```
<!-- Please see the beat_demo.html for more details on how this code works.  All
 comments will be related to
new features/functions that were used for this specific example. -->
<!DOCTYPE HTML>
<html>
    <head>
        <title>Connect Four</title>
        <script type="text/javascript">
            /* set the global variables so we can use them wihin functions */
            var game_active = false; // this is a boolean (true/false values
             only).  This will be used to prevent being able to drop pieces once
             the game is over
            var active_player = 0; // the # of the active player - 1 or 2.
             Default is 0, meaning no active player
            var gameboard = []; // define the gameboard as an array.  We will
             later set it up as a multi-dimensional array, to represent the col/
             row value for the game board
            var player_color = []; //define player_color as an array
            player_color[1] = "red"; //set the player_color for player 1 to "red"
            player_color[2] = "blue"; // set the player_color for player 2 to
             "blue"

            function beginGame() {
                //don't reset the game until the last one is done.  So if the game
                 is still active, return (don't continue with
                //the function)  When you return, the rest of the function is
                 skipped, and the value you return will be
                //passed to the function that called the function.  Since
                 beginGame is not called as a part of a different
                //function, it doesn't matter if you return true or false.
                if (game_active == true) return false;

                game_active = true;  //We're starting the game, so make
                 game_active = true
                /* Reset the gameboard to be all 0.  We are going to use a multi-
                 dimensional array - so every
                section of the board will be represented by a point on the grid -
                 X and Y (col and row).  Top left will be
                0,0 and as it moves to the right, that will be +x and down will be
                 +y.
                For example, the values of the board will be:
                | 0,0 | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 |
                | 1,0 | 1,1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,6 |
                | 2,0 | 2,1 | 2,2 | 2,3 | 2,4 | 2,5 | 2,6 |
                | 3,0 | 3,1 | 3,2 | 3,3 | 3,4 | 3,5 | 3,6 |
                | 4,0 | 4,1 | 4,2 | 4,3 | 4,4 | 4,5 | 4,6 |
                | 5,0 | 5,1 | 5,2 | 5,3 | 5,4 | 5,5 | 5,6 |

                */
                for (row=0; row<=5; row++) {
                    gameboard[row] = [];
                    for (col=0; col<=6; col++) {
                        gameboard[row][col] = 0;
                    }
                }

                drawBoard(); // call the function to draw the board.
```

```
    active_player = 1; //set the first player as their turn
    setUpTurn(); //get ready for the player's turn
}

/* drawBoard will draw the board - it will update each item to make
 sure it is the appropriate value */
function drawBoard() {
    checkForWin(); //check to see if any player has won.
    for (col = 0; col<=6; col++) {
        for (row=0; row<=5; row++) {
            //Set the inner HTML of the square (a td) to be a span
             with the class of 'piece' and 'player' + the value of
             that
            //gameboard piece.  Using CSS, you can style player0,
             player1 and player2 so that the square will appear
             correctly.
            document.getElementById('square_'+row+'_'+col).innerHTML
             ="<span class='piece player"+gameboard[row][col]+"'> </
             span>";
        }
    }
}

function checkForWin() {
    /* There are many ways this algorithm can be accomplished.
     Basically you want to check all possibility for a win.
    Given the size of the board, checking all possibilities will not
     be a huge task for the computer, so I will go
    with an easy to understand, straightforward algorithm */
    /* Ultimately there are 4 ways to win - left-to-right, diagnol up,
     diagnol down, and top to bottom.  This
    function will check each of these situations twice - one for each
     player. */

    //check left-to-right
    //check for player 1 and 2
    for (i=1; i<=2; i++) {
        //since a winning row must be 4 long, we only need to check
         the first 3 rows, 0,1,and 2
        for (col = 0; col <=3; col++) {
            for (row = 0; row <=5; row++) {
                //check to see if the gameboard item is set to the
                 player we are checking, if so, lets check the next 3
                 for a match
                if (gameboard[row][col] == i) {
                    if ((gameboard[row][col+1] == i) &&
                     (gameboard[row][col+2] == i) && (gameboard[row]
                     [col+3] == i)) {
                        endGame(i);//a match has been made, so run
                         EndGame with the player that had the win
                        return true; //stop checking for a win - the
                         game is over.
