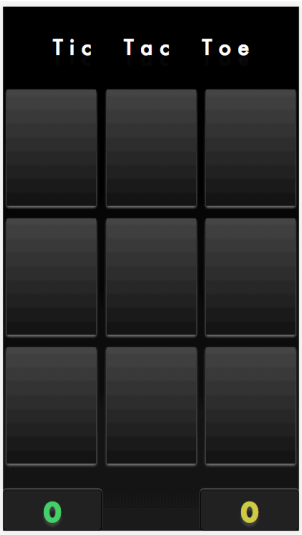
ET712 – Project 1b

Setup content locations: Project_1b, Project/styles and Project/js

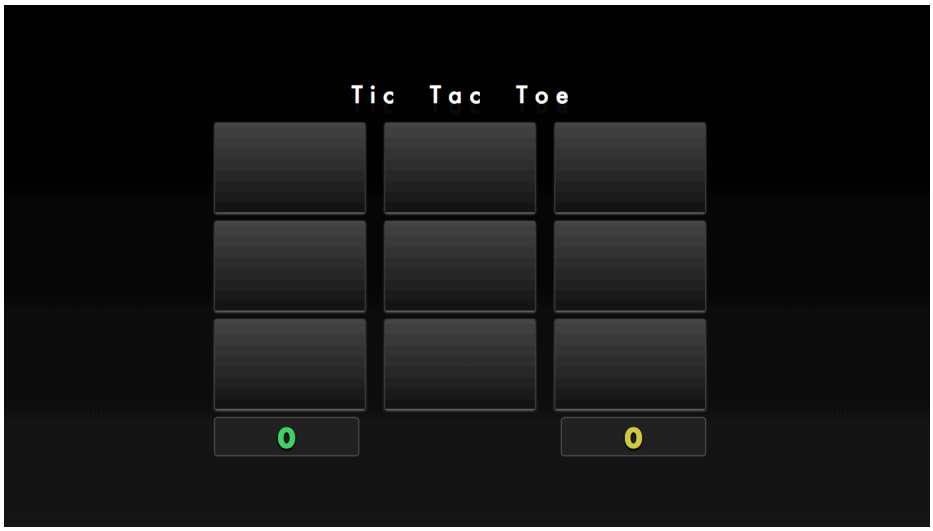
Create a Tic Tac Toe board with the following specifications:

☐ CSS: Mobile first implementation (up to 1024-pixel width).
☐ CSS: Full screen implementation in mobile view.
☐ CSS: Centered implementation in standard view.
☐ CSS: Gradient background for the body of the page.

Mobile



Standard



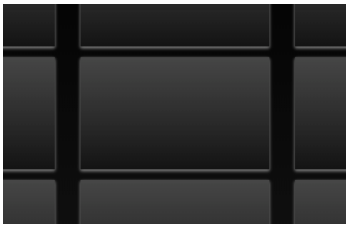
CSS Button Class

☐	add texture:	linear gradient overlay for the background
☐	add depth:	box drop shadow
☐	add space:	transform scaling to reduce button size, perhaps at 90% (room for animations)
☐	add shape:	slightly round the edges of the button

CSS Button Hover Class

☐	add focus:	slightly lighter background
☐	add focus:	transform scaling to a larger button size then regular style, perhaps 98%

Button Style



Hover Style

CSS Alternate Button Classes (one class for <u>each</u> player, one for inactive player)

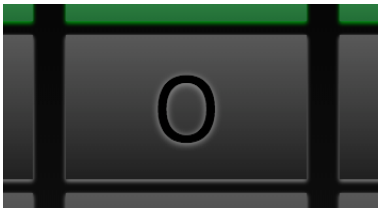
- ☐ Select an X color and an O color
- ☐ Create an X class
- ☐ Create an O class
- ☐ Create an inactive class

- ☐ x class: set the button background to the X color
- ☐ x class: set a dark text drop shadow
- ☐ x class: color matching box drop shadow
- ☐ o class: set the button background to the O color
- ☐ o class: set a dark text drop shadow
- ☐ o class: color matching box drop shadow
- ☐ inactive class: no background (default button background)
- ☐ inactive class: set a light text drop shadow
- ☐ inactive class: color matching box drop shadow

X class

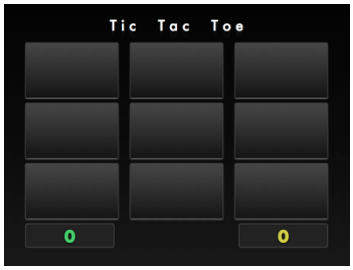
O class

Inactive class

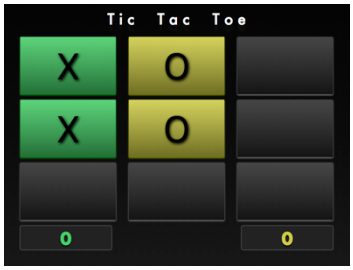


These classes will be used to visualize the various game states:

