Submission Worksheet

Submission Data

Course: IT114-003-F2025
Assignment: IT114 - Milestone 3 - RPS
Student: Rayyan K. (rk975)
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 3:08:43 PM
Updated: 12/8/2025 4:59:06 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-3-rps/grading/rk975
View Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-3-rps/view/rk975

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%

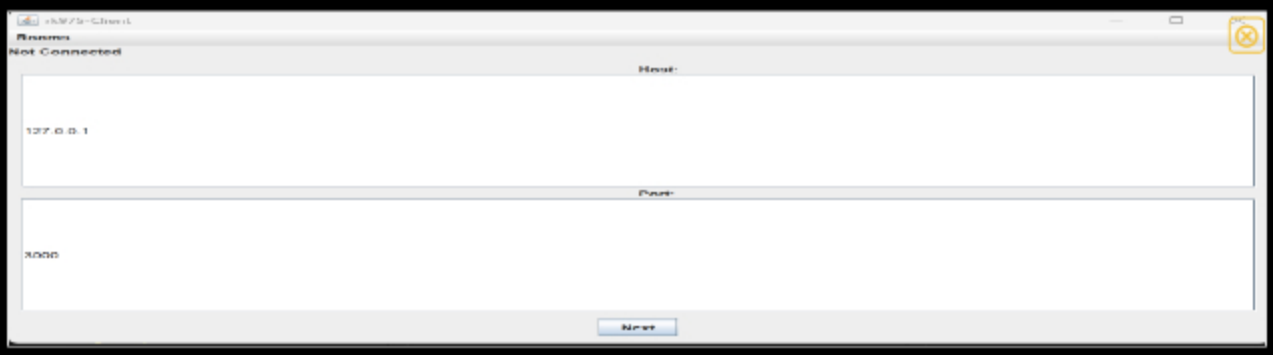
Part 1:

Progress: 100%

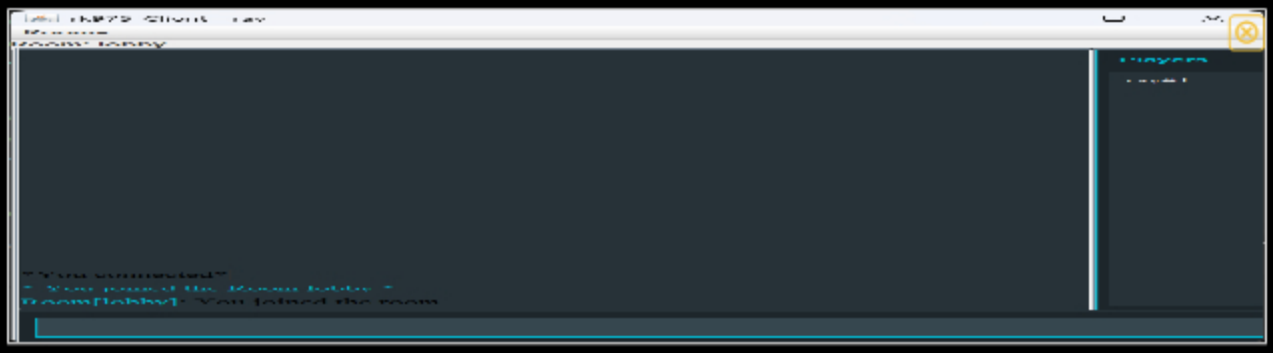
Details:

- Show the connection panel with valid data

- Show the user details panel with valid data



connection



connection

Saved: 12/8/2025 3:11:00 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:

The UserDetailsView captures the username and ConnectionView captures the host/port. When the user clicks connect, ClientUI.connect() retrieves these values and calls Client.INSTANCE.connect(host, port, username), which opens a socket connection to the server. The Client then sends a ConnectionPayload with the username via sendClientName(), and the server's ServerThread receives and processes this payload to complete the handshake by assigning a client ID and confirming the connection.

Saved: 12/8/2025 3:11:00 PM

Task #2 (0.50 pts.) - Ready Panel

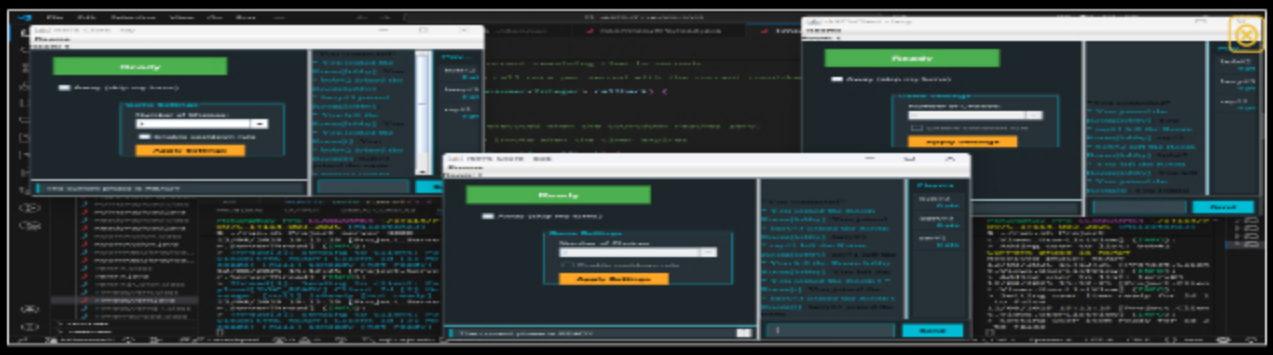
Progress: 100%

Part 1:

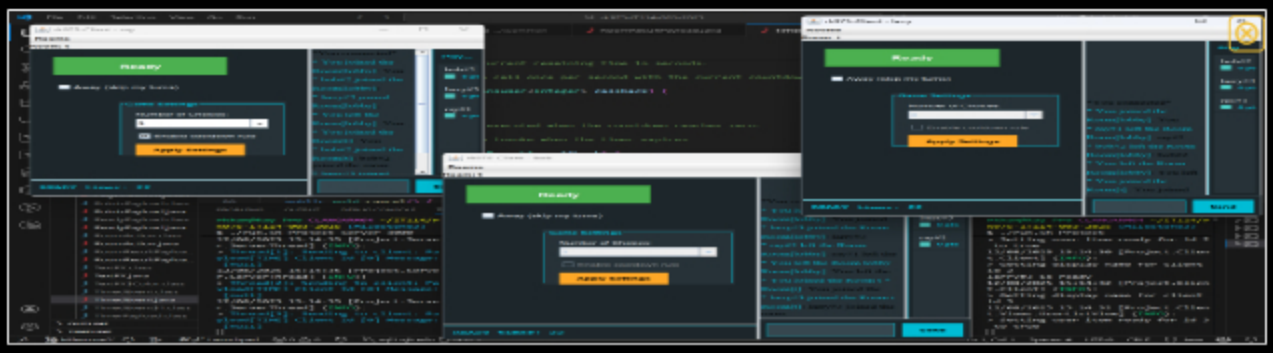
Progress: 100%

Details:

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



ready



ready

Saved: 12/8/2025 3:16:25 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:

When the user clicks the "Ready" button in ReadyView, it calls Client.INSTANCE.sendReady(), which creates a ReadyPayload and sends it to the server via the socket connection. The server's ServerThread receives this payload and routes it to GameRoom.handleReady(), which marks the player as ready, starts the ready timer, and broadcasts the updated ready status to all clients in the room.

When a client receives a READY payload, Client.processPayload() routes it to processReadyStatus(), which updates the player's ready state in the knownClients map. The

processReadyStatus(), which updates the player's ready state in the knownClients map. The method then calls passToUICallback() to notify all registered IReadyEvent listeners. UserListView.onReceiveReady() receives this callback and updates the turn indicator color for that player, while GameEventsView.onReceiveReady() displays a message indicating the player is ready.

Saved: 12/8/2025 3:16:25 PM

Section #2: (2 pts.) Project Ui

Progress: 100%

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

Progress: 100%

Details:

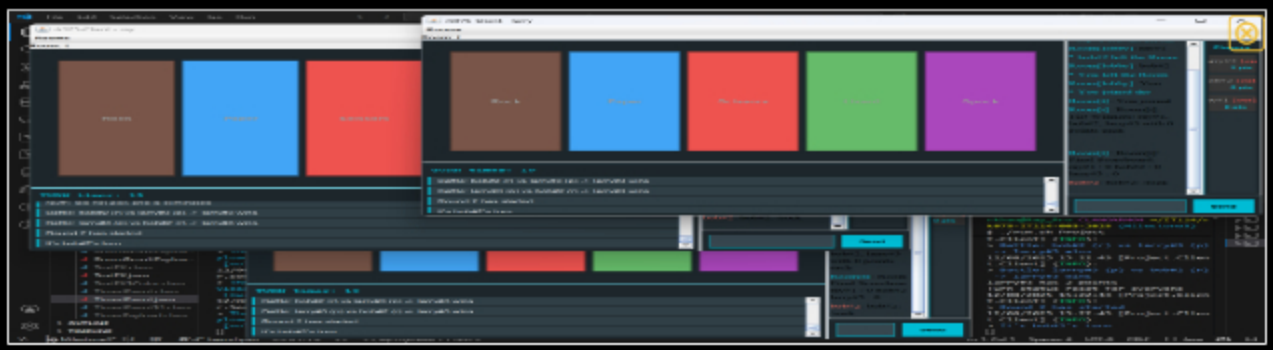
- Show the username and id of each user
- Show the current points of each user
- Users should appear in score order, sub-sort by name when ties occur
- Pending-to-pick users should be marked accordingly
- Eliminated users should be marked accordingly

