

## Assignment #7—Match the Flag

---

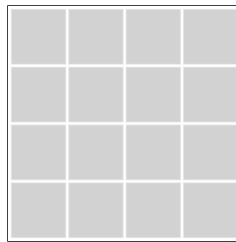
This week’s assignment marks our return to pure JavaScript, as you’ll implement a small, easily described game that relies on your understanding of HTML, CSS, and your ability to programmatically analyze and perform surgery on the DOM tree.

The amount of code that needs to be written is in line with the amount that was needed for Assignments 2 and 3. In particular, this assignment, while hardly trivial, lacks the heft and gravity of the more elaborate Enigma, Reassemble, and Adventure assignments. We recognize you’re all swamped and working very hard, so we’re hoping this assignment is easier—particularly given the break that really should be a break—without sacrificing academic rigor.

**Due: Friday, December 1<sup>st</sup>, 5:00 P.M.**

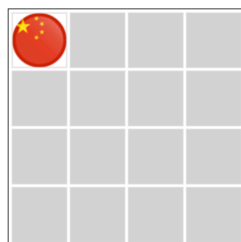
### The Game

Match the Flags is a memory game where a single player is presented with a 4x4 grid of concealed images, like so:



Each image is one of eight different flags, so each image is present not once, but twice. The goal of the game is to click on squares to reveal flags and match pairs.

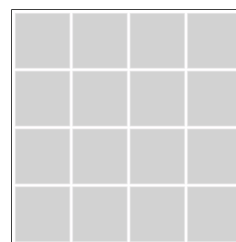
For instance, clicking the upper left square and the one below it would reveal two images, as with:



*after first click*



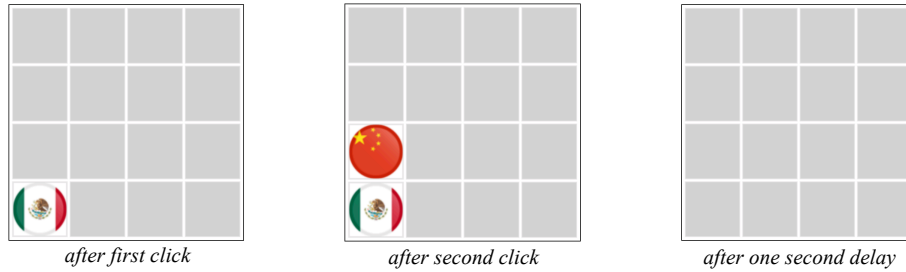
*after second click*



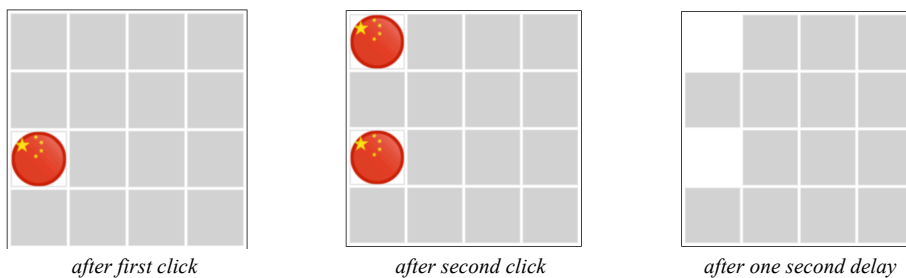
*after one second delay*

When the two selections mismatch as they do above, you’re given one second to commit the exposed images to memory before their covers are restored.

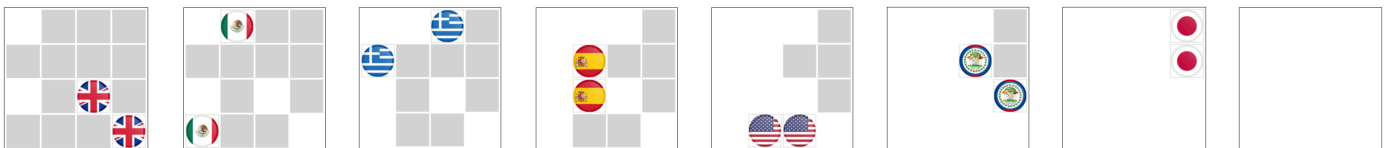
The next pair of clicks might reveal another mismatch:



Of course, now you've seen both Chinese flags, so you'd surely click to reveal each of them a second time. Once the two have been exposed, you're given one second to relish your tiny victory before the two images are removed from the screen.



The player continues to expose pairs of matching flags—presumably with intermittent mismatches I don't bother to show—until all pairs have been matched.



For this assignment, you're to implement a program to include the mouse event and timer functionality needed to realize the full game experience. You're to do this without the services of the CS106AX graphics library. Instead, you're to leverage your recently acquired understanding of the DOM and how to manipulate it via JavaScript code.

### The Program

The starter code includes one HTML file, one CSS file, and two JavaScript files. The HTML file is short, as it just defines a skeletal structure you'll immediately modify once your JavaScript program gets a change to load and execute.

