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

Team Name: Byte Bros

Sprint #: 1

Date: 9/22/25

1. AI Tool(s) Used

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

- Tool 1: Copilot
- Tool 2: Cursor
- Tool 3: ________________________

2. Purpose of AI Use

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

[X] Brainstorming requirements or design alternatives
[ ] Drafting user stories / backlog items
[X] Writing or refining code
[X] 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

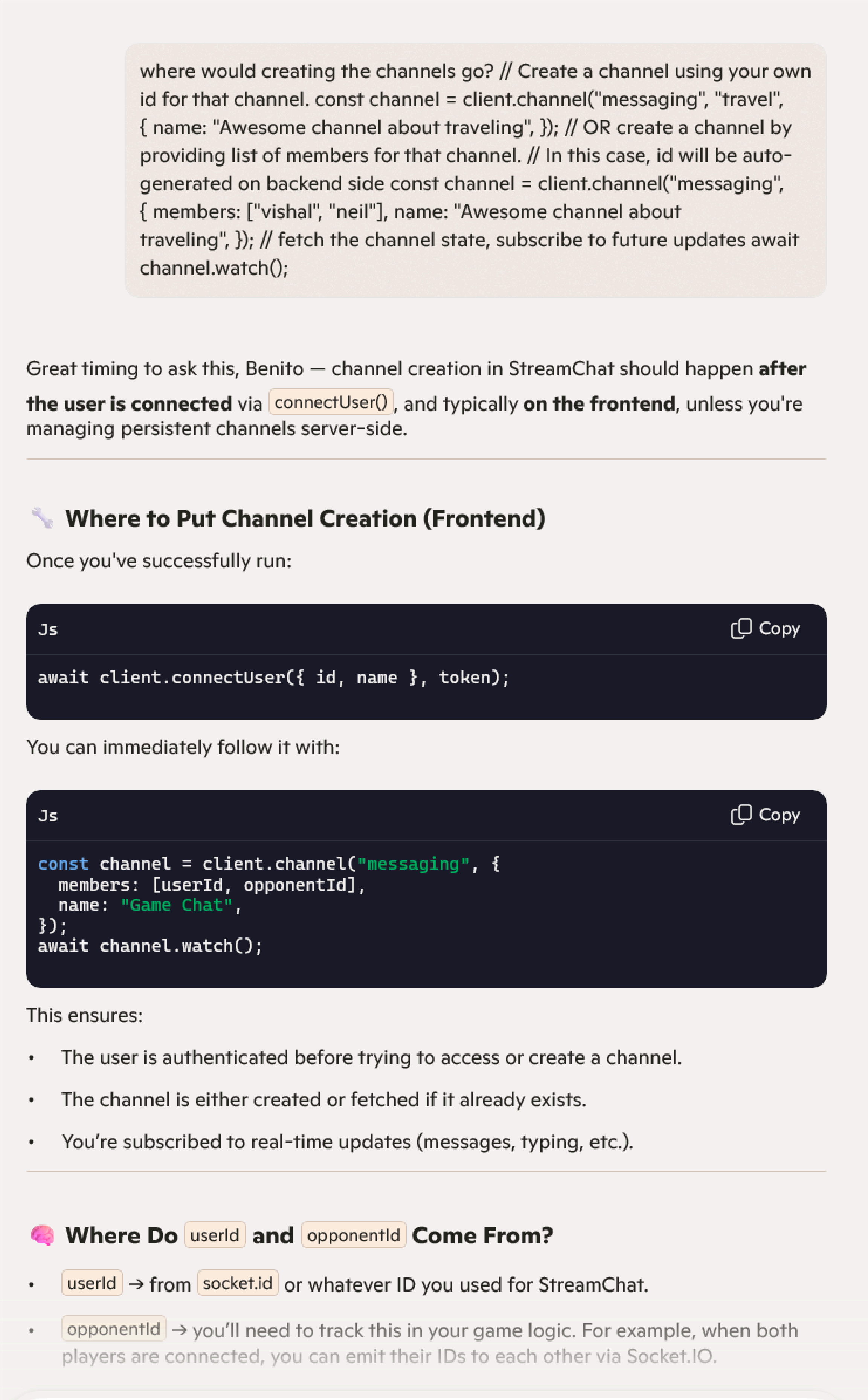
Document at least two iterations of prompts and outputs. Show how the team improved or refined the request to get better results.

Iteration	Prompt	AI Response (summary)	Team Action (Accepted / Edited / Rejected)
1	where would creating the channels go? // Create a channel using your own id for that channel. const channel = client.channel("mes	(See Below) *1	Accepted / Edited / Rejected

	saging", "travel", { name: "Awesome channel about traveling", }); // OR create a channel by providing list of members for that channel. // In this case, id will be auto-generated on backend side const channel = client.channel("messaging", { members: ["vishal", "neil"], name: "Awesome channel about traveling", }); // fetch the channel state, subscribe to future updates await channel.watch();		
2	i would like to add a multiplayer component using sockets and node.js can you explain each step in implementing that, one player hosting, and another player joining	(See Below) *2	Accepted / Edited / Rejected
3	I am having troble with the game, the problem i have is that the game is not registering the win	(See Below) *3	Accepted / Edited / Rejected

	condition, only the draw condition		
4	Check the logic in the script.js file. I want there to be a function that detects if there is a draw so that the match ends fairly if the board is full.	AI Code here	Accepted / Edited / Rejected
5	I now want the game to reset after a match ends, so that a new game can start without restarting the entire application.	AI Code here	Accepted / Edited / Rejected

*1



where would creating the channels go? // Create a channel using your own id for that channel. const channel = client.channel("messaging", "travel", { name: "Awesome channel about traveling", }); // OR create a channel by providing list of members for that channel. // In this case, id will be auto-generated on backend side const channel = client.channel("messaging", { members: ["vishal", "neil"], name: "Awesome channel about traveling", }); // fetch the channel state, subscribe to future updates await channel.watch();

Great timing to ask this, Benito — channel creation in StreamChat should happen **after the user is connected** via `connectUser()`, and typically **on the frontend**, unless you're managing persistent channels server-side.

