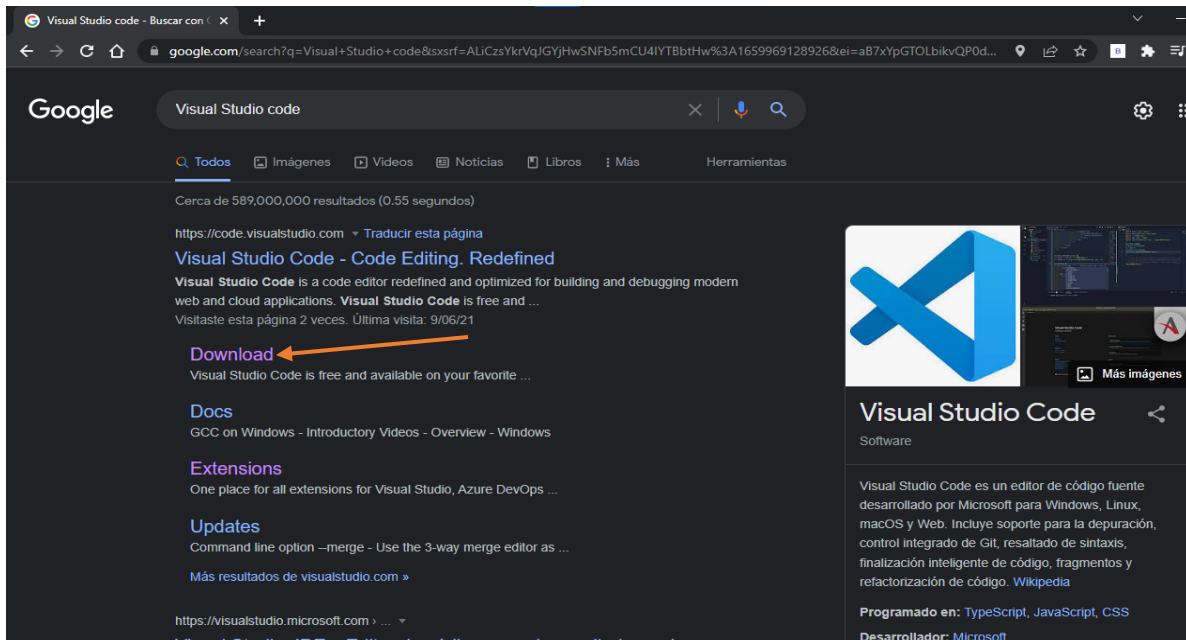
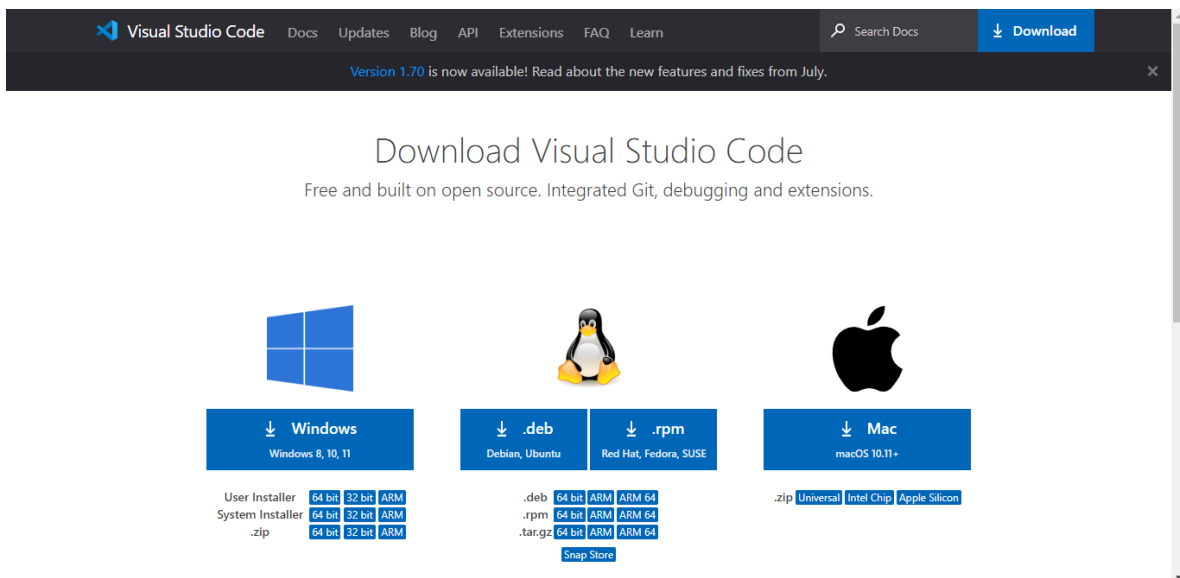


