted | **Worksheet Progress:** 100%
**Potential Grade:** 10.00/10.00 (100.00%)
**Received Grade:** 0.00/10.00 (0.00%)
**Started:** 8/6/2025 12:46:06 PM
**Updated:** 8/6/2025 12:46:06 PM
**Grading Link:** https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-3-rps/grading/mi348
**View Link:** https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-3-rps/view/mi348

## 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

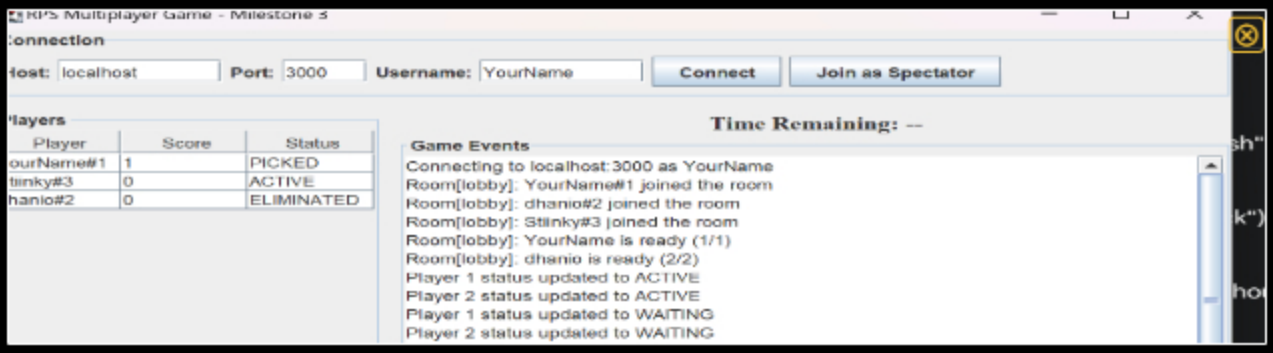
Progress: 100%

### Part 1:

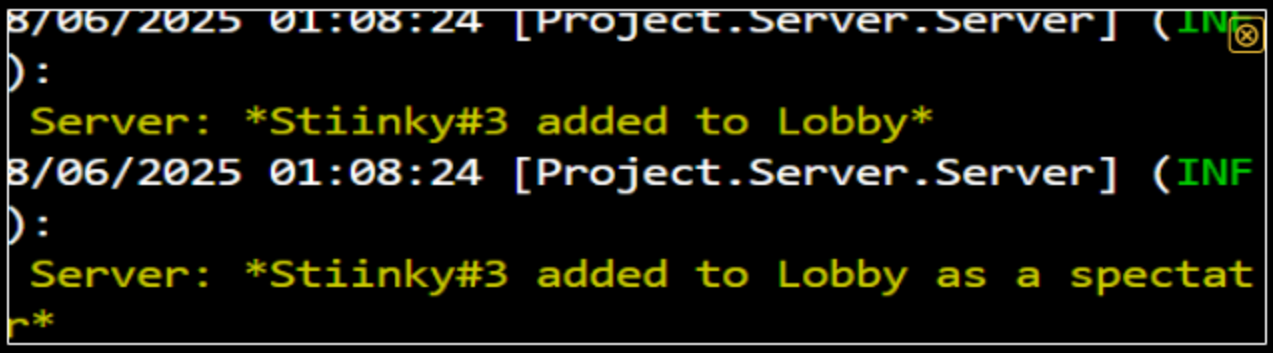
Progress: 100%

**Details:**

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




User Panel With SCreenshots




Connection Panel

Saved: 8/6/2025 11:17:24 AM

## Part 2:

Progress: 100%

**Details:**

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

Your Response:

After you click "Connect," the client's code wraps your selected username into a unique message known as a ConnectionPayload, which it then transmits to the server. After receiving this message, the server gives your player a special ID and notifies everyone in the room that you have joined. Your username and other information show up on the screen when your client receives this notification and updates the "Players" table on the left.

