Tic-Tac-Toe w/ ChatGPT

Prompt Engineering Project

OpenAI

ChatGPT

ChatGPT is an AI chatbot that uses machine learning to generate human-like responses to user input in real-time. It was developed by OpenAI, an artificial intelligence research lab.

ChatGPT is a type of generative AI that can create text, images, or videos based on user prompts. It uses a deep learning neural network that's modeled after the human brain to learn patterns and relationships in text data.

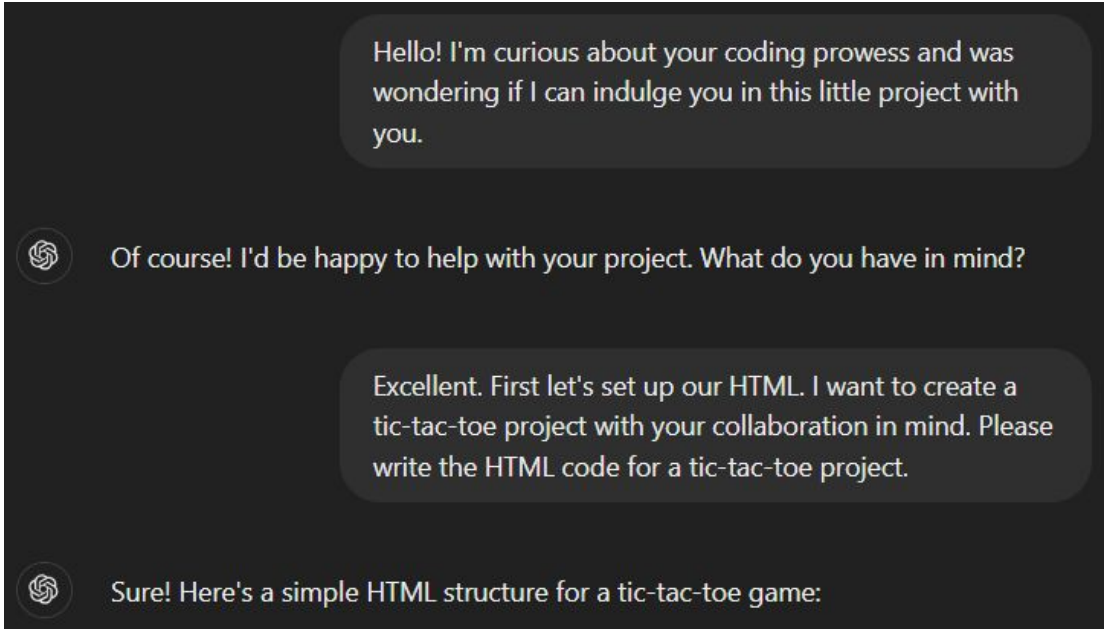
The network is then able to predict what text should come next in a sentence based on the **prompt** and previously predicted text. ChatGPT can answer questions, write copy, draft emails, and hold conversations in a conversational tone.

Prompt Engineering

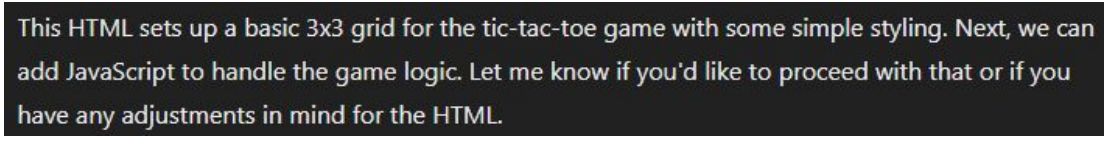
Prompt Engineering is the process of designing inputs for artificial intelligence (AI) tools to produce desired outputs. It involves writing, refining, and optimizing prompts to elicit specific responses from AI models.

Prompt

Here is a screenshot of the prompts below for the HTML structure.



NOTE: The <html> code also included a simple <style> section generated in the <head>!



