Tic Tac Toe Walkthrough

Overview:

RULES FOR TIC-TAC-TOE

1. The game is played on a grid that's 3 squares by 3 squares.

2. You are X, your friend (or the computer in this case) is O. Players take turns putting their marks in empty squares.

3. The first player to get 3 of her marks in a row (up, down, across, or diagonally) is the winner.

4. When all 9 squares are full, the game is over. If no player has 3 marks in a row, the game ends in a tie.

HOW CAN I WIN AT TIC-TAC-TOE?

- To beat the computer (or at least tie), you need to make use of a little bit of strategy. Strategy means figuring out what you need to do to win.
- Part of your strategy is trying to figure out how to get three Xs in a row. The other part is trying to figure out how to stop the computer from getting three Os in a row.
- After you put an X in a square, you start looking ahead. Where's the best place for your next X? You look at the empty squares and decide which ones are good choices—which ones might let you make three Xs in a row.
- You also have to watch where the computer puts its O. That could change what you do next. If the computer gets two Os in a row, you have to put your next X in the last empty square in that row, or the computer will win. You are forced to play in a particular square or lose the game.
- If you always pay attention and look ahead, you'll never lose a game of Tic-Tac-Toe. You may not win, but at least you'll tie.

1. TTT: Player Clicks

- 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.

2. TTT: Player Clicks Hints

> 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.

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 .
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:
9. 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".

```
let currentPlayerSymbol = 'x';
let squareValues = ['', '', '', '', '', '', '', '', ''];

window.addEventListener('DOMContentLoaded', () => {
  document
    .getElementById('tic-tac-toe')
    .addEventListener('click', e => {
      const targetId = e.target.id;

      if (!targetId.startsWith('square-')) return;

      const squareIndex = Number.parseInt(targetId[targetId.length - 1]);

      if (squareValues[squareIndex] !== '') return;

      const img = document.createElement('img');
      img.src = `https://assets.aaonline.io/Module-DOM-API/formative-project-tic-tac-toe
      e.target.appendChild(img);

      squareValues[squareIndex] = currentPlayerSymbol;

      if (currentPlayerSymbol === 'x') {
        currentPlayerSymbol = 'o';
      } else {
        currentPlayerSymbol = 'x';
      }
    });
});
```

3. TTT: Game Status

- 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".

4. TTT: Game Status Hints

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

```
let currentPlayerSymbol = 'x';
                //  s0  s1  s2  s3 ...                s8
let squareValues = ['', '', '', '', '', '', '', '', ''];
let gameStatus = '';

function checkGameStatus() {
  // Check rows
  for (let i = 0; i < 9; i += 3) {
    if (squareValues[i] !== ''
        && squareValues[i] === squareValues[i + 1]
        && squareValues[i] === squareValues[i + 2]) {
      gameStatus = squareValues[i];
      break;
    }
  }

  // Check columns
  for (let i = 0; i < 3; i += 1) {
    if (squareValues[i] !== ''
        && squareValues[i] === squareValues[i + 1]
        && squareValues[i] === squareValues[i + 2]) {
      gameStatus = squareValues[i];
      break;
    }
  }

}

window.addEventListener('DOMContentLoaded', () => {
  document
    .getElementById('tic-tac-toe')
    .addEventListener('click', e => {
      const targetId = e.target.id;
```

```
        && squareValues[i] === squareValues[i + 2]) {
      gameStatus = squareValues[i];
      break;
    }
  }

  // Check columns
  for (let i = 0; i < 3; i += 1) {
    if (squareValues[i] !== ''
        && squareValues[i] === squareValues[i + 3]
        && squareValues[i] === squareValues[i + 6]) {
      gameStatus = squareValues[i];
      break;
    }
  }

  // Check the diagonals
  if (squareValues[0] !== ''
      && squareValues[0] === squareValues[4]
      && squareValues[0] === squareValues[8]) {
    gameStatus = squareValues[0];
  }
  if (squareValues[2] !== ''
      && squareValues[2] === squareValues[4]
      && squareValues[2] === squareValues[6]) {
    gameStatus = squareValues[2];
  }

  if (gameStatus !== '') {
    document
      .getElementById('game-status-message')
      .innerHTML = `Winner: ${gameStatus.toUpperCase()}`;
  }
}

window.addEventListener('DOMContentLoaded', () => {
  document
    .getElementById('tic-tac-toe')
    .addEventListener('click', e => {
      const targetId = e.target.id;

      if (!targetId.startsWith('square-')) return;

      const squareIndex = Number.parseInt(targetId[targetId.length - 1]);
```