Part 1:

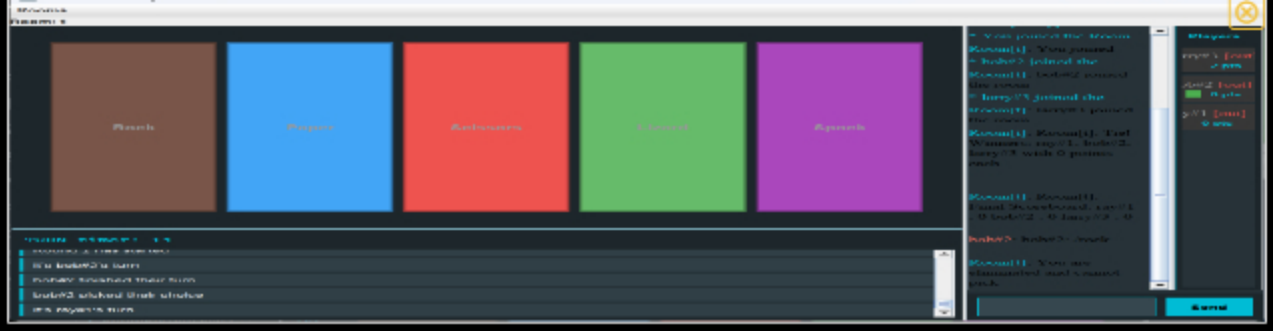
Progress: 100%

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



game



game

```
protected void sendReadyStatus(ServerThread incomingSP, boolean isReady) {
    clientsInRoom.values().removeIf(spInRoom -> {
        boolean failedToSend = !spInRoom.sendReadyStatus(incomingSP.getClientId(), incomingSP.isRe
        if (failedToSend) {
            removeClient(spInRoom);
        }
        return failedToSend;
    });
}
```

serverside ui

```
public boolean sendReadyStatus(long clientId, boolean isReady, boolean quiet) {
    ReadyPayload rp = new ReadyPayload();
    rp.setClientId(clientId);
    rp.setReady(isReady);
    if (quiet) {
        rp.setPayloadType(PayloadType.SYNC_READY);
    }
    return sendToClient(rp);
}
```

serverside ui

```
private void processReadyStatus(Payload payload, boolean isQuiet) {
    if (!(payload instanceof ReadyPayload)) {
        error("Invalid payload subclass for processRoomsList");
        return;
    }
    ReadyPayload rp = (ReadyPayload) payload;
    if (!knownClients.containsKey(rp.getClientId())) {
        LoggerUtil.INSTANCE.severe(String.format("Received ready status [%s] for client id %s who
                rp.isReady() ? "ready" : "not ready", rp.getClientId()));
        return;
    }
    User cp = knownClients.get(rp.getClientId());
    cp.setReady(rp.isReady());
    if (!isQuiet) {
        System.out.println(
                String.format("%s is %s", cp.getDisplayName(),
                        rp.isReady() ? "ready" : "not ready"));
    }

    passToUICallback(IReadyEvent.class, e -> e.onReceiveReady(cp.getClientId(), cp.isReady(), isQu
}
```

serverside ui

```
@Override
public void onReceiveReady(long clientId, boolean isReady, boolean isQuiet) {
    if (clientId == Constants.DEFAULT_CLIENT_ID) {
        SwingUtilities.invokeLater(() -> {
            try {
                userItemsMap.values().forEach(u -> u.setTurn(false));// reset all
            } catch (Exception e) {
                LoggerUtil.INSTANCE.severe("Error resetting user items", e);
            }
        });
    } else if (userItemsMap.containsKey(clientId)) {

        SwingUtilities.invokeLater(() -> {
            try {
                LoggerUtil.INSTANCE.info("Setting user item ready for id " + clientId + " to " + i
                userItemsMap.get(clientId).setTurn(isReady, READY_COLOR);
            } catch (Exception e) {
                LoggerUtil.INSTANCE.severe("Error setting user item", e);
            }
        });
    }
```

serverside ui

```
private void sendPlayerPoints(ServerThread sp) {
    clientsInRoom.values().removeIf(spInRoom -> {
        boolean failedToSend = !spInRoom.sendPlayerPoints(sp.getClientId(), sp.getPoints());
        if (failedToSend) {
            removeClient(spInRoom);
        }
        return failedToSend;
    });
}
```

maintained acorss clients

```
private void processPoints(Payload payload) {
    if (!(payload instanceof PointsPayload)) {
        error("Invalid payload subclass for processPoints");
        return;
    }
    PointsPayload pp = (PointsPayload) payload;
    long id = pp.getClientId();
    int pts = pp.getPoints();
    if (!knownClients.containsKey(id)) {
        User user = new User();
        user.setClientId(id);
        knownClients.put(id, user);
    }
    User u = knownClients.get(id);
    u.setPoints(pts);
    System.out.println(String.format("%s has %d points", u.getDisplayName(), pts));
    // notify UI listeners so the user list and other components update
    passToUICallback(IPointsEvent.class, e -> e.onPointsUpdate(id, pts));
}
```

maintained acorss clients

```
@Override
public void onPointsUpdate(long clientId, int points) {
    if (clientId == Constants.DEFAULT_CLIENT_ID) {
        SwingUtilities.invokeLater(() -> {
            try {
                userItemsMap.values().forEach(u -> u.setPoints(-1));// reset all
            } catch (Exception e) {
                LoggerUtil.INSTANCE.severe("Error resetting user items", e);
            }
        });
    } else if (userItemsMap.containsKey(clientId)) {
        SwingUtilities.invokeLater(() -> {
            try {
                userItemsMap.get(clientId).setPoints(points);
                // resort the list so ordering reflects updated scores
                refreshUserList();
            } catch (Exception e) {
                LoggerUtil.INSTANCE.severe("Error setting user item", e);
            }

        });
    }
}
```

maintained acorss clients

```
private void refreshUserList() {
    SwingUtilities.invokeLater(() -> {
        try {
            userListArea.removeAll();
            // sort entries
            List<UserListItem> items = userItemsMap.values().stream()
                    .sorted((a, b) -> {
                        int cmp = Integer.compare(b.getPoints(), a.getPoints());
                        if (cmp != 0) return cmp;
                        return a.getName().compareToIgnoreCase(b.getName());
                    }).toList();

            GridBagConstraints gbc = new GridBagConstraints();
            gbc.gridx = 0;
            gbc.gridy = GridBagConstraints.RELATIVE;
            gbc.weightx = 1;
            gbc.anchor = GridBagConstraints.NORTH;
            gbc.fill = GridBagConstraints.BOTH;
            gbc.insets = new Insets(0, 0, 0, 0);

            for (UserListItem item : items) {
                userListArea.add(item, gbc);
            }
            userListArea.add(Box.createVerticalGlue(), lastConstraints);
            userListArea.revalidate();
            userListArea.repaint();
        } catch (Exception e) {
            LoggerUtil.INSTANCE.severe("Error refreshing user list", e);
        }
    });
}
```

maintained acorss clients

```
@Override
public void onReceivePhase(Project.Common.Phase phase) {
    // When entering choosing phase, mark non-eliminated, non-spectator users as pending
    SwingUtilities.invokeLater(() -> {
        if (phase == Project.Common.Phase.CHOOSING) {
            userItemsMap.values().forEach(u -> {
                // if not eliminated and not spectator, mark pending
                if (!u.isEliminated() && !u.isSpectator()) {
                    u.setPending(true);
                } else {
                    // ensure eliminated/spectator users are not pending
```

```
                    u.setPending(false);
                }
            });
        } else {
            // clear pending markers outside choosing
            userItemsMap.values().forEach(u -> u.setPending(false));
        }
    });
}
```

pending pick

```
currentUser.setTookTurn(true);
// broadcast turn status to all clients so UI can mark them as picked
clientsInRoom.values().forEach(st -> st.sendTurnStatus(currentUser.getClientId(), true));
// send pick notification to Game Events panel instead of chat
clientsInRoom.values().forEach(st -> st.sendGameEvent(String.format("%s picked their choic
```

pending pick

```
@Override
public void onTookTurn(long clientId, boolean didtakeCurn) {
    if (clientId == Constants.DEFAULT_CLIENT_ID) {
        SwingUtilities.invokeLater(() -> {
            userItemsMap.values().forEach(u -> u.setTurn(false));// reset all
        });
    } else if (userItemsMap.containsKey(clientId)) {
        SwingUtilities.invokeLater(() -> {
            userItemsMap.get(clientId).setTurn(didtakeCurn);
            // when someone takes their turn, they are no longer pending
            if (didtakeCurn) {
                userItemsMap.get(clientId).setPending(false);
            }
        });
    }
}
```

