# WEB PROGRAMMING
# TEAM PROJECT 2: FIFTEEN PUZZLE
# DUE DATE 07/23/2023 @ 11:59 PM

### EACH TEAM WILL PROVIDE A SECOND PAGE THAT DOCUMENTS TEAM MEMBERS, CREATE AND TEST CASE

FAQS:

Group Requirements: Same group members as assigned

Graduates: Same group members as assigned

## Today's Agenda

1. **Please reference the group assignment -posted (for Modification contact TA)**
2. **Choose a leader - liaison to the instructor**
3. **Brainstorm ideas**
4. **Plan how you will collaborate/communicate**
5. **Choose someone to "integrate" the parts done by the team members**
6. **Decide the responsibilities for each team member. e.g.**

## SUBMISSION REQUIREMENT

**Create a YouTube Video "Name of the task Project 03_TeamName "**

- ✓ Provide a description: one- minute introductions description of your project and group members
- ✓ This video must range for 12 - 15 minutes in presenting which will include the demo run and code snippets of your project.
- ✓ You must be submitted on I-College by the due date and time. Late homework will not be graded, as stated in the course grading policy.
  No email or hard copies of homework will be accepted by me (submit to drop-box).
- ✓ Every team member must participate in this video if a member is missing that member will Earn a grade of zero.
- ✓ Screen record your code + you can show you via video clips during the life cycle development failure will result in deduction of points)
- ✓ Record as normal and make sure the voice is clear.
- ✓ Create a channel at YouTube and name it as your group name
- ✓ Once ready, upload the video to your channel.
- ✓ Include the link to this channel with your submission and you will incorporate the video for the core of your PowerPoint presentation.
- ✓ Log into (ICollege), select the class drop box folder, please be sure to select the correct folder for the given assignment and upload the file there.

**Please also attach a pdf document with the TEST CASE for your project)**

# Requirements

**You shall choose one team member as leader for purposes of coordinating the project and reporting to the instructor.**

**Each team shall make a presentation lasting five minutes or less in which you present your completed project. At the beginning of the**

**Presentation the leader shall present to the instructor a single sheet of paper, which states the following:**

- **Leader's Name**
- **Project Name**
- **Description: a one-sentence description of your project**
- **Supply Test Case for project especially if you have an extra feature**
- **Make sure you have code snippets in you PowerPoint presentation of you will lose points.**

**Team Members:** Provide a list of your team members (last and first names) and their project responsibilities.

**Key for the design see the following: >> Scrum Framework**

Please provide a 1ppt- slide brief assessment how this framework benefited you on the project
All members are Designers - work with the User to determine the program requirements.

Implement a sketch for the User Interface. Design the program - determine the classes, fields, methods, objects, etc. Write pseudo code for all methods.

**All are Programmers -** Create the interface and write all the code.

**All are Testers** – (if applicable) Develop a test plan including test procedures, test data, method of tracking and reporting bugs, and assigning priorities to bugs. May also help write code to fix bugs. Put together the PowerPoint slide show using input from the other team members.

**Decide on a schedule;** estimate hours for each phase; determine when, where, and how you will communicate and coordinate your work. Part of class time will be available for teamwork and I will be available to help you. Email is a good way to communicate.

1. PowerPoint slide show MUST include the following:
2. User - statement of problem, and general requirements (inputs, outputs, etc.)
3. Design - Overview of solution, key design features, user interface, UML class diagrams, pseudo code etc.
4. Testing - (if applicable) how tested (e.g., test plan, data used, tracking and reporting bugs, bugs fixed/not fixed, etc.)

**Presenters**

Choose one or more presenters. You may choose to have one person do the entire presentation. Or perhaps one will do the slide show, and a second team member will demo the program. Or, each team member may wish to present his own work.

**Grading**

Meet the requirements (see the Requirements section above), you will receive credit however full credit will be based on group that used creativity and whom went above and beyond my requirements. Your team must turn in the paper as specified above and do the presentations in order to get credit. It will not be sufficient to simply turn in the files to I-College and have it posted on codd. Post the URL or Project to your student account.

**Please use either the leader's/Team name or the project name as the folder name (optional).**

**(Please submit your link of project 3 to iCollege) …NO Link NO GRADE**

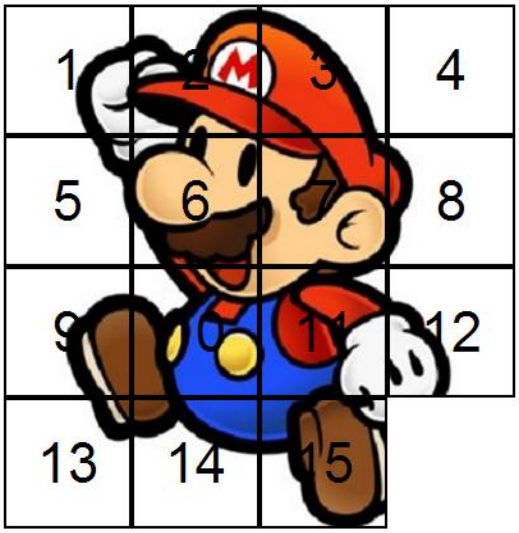
## Suggestions Summary for development and presentation

