

# Tic-Tac-Toe (Naughts and Crosses)

Welcome to the Tic-Tac-Toe project! In this multi-step project, you will create a well-designed HTML+CSS+JS browser-based application that will allow you to play tic-tac-toe. It will combine all of the concepts that you learned in browser basics, element selection and handling, storage, and JSON.

Each step has three parts:

- The requirements for the next step of the application
- A reading that will help you design it
- A video walk-through of a "good" solution, where "good" means clear, concise, and maintainable

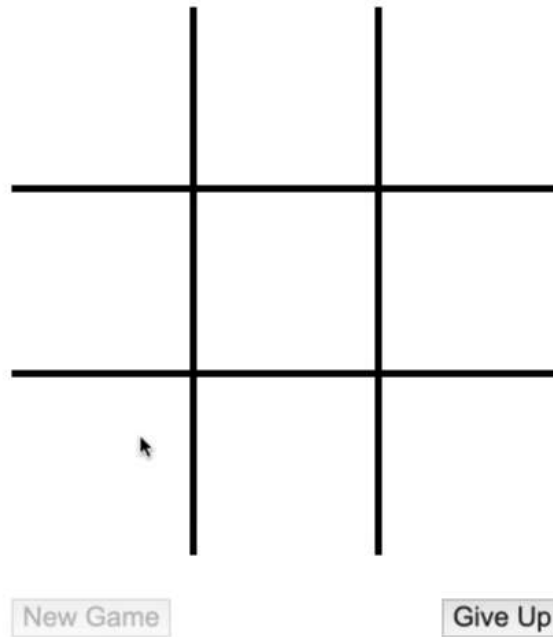
The reason they're set up this way is so that you can attempt to do the requirements yourself. If you get stuck on how to do it, then there is a reading available for you to help you take the next step in code. You can watch the video if you get really stuck or to compare your solution to the way that we implemented it.

*Remember: working software with **nicely-formatted source code** is the best software.*

Please fork and clone the repository found at <https://github.com/appacademy-starters/dom-api-tic-tac-toe>. In that repository, you will find the following files:

- **index.html**: The HTML file to which you'll make small modifications that will make the UI work
- **site.css**: The CSS file that contains all of the styles for your UI
- **tic-tac-toe.js**: The JavaScript file you'll modify to add behavior to your game

Here's what it will look like when you're done.



## The rules of tic-tac-toe

---

In case you don't know the rules of tic-tac-toe, here are some links for you to investigate.

- Solitaire Classic's [How to play Tic Tac Toe](#) video
- [Wikipedia's tic-tac-toe article](#)

## Interesting sections of the index.html

---

You'll find three sections of interest in the **index.html**.

Near the top of the body of content, you'll see an `h1` with an `id=game-status`. You'll use that element to indicate which player won the game (or if there is a tie).

In the middle of the body of content, you'll see the following HTML that represents the tic-tac-toe board. The outer `<div>` makes the black background that you see as the vertical and horizontal lines of the board. The nine children `<div>` elements are the nine squares of the board. Those are the HTML elements that players will click on to place their symbol there.

```
<div id="tic-tac-toe-board">
  <div id="square-0" class="square row-1 col-1"></div>
  <div id="square-1" class="square row-1 col-2"></div>
  <div id="square-2" class="square row-1 col-3"></div>
  <div id="square-3" class="square row-2 col-1"></div>
  <div id="square-4" class="square row-2 col-2"></div>
  <div id="square-5" class="square row-2 col-3"></div>
  <div id="square-6" class="square row-3 col-1"></div>
  <div id="square-7" class="square row-3 col-2"></div>
  <div id="square-8" class="square row-3 col-3"></div>
</div>
```

At the bottom of the body of content are the action buttons. You'll subscribe to each of their `click` events to make the game work

```
<div class="actions">
  <button>New Game</button>
  <div class="spacer"></div>
  <button>Give Up</button>
</div>
```

## Requirement 2: Tracking Grid Clicks

In this requirement, you will now track the clicks of the players and fill the grid with the appropriate symbols.