Saved: 8/6/2025 11:17:24 AM

# Task #2 ( 0.50 pts.) - Ready Panel

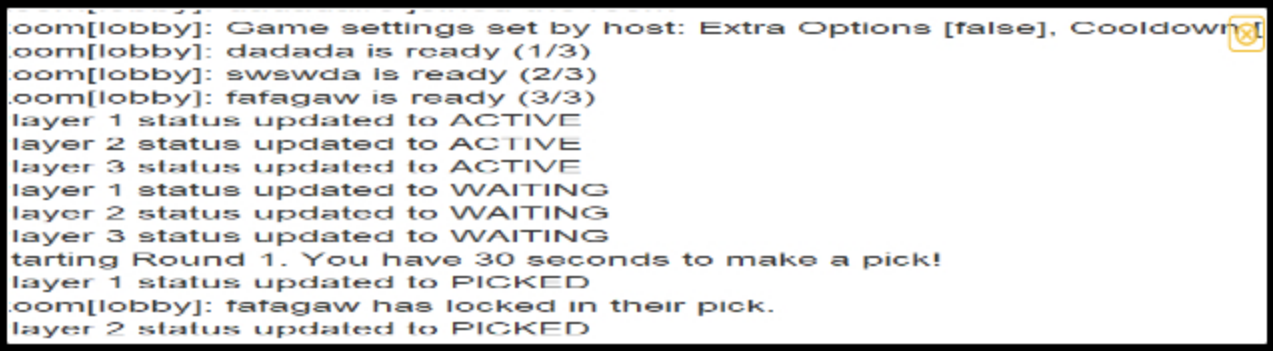
Progress: 100%

## Part 1:

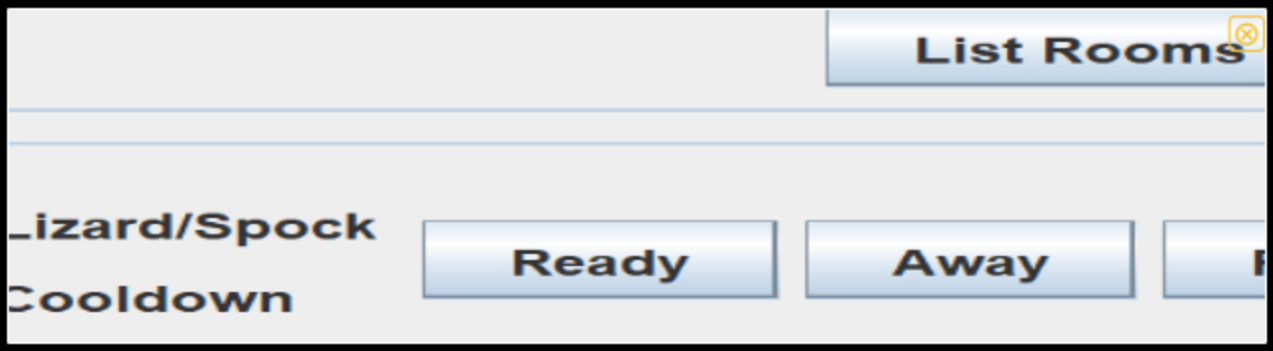
Progress: 100%

**Details:**

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




Clients ready up




ready up button

Saved: 8/6/2025 11:40:12 AM

## 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:

READY UI

When a player clicks the "Ready" button, the client's handleReady method is triggered, then creates a ReadyPayload containing any game settings selected by the host. This payload is then sent to the server, and the "Ready" button in the UI is immediately disabled to prevent multiple clicks.

Ready DATA After the server processes a "Ready" command, it broadcasts a message to all

Ready DATA After the server processes a "Ready" command, it broadcasts a message to all clients in the room, such as "YourName is ready (1/2)". The client's processPayload method receives this message and appends it to the "Game Events" log for everyone to see. If enough players are ready, the server then sends a ROUND_START payload, which triggers the client to update the player statuses in the table to "WAITING" and start the countdown timer.

Saved: 8/6/2025 11:40:12 AM

# 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

Players

| Player | Score | Status |
|---|---|---|
| dadada#3 | 1 | WAITING |
| afagaw#1 | 1 | WAITING |
| swswda#2 | 1 | WAITING |

visible Users/Current Points/Username + ID

```
if (result == 1) {
    p1.addPoint();
    syncPoints(p1);
    broadcast(battleLog + " -> " + p1.getName() + " wins!");
} else if (result == -1) {
    p2.addPoint();
    syncPoints(p2);
    broadcast(battleLog + " -> " + p2.getName() + " wins!");
} else {
    broadcast(battleLog + " -> Tie!");
}
```

Point sorting in GamesSession

```
ublic void setPlayers(List<User> playerList) {
    players.clear();
    players.addAll(playerList);
    players.sort(Comparator.comparing(User::getPoints).reversed()
                .thenComparing(User::getClientName));
    fireTableDataChanged();
```

PlayerTableModel.java

```
private void startRound() {
    round++;
    LoggerUtil.INSTANCE.info(TextFX.colorize("GameSession: Starting Round " + round, TextFX.Color.YELLOW));
    getActivePlayers().forEach(p -> {
        p.setPick(pick:null);
        p.setStatus(PlayerStatus.WAITING);
        syncPlayerStatus(p);
    });
    RoundStartPayload roundStartPayload = new RoundStartPayload(round, ROUND_TIME_SECONDS);
    room.broadcastPayload(roundStartPayload);
```

Pending-To-Pick

```
        break;
    case PLAYER_STATUS:
        if (p instanceof PlayerStatusPayload psp) {
            User user = knownClients.get(psp.getClientId());
            if (user != null) {
                user.setStatus(psp.getStatus());
                updatePlayerList();
            }
        }
```

Player Status

Saved: 8/6/2025 11:54:28 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:

Points Updates When a player wins a battle, the server's GameSession.java file gives them a point and sends a SYNC_POINTS message to all clients. The client's Client.java code receives this message, updates the score for that player in its local list, and tells the UI's table to refresh.

User List Sorting Every time the player list is updated, the Client.java code gives the fresh list to the PlayerTableModel.java file. Before displaying the data, the table model automatically sorts this list, putting the player with the highest score at the top. This ensures that all players see the same sorted list.

Pending-to-Pick Indicators At the start of a round, the server's GameSession.java sets every active player's status to "WAITING" and sends this update to all clients. When a player makes a move, the server changes only that player's status to "PICKED" and sends another update. The client's UI receives these status messages and updates the "Status" column in the player table to show who is waiting and who has picked.

Elimination Indicators When the server's GameSession.java determines a player has lost a round, it changes that player's status to "ELIMINATED" and broadcasts this change to all clients. The client's UI receives this "ELIMINATED" status update and displays it in the player table, clearly showing who is out of the game. This status will remain for the rest of the game session.

