**Noughts and Crosses –**

**JavaScript Game Report**

**David Thompson. Student ID: 30047616**

IS3S664 - Advanced Internet and Mobile Computing

**URL TO GAME:**

<https://ces-web2.southwales.ac.uk/students/30047616/Tic%20Tac%20Toe/noughts.html>

# 

# David Thompson

2023

Table of Contents

[1 Introduction to Game 2](#_Toc155976920)

[2 Description of code and enhancements 3](#_Toc155976921)

[2.1 Game interface with HTML and CSS 3](#_Toc155976922)

[2.2 Core Variables and Board in JavaScript 3](#_Toc155976923)

[2.3 Restart game 4](#_Toc155976924)

[2.4 Grid square event 4](#_Toc155976925)

[3 Timers 7](#_Toc155976926)

[4 Main Menu Screen 9](#_Toc155976927)

[5 Computer opponent 10](#_Toc155976928)

[6 Appendix A: Html 10](#_Toc155976929)

[7 Appendix B: CSS 13](#_Toc155976930)

[8 Appendix C: JavaScript 17](#_Toc155976931)

[9 References 27](#_Toc155976932)

# Introduction to Game

My Tic tact toe game is built upon the basic noughts and crosses game that was developed during the course, with many low to high level enhancements such as 4x4 grid, best of 3 play, timers, main menu screen, cosmetic enhancements, option to change game icons by selecting a theme, different game difficulties, as well as an AI opponent option. My game starts on a main menu screen where you can select a difficulty level and an opponent ( 2 player or vs the computer) with a start button.

The start button will start the game and take the user to the game play screen that features a 4x4 grid tic tac toe game. Here the user can start the game by selecting a tile on the grid as player X, thus starting the other players turn. Both players continue to place their icon on the tiles of the grid until one achieves 4 in a row, the game is drawn ( when there are no available tiles left) , or when a player runs out of time. Each player will have a timer which length is dependent on the difficulty level chosen on the main menu screen. A player’s timer will run during their turn and stop when their turn is over. If a player runs out of time the other player automatically wins that game.

The Game will automatically keep track of each players scores and the number of games played. This will automatically reset after 3 games in this best of 3 tic tac toe game and declare the winner to the player with the most wins during the round. The user has the option to change the theme of the icons at any point during the game by selecting the radio buttons under the “change theme header”.

Other options include the ability to reset the score at any time by clicking on the “Reset Score” button, return to main meu with the “Return to main menu button” , and to Restart the game when the user is ready at the end of each game with the “ Restart Game” button.

**URL TO GAME:**

<https://ces-web2.southwales.ac.uk/students/30047616/Tic%20Tac%20Toe/noughts.html>

# Description of code and enhancements

The game comes with the following main enhancements:

1. 4x4 Grid
2. Best of 3 Play
3. Themes: Themed icons such as Classic, Minions, and Star Wars
4. CSS and cosmetic enhancements
5. Timers for each player, dependent on the difficulty level chosen.
6. Main menu screen
7. AI / Computer Opponent
8. Different game modes such as different difficulty levels.
9. Score tracking system.

## Game interface with HTML and CSS

Each square on the board is created as a separate div as seen in Appendix A that shows the html code with each of the cells containing its own unique identifier, “cell-data-index”, numbered 0 to 15 in this case for the enhanced 4x4 grid game. These cells are wrapped in a div of class “game—container”. A h2 is added of class “game—status”, and “round—status” to display the status of the game and the round. In addition to this there is a div of class “ Game—score” with spans with unique Ids to record Games played, players win and draws, and a div of class “ Select—Theme” which features radio buttons that make it possible for the user to select the theme of the icons they wish to use. The previously mentioned html is contained in section with id “game-screen”. The other section found in the html code is called “menu-screen” which contains the radio buttons that allow the user to select an opponent and a difficulty before starting the game. These two sections in the html code makes it possible to switch between start screen and the game screen with the help of JavaScript code.

Following this the modified CSS code was added, enhanced from the basic game code to provide a more visually appealing game and the 4x4 grid which can be found in Appendix B.

## Core Variables and Board in JavaScript

The first step of the Javascript code defines the variables such as constant variable called statusdisplay and RoundDisplay which uses query selectors that updates the h2 of class “game—status” & “round—status” as the game is played, and some let variables such as “gamestate” which uses an array to represent the current state of the board.