Code

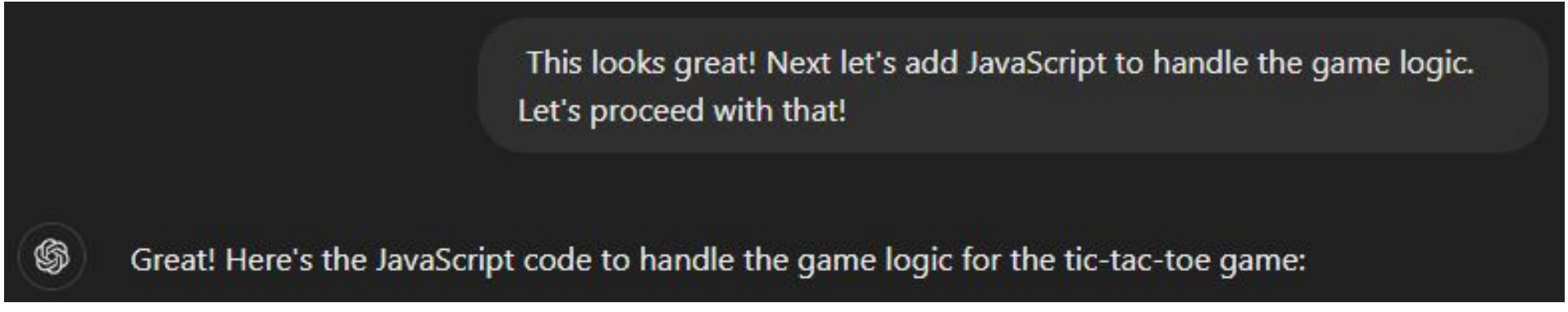
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tic-Tac-Toe Game</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            background-color: #f0f0f0;
        }
```

```css
.container {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    grid-template-rows: repeat(3, 100px);
    gap: 5px;
}
.cell {
    display: flex;
    justify-content: center;
    align-items: center;
    width: 100px;
    height: 100px;
    background-color: #ffffff;
    border: 1px solid #000000;
    font-size: 2em;
    cursor: pointer;
}
.cell:hover {
    background-color: #f1f1f1;
}
```

```html
    </style>
</head>
<body>
    <div class="container">
        <div class="cell" data-cell></div>
        <div class="cell" data-cell></div>
        <div class="cell" data-cell></div>
        <div class="cell" data-cell></div>
        <div class="cell" data-cell></div>
        <div class="cell" data-cell></div>
        <div class="cell" data-cell></div>
        <div class="cell" data-cell></div>
        <div class="cell" data-cell></div>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

Prompt

Here is a screenshot of a prompt below for JavaScript.



Code

From left to right, here is the generated JavaScript code.

```
document.addEventListener('DOMContentLoaded', () => {
    const cells = document.querySelectorAll('[data-cell]');
    const WINNING_COMBINATIONS = [
        [0, 1, 2],
        [3, 4, 5],
        [6, 7, 8],
        [0, 3, 6],
        [1, 4, 7],
        [2, 5, 8],
        [0, 4, 8],
        [2, 4, 6]
    ];
    let circleTurn;
    const X_CLASS = 'x';
    const CIRCLE_CLASS = 'circle';
```

```
startGame();

function startGame() {
    circleTurn = false;
    cells.forEach(cell => {
        cell.classList.remove(X_CLASS);
        cell.classList.remove(CIRCLE_CLASS);
        cell.removeEventListener('click', handleClick);
        cell.addEventListener('click', handleClick, { once: true });
    });
}
```

```
function handleClick(e) {
    const cell = e.target;
    const currentClass = circleTurn ? CIRCLE_CLASS : X_CLASS;
    placeMark(cell, currentClass);
    if (checkWin(currentClass)) {
        endGame(false);
    } else if (isDraw()) {
        endGame(true);
    } else {
        swapTurns();
    }
}
```

```
function endGame(draw) {
    if (draw) {
        alert('Draw!');
    } else {
        alert(`${circleTurn ? "O's" : "X's"} Wins!`);
    }
    startGame();
}

function isDraw() {
    return [...cells].every(cell => {
        return cell.classList.contains(X_CLASS) || cell.classList.contains(CIRCLE_CLASS);
    });
}

function placeMark(cell, currentClass) {
    cell.classList.add(currentClass);
}
```

```
    function swapTurns() {
        circleTurn = !circleTurn;
    }

    function checkWin(currentClass) {
        return WINNING_COMBINATIONS.some(combination => {
            return combination.every(index => {
                return cells[index].classList.contains(currentClass);
            });
        });
    }
});
```

Save this code in a file named `script.js` and ensure it is in the same directory as your HTML file.

