Jacob Barner
CSC343
4/28/2022
Final Project - Report

Requirements:

The goal of this project was to enable two opposing players to play through multiple games of Connect-4 and keep track of both players' scores as they continue to play.
Since this program will be locally hosted, it was made in a way that it could be played by any two people, anywhere, at any time, so long as they had a functioning computer and access to the Connect-4 program.

Since the game is being played on a static webpage, rather than the controllable three dimensional model being played in the real world, there were some additional requirements I needed to keep track of when designing the game for all users. The first requirement was visibility. Let's say for example that two players, John and Mary decide they both wanted to play each other in the game. John, however, is colorblind and would have trouble viewing the game the same way as Mary, if the original game colors were kept. While both players could have the same level of skill and perceptiveness ordinarily, Mary would have a competitive advantage in this game because she would have an easier time distinguishing the pieces from one another. In order to make sure both players could still have a fair game, a very diverse color palette was needed that ensured both players could easily see the game and distinguish all components.

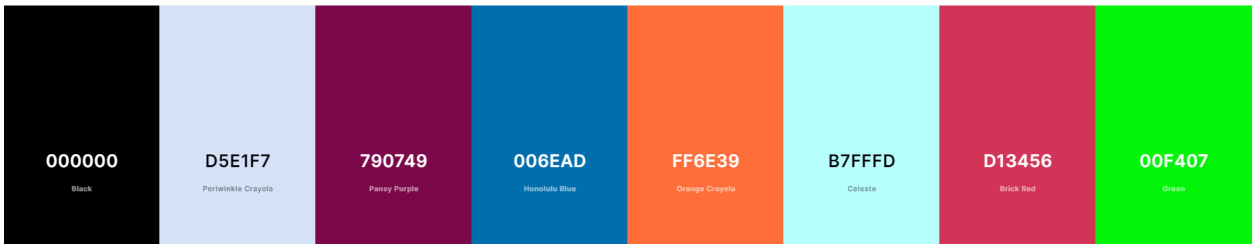
The second essential requirement I needed to keep in mind was useability. When playing the physical game, it is much easier to see where your piece will go, simply by moving your hand across the top of the game. On a screen, if there is no indicator, it can be difficult to know just where your piece will go once you click, especially if you click in between columns. So a visual indicator that displays which column your piece will be placed in is essential to help players know exactly where their piece will go.

The final requirement that I needed to meet was screen clutter, or in this case, lack thereof. Connect-4 is a game that requires both players to be able to focus on the game board and not get distracted easily by everything else that's flooding the screen. In order to meet this requirement, there should be very little on the screen except for what is essential, for example, the board, the score, and the reset/play again buttons.

The program also needs to accept both players names before the game will begin, however this is a fundamental design decision rather than a necessity. Similarly, the game will need to pause and display a game over screen whenever any of the win conditions are met (diagonal win, horizontal win, vertical win, filled board). This prevents players from continuing to play on accident when a game is won.

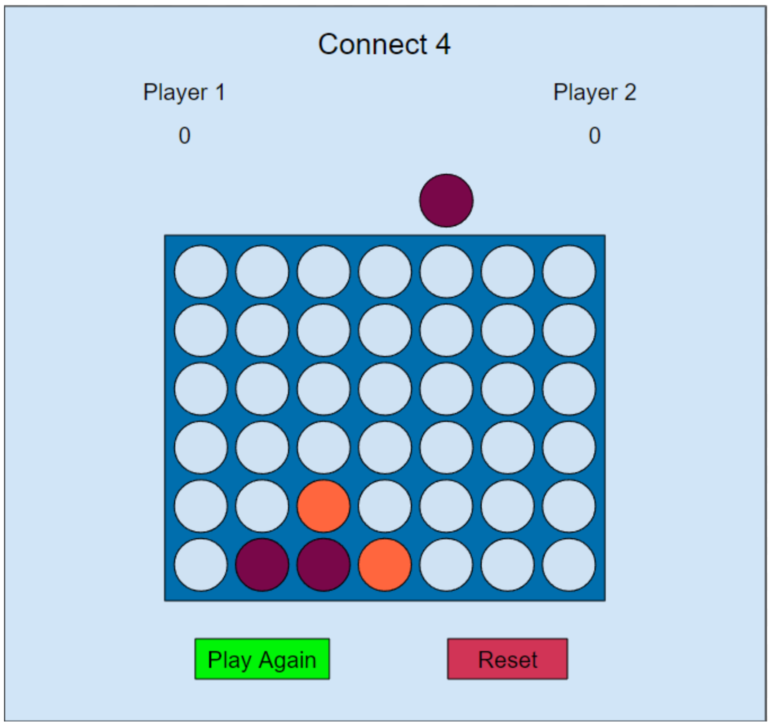
Design:

In order to solve the color blind palette issue, I decided on the final color palette shown below.

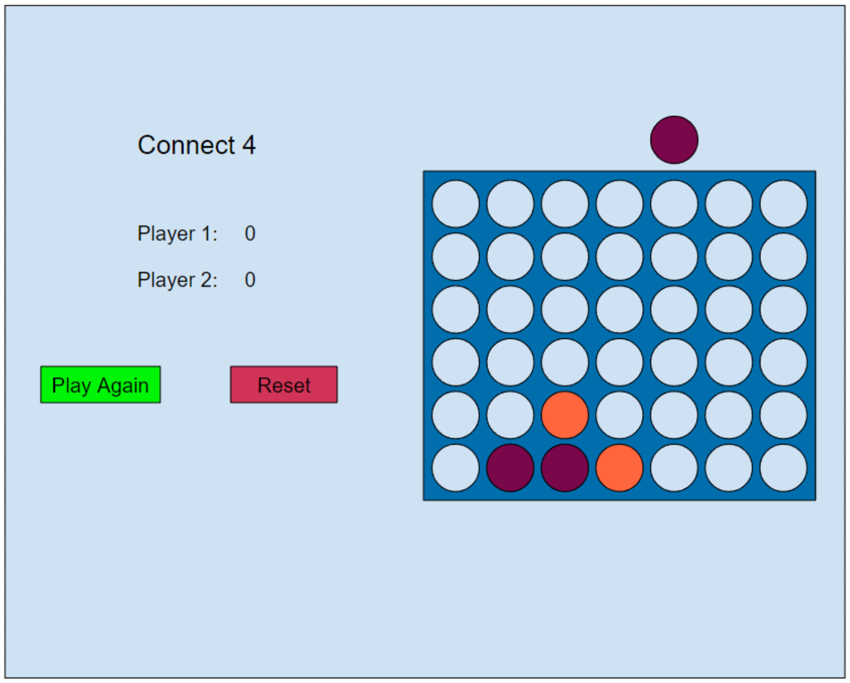


The two lightest colors were used as the background. The blue, purple and orange became the board and game pieces. The greed and pink were used for play again and reset respectively. And all text except for player names, which are their respective game piece colors, were black.