🔧 Where to Put Channel Creation (Frontend)

Once you've successfully run:

```js
await client.connectUser({ id, name }, token);
```

You can immediately follow it with:

```js
const channel = client.channel("messaging", {
  members: [userId, opponentId],
  name: "Game Chat",
});
await channel.watch();
```

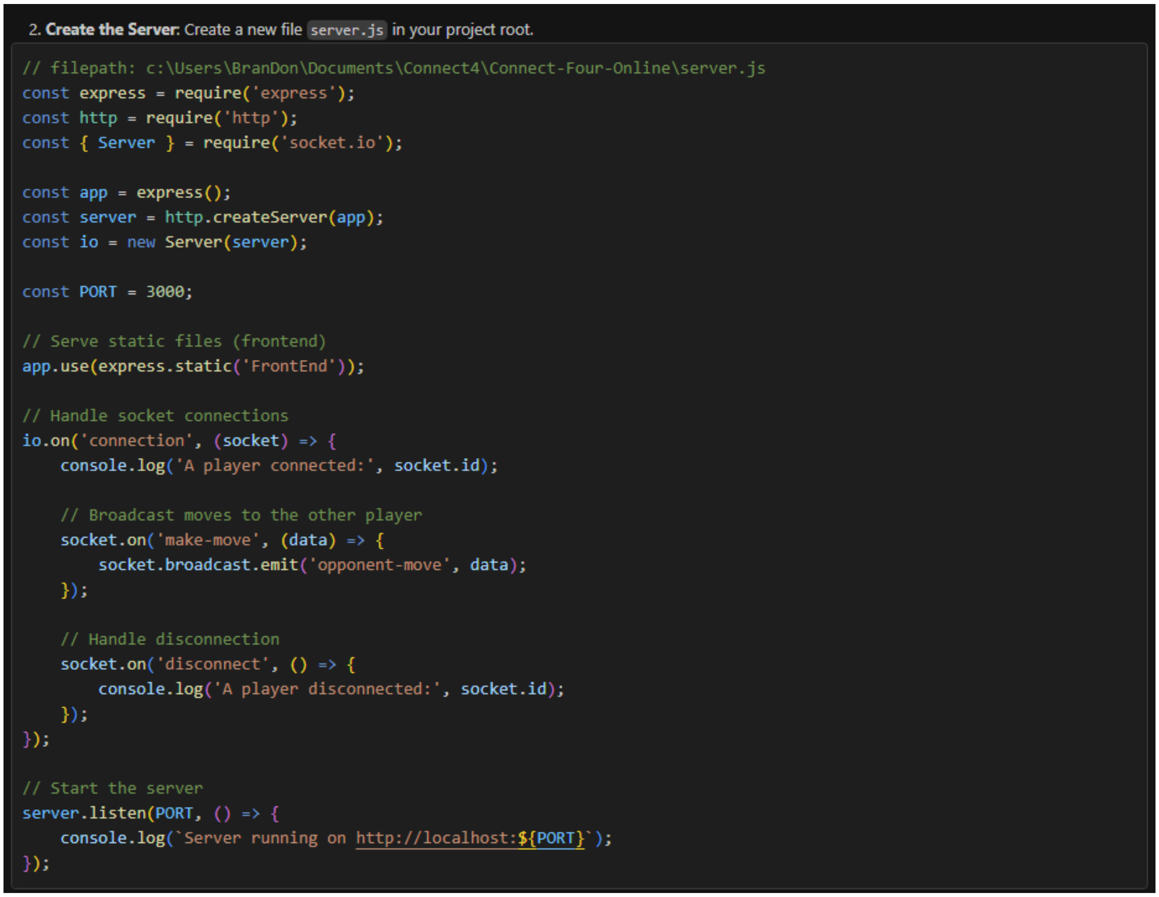
This ensures:

- The user is authenticated before trying to access or create a channel.
- The channel is either created or fetched if it already exists.
- You're subscribed to real-time updates (messages, typing, etc.).

🧠 Where Do `userId` and `opponentId` Come From?

- `userId` → from `socket.id` or whatever ID you used for StreamChat.
- `opponentId` → you'll need to track this in your game logic. For example, when both players are connected, you can emit their IDs to each other via Socket.IO.