Saved: 8/6/2025 11:54:28 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

```
Starting Round 1. You have 30 seconds to make a pick!
Player 1 status updated to PICKED
Room[lobby]: fafagaw has locked in their pick.
Player 2 status updated to PICKED
Room[lobby]: swswda has locked in their pick.
Picked SCISSORS
Player 3 status updated to PICKED
Room[lobby]: dadada has locked in their pick.
Room[lobby]: --- Round 1 Results ---
Room[lobby]: fafagaw (rock) vs swswda (paper) -> swswda wins!
Room[lobby]: fafagaw (rock) vs dadada (scissors) -> fafagaw wins!
Room[lobby]: swswda (paper) vs dadada (scissors) -> dadada wins!
Player 1 status updated to WAITING
```

Player Picking and shows winning and etc

```java
public synchronized void registerPick(ServerThread sender, String rawPick) {
    if (!inProgress) {
        sender.sendMessage(Constants.DEFAULT_CLIENT_ID, message:"Game has not started.");
        return;
    }
    PlayerState p = players.get(sender.getClientId());
    if (p == null || p.isEliminated() || p.isAway() || p.isSpectator()) return;
    String pick = rawPick.trim().toLowerCase();
```

GameSession.java

```java
private void startRound() {
    round++;
    LoggerUtil.INSTANCE.info(TextFX.colorize("GameSession: Starting Round " + round, TextFX.Color.YELLOW));
    getActivePlayers().forEach(p -> {
        p.setPick(pick:null);
        p.setStatus(PlayerStatus.WAITING);
        syncPlayerStatus(p);
```

Start Round Status

Saved: 8/6/2025 12:04:27 PM

## Part 2:

Progress: 100%

**Details:**

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

Your Response:

Player Status Updates The server updates a player's status to "WAITING" or "PICKED" and sends a status update message to everyone. Each client then receives this message and refreshes the "Status" column in the player list to show the change.

Battle and Elimination Messages The server decides who wins or loses a round and sends the results as simple text messages, like "Player 1 wins!". The client's UI receives these messages and just adds them to the "Game Events" log for everyone to see.

Saved: 8/6/2025 12:04:27 PM

# Task #3 ( 0.67 pts.) - Game Area

Progress: 100%

**Details:**

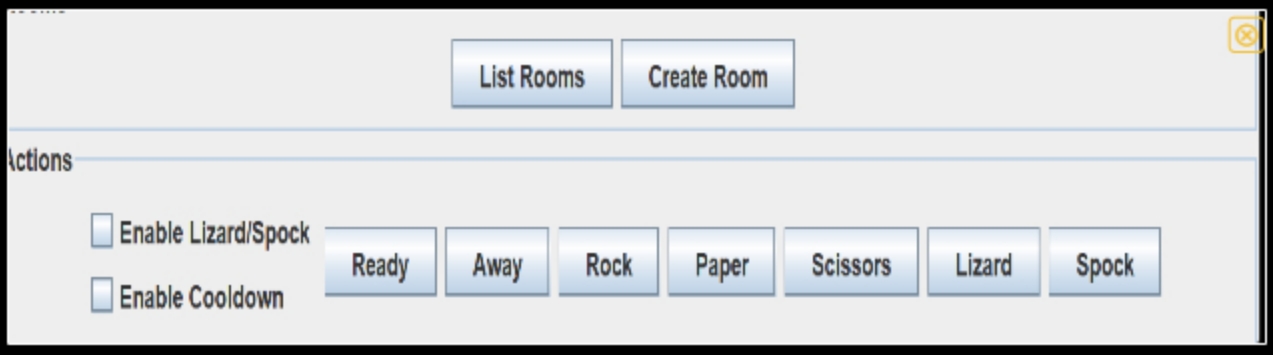
- UI should have components to allow the user to select their choice

## 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




Choices they can select

```
private void attachListeners() {
    ui.getConnectButton().addActionListener(e -> handleConnect(asSpectator:false));
    ui.getSpectatorButton().addActionListener(e -> handleConnect(asSpectator:true));
    ui.getReadyButton().addActionListener(e -> handleReady());
    ui.getAwayButton().addActionListener(e -> sendCommand(command:"/toggleaway"));
    ui.getListRoomsButton().addActionListener(e -> sendCommand(command:"/listrooms"));
    ui.getCreateRoomButton().addActionListener(e -> handleCreateRoom());
    ActionListener pickListener = e -> {
        JButton button = (JButton) e.getSource();
        sendPick(button.getText().toLowerCase());
    };
    ui.getRockButton().addActionListener(pickListener);
    ui.getPaperButton().addActionListener(pickListener);
    ui.getScissorsButton().addActionListener(pickListener);
    ui.getLizardButton().addActionListener(pickListener);
    ui.getSpockButton().addActionListener(pickListener);
```

ActionListener is attached to each of the five choice buttons

```
    if (myUser.isSpectator()) {
        logToUI(message:"Spectators can't play!");
        return;
    }
    // Immediately disable all buttons to prevent multiple pic
    ui.getRockButton().setEnabled(b:false);
    ui.getPaperButton().setEnabled(b:false);
    ui.getScissorsButton().setEnabled(b:false);
```

```
    ui.getLizardButton().setEnabled(b:false);
    ui.getSpockButton().setEnabled(b:false);
```

first disables all the choice buttons to prevent the player from picking more than once.

```
public synchronized void registerPick(ServerThread sender, String rawPick) {
    if (!inProgress) {
        sender.sendMessage(Constants.DEFAULT_CLIENT_ID, message:"Game has not started.");
        return;
    }
```

updates the player's internal state with their choice and changes their status to PICKED

Saved: 8/6/2025 12:12:01 PM

### 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:

Selecting and Sending a Choice When you click a choice button like "Rock" in the UI, an event listener in Client.java is triggered, which calls the sendPick method.