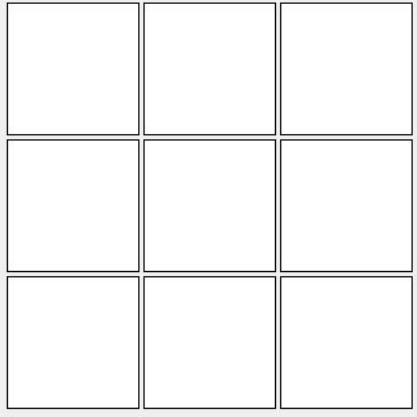
Result

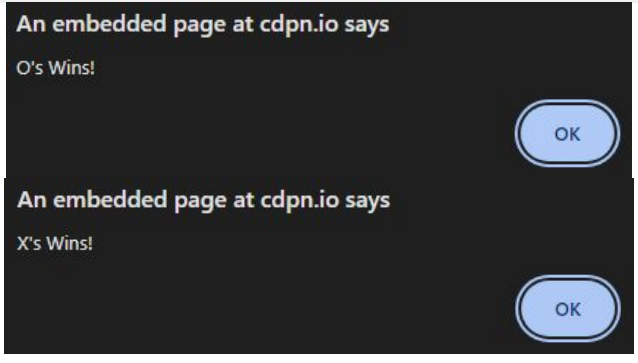
After providing ChatGPT with prompts to generate a simple tic-tac-toe game, I put together the code to create a functioning version by the few generated lines of HTML, CSS, and JavaScript.

While the game is not visually complex and lacks elaborate styling, it becomes interactive as I click on the squares to place their X's and O's.

Despite the minimalistic design, the game demonstrates functionality as it responds to my actions with prompts for the moves of O's and X's.

This project showcases the potential of using AI-driven prompts to quickly develop interactive web applications, highlighting my ability to leverage modern tools to create engaging content for my portfolio.





GPT-4o vs. Code Copilot

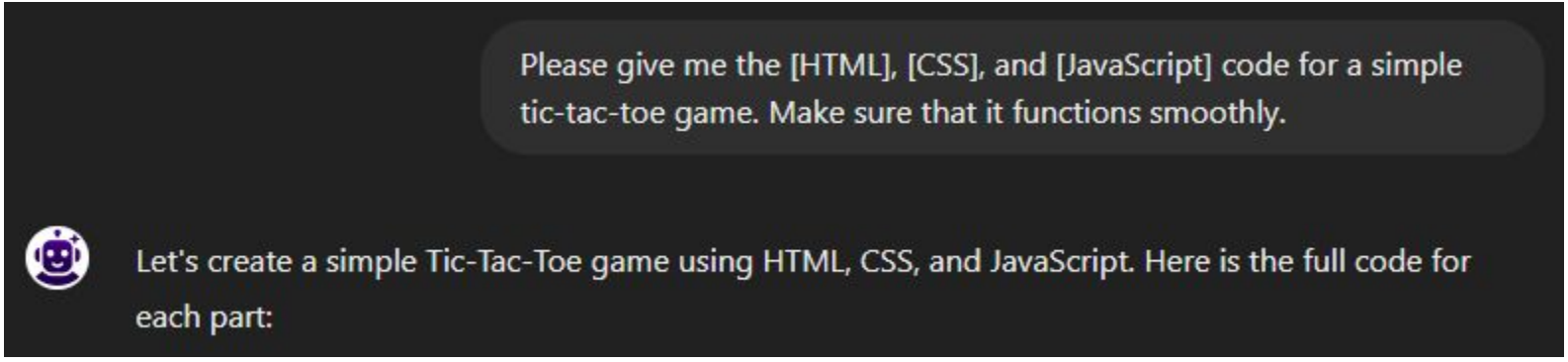
I initially used ChatGPT-4o to build a tic-tac-toe application. While it made some progress (see previous slides), the code remained incomplete, and the X's and O's did not appear in the squares when clicked.

Fortunately, using **Code Copilot**, an advanced AI tool designed for developers, I was able to complete the application with a single prompt (see below).