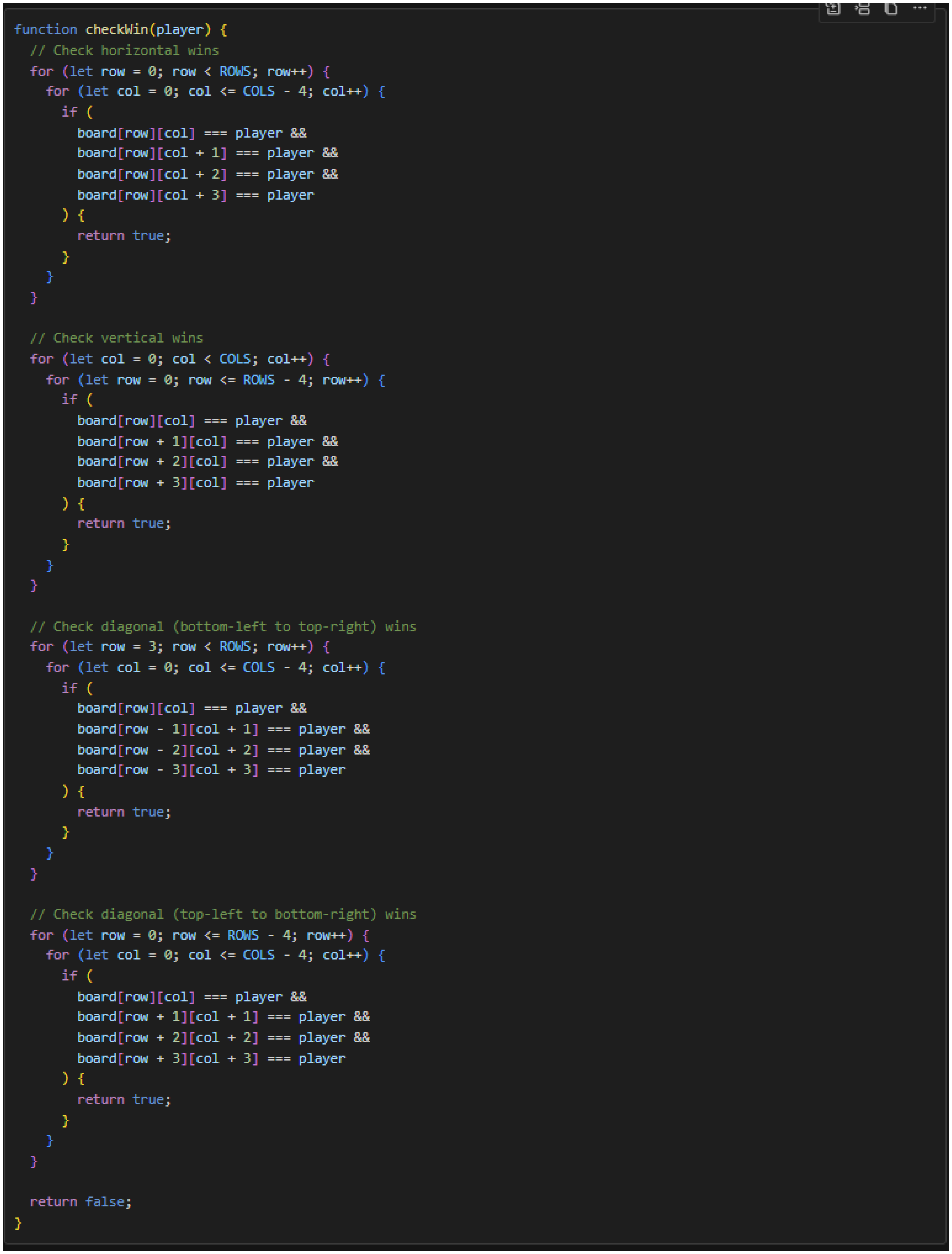
*2



2. **Create the Server**: Create a new file `server.js` in your project root.

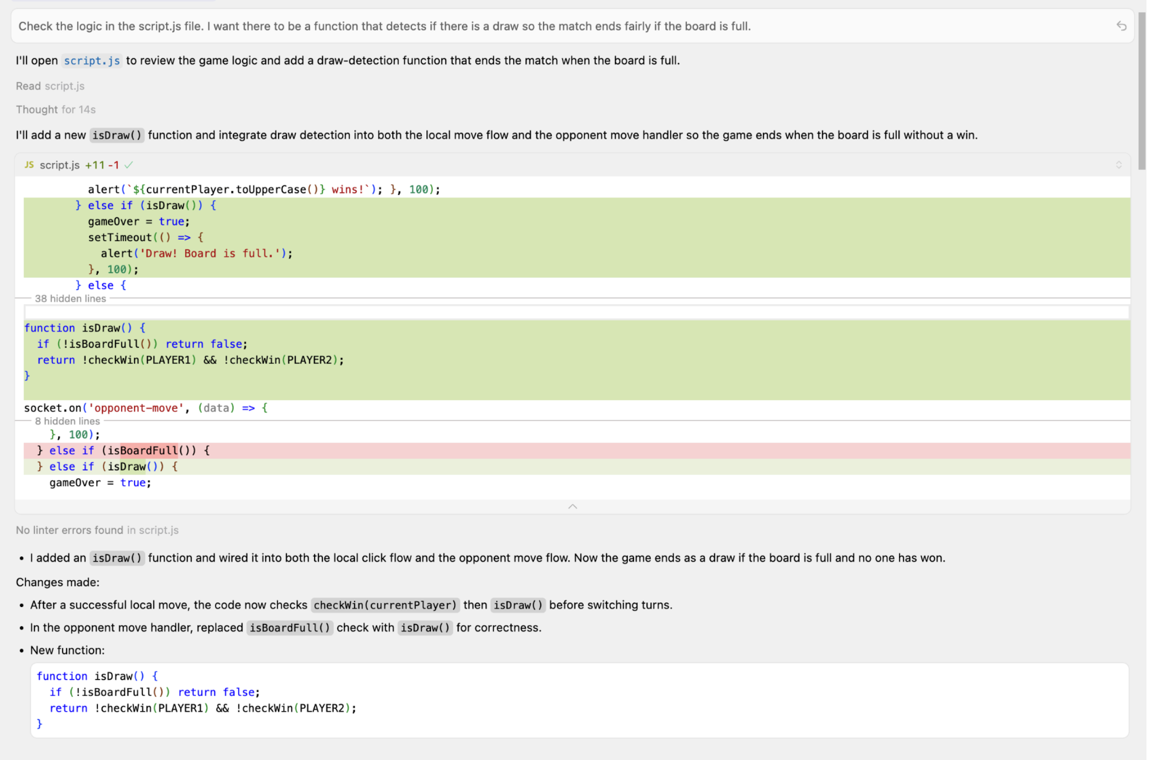
```
// filepath: c:\Users\BranDon\Documents\Connect4\Connect-Four-Online\server.js
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = new Server(server);

const PORT = 3000;

// Serve static files (frontend)
app.use(express.static('FrontEnd'));

// Handle socket connections
io.on('connection', (socket) => {
    console.log('A player connected:', socket.id);

    // Broadcast moves to the other player
    socket.on('make-move', (data) => {
        socket.broadcast.emit('opponent-move', data);
    });

    // Handle disconnection
    socket.on('disconnect', () => {
        console.log('A player disconnected:', socket.id);
    });
});

// Start the server
server.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```

*3



```
function checkWin(player) {
  // Check horizontal wins
  for (let row = 0; row < ROWS; row++) {
    for (let col = 0; col <= COLS - 4; col++) {
      if (
        board[row][col] === player &&
        board[row][col + 1] === player &&
        board[row][col + 2] === player &&
        board[row][col + 3] === player
      ) {
        return true;
      }
    }
  }

  // Check vertical wins
  for (let col = 0; col < COLS; col++) {
    for (let row = 0; row <= ROWS - 4; row++) {
      if (
        board[row][col] === player &&
        board[row + 1][col] === player &&
        board[row + 2][col] === player &&
        board[row + 3][col] === player
      ) {
        return true;
      }
    }
  }

  // Check diagonal (bottom-left to top-right) wins
  for (let row = 3; row < ROWS; row++) {
    for (let col = 0; col <= COLS - 4; col++) {
      if (
        board[row][col] === player &&
        board[row - 1][col + 1] === player &&
        board[row - 2][col + 2] === player &&
        board[row - 3][col + 3] === player
      ) {
        return true;
      }
    }
  }