- For the "X" player, use the image found at <https://assets.aonline.io/Module-DOM-API/formative-project-tic-tac-toe/player-x.svg>.
- For the "O" player, use the image found at <https://assets.aonline.io/Module-DOM-API/formative-project-tic-tac-toe/player-o.svg>.
- When the player clicks an empty square, then it is filled with that player's symbol.
- When the player clicks a square that already contains a symbol, the game does nothing.
- The first click results in an "X". After that, the symbols "O" and "X" alternate with each click per the rules of tic-tac-toe.

During development, you can just refresh the browser to clear the board.

To make this game work, you'll eventually get around to checking if one of the players won or if there is a tie. Plan ahead for how you track the grid clicks because you'll need to know the "value" of each square to calculate wins.

## Hints 2: Tracking Grid Clicks

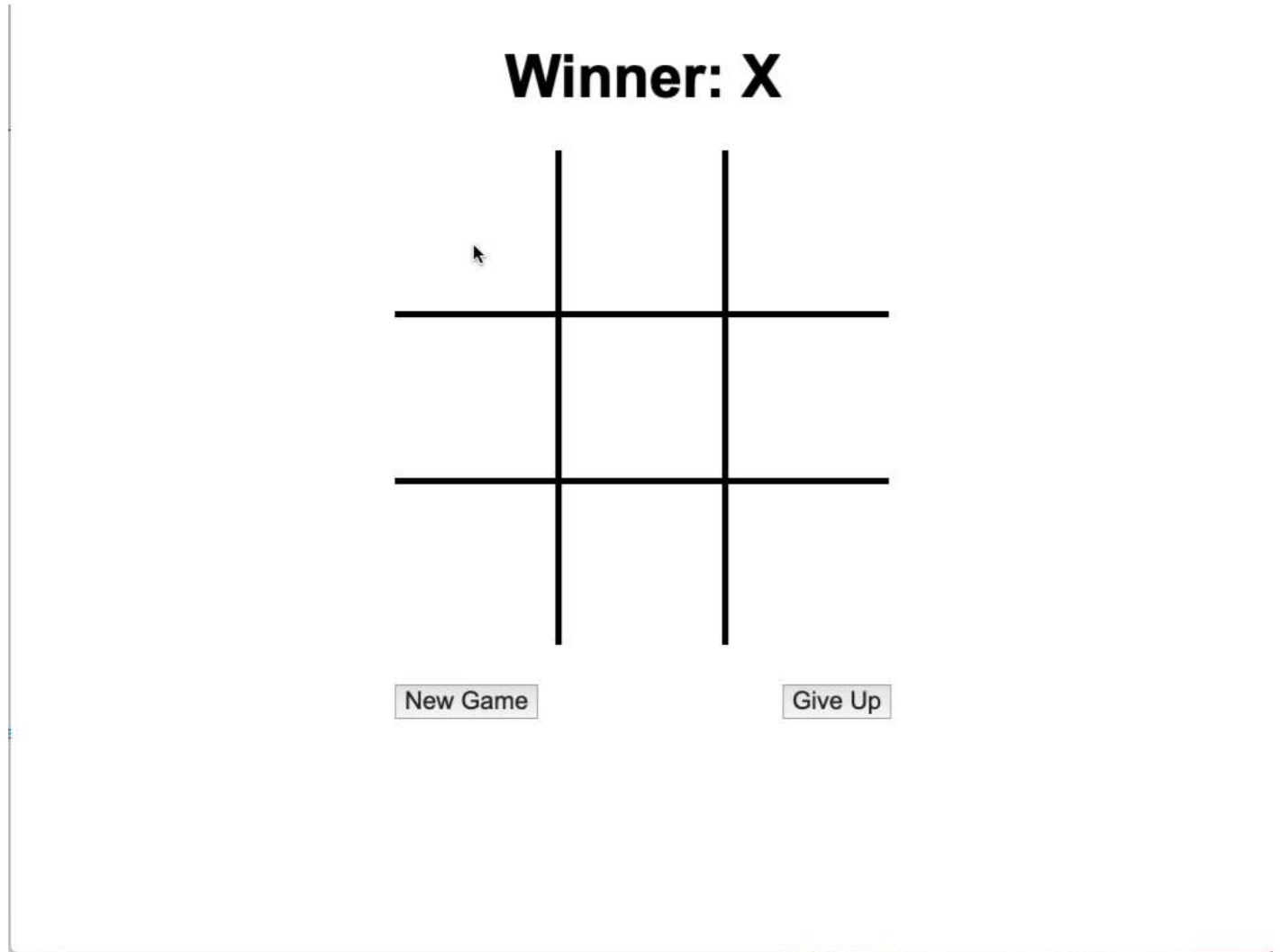
To track and store the grid clicks, you can perform the following steps in the JavaScript file:

1. Declare a variable named `currentPlayerSymbol` and set it to "x".
2. Declare a variable named `squareValues` and initialize it to an array with nine empty string entries.
3. Add an event listener to the `window` object for the `DOMContentLoaded` event.
4. In the event handler for the `DOMContentLoaded` event, add a click event handler to the tic-tac-toe grid `<div>`.
5. In the click event handler, inspect the target of the event. Ignore the event if its `id` does not begin with "square-".
6. If the event target's `id` attribute does begin with "square-", then parse the number after "square-" using the `Number.parseInt` method.
7. If the value in the `squareValues` array for the index of the parsed number is not an empty string, then ignore the event.
8. There must not be a play in that square, so:
  - Programmatically create an `img` element, set its source to appropriate value from the requirements by using the value in `currentPlayerSymbol`, and append the `img` element to the event target.
  - Store the value of `currentPlayerSymbol` in the corresponding slot in the `squareValues` array.
  - If the `currentPlayerSymbol` is "x", then set it to "o". Otherwise, set `currentPlayerSymbol` to "x".

**Why do these hints suggest creating an array in which to track the symbols?** In the world of good programming, the *state* of the application is kept separate from the *visualization* of the program's state. What that practically means is that the Tic Tac Toe game really only cares about the "x" values and the "o" values, not the fact that there are fancy SVG images in the UI. For it to figure out the winner or tie, it cares about the *data* that describes the game board at any given point, not the HTML elements that the players

see. If the program knows the state of the board at any given point, then it can make those determinations as well as restoring the game board from that point in the future.

When you finish, you should have something similar to the following.



# Requirement 3: Determining Game Status

In this requirement, you will now determine if a game is won or in a tie state. To do this, you will need to check the following:

- If a player has any three in a row, then that player wins.
- If a player has any three in a column, then that player wins.
- If a player has either of the diagonals, then that player wins.
- If there is no win *and* all squares have a player symbol in there, then the game is a tie.
- When the game begins, the header at the top should have no text in it.
- When a player wins the game, then the following happens:
  - The header at the top should read "Winner: X" or "Winner: Y" depending on which player won.
  - Empty squares in the grid no longer react to clicks.
- When the game goes into a tied state, the header at the top should read "Winner: None".

During development, you can just refresh the browser to clear the board.

# Hints 3: Determining Game Status

To do this, you can write a function that checks the values of each of the squares that have been played in. If any three in a row have the same value, then the player associated with that symbol is the winner. If there is no winner and all of the spaces have a symbol, then it is a tie. That's the general idea.

Please go back and read the Hints for Requirement 2 to understand the following variables that have been declared in the game.

- `currentPlayerSymbol`: the symbol of the player that has the next click
- `squareValues`: an array that contains the symbols for each of the squares

With those variables declared (or ones like them), add code that will perform the following functionality.

