

CIS 4910 – Team AI Usage Log Template (with Prompt History)

Team Name: GameHub

Sprint #: 2

Date: 10/14/2025

1. AI Tool(s) Used

List all AI tools used this sprint (e.g., ChatGPT, GitHub Copilot, DALL·E, diagramming tools).

- Tool 1: GitHub Copilot
- Tool 2: Gemini
- Tool 3: N/A

2. Purpose of AI Use

Briefly describe what task(s) the AI supported. (Check all that apply)

- ☒ Brainstorming requirements or design alternatives
- ☐ Drafting user stories/backlog items
- ☒ Writing or refining code
- ☒ Debugging / troubleshooting errors
- ☐ Generating or suggesting test cases
- ☐ Drafting documentation (manuals, tutorials, design specs)
- ☐ Reviewing code for quality or security
- ☐ Other:

3. Prompt History & Refinement

Document at least two iterations of prompts and outputs. Show how the team improved or refined the request to get better results.

Iteration	Prompt	AI Response (summary) (Attached Below)	Team Action (Accepted / Edited / Rejected)
1	Hello, so currently in my FrontEnd script.js file when a play wins it gets announced using the initializeBoard() function. However, what I want to	*1	Accepted / Edited / Rejected

change is instead of the user clicking "Ok" when the player wins screen pops up they should see a "Play Again" button. So I would keep the message stating who won but they would click a button that says "Play Again" to reset the game rather than "Ok".

2

Okay now I want to add scores. Once a game has started scores should be kept for each player until the application is closed. That means after a game is reset the score is still kept for each player. Put these scores in the top left and top right corners. The Red player should be in the top left. The Yellow player should be in the top right.

*2

Accepted / Edited / Rejected

3

so currently I have a connect four game with assets that are connect four but im trying to disassociate with connect four so i would like new two new square assets


Accepted / Edited / Rejected

that have a cool GH
in the middle and
the color scheme is
royal blue and coral
and have it be
transparent please
and lets do an
engraved font style
and lets do a 3d
textured like a game
piece and keep
them both identical
in everything except
color that includes
size

4. Share Links to AI Sessions

Provide links to the AI chat(s) or session(s) used.

- Link 1*:

 Restore Checkpoint

Hello, so currently in my FrontEnd script.js file when a play wins it gets announced using the initializeBoard() function. However, what I want to change is instead of the user clicking "Ok" when the player wins screen pops up they should see a "Play Again" button. So I would keep the message stating who won but they would click a button that says "Play Again" to reset the game rather than "Ok".

JS script.js

I'll update `FrontEnd/script.js` to replace the blocking `alert()` dialogs with an in-page modal that shows the win/draw message and a "Play Again" button. The button will close the modal, emit the 'reset-game' event, and call the existing `resetGame()`.

I'll now apply the patch to modify the two alert usages and add the `showResultModal` helper.

JS script.js +65 -0



I added an in-page modal helper to `FrontEnd/script.js` that shows the win/draw message and a "Play Again" button. The button hides the modal, emits 'reset-game' to the server, and calls the existing `resetGame()`.

What I changed

- Added `showResultModal(message)` near the top of `script.js`. It creates (once) a centered modal with the message and a "Play Again" button.

Next steps (small)

- Replace the existing `alert(...)` calls with `showResultModal(...)` where wins/draws are detected (both local moves and opponent moves). I can do that now unless you'd prefer to keep some alerts.