The board was extended to have 16 spaces in the array for the 4x4 grid.

Variables such as timerXValue are used to store the time allowed for a given player during a game with variable timerXinterval, variables such as GamesPlayed, WinX, WinO, draw to keep track of the game scores.

Theme variables are stored in a constant variable called “ themes”. Each theme is represented with a key value inside themes such as Classic, minion, and star which all contain their own unique properties such as name, xImage, and oImage. This allows for easy access later in the code to icons image based on the theme selected on the radio buttons.

## Restart game

The restart button resets the game, including players timers, and clears the board for another game with the game settings chosen by the user such as difficulty, opponent, and themed icon. This button also resets the game scores but only if it’s the 3rd game being played in this best of 3 tic tac toe game.

This is achieved in code by a function that is triggered by a “click” on the restart button that calls handleRestartGame to reset the game to its initial state and updates the statusDisplay to show the player’s turn. This is done by creating a Node list of elements in a DOM Tree of class=cell and setting its contents to “ “.

The function calls “setDifficulty” which is used to set the game difficulty and updates timers using ‘updateTimerXDisplay’ and ‘updateTimerODispplay”. After that both timers will be stopped using a function called “stoptimers”

If the total number of games played is equal to 3 it calls the function “declareRoundWinner” to determine and display the winner in the best of 3 game. Additionally, it resets the variable of total number of games played , and if fewer than 3 games have been played then it updates the scores using the function “updateScore”.

This can be seen in Appendix C, displaying the JavaScript Code.

## Grid square event

In order for the click on square event to function for the tic tac toe game, 4 core functions are used starting with the HandleCellClick and then HandleCellPlayed, HandleResultValidation, and finally handlePlayerChange as follows:

1. **HandleCellClick:**

This function is triggered by clicking on a cell in the game grid, if checks if the game is active or if it has been ended in a win or a draw, retrieves information about the clicked cell such as its index and whether it is already occupied. If the game is not active, then it will do nothing.

Additionally, it determines the selected opponent(human or computer) in this enhanced game based on radio buttons, stops both player’s timers, calls ‘HandleCellPlayed’ to update the game state and board, and then calls ‘handleResultValidation’ to check the game result.

If the game is still active and the opponent is the computer, it sets timeout to delay the computer’s move by calling ‘handleComputerMove’. It also starts the timer for the player making a move.

This is achieved this in the following way:

* if (!gameActive) {

return;

}

This checks If the game is active ( gameActive = true). If it is not, it exits the function and will not proceed with the function.

* const selectedOpponent = document.querySelector('input[name="opponent"]:checked').value;

This retrieves the value selected on the radio buttons (2 Player or computer) for the opponent in the html form.

* if (selectedOpponent === 'computer' && currentPlayer !== 'X') {

return;

}

If the selected opponent is the computer and not player X it exits the function to prevent the human player making a move during the computer’s turn

* const clickedCell = clickedCellEvent.target;

const clickedCellIndex = parseInt(clickedCell.getAttribute('data-cell-index'));

if (gameState[clickedCellIndex] !== "" || !gameActive) {

return;

}

stopTimers(); // stop both players' timers

This retrieves information from the clicked cell such as the DOM element (‘clickedCell’) and its index (‘clickedCellIndex’) from the data attribute “data-cell-index” and checks if the cell is already occupied (‘gameState[clickedCellIndex] !== ""’) or if the game is not active. And finally stops both player’s timers to ensure only the timer of the player making the move will be active.

* An if statement is used to set the timeout to delay computer’s move by calling ‘handleComputerMove’ if the game is active and the selectedOpponent is the computer.

Else it starts the timer for the player making a move in an active game vs another human player.

1. **HandleCellPlayed**

This will update the gameState to show which player has played that square and will update the HTML board.

An enhancement here includes code that updates the HTML of the board to show the player’s move with the selected theme image icons depending on the selected theme on the radio buttons.

This is achieved by first updating the game State array by writing the corresponding themed icon for player “X” and “O” to the item indexed by the data-cell-index value received.

The function is passed the element node representing the cell and integer data-cell-index which identifies the cell.

Finally, it updates the content of the element node received to contain HTML that will show the active players token or icon.