Receiving and Displaying the Choice The server's GameSession.java receives the GAME_PICK payload and updates your player's status to "PICKED". It then broadcasts this status change to all clients in the room, including you. Your client's code receives this broadcast and refreshes the player table, which makes your "Status" in the UI change from "WAITING" to "PICKED" for everyone to see.

Saved: 8/6/2025 12:12:01 PM

# 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)

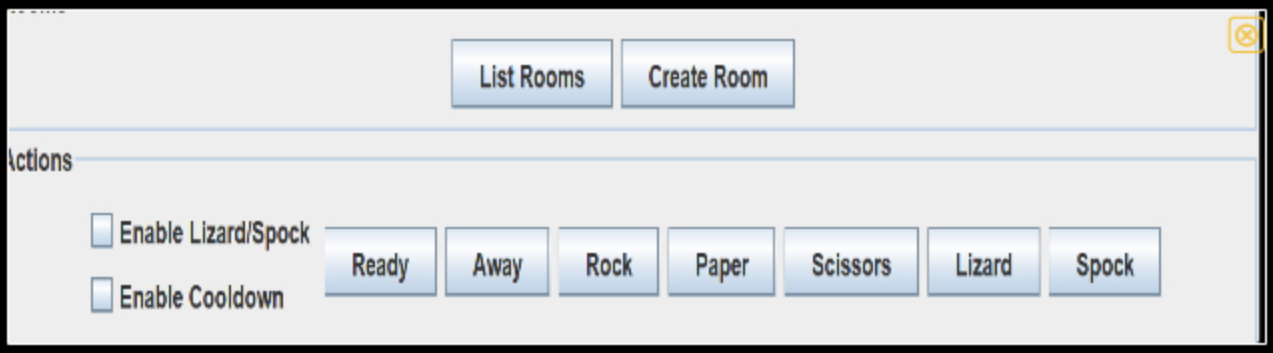
- There should be at least 2 extra options for rps-5

## 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 options

```
JPanel settingsPanel = new JPanel();
settingsPanel.setLayout(new BoxLayout(settingsPanel, BoxLayout.Y_AXIS))
extraOptionsCheck = new JCheckBox(text:"Enable Lizard/Spock");
cooldownCheck = new JCheckBox(text:"Enable Cooldown");
settingsPanel.add(extraOptionsCheck);
settingsPanel.add(cooldownCheck);
```

ClientGameGUI

```
private void handleReady() {
    if (myUser.isSpectator()) return;

    ReadyPayload payload = new ReadyPayload();
    // The first player in the list is considered the host and can set rules
    if (knownClients.size() <= 1 || knownClients.keySet().iterator().next() == myUser.getClientId()) {
        payload.setExtraOptionsEnabled(ui.getExtraOptionsCheck().isSelected());
        this.cooldownEnabled = ui.getCooldownCheck().isSelected(); // Update client-side rule
        payload.setCooldownEnabled(this.cooldownEnabled);
    }

    try {
        sendToServer(payload);
        ui.getReadyButton().setEnabled(b:false);
    } catch (IOException e) {
        logToUI(message:"Error sending ready status.");
```

When any player clicks the "Ready" button, the handleReady method is called.

Saved: 8/6/2025 12:18:13 PM

### Part 2:

Progress: 100%

**Details:**

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

Host's Option to Toggle the Feature When a player clicks the "Ready" button, the client's code checks if they are the "host" (the first player in the room). If they are, it reads whether the "Enable Lizard/Spock" and "Enable Cooldown" checkboxes in the UI are selected and packages these true/false values into a special ReadyPayload message that is sent to the server.

Handling Options in Battle Logic The server's GameSession.java file receives the ReadyPayload from the host and stores the settings for the entire game session. When any player tries to make a pick, the server first checks the extraOptionsEnabled setting. If it's false, "lizard" and "spock" are rejected as invalid moves.

I decided to go with Option 1

Saved: 8/6/2025 12:18:13 PM

## Task #2 ( 2 pts.) - Choice cooldown

Progress: 100%

**Details:**

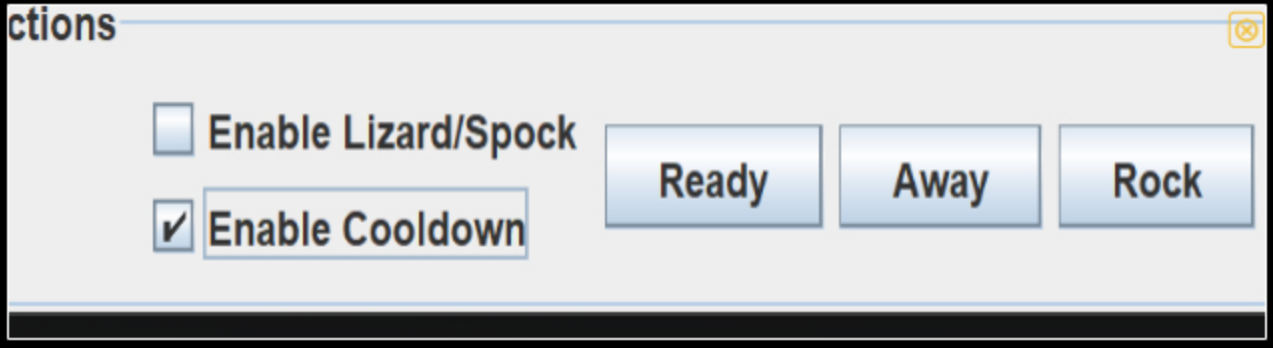
- Setting should be toggleable during Ready Check by session creator
- The choice on cooldown must be disable on the UI for the User