GPT-5 mini • 1x

```

JS script.js  X  index.html  # styles.css  JS script.js  script.js  X
~/Desktop/GameHub/GameHubFrontEnd/script.js
21+
22+ function showResultModal(message) {
23+     let modal = document.getElementById('resultModal');
24+     if (!modal) {
25+         modal = document.createElement('div');
26+         modal.id = 'resultModal';
27+         Object.assign(modal.style, {
28+             position: 'fixed',
29+             top: '0',
30+             left: '0',
31+             width: '100%',
32+             height: '100%',
33+             display: 'flex',
34+             alignItems: 'center',
35+             justifyContent: 'center',
36+             backgroundColor: 'rgba(0,0,0,0.5)',
37+             zIndex: '9999',
38+         });
39+
40+         const content = document.createElement('div');
41+         content.id = 'resultContent';
42+         Object.assign(content.style, {
43+             background: '■ #fff',
44+             padding: '20px',
45+             borderRadius: '8px',
46+             textAlign: 'center',
47+             minWidth: '260px',
48+             boxShadow: '0 6px 20px 0 rgba(0,0,0,0.3)'
49+         });
50+
51+         const msg = document.createElement('p');
52+         msg.id = 'resultMessage';
53+         msg.style.marginBottom = '16px';
54+         content.appendChild(msg);
55+
56+         const btn = document.createElement('button');
57+         btn.id = 'playAgainBtn';
58+         btn.textContent = 'Play Again';
59+         Object.assign(btn.style, {
60+             padding: '8px 14px',
61+             fontSize: '14px',
62+             cursor: 'pointer',
63+             borderRadius: '6px',
64+             border: 'none',
65+             background: '■ #007bff',
66+             color: '■ #fff'
67+         });
68+
69+         btn.addEventListener('click', () => {
70+             modal.style.display = 'none';
71+             // full screen and reset locally
72+             try { socket.emit('reset-game'); } catch (e) { /* ignore */ }
73+             try { resetGame(); } catch (e) { /* ignore */ }
74+         });
75+
76+         content.appendChild(btn);
77+         modal.appendChild(content);
78+         document.body.appendChild(modal);
79+     }
80+
81+     const msgEl = document.getElementById('resultMessage');
82+     if (msgEl) msgEl.textContent = message;
83+     modal.style.display = 'flex';
84+ }

```

- Link 2*:

Okay now I want to add scores. Once a game has started scores should be kept for each player until the application is closed. That means after a game is reset the score is still kept for each player. Put these scores in the top left and top right corners. The Red player should be in the top left. The Yellow player should be in the top right.

JS script.js:58

I'll add in-memory scores and render them in the top-left/top-right via JavaScript, and increment scores when a player wins (both local and opponent). I'll update `FrontEnd/script.js` now and then mark the todo items completed.

✓ Created 4 todos

I'll update `FrontEnd/script.js` to add score variables, UI elements, and logic to increment and display scores.

JS script.js +51 -0

Now I'll patch the places where a win is detected to increment the appropriate player's score and re-render the score UI. This includes the local click handler (`initializeBoard`) and the opponent-move handler.

JS script.js +6 -1

I'll update the todo list: mark the first two items completed and the third (keep scores across resets) in-progress.

✓ Starting (3/4) *Keep scores across resets*

Progress update — what I changed

- Added in-memory scores: `redScore` and `yellowScore`.
- Added `renderScores()` which creates two small, fixed UI elements:
 - Red score at top-left (`#redScore`)
 - Yellow score at top-right (`#yellowScore`)
- Incremented the appropriate score when a win is detected (both local move and opponent move), and call `renderScores()` so the UI updates.
- Left scores persistent in-memory across resets (I did not modify `resetGame()` to touch scores).

Files changed

- `FrontEnd/script.js` — added score state, UI helper, and win-score updates.

How it works

- Scores are kept in JavaScript memory while the page is open. They survive board resets because `resetGame()` only clears the board and does not reset the score variables.
- When a player wins:
 - We increment `redScore` or `yellowScore` depending on the winner.
 - Call `renderScores()` to update the displayed values.
 - Show the result modal (you already have the "Play Again" modal added earlier) which will emit `reset-game` and call `resetGame()` when the user clicks "Play Again".

```

let chatClient; // will hold the StreamChat client instance

// Scores (kept in-memory while the page is open)
let redScore = 0;
let yellowScore = 0;

function renderScores() {
  let redEl = document.getElementById('redScore');
  let yellowEl = document.getElementById('yellowScore');

  if (!redEl) {
    redEl = document.createElement('div');
    redEl.id = 'redScore';
    Object.assign(redEl.style, {
      position: 'fixed',
      top: '8px',
      left: '12px',
      fontSize: '16px',
      fontWeight: '600',
      color: 'red',
      background: '■ rgba(255,255,255,0.8)',
      padding: '6px 10px',
      borderRadius: '6px',
      zIndex: '1000'
    });
    document.body.appendChild(redEl);
  }

  if (!yellowEl) {
    yellowEl = document.createElement('div');
    yellowEl.id = 'yellowScore';
    Object.assign(yellowEl.style, {
      position: 'fixed',
      top: '8px',
      right: '12px',
      fontSize: '16px',
      fontWeight: '600',
      color: 'goldenrod',
      background: '■ rgba(255,255,255,0.8)',
      padding: '6px 10px',
      borderRadius: '6px',
      zIndex: '1000'
    });
    document.body.appendChild(yellowEl);
  }

  redEl.textContent = `Red: ${redScore}`;
  yellowEl.textContent = `Yellow: ${yellowScore}`;
}

// Initialize scores UI
renderScores();