5. TTT: New Game

- 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.

6. TTT: New Game Hints

In the HTML file

- Add an `id` attribute with a value of `new-game` to the `<button>` with the text "New Game"
- Add the `disabled` attribute to the button.

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 `new-game` button's `disabled` property to `false`.
- In the `DOMContentLoaded` event handler, add a click event handler to the element with the `new-game` id.
- In the click event handler, reset the variables to their original values
  - `currentPlayerSymbol = 'x';`
  - `squareValues = ['', '', '', '', '', '', '', '', ''];`
  - `gameStatus = '';`
- Also, clear out the header that contains the game status.
- Also, loop through the values 0 through 8, get a reference to the HTML element with the id `square-X` where X is one of those numeric values, and set that HTML elements `innerHTML` property to an empty string to clear out an image that may be in there.
- Also, set the `new-game` buttons' `disabled` property to `true`.

```
  if (squareValues[2] !== ''
    && squareValues[2] === squareValues[4]
    && squareValues[2] === squareValues[6]) {
   gameStatus = squareValues[2];
  }

  let boardIsFilled = true;
  for (let i = 0; i < 9; i += 1) {
   if (squareValues[i] === '') {
    boardIsFilled = false;
    break;
   }
  }
  if (boardIsFilled) {
   gameStatus = 'None'
  }

  if (gameStatus !== '') {
   document
    .getElementById('game-status-message')
    .innerHTML = `Winner: ${gameStatus.toUpperCase()}`;

   document
    .getElementById('new-game')
    .disabled = false;
  }
}

window.addEventListener('DOMContentLoaded', () => {
  document
   .getElementById('tic-tac-toe')
   .addEventListener('click', e => {
    const targetId = e.target.id;

    if (!targetId.startsWith('square-')) return;

    const squareIndex = Number.parseInt(targetId[targetId.length - 1]);

    if (squareValues[squareIndex] !== '') return;

    const img = document.createElement('img');
    img.src = `https://assets.aaonline.io/Module-DOM-API/formative-projec
    e.target.appendChild(img);
```

```
    const targetId = e.target.id;

    if (!targetId.startsWith('square-')) return;

    const squareIndex = Number.parseInt(targetId[targetId.length - 1]);

    if (squareValues[squareIndex] !== '') return;

    const img = document.createElement('img');
    img.src = `https://assets.aaonline.io/Module-DOM-API/formative-proje
    e.target.appendChild(img);

    squareValues[squareIndex] = currentPlayerSymbol;

    if (currentPlayerSymbol === 'x') {
     currentPlayerSymbol = 'o';
    } else {
     currentPlayerSymbol = 'x';
    }

    checkGameStatus();
   });

  document
   .getElementById('new-game')
   .addEventListener('click', () => {
    gameStatus = '';
    document
     .getElementById('game-status-message')
     .innerHTML = '';
    for (let i = 0; i < 9; i += 1) {
     document
      .getElementById(`square-${i}`)
      .innerHTML = '';
    }
    currentPlayerSymbol = 'x';
    document
     .getElementById('new-game')
     .disabled = true;
    squareValues = ['', '', '', '', '', '', '', '', ''];
   });
});
```

7.TTT: 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.

8.TTT: Giving Up Hints

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`.

9. TTT: 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.

10. TTT: Saving Game State Hints

- 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.

11. TTT: Nightmare Mode: You vs. Machine