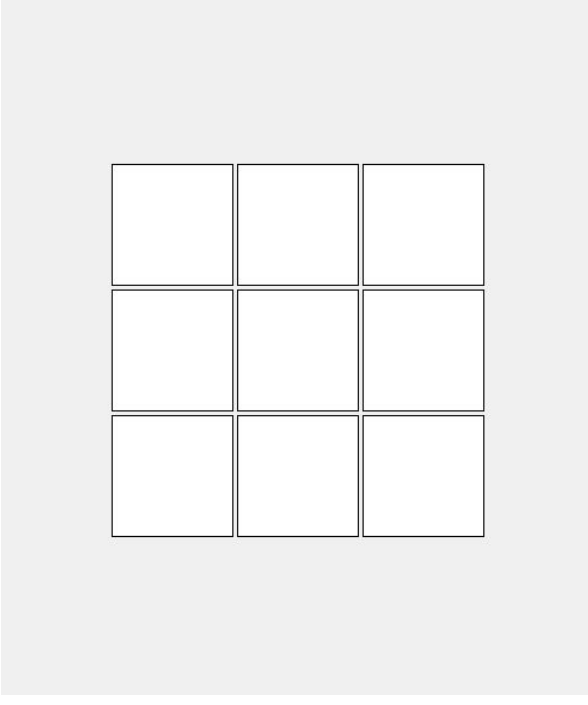
Code Copilot integrates with popular IDEs, supports multiple programming languages, and offers features that enhance productivity, improve code quality, and support learning.

Code Copilot

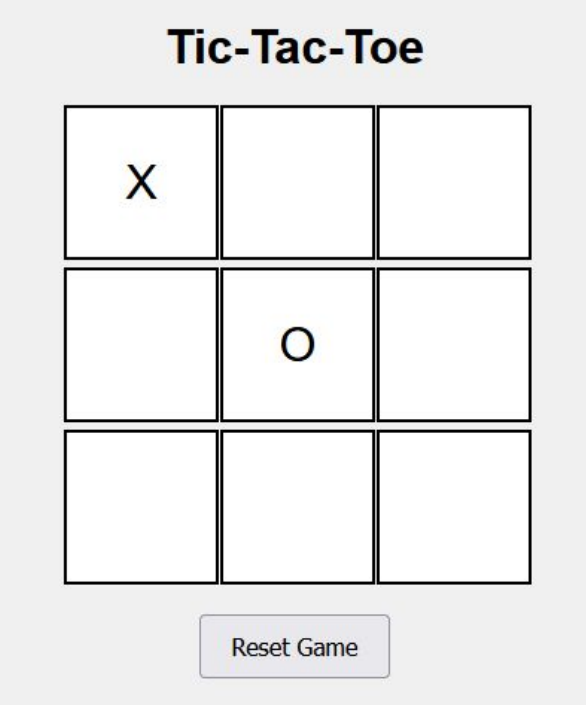
By promptspellsmith.com ✓ X



Before



After



The initial version of the project was developed using ChatGPT-4o. The updated version, created with **Code Copilot**, maintains its minimalist design while introducing several new features: a game title, six interactive squares, and a reset button.

Code

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tic-Tac-Toe</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Tic-Tac-Toe</h1>
    <div class="board" id="board">
        <div class="cell" data-index="0"></div>
        <div class="cell" data-index="1"></div>
        <div class="cell" data-index="2"></div>
        <div class="cell" data-index="3"></div>
        <div class="cell" data-index="4"></div>
        <div class="cell" data-index="5"></div>
        <div class="cell" data-index="6"></div>
        <div class="cell" data-index="7"></div>
        <div class="cell" data-index="8"></div>
    </div>
    <button id="resetButton">Reset Game</button>
    <script src="script.js"></script>
</body>
</html>
```

```css
body {
    font-family: Arial, sans-serif;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
    margin: 0;
    background-color: #f0f0f0;
}

h1 {
    margin-bottom: 20px;
}

.board {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    grid-gap: 5px;
}

.cell {
    width: 100px;
    height: 100px;
    background-color: #fff;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 2em;
    cursor: pointer;
    border: 2px solid #000;
}

#resetButton {
    margin-top: 20px;
    padding: 10px 20px;
    font-size: 1em;
    cursor: pointer;
}
```

Code Copilot

By promptspellsmith.com