### Part 1:

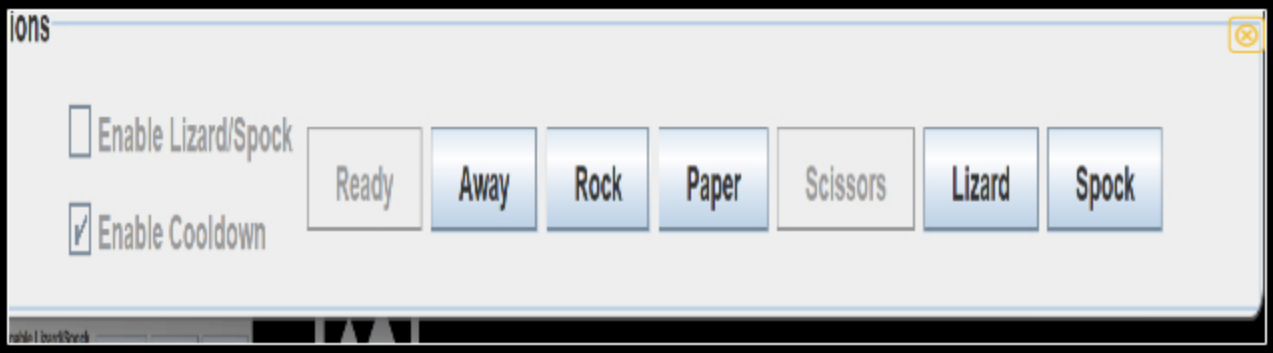
Progress: 100%

**Details:**

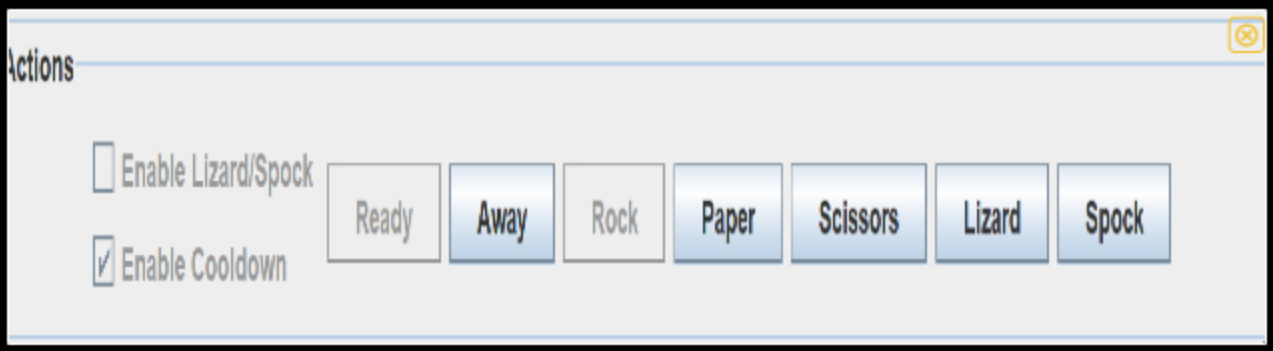
- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show a few examples of the play screen with the choice on cooldown
  - Show the related code for the UI and handling of the cooldown and server-side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




Enable Cooldown




Scissors on cooldown




Rock on cooldown

```
private void handleReady() {
    if (myUser.isSpectator()) return;

    ReadyPayload payload = new ReadyPayload();
    // The first player in the list is considered the host and can set rules
    if (knownClients.size() <= 1 || knownClients.keySet().iterator().next() == myUser.getClientId()) {
        payload.setExtraOptionsEnabled(ui.getExtraOptionsCheck().isSelected());
        this.cooldownEnabled = ui.getCooldownCheck().isSelected(); // Update client-side rule
        payload.setCooldownEnabled(this.cooldownEnabled);
    }

    try {
        sendToServer(payload);
        ui.getReadyButton().setEnabled(b:false);
    } catch (IOException e) {
        logToUI(message:"Error sending ready status.");
```

When any player clicks the "Ready" button, the handleReady method is called

```
ase RESET_GAME_STATE:
    ui.getReadyButton().setEnabled(b:true);
    ui.getAwayButton().setSelected(b:false);
    ui.getExtraOptionsCheck().setSelected(b:false);
    ui.getExtraOptionsCheck().setEnabled(b:true);
    ui.getCooldownCheck().setSelected(b:false);
    ui.getCooldownCheck().setEnabled(b:true);
    lastPick = "";
    ui.getRockButton().setEnabled(b:true);
    ui.getPaperButton().setEnabled(b:true);
    ui.getScissorsButton().setEnabled(b:true);
    ui.getLizardButton().setEnabled(b:true);
    ui.getSpockButton().setEnabled(b:true);
    break;
```

```
ase SESSION_END:
```

payload is received at the end of a game, the checkboxes are re-enabled

Saved: 8/6/2025 12:30:25 PM

### Part 2:

Progress: 100%

**Details:**

- Briefly explain the code for the host's option to toggle this feature
- Briefly explain the code related to handling and enforcing the cooldown period (include how this is recorded per user and reset when applicable)

Your Response:

Host's Option to Toggle The host's ability to toggle game features is managed by a client-server interaction where the client checks if the player is the host and, if so, sends a special payload with the settings to the server.

