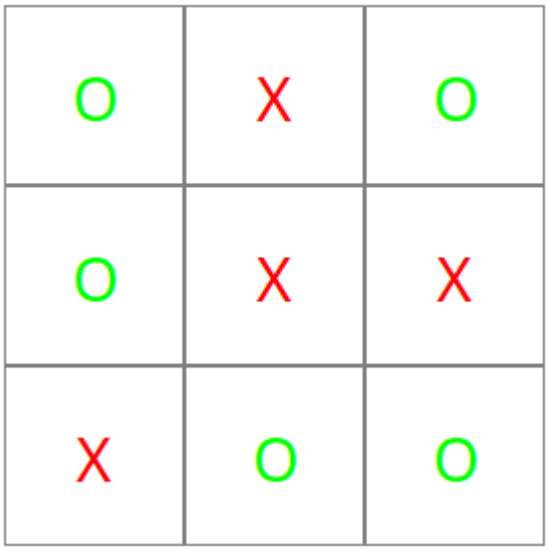
Tic-Tac-Toe practice-lecture

1. **Introduction to the game and the rules** *(5-15 minutes)* – Quick intro to what we will be doing and the logical path that we will be following:
   - How the game will look – (9 squares in our case but maybe more and X / O for players signs).
     Mentioning that we can accomplish this with multiple HTML tags (table, div, span)
   - How we will interact with the game – Mentioning the JavaScript selectors, events (click event).
   - What are the rules of the game: (We ask the students to try to explain the rules before revealing the points below)
     - Once a box is filled, it cannot be changed,
     - If we have 3 in a row of the same type, the game ends and we have a winner. (Row or Column or Diagonal).
     - If all the fields are filled the game ends in draw.



2. **HTML Structure:** We use div elements to form the game board structure: *(5 minutes)*

```
<div id="board">
        <div class="box"></div><div class="box"></div><div
class="box"></div><div class="box"></div><div class="box"></div><div
class="box"></div><div class="box"></div><div class="box"></div><div
class="box"></div>
</div>
```

Explanation to why we use this structure of divs instead of separated by new row (Escaping empty text nodes generated by JavaScript on new lines or tabs within the HTML structure).

3. **Adding some styling to position the game board and boxes:** *(5-10 minutes).*

```
#board{
        width: 300px;
        margin: 0 auto;
        margin-top: 200px;
}
.box{
        width: 90px;
        height: 90px;
        border: 1px solid gray;
        float: left;
        font-size: 50px;
        line-height: 90px;
        text-align: center;
}
```

Width of the board

We insert margins to position the element

We allow the element to float in order to arrange them in pairs of 3

We set a big font size and center the Sign of the player in the box

4. **Game Mechanics + code example:** (30-*40 minutes)*
    1. **The *Javascript* –** We either use the <script> tag that allows us to type JavaScript code within the HTML document or we can add the file in the head of HTML depending of preferences of the course.
    2. **Add events –**
        - **Events explanation** – Events capture some action by the user (examples with the mouse click.)
        - "*We need to set events on each box, so how do we select the boxes?*" (**Explanation of getElementsByClass**).
        - **window.onload explanation –** When the whole page is loaded, we start executing the code.

```
window.onload = function(){
        //the sign of the players
    let playersSigns =['X', 'O'],
        //the color for each sign
        color =['#FF0000','#00FF00'],
        //the current player turn
        currentPlayer =0,
        //array of the div box elements
        boxes,
        //counter of the number of boxes with assigned signs
        boxFilled=0,
        //if winner criteria is met, we change it to true
        winner = false;
    //initialization fuction
    init();

    function init(){
        boxes=document.getElementsByClassName('box');
        for (let i = 0; i < boxes.length; i++) {
            boxes[i].addEventListener('click',markBox);
        }
```