pending pick

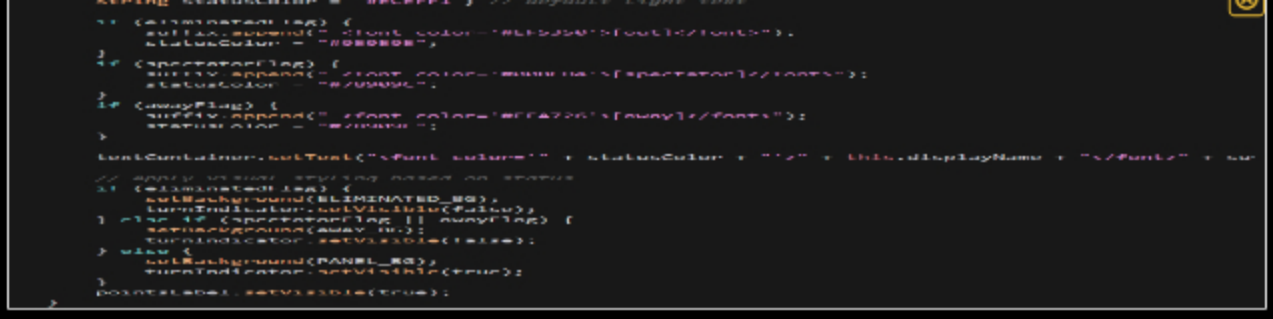
```
clientsInRoom.values().forEach(sp -> {
        if (!sp.user.isEliminated() && !sp.user.isSpectator() && !sp.user.isAway() && sp.user.
        sp.user.setEliminated(true);
        // announce non-picker elimination to Game Events panel
        clientsInRoom.values().forEach(st -> st.sendGameEvent(String.format("%s did not pick a
        PlayerStatePayload out = new PlayerStatePayload();
        out.setPayloadType(PayloadType.PLAYER_STATE);
        out.setClientId(sp.getClientId());
        out.setPoints(sp.user.getPoints());
        out.setEliminated(true);
        out.setAway(sp.user.isAway());
        out.setSpectator(sp.user.isSpectator());
        clientsInRoom.values().forEach(st -> st.sendToClient(out));
    }
});
```

elim

```
private void processPlayerState(Payload payload) {
    if (!(payload instanceof PlayerStatePayload)) {
        error("Invalid payload subclass for processPlayerState");
        return;
    }
    PlayerStatePayload pp = (PlayerStatePayload) payload;
    long id = pp.getClientId();
    if (!knownClients.containsKey(id)) {
        User u = new User();
        u.setClientId(id);
        knownClients.put(id, u);
    }
    User u = knownClients.get(id);
    u.setPoints(pp.getPoints());
    u.setEliminated(pp.isEliminated());
    u.setAway(pp.isAway());
    u.setSpectator(pp.isSpectator());
    passToUICallback(Project.Client.Interfaces.IPlayerStateEvent.class,
            e -> e.onPlayerStateUpdate(id, pp.getPoints(), pp.isEliminated(), pp.isAway(), pp.isSp
}
```

elim

```
private void updateDisplayText() {
    StringBuilder buffie = new StringBuilder();
```



elim

Saved: 12/8/2025 3:36:08 PM

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:

When a player earns points, the server calls sendPlayerPoints() which broadcasts a PointsPayload to all clients containing the player's ID and updated score. Each client's processPoints() method updates the local knownClients map and triggers onPointsUpdate() in UserListView, which calls refreshUserList() to re-sort and rebuild the list.

The user list sorting is handled entirely client-side in refreshUserList(), which sorts all UserListItems first by points descending, then alphabetically by name as a tiebreaker. Since all clients receive identical point values from the server and use the same sorting algorithm, the list order stays consistent across all connected clients.

For pending-to-pick indicators, when the server changes to CHOOSING phase, all clients receive the phase update and UserListView.onReceivePhase() marks active players with an orange pending indicator. When a player picks, the server broadcasts their turn status, and UserListView.onTookTurn() changes their indicator to green and clears the pending state.

For elimination, when a player fails to pick within the round, the server sets their eliminated flag and broadcasts a PlayerStatePayload to all clients. The client's processPlayerState() updates the user data and triggers UserListItem.updateDisplayText(), which grays out the name, adds an "out" tag, and hides the turn indicator.

Saved: 12/8/2025 3:36:08 PM

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

Progress: 100%

Details:

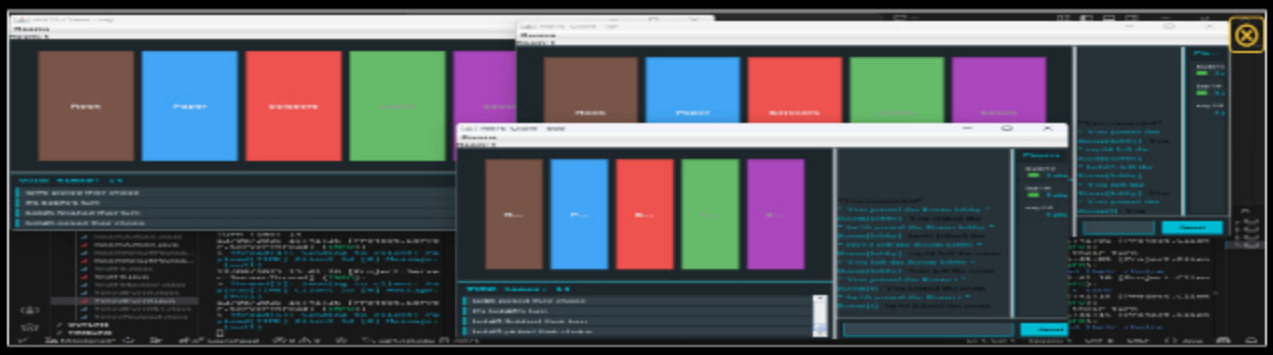
- Show the status of users picking choices
- Show the battle resolution messages from Milestone 2
    - Include messages about elimination
- Show the countdown timer for the round

Part 1:

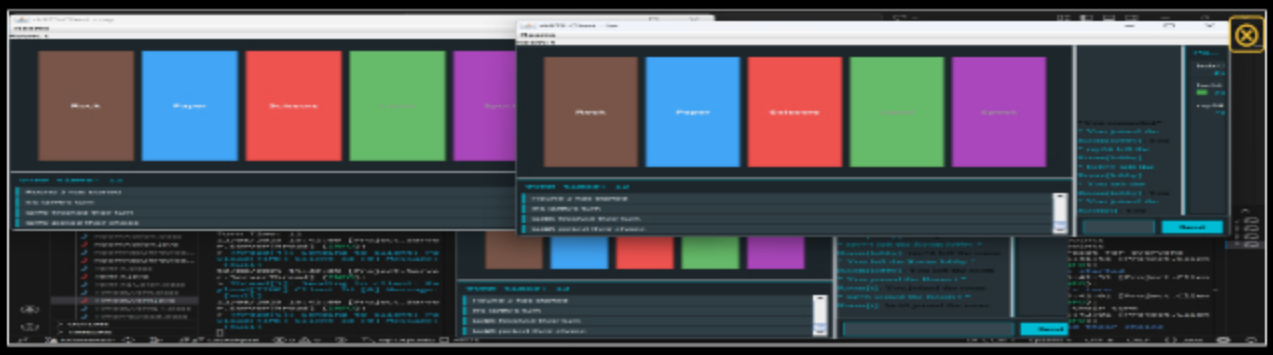
Progress: 100%

Details:

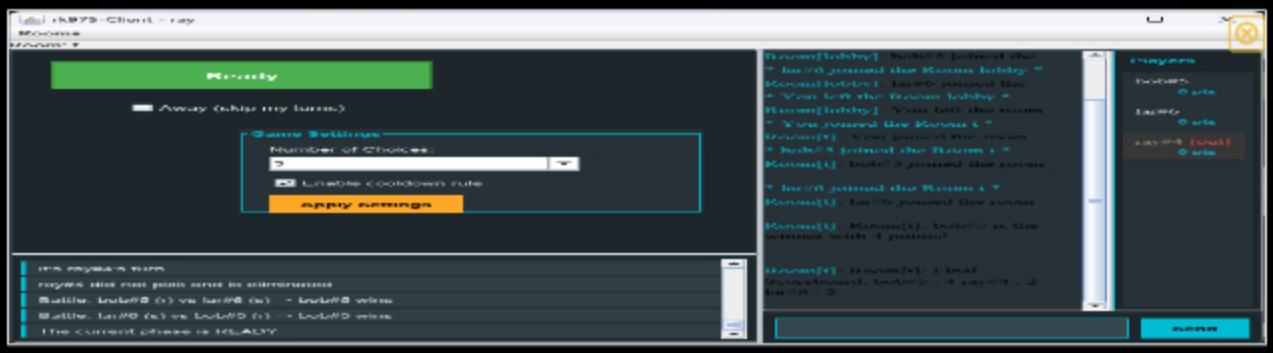
- Show various examples of each of the messages/visuals
- Show code snippets related to these messages from server-side to UI



game



game



game

```
        if (beats(aChoice, dChoice)) {
            addPoints.put(attacker, addPoints.getOrDefault(attacker, 0) + 1);
            resultMsg = String.format("Battle: %s (%s) vs %s (%s) -> %s wins", attacker.getDisplayN
                    defender.getDisplayName(), dChoice, attacker.getDisplayName());
        } else if (beats(dChoice, aChoice)) {
            addPoints.put(defender, addPoints.getOrDefault(defender, 0) + 1);
            resultMsg = String.format("Battle: %s (%s) vs %s (%s) -> %s wins", attacker.getDisplayN
                    defender.getDisplayName(), dChoice, defender.getDisplayName());
        } else {
            resultMsg = String.format("Battle: %s (%s) vs %s (%s) -> tie", attacker.getDisplayName
                    defender.getDisplayName(), dChoice);
        }
        // send battle result to Game Events panel (separate from chat)
        clientsInRoom.values().forEach(st -> st.sendGameEvent(resultMsg));
```

code snippets

```
@Override
public void onMessageReceive(long id, String message) {
    if (id == Constants.GAME_EVENT_CHANNEL) {// using -2 as an internal channel for GameEvents
        addText(message);
    }
}
```