                    }
                }
            }
        }
    }
```

```
//check top-to-bottom
for (i=1; i<=2; i++) {
    //since a winning row must be 4 long, we only need to check
     the first 3 rows, 0,1,and 2
    for (col = 0; col <=6; col++) {
        for (row = 0; row <=2; row++) {
            //check to see if the gameboard item is set to the
             player we are checking, if so, lets check the next 3
             for a match
            if (gameboard[row][col] == i) {
                if ((gameboard[row+1][col] == i) &&
                 (gameboard[row+2][col] == i) && (gameboard[row+3]
                 [col] == i)) {
                    endGame(i); //a match has been made - run
                     endGame for the player who had the match.
                    return true; //stop checking for a win - the
                     game is over.
                }
            }
        }
    }
}

//check diagnol down
for (i=1; i<=2; i++) {
    //since a winning row must be 4 long, we only need to check
     the first 3 rows, 0,1,and 2
    for (col = 0; col <=3; col++) {
        //we also only need to check the bottom most columns - as
         the win must be upwards
        for (row = 0; row <=2; row++) {
            //check to see if the gameboard item is set to the
             player we are checking, if so, lets check the next 3
             for a match
            if (gameboard[row][col] == i) {
                if ((gameboard[row+1][col+1] == i) &&
                 (gameboard[row+2][col+2] == i) &&
                 (gameboard[row+3][col+3] == i)) {
                    endGame(i);
                    return true;
                }
            }
        }
    }
}

//check diagnol up
for (i=1; i<=2; i++) {
    //since a winning row must be 4 long, we only need to check
     the first 3 rows, 0,1,and 2
    for (col = 0; col <=3; col++) {
        //we also only need to check the bottom most columns - as
         the win must be upwards
        for (row = 3; row <=5; row++) {
```

```
                    //check to see if the gameboard item is set to the
                     player we are checking, if so, lets check the next 3
                     for a match
                    if (gameboard[row][col] == i) {
                        if ((gameboard[row-1][col+1] == i) &&
                         (gameboard[row-2][col+2] == i) &&
                         (gameboard[row-3][col+3] == i)) {
                            endGame(i);
                            return true;
                        }
                    }
                }
            }
        }
    }

    /* endGame will end the game - any additional functions or things you
     want to happen when the game is over can go here */
    function endGame(winningPlayer) {
        game_active = false; //set the "game_active" to false, so that it
         can be started again.
        document.getElementById('game_info').innerHTML = "Winner: " +
         winningPlayer; //set the "game_info" to the winner and the
         winning player #
    }

    /* setUpTurn will display who is the active player */
    function setUpTurn() {
        if (game_active) { //only run this is the game is active.
            //display the current player, and create a <span> with the
             class of the player# so that it will show the color.
            document.getElementById('game_info').innerHTML = "Current
             Player: Player " + active_player + " <span
             class='player"+active_player+"'>(" +
             player_color[active_player] + ")</span>";
        }
    }

    /* drop will add a piece to the lowest available column */
    function drop(col) {
            /* Look for the lowest point in this row that is open */
            //the opposite of our loop above - as we're going to start
             from the bottom looking for an open slot
            for (row=5; row>=0; row--) { //note: we are using row--, which
             will reduce the value of row by 1, the opposted of ++
                if (gameboard[row][col] == 0) {
                    //set the empty row to the active player's number
                    gameboard[row][col] = active_player;
                    drawBoard(); // draw the board.