## Proceso instalación Visual Studio Code

**Paso 1:** Escribimos en Google Visual Studio Code y seleccionamos donde dice “Download”.



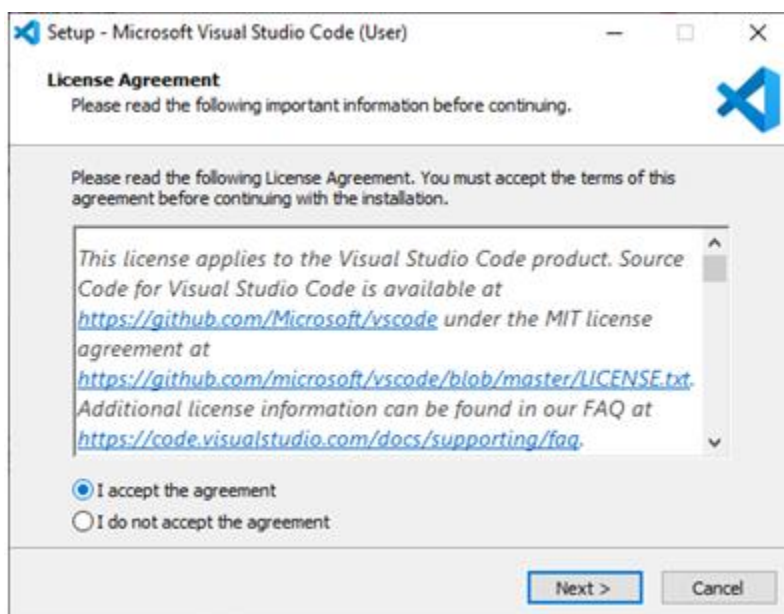
**Paso 2:** Seleccionamos el sistema operativo que tenemos y lo descargamos.



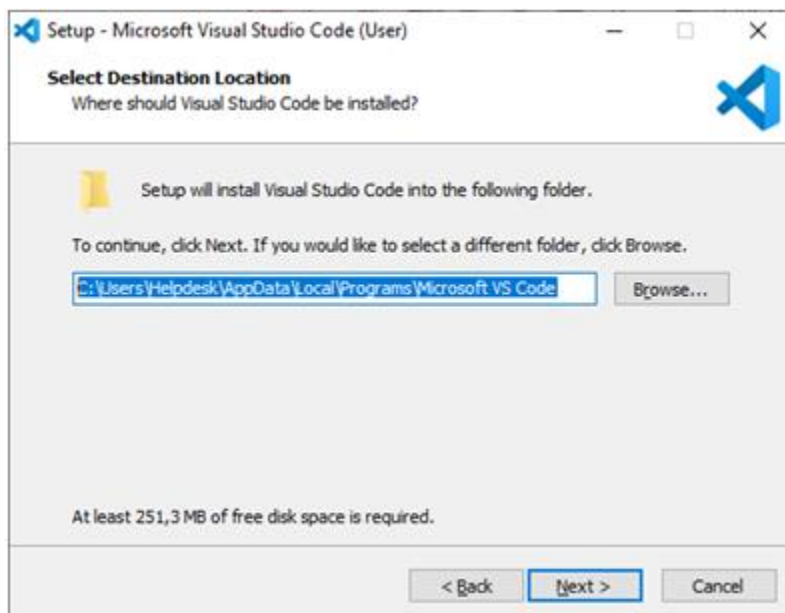
**Paso 3:** Al darle clic nos descargará un .exe, al cual le daremos clic encima.