code snippets

```
protected synchronized void relay(ServerThread sender, String message) {
    if (!isRunning) { // block action if Room isn't running
        return;
    }

    final long senderId = sender == null ? Constants.DEFAULT_CLIENT_ID : sender.getClientId();
    final String formattedMessage = String.format("%s: %s", sender == null ? String.format("Room[%s

    clientsInRoom.values().removeIf(serverThread -> {
        boolean failedToSend = !serverThread.sendMessage(senderId, formattedMessage);
        if (failedToSend) {
            disconnect(serverThread);
        }
        return failedToSend;
    });
}
```

code snippets

```
@Override
public void onMessageReceive(long clientId, String message) {
    if (clientId < Constants.DEFAULT_CLIENT_ID) {
        return;
    }
    String displayName = Client.INSTANCE.getDisplayNameFromId(clientId);
    // Custom teal/coral color scheme for messages
    String name = clientId == Constants.DEFAULT_CLIENT_ID ? "<font color='#00BCD4'>%s</font>"  //
            : "<font color='#FF6F61'>%s</font>";  // Coral for users
    name = String.format(name, displayName);
    addText(String.format("%s: %s", name, message));
}
```

code snippets

```
protected void sendCurrentTime(TimerType timerType, int time) {
    clientsInRoom.values().removeIf(spInRoom -> {
        boolean failedToSend = !spInRoom.sendCurrentTime(timerType, time);
        if (failedToSend) {
            removeClient(spInRoom);
        }
        return failedToSend;
    });
}
```

code snippets

```
@Override
public void onTimerUpdate(TimerType timerType, int time) {
    if (time >= 0) {
        timerText.setText(String.format("%s timer: %s", timerType.name(), time));
    } else {
        timerText.setText(" ");
    }
    timerText.setVisible(true);
}
```

code snippets

Saved: 12/8/2025 3:48:42 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

Game event messages are generated server-side using String.format() to build descriptive text like battle results, then sent via sendGameEvent() which uses a special channel ID (GAME_EVENT_CHANNEL = -2) to route them to GameEventsView instead of the chat. Chat messages go through Room.relay(), which formats the sender's display name with the message and broadcasts it to all clients via sendMessage(), where ChatView.onMessageReceive() applies color styling based on whether it's from a user (coral) or system (teal).

Timer updates are sent every second from the server's TimedEvent tick callback, which calls sendCurrentTime() to broadcast the remaining time and timer type to all clients. The client's processCurrentTimer() receives this and triggers onTimerUpdate() in GameEventsView, which updates a JLabel at the top of the panel showing "READY timer: 25" format. When time expires or resets, the server sends -1 which clears the timer display.

Saved: 12/8/2025 3:48:42 PM

Task #3 (0.67 pts.) - Game Area

Progress: 100%

Details:

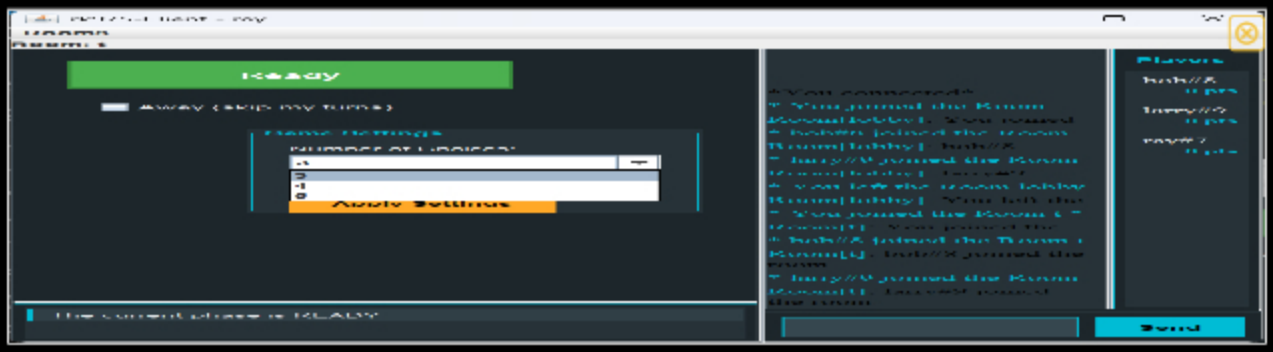
- UI should have components to allow the user to select their choice

Part 1:

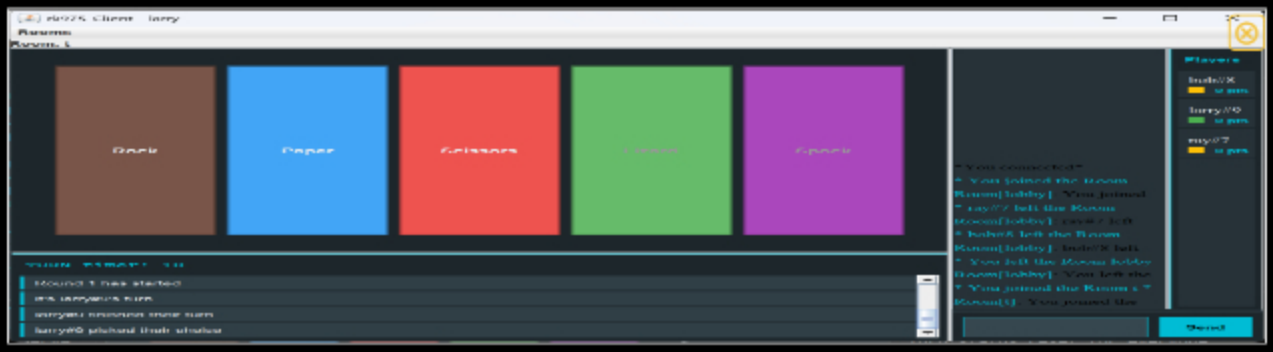
Progress: 100%

Details:

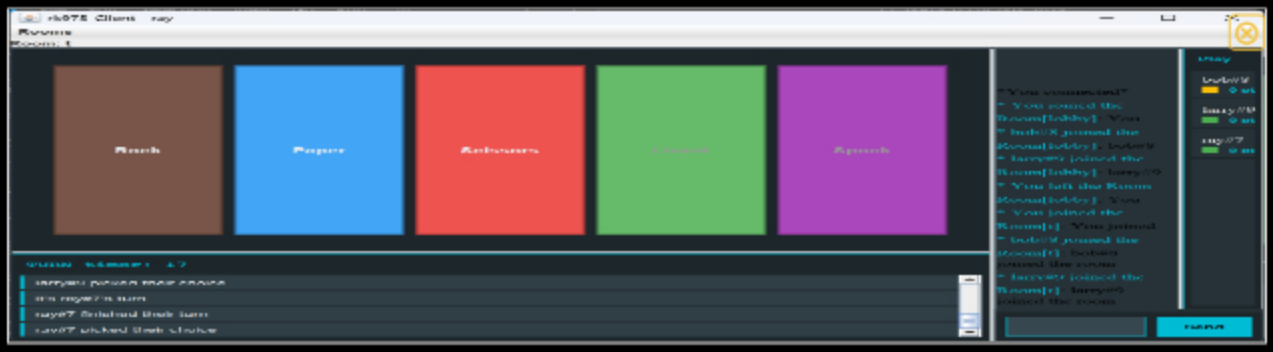
- Show various examples of selections across clients (3+ clients visible)
- Show the code related to sending choices upon selection
- Show the code related to showing visually what was selected



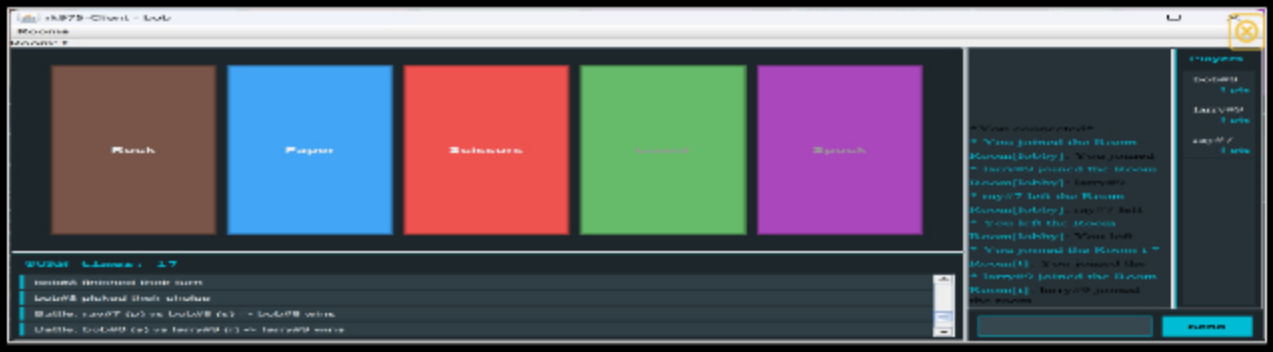
selection



selection



selection



selection

selection

```
private void sendPick(String choice) {
    try {
        // send single-letter legacy choices as r/p/s to server convenience
        String shortChoice = choice;
        if (choice.equalsIgnoreCase("rock")) shortChoice = "r";
        if (choice.equalsIgnoreCase("paper")) shortChoice = "p";
        if (choice.equalsIgnoreCase("scissors")) shortChoice = "s";
        if (choice.equalsIgnoreCase("lizard")) shortChoice = "l";
        if (choice.equalsIgnoreCase("spock")) shortChoice = "k";
        // remember pending pick until server confirms via turn event
        this.pendingSentPick = shortChoice;
        Client.INSTANCE.sendPick(shortChoice);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

code snippets