Cooldown Period The cooldown period is a client-and-server-enforced rule that prevents a player from using the same move twice in a row, with the client visually disabling the button for the last-used pick to provide immediate feedback.

Saved: 8/6/2025 12:30:25 PM

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

Progress: 100%

## Task #1 ( 1 pt.) - Away Status

Progress: 100%

**Details:**

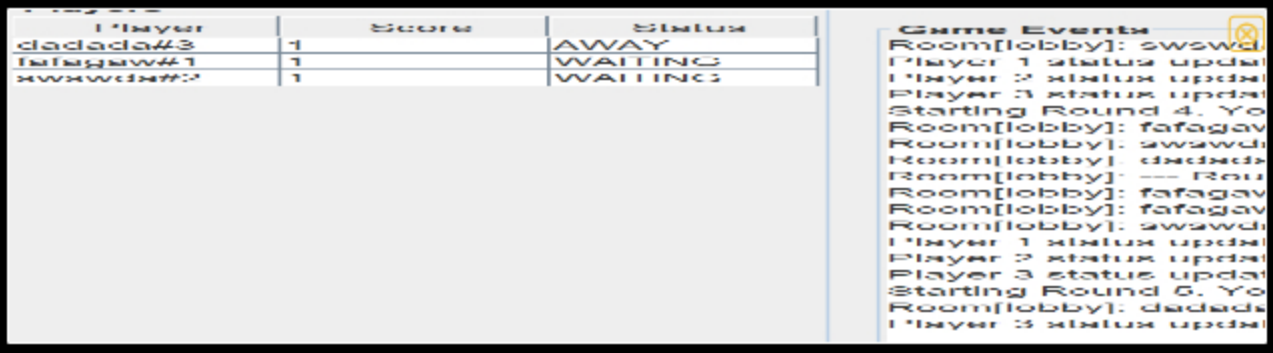
- Clients can mark themselves away and be skipped in turn flow but still part of the game
- The status should be visible to all participants
- A message should be relayed to the Game Events Panel (i.e., Bob is away or Bob is no longer away)
- The user list should have a visual representation (i.e., grayed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

### Part 1:

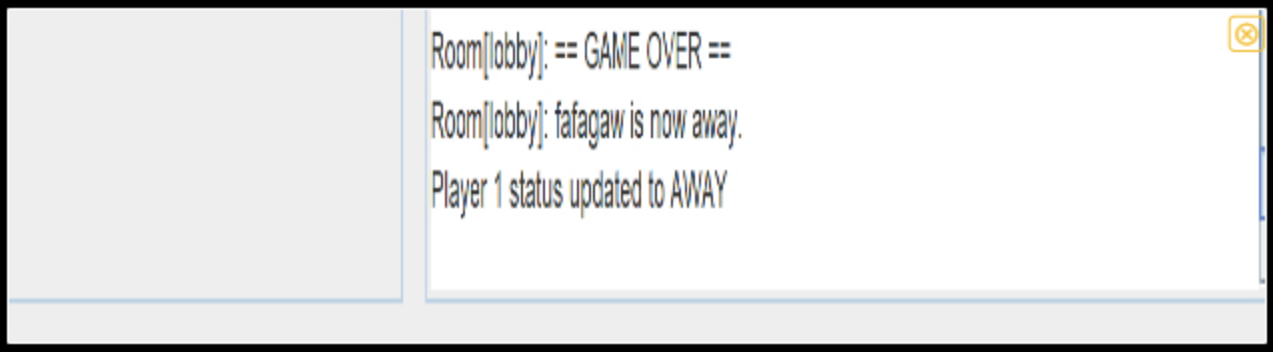
Progress: 100%

**Details:**

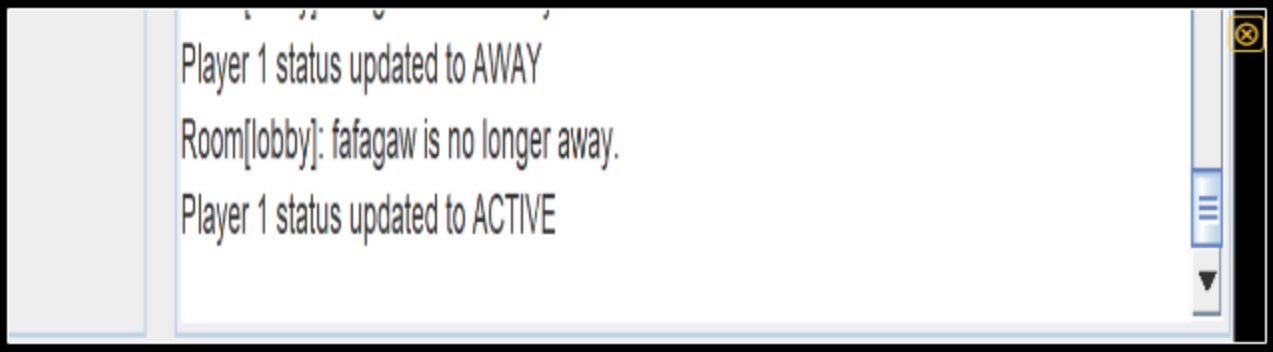
- Show the UI button to toggle away
- Show the related code flow from UI to server-side back to UI for showing the status
- Show the related code flow for sending the message to Game Events Panel
- Show various examples across 3+ clients of away status (including Game Events Panel messages)
- Show the code that ignores an away user from turn/round logic