The value of the selected theme from the radio buttons named “theme” in the html form is achieved with the following variable and query selector:

const selectedTheme = document.querySelector('input[name="theme"]:checked').value;

const theme = themes[selectedTheme];

const theme retrieves and stores in a variable the details of the selected theme from the ‘themes’ properties such as ‘xImage’ and ‘oImage’ previously defined.

This is then applied when the code checks the current player and updates the clicked cell as follows:

gameState[clickedCellIndex] = currentPlayer;

if (currentPlayer == "X") { clickedCell.innerHTML = `<img src= "${theme.xImage}" alt = 'X' width = '100' height = '100'>`; }

else { clickedCell.innerHTML = `<img src= "${theme.oImage}" alt = 'O' width = '100' height = '100'>`; }

1. **HandleResultValidation**

This function checks for player wins or if the game ended in a draw based on the current state of the game. It updates the displays the scores accordingly.

This is achieved by first defining an array of winning conditions, modified for a 4x4 grid, called “winningConditions”:

const winningConditions = [

[0, 1, 2, 3],

[4, 5, 6, 7],

[8, 9, 10, 11],

[12, 13, 14, 15],

[0, 4, 8, 12],

[1, 5, 9, 13],

[2, 6, 10, 14],

[3, 7, 11, 15],

[0, 5, 10, 15],

[3, 6, 9, 12]

];

The function then uses a loop to iterate through the winning conditions array, and checks if any of the 4 cells are empty with each winning condition.

If all four cells in the winning conditions have been played, the code checks to see if it has been played by the same player. If it was then it sets ‘roundWon’ to ‘true’ and breaks out of the loop.

Using an if statement it determines if a win is detected it stops the timers and updates the status display and increments the scores.

If the game is a draw ( no empty cells left), it stops the timers, updates the status display, and increments the draw count.

Else the game is neither won nor drawn, and proceeds to call the ‘hanldePlayerChange’ function to switch turns.

1. **HandlePlayerChange**

This function switches the turns between the active players of the game.

To do this a ternary operator is used (‘currentPlayer = currentPlayer === "X" ? "O" : "X";’) that toggles the value of ‘currentPlayer’ between “X” and “O”. If the current player is “X” then it changes to “O” , and vice versa.

It then updates the HTML of the statusDisplay to inform the players turn (‘statusDisplay.innerHTML = `It's ${currentPlayer}'s turn`’). The ${currentPlayer} dynamically inserts the current player into the html display.

# Timers

Both players have their own unique timer during a game. The amount a time the players have is adjusted based on the difficulty level chosen: Easy(30sec), medium(20sec), and hard (5sec). When a player runs out of time the other player wins that game, or possibly the round if the player has the most amount of wins at the end of the best of 3 round. The players timers stop while the other player takes their turn, and vice versa.

To accomplish this, the initial values for the timers are set using the variables:

let timerXValue = 30;

let timerOValue = 30;

let timerXInterval;

let timerOInterval;

function startTimerX() {

        timerXInterval = setInterval(() => {

            if (timerXValue > 0) {

                timerXValue--;

                updateTimerXDisplay();

            } else {

                //Player x runs out of time

                clearInterval(timerXInterval);

                handlePlayerTimeout('X');

            }

        }, 700 //timer updates every half a second approx

        )

    }

    function startTimerO() {

        timerOInterval = setInterval(() => {

            if (timerOValue > 0) {

                timerOValue--;

                updateTimerODisplay();

            } else {

                //Player x runs out of time

                clearInterval(timerOInterval);

                handlePlayerTimeout('O');

            }

        }, 700 //timer updates every half a second approx

        )

    }

The timers are started when their respective functions are called, StartTimerX and StartTimerO , using ‘setInterval’ to run the timer values at a regular interval. In this case, approx. every half a second (700miliseconds).

The functions ‘updateTimerxDisplay’ and ‘updateTimerODisplay’ update the html with the current timer values. This is done by using an ID in the html and retrieving it with ‘document.getElementById’ with the ‘textcontent’ property of the element to the current value of the respective timer variable.

There is a function to stop the timers called ‘stopTimers’ which stops both players timers when called, and a handlePlayerTimeout function which is called when a players timer reaches 0. When this happens it updates the games played as well as the wins of player O if it was player X that ran out of time, and vice versa with the help of an If statement.

The setDiffuclty function uses a querySelector to retrieve the difficulty value selected on the radio buttons and stores it in a variable called ‘difficulty’. An If statement is used to set TimerXvalue and set TimerOvalue based on the difficulty level selected: easy, medium, or hard.