The primary design plan is to build the board so that it is accessible to both players, and have as little distraction on the screen as possible. For this reason, the designs only contain the title, players names, scores, and the replay buttons. The name entry is not displayed, however this would take place in the same location before the first piece is played. Similarly, the game over screen is not shown, however it only appears after each game and displays who won and asks if the users want a rematch.



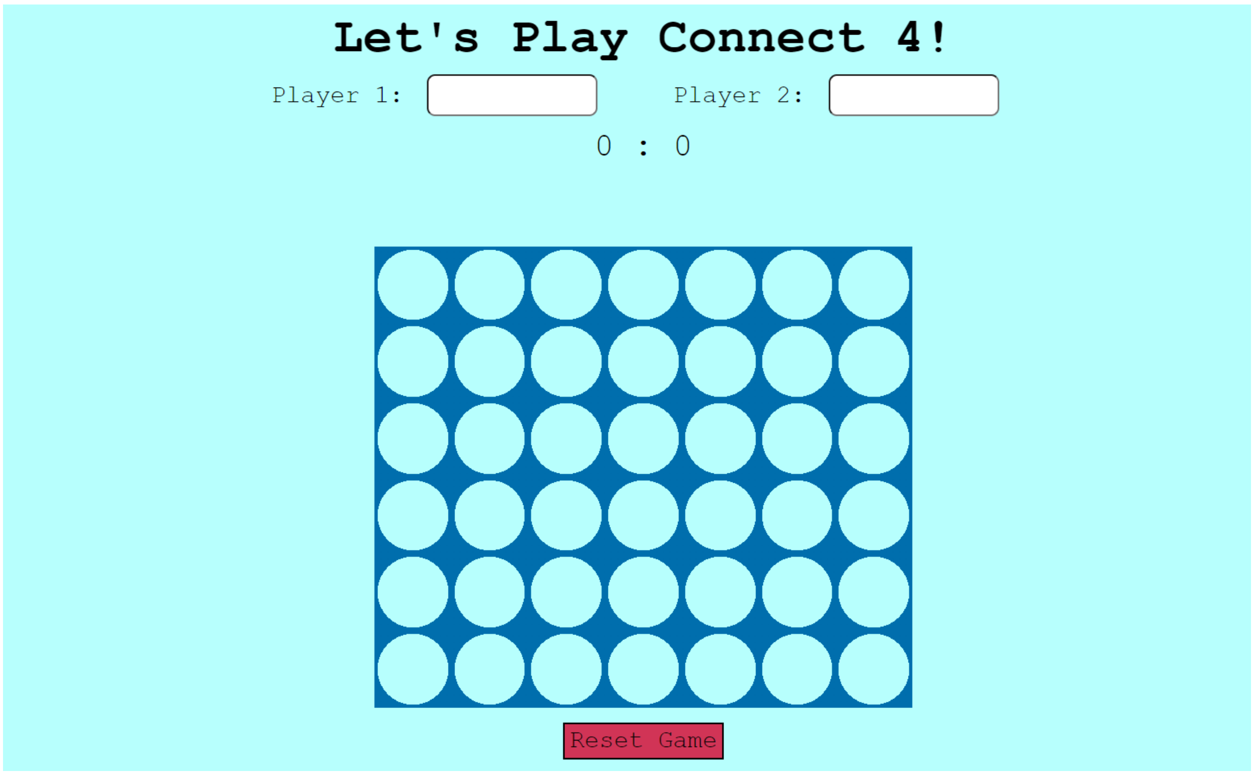
Design 1 has the board centered, with the players scores on either side of the board. The restart buttons are then displayed below the board. This design is more visually appealing, and keeps everything visible without over burdening the user. This design became the eventual layout of the real implementation for the game.



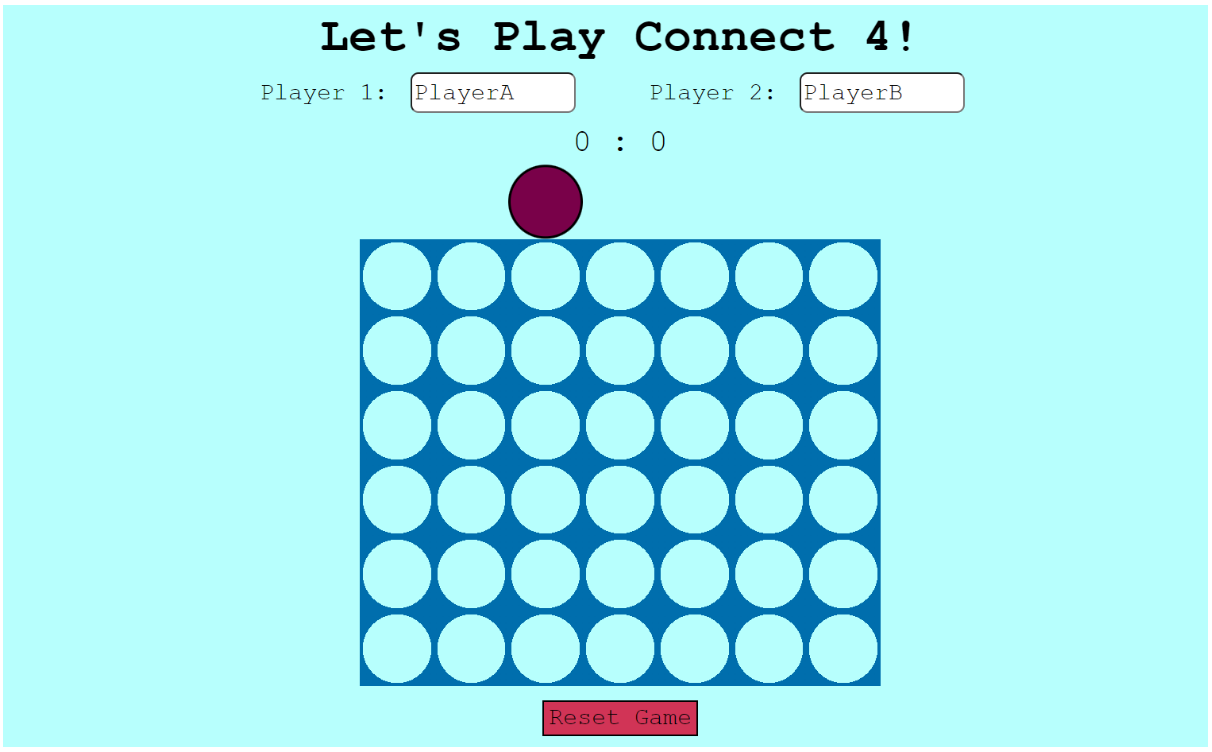
Design 2, divides the screen in half, the left is the players, scores, and reset buttons. While the right contains only the game. This design is less appealing because it is not even on either side. The board takes up significantly more space than the score board. There is too much unnecessary empty space. I also was personally not as big of a fan of this design because it forces both players to look across the screen to see the opposite side if both players are sitting next to each other. Whereas, if all of the components are centered, you can look at the same central area and see everything at once.