That HTML file is presented right here:



```

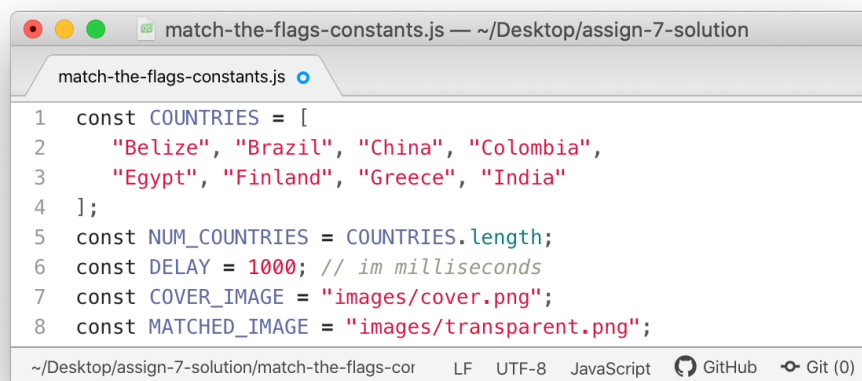
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title>Match The Flag</title>
6     <link rel="stylesheet" type="text/css" href="match-the-flags.css"/>
7     <script type="text/javascript" src="match-the-flags-constants.js"></script>
8     <script type="text/javascript" src="match-the-flags.js"></script>
9   </head>
10  <body>
11    <div id="board"></div>
12  </body>
13 </html>
14

```

As you can see, all you get is an empty `div` that's programmatically discoverable through its `id` attribute.

The CSS file is also very small, as it only defines rules for `body`, `div`, and `img` tags. As far as CSS files go, it's so easily ignored that I won't even present it here. You can just assume the CSS rules do what they need to do for you. There are no `class` or `id` selectors, so you won't need to add and remove any `class` attributes or `classList` properties for this particular application.

The first of the two JavaScript files is called `match-the-flag-constants.js`, and a minified, uncommented version of it looks like this:



```

1 const COUNTRIES = [
2   "Belize", "Brazil", "China", "Colombia",
3   "Egypt", "Finland", "Greece", "India"
4 ];
5 const NUM_COUNTRIES = COUNTRIES.length;
6 const DELAY = 1000; // in milliseconds
7 const COVER_IMAGE = "images/cover.png";
8 const MATCHED_IMAGE = "images/transparent.png";

```

The `COUNTRIES` array names the eight countries whose flags contribute to the game. Of

course, you need each country to be represented twice in any given play, so all of the images needed for each of the eight countries reside in a subfolder called `images`. The image names are just the country names, except that image names are all lowercase. That means that "`images/finland.png`" is an image of Finland's flag, "`images/greece.png`" is an image of Greece's flag, as so forth. (Note that these filenames are all lowercase, even though the `COUNTRIES` array has proper capitalization.)

You'll also notice that I have two other named images.

The cover image, as it's called, is pretty dull. It's just a solid gray swatch that can be used in place of an actual flag when the flag is concealed. The second one is even more dull than the first! It's a fully transparent image—that is, it's totally see-through. The second one is useful, because I don't actually want to **remove `img`** nodes from the window, since other images would move to close any gaps and destroy the stable 4x4 grid structure. We can, however, give the impression that an image has been removed by replacing that image with a see-through one. That's what this second image is for.

### Suggested Milestones

Here's a series of milestones you should work through to arrive at a working product as efficiently as possible:

1. Create an array of 16 flag image filenames, as strings, two for each country, and scramble it using the provided `shuffle` function. The shuffle function relies on some basic probability to ensure that all possible permutations are equally likely.
2. Create 16 `img` nodes on behalf of each entry in the array and insert each in turn to the one `div` where all images belong. The CSS is set up so that images flow left to right just as we read, say, assignment handouts, and the width and height of the `div` are set so that the image flow ends up being a perfect 4x4 grid.

Don't worry about event handling just yet. Just get those images to show up in their full glory. Create `img` nodes using `document.createElement`, and set image sources by calling `element.setAttribute("src", imageName)`.

3. Update the code you wrote for the second milestone so that each image shows up as a gray square instead of a flag. This will require replacing the image filename in the `src` attribute with the gray square filename, so you should add a new `data-country-image` attribute that contains the original image name. That way, you can easily replace the dull gray image with the original country image when the square is clicked. You can always retrieve the value of any attribute—and in particular, the `src` or `data-country-image` attribute—by calling `element.getAttribute(attributeName)`.
4. Attach mouse click event listeners to each `img` node and implement a simple version of the click-handling function that toggles a clicked tile between the gray and the flag it conceals. Restated, if a tile showing a country flag is clicked, replace it with the

gray image, and if a tile showing the gray image is clicked, replace it with the country flag.

Don't create new `img` nodes, or you'll make yourself sad. Just examine and update the `src` attribute of existing `img` nodes by relying on `setAttribute` and `getAttribute`.

5. Now **update** the mouse click event callback function so that you really do play the game. Detect all mouse clicks on image tags, though immediately return from the callback if the `src` of the clicked image is anything other than the gray swatch. Otherwise, proceed to reveal the flag, and if there are two revealed flags, then wait one seconds before replacing the source of two images to be either gray or transparent.

During that one second wait time, make sure you programmatically ignore any other mouse clicks. In particular, don't allow a third flag to be revealed by a third click until the one second has passed.

When all covered flags have been removed, the game is over. No additional code need be written to address the end of game. User clicks are effectively ignored, since all flags have been matched.

That's all! I hope you enjoy the assignment! By completing it, you'll bolster your understanding of JavaScript and how it can update the native data structures used by all modern browsers to manage the presentation and interactivity of a web application!