# Main Menu Screen

As mentioned, the html has two sections of class ‘menu-screen’ and ‘game-screen’.

This was learned via research found at: <https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event>

const menuScreen = document.querySelector('.menu-screen').

const gameScreen = document.querySelector('.game-screen').

In JavaScript, the code uses ‘document.querySelector’ to get the elements from the html that will represent the menu screen and the game screen stored in a variable. These variables are within the ‘document.addEventListerner(“DOMContentLoaded”, function()’. This listens out for a trigger to execute the function of type “DOMcontentLoaded” when the initial HTML has been loaded and parsed.

In this case, the “Start button” triggers the transition between the menu to the game screen with the help of an event listener(‘startBtn.addEventListener(‘click’, function()’). When it detects a “click” on the start button it executes:

menuScreen.style.display = "none”.

        //show game screen

        GameScreen.style.display = "block”.

        handleRestartGame();

This hides the HTML and styles in the menuScreen section and displays the game Screen and starts the game.

This can be reverted to the main menu screen with the “Return to main menu” button with the help of another event listener that does the opposite to the code presented in the startBtn function.

# Computer opponent

As seen in the ‘handleCellClick’ function, within the if statement of an active game and that has the SelectedOpponent as ‘computer’ it calls the handleComputerMove function.

if (!gameActive || currentPlayer === "X") {

return;

}

The handleComputerMove checks if the game is active and if the current player is “O”. If it is not, then it does nothing, and if it is then it calls the function ‘GenerateComputerMove’ to get a randomly chosen index for the computer’s move and stores in a variable(‘const computerIndex’).

Following this, the function then gets the HTML element from the chosen cell which is stored in a variable called ‘computerTurn’, and calls handleCellPlayed with ‘computerTurn’ and ‘computerIdex’ passed as parameters to make its move.

After the move it calls ‘handleResultValidation’ to check if the game has been won or drawn.

The function GenerateComputerMove starts by creating an empty array which later is used to store the current empty cells of the game. The function then proceeds by using a loop to iterate through the gameState array to find currently empty cells and then stores them in the empty cells array that was previously defined.

The next step in GenerateComputerMove is to generate a random index from the empty cell array using ‘Math.floor(Math.random() \* emptyCell.length’ which givens a random number between 0 and then length of ‘emptyCell’ array. This is then stored into a variable called “random” and is returned from the function to be used.

This was created with the help of the information found at: <https://www.tutorialspoint.com/how-to-create-an-array-with-random-values-with-the-help-of-javascript#:~:text=Using%20Math.&text=random()%20is%20one%20of>

# Appendix A: Html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport"

        content="width=device-width, user-scalable=no, initial-scale=1.0,maximum-scale=1.0, minimum-scale=1.0">

