Connect Four OO: Solution

Springboard

You can download our solution code

Our HTML, with the "enter player colors" form:

index.html

```
<!doctype html>
<head>
  <title>Connect 4</title>
  <link href="connect4.css" rel="stylesheet">
</head>
<body>

<input id="p1-color" placeholder="Player 1 color">
<input id="p2-color" placeholder="Player 2 color">
<button id="start-game">Start Game!</button>

<div id="game">
  <table id="board"></table>
</div>

<script src="connect4.js"></script>

</body>
</html>
```

Our CSS (the same, but no longer needs hard-coded player colors):

connect4.css

```
/* game board table */

#board td {
  width: 50px;
  height: 50px;
  border: solid 1px #666;
}

/* pieces are div within game table cells: draw as colored circles */

.piece {
  margin: 5px;
  width: 80%;
  height: 80%;
  border-radius: 50%;
}

/* column-top is table row of clickable areas for each column */

#column-top td {
  border: dashed 1px lightgray;
}

#column-top td:hover {
  background-color: gold;
}
```

Our JS:

connect4.js

```
/** Connect Four
 *
 * Player 1 and 2 alternate turns. On each turn, a piece is dropped down a
 * column until a player gets four-in-a-row (horiz, vert, or diag) or until
 * board fills (tie)
 */

class Game {
  constructor(p1, p2, height = 6, width = 7) {
    this.players = [p1, p2];
    this.height = height;
    this.width = width;
    this.currPlayer = p1;
    this.makeBoard();
    this.makeHtmlBoard();
    this.gameOver = false;
  }

  /** makeBoard: create in-JS board structure:
   *   board = array of rows, each row is array of cells  (board[y][x])
   */
  makeBoard() {
    this.board = [];
    for (let y = 0; y < this.height; y++) {
      this.board.push(Array.from({ length: this.width }));
    }
  }

  /** makeHtmlBoard: make HTML table and row of column tops.  */

  makeHtmlBoard() {
    const board = document.getElementById('board');
    board.innerHTML = '';

    // make column tops (clickable area for adding a piece to that column)
    const top = document.createElement('tr');
    top.setAttribute('id', 'column-top');

    // store a reference to the handleClick bound function
    // so that we can remove the event listener correctly later
    this.handleGameClick = this.handleClick.bind(this);

    top.addEventListener("click", this.handleGameClick);

    for (let x = 0; x < this.width; x++) {
      const headCell = document.createElement('td');
      headCell.setAttribute('id', x);
      top.append(headCell);
    }

    board.append(top);

    // make main part of board
    for (let y = 0; y < this.height; y++) {
      const row = document.createElement('tr');

      for (let x = 0; x < this.width; x++) {
        const cell = document.createElement('td');
        cell.setAttribute('id', `${y}-${x}`);
        row.append(cell);
      }

      board.append(row);
    }
  }

  /** findSpotForCol: given column x, return top empty y (null if filled) */

  findSpotForCol(x) {
    for (let y = this.height - 1; y >= 0; y--) {
      if (!this.board[y][x]) {
        return y;
      }
    }
    return null;
  }

  /** placeInTable: update DOM to place piece into HTML board */

  placeInTable(y, x) {
    const piece = document.createElement('div');
    piece.classList.add('piece');
    piece.style.backgroundColor = this.currPlayer.color;
    piece.style.top = -50 * (y + 2);

    const spot = document.getElementById(`${y}-${x}`);
    spot.append(piece);
  }

  /** endGame: announce game end */

  endGame(msg) {
    alert(msg);
    const top = document.querySelector("#column-top");
    top.removeEventListener("click", this.handleGameClick);
  }

  /** handleClick: handle click of column top to play piece */

  handleClick(evt) {
    // get x from ID of clicked cell
    const x = +evt.target.id;

    // get next spot in column (if none, ignore click)
    const y = this.findSpotForCol(x);
    if (y === null) {
      return;
    }

    // place piece in board and add to HTML table
    this.board[y][x] = this.currPlayer;
    this.placeInTable(y, x);

    // check for tie
    if (this.board.every(row => row.every(cell => cell))) {
      return this.endGame('Tie!');
    }

    // check for win
    if (this.checkForWin()) {
      this.gameOver = true;
      return this.endGame(`The ${this.currPlayer.color} player won!`);
    }

    // switch players
    this.currPlayer =
      this.currPlayer === this.players[0] ? this.players[1] : this.players[0];
  }

  /** checkForWin: check board cell-by-cell for "does a win start here?" */

  checkForWin() {
    // Check four cells to see if they're all color of current player
    //  - cells: list of four (y, x) cells
    //  - returns true if all are legal coordinates & all match currPlayer
    const _win = cells =>
      cells.every(
        ([y, x]) =>
          y >= 0 &&
          y < this.height &&
          x >= 0 &&
          x < this.width &&
          this.board[y][x] === this.currPlayer
      );

    for (let y = 0; y < this.height; y++) {
      for (let x = 0; x < this.width; x++) {
        // get "check list" of 4 cells (starting here) for each of the different
        // ways to win
        const horiz = [[y, x], [y, x + 1], [y, x + 2], [y, x + 3]];
        const vert = [[y, x], [y + 1, x], [y + 2, x], [y + 3, x]];
        const diagDR = [[y, x], [y + 1, x + 1], [y + 2, x + 2], [y + 3, x + 3]];
        const diagDL = [[y, x], [y + 1, x - 1], [y + 2, x - 2], [y + 3, x - 3]];

        // find winner (only checking each win-possibility as needed)
        if (_win(horiz) || _win(vert) || _win(diagDR) || _win(diagDL)) {
          return true;
        }
      }
    }
  }
}

class Player {
  constructor(color) {
    this.color = color;
  }
}

document.getElementById('start-game').addEventListener('click', () => {
  let p1 = new Player(document.getElementById('p1-color').value);
  let p2 = new Player(document.getElementById('p2-color').value);
  new Game(p1, p2);
});
```