In the JavaScript file:

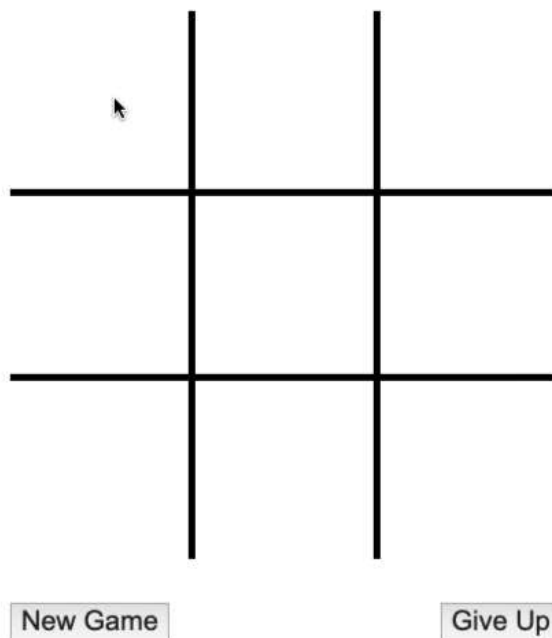
- Create a top-level variable named `gameStatus` and set it to the empty string.
- Create a top-level function named `checkGameStatus` that will loop through all of the rows and columns, and check the diagonals to determine if there is a winner. In it, perform these checks
  - If there is a winner, set the `gameStatus` value to the uppercase value of the value that is winning symbol.
  - If there is no winner, then check if all of the squares are full. If they are, then set the `gameStatus` equal to "None".
  - If the `gameStatus` is not an empty string, set `document.getElementById('game-status').innerHTML` to the concatenated value of "Winner: " and the value of `gameStatus`. Use the `id` value of `game-status` to get a reference to that element.
- Add a check at the beginning of the click handler for the grid that prevents the rest of the event handler from running if the `gameStatus` value is not an



empty string. (This is as simple as adding an `if` statement that tests for it and has a `return` statement inside the curly braces.)

```
if (gameStatus !== '') return; // This just stops
                                // the rest of the function
                                // from running.
```

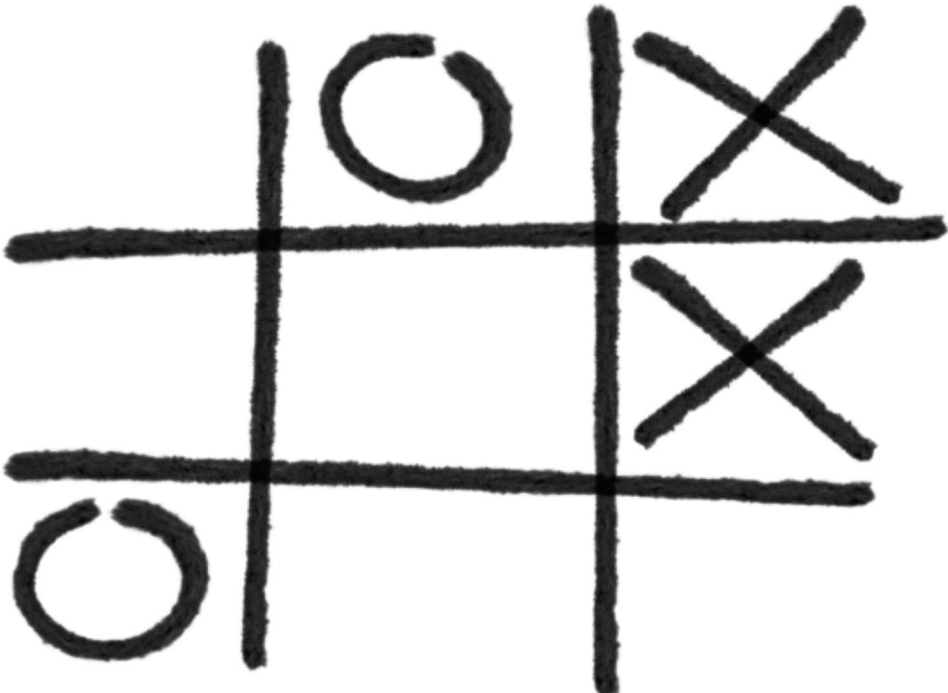
- At the end of the click handler for the grid to call the `checkGameStatus` function.