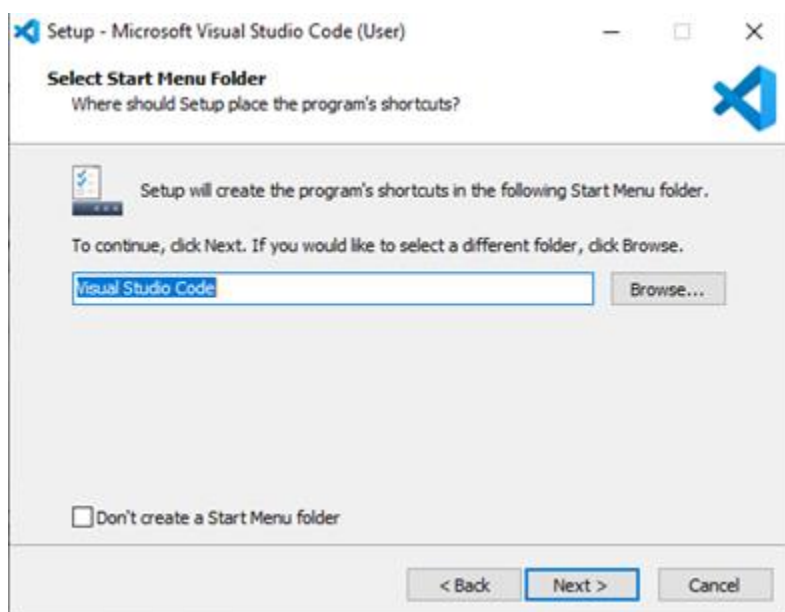
**Paso 4:** Lee y acepta el acuerdo de licencia. Haz clic en Next para continuar.



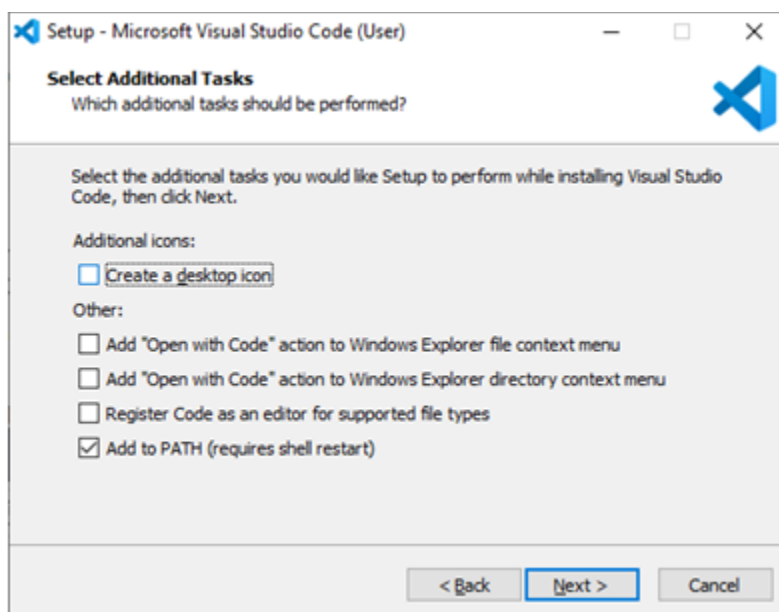
**Paso 5:** Puedes cambiar la ubicación de la carpeta de instalación o mantener la configuración predeterminada. Haz clic en Next para continuar.



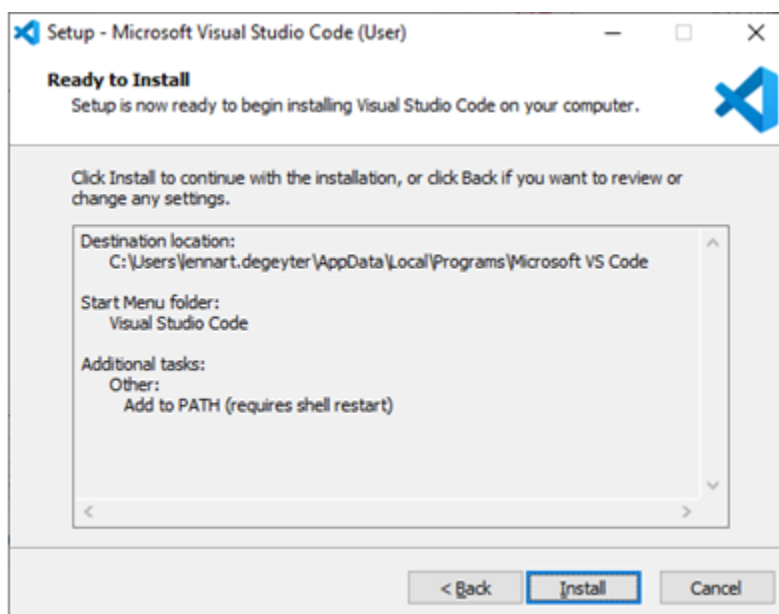
**Paso 6:** Elige si deseas cambiar el nombre de la carpeta de accesos directos en el menú Inicio o si no deseas instalar accesos directos en absoluto. Haz clic en Next.



