Homework Assignment 8: Fifteen Puzzle
EXTRA FEATURES

CSE majors (B and D sections) must complete **at least one (1)** of the following extra features to add new functionality to the program. Non-CSE majors (A and C sections) are not required to complete any of these extra features.

Regardless of how many additions you write, the page behavior and appearance should still work as described in the main spec document. If you have a different idea for an extra feature, please ask us and we may approve it.

If you choose to do any of these extra features, **indicate in your file's comment header** which one(s) you did.

Extra Feature #1. End-of-game Notification

Add code to detect when the user has **solved the puzzle**. When the user moves all of the pieces of the puzzle into their proper places, your page should display a congratulatory message to the user. Any message is fine, but it must contain the word "congratulations" or "win" or "won" as part of its text (so we can find it easily when grading!). It should appear as part of the page, as content that you inject using the DOM; not as an `alert` or pop-up window. For example, you could place an empty `div` in your HTML file and inject text into it when the game is won. The congratulatory message should have some kind of styling to make it stand out on the page, such as colors or font changes.

Note that the page initially appears with the page in a solved state, but this does not count as having solved the puzzle, and your congratulatory message should not be showing initially. But, for example, if the user moves a single tile from the sorted board out of place and then moves it back, this counts as "solving" the board.

If the "You won!" message is showing and the user makes a move or Shuffles, causing the puzzle to no longer be in its solved state, the message should disappear. The message should show only when the puzzle is in a solved state.

When the user presses the Shuffle button, it is possible (though not very likely) that the shuffle will result in a board that is back in the solved state. The behavior in this case is unspecified. It would probably be better to not count that as having "solved" the puzzle, but we will not test you on that case. But it is also possible that your shuffle code, while it is making its many random moves, will temporarily put the board into an intermediate state where it is back into its proper solved order. Your code should *not* count this as having "solved" the board.

To implement this feature, you will need some way of testing whether the board is currently in its "solved" state after moves. We suggest that you implement this test by examining each puzzle piece's current x/y position relative to what it should be when the board is solved. For example, the top/left corner of the "1" square is x=0, y=0, the "2" square is x=0, y=100, and so on. If all squares are in their expected proper positions, the board is solved.

It can be hard to **test this feature** because it takes a while to actually solve a shuffled board. You can try temporarily reducing the amount of shuffling done by your Shuffle algorithm, such as reducing the loop passes from 1000 to 10, which makes the board closer to its initial state and easier to solve. If you want to try actually solving the puzzle, first get the 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.

Extra Feature #2. Ability to Slide Multiple Squares at Once

Make it so that if you click any square in the empty square's row or column, even ones more than one spot away from the empty square, all squares between it and the empty square slide over. (Much more pleasant to play!) If you do this extra feature, make it so that all movable squares (including ones several rows or columns away from the empty square) highlight on hover as described before.

Extra Feature #3. Animations and/or Transitions

Instead of each tile immediately appearing in its new position, make them animate. You can do any sort of animation or other styling, as long as the game ends up in the proper state after a reasonable amount of time.

It can be tricky to implement animation yourself. You could do animation using a timer such as with the `setInterval` function. Or you could use a CSS3 transformation. One source of bugs is when the user tries to click pieces quickly before the previous move is done animating. Your puzzle should not have buggy behavior or errors if the user tries to do this. It's fine to disable/ignore clicks on puzzle squares that occur during animation, so that it is not possible to make another click until the previous click is done animating.

Extra Feature #4. Game Timer

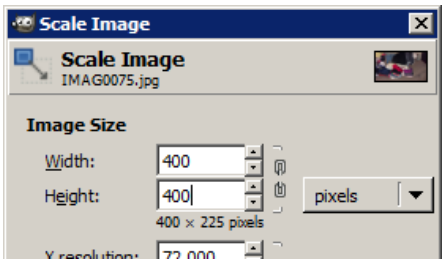
Keep track of the game time elapsed in seconds and the total number of moves, and when the puzzle has been solved, display them along with the best time and fewest total moves seen so far.

You can implement this feature using a timer and the `setInterval` function in JavaScript. It's fine to just show the total seconds needed, or if you want to split it into minutes and seconds, that is fine, too.

Extra Feature #5. Multiple Backgrounds

In the standard assignment you provide a single background image. If you do this feature, **find three or more images** of your own choosing and allow the user to choose between them. Store your images on the Webster server in the same folder as your HTML/JS files and link to the images using absolute URLs. You should still have **background.jpg** as one of your images (the default selection) and still turn that in. You will not turn in the other images, but you will put them on Webster so that your page can use them.

If you implement this feature, use the JS DOM to inject a drop-down **select box** underneath the main puzzle that allows the user to choose different images to appear behind the tiles. Initially **background.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.



You can find images online using a service such as Google Image Search. The images you find may not be the right size to use on this page; they should be 400x400 pixels. If you need to resize an image, you can use an image editor such as Photoshop, or download a free image editor such as The Gimp (**www.gimp.org**). To resize an image in The Gimp, click Image, Scale Image... and enter a Width and Height of 400 pixels. (You may need to click the "chain link" icon right of the width/height boxes to allow you to set the width and height individually, since by default it tries to maintain the image's original *w*/*h* aspect ratio on a resize operation).

Extra Feature #6. Different Puzzle Sizes

Place a control on the board to allow the game to be broken apart in other sizes besides 4x4, such as 3x3 or 6x6. It doesn't need to be possible to change the board size in the middle of a game; changing the size should reset the board back to an initial "solved" state for that board size. You can either use a drop-down box of available board sizes, or allow the user to type in their own board size.

The overall puzzle size should stay at 400x400. This means that each square's size must change if you change the number of rows and columns; for example, if it is a 5x5 game, each square is 80x80 rather than 100x100. Certain sizes don't evenly divide into 400, such as 6x6. In such a case, just choose the nearest integer to the proper size.