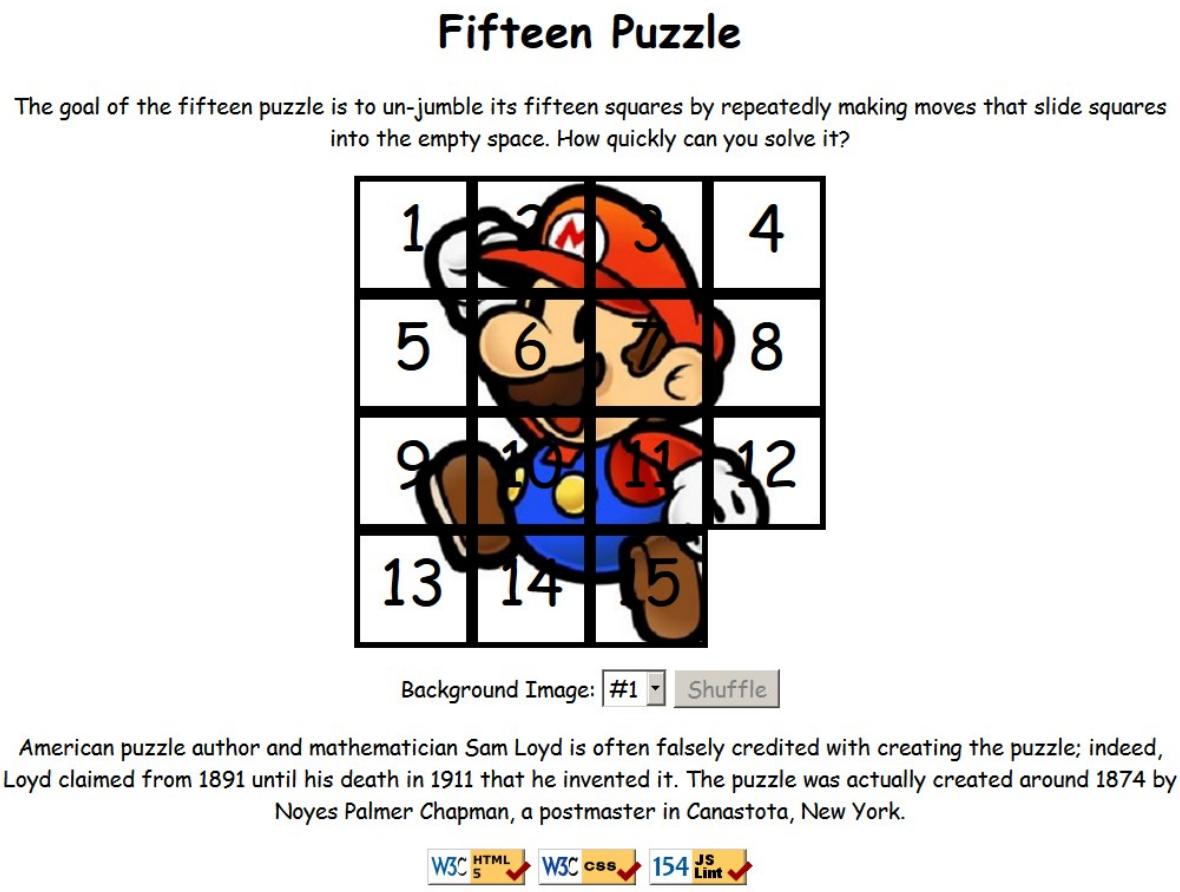
# University of Washington, CSE 154
# Homework Assignment 8: Fifteen Puzzle (Part A)

This assignment is about JavaScript's Document Object Model (DOM) and events. You'll write the following page:




**Background Information:**

The Fifteen Puzzle (also called the Sliding Puzzle) is a simple classic game consisting of a 4x4 grid of numbered squares with one square missing. The object of the game is to arrange the tiles into numerical order by repeatedly sliding a square that neighbors the missing square into its empty space.