Implementation:

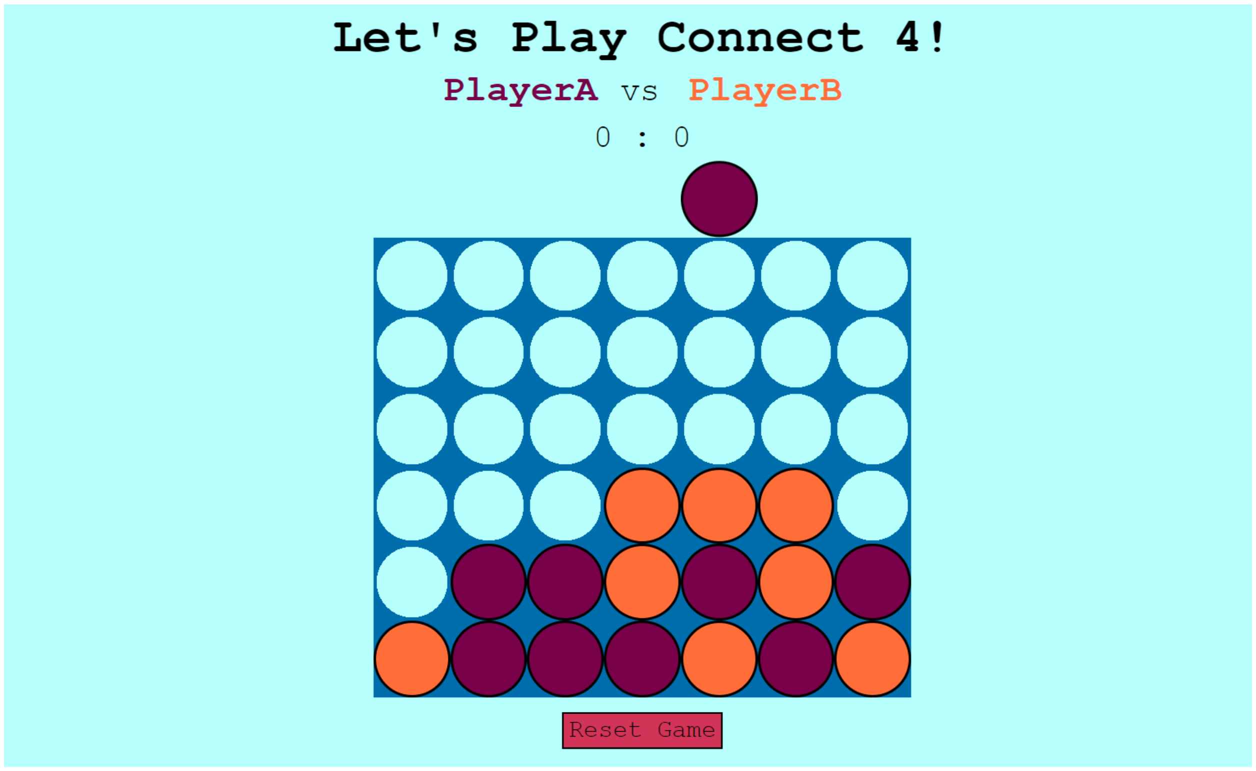
Below I have included the implementation for the program design. At the beginning of the game, everything is already displayed on screen, however the players are required to enter their names before the first game piece is allowed to be played.



When trying to select the location for your next piece to go, a visual indicator appears above the game board to verify where the piece will fall once you click to move.