  // Check diagonal (top-left to bottom-right) wins
  for (let row = 0; row <= ROWS - 4; row++) {
    for (let col = 0; col <= COLS - 4; col++) {
      if (
        board[row][col] === player &&
        board[row + 1][col + 1] === player &&
        board[row + 2][col + 2] === player &&
        board[row + 3][col + 3] === player
      ) {
        return true;
      }
    }
  }

  return false;
}
```

*4



Check the logic in the script.js file. I want there to be a function that detects if there is a draw so the match ends fairly if the board is full.

I'll open `script.js` to review the game logic and add a draw-detection function that ends the match when the board is full.

Read script.js

Thought for 14s

I'll add a new `isDraw()` function and integrate draw detection into both the local move flow and the opponent move handler so the game ends when the board is full without a win.

script.js +11 -1

```
          alert(`${currentPlayer.toUpperCase()} wins!`); }, 100);
        } else if (isDraw()) {
          gameOver = true;
          setTimeout(() => {
            alert('Draw! Board is full.');
          }, 100);
        } else {
38 hidden lines

function isDraw() {
  if (!isBoardFull()) return false;
  return !checkWin(PLAYER1) && !checkWin(PLAYER2);
}

socket.on('opponent-move', (data) => {
8 hidden lines
    }, 100);
  } else if (isBoardFull()) {
  } else if (isDraw()) {
    gameOver = true;
```

No linter errors found in script.js

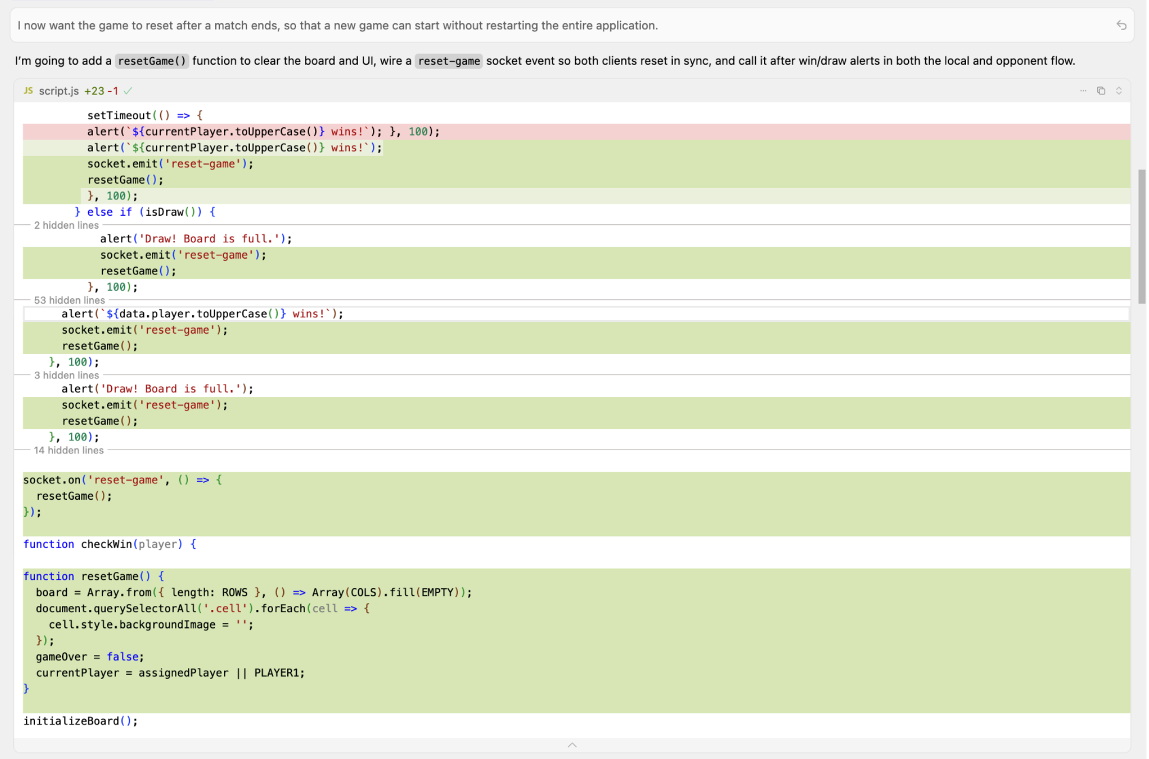
- I added an `isDraw()` function and wired it into both the local click flow and the opponent move flow. Now the game ends as a draw if the board is full and no one has won.

Changes made:

- After a successful local move, the code now checks `checkWin(currentPlayer)` then `isDraw()` before switching turns.
- In the opponent move handler, replaced `isBoardFull()` check with `isDraw()` for correctness.
- New function:

```
function isDraw() {
  if (!isBoardFull()) return false;
  return !checkWin(PLAYER1) && !checkWin(PLAYER2);
}
```

*5

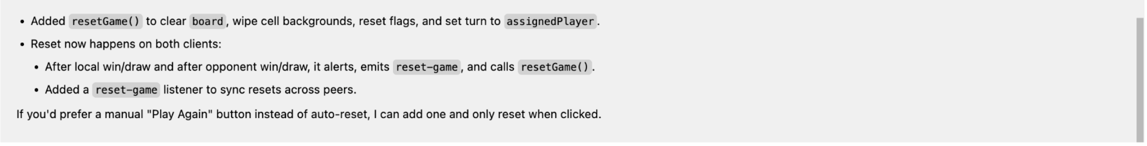


I now want the game to reset after a match ends, so that a new game can start without restarting the entire application.

I'm going to add a `resetGame()` function to clear the board and UI, wire a `reset-game` socket event so both clients reset in sync, and call it after win/draw alerts in both the local and opponent flow.

script.js +23 -1