- The *init()* function sets a click event on each box and the second parameter is the function that we call upon activating the event (When we click the box, we get the result of the *markBox* function =]).

```
window.onload = function(){
                    //the sign of the players
                let playersSigns =['X', 'O'],
                    //the color for each sign
                    color =['#FF0000','#00FF00'],
                    //the current player turn
                    currentPlayer =0,
                    //array of the div box elements
                    boxes,
                    //counter of the number of boxes with assigned signs
                    boxFilled=0,
                    //if winner criteria is met, we change it to true
                    winner = false;
                //initialization fuction
                init();

                function init(){
                    boxes=document.getElementsByClassName('box');
                    for (let i = 0; i < boxes.length; i++) {
                        boxes[i].addEventListener('click',markBox);
                    }
                }
```

- **The *markDown()* function:** *"This function is where the magic happens"* - Defines the process of marking the box upon clicking and verify's the winning conditions.

```
function markBox(){
    //if the winner condition is met, we disable the event logic :)
    if(winner) {
        return;
    }
    let sign,
        currentSign;
    sign = currentPlayer %2;
    currentSign = playersSigns[sign];
    if(!this.innerHTML){
        this.innerHTML = currentSign;
        this.style.color = color[sign];
        if(winnerCondition(this)){
            winner = true;
            alert('Player '+currentPlayer+' is the winner!');

        }
        boxFilled++;
        sign++;
        currentPlayer=sign;
    }

    if(boxFilled == boxes.length && !winner){
        alert('Game Draw');
    }
}
```

Choose of the sign to be used in the board box

If the box is empty, we assign the choosen sign and color

After, we check if a winner condition is met and notify with alert.

Increase the number of boxes filled and we give the turn to the other player.

We check if all the boxes are filled and if there is no winner – we draw the game :)

```
function markBox(){
                        //if the winner condition is met, we disable the event logic
:)
                        if(winner) {
                                return;
                        }
                        let sign,
                                currentSign;
                        sign = currentPlayer %2;
                        currentSign = playersSigns[sign];
                        if(!this.innerHTML){
                                this.innerHTML = currentSign;
                                this.style.color = color[sign];
                                if(winnerCondition(this)){
                                        winner = true;
                                        alert('Player '+currentPlayer+' is the
winner!');
                                }
                                boxFilled++;
                                sign++;
                                currentPlayer=sign;
                        }

                        if(boxFilled == boxes.length && !winner){
                                alert('Game Draw');
                        }
                }
```

- The ***winnerCondition(currentBox)*** function – Checks if the new filled box creates a winning row, column or diagonal and returns a boolean (true or false).

```
function winnerCondition(currentBox){
    //decrease index with one to match with boxes array keys.
    let index = -1,
        winnerRow,
        winnerCol,
        winnerDiagonal;
    for (index; currentBox=currentBox.previousSibling; index++);
    winnerRow = checkRows(index)
    winnerCol = checkColumns(index)
    winnerDiagonal = checkDiagonals(index);
    if(winnerRow || winnerCol || winnerDiagonal){
        return true;
    }
    return false;
}
```

We start with index = -1 to match the array keys.

The winner condition variables.

We at which position is the element located (if the center box, from the beginning is the 5-th box

0	1	2
3	4	5
6	7	8

We check the winning rows, columns and diagonals for the new checked box and we return the result.

```
function winnerCondition(currentBox){
                        //decrease index with one to match with boxes array keys.
                        let index = -1,
                                winnerRow,
                                winnerCol,
                                winnerDiagonal;
                        for (index; currentBox=currentBox.previousSibling; index++);
                        winnerRow = checkRows(index)
                        winnerCol = checkColumns(index)
                        winnerDiagonal = checkDiagonals(index);
                        if(winnerRow || winnerCol || winnerDiagonal){
                                return true;
                        }
                        return false;
                }
```

- The ***checkRows(index)*** function – we check the rows that match the criteria based on the currently checked box using the *boxes* array (that's why we needed the find the index of the current box).

```
function checkRows(index){
    switch (Math.floor(index/3)){
        case 0:
        return ((boxes[index].innerHTML === boxes[index+3].innerHTML) && (boxes[index].innerHTML === boxes[index+6].
            innerHTML));
        case 1:
        return ((boxes[index].innerHTML === boxes[index-3].innerHTML) && (boxes[index].innerHTML === boxes[index+3].
            innerHTML));
        case 2:
        return ((boxes[index].innerHTML === boxes[index-3].innerHTML) && (boxes[index].innerHTML === boxes[index-6].
            innerHTML));
    }
}
```