                    //change the active players turn:
                    if (active_player == 1) {
                        active_player = 2;
                    } else {
                        active_player = 1;
                    }
```

```
                    //there is also a fancy way of doing this call a
                     ternary assignment that looks like this:
                     active_player = (active_player == 1) ? 2 : 1;

                    setUpTurn(); //display who is the active player

                    //stop looking for empty spaces
                    return true;
                }
            }
    }


</script>

<style>
    .click_button {
        height: 42px;
        width: 42px;
        border-radius: 24px;
        background-color: rgba(255,0,0,1);
        cursor: pointer;
    }

    /* Make each piece square - 32x32, set the border-radius to 16px (so
     that it appears round) and center it using margin-left
    and margin-right: auto */
    .piece {
        height: 32px;
        width: 32px;
        display:block;
        border-radius: 16px;
        margin-left: auto;
        margin-right: auto;
    }

    /* Player 0 (no player) will have a light-grey background */
    .player0 {
        background-color: #DDDDDD; /* hex colors were used, but rgb/rgba
         could be used as well */
    }

    /* player 1 will have a red background */
    .player1 {
        background-color: #FF0000;
    }

    /* player 2 will have a blue background */
    .player2 {
        background-color: #0000FF;
    }

    /* set the board square to having a dark-grey border, and to be 36x36
     square */
    .board_square {
        border: 1px solid #555555;
        height: 36px;
```

```
            width: 36px;
        }

    </style>

</head>
<body onload="beginGame();">
    <!-- This is the heading HTML.  It is instructions on how to play -->
    <h1>Connect Four</h1>
    <p>Connect Four is a game where players take turn dropping colored circles
     in the top of the board, along different rows.  The goal is to make a row
        of 4 circles in a row - straight or diagnol.</p>
        <!-- Challenge: write an AI player -->
    <p>This demo does not have a computer partner.</p>

    <!-- The begin_game button will run "beginGame()" function when clicked.
     The () is required at the end to show it is a
    function, but no parameters are displaying -->
    <button id="begin_game" onclick="beginGame();">Begin Game</button>

    <!-- All the action takes place in the gameboard, so I'm containing the
     gameboard in a div with the ID "gameboard" so that
    you can always target any items within easily. -->
    <div id="gameboard">
        <!-- The game_info div allows game-specific information to easily be
         displayed -->
        <div id="game_info">
        </div>
        <!-- The game table is the actual board.  A table seems the most
         logical structure for the data, as tables have rows and columns
        although you most certainly could use divs, or other HTML elements.
         The top row (and tablehead - 'thead') is reserved for the
        buttons to drop the game pieces. -->
        <table id="game_table">
            <thead>
                <tr>
                    <!-- Set up the buttons that will drop the pieces in to
                     the column.  This could be programmatically generated
                     like
                    the rows/columns below, but since it was just 7, it
                     wouldn't really save code -->
                    <td><button onclick="drop(0);">Drop</button></td>
                    <td><button onclick="drop(1);">Drop</button></td>
                    <td><button onclick="drop(2);">Drop</button></td>
                    <td><button onclick="drop(3);">Drop</button></td>
                    <td><button onclick="drop(4);">Drop</button></td>
                    <td><button onclick="drop(5);">Drop</button></td>
                    <td><button onclick="drop(6);">Drop</button></td>
                </tr>
            </thead>
            <script>
            //To make the code much more compact, javascript will actually be
             used to write the code for the game board
            /*
            a for loop is a basic concept for running a set of code multiple
             times.  It has 3 parameters, the first is a variable
```

```
                name, in this case I used "row" and "col" I then set the value to
                 0.  The next parameter is 'under what conditions should
                i repeat the code below.  As long as that 2nd condition is "true"
                 the code below will run.  The last parameter
                is the increment.  This is run every time the code below is run.
                 In the loops below, I have i++ and j++ which means
                "add one to the value of row/col"
                */
                for (var row=0; row<=5; row++) {
                    /*document.writeln() function will write HTML code to the
                     browser.  If you "inspect" this page after the browser has
                    rendered the page, you will see a bunch of HTML
                    write the start of the table row tag
                    */
                    document.writeln("<tr>");
                    for (var col=0; col<=6; col++) {
                        //write each table data element - with the row and col
                         variables in the ID so it can be accessed later.
                        document.writeln("<td id='square_" + row + "_"+ col +"'
                         class='board_square'></td>");
                    }
                    //write the closing table row tag.
                    document.writeln("</tr>");
                }

                </script>
            </table>
            <!-- game_status is a place for the game to provide status updates. --
             >
            <div id="game_status">
            </div>
        </div>
    </body>
</html>
```