```
public void sendPick(String text) throws IOException {
    String c = text.toLowerCase();
    // accept full words as well
    if (c.equals("rock")) c = "r";
    if (c.equals("paper")) c = "p";
    if (c.equals("scissors")) c = "s";
    if (c.equals("lizard")) c = "l";
    if (c.equals("spock")) c = "k"; // k for spock
    if (!(c.equals("r") || c.equals("p") || c.equals("s") || c.equals("l") || c.equals("k"))) {
        LoggerUtil.INSTANCE.warning(TextFX.colorize("Invalid pick. Use r,p,s or extras if enabled"
        return;
    }
    PickPayload pp = new PickPayload();
    pp.setPayloadType(PayloadType.PICK);
    pp.setChoice(c);
    sendToServer(pp);
}
```

code snippets

```
    currentUser.user.setChoice(c);
    currentUser.user.setLastChoice(c);
    // mark that this player has taken their pick for the round
    currentUser.setTookTurn(true);
    // broadcast turn status to all clients so UI can mark them as picked
    clientsInRoom.values().forEach(st -> st.sendTurnStatus(currentUser.getClientId(), true));
    // send pick notification to Game Events panel instead of chat
    clientsInRoom.values().forEach(st -> st.sendGameEvent(String.format("%s picked their choic
```

code snippets

```
@Override
public void onTookTurn(long clientId, boolean didtakeCurn) {
    if (clientId == Constants.DEFAULT_CLIENT_ID) {
        SwingUtilities.invokeLater(() -> {
            userItemsMap.values().forEach(u -> u.setTurn(false));// reset all
        });
    } else if (userItemsMap.containsKey(clientId)) {
        SwingUtilities.invokeLater(() -> {
            userItemsMap.get(clientId).setTurn(didtakeCurn);
            // when someone takes their turn, they are no longer pending
            if (didtakeCurn) {
                userItemsMap.get(clientId).setPending(false);
            }
        });
    }
}
```

code snippets

```
public void setTurn(boolean didTakeTurn, Color trueColor) {
    // when user has taken their turn, show the indicator in trueColor
    turnIndicator.setBackground(didTakeTurn ? trueColor : new Color(0, 0, 0, 0));
    turnIndicator.revalidate();
    turnIndicator.repaint();
    this.revalidate();
```

```
        this.repaint();
    }
```

code snippets

Saved: 12/8/2025 3:59:40 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:

When a player clicks a choice button in PlayView, the sendPick() method converts the choice to a single letter code (r/p/s/l/k) and calls Client.INSTANCE.sendPick(). The Client creates a PickPayload containing the choice and sends it to the server via the socket connection using sendToServer(). The server's ServerThread.processPayload() receives the PICK payload type and routes it to GameRoom.handlePick(), which validates the choice, stores it in the user's data, and marks them as having taken their turn.

After storing the choice, the server broadcasts the turn status to all connected clients by calling sendTurnStatus() for each client in the room. Each client's processPayload() routes this to processTurn(), which updates the knownClients map and triggers passToUICallback() for ITurnEvent listeners. UserListView.onTookTurn() receives this callback and updates the turn indicator from orange (pending) to green (picked), while also clearing the pending state for that player.

Saved: 12/8/2025 3:59:40 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

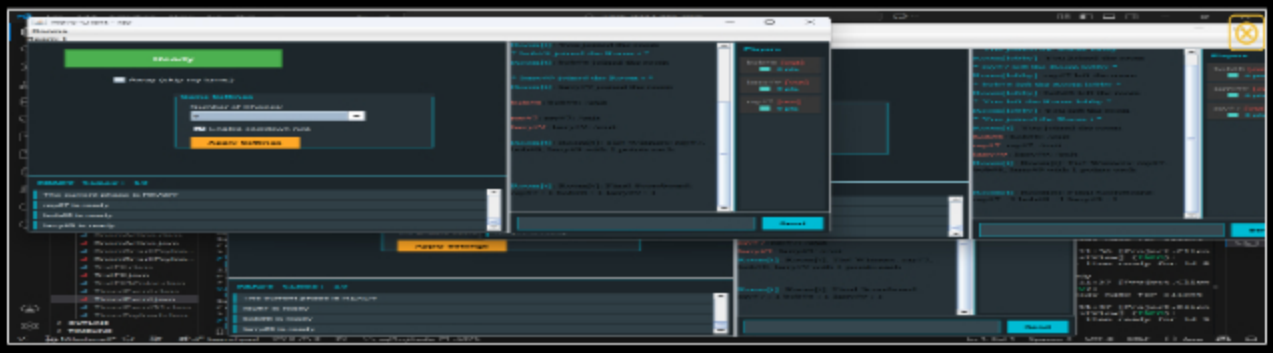
Part 1:

Progress: 100%

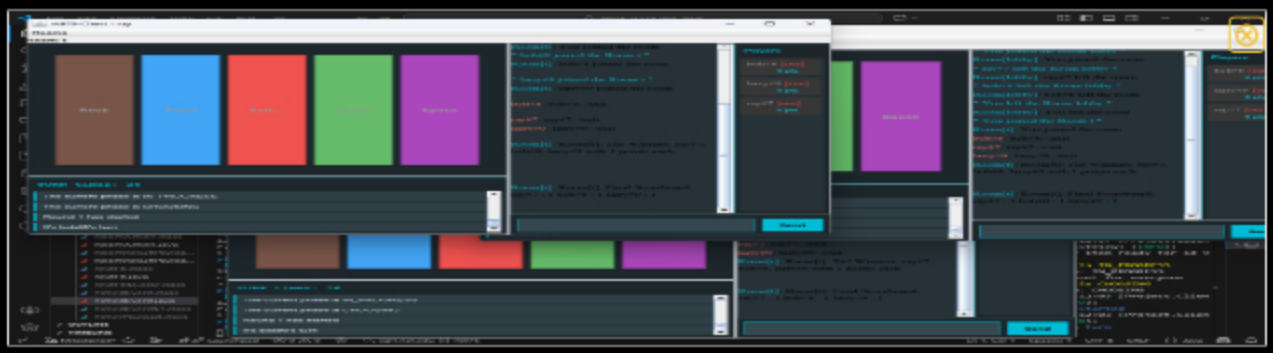
Details:

- 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)

extra choices



extra choices



extra choices

```java
@Override
public void onGameSettings(int optionCount, boolean cooldownEnabled) {
    // Check if current user is the creator
    boolean isCreator = Client.INSTANCE.isMyClientId(Client.INSTANCE.getCreatorClientId());

    // Disable cooldown checkbox if not the creator
    cooldownBox.setEnabled(isCreator);
    optionCombo.setEnabled(isCreator);

    // Update visual feedback
    if (!isCreator) {
        cooldownBox.setToolTipText("Only room creator can change cooldown setting");
```

```
        optionCombo.setToolTipText("Only room creator can change game options");
    } else {
        cooldownBox.setToolTipText("");
        optionCombo.setToolTipText("");
    }
}
```

code snippets

```
public void handleSettings(ServerThread sender, int optionCount, boolean cooldown) {
    // only allow if sender is in room
    try {
        checkPlayerInRoom(sender);
    } catch (Exception e) {
        sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "You must be in a GameRoom to change setti
        return;
    }
    // Only allow the room creator to change settings
    if (sender.getClientId() != creatorClientId) {
        sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "Only the room creator can change game set
        return;
    }
    this.optionCount = Math.max(3, Math.min(5, optionCount));
    this.cooldownEnabled = cooldown;
    // Inform clients
    GameSettingsPayload gp = new GameSettingsPayload();
    gp.setPayloadType(PayloadType.GAME_SETTINGS);
    gp.setOptionCount(this.optionCount);
    gp.setCooldownEnabled(this.cooldownEnabled);
    gp.setCreatorClientId(this.creatorClientId);
    clientsInRoom.values().forEach(st -> st.sendToClient(gp));
    relay(null, String.format("Game settings updated: options %d cooldown %b", this.optionCount, t
}
```

code snippets

```
@Override
public void onGameSettings(int optionCount, boolean cooldownEnabled) {
    this.currentOptionCount = optionCount;
    this.cooldownEnabled = cooldownEnabled;
    // only enable if local player is not away/spectator/eliminated and phase allows
    boolean canPlay = !(isAway || isSpectator || isEliminated) && (currentPhase == Phase.IN_PROGRE
    rockBtn.setEnabled(canPlay);
    paperBtn.setEnabled(canPlay);
    scissorsBtn.setEnabled(canPlay);
    // extras: enable lizard only for optionCount >= 4, Spock only for optionCount >= 5
    boolean lizardEnabled = optionCount >= 4 && canPlay;
    boolean spockEnabled = optionCount >= 5 && canPlay;
    lizardBtn.setEnabled(lizardEnabled);
    spockBtn.setEnabled(spockEnabled);
    // apply cooldown UI if applicable
    applyCooldownUI();
}
```

code snippets

```
        // validate against allowed options
        if (optionCount == 3) {
            if (!(c.equals("r") || c.equals("p") || c.equals("s"))) {
                currentUser.sendMessage(Constants.DEFAULT_CLIENT_ID, "Choice must be r, p, or s");
                return;
            }
        } else {
            if (!(c.equals("r") || c.equals("p") || c.equals("s") || c.equals("l") || c.equals("k"
                currentUser.sendMessage(Constants.DEFAULT_CLIENT_ID, "Invalid choice for current g
                return;
            }
        }
```

code snippets