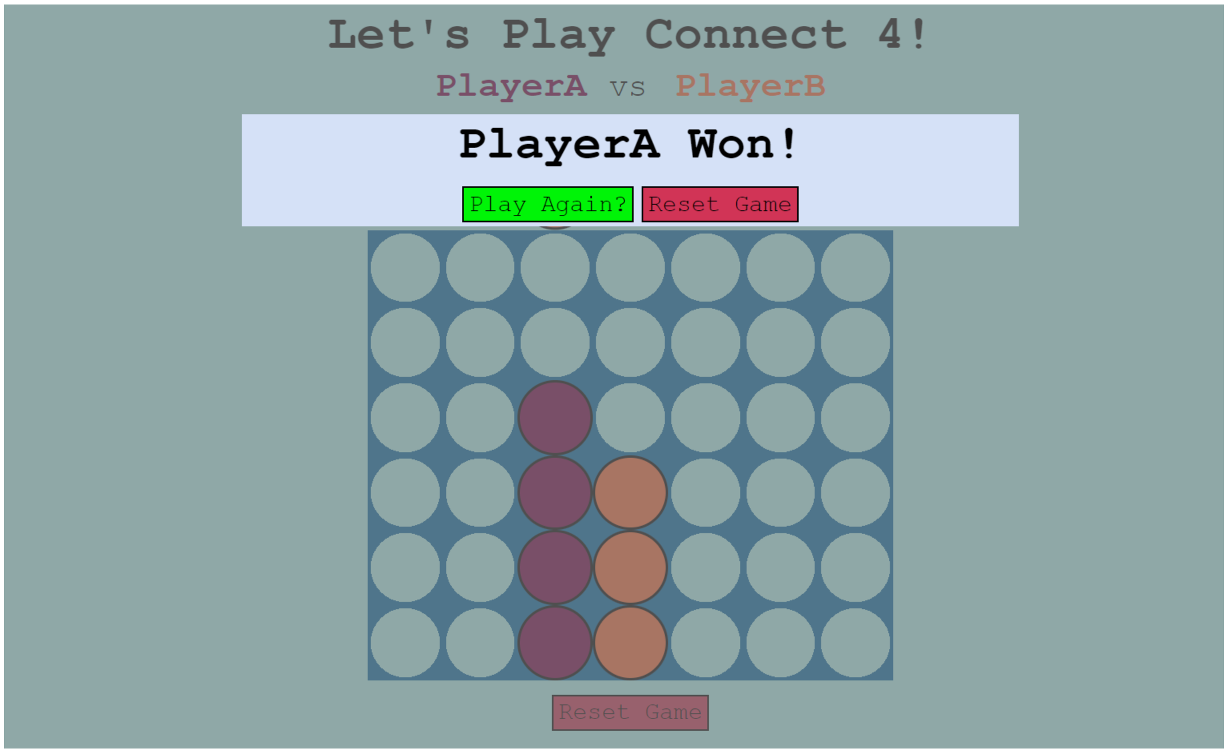


Following the first piece, players' names are highlighted with their respective piece colors (purple or orange). They then take alternating turns playing pieces until one of the win conditions are met.

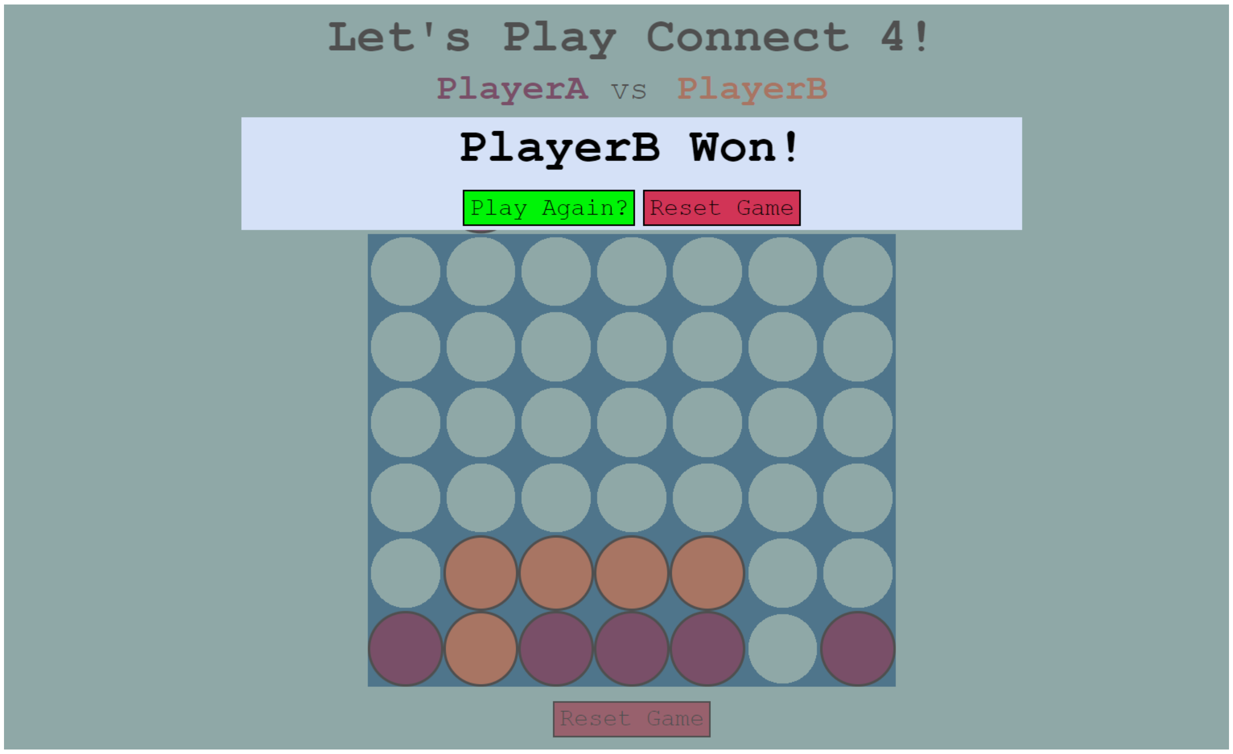


Once one of the players completes one of the win conditions, the screen is grayed out and the game over screen is presented, prompting the players to play again, or reset the game entirely, which will ask for new player names.

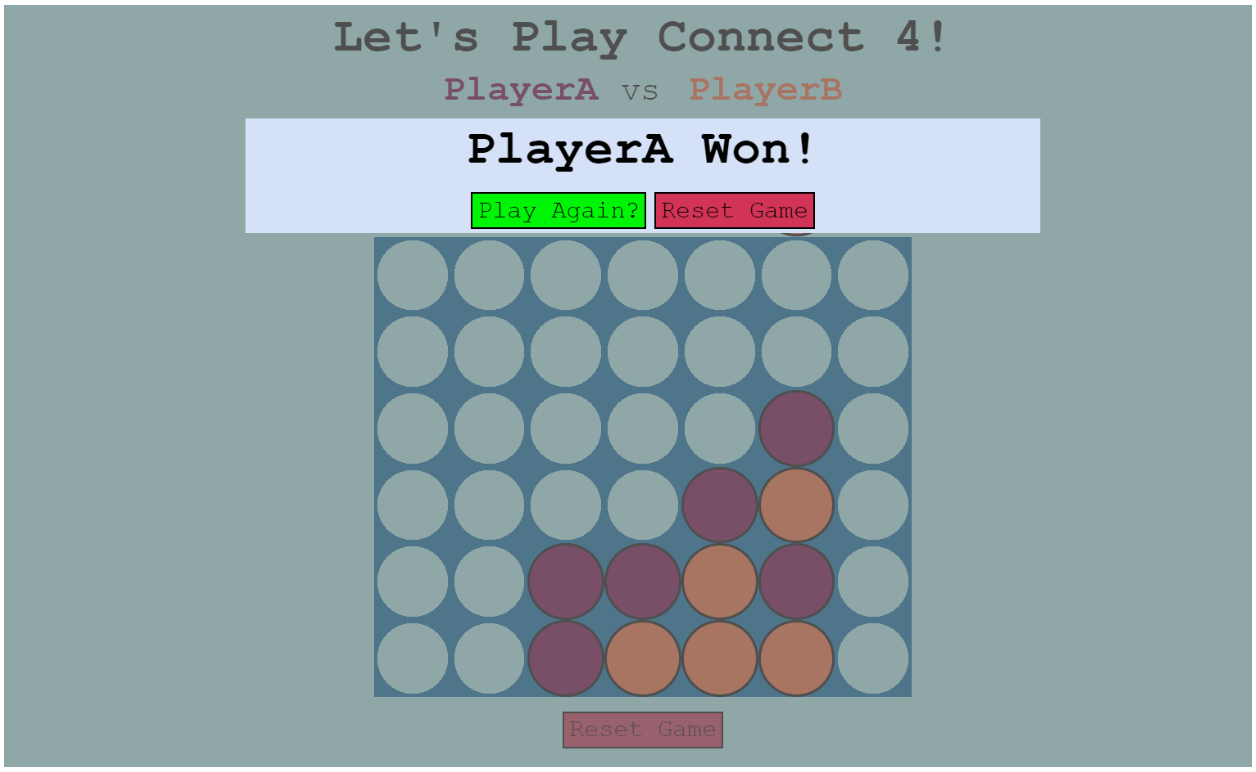
The first win condition is a vertical win, where one player has 4 pieces in an unbroken column.