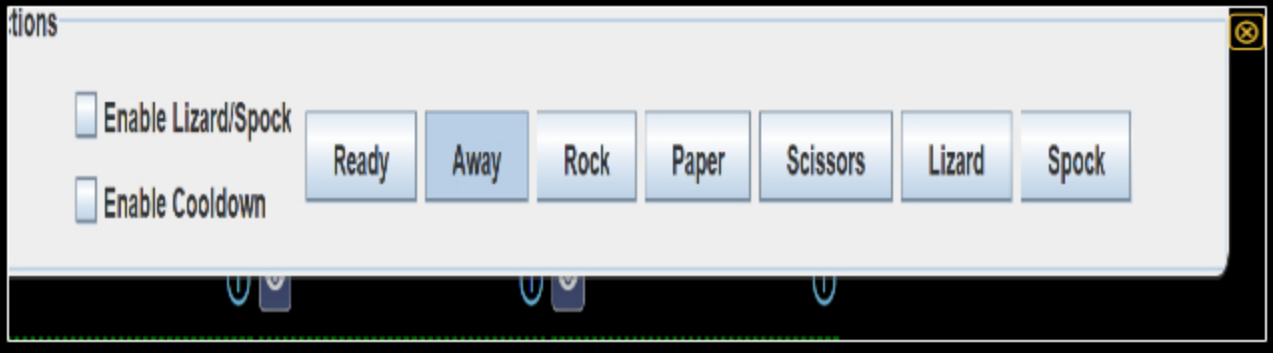

marked away globally




Sent in the event log




Changed back to active no longer away




Darkened Button

```java
public synchronized void toggleAwayStatus(ServerThread sender) {
    if(sender.isSpectator()) return;
    PlayerState p = players.computeIfAbsent(sender.getClientId(), id -> new PlayerState(sender));
    p.setAway(!p.isAway());
    if (p.isAway()) {
        p.setStatus(PlayerStatus.AWAY);
        broadcast(p.getName() + " is now away.");
    } else {
        p.setStatus(PlayerStatus.ACTIVE);
        broadcast(p.getName() + " is no longer away.");
    }
```

method will simply return and do nothing if p.isAway() is true.

Saved: 8/6/2025 12:32:49 PM

## Part 2:

Progress: 100%

**Details:**

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

Your Response:

Away Action Code Flow When you hit the "Away" button, your computer sends a message to the server saying, "I'm away now." The server then flips a switch on your player profile and tells everyone else in the room that your status has changed. Your screen, and everyone else's, then updates to show you as "Away."

Ignoring Away Users The game's code on the server just checks if your "away switch" is on before doing anything. If the switch is on, the game skips over you. So, when it's time to see who is ready or who has made a move, the game simply ignores your name and moves on to the next person.

Saved: 8/6/2025 12:32:49 PM

# Task #2 ( 1 pt.) - Spectators

Progress: 100%

**Details:**

- Spectators are users who didn't mark themselves ready
    - Optionally you can include a toggle on the Ready Check page
- The can see all chat but are ignored from turn/round actions and can't send messages
- Spectators will have a visual representation in the user list to distinguish them from other players
- A message should be relayed to the Game Events Panel that a spectator joined (i.e., during an in-progress session)

## Part 1:

Progress: 100%

**Details:**

- Show the UI indicator of a spectator (visual and message)
- Show the related code flow from UI to server-side back to UI for showing the status
- Show the related code flow for sending the message to Game Events Panel
- Show various examples across 3+ clients of spectator status (including Game Events Panel messages)
- Show the code that ignores a spectator from turn/round logic
- Show the code that prevents spectators from sending messages (server-side)
- Show the spectator's view of the session
- Show the code related to the spectator seeing the session data (including things participants won't see)

| Player | Score | Status |
|---|---|---|
| Speccy#4 | 0 | SPECTATING |
| dadada#3 | 0 | ACTIVE |
| fafagaw#1 | 0 | ACTIVE |
| swswda#2 | 0 | ACTIVE |

**Game Events**

Connecting to localhost:3000 as Speccy as a spectator
Room[lobby]: Speccy#4 joined the room

Shows the spectator in specatator status

Room[lobby]: dadada is ready (2/3)

can see live logs

```
public synchronized void markReady(ServerThread sender, Payload readyPayload) {
    if (inProgress || sender.isSpectator()) return;


    // The first non-spectator to ready up is the host and sets the rules
```

Spectators are explicitly excluded from all game logic.

```
case JOIN_SPECTATOR:
    setClientName(((ConnectionPayload) incoming).getClientName().trim());
    Server.INSTANCE.addSpectatorToLobby(this);
    break;
```

ServerThread

Saved: 8/6/2025 12:40:26 PM

### Part 2:

Progress: 100%

**Details:**

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

Your Response:

Spectator Code Flow When a spectator joins, the server flips a switch on their profile to mark them as a spectator, then sends a message to everyone's computer saying, "Hey, a spectator joined!" This makes their status show up as "SPECTATING" in the player list.