```
          setTimeout(() => {
          alert(`${currentPlayer.toUpperCase()} wins!`); }, 100);
          alert(`${currentPlayer.toUpperCase()} wins!`);
          socket.emit('reset-game');
          resetGame();
          }, 100);
        } else if (isDraw()) {
2 hidden lines
            alert('Draw! Board is full.');
            socket.emit('reset-game');
            resetGame();
          }, 100);
53 hidden lines
      alert(`${data.player.toUpperCase()} wins!`);
      socket.emit('reset-game');
      resetGame();
    }, 100);
3 hidden lines
      alert('Draw! Board is full.');
      socket.emit('reset-game');
      resetGame();
    }, 100);
14 hidden lines

socket.on('reset-game', () => {
  resetGame();
});

function checkWin(player) {

function resetGame() {
  board = Array.from({ length: ROWS }, () => Array(COLS).fill(EMPTY));
  document.querySelectorAll('.cell').forEach(cell => {
    cell.style.backgroundImage = '';
  });
  gameOver = false;
  currentPlayer = assignedPlayer || PLAYER1;
}

initializeBoard();
```

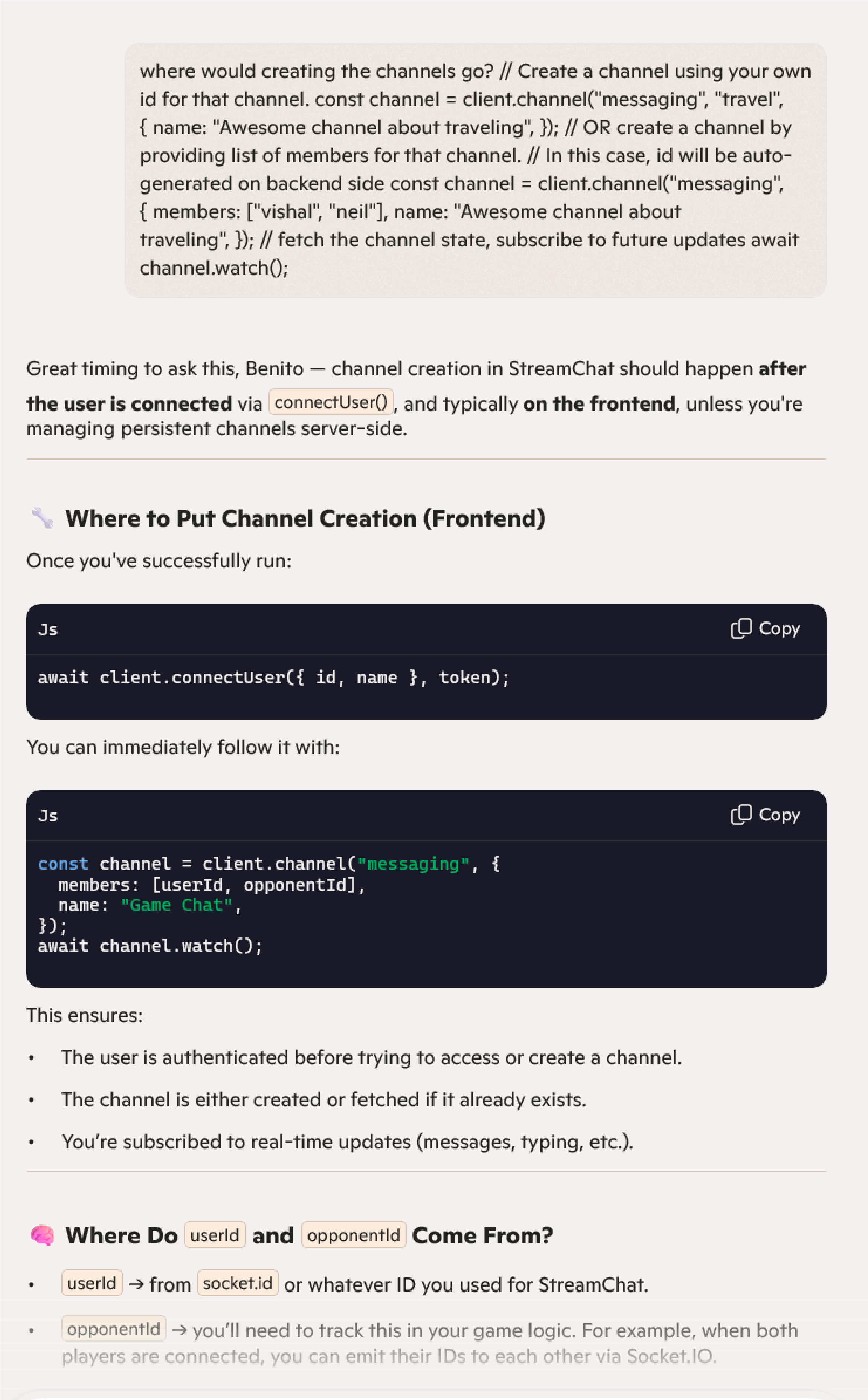


- Added `resetGame()` to clear `board`, wipe cell backgrounds, reset flags, and set turn to `assignedPlayer`.
- Reset now happens on both clients:
  - After local win/draw and after opponent win/draw, it alerts, emits `reset-game`, and calls `resetGame()`.
  - Added a `reset-game` listener to sync resets across peers.

If you'd prefer a manual "Play Again" button instead of auto-reset, I can add one and only reset when clicked.

4. Share Links to AI Sessions

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



where would creating the channels go? // Create a channel using your own id for that channel. const channel = client.channel("messaging", "travel", { name: "Awesome channel about traveling", }); // OR create a channel by providing list of members for that channel. // In this case, id will be auto-generated on backend side const channel = client.channel("messaging", { members: ["vishal", "neil"], name: "Awesome channel about traveling", }); // fetch the channel state, subscribe to future updates await channel.watch();

Great timing to ask this, Benito — channel creation in StreamChat should happen **after the user is connected** via `connectUser()`, and typically **on the frontend**, unless you're managing persistent channels server-side.

🔧 Where to Put Channel Creation (Frontend)

Once you've successfully run:

```js
await client.connectUser({ id, name }, token);
```

You can immediately follow it with:

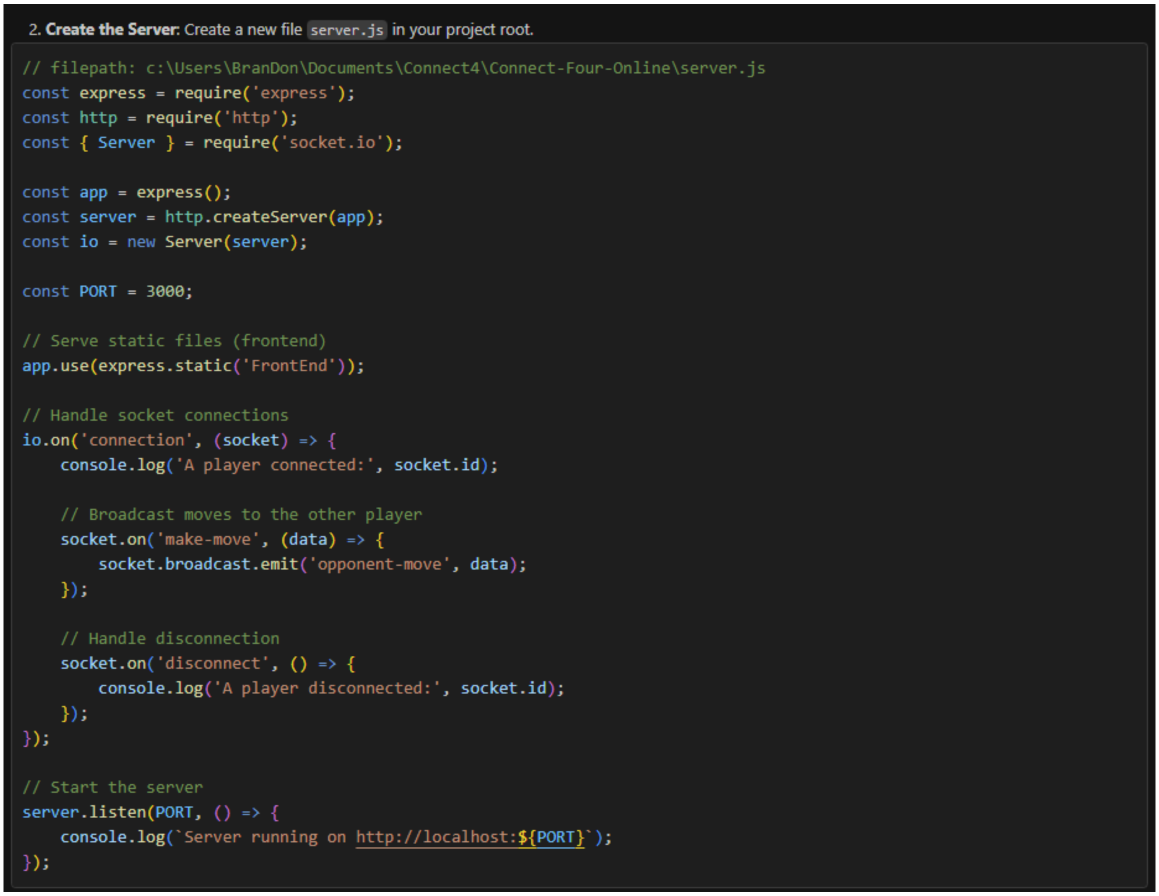
```js
const channel = client.channel("messaging", {
  members: [userId, opponentId],
  name: "Game Chat",
});
await channel.watch();
```

This ensures:

- The user is authenticated before trying to access or create a channel.
- The channel is either created or fetched if it already exists.
- You're subscribed to real-time updates (messages, typing, etc.).

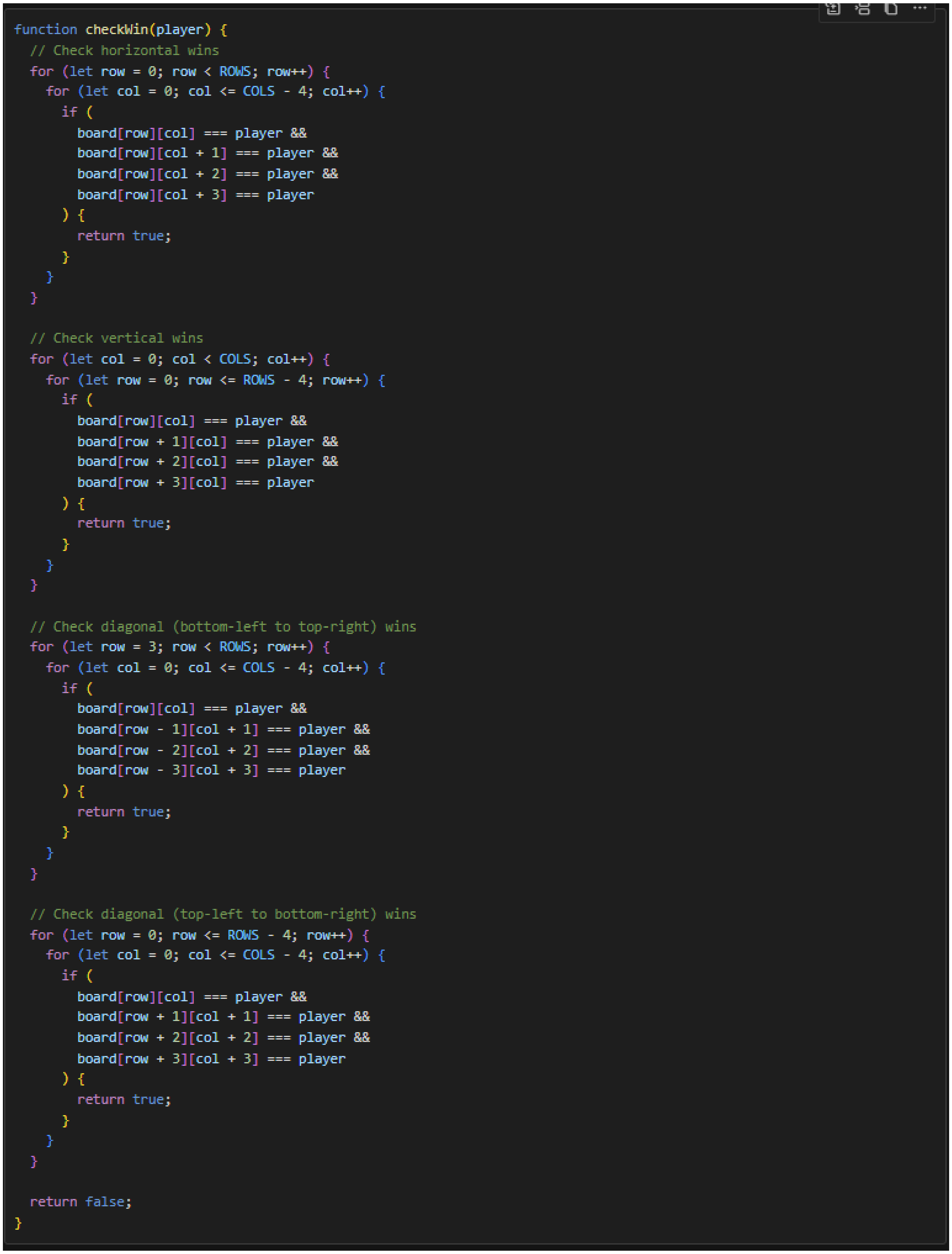
🧠 Where Do `userId` and `opponentId` Come From?

- `userId` → from `socket.id` or whatever ID you used for StreamChat.
- `opponentId` → you'll need to track this in your game logic. For example, when both players are connected, you can emit their IDs to each other via Socket.IO.

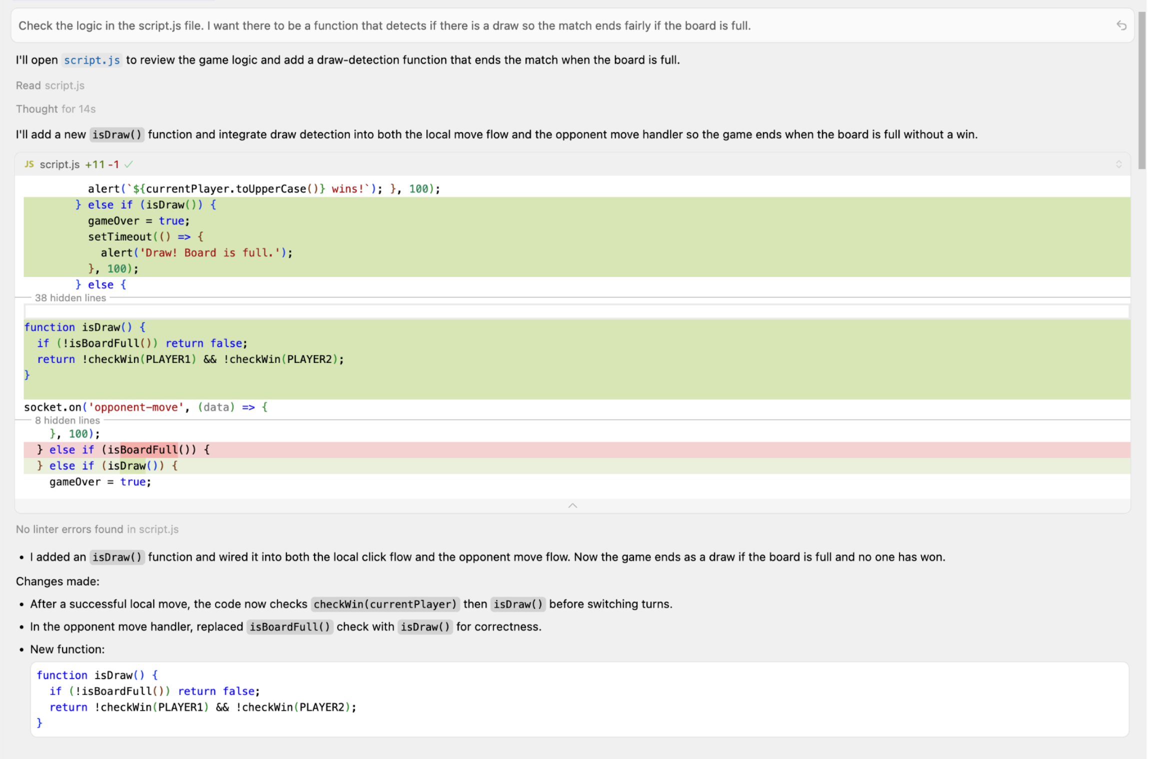


2. **Create the Server:** Create a new file `server.js` in your project root.