## How to find a winning game

Consider using a one-dimensional array with nine entries to store your representation of the board. Then, you can map each of the nine entries to a square on the board kind of like what you see in the following picture where the 0-index entry in the array is the top-left square of the board, the 1-index entry in the array is the top-middle square, the 2-index entry is the top-right, the 3-index entry is the middle-left, and so on.

	' O '	' X '			' X '	' O '		
0	1	2	3	4	5	6	7	8



If that's the model that you have, then you can check the rows of the board for a win by comparing the first three entries of the array with one another, the second three entries with one another, and the third three entries with one another. Here's the breakdown of the rows of the tic-tac-toe-board mapped to the entries in the array.

Top row			Middle row			Bottom row		
	'o'	'x'			'x'	'o'		
0	1	2	3	4	5	6	7	8

Assume the name of your array that holds the board values is named `board`. You can determine if the first row has a winning combination of entries by doing the following comparison:

```
if (board[0] === board[1] && board[1] === board[2] && board[2] !== '') {
    // do something because a player won on the first row!
}
```

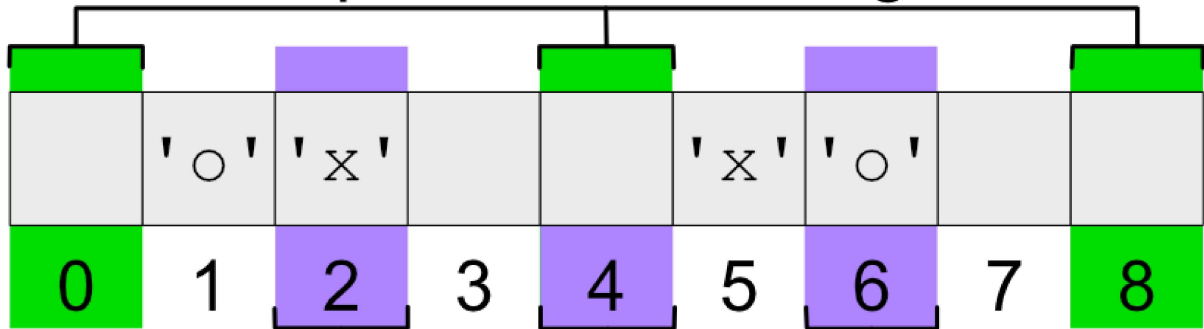
You can just do the same for the other two rows, too.

To find the columns in the array, you can refer to the following image.

Left column			Right column					
	'o'	'x'		'x'	'o'			
0	1	2	3	4	5	6	7	8
Middle column								

Finally, to find the diagonals, you need to map squares 0, 4, and 9 for the top-left to bottom right and the 2, 4, 6 squares for the top-right to the bottom-left like in the following picture.

Top-left to bottom-right



Bottom-left to top-right

# Requirement 4: Creating A New Game

In this requirement, you will now make the "New Game" button work. To do this, you will need to perform the following:

- When the game status is not "won" or "tied", then the "New Game" button is disabled.
- When the game status is "won" or "tied", then the "New Game" button is enabled.
- When a player clicks the "New Game" button, then it
  - clears the game status,
  - clears the header,
  - clears the board, and
  - makes it so the next click of the tic-tac-toe board is an "X"
  - (disables the "New Game" button)

During development, you can just refresh the browser to clear the board.

# Requirement 5: Giving Up

In this requirement, you will now make the "Give Up" button work. To do this, you will need to perform the following:

- When a player clicks the "Give Up" button:
  - Set the status of the game as "won" by the "other" player. That is, if "X" is the current player, when that player clicks the "Give Up" button, then "O" wins the game.
  - Show the winner status as won by the "other" player.
  - Disable the "Give Up" button.
  - Enable the "New Game" button.
- When a game is ongoing:
  - Enable the "Give Up" button.

During development, you can just refresh the browser to clear the board.

# Hints 5: Giving Up

A way to tackle this is to subscribe to the click event of the "Give Up" button. Then, in that click event handler, end the game for the "other" player according to your game logic.

If you've reviewed the hints for steps 1, 2, 3, and 4, then you can follow these directions to make it work.