```
@SuppressWarnings("all")
private boolean beats(String a, String b) {
    // returns true if a beats b in RPS
    if (a == null || b == null)
        return false;
    // standard 3-option rules
    if (optionCount == 3) {
        if (a.equals("r") && b.equals("s"))
            return true;
        if (a.equals("s") && b.equals("p"))
            return true;
        return a.equals("p") && b.equals("r");
    }
    // 5-option (Rock Paper Scissors Lizard Spock)
    // r beats s and l
    if (a.equals("r") && (b.equals("s") || b.equals("l"))) return true;
    // p beats r and k (spock doesn't make sense, but standard is paper disproves spock?)
    if (a.equals("p") && (b.equals("r") || b.equals("k"))) return true;
    // s beats p and l
    if (a.equals("s") && (b.equals("p") || b.equals("l"))) return true;
    // l beats p and k
    if (a.equals("l") && (b.equals("p") || b.equals("k"))) return true;
    // k (spock) beats r and s
    if (a.equals("k") && (b.equals("r") || b.equals("s"))) return true;
    return false;
}
```

code snippets

Saved: 12/8/2025 4:16:10 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:

The host controls game options through ReadyView, which has a JComboBox with values 3, 4, or 5 representing the number of available choices. When the host clicks "Apply Settings", Client.sendGameSettings() sends a GameSettingsPayload to the server with the selected optionCount and cooldown boolean. The server's handleSettings() first verifies the sender is the room creator by comparing their clientId to creatorClientId, rejecting the request if they don't match, then broadcasts the settings to all clients.

I went with the optionCount approach where 3 enables Rock/Paper/Scissors only, 4 adds Lizard, and 5 adds both Lizard and Spock. When clients receive the GameSettingsPayload, PlayView.onGameSettings() enables or disables the Lizard and Spock buttons based on whether optionCount >= 4 or >= 5 respectively. On the server side, handlePick() validates that the player's choice is allowed for the current optionCount before accepting it.

The beats() method in GameRoom handles battle logic differently based on optionCount. For 3 options it uses standard Rock-Paper-Scissors rules, while for 5 options it implements the extended rules where each choice beats two others (e.g., Rock beats Scissors and Lizard, Spock beats Rock and Scissors).

Saved: 12/8/2025 4:16:10 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

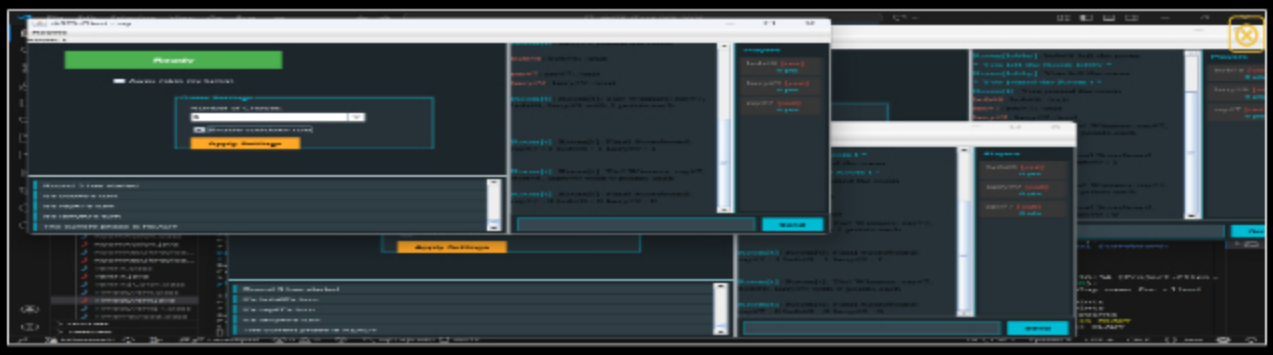
Part 1:

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 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



cooldown

cooldown

```
// Game settings UI (option count and cooldown)
JLabel optionsLabel = new JLabel("Number of Choices:");
optionsLabel.setForeground(TEXT_LIGHT);
optionsLabel.setFont(new Font("SansSerif", Font.PLAIN, 12));
optionsLabel.setAlignmentX(Component.LEFT_ALIGNMENT);

Integer[] opts = {3, 4, 5};
optionCombo = new JComboBox<>(opts);
optionCombo.setBackground(Color.WHITE);
optionCombo.setMaximumSize(new Dimension(200, 30));
optionCombo.setAlignmentX(Component.LEFT_ALIGNMENT);

cooldownBox = new JCheckBox("Enable cooldown rule");
cooldownBox.setFont(new Font("SansSerif", Font.PLAIN, 12));
cooldownBox.setForeground(TEXT_LIGHT);
```

code snippets

```
// Only allow the room creator to change settings
if (sender.getClientId() != creatorClientId) {
    sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "Only the room creator can change game set
    return;
}
```

code snippets

```
@Override
public void onTookTurn(long clientId, boolean didTakeTurn) {
    // RESET_TURN is signaled via clientId == Constants.DEFAULT_CLIENT_ID
    long resetId = Project.common.Constants.DEFAULT_CLIENT_ID;
    // Only care about local client's confirmed turn or global reset
    if (clientId == resetId) {
        // Round has reset: move this round's confirmed pick into previousRoundPick
        // so it becomes the cooldowned option for this new round, then clear currentRoundPick
        this.previousRoundPick = this.currentRoundPick;
```

```
            this.currentRoundPick = null;
            this.pendingSentPick = null;
            javax.swing.SwingUtilities.invokeLater(this::applyCooldownUI);
            return;
        }
        // if this is a confirmation that "we" took our turn, apply cooldown
        if (Client.INSTANCE.isMyClientIdSet() && Client.INSTANCE.isMyClientId(clientId) && didTakeCurr
            // confirm pending pick as our last pick
            if (this.pendingSentPick != null) {
                // store as current round pick; cooldown will be applied at next round reset
                this.currentRoundPick = this.pendingSentPick;
                this.pendingSentPick = null;
                javax.swing.SwingUtilities.invokeLater(this::applyCooldownUI);
            }
        }
    }
```

code snippets

```
private void applyCooldownUI() {
    // if cooldown not enabled, ensure no cooldown disables linger
    boolean canPlay = !(isAway || isSpectator || isEliminated) && (currentPhase == Phase.IN_PROGRE
    // base enablement
    rockBtn.setEnabled(canPlay);
    paperBtn.setEnabled(canPlay);
    scissorsBtn.setEnabled(canPlay);
    lizardBtn.setEnabled(canPlay && currentOptionCount >= 4);
    spockBtn.setEnabled(canPlay && currentOptionCount >= 5);

    if (!cooldownEnabled || previousRoundPick == null) return;
    JButton btn = buttonForShortChoice(previousRoundPick);
    if (btn != null) btn.setEnabled(false);
}
```

code snippets

```
        // cooldown enforcement
        if (cooldownEnabled && currentUser.user.getLastChoice() != null && currentUser.user.getLa
            currentUser.sendMessage(Constants.DEFAULT_CLIENT_ID, "This option is on cooldown for
            return;
        }
        currentUser.user.setChoice(c);
        currentUser.user.setLastChoice(c);
```

code snippets

Saved: 12/8/2025 4:33:12 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:

The host toggles the cooldown feature through a checkbox in ReadyView, which sends a GameSettingsPayload to the server containing the cooldownEnabled boolean. The server's handleSettings() verifies the sender is the room creator before accepting the change, then broadcasts the updated settings to all clients so PlayView can enable or disable the cooldown UI logic.

Each user's choice is tracked in two fields on the server: choice stores the current round's pick, and lastChoice stores the previous round's pick for cooldown enforcement. When a player

submits a pick, handlePick() checks if cooldownEnabled is true and if their new choice matches lastChoice, rejecting it with an error message if so. After a valid pick is accepted, the server stores it in both choice and lastChoice so it becomes the cooldowned option for the next round.

On the client side, PlayView tracks currentRoundPick and previousRoundPick separately. When a new round starts (signaled by RESET_TURN), the current pick moves into previousRoundPick, and applyCooldownUI() disables the button matching that choice. When the game session ends in onSessionEnd(), all user data including lastChoice is reset to null so cooldown does not carry over to the next game.

Saved: 12/8/2025 4:33:12 PM

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

Progress: 100%

Task #1 (1 pt.) - Away Status

Progress: 100%

Details:

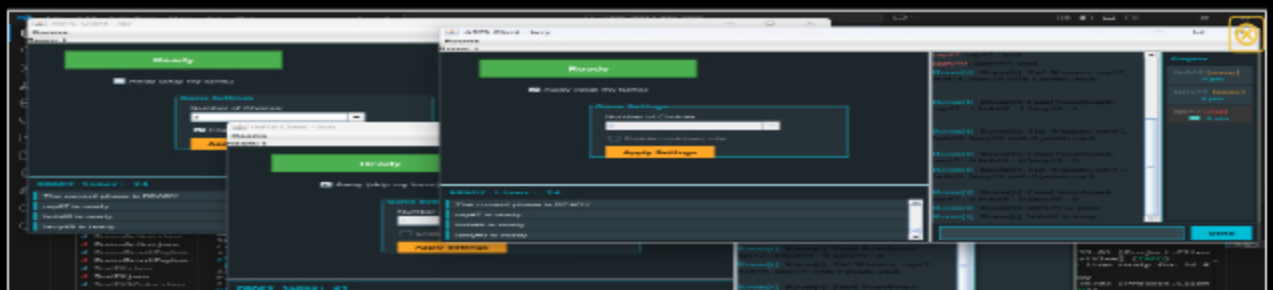
- Clients can mark themselves away and be skipped in turn flow but still part of the game
- The status should be visible to all participants
- A message should be relayed to the Game Events Panel (i.e., Bob is away or Bob is no longer away)
- The user list should have a visual representation (i.e., grayed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



away