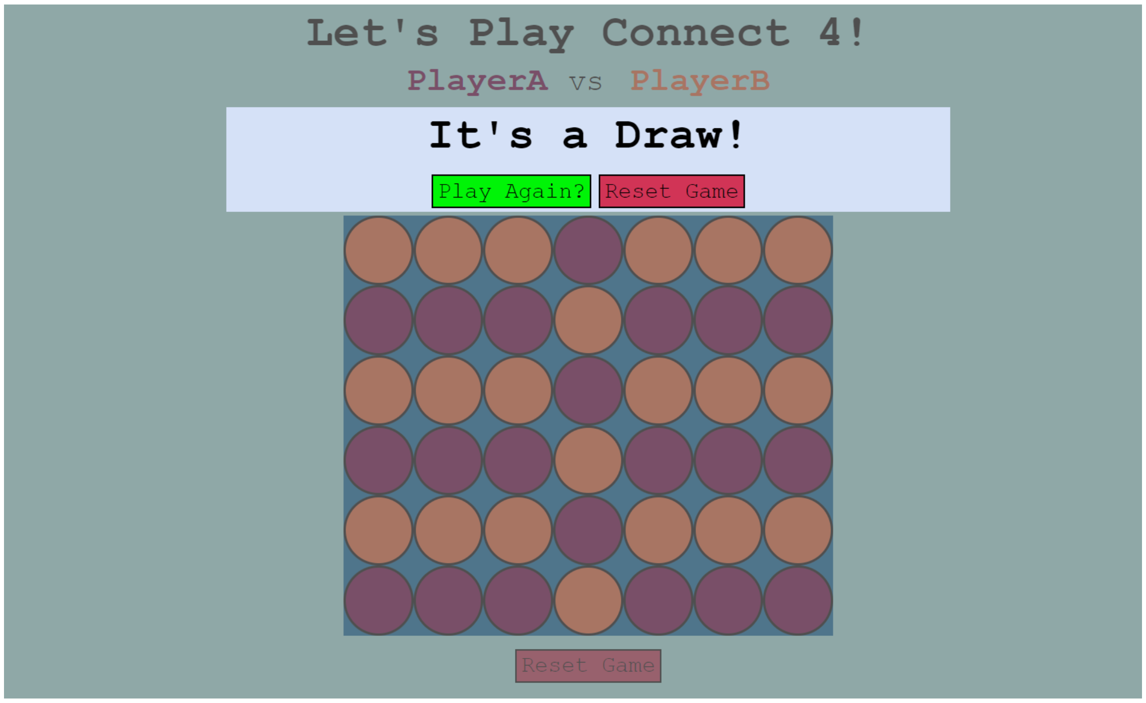
The second win condition is a horizontal win in which one of the players has 4 of their pieces in an unbroken row.



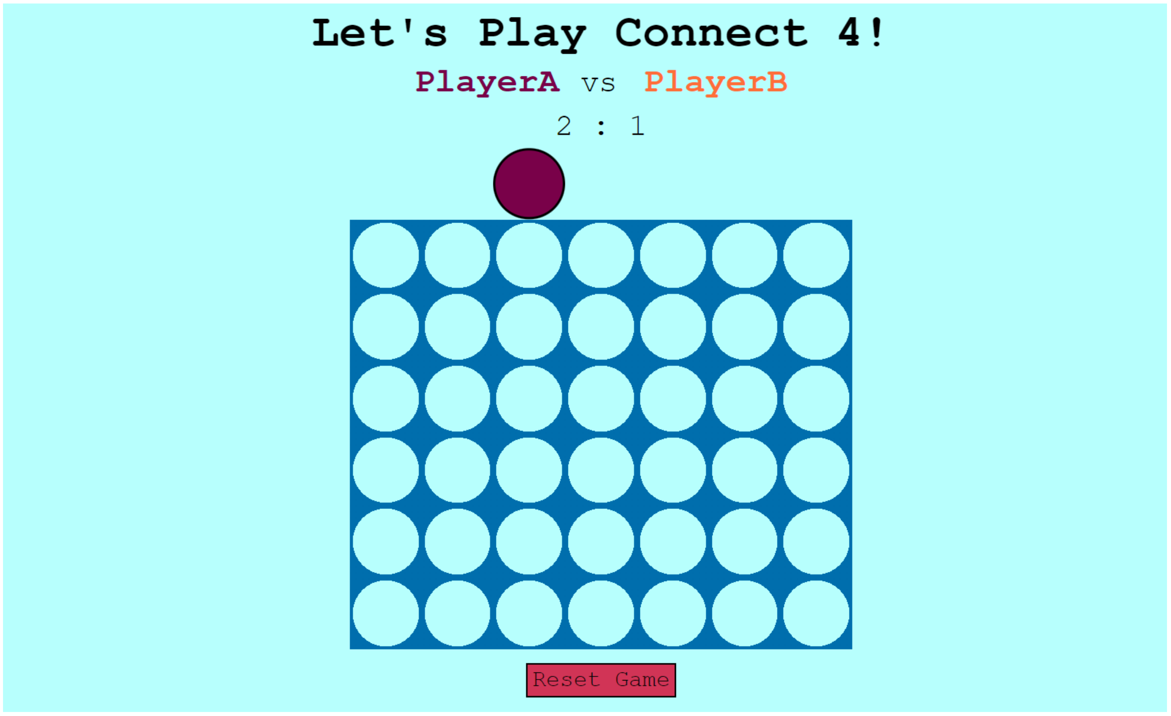
The third win condition is a diagonal win, which is only met if 4 of a player's pieces connect in a diagonal line across the board.