```

```

gameover = true;

// Increment score for current player
if (currentPlayer === PLAYER1) redScore++; else yellowScore++;
renderScores();
setTimeout(() => {
  showResultModal(`${currentPlayer.toUpperCase()} wins!`);
}, 100);

```

```

// Increment score for winning player
if (data.player === PLAYER1) redScore++; else yellowScore++;
renderScores();

```

- Link 3: <https://g.co/gemini/share/3306ce0d74dd>

(If a tool doesn't allow link sharing, attach screenshots as evidence.)

5. Outputs Received & Team Revisions

Summarize final outputs adopted into the project and explain how they were refined by the team.

AI Generated several Assets, We rejected most and kept 2. AI additionally helped in implementing several stories and debugging.

Example:

AI generated 8 user stories. We kept 5, rephrased 2, and rejected 1. Final stories were added to the backlog.

6. Ethical & Professional Considerations

- Did the AI output raise accuracy, bias, or copyright concerns? No
- How did the team ensure compliance with the course AI policy? Accurately recorded use of AI

7. Contribution to Project

Briefly describe how this AI has advanced your project this sprint.

The use of AI helped save the time it would take to create an interactive gameboard with login and live game chat features. Additionally, AI filled in the gap for skills the team does not possess (E.g Asset creation)

9. Sign-Off

We confirm this log accurately reflects the team's AI usage.

Team Members (type names):

- Benito Reyes
- Mekhi Green
- Ramsey Burls
- Brandon Brown

☒ Submission Instructions: Upload this log with each sprint package in Canvas. Ensure prompt history and share links are included.

Ethical Implications

The team recognizes that building an online multiplayer platform involving user accounts, chat functionality, and real-time gameplay introduces ethical considerations regarding data privacy, responsible communication, and intellectual property. Our system features a secure user login, randomized game room codes, and a real-time chat system for players and spectators. The following sections outline the ethical risks, mitigation measures, and ongoing monitoring plans.

1. Ethical Risks

a. Copyright

- i. Established companies originally own similar games included. Replicating its rules, name, or visual features without permission could infringe on intellectual property rights.

b. Data Security and Privacy

- i. Enabling user accounts introduces risks like unauthorized access, password leaks, or improper data storage.
- ii. Although our design doesn't store unnecessary personal data, handling credentials can risk confidentiality if not correctly secured.

c. Chat Misuse and User Conduct

- i. The live chat feature could be exploited for harassment, hate speech, or NSFW language, creating a toxic environment for players and spectators.

2. Mitigation Strategies

a. Copyright Compliance

- i. Our implementation of games avoids copyrighted assets, using only the generic gameplay mechanics under fair-use principles.
- ii. The final configuration will feature original designs, artwork, and naming conventions to ensure the project's independence from commercial versions.

b. Data Protection and Security

- i. All user passwords are encrypted before transmission and storage.
- ii. No sensitive personal data (such as addresses or payment details) is collected or stored.
- iii. Access tokens and game room identifiers are randomly generated, preventing unauthorized access to sessions.

c. Safe Communication Channels

- i. A basic moderation system will be implemented to minimize inappropriate messages.

3. Monitoring and Continuous Improvement

a. Routine Security Testing

- i. We will conduct regular vulnerability checks and audits of authentication endpoints and encryption methods to prevent data leaks.

- b. Feedback Mechanisms**
 - i. Encourage player reporting and feedback on technical or ethical concerns to promote transparency and responsiveness.**
- c. Version Control documentation**
 - i. Maintain detailed commit logs for updates, ensuring traceability of security-related changes.**
- d. Ethical Review Iterations**
 - i. With each major update, we will re-evaluate emerging risks, especially as more games are added to adapt to policies and protections.**