You will write the CSS and JavaScript code for a page **fifteen.html** that plays the Fifteen Puzzle. You will also submit a **background image** of your own choosing, displayed underneath the tiles of the board. Choose any image you like, so long as its tiles can be distinguished on the board. Turn in the following files:

- **fifteen.js**: the JavaScript code for your web page
- **fifteen.css**: the CSS styles for your web page

We will provide you with the HTML code to use, which you should not modify for Part A. You will write JavaScript code that interacts with the page using the DOM. To modify the page's appearance, write appropriate DOM code to change styles of on-screen elements by setting classes, IDs, and/or style properties on them.

This is a two-part assignment. In the second part, you will later add features such as a puzzle shuffling algorithm.

*(Tip for playing the game: First get the entire top/left sides into proper position. That is, put squares number 1, 2, 3, 4, 5, 9, and 13 into their proper places. Now never touch those squares again. Now what's left to be solved is a 3x3 board, which is much easier.*

**Appearance Details:**

All text on the page is displayed in a "cursive" font family, at a default font size of 14pt. Everything on the page is centered, including the top heading, paragraphs, the puzzle, the controls, and the W3C buttons at bottom.

In the center of the page are **fifteen tiles** representing the puzzle. The overall puzzle occupies 400x400 pixels on the page, horizontally centered. Each puzzle tile occupies a total of 100x100 pixels, but 5px on all four sides are occupied by a black border. This leaves 90x90 pixels of area inside each tile. The HTML file given to you does not contain the fifteen `div` elements to represent the puzzle pieces, and you are not supposed to modify the HTML file; so you will have to create the puzzle pieces and add them to the page yourself using the JavaScript DOM. Initially the page should appear with the puzzle in its properly arranged order, from 1 at top-left to 15 at bottom-right.

Each tile displays a number from 1 to 15, in a 40pt font. When the page loads, initially the tiles are arranged in their correct order with the missing square in the bottom-right. Each tile displays part of a background image. There is a **select box** that allows the user to choose different images to appear behind the tiles. Initially **background1.jpg** is chosen. When the user chooses each option from the select box, the following image should display. It should be possible to change the background image at any time, including while in the middle of solving the puzzle. Changing the background image should not change the state of the puzzle; all pieces should remain where they are.

- #1: **https://webster.cs.washington.edu/images/fifteen/background1.jpg**
- #2: **https://webster.cs.washington.edu/images/fifteen/background2.jpg**
- #3: **https://webster.cs.washington.edu/images/fifteen/background3.jpg**

*(In Part B you will be able to use your own custom images here, but for Part A, please use exactly these images.)*

Which part of the image is displayed by each tile is related to that tile's number. The "1" tile shows the top-left 100x100 portion of the image. The "2" tile shows the next 100x100px of the background that would be to the right of the part shown under the "1" tile, and so on.

Your **background image** appears on the puzzle pieces when you set it as the `background-image` of each piece. By adjusting the `background-position` of each div, you can show a different part of the background on each piece. One confusing thing about `background-position` is that the x/y values shift the background behind the element, not the element itself. The offsets are the negation of what you may expect. For example, if you wanted a 100x100px `div` to show the top-right corner of a 400x400px image, set its `background-position` property to `-300px 0px` . The following is a complete listing of the exact `background-position` values each location on the board should have:

| | | | |
|---|---|---|---|
| `0px 0px` | `-100px 0px` | `-200px 0px` | `-300px 0px` |
| `0px -100px` | `-100px -100px` | `-200px -100px` | `-300px -100px` |
| `0px -200px` | `-100px -200px` | `-200px -200px` | `-300px -200px` |
| `0px -300px` | `-100px -300px` | `-200px -300px` | |

Centered under the puzzle tiles is a **Shuffle** button. For now, the button does nothing; we will implement this later.

All other style elements on the page are subject to the preference of the web browser. The screenshots in this document were taken on Windows in Firefox, which may differ from your system.