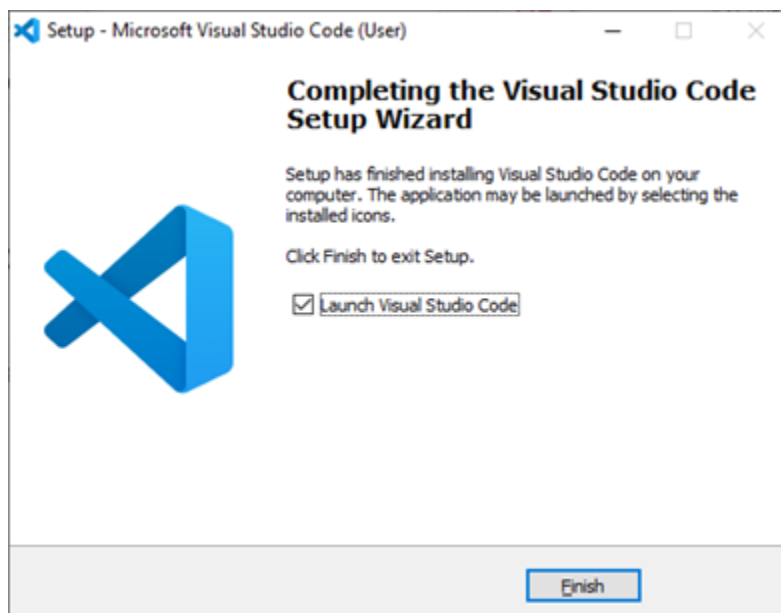
**Paso 7:** Selecciona las tareas adicionales, por ej. crear un icono en el escritorio o añadir opciones al menú contextual de Windows Explorer. Haz clic en Next.



**Paso 8:** Haz clic en Install para iniciar la instalación.

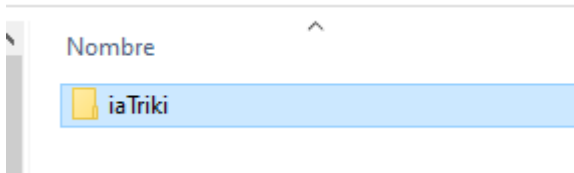


**Paso 9:** El programa está instalado y listo para usar. Haz clic en Finish para finalizar la instalación y lanzar el programa.



## Creación de archivos y carpetas

**Paso 1:** Creamos una carpeta raíz. En nuestro caso la llamaremos “iatriki”.

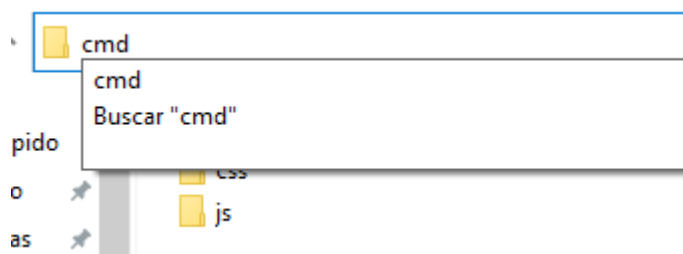


**Paso 2:** Dentro de esta, crearemos otras dos carpetas llamadas “css”, la cual contendrá todo lo estético de nuestra página y otra llamada “js” la cual contendrá la lógica de nuestra página.

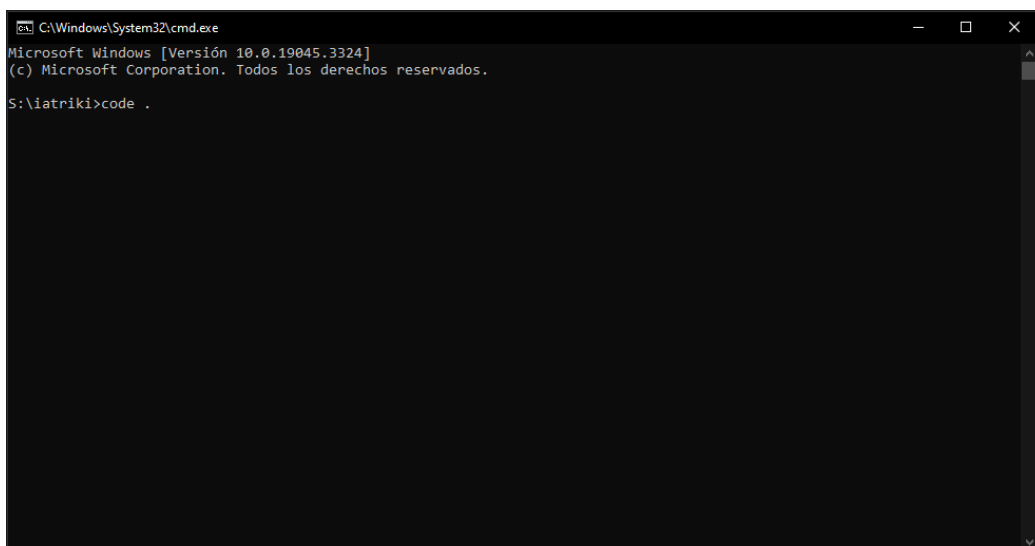


## Tarjeta

**Paso 1:** Para abrir el Visual Studio Code, haremos lo siguiente: Dentro de la carpeta raíz, en la barra superior, escribiremos cmd y le damos enter.