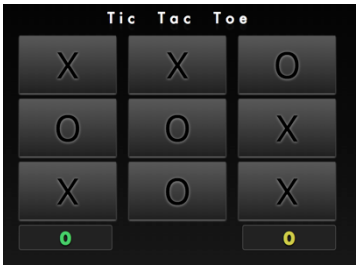
Start



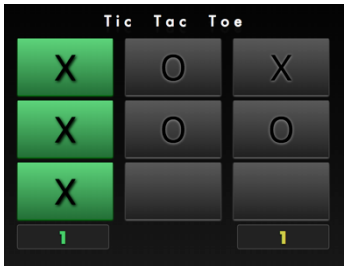
Mid-Game



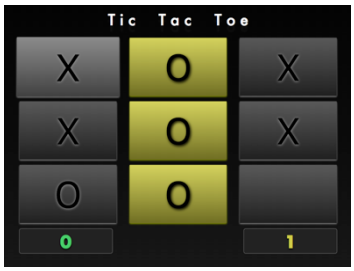
Draw



X Wins



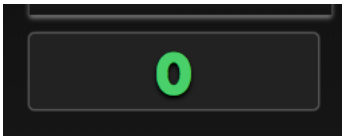
O Wins



CSS Score Classes (one class for each player)

- ☐ x score class: match font color to player color
- ☐ x score class: set a background color
- ☐ x score class: set a border around the box
- ☐ o score class: match font color to player color
- ☐ o score class: set a background color
- ☐ o score class: set a border around the box

X score class



O score class

JS Variables

- ☐ player: used to track which player is active, default is 1
- ☐ numPlays: used to track the number of played boxes in the current game (9 total)
- ☐ winner: used to track who won the game
- ☐ winsPlayer1: used to maintain a count of the number of player 1 wins
- ☐ winsPlayer2: used to maintain a count of the number of player 2 wins

JS Arrays

One array is used to reference the buttons, the other is used to store the button values (to check winner).

- ☐ buttons: array variable used to access each button
- ☐ data: stores the current value of each button as a number (1 for X, -1 for O, 0 for empty)

Both arrays can be visualized as follows (small number is the index, large number is the value):

Buttons Array

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

Data Array

0 1	1 -1	2 1
3 -1	4 1	5 0
6 -1	7 0	8 1

JS Event Listeners

- ☐ button click: listens for a mouse click on any button to call the *clkBox* function
- ☐ header click: listens for a mouse on the Tic Tac Toe header to call the *reset* function

JS Functions

clkBox()

- ☐ step 1: assign a variable to store the *event target* (button that called the function)
- ☐ step 2: assign a variable named *index* to the button index in the buttons class
- ☐ step 3: if the current button content is empty and no one has won (winner == 0):
- ☐ if player is X:
- ☐ set button content to X
- ☐ set button class attribute to *button* and *x class*
- ☐ set player to 2
- ☐ set the data array at *index* to the value 1
- ☐ else (if player is O)
- ☐ set button content to O
- ☐ set button class attribute to *button* and *o class*
- ☐ set player to 1
- ☐ set the data array at *index* to the value -1
- ☐ step 4: call the *checkWinner* function with *index* as a parameter

checkWinner(indexOfCurrentElement)

- ☐ step 1: increment *numPlays* to record that a play has taken place
- ☐ step 2: sum the values for row 1 (indices 0, 1, 2)
- ☐ if sum is equivalent to 3:
- ☐ assign 1 to the *winner* variable
- ☐ loop through the array of *buttons* (where i is the index of each button)
- ☐ if the current i (a button) is <u>not</u> in row 1
- ☐ set button class to *button* and *inactive*
- ☐ else if sum is equivalent to -3:
- ☐ assign 2 to the *winner* variable
- ☐ loop through the array of *buttons* (where i is the index of each button)
- ☐ if the current i (a button) is <u>not</u> in row 1
- ☐ set button class to *button* and *inactive*
- ☐ step 3: repeat step 2 for all rows, all columns and both diagonals (see array visualization)
- ☐ step 4: if the *winner* is player 1:
- ☐ increment the *winsPlayer1* variable
- ☐ update the *x score* box to display the updated value of *winsPlayer1*
- ☐ otherwise if the *winner* is player 2:
- ☐ increment the *winsPlayer2* variable
- ☐ update the *o score* box to display the updated value of *winsPlayer2*
- ☐ otherwise if *numMoves* is equivalent to 9 (a draw)
- ☐ loop through the array of boxes
- ☐ set each button class attribute to *button* and *inactive*

reset()

- ☐ step 1: reset *numMoves*, *player* and *winner* variables to default values
- ☐ step 2: loop through the array of *buttons* (time to reset the board)
- ☐ set each button's content to an empty string
- ☐ set each button's class attribute to *button* (removes x/o class or inactive)
- ☐ set each element in the *data* array equivalent to 0