**Playing the Game:**

When the mouse button is pressed on a puzzle square, if that square is next to the blank square, it is moved into the blank space. If the square does not neighbor the blank square, no action occurs. Similarly, if the mouse is pressed on the empty square or elsewhere on the page, no action occurs.

When the mouse **hovers** over a square that can be moved (neighbors the blank spot), its border and text color should become `red`. Also, the mouse cursor should change into a **"hand" cursor** pointing at the square (do this by setting the CSS `cursor` property to `pointer`.) Once the cursor is no longer hovering on the square, its appearance should revert to its original state. When the mouse cursor hovers over a square that cannot be moved, it should use the system's standard **arrow cursor** (set the `cursor` property to `default`), but it should not have the red text or borders or other effects. (You may find it useful to use the `:hover` CSS pseudo-class to help deal with hovering.)

The game is not required to take any particular action when the puzzle has been won. You can decide if you'd like to pop up an `alert` box congratulating the user or add any other optional behavior to handle this event for now.

**Development Strategy:**

Students generally find this to be a tricky assignment. Here is a suggested ordering for tackling the steps:

- Make the fifteen **puzzle pieces appear** in the correct positions without any background behind them.
- Make the correct parts of the **background** show through behind each tile.
- Write the code that **moves a tile** when it is clicked from its current location to the empty square's location. Don't worry initially about whether the clicked tile is "movable" (whether it is next to the empty square).
- Write code to determine whether a **square can move** or not (whether it neighbors the empty square). Implement a highlight when the mouse hovers over movable tiles. Track where the empty square is at all times.

**Hints:**

- Use **absolute positioning** to set the x/y locations of each puzzle piece. The overall puzzle area must use a `relative` position in order for the x/y offsets of each piece to be relative to the puzzle area's location.
- Convert a string to a number using `parseInt`. This also works for strings that start with a number and end with non-numeric content. For example, `parseInt("123four")` returns 123.
- Many students have bugs related to not setting their DOM style properties using **proper units** and formatting. The string you assign in your JS code must exactly match what would have been in the CSS file. For example, if you want to set the size of an element, a value like `42` or `"42"` will fail, but `"42px"` or `"42em"` will succeed. When setting a background image, a value like `"foo.jpg"` will fail, but `"url(foo.jpg)"` will succeed. When setting a background position, `"42 35"` will fail but `"42px 35px"` will succeed. And so on.
- We suggest that you do *not* explicitly make a `div` to represent the empty square. Keep track of where it is, either by row/column or by x/y position, but don't create an actual element for it. We also suggest that you not store your puzzle squares in a 2-D array. This might seem like a good structure because of the 4x4 appearance of the grid, but it is bad style and will be difficult to keep it up to date as the squares move.
- Many students have redundant code because they don't create **helper functions**. You should consider writing functions for common operations, such as moving a particular square, or for determining whether a given square currently can be moved. The `this` keyword (see book section 9.1) can be helpful for reducing redundancy.

**Extra Features for CSE Majors:**

There are **no extra features** posted for Part A. CSE majors (B sections) do not need to complete any "extra" features on Part A. Some extra features will be posted for Part B that CSE majors will complete.

**Implementation and Grading:**

Since this is a two-part assignment, almost all of the points will come from external correctness (correct appearance and behavior). Other than the following items, coding style and **internal correctness will be ignored** on Part A.

For full credit, your CSS code must pass the **W3C CSS validator**. Your JavaScript code should pass our **JSLint** tool with no errors. Your **.js** file must run in **JavaScript strict mode** by putting `"use strict";` at the top.

The top of each file you turn in should have a descriptive **comment header** describing yourself and the assignment. Other than these comment headers, comments are not graded on Part A.

You should not use any external **JavaScript frameworks or libraries** such as jQuery to solve this assignment.

Do not place your solution on a public web site. Submit your own work and follow the course misconduct policy. Put your files on **Webster** at: **https://webster.cs.washington.edu/*your_uwnetid*/hw8/fifteen.html**