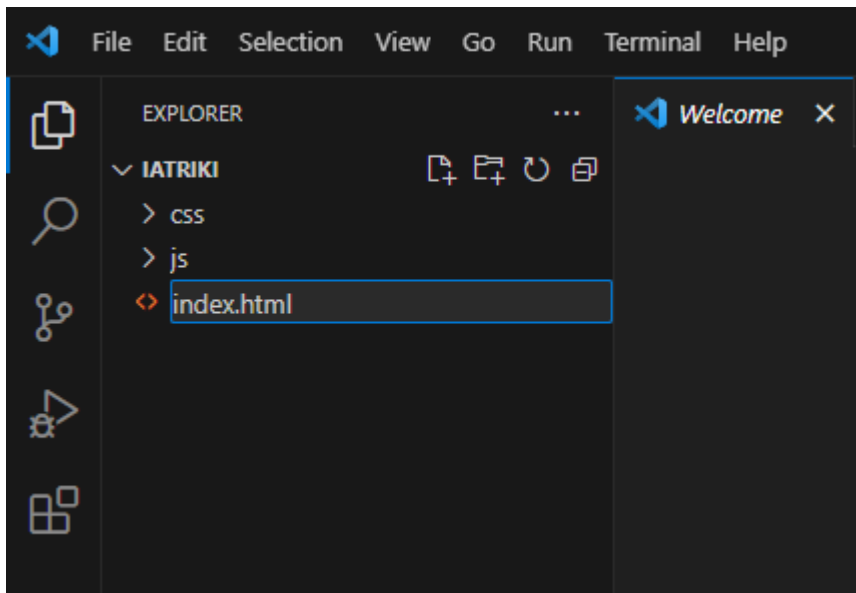
**Paso 2:** Eso nos abrirá una terminal, solo tendremos que escribir “code .”.



**Paso 3:** Esto nos abrirá el visual studio code. Ahora, ya que estamos dentro, crearemos un archivo llamado “index.html” dándole en la carpeta raíz dando click donde indica la flecha.