    <title>Tic tac toe</title>

    <link rel="stylesheet" href="noughts.css">

</head>

<body>

    <section class="menu-screen">

        <div class="game--title">

            <h1>Tic tac toe</h1>

            <button class="start">Start Game!</button>

        </div>

        <section id="main\_menu">

            <div class="setting\_opt">

                <h2>Choose your opponent!</h2>

                <label>

                    <input type="radio" name="opponent" value="2 Player" checked> 2 Player

                </label> <br>

                <label>

                    <input type="radio" name="opponent" value="computer"> Computer

                </label><br>

            </div>

            <div class="setting\_opt">

                <h2>Choose your difficulty!</h2>

                <label>

                    <input type="radio" name="difficulty" value="easy"> Easy

                </label> <br>

                <label>

                    <input type="radio" name="difficulty" value="medium" checked> Medium

                </label> <br>

                <label>

                    <input type="radio" name="difficulty" value="hard"> Hard

                </label> <br>

            </div>

        </section>

    </section>

    <section class="game-screen" style="display:none;">

        <div class="game-info">

            <h1 class="game--title">Tic tac toe</h1>

            <div class="timer">Player X's timer: <span id="timerX">60</span>seconds</div>

            <div class="timer">Player O's timer: <span id="timerO">60</span>seconds</div>

            <h2 class="game--status"></h2>

            <h2 class="round--status"></h2>

            <div class="game--container">

                <div data-cell-index="0" class="cell"></div>

                <div data-cell-index="1" class="cell"></div>

                <div data-cell-index="2" class="cell"></div>

                <div data-cell-index="3" class="cell"></div>

                <div data-cell-index="4" class="cell"></div>

                <div data-cell-index="5" class="cell"></div>

                <div data-cell-index="6" class="cell"></div>

                <div data-cell-index="7" class="cell"></div>

                <div data-cell-index="8" class="cell"></div>

                <div data-cell-index="9" class="cell"></div>

                <div data-cell-index="10" class="cell"></div>

                <div data-cell-index="11" class="cell"></div>

                <div data-cell-index="12" class="cell"></div>

                <div data-cell-index="13" class="cell"></div>

                <div data-cell-index="14" class="cell"></div>

                <div data-cell-index="15" class="cell"></div>

            </div>

        </div>

        <section class="Game--settings">

            <div class="Game--score">

                <h2>Game scores:</h2>

                <h3>Best of 3</h3>

                <p>Games Played: <span id="Games">0</span></p>

                <p>Player X wins: <span id="Win01">0</span></p>

                <p>Player O wins: <span id="Win02">0</span></p>

                <p>Game Draws: <span id="GameDraw">0</span></p>

            </div>

            <div class="Select--Theme">

                <h3>Change theme:</h3>

                <label>

                    <input type="radio" name="theme" value="classic" onclick="url('clasic.png')" checked> Classic Theme

                </label> <br>

                <label>

                    <input type="radio" name="theme" value="minion" onclick="url('back.jpg')"> Minion Theme

                </label><br>

                <label>

                    <input type="radio" name="theme" value="star" onclick="url('stars.jpg')"> Star Wars Theme

                </label><br>

                <button class="game--restart">Retart Game!</button>

                <button class="reset--score">Reset Score</button>

                <button class="returnbtn">Return to main Menu</button>

            </div>

        </section>

        <div class="clearfix"></div>

    </section>

    <script src="noughts.js"></script>

</body>

</html>

# Appendix B: CSS

body {

    font-family: "Arial", sans-serif;

    background: rgb(70, 70, 85);

    background: linear-gradient(90deg, rgba(70, 70, 85, 1) 0%, rgba(210, 210, 233, 1) 35%, rgba(110, 110, 119, 1) 100%);

    margin: 0px;

    padding: 0px;

}

section {

    text-align: center;

    margin-top:20px;

}

button{

    cursor: pointer;

}

.Game--settings .returnbtn{

    margin-top: 70px;

}

.game--status {

    margin: 20px;

    font-size: 24px;

    color:#210af7;

}

.game--container {

    width: 65%;

    display: grid;

    grid-template-columns: repeat(4, 1fr);

    /\* change to (4, 1fr) for equal width columns\*/

    width: 406px;

    /\*changed to accomodate space of 4x4 grid\*/

    margin: 50px auto;

}

.game-info{

   float: left;

   margin:50px auto;

   margin-top: 2px;

   margin-left: 35%;

}

.Game--settings {

    float:left;

    width: 25%;

    height: 725px;

    margin: 5px;

    margin-left: 100px;

    border: 1px solid #ccc;

    font-size: 20px;

}

.Select--Theme {

    height: 250px;

    margin-top: 45px !important;

    margin: 5px;

}

.Select--Theme input {

    padding: 5px;

    margin: 12px;

}

.Select--Theme button,

.game--restart {

    width: 150px;

    height: 60px;

    font-size: 18px;

    font-weight: bold;

    letter-spacing: 1px;

    border-radius: 20px;

    margin: 10px;

}

.menu-screen .start{

    display: block;

    width: 350px;

    height:120px;

    background-color: transparent;

    border:none;

    text-align: center;

    margin: 0px auto;

    margin-top:60px;

    font-size: 40px;

    letter-spacing: 3px;

    line-height: 105px;

    color:#00538C;

    transition:all 1s;

}

.game--title{

    font-family:cursive;

    font-size: 41px;

    letter-spacing: 2px;

    text-shadow: rgb(17, 236, 240) 1px 0 10px;

}

#main\_menu{

    width:500px;

    height:450px;

    border:5px solid white;

    box-shadow: 0px 0px 2px gray;

    margin:20px auto;

    text-align: center;

}

#main\_menu h2{

    font-size: 25px;

}

#main\_menu .setting\_opt{

  vertical-align: center;