```
// filepath: c:\Users\BranDon\Documents\Connect4\Connect-Four-Online\server.js
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = new Server(server);

const PORT = 3000;

// Serve static files (frontend)
app.use(express.static('FrontEnd'));

// Handle socket connections
io.on('connection', (socket) => {
    console.log('A player connected:', socket.id);

    // Broadcast moves to the other player
    socket.on('make-move', (data) => {
        socket.broadcast.emit('opponent-move', data);
    });

    // Handle disconnection
    socket.on('disconnect', () => {
        console.log('A player disconnected:', socket.id);
    });
});

// Start the server
server.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```



```
function checkWin(player) {
  // Check horizontal wins
  for (let row = 0; row < ROWS; row++) {
    for (let col = 0; col <= COLS - 4; col++) {
      if (
        board[row][col] === player &&
        board[row][col + 1] === player &&
        board[row][col + 2] === player &&
        board[row][col + 3] === player
      ) {
        return true;
      }
    }
  }

  // Check vertical wins
  for (let col = 0; col < COLS; col++) {
    for (let row = 0; row <= ROWS - 4; row++) {
      if (
        board[row][col] === player &&
        board[row + 1][col] === player &&
        board[row + 2][col] === player &&
        board[row + 3][col] === player
      ) {
        return true;
      }
    }
  }

  // Check diagonal (bottom-left to top-right) wins
  for (let row = 3; row < ROWS; row++) {
    for (let col = 0; col <= COLS - 4; col++) {
      if (
        board[row][col] === player &&
        board[row - 1][col + 1] === player &&
        board[row - 2][col + 2] === player &&
        board[row - 3][col + 3] === player
      ) {
        return true;
      }
    }
  }

  // Check diagonal (top-left to bottom-right) wins
  for (let row = 0; row <= ROWS - 4; row++) {
    for (let col = 0; col <= COLS - 4; col++) {
      if (
        board[row][col] === player &&
        board[row + 1][col + 1] === player &&
        board[row + 2][col + 2] === player &&
        board[row + 3][col + 3] === player
      ) {
        return true;
      }
    }
  }

  return false;
}
```



Check the logic in the script.js file. I want there to be a function that detects if there is a draw so the match ends fairly if the board is full.

I'll open `script.js` to review the game logic and add a draw-detection function that ends the match when the board is full.

Read script.js

Thought for 14s

I'll add a new `isDraw()` function and integrate draw detection into both the local move flow and the opponent move handler so the game ends when the board is full without a win.