```javascript
const board = document.getElementById('board');
const cells = document.querySelectorAll('.cell');
const resetButton = document.getElementById('resetButton');
let currentPlayer = 'X';
let boardState = Array(9).fill(null);
let isGameOver = false;

const winningCombinations = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6]
];

function handleCellClick(event) {
    const index = event.target.getAttribute('data-index');

    if (boardState[index] !== null || isGameOver) return;

    boardState[index] = currentPlayer;
    event.target.textContent = currentPlayer;

    if (checkWin()) {
        alert(`Player ${currentPlayer} wins!`);
        isGameOver = true;
        return;
    }

    if (boardState.every(cell => cell !== null)) {
        alert('It\'s a tie!');
        isGameOver = true;
        return;
    }

    currentPlayer = currentPlayer === 'X' ? 'O' : 'X';
}

function checkWin() {
    return winningCombinations.some(combination => {
        return combination.every(index => {
            return boardState[index] === currentPlayer;
        });
    });
}

function resetGame() {
    boardState.fill(null);
    cells.forEach(cell => {
        cell.textContent = '';
    });
    currentPlayer = 'X';
    isGameOver = false;
}

cells.forEach(cell => cell.addEventListener('click', handleCellClick));
resetButton.addEventListener('click', resetGame);
```

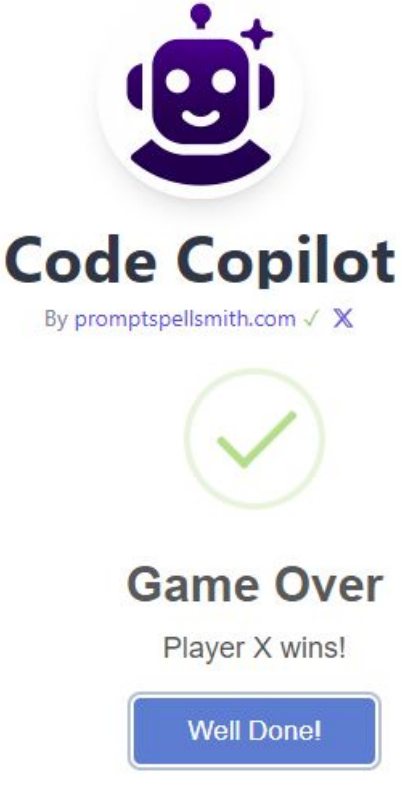
SweetAlert2

Next, with the assistance of Code Copilot, I replaced the regular popup box with SweetAlert2, a more visually appealing design.

SweetAlert2 is a beautiful, responsive, customizable, and accessible (WAI-ARIA) replacement for JavaScript popup boxes. It has zero dependencies.

On the right, you can see the JavaScript function, showAlert, which uses the swal method from the SweetAlert2 library to display a customized alert popup. This popup includes a custom message, title, button text, and backdrop color. It also ensures the alert cannot be dismissed by pressing the escape key or clicking outside the alert.

The code below is part of the game logic. After a player's move, it checks for a win or a tie condition. If a win or a tie is detected, the showAlert function is called to display the appropriate message.



sweetalert2

```
function showAlert(message) {
    swal({
        title: 'Game Over',
        text: message,
        type: 'success',
        confirmButtonText: 'Well Done!',
                allowEscapeKey: false,
                allowOutsideClick: false,
                backdrop: `
                    rgba(255,116,0,0.4)
                `
    });
}
```

```
if (checkWin()) {
    showAlert(`Player ${currentPlayer} wins!`);
    isGameOver = true;
    return;
}