}

#main\_menu input{

    padding:15px;

    margin: 5px auto;

    margin-top: 30px;

}

#main\_menu label{

    font-size: 24px;

    cursor: pointer;

}

.start:hover{

    background-color: #333;

    border: 10px solid #37BCF9;

    border-radius: 240px;

    box-shadow: 0px 0px 10px gray;

    color:white ;

    text-shadow: 1px 1px 1px black, 1px 2px 1px black, 1px 3px 1px black, 1px 4px 1px black, 1px 5px 1px black, 1px 6px 1px black;

}

.round--status {

    font-size: 18px;

    font-weight: bold;

    color: red;

}

.music\_btn {

   text-align: center;

}

.fade-out {

    opacity: 0;

    transition: opacity 1s ease-in-out;

}

.clearfix {

    float: none;

    clear: both;

}

.cell {

    font-family: "Permanent marker", cursive;

    width: 100px;

    height: 100px;

    box-shadow: 0 0 0 1px #333333;

    border: 1px solid #333333;

    cursor: pointer;

    line-height: 100px;

    font-size: 60px;

}

# Appendix C: JavaScript

document.addEventListener("DOMContentLoaded", function () {

    const menuScreen = document.querySelector('.menu-screen');

    const GameScreen = document.querySelector('.game-screen');

    const startBtn = document.querySelector('.start');

    //Event listener for the "Start" button

    startBtn.addEventListener("click", function () {

        //hide menu screen

        menuScreen.style.display = "none";

        //show game screen

        GameScreen.style.display = "block";

        handleRestartGame();

    });

    const statusDisplay = document.querySelector('.game--status');

    const RoundDisplay = document.querySelector('.round--status');

    let gameActive = true;

    let currentPlayer = "X"

    let gameState = ["", "", "", "", "", "", "", "", "", "", "", "", "", "", "", ""]; // extended to 16 spaces fo 4x4 grid

    let timerXValue = 30;

    let timerOValue = 30;

    let timerXInterval;

    let timerOInterval;

    let timerStarted = false;

    //keep track of games and wins with variables:

    let gamesPlayed = 0;

    let winX = 0;

    let winO = 0;

    let draw = 0;

    //Game themes

    const themes = {

        classic: {

            name: 'Classic',

            xImage: 'X.png',

            oImage: 'O.png',

        },

        minion: {

            name: 'Minion',

            xImage: 'minion.png',

            oImage: 'minion\_evil.png',

        },

        star: {

            name: 'Star Wars',

            xImage: 'R2D2.png',

            oImage: 'vader.png',

        },

    };

    //Update Game Timers

    function updateTimerXDisplay() {

        document.getElementById('timerX').textContent = timerXValue;

    }

    function updateTimerODisplay() {

        document.getElementById('timerO').textContent = timerOValue;

    }

    function updateScore() {

        document.getElementById('Games').textContent = gamesPlayed;

        document.getElementById('Win01').textContent = winX;

        document.getElementById('Win02').textContent = winO;

        document.getElementById('GameDraw').textContent = draw;

    }

    function startTimerX() {

        timerXInterval = setInterval(() => {

            if (timerXValue > 0) {

                timerXValue--;

                updateTimerXDisplay();

            } else {

                //Player x runs out of time

                clearInterval(timerXInterval);

                handlePlayerTimeout('X');

            }

        }, 700 //timer updates every half a second approx

        )

    }

    function startTimerO() {

        timerOInterval = setInterval(() => {

            if (timerOValue > 0) {

                timerOValue--;

                updateTimerODisplay();

            } else {

                //Player x runs out of time

                clearInterval(timerOInterval);

                handlePlayerTimeout('O');

            }

        }, 700 //timer updates every half a second approx

        )

    }

    function stopTimers() {

        clearInterval(timerXInterval);

        clearInterval(timerOInterval);

    }

    function setDifficulty() {

        const selectedDifficulty = document.querySelector('input[name="difficulty"]:checked').value; //retrieves value selected on radio buttons, which is then stored in a variable

        const difficulty = selectedDifficulty;

        if (difficulty === 'hard') {

            timerXValue = 5;

            timerOValue = 5;

        } else if (difficulty === 'medium') {

            timerXValue = 20;

            timerOValue = 20;

        } else {

            timerXValue = 30;

            timerOValue = 30;