- You may organize your team any way you like. One way is user, designer, coder/programmer, and tester. Another way is an architect / chief programmer with a team of programmers each of whom works on one part of the program.
- The presentation could be structured as follows:
    a. a PowerPoint slide show to introduce the problem
    b. a demo run of the program
    c. a display and explanation of some of the code
    d. a FAQ Page if desired.

**This Project is about JavaScript's and event handling**. You must match in appearance and behavior the following web page (between but not including the thick black lines):

# Fifteen Puzzle

The goal of the fifteen puzzle is to un-jumble its fifteen squares by repeatedly making moves that slide squares into the empty space. How quickly can you solve it?




Shuffle

American puzzle author and mathematician Sam Loyd is often falsely credited with creating the puzzle; indeed, Loyd claimed from 1891 until his death in 1911 that he invented it. The puzzle was actually created around 1874 by Noyes Palmer Chapman, a postmaster in Canastota, New York.

W3C XHTML 1.1 W3C CSS 190M JS Lint

## Learning Objectives

I. Carefully reading a specification.
II. Choosing appropriate CSS rules to style a page.
III. Reducing redundancy in your CSS while producing expected output.
IV. Retrieving information from a user using events on the Document Object Model (DOM) objects within your JS.
V. Modifying your web page using JS and DOM objects.
VI. Producing quality readable and maintainable code including unobtrusive JavaScript that is modular and kept separate from the HTML/CSS.
VII. Responding to the user's mouse events on DOM objects
VIII. Manipulating the DOM tree using JS functions.
IX. Manipulating DOM object styles through JS.

## Background Information:

The "Fifteen puzzle (Explanation 1)" (Explanation 2) (more generally 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 JavaScript code for a page fifteen.html that plays the Fifteen Puzzle. You create 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. Please see file names below in which you should have on the codd server:

- fifteen.js, the JavaScript code for your web page
- background.jpg, your background image, suitable for a puzzle of size 400x400px

You CAN create your own XHTML or CSS code in this assignment or instead, use the CSS we provide. Write JavaScript code that interacts with the page using the DOM. In order to create its appearance, write appropriate DOM code to change styles of onscreen elements by setting classes, IDs, and/or style properties on them.

**Tips for solving a fifteen puzzle:** 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:

In the center of the page is a set of tiles representing the playable Fifteen Puzzle game. Each tile is 100x100 pixels in total size, including a 2px black border around all four of its sides. (This leaves 96x96 pixels visible area inside the tile.) Each tile displays a number from 1 to 15 in 32pt text using the default sans-serif font available on the system. When the page loads, initially the tiles are arranged in their correct order, top to bottom, left to right, with the missing square in the bottom-right. The tiles also should display a chunk of the image background.jpg, assumed to be located in the same folder as your page. Useful CSS code. < (GSU VPN maybe needed to Access)

Please ensure that your background image appears on the 15 puzzle pieces. By adjusting the background-position on each **div** which is a container, you'll be able to show a different piece of the background on each of the 15 puzzle pieces. One confusing thing about the background-position property is that the x/y offsets shift the background behind an element, not the element itself; this means that 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 background image, you would set its background-position to -300px 0px.

Centered underneath the puzzle tiles is a Shuffle button that can be clicked to randomly rearrange the tiles of the puzzle. The last content on the page is a right-aligned paragraph containing two links to the W3C validators and (optional) JSLint. These are the same images and links as used in the previous assignments. The images should not have borders.

All other style elements on the page are subject to the preference of the web browser. The screenshots in this document is a demo.

## Behavior Details:

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.

Whenever the mouse cursor hovers over a square that can be moved (one that neighbors the blank square), its appearance should change. Its border color should change from black to red. Its text should become underlined and should become drawn in a green color of#006600. (This can be done by creating CSS class call it **movablepiece** that would represent these styles.) Once the cursor is no longer hovering over the square, its appearance should revert to its original state. Hovering the mouse over a square that cannot be moved has no effect.

When the Shuffle button is clicked, the tiles of the puzzle are randomized. The tiles must be rearranged into a solvable state. Please note that many states of the puzzle are not solvable; for example, it has been proven that the puzzle **cannot be solved** if you switch only its 14 and 15 tiles. We suggest that you generate a random solvable puzzle state by repeatedly choosing a random neighbor of the missing tile and sliding it onto the missing tile's space. A few hundred such random movements should produce a shuffled board. Your shuffle algorithm should be relatively efficient; if it takes more than a second to run or performs a large number of unnecessary tests and calls, you may lose points.