To get the row on which the box is positioned, we divide and use the **Math.floor()** method to get the integer value of the division (removing the floating point part).

```
function checkRows(index){
                        switch (Math.floor(index/3)){
                                case 0:
                                return ((boxes[index].innerHTML ===
boxes[index+3].innerHTML) && (boxes[index].innerHTML === boxes[index+6].innerHTML));
                                case 1:
                                return ((boxes[index].innerHTML === boxes[index-
3].innerHTML) && (boxes[index].innerHTML === boxes[index+3].innerHTML));
                                case 2:
                                return ((boxes[index].innerHTML === boxes[index-
3].innerHTML) && (boxes[index].innerHTML === boxes[index-6].innerHTML));
                        }
                }
```

- The ***checkColumns(index)*** function – Similar to the checkRows function but we check for the columns now.

```
function checkColumns(index){
    switch (index % 3){
        case 0:
        return ( (boxes[index].innerHTML === boxes[index+1].innerHTML) && (boxes[index].innerHTML === boxes[index+2].
            innerHTML) );
        case 1:
        return ((boxes[index].innerHTML === boxes[index+1].innerHTML) && (boxes[index].innerHTML === boxes[index-1].
            innerHTML));
        case 2:
        return ((boxes[index].innerHTML === boxes[index-1].innerHTML) && (boxes[index].innerHTML === boxes[index-2].
            innerHTML));
    }
}
```

We get the row in which the box is located by getting the reminder of the division. Depending if it is in the middle or in the corner, we check either the boxes on the side or the 2 adjacent boxes in the same row.

```
function checkColumns(index){
                    switch (index % 3){
                        case 0:
                        return ( (boxes[index].innerHTML === boxes[index+1].innerHTML)
&& (boxes[index].innerHTML === boxes[index+2].innerHTML) );
                        case 1:
                        return ((boxes[index].innerHTML === boxes[index+1].innerHTML)
&& (boxes[index].innerHTML === boxes[index-1].innerHTML));
                        case 2:
                        return ((boxes[index].innerHTML === boxes[index-1].innerHTML)
&& (boxes[index].innerHTML === boxes[index-2].innerHTML));
                    }
            }
```

- The ***checkDiagonals(index)*** function – If we look at the square with the index numbers on them, we can see that all the numbers that are multiple to ***2*** are position in a diagonal condition to win. So if we get a a box checked in any of those values, we can check both diagonals for winning (**need optimization, but it is simpler**).

```
function checkDiagonals(index){
    var diagonalResult = false;
    if(index % 2 === 0){
        diagonalResult= ( boxes[0].innerHTML.length>0 && ((boxes[0].innerHTML === boxes[4].innerHTML) && (boxes[0].
            innerHTML === boxes[8].innerHTML)));
        if(!diagonalResult){
        diagonalResult = (boxes[2].innerHTML.length>0 && ((boxes[2].innerHTML === boxes[4].innerHTML) && (boxes[2].
            innerHTML === boxes[6].innerHTML)));
        }
    }
    return diagonalResult;
}
```

```
function checkDiagonals(index){
                        var diagonalResult = false;
                        if(index % 2 === 0){
                                diagonalResult= ( boxes[0].innerHTML.length>0 &&
((boxes[0].innerHTML === boxes[4].innerHTML) && (boxes[0].innerHTML ===
boxes[8].innerHTML)));
                                if(!diagonalResult){
                                diagonalResult = (boxes[2].innerHTML.length>0 &&
((boxes[2].innerHTML === boxes[4].innerHTML) && (boxes[2].innerHTML ===
boxes[6].innerHTML)));
                                }
                        }
                        return diagonalResult;
                }
```

5. Task during the rest of the time (60 minutes):
    1. Create a button that reset the game
    2. Create a button to choose which symbol goes first
    3. Create a score counter.