        }

    }

    function handlePlayerTimeout(player) {

        if (!gameActive) {

            return;

        }

        stopTimers();

        statusDisplay.innerHTML = `Player ${player} ran out of time.  Player ${player === 'X' ? 'O' : 'X'} wins!`;

        gameActive = false;

        if (player === 'X') {

            winO++;

        } else {

            winX++;

        }

        gamesPlayed++;

        updateScore();

        if (gamesPlayed === 3) {

            declareRoundWinner();

        } else {

            RoundDisplay.innerHTML = '';

        };

    }

    statusDisplay.innerHTML = `It's ${currentPlayer}'s turn`;

    function handleRestartGame() {

        gameActive = true;

        currentPlayer = "X";

        gameState = ["", "", "", "", "", "", "", "", "", "", "", "", "", "", "", ""];  // extended to 16 spaces for the 4x4 grid

        statusDisplay.innerHTML = `It's ${currentPlayer}'s turn`;

        document.querySelectorAll('.cell').forEach(cell => cell.innerHTML = "");

        //Start timer for Player X

        //reset both players timers

        setDifficulty();

        updateTimerXDisplay();

        updateTimerODisplay();

        stopTimers(); // stop both timers when game is restarted

        if (gamesPlayed === 3) {

            declareRoundWinner();

            //reset scores after 3 games

            winX = 0;

            winO = 0;

            draw = 0;

            gamesPlayed = 0;

        } else {

            RoundDisplay.innerHTML = '';

            updateScore();

        }

    }

    function handleReset() {

        winX = 0;

        winO = 0;

        draw = 0;

        gamesPlayed = 0;

        updateScore();

    }

    /\*Return to main menu\*/

    function main\_menu() {

        //show menu screen

        menuScreen.style.display = "block";

        //hide game screen

        GameScreen.style.display = "none";

    }

    function handleCellClick(clickedCellEvent) {

        if (!gameActive) {

            return;

        }

        const selectedOpponent = document.querySelector('input[name="opponent"]:checked').value;

        //Prevents human player from making a move when it is the computers turn

        if (selectedOpponent === 'computer' && currentPlayer !=='X'){

            return;

        }

        const clickedCell = clickedCellEvent.target;

        const clickedCellIndex = parseInt(clickedCell.getAttribute('data-cell-index'));

        if (gameState[clickedCellIndex] !== "" || !gameActive) {

            return;

        }

        stopTimers();  //stop both players timers

        handleCellPlayed(clickedCell, clickedCellIndex);

        handleResultValidation();

        if (gameActive && selectedOpponent === 'computer' ) {

            setTimeout(() => {

                handleComputerMove();

                // start timer for the player that makes a move

                if (gameActive) {

                    if (currentPlayer === "X") {

                        startTimerX();

                    } else {

                        startTimerO();

                    }

                }

            }, 500)

        } else { // start timer for the player that makes a move

            if (gameActive) {

                if (currentPlayer === "X") {

                    startTimerX();

                } else {

                    startTimerO();

                }

            }

        }

    }

    function handleCellPlayed(clickedCell, clickedCellIndex) {

        const selectedTheme = document.querySelector('input[name="theme"]:checked').value;

        const theme = themes[selectedTheme];

        gameState[clickedCellIndex] = currentPlayer;

        if (currentPlayer == "X") { clickedCell.innerHTML = `<img src= "${theme.xImage}" alt = 'X' width = '100' height = '100'>`; }

        else { clickedCell.innerHTML = `<img src= "${theme.oImage}" alt = 'O' width = '100' height = '100'>`; }

    }

    function GenerateComputerMove() {

        //creates an array , which later will be used to store currently empty cells

        const emptyCell = [];

        //iterats through the gameState array to find currrently empty cells and stores them in "emptyCell"

        gameState.forEach((cell, index) => {

            if (cell === "") {

                emptyCell.push(index);

            }

        });

        //calculates a random empty cell by multiplying a random decimal and storing it in a variable

        const random = Math.floor(Math.random() \* emptyCell.length);

        return emptyCell[random];

    }

    function handleComputerMove() {

        if (!gameActive || currentPlayer === "X") {

            return;

        }

        const computerIndex = GenerateComputerMove();

        const computerTurn = document.querySelector(`.cell[data-cell-index="${computerIndex}"]`);

        handleCellPlayed(computerTurn, computerIndex);

        handleResultValidation();