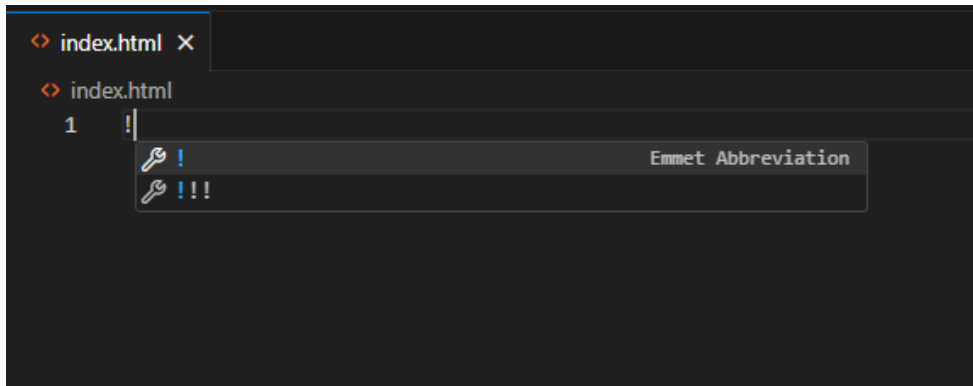


**Paso 4:** Llamaremos el archivo “index.html”



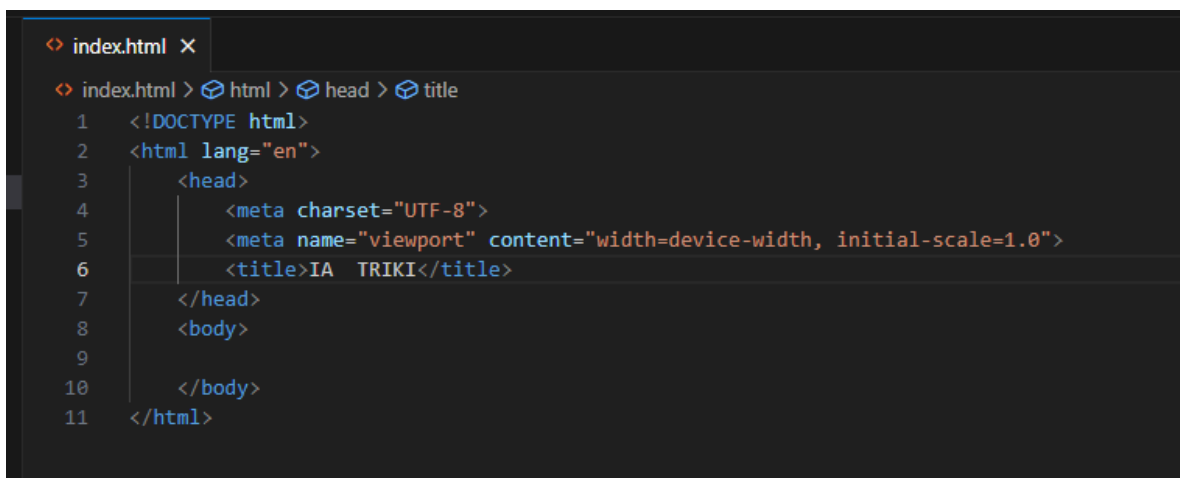


**Paso 5:** Una vez en el archivo recién creado, colocamos el código “!” y escogemos la primera opción.



```
index.html x
index.html
1  !|
   !
   !!!
   Emmet Abbreviation
```

**Paso 6:** Una vez colocada la estructura, cambiaremos el título (línea 6)



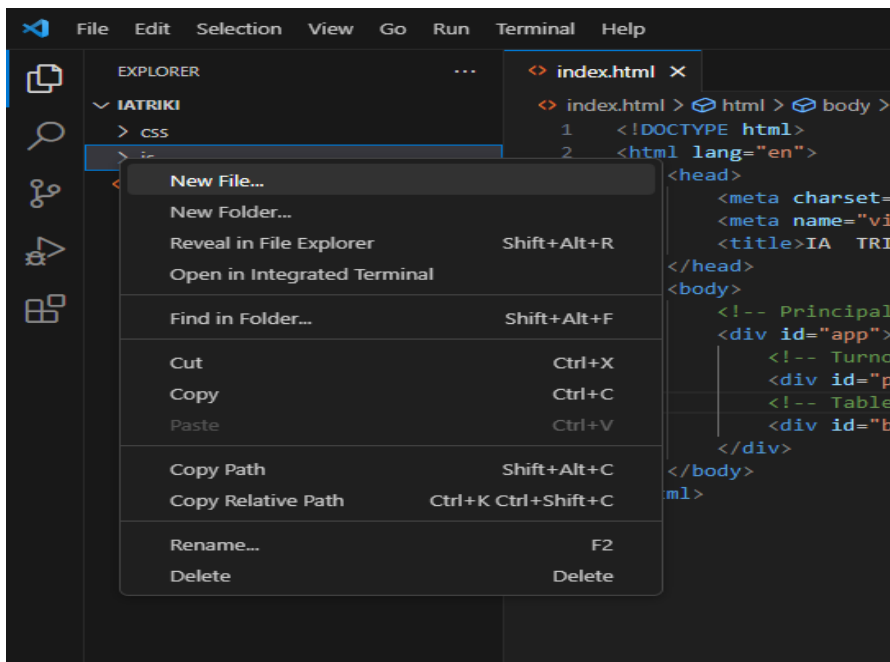
```
index.html x
index.html > html > head > title
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>IA TRIKI</title>
7    </head>
8    <body>
9
10   </body>
11 </html>
```