Lastly, there is a draw “win” condition which still ends the game when the board is completely filled and there has not been a winner.



If the players choose to play again, the game will reset, however their names will carry over, and the score will be updated adding a point to the winner's score. No point will be added if the previous game resulted in a draw. If the users instead reset the game at any point, the game will revert to the initial state, where the new players must enter their names, and the score is reset to 0.



Testing and Feedback:

In order to test the functionality of the program, in terms of the usability, visibility, and overall appeal of the game, I asked several of my peers, family members, and friends to try out the game and give me feedback about their experience. I also watched while they played to see any common mistakes or features that confused them while they played. Below is a table to the positives and negatives of my program based on the user feedback.

Positive Experiences	Negative Experiences
● Overall, the color palette was enjoyed by all players because it makes everything clearly distinguishable. ● The game is easy to pick up and play when users already have knowledge on the rules of the game. ● Many players also particularly liked that the game is covered when a winning move is played so that the other player does not try and continue to play. ● Similarly, being able to still see the game board on the win screen was appreciated so that both players would still be able to see how they won or lost.	● Many players tried to click above the board, where the indicator appears to place their piece. However this is not how the game functions, you must click somewhere on the active board. This will need further implementation to not display the preview piece above the board when the cursor is hovering above the game. ● There were also issues with players regretting their piece placement and wanting to revert their move or reset the game while maintaining the score, while in the middle of the game. The implementation was designed specifically so that users could not reset or revert the game before a win condition, so this will not be further implemented. ● Two of the players had never played connect-4 before so the rules had to be explained to them. Before the beginning of a new matchup, it could be useful to inform the users of the rules of the game before play begins.

Overall, very few visual and or functional changes would need to be changed. A rules or "How to play" screen should be further implemented, and the preview pieces above the board should be removed when the player is not hovering the cursor on the board.

Appendix

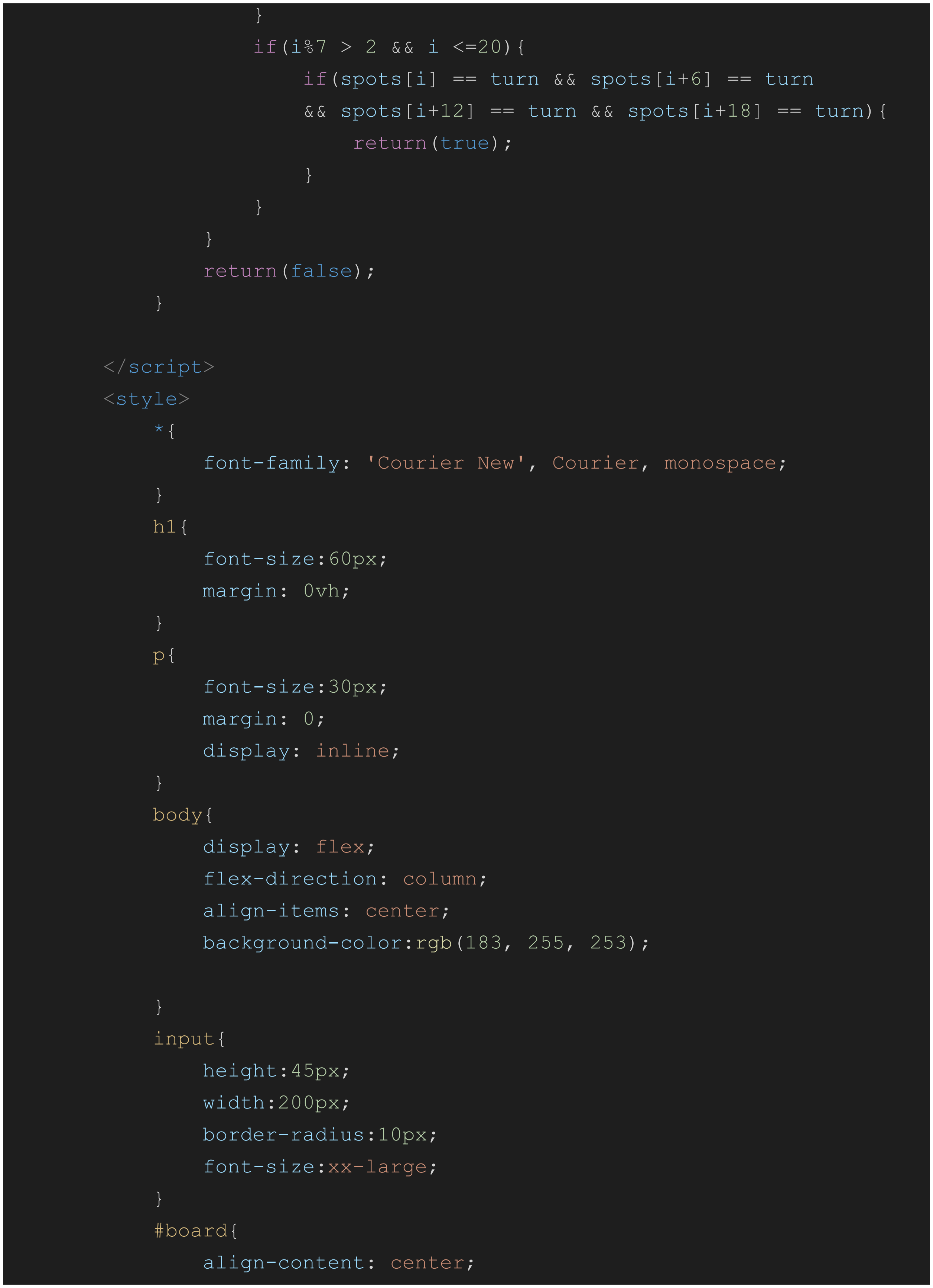
Game Code

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Connect 4!</title>
    </head>
    <body>
        <h1>Let's Play Connect 4!</h1>
        <div id="players">
            <div id="player1">
                Player 1:
            </div>
            <input type"text" id="p1name">
            <b><div id="p1Appear">
            </div></b>
            <div id="player2">&nbsp;&nbsp;
                Player 2:
            </div>
            <input type"text" id="p2name">
            <div id="vs">
            </div>
            <b><div id="p2Appear">
            </div></b>
        </div>
        <div id = "score"></div>
        <div id="board">
        </div>
        <div id="winDisplay">
            <div id="innerText">
                <h1 id="winner"></h1>
                <br>
                <div id="againButtons">
                    <button onclick="playAgain()" id="playAgain">Play
Again?</button>
                    <button onclick="reset()" id="reset2">Reset
Game</button>
                </div>
            </div>
        </div>
        <br>
        <button onclick="reset()" id="reset">Reset Game</button>
        <script>
            p1score = 0;
            p2score = 0;