```
    // Away toggle
    JCheckBox awayBox = new JCheckBox("Away (skip my turns)");
    awayBox.setFont(new Font("SansSerif", Font.PLAIN, 13));
    awayBox.setForeground(TEXT_LIGHT);
    awayBox.setBackground(BACKGROUND_DARK);
    awayBox.setAlignmentX(Component.CENTER_ALIGNMENT);
    awayBox.addActionListener(e -> {
        boolean away = awayBox.isSelected();
        try {
            Client.INSTANCE.sendAway(away);
        } catch (IOException ex) {
```

code snippets

```
public void sendAway(boolean away) throws IOException {
    PlayerStatePayload pp = new PlayerStatePayload();
    pp.setPayloadType(PayloadType.PLAYER_STATE);
    pp.setClientId(myUser.getClientId());
    pp.setAway(away);
    sendToServer(pp);
}
```

code snippets

```
public void handlePlayerState(ServerThread sender, PlayerStatePayload pp) {
    try {
        checkPlayerInRoom(sender);
        boolean prevAway      = sender.user.isAway();
        boolean prevSpectator = sender.user.isSpectator();
        if (pp.isAway() != prevAway) {
            sender.user.setAway(pp.isAway());
        }
        if (pp.isSpectator() != prevSpectator) {
            sender.user.setSpectator(pp.isSpectator());
        }
        // broadcast updated player state to all clients
        PlayerStatePayload out = new PlayerStatePayload();
        out.setPayloadType(PayloadType.PLAYER_STATE);
        out.setClientId(sender.getClientId());
        out.setPoints(sender.user.getPoints());
        out.setEliminated(sender.user.isEliminated());
        out.setAway(sender.user.isAway());
        out.setSpectator(sender.user.isSpectator());
        currentRoom.values().forEach(sr -> sr.sendToClient(out));
        // send a human-readable game event when away status changes
        if (prevAway != sender.user.isAway()) {
            if (sender.user.isAway()) {
                relay(null, String.format("%s is away", sender.getDisplayName()));
            } else {
                relay(null, String.format("%s is no longer away", sender.getDisplayName()));
            }
```

code snippets

```
@Override
public void onPlayerStateUpdate(long clientId, int points, boolean eliminated, boolean away, boo
    if (clientId == Constants.DEFAULT_CLIENT_ID) {
        SwingUtilities.invokeLater(() -> {
            userItemsMap.values().forEach(u -> u.setEliminated(false));
            userItemsMap.values().forEach(u -> u.setAway(false));
        });
        return;
    }
    if (userItemsMap.containsKey(clientId)) {
        SwingUtilities.invokeLater(() -> {
            UserListItem item = userItemsMap.get(clientId);
            item.setPoints(points);
            item.setEliminated(eliminated);
            item.setAway(away);
            item.setSpectator(spectator);
            refreshUserList();
        });
    }
}
```

code snippets

```
private void setTurnOrder() {
    turnOrder.clear();
    // Exclude spectators, away, and eliminated players from the turn order
```

```
        turnOrder = clientsInRoom.values().stream()
                .filter(sp -> sp.isReady() && !sp.user.isSpectator() && !sp.user.isAway() && !sp.user.
                .collect(Collectors.toList());
        Collections.shuffle(turnOrder);
    }
```

code snippets

```
clientsInRoom.values().forEach(sp -> {
        if (!sp.user.isEliminated() && !sp.user.isSpectator() && !sp.user.isAway() && sp.user.
        sp.user.setEliminated(true);
```

code snippets

Saved: 12/8/2025 4:43:32 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:

When a player clicks the Away checkbox in ReadyView, it calls Client.sendAway() which creates a PlayerStatePayload with the away boolean and sends it to the server. The server's handlePlayerState() updates the user's away flag, creates a new PlayerStatePayload with all current state values, and broadcasts it to every client in the room. Each client's processPlayerState() receives this and triggers onPlayerStateUpdate() in UserListView, which updates the display to show the "away" tag and grayed styling. The server also sends a human-readable message like "PlayerName is away" through relay() which appears in the chat.

On the server side, away users are excluded from game logic in multiple places. The setTurnOrder() method filters out away users when building the turn order list, so they never get assigned a turn. The allActivePlayersHaveChosen() method excludes away users from the count, so the round can end without waiting for them to pick. In processRoundResults(), the elimination check explicitly skips away users so they are not eliminated for failing to pick during rounds they were marked away.

Saved: 12/8/2025 4:43:32 PM

Task #2 (1 pt.) - Spectators

Progress: 100%

Details:

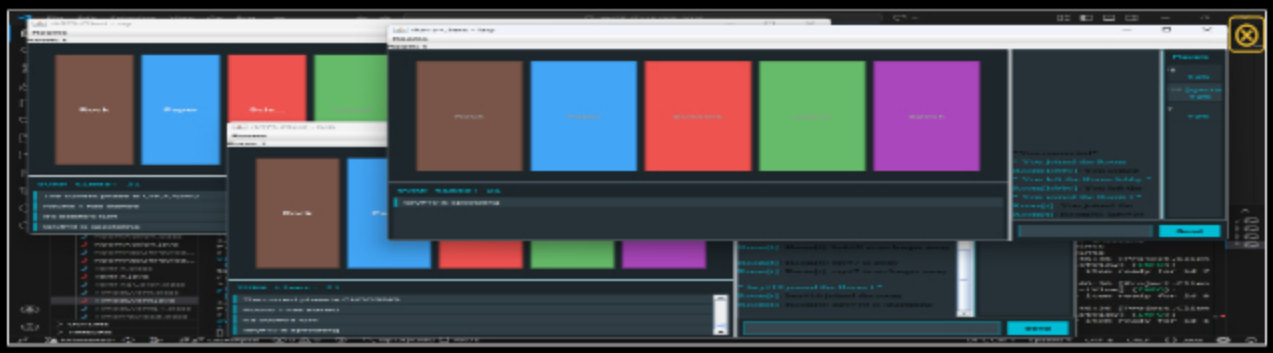
- Spectators are users who didn't mark themselves ready
  - Optionally you can include a toggle on the Ready Check page
- The can see all chat but are ignored from turn/round actions and can't send messages
- Spectators will have a visual representation in the user list to distinguish them from other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

Part 1:

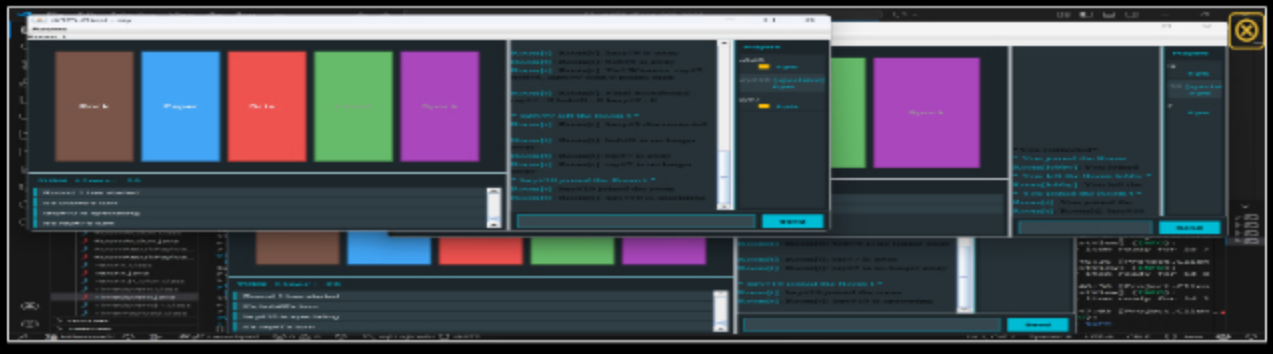
Progress: 100%

Details:

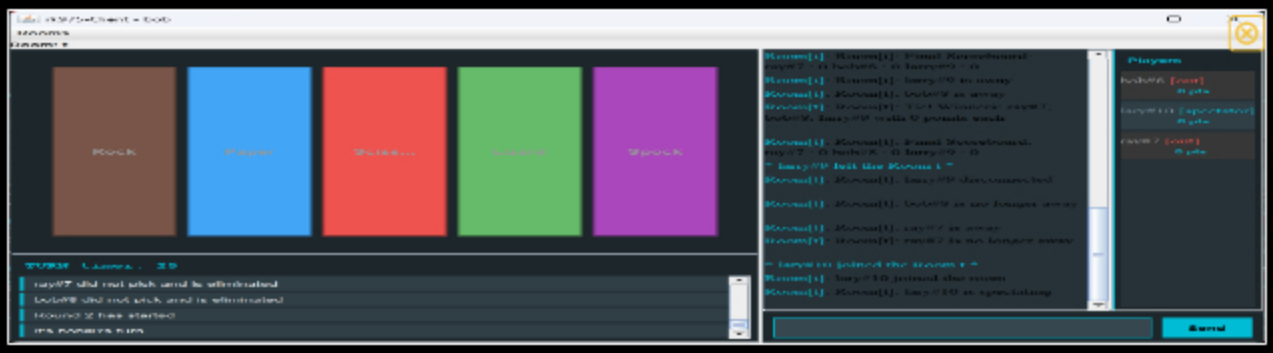
- Show the UI indicator of a spectator (visual and message)
- Show the related code flow from UI to server-side back to UI for showing the status
- Show the related code flow for sending the message to Game Events Panel
- Show various examples across 3+ clients of spectator status (including Game Events 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)



spectator



spectator



spectator