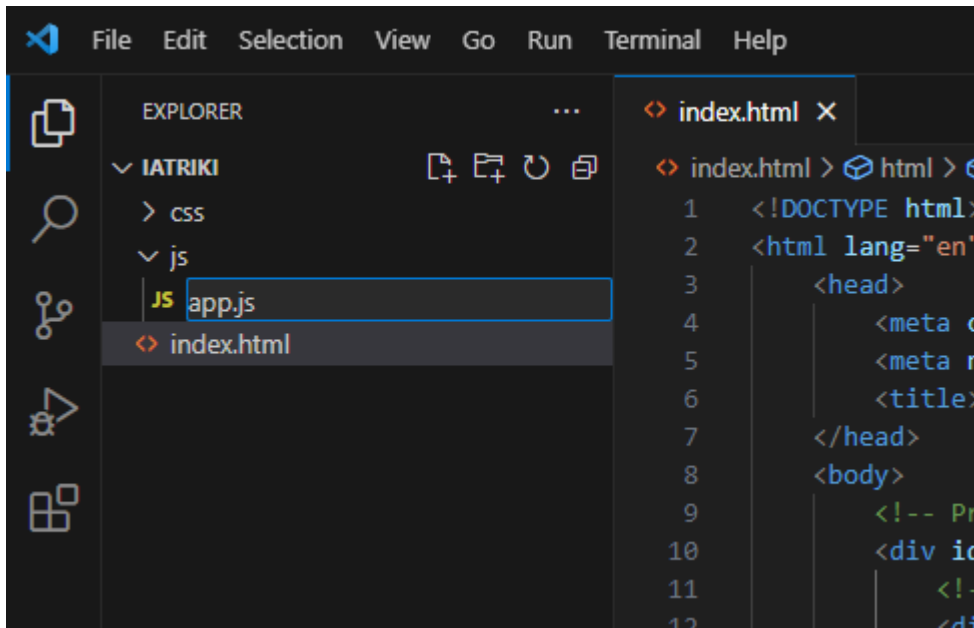
**Paso 7:** Ahora dentro del body crearemos 1 div principal para nuestra página (Línea 10) le pondremos un id llamado “app”, y además crearemos otro div dentro de ese mismo div (línea 12) y le pondremos el id “player” ya que será el que nos diga de quien es turno, y por ultimo otro div con el id “board” (línea 14), el cual tendrá nuestro tablero.



```
index.html X
index.html > html > body > div#app
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>IA TRIKI</title>
7   </head>
8   <body>
9     <!-- Principal -->
10    <div id="app">
11      <!-- Turno -->
12      <div id="player"></div>
13      <!-- Tablero de juego -->
14      <div id="board"></div>
15    </div>
16  </body>
17 </html>
```