            turn = 1;
            spots = [0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0];


            board = document.querySelector("#board");
            let currentHover = -1;
            let dropping = false;

            for(let i = 0; i < 42; i++){
                let space = document.createElement("div");
                space.className = "space";
                board.appendChild(space);

                space.onmouseenter = () => {
                    onMouseEntered(i%7);
                }
                space.onclick = () => {
                    if(!dropping){
                        onMouseClicked(i%7);
                    }

                }
            }
            document.getElementById("score").innerHTML = p1score + " : " +
p2score;
            function reset(){
                setTimeout(function(){
                    window.location.reload(1);
                }, 0);

            }
            function onMouseEntered(column){
                currentHover = column;
                if(!dropping){
                    updateTop();
                }
            }
            function playAgain(){
                document.getElementById("winDisplay").style.display =
"none";
                turn = 1;
                spots = [0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0];
                board.innerHTML = '';
                for(let i = 0; i < 42; i++){
                    let space = document.createElement("div");
                    space.className = "space";
                    board.appendChild(space);

                    space.onmouseenter = () => {
                        onMouseEntered(i%7);
                    }
                    space.onclick = () => {
                        if(!dropping){
                            onMouseClicked(i%7);
                        }

                    }
                }
            }

            function onMouseClicked(column){
                if(!document.getElementById("p1name").value == "" &&
!document.getElementById("p2name").value == ""){
```

```
                    document.getElementById("p1name").style.display =
"none";
                    document.getElementById("p2name").style.display =
"none";
                    document.getElementById("player1").style.display =
"none";
                    document.getElementById("player2").style.display =
"none";
                    document.getElementById("p1Appear").innerHTML =
document.getElementById("p1name").value;
                    document.getElementById("p1Appear").style.display =
"inline";
                    document.getElementById("vs").innerHTML = " vs ";
                    document.getElementById("vs").style.display =
"inline";
                    document.getElementById("p2Appear").innerHTML =
document.getElementById("p2name").value
                    document.getElementById("p2Appear").style.display =
"inline";
                    let nextUp = spots.filter( (_, index) => index%7 ==
column).lastIndexOf(0);
                    let space = board.children[column + (nextUp*7)];
                    let top = document.createElement("div");
                    top.className = "top";
                    top.dataset.added = true;
                    top.dataset.turn = turn;
                    space.appendChild(top);
                    let hoverPiece =
document.querySelector("[data-added='false']");
                    let hoverY = hoverPiece.getBoundingClientRect().y;
                    let bottomY = top.getBoundingClientRect().y;
                    let yDrop = hoverY - bottomY;
                    let drop =
top.animate([{transform:`translateY(${yDrop}px`, offset:
0}],{duration:200,easing:"linear",iteration: 1});
                    dropping = true;
                    removeHover();
                    drop.addEventListener('finish', () => checkForWin());
                    spots[column + (nextUp*7)] = turn;
                }
```

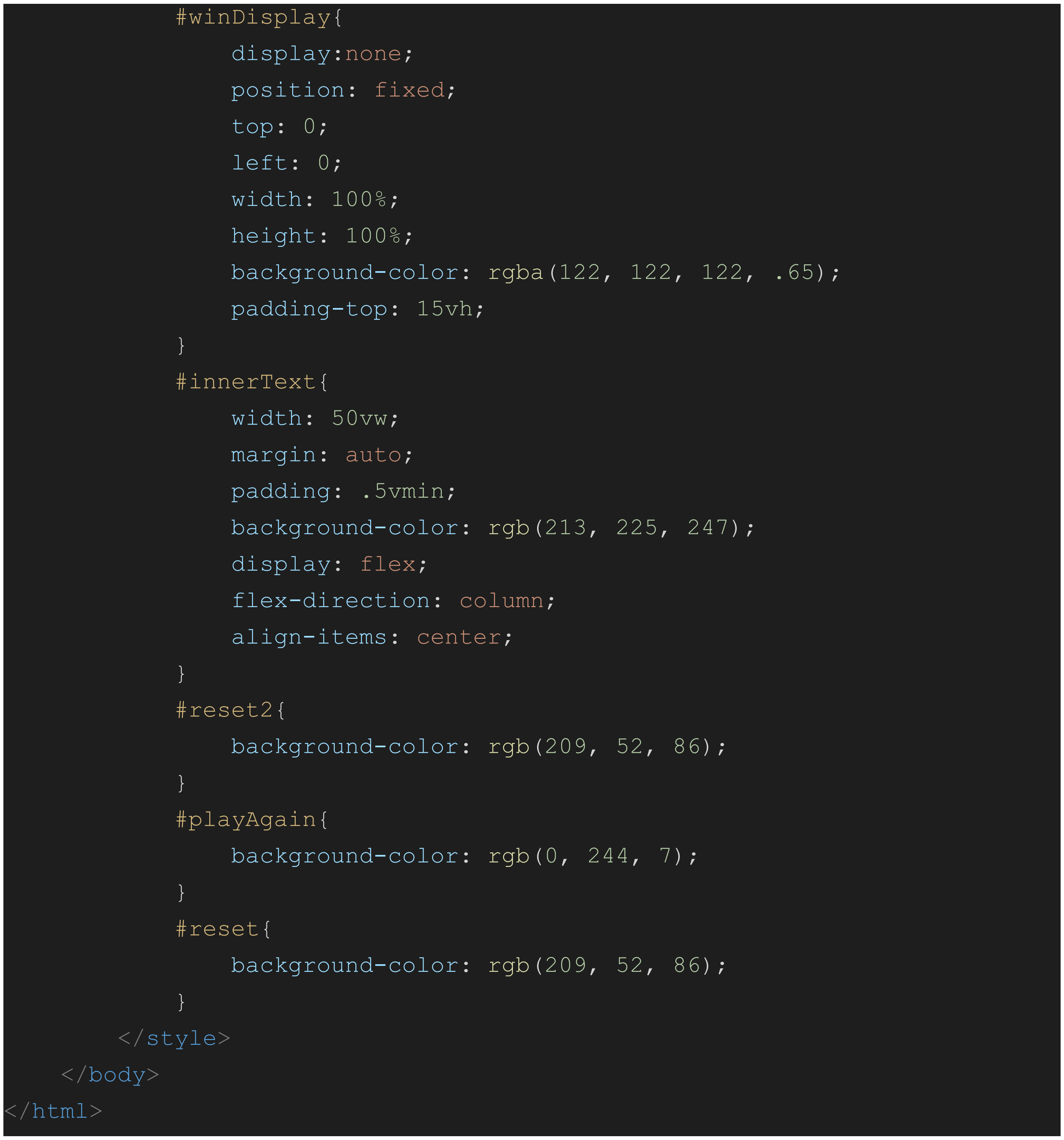
```
            }