## In the HTML file

---

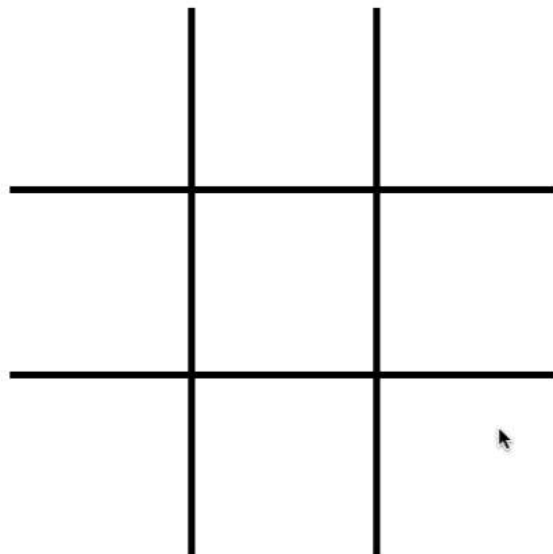
- Add an `id` attribute with a value of `give-up` to the `<button>` with the text "Give Up"

## In the JavaScript file

---

- In the `checkGameStatus` function, in the last check to see if `gameStatus` is not an empty string, add new code in there to set the `give-up` button's `disabled` property to `true`.
- In the `DOMContentLoaded` event handler, add a click event handler to the element with the `give-up` id.
- In the click event handler, set the variable `gameStatus` to the upper-cased version of the "other" player's symbol.
- Also, perform the same logic as found in the last steps of `checkGameStatus` to set the header, enable the "New Game" button, and disable the "Give Up" button.
- In the click event handler for the "new-game" button, add a line in there to set the "give-up" button's `disabled` state to `false`.

When you get done, it should look something like this.



New Game

Give Up



# Requirement 6: Saving Game State

In this requirement, you will make it so that if someone refreshes the screen, the game state will be restored when the page shows back up.

# Hints 6: Saving Game State

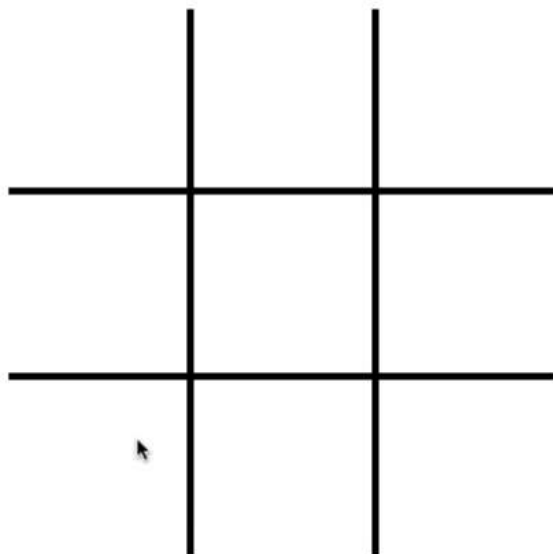
A way to tackle this is to use local storage and JSON serialization to save the state of the game with each change.

You can do that by creating a function that saves the game state to local storage. You should call that function everywhere that the state of the game changes.

You can then create a function that loads the state from the local storage (if it exists) and restores the state of the game and the UI. You can call that function after the DOM content loads.

You should probably create top-level constant variable(s) that hold the key(s) that you will use for local storage so that you declare once and use often. Make sure that the keys that you use are "unique" in that, if another developer comes along and wants to store something for the same domain, that they won't accidentally overwrite your game status. Maybe prefix your key name(s) with 'tic-tac-toe-' or something similar.

When you get done, it should look something like this. The "flashing" in the screen recording is the page refreshing.



New Game

Give Up

# Bonus Requirement: Make The Computer Play

When you click "New Game", randomly assign the computer as Player X or Player O. Then, have the computer play automatically in response to its turn.

For example, if you click "New Game" and the computer becomes Player X, then it will play an "X" on the board. Then, you will play an "O". After you click your square, the computer will automatically play its "X". And, so on.

If the computer is Player O, then it will play after you play your first "X".