The game is not required to take any particular action when the puzzle has been won, that is, when the tiles have been rearranged into the correct order (See Extra feature)

## More Basic Requirements - 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. The mouse cursor also 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) and 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 following is a sample listing of the 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 | |

1. Centered under the puzzle tiles is a Shuffle button that can be clicked to randomly rearrange the tiles of the puzzle. See the "Shuffle Algorithm" section of this spec for more details about implementing the shuffle behavior.
2. The HTML file you are given also does not have the citation information displayed on the screen, so you will need to create and add these DOM elements to your page programmatically within your .js file.

All other style elements on the page are subject to the preference of the web browser. While the screenshot in this document was rendered on Mac OSX, default cursive fonts vary quite a bit from system to system, so it is possible that a screenshot of your page will look somewhat different than the expected output.

## Extra Features:

In addition to the previous requirements: You must specify the extra feature you used during your presentation failure to do so will result in points deductions on an average of 10 points each one that is missing Requirements.

- **End-of-game notification**: Provide some sort of visual notification when the game has been won; that is, when the tiles have been rearranged into their original order. An alert is not sufficient; you should modify the appearance of the page. You may display an image(s) if you like, but there is no way to turn them in, so put them on your web space and use full path URLs when linking to them.
- **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.
- **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 you like, as long as the game ends up in the proper state after a reasonable amount of time.
- **Game time with some music file**: 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/moves seen so far. The Music files should be a sound to create some adrenalin
- **Multiple backgrounds**: Provide several background images (at least 4) to choose from. The game should choose a random background on startup, and should have a UI (such as a select box) by which the player can choose a different image while playing. Host your additional backgrounds on the web server and link to them using full path URLs.
- **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, 6x6, 8x8, 10x10 etc.
- **Extra Animation:** Create some extra animation congratulating the user or add any other optional behavior to handle this event when the puzzle has been won.
- **Cheat button**: This will solve and show the shuffle that was created **not a static page** to display the titles in order.

- **Graduate Students Only – YOU MUST CREATE AN EXTRA FEATURES OF CHOICE ALONG WITH ANY 5 OF THE FEATUES LISTES**

- **UNDERGRAD – MUST IMPLEMENT ANY 4 OF THE EXTRA FEATURES**

*At the top or bottom of your html file, you can display a comment/link stating which extra feature(s) you have completed.* If you implement more than one, comment which one you want us to grade (the others will be ignored). Regardless of how many additions you implement, the main behavior and appearance should still work as specified.

## Shuffle Algorithm

Centered under the puzzle tiles is a Shuffle button that can be clicked to randomly rearrange the tiles of the puzzle. When the Shuffle button is clicked, the puzzle tiles are rearranged into a random ordering so that the user has a challenging puzzle to solve.

The tiles must be rearranged into a solvable state. Some puzzle states are not solvable; for example, the puzzle cannot be solved if you swap only its 14 and 15 tiles. Therefore, your algorithm for shuffling cannot simply move each tile to a completely random location. It must generate only solvable puzzle states if you want full credit.

We suggest that you generate a random valid solvable puzzle state by repeatedly choosing a random neighbor of the missing tile and sliding it onto the missing tile's space. Roughly 1000 such random movements should produce a well-shuffled board. Here is a rough pseudo-code of the algorithm we suggest for shuffling.

```
for (~1000 times): neighbors = [] for each neighbor n that is directly up, down, left, right from
    empty square:
        if n exists and is movable: neighbors.push(n)
    randomly choose an element i from neighbors move
    neighbors[i] to the location of the empty square
```

Notice that on each pass of our algorithm, it is guaranteed that one square will move. Some students write an algorithm that randomly chooses any one of the 15 squares and tries to move it; but this is a poor way to shuffle because many of the 15 squares are not neighbors of the empty square. Therefore, the loop must repeat many more times in order to shuffle the elements effectively, making it slow and causing the page to lag. This is not acceptable.

**A few requirements about the shuffle algorithm:**

- You are not required to follow exactly the algorithm above, but if you don't, your algorithm must exhibit the desired properties described in this section to receive full credit.
- Your algorithm should be efficient; if it takes more than 1 second to run or performs a large number of tests and calls unnecessarily, you may lose points.
- For full credit, your shuffle should have a good random distribution and thoroughly rearrange the tiles and the blank position.
- Your shuffle algorithm will need to incorporate randomness. You can generate a random integer from 0 to K, which is helpful to randomly choose between K choices, by writing, parseInt(Math.random() * K). Some development hints about the shuffle algorithm:
- I suggest first implementing code to perform a single random move; that is, when Shuffle is clicked, randomly pick one square near the empty square and move it. Get it to do this once then work on doing it many times in a loop.
- In your shuffle algorithm, or elsewhere in your program, you may want access to the DOM object for a square at a particular row/column or x/y position. We suggest you write a function that accepts a row/column as parameters and returns the DOM object for the corresponding square. It may be helpful to you to give an id to each square, such as "square_2_3" for the square in row 2, column 3, so that you can more easily access the squares later in your JavaScript code. (If any square moves, you will need to update its id value to match its new location.) Use these ids appropriately. Don't break apart the string "square_2_3" to extract the 2 or 3. Instead, use the ids to access the corresponding DOM object: e.g., for the square at row 2, column 3, make an id string of "square_2_3".