    }

    const winningConditions = [

        [0, 1, 2, 3],

        [4, 5, 6, 7],

        [8, 9, 10, 11],

        [12, 13, 14, 15],

        [0, 4, 8, 12],

        [1, 5, 9, 13],

        [2, 6, 10, 14],

        [3, 7, 11, 15],

        [0, 5, 10, 15],

        [3, 6, 9, 12]

    ];

    function handleResultValidation() {

        let roundWon = false;

        for (let i = 0; i <= 9; i++) {  //changed to 9 to iterate through the winning conditions of the 4x4 grid

            const winCondition = winningConditions[i];

            let a = gameState[winCondition[0]];

            let b = gameState[winCondition[1]];

            let c = gameState[winCondition[2]];

            let d = gameState[winCondition[3]];   //added extra winning conditions

            if (a === '' || b === '' || c === '' || d === '') {  //checks if all 4 conditions are empty, if so it moves to next condition

                continue;

            }

            if (a === b && b === c && c === d) {  //added the extra condition that checks if the values match

                roundWon = true;

                break

            }

        }

        if (roundWon) {

            stopTimers();

            statusDisplay.innerHTML = `Player ${currentPlayer} has won!`;

            gameActive = false;

            if (currentPlayer === 'X') {

                winX++;

            } else {

                winO++;

            }

            gamesPlayed++;

            updateScore();

            return;

        }

        let roundDraw = !gameState.includes("");

        if (roundDraw) {

            stopTimers();

            statusDisplay.innerHTML = `Game ended in a draw!`;

            gameActive = false;

            draw++;

            gamesPlayed++;

            updateScore();

            return;

        }

        handlePlayerChange();

    }

    function handlePlayerChange() {

        console.log('changing player')

        currentPlayer = currentPlayer === "X" ? "O" : "X";

        statusDisplay.innerHTML = `It's ${currentPlayer}'s turn`;

    }

    function declareRoundWinner() {

        if (winX > winO) {

            RoundDisplay.innerHTML = 'Player 1 wins the round!';

        } else if (winO > winX) {

            RoundDisplay.innerHTML = 'Player 2 wins the round!';

        } else {

            RoundDisplay.innerHTML = 'The round ended in a draw!';

        }

        updateScore();

    }

    document.querySelectorAll('.cell').forEach(cell => cell.addEventListener('click', handleCellClick));

    document.querySelector('.game--restart').addEventListener('click', handleRestartGame);

    document.querySelector('.reset--score').addEventListener('click', handleReset);

    document.querySelector('.returnbtn').addEventListener('click', main\_menu);

});

**URL TO GAME:**

<https://ces-web2.southwales.ac.uk/students/30047616/Tic%20Tac%20Toe/noughts.html>

# References

1. Bolts, D. ([no date][a]) *Minion, angry icon - Free download on Iconfinder*. Available at: https://www.iconfinder.com/icons/173992/minion\_angry\_icon (Accessed: 11 February 2023).
2. Bolts, D. ([no date][b]) *Minion, evil icon - Free download on Iconfinder*. Available at: https://www.iconfinder.com/icons/173990/minion\_evil\_icon (Accessed: 11 February 2023).
3. *Circle Icons, Logos, Symbols – Free Download PNG, SVG* ([no date]). Available at: https://icons8.com/icons/set/circle (Accessed: 11 February 2023).
4. *Cross Icons, Logos, Symbols – Free Download PNG, SVG* ([no date]). Available at: https://icons8.com/icons/set/cross (Accessed: 11 February 2023).
5. *Document: DOMContentLoaded event - Web APIs | MDN* (2023). Available at: https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded\_event (Accessed: 11 February 2023).
6. *How to create an array with random values with the help of JavaScript?* ([no date]). Available at: https://www.tutorialspoint.com/how-to-create-an-array-with-random-values-with-the-help-of-javascript#:~:text=Using%20Math.&text=random()%20is%20one%20of (Accessed: 22 November 2023).
7. *R2-D2 color icon in PNG, SVG* ([no date]). Available at: https://icons8.com/icon/39423/r2-d2 (Accessed: 11 February 2023).
8. *Star wars Icons, Logos, Symbols – Free Download PNG, SVG* ([no date]). Available at: https://icons8.com/icons/set/star-wars (Accessed: 11 February 2023).