**Paso 8:** Dentro la carpeta “js” vamos a crear un nuevo archivo llamado “app.js”

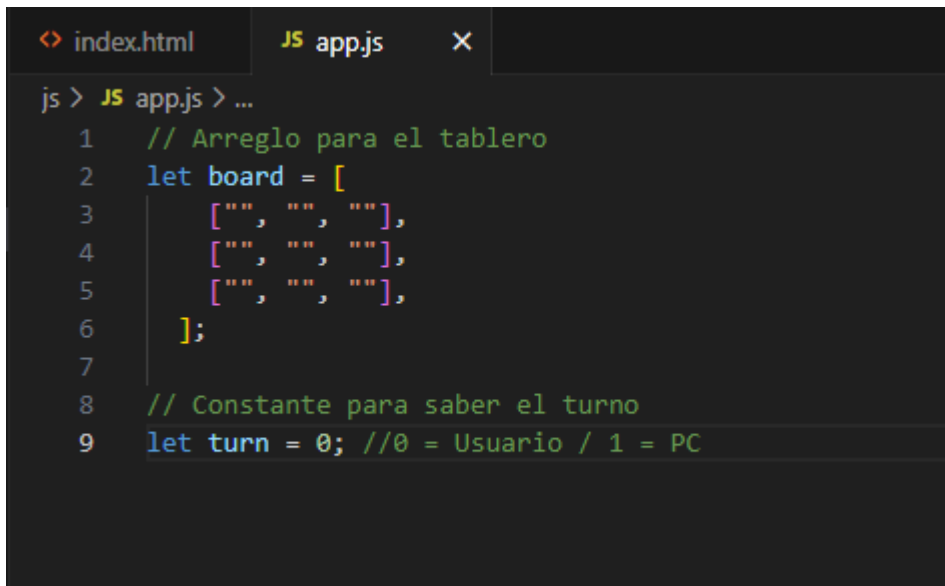




**Paso 9:** Ahora vamos a vincular nuestro javascript en el Html (línea 18)

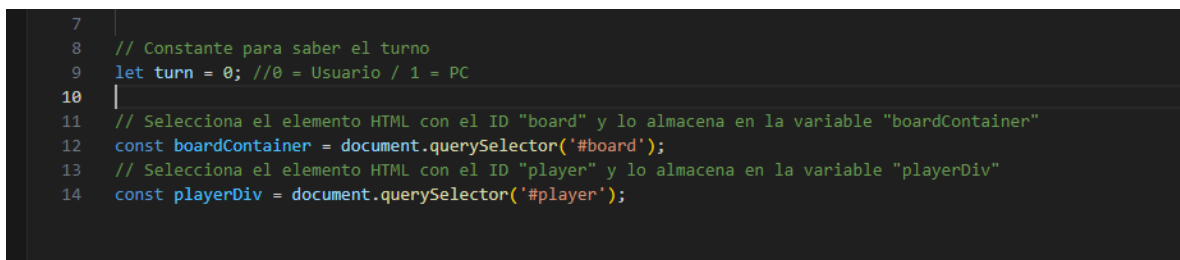


**Paso 10:** Una vez vinculado el javascript iremos a nuestro archivo “app.js” y crearemos un arreglo para el tablero (línea 2 hasta la 6) y crearemos otra variable llamada “turn” el cual servirá para saber de quién es el turno



```
index.html JS app.js X
js > JS app.js > ...
1 // Arreglo para el tablero
2 let board = [
3   ["", "", ""],
4   ["", "", ""],
5   ["", "", ""],
6 ];
7
8 // Constante para saber el turno
9 let turn = 0; //0 = Usuario / 1 = PC
```

**Paso 11:** Ahora vamos a seleccionar los elementos de nuestro html y los almacenaremos en variables con el siguiente código: const (nombre de la constante) = document.querySelector('#nombre de id');. Usamos el “#” porque los estamos capturando de un “id” capturamos el tablero (línea 12) y también el turno del jugador (línea 14)



```
7
8 // Constante para saber el turno
9 let turn = 0; //0 = Usuario / 1 = PC
10
11 // Selecciona el elemento HTML con el ID "board" y lo almacena en la variable "boardContainer"
12 const boardContainer = document.querySelector('#board');
13 // Selecciona el elemento HTML con el ID "player" y lo almacena en la variable "playerDiv"
14 const playerDiv = document.querySelector('#player');
```

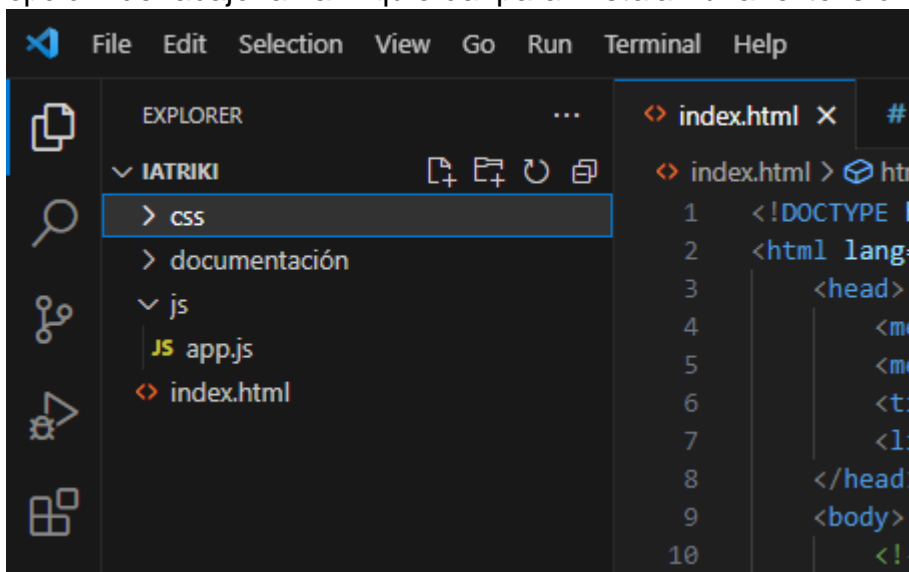
**Paso 12:** Ahora vamos a crear una función para renderizar el tablero. La llamaremos “renderBoard”, y vamos a mapear las celdas(línea 19) y crear un botón por cada celda (línea 21 a 23) y vamos a guardar esas celdas en una constante llamada “cells” después de crear los botones para cada celda en la fila, se utiliza el método join("") para unir todos los botones en un solo string (línea 25), y por ultimo actualizamos nuestro tablero del html con la información que le acabamos de poner. (línea 28).

```
16 // Definición de una función llamada "renderBoard" que renderiza el tablero en HTML
17 function renderBoard() {
18   // Recorre el array con .map
19   const html = board.map((row) => {
20     // Mapea cada celda en la fila actual y crea un botón para representarla
21     const cells = row.map((cell) => {
22       return `<button class="cell">${cell}</button>`;
23     });
24     // Une las celdas de la fila actual y crea una fila completa con ellas
25     return `<div class="row">${cells.join("")}</div>`;
26   });
27   // Establece el contenido HTML del contenedor del tablero con las filas generadas
28   boardContainer.innerHTML = html.join("");
29 }
```