            function removeHover(){
                let hoverPiece =
document.querySelector("[data-added='false']");
                if(hoverPiece){
                    hoverPiece.parentElement.removeChild(hoverPiece);
                }
            }


            function updateTop(){
                removeHover();
                if(spots[currentHover] == 0){
                    let space = board.children[currentHover];
                    let top = document.createElement("div");
                    top.className = "top";
                    top.dataset.added = false;
                    top.dataset.turn = turn;
                    space.appendChild(top);
                }
            }

            function checkForWin(){
                dropping = false;
                if(playerWin(turn, spots)){
                    if(turn == 1){
                        p1score += 1;

document.getElementById("winDisplay").style.display = "block";
                        document.getElementById("winner").innerHTML =
document.getElementById("p1name").value + " Won!"
                    }
                    else{
                        p2score += 1;

document.getElementById("winDisplay").style.display = "block";
```

```
                        document.getElementById("winner").innerHTML =
document.getElementById("p2name").value + " Won!"
                    }
                }
                if(!spots.includes(0)){
                    document.getElementById("winDisplay").style.display =
"block";
                    document.getElementById("winner").innerHTML = "It's a
Draw!"
                }

                if(turn == 1){
                    turn = 2;
                }
                else{
                    turn = 1;
                }
                document.getElementById("score").innerHTML = p1score + " :
" + p2score;
                updateTop();
            }
            function playerWin(turn, spots){
                for (let i = 0; i <42; i++){
                    if(i%7 < 4){
                        if(spots[i] == turn && spots[i+1] == turn
                        && spots[i+2] == turn && spots[i+3] == turn){
                            return(true);
                        }
                    }
                    if(i <= 20){
                        if(spots[i] == turn && spots[i+7] == turn
                        && spots[i+14] == turn && spots[i+21] == turn){
                            return(true);
                        }
                    }
                    if(i%7 < 4 && i <=20){
                        if(spots[i] == turn && spots[i+8] == turn
                        && spots[i+16] == turn && spots[i+24] == turn){
                            return(true);
                        }
```



```
                    }
                    if(i%7 > 2 && i <=20){
                        if(spots[i] == turn && spots[i+6] == turn
                        && spots[i+12] == turn && spots[i+18] == turn){
                            return(true);
                        }
                    }
                }
                return(false);
            }

        </script>
        <style>
            *{
                font-family: 'Courier New', Courier, monospace;
            }
            h1{
                font-size:60px;
                margin: 0vh;
            }
            p{
                font-size:30px;
                margin: 0;
                display: inline;
            }
            body{
                display: flex;
                flex-direction: column;
                align-items: center;
                background-color:rgb(183, 255, 253);

            }
            input{
                height:45px;
                width:200px;
                border-radius:10px;
                font-size:xx-large;
            }
            #board{
                align-content: center;
                display: grid;
                grid-template-columns: repeat(7, 1fr);
                grid-template-rows: repeat(6, 1fr);
                width: 70vmin;
                height: 60vmin;
                justify-content: center;
                padding-top:10vmin;
            }
            .space{
                display: flex;
                background-image: radial-gradient(transparent 65%, rgb(0,
110, 173) 50%)
            }
            .top{
                border-radius: 50%;
                flex-grow: 1;
            }
            .top[data-added='false']{
                transform: translateY(-10vmin);
            }
            .top[data-turn='1']{
                border: 3px solid black;
                background-color: rgb(121, 1, 73);
            }
            .top[data-turn='2']{
                border: 3px solid black;
                background-color: rgb(255, 110, 57);
            }
            button{
                height:45px;
                font-size:30px;
            }
            #players{
                margin: 5px;
                display: inline;
                font-size:30px;
            }
            #player1{
                margin: 5px;
                display: inline;
```

```
    font-size:30px;
}
#p1name{
    margin: 5px;
    display: inline;
    font-size:30px;
}
#player2{
    margin: 5px;
    display: inline;
    font-size:30px;
}
#p2name{
    margin: 5px;
    display: inline;
    font-size:30px;
}
#p1Appear{
    margin: 5px;
    display: inline;
    font-size:45px;
    color:rgb(121, 1, 73);
}
#p2Appear{
    margin: 5px;
    display: inline;
    font-size:45px;
    color:rgb(255, 110, 57);
}
#vs{
    margin: 5px;
    display: inline;
    font-size:40px;
}
#score{
    margin: 5px;
    display: inline;
    font-size:40px;
}
```



```
            #winDisplay{
                display:none;
                position: fixed;
                top: 0;
                left: 0;
                width: 100%;
                height: 100%;
                background-color: rgba(122, 122, 122, .65);
                padding-top: 15vh;
            }
            #innerText{
                width: 50vw;
                margin: auto;
                padding: .5vmin;
                background-color: rgb(213, 225, 247);
                display: flex;
                flex-direction: column;
                align-items: center;
            }
            #reset2{
                background-color: rgb(209, 52, 86);
            }
            #playAgain{
                background-color: rgb(0, 244, 7);
            }
            #reset{
                background-color: rgb(209, 52, 86);
            }
        </style>
    </body>
</html>
```