Ignoring Spectators The server's game code simply checks if a user is a spectator before doing anything game-related. If the check says "yes, they're a spectator," the code just skips over them, so they can't ready up, make a pick, or do anything else in the game.

Preventing Spectator Messages The server has a rule that says, "If you're a spectator, you can't send messages." So, if a spectator tries to send a chat message, the server sees their status and just ignores the message completely.

Saved: 8/6/2025 12:40:26 PM

# Section #5: ( 1 pt.) Misc

Progress: 100%

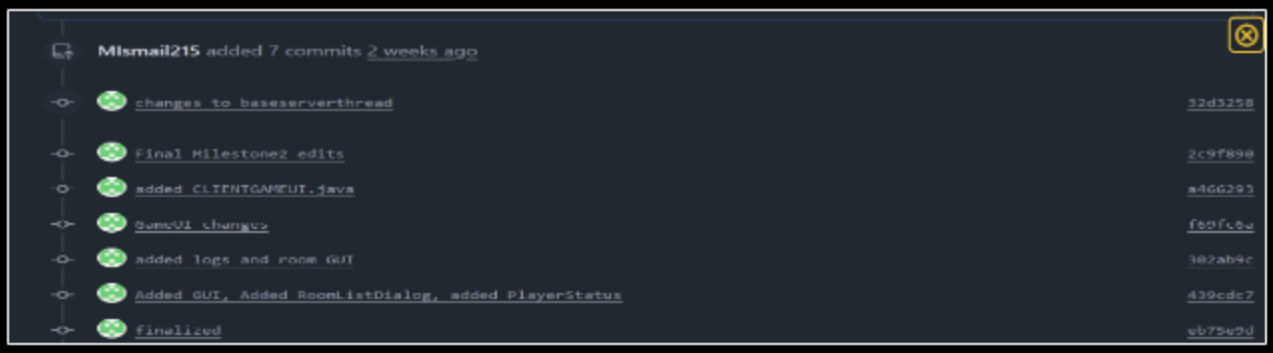
## Task #1 ( 0.33 pts.) - Github Details

Progress: 100%

### Part 1:

Progress: 100%

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




Github

Saved: 8/6/2025 12:46:06 PM

## Part 2:

Progress: 100%

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

URL #1
https://github.com/MIsmail215/mi348-IT114-450/1

URL
https://github.com/MIsmail215/m

Saved: 8/6/2025 12:46:06 PM

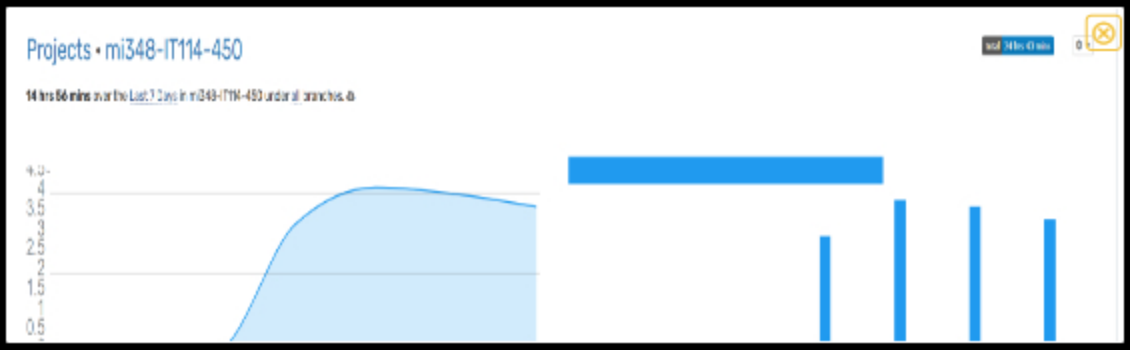
# Task #2 ( 0.33 pts.) - WakaTime - Activity

Progress: 100%

**Details:**

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

Bottom of Page




Top of Page

Saved: 8/6/2025 12:43:26 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:

I learned how to code several new features for a multiplayer game, including how to handle a host's options to toggle game rules like "Lizard/Spock" and a cooldown period. I also figured out how to implement a spectator mode, making sure spectators can watch the game without messing with the action, and how to prevent them from chatting. The biggest takeaway was understanding how to make the UI, client, and server all work together smoothly to enforce these rules and show the right information to each user. I also got to see how to make sure the game ignores "away" players and spectators during the main gameplay loop.

Saved: 8/6/2025 12:44:59 PM

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

Progress: 100%

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

Your Response:

The easiest part of the project was definitely setting up the ReadyPayload.java file. It was just a simple class with two boolean fields, extraOptionsEnabled and cooldownEnabled, and their getters and setters. There wasn't any complex logic to figure out or bugs to fix; I just had to create a new file and define the data it would carry. It was a straightforward way to get the host's choices from the client's checkboxes over to the server.

Saved: 8/6/2025 12:45:28 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 definitely getting the UI to talk to the server and back again for the host's settings. I had to create a new payload, ReadyPayload.java, to even send the checkbox data. Then, I had to make sure the client's code in Client.java correctly identified the host and only let them send those settings. On the server side, I had to modify GameSession.java to actually receive and store those booleans, and then use them to change how the game worked, like checking the extraOptionsEnabled flag to allow "Lizard/Spock." It was a lot of small, specific changes across multiple files that all had to be perfect to work.

Saved: 8/6/2025 12:45:49 PM