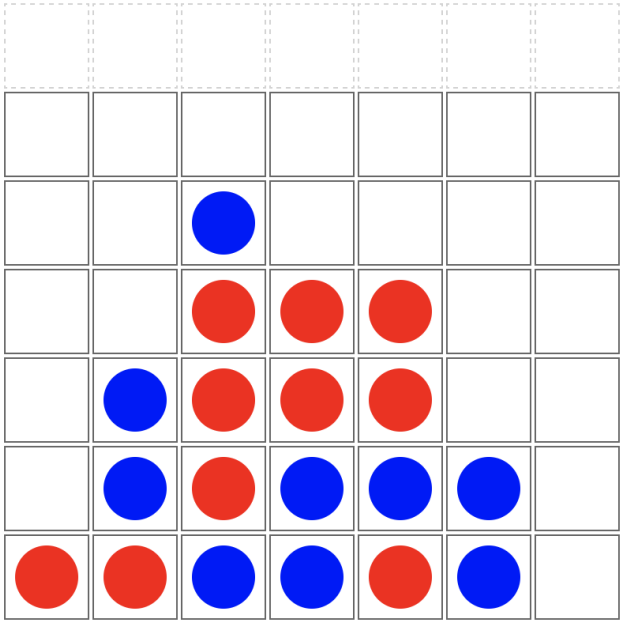
Connect Four

Download code <../connect-four.zip>

In this exercise, you plan & help code a Connect Four game in Javascript.

The Game



<_images/connect4.png>

Connect Four is played on a grid, 7 wide by 6 deep, with two players, 1 (red) and 2 (blue). The players alternate turns, dropping a piece of their color in the top of a column. The piece will fall down to the further-down unoccupied slot.

The game is won when a player makes four in a row (horizontally, vertically, or diagonally). The game is a tie if the entire board fills up without a winner.

You can try out the game online <http://connect4-rithm.surge.sh>.

Step One: Planning

Before looking at our code, take a few minutes to think about how you would build a game like this using HTML/JS/CSS:

- what HTML would be useful for the game board itself?
- how could you represent a played-piece in the HTML board?
- in the JavaScript, what would be a good structure for the in-memory game board?
- what might the flow of the game be?

Then, write down some functions names/descriptions that would be useful for this game.

Step Two: ES2015

This code would benefit from updating to ES2015 style — there are lots of place where ***var*** is used that could be changed to either ***let*** or ***const*** to improve readability. Are there other style fixes you can make?

Step Three: *makeBoard*

The ***makeBoard()*** function needs to be implemented. It should set the global ***board*** variable to be an array of 6 arrays (height), each containing 7 items (width).

You *could* do this like:

```
const board = [
  [ null, null, null, null, null, null, null ],
  [ null, null, null, null, null, null, null ],
  [ null, null, null, null, null, null, null ],
  [ null, null, null, null, null, null, null ],
  [ null, null, null, null, null, null, null ],
  [ null, null, null, null, null, null, null ],
];
```

However, it's far better to make the game flexible about the height and width of the board and use the ***WIDTH*** and ***HEIGHT*** constants in ***connect4.js***. Implement this function to make this board dynamically.

Step Four: *makeHTMLBoard*

This function is missing the first line, that sets the ***board*** variable to the HTML board DOM node. Fix this.

Add comments to the code that dynamically creates the HTML table.

Step Five: *placeInTable* & Piece CSS

This function should add a ***div*** inside the correct ***td*** cell in the HTML game board. This div should have the ***piece*** class on it, and should have a class for whether the current player is 1 or 2, like ***p1*** or ***p2***.

Update the CSS file to:

- make the piece ***div*** round, not square
- be different colors depending on whether it's a player #1 or #2 piece

While not everything will work, you should now be able to click on a column and see a piece appear at the very bottom of that column. (They won't yet appear in the right row and will always be player #1 pieces)

Step Six: *handleClick*

There are several pieces to write/fix here:

- this never updates the ***board*** variable with the player #. Fix.
- add a check for "is the entire board filled" [hint: the JS ***every*** method on arrays would be especially nice here!]
- add code to switch ***currPlayer*** between 1 and 2. This would be a great place for a ternary function.

Step Seven: *findSpotForCol* and *endGame*

Right now, the game drops always drops a piece to the top of the column, even if a piece is already there. Fix this function so that it finds the lowest empty spot in the game board and returns the y coordinate (or ***null*** if the column is filled).

Once you have this working, make sure that when a game has ended, the endGame function runs and alerts which user has won!

Step Eight: CELEBRATE!

If you got this far, you should have a fully functional Connect Four game. Congratulations!

Further Study

Optional Step Nine: Read & Comment *checkForWin*

The ***checkForWin()*** function is already written, but it needs comments to help explain how it works. Add some!

Note: this is a good strategy for finding a winner, but it's not the most efficient. Later, you may learn ways to find winners that don't keep re-checking the same area of the board [using techniques for "dynamic programming", you can make this code more efficient, though it's much more advanced than the rest of this exercise. You can come back to this code much later!]

Optional Step Ten: Add Animation!

You can learn about CSS animation features (check out MDN!). If you change the ***.piece*** divs to be positioned absolutely, you can animate the ***top*** CSS property to animate the pieces so they appear to drop down. This is tricky, but will give you a chance to play with animations, as well as working with relative/absolute positioning.

Solution

See Our solution <solution/index.html>.