## Development Strategy and Hints:

Past students claim that this program is hard! However, I suggest the following development strategy:

- 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 next to the empty square.
- Write the code to determine whether a given square can move or not (whether it neighbors the empty square). Use *this* to implement the highlighting style that occurs when the user's mouse hovers over tiles that can be moved. This will require you to keep track of where the empty square is at all times.
- Write the shuffling algorithm. We suggest first implementing the code to perform a single random move; that is, randomly picking one square that neighbors the empty square and moving that square to the empty spot. Get it to do this one time when Shuffle is clicked, then work on doing it many times.

## Here are some hints to help you solve particular aspect of the assignment:

- 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.
- You can 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 must exactly match what it would be 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.
- I suggest that you do not explicitly make a div to represent the empty square of the puzzle. Keep track of where it is, such as by row/column or by x/y position, but don't create an actual DOM 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 will likely make it more difficult to keep it up to date as the squares move. Furthermore, it is easy to write 2-D array code that ends up being poor style because of how difficult it is to maintain the proper state of the arrays.
- Many students have redundant code because they don't create helper functions. You should write functions for common operations, such as moving a particular square, or for determining whether a given square currently can be moved. The ***this*** keyword can be helpful for reducing redundancy.

## Internal Requirements

I. Your CSS and JS should also maintain good code quality by following the guide posted on the class web site. We also expect you to implement relevant feedback from previous assignments.

II. Your.js file must be in the module pattern and run in strict mode by putting "use strict"; within your file.

III. Your JavaScript code and CSS files should have adequate commenting. The top of both files should have a descriptive header describing the assignment. You are to use the JSDoc commenting style for each function and module-global variable in your JS file (see link >> here). Additionally, complex sections of code should be documented. You may use inline commenting within JS functions.

IV. Format your code similarly to the examples from class. Properly use whitespace and indentation. Use good variable and method names. Lines of code should be fewer than 100 characters long.

V. Variables should be localized as much as possible. Minimize the use of module-global variables. Do not ever store DOM element objects, such as those returned by the document.getElementById or or document.querySelectorAll functions, as global variables. For reference, my solution has only four module-global variables: the empty square's row and column (initially both are set to 3), the

number of rows/cols in the puzzle (4), and the pixel width/height of each tile (100). The last two are essentially constants.

VI. If a particular literal value is used frequently, declare it as a module-global "constant" IN_UPPER_CASE and use the constant in your code.

VII. You should make an extra effort to minimize redundant JavaScript code. Capture common operations as functions to keep code size and complexity from growing. You can reduce your code size by using the ***this*** keyword in your event handlers.

VIII. Separate content (HTML), presentation (CSS), and behavior (JS). Your JS code should use styles and classes from the CSS rather than manually setting each style property in the JS. For example, rather than setting the .style of a DOM object, instead, give it a className and put the styles for that class in your CSS file. Note: Style properties related to x/y positions of tiles and their backgrounds are impractical to put in the CSS file, so those can be in your JS code.

## Implementation and Grading:

**Once again, submit your assignment to iCollege and post the link on each team member course web site.**

Follow reasonable stylistic guidelines. For example, utilize parameters and return values properly, correctly use indentation and spacing, and place a comment header on top of the file, atop every function explaining that function's behavior, and on complex code sections.

Global variables are allowed, but it is not appropriate to declare lots of them; values should be local as much as possible. If a particular value is used frequently throughout your code, declare it as a global "constant" variable named IN_UPPER_CASE and use the constant throughout your code.

You should only use material discussed in the textbook or lectures unless given permission by the instructor.

You may be able to find Fifteen Puzzle games written in JavaScript on the internet.

**Please do not look at any such games. Using or borrowing ideas from such code is a violation of this course academic integrity policy. We have done searches to find several such games and will include them in our similarity detection process.**

Remember to place the link solution to this assignment online on each team member's web page.

**Also, All Team member can use Google Plus or GIT HUB to collaborate. If you'd like me to see the communication in your circle, send me your Google plus info and I'll add you to the circle.**

**Important:** Your page must pass the validator at http://validator.w3.org.