```
private void updateDisplayText() {
    StringBuilder suffix = new StringBuilder();
    String statusColor = "#ECEFF1"; // default light text

    if (eliminatedFlag) {
        suffix.append(" <font color='#EF5350'>[out]</font>");
        statusColor = "#9E9E9E";
    }
    if (spectatorFlag) {
        suffix.append(" <font color='#00BCD4'>[spectator]</font>");
        statusColor = "#78909C";
    }
    if (awayFlag) {
        suffix.append(" <font color='#FFA726'>[away]</font>");
        statusColor = "#78909C";
    }
```

code snippets

```
    // If joining during a non READY phase (game in progress), auto mark as spectator
    boolean joinsAsSpectator = false;
    if (currentPhase != Phase.READY) {
        sp.user.setSpectator(true);
        joinsAsSpectator = true;
        // Announce spectator join to Game Events panel
        String spectatorMessage = String.format("%s is spectating", sp.getDisplayName());
        clientsInRoom.values().forEach(st -> st.sendGameEvent(spectatorMessage));
        // Also relay to chat so it appears in chat area
        relay(null, spectatorMessage);
        // Broadcast the spectator state to all clients
        PlayerStatePayload out = new PlayerStatePayload();
        out.setPayloadType(PayloadType.PLAYER_STATE);
        out.setClientId(sp.getClientId());
        out.setPoints(sp.user.getPoints());
        out.setEliminated(sp.user.isEliminated());
        out.setAway(sp.user.isAway());
        out.setSpectator(true);
        clientsInRoom.values().forEach(st -> st.sendToClient(out));
    }
```

code snippets

```
private void setTurnOrder() {
    turnOrder.clear();
    // Exclude spectators, away, and eliminated players from the turn order
    turnOrder = clientsInRoom.values().stream()
            .filter(sp -> sp.isReady() && !sp.user.isSpectator() && !sp.user.isAway() && !sp.user.
            .collect(Collectors.toList());
    Collections.shuffle(turnOrder);
}
```

code snippets

```
    if (currentUser.user.isSpectator()) {
        currentUser.sendMessage(Constants.DEFAULT_CLIENT_ID, "Spectators cannot pick");
        return;
    }
```

```
    }
```

code snippets

```
protected synchronized void handleMessage(ServerThread sender, String text) {
    // Spectators may see chat but are not allowed to send messages
    try {
        if (sender != null && sender.user != null && sender.user.isSpectator()) {
            // Inform the spectator that they cannot send messages
            sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "Spectators cannot send messages in th
            return;
        }
    } catch (Exception e) {
        // fall through to normal behavior if any unexpected error occurs
    }
    relay(sender, text);
}
```

code snippets

```
    // sync GameRoom state to new client
    syncCreatorInfo(sp);
    syncCurrentPhase(sp);
    // sync only what's necessary for the specific phase
    // if you blindly sync everything, you'll get visual artifacts/discrepancies
    syncReadyStatus(sp);
    if (currentPhase != Phase.READY) {
        syncTurnStatus(sp); // turn/ready use the same visual process so ensure turn status is onl
                            // outside of ready phase
        syncPlayerPoints(sp);
    }
```

code snippets

```
@Override
protected void handleReady(ServerThread sender) {
    // If this is a spectator in READY phase, convert them back to a regular player
    if (sender != null && sender.user != null && sender.user.isSpectator() && currentPhase == Phas
        sender.user.setSpectator(false);
        // Notify all clients that this spectator is now a regular player
        PlayerStatePayload out = new PlayerStatePayload();
        out.setPayloadType(PayloadType.PLAYER_STATE);
        out.setClientId(sender.getClientId());
        out.setPoints(sender.user.getPoints());
        out.setEliminated(sender.user.isEliminated());
        out.setAway(sender.user.isAway());
        out.setSpectator(false);
        clientsInRoom.values().forEach(st -> st.sendToClient(out));
        // Announce the conversion
        clientsInRoom.values().forEach(st -> st.sendGameEvent(String.format("%s is now a player",
    }
    // Call parent implementation to handle the ready logic
    super.handleReady(sender);
}
```

code snippets

Saved: 12/8/2025 4:52:44 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:

When a player joins a room during an active game (phase is not READY), the server automatically marks them as a spectator by setting user.setSpectator(true) in onClientAdded(). The server then broadcasts a PlayerStatePayload to all clients containing the spectator flag, and each client's UserListView updates to show the "spectator" tag with grayed styling. A game event message announcing the spectator is also sent to all clients so it appears in the Game Events panel.

The server excludes spectators from game logic in multiple places. The setTurnOrder() method filters out spectators when building the turn order, and allActivePlayersHaveChosen() excludes them from the count so rounds can end without waiting for spectators. In handlePick(), the server explicitly checks if the user is a spectator and rejects their pick with an error message before any choice processing occurs.

Spectators are blocked from sending chat messages in Room.handleMessage(), which checks sender.user.isSpectator() and returns early with a message saying spectators cannot send messages in the room. This check happens before the relay() call so the message never reaches other clients.

Spectators receive full game state data through the same sync methods as regular players. When they join, syncCurrentPhase(), syncReadyStatus(), syncTurnStatus(), and syncPlayerPoints() are all called to give them complete visibility of the game in progress. They see all battle results, point updates, and turn indicators in real-time, but the key difference is they see other players' turn completion status without being able to participate themselves.

Saved: 12/8/2025 4:52:44 PM

Section #5: (1 pt.) Misc

Progress: 100%

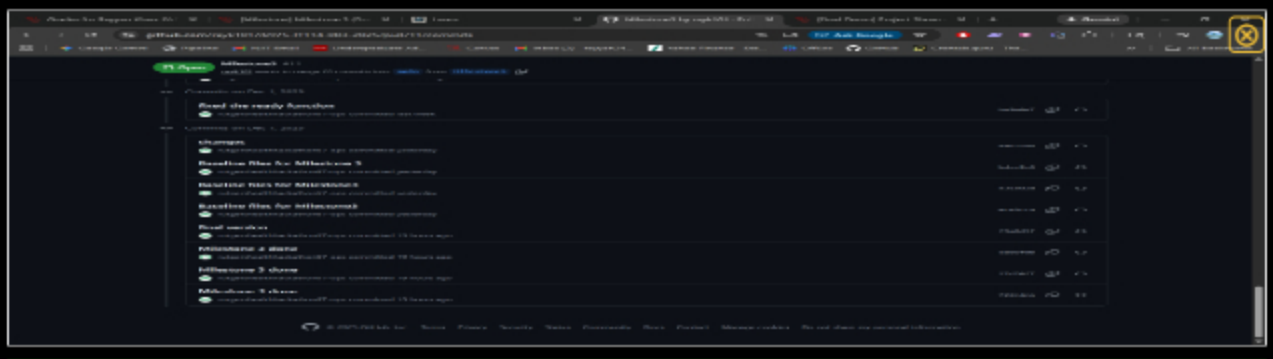
Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



commits

Saved: 12/8/2025 4:54:46 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/rayk101/rk975-IT114-003-2025/

URL
https://github.com/rayk101/rk975

Saved: 12/8/2025 4:54:46 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

wakatime

wakatime

Saved: 12/8/2025 4:57:24 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:

When a player joins during an active game, the server automatically marks them as a spectator and broadcasts a PlayerStatePayload to all clients, which updates the user list to show the "spectator" tag. The server excludes spectators from game logic by filtering them out in setTurnOrder() and allActivePlayersHaveChosen(), so rounds proceed without waiting for them.

Spectators cannot participate or communicate because handlePick() rejects their choices and handleMessage() blocks their chat messages before they reach other clients. However, spectators receive full visibility of the game through the same sync methods as regular players, seeing all battle results, point updates, and turn indicators in real-time. When the game ends and returns to READY phase, spectators can click Ready to convert back to regular players for the next session.

Saved: 12/8/2025 4:58:06 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 assignment was implementing the core Rock Paper Scissors game logic in the beats() method, since it follows simple conditional rules where each choice beats specific others. Setting up the basic UI components in Swing was also straightforward because the framework for button

panels, labels, and layouts was already familiar from earlier milestones. The ready status flow was relatively simple to implement since it only required toggling a boolean flag and broadcasting it to all clients using the existing payload system.

The message relay system for chat was also easy to work with because the base Room class already had the infrastructure for broadcasting messages to all connected clients. Extending the existing payload types to add new ones like PickPayload and PlayerStatePayload followed the same pattern as previous payloads, making it a repetitive but simple task.

Saved: 12/8/2025 4:58:40 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 of the assignment was managing state synchronization across multiple clients, ensuring that all players see consistent data for points, turn indicators, and player statuses at the same time. Implementing the spectator feature was challenging because it required handling mid-game joins, syncing all existing game state to the new client, and preventing them from participating while still showing them everything.

The phase transitions between READY, IN_PROGRESS, and CHOOSING required careful coordination to ensure the UI switched views correctly and buttons enabled or disabled at the right times. Tracking the cooldown feature was tricky because it needed to remember the previous round's choice separately from the current round and only apply the restriction after a round reset. Debugging issues where one client saw different data than another was time-consuming since it required tracing the payload flow from server broadcast through client processing to UI callback.

Saved: 12/8/2025 4:59:06 PM