JS script.js +11 -1 ✓

```
          alert(`${currentPlayer.toUpperCase()} wins!`); }, 100);
        } else if (isDraw()) {
          gameOver = true;
          setTimeout(() => {
            alert('Draw! Board is full.');
          }, 100);
        } else {
38 hidden lines

function isDraw() {
  if (!isBoardFull()) return false;
  return !checkWin(PLAYER1) && !checkWin(PLAYER2);
}

socket.on('opponent-move', (data) => {
8 hidden lines
    }, 100);
  } else if (isBoardFull()) {
  } else if (isDraw()) {
    gameOver = true;
```

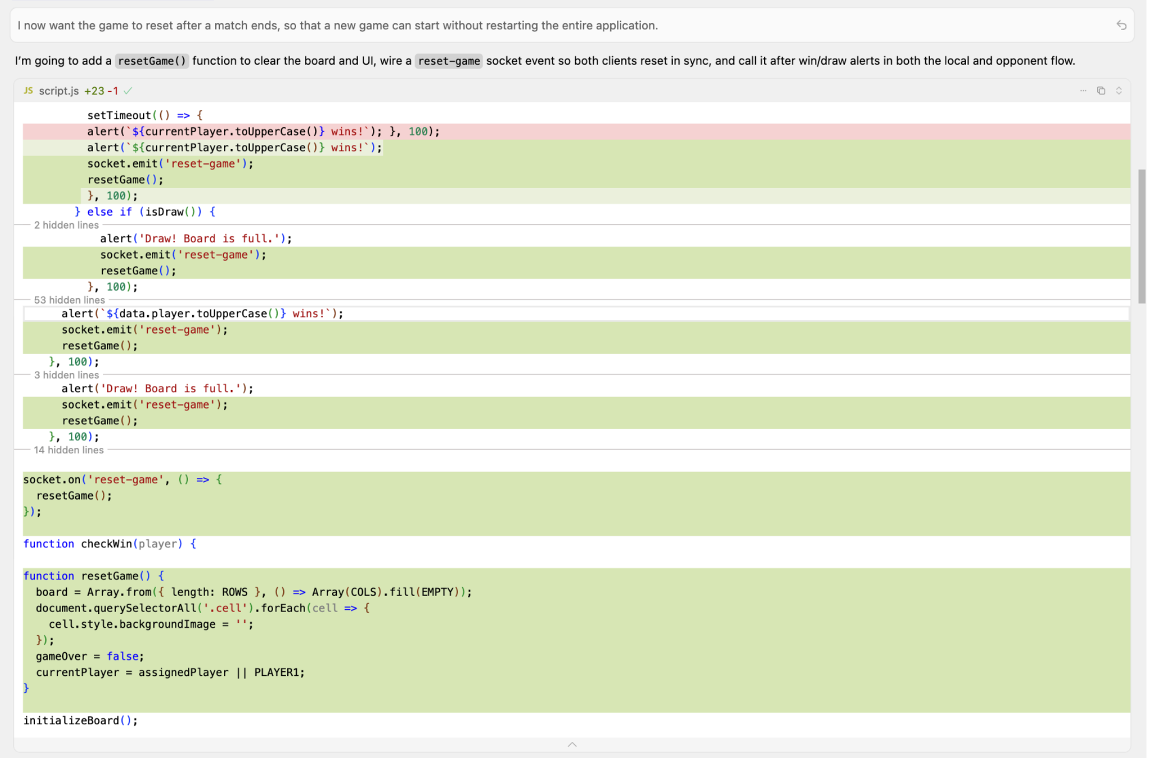
No linter errors found in script.js

- I added an `isDraw()` function and wired it into both the local click flow and the opponent move flow. Now the game ends as a draw if the board is full and no one has won.

Changes made:

- After a successful local move, the code now checks `checkWin(currentPlayer)` then `isDraw()` before switching turns.
- In the opponent move handler, replaced `isBoardFull()` check with `isDraw()` for correctness.
- New function:

```
function isDraw() {
  if (!isBoardFull()) return false;
  return !checkWin(PLAYER1) && !checkWin(PLAYER2);
}
```

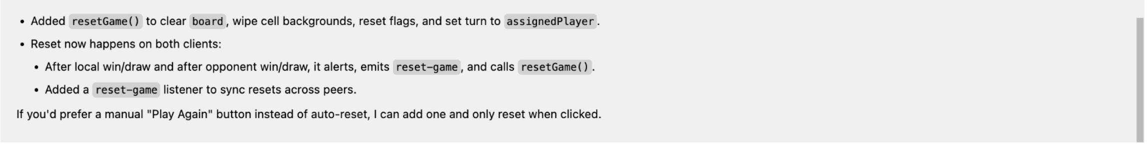


I now want the game to reset after a match ends, so that a new game can start without restarting the entire application.

I'm going to add a `resetGame()` function to clear the board and UI, wire a `reset-game` socket event so both clients reset in sync, and call it after win/draw alerts in both the local and opponent flow.

JS script.js +23 -1 ✓

```
          setTimeout(() => {
          alert(`${currentPlayer.toUpperCase()} wins!`); }, 100);
          alert(`${currentPlayer.toUpperCase()} wins!`);
          socket.emit('reset-game');
          resetGame();
          }, 100);
        } else if (isDraw()) {
2 hidden lines
            alert('Draw! Board is full.');
            socket.emit('reset-game');
            resetGame();
          }, 100);
53 hidden lines
      alert(`${data.player.toUpperCase()} wins!`);
      socket.emit('reset-game');
      resetGame();
    }, 100);
3 hidden lines
      alert('Draw! Board is full.');
      socket.emit('reset-game');
      resetGame();
    }, 100);
14 hidden lines

socket.on('reset-game', () => {
  resetGame();
});

function checkWin(player) {

function resetGame() {
  board = Array.from({ length: ROWS }, () => Array(COLS).fill(EMPTY));
  document.querySelectorAll('.cell').forEach(cell => {
    cell.style.backgroundImage = '';
  });
  gameOver = false;
  currentPlayer = assignedPlayer || PLAYER1;
}

initializeBoard();
```



- Added `resetGame()` to clear `board`, wipe cell backgrounds, reset flags, and set turn to `assignedPlayer`.
- Reset now happens on both clients:
  - After local win/draw and after opponent win/draw, it alerts, emits `reset-game`, and calls `resetGame()`.
  - Added a `reset-game` listener to sync resets across peers.

If you'd prefer a manual "Play Again" button instead of auto-reset, I can add one and only reset when clicked.

- Link 2: ______________________
- Link 3: ______________________

(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 4 user stories, We kept 2, rephrased 2, and rejected 0.

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? Acknowledged use of AI

7. Contribution to Project

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

It massively condensed the time it would take to understand the StreamChat, render, and servers.

8. Sign-Off

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

Team Members (type names):
- Benito Reyes
- Brandon Brown
- Mekhi Green
- Ramsey Burls

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