if (boardState.every(cell => cell !== null)) {
    showAlert('Congratulations, it\'s a tie!');
    isGameOver = true;
    return;
}
```

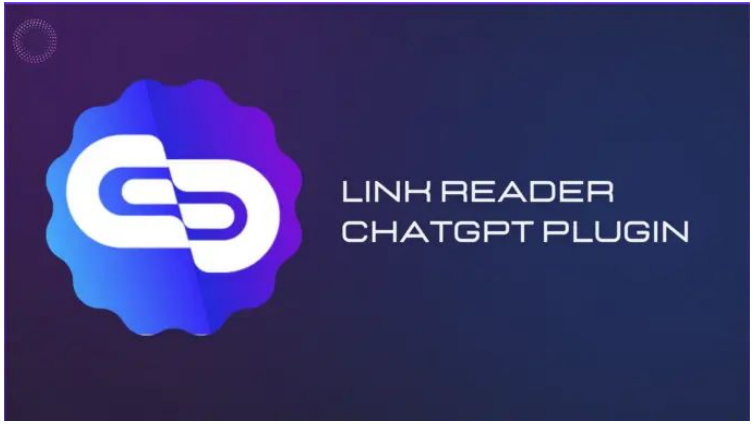
Code Copilot + Link Reader

This is the prompt I wrote on Code Copilot, using the pinned Link Reader to assist with integrating the SweetAlert2 library into my JavaScript code.

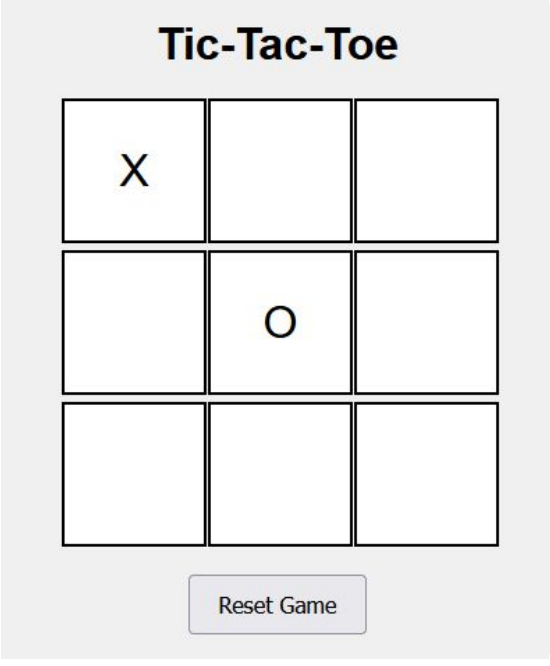
```
Here's the [HTML] code and [JavaScript] code for a simple tic-tac-toe game. When a player is declared a winner, they press the [resetButton] to play again.

I want you to input the code from these links:
- https://cdnjs.cloudflare.com/ajax/libs/limonte-sweetalert2/7.26.9/sweetalert2.min.css
- https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.2/jquery.min.js'>
</script
- https://cdnjs.cloudflare.com/ajax/libs/limonte-sweetalert2/7.26.9/sweetalert2.min.js

Create a [swal] function to make a pop-up window as soon as a winner is declared.
```

Code Copilot

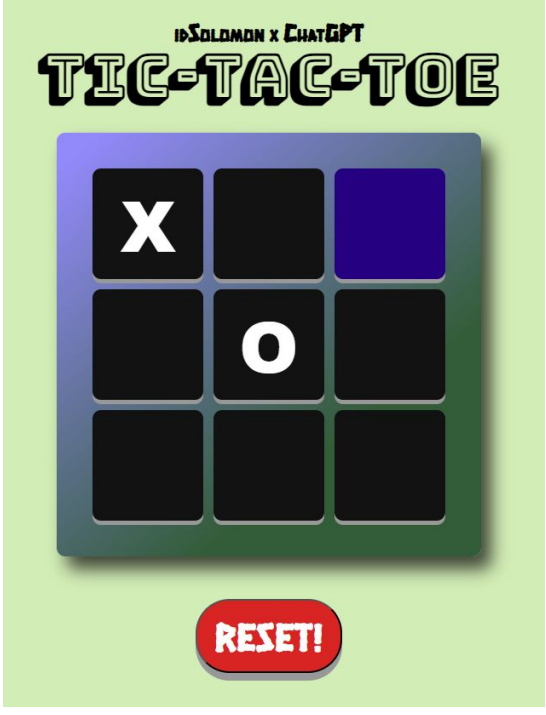
By promptspellsmith.com ✓ 𝕏



Before



After



The original version of the game had basic functionality but lacked a sophisticated design. Now, the game not only functions well but also looks visually appealing and aesthetically pleasing. I added the design elements myself without the aid of artificial intelligence. To see and interact with the project, you can go here at **codePen.io/idSolomon**.

Conclusion

I created this project to explore the capabilities of prompt engineering and ChatGPT's generative artificial intelligence.

For a small, simple project, I chose to develop a tic-tac-toe game. Initially, I used ChatGPT-4o to generate the basic HTML, CSS, and JavaScript code to assess the chatbot ability to produce functional, interactive web applications.

However, the initial code had limitations in functionality and visual appeal. By incorporating Code Copilot, the project advanced significantly, with AI-driven tools simplifying and accelerating the development process.

I also integrated SweetAlert2 to enhance the visual alerts, adding a user-friendly feature.

This project showcases the future potential of prompt engineering and artificial intelligence, highlighting their ability to streamline coding processes, improve productivity, and enhance user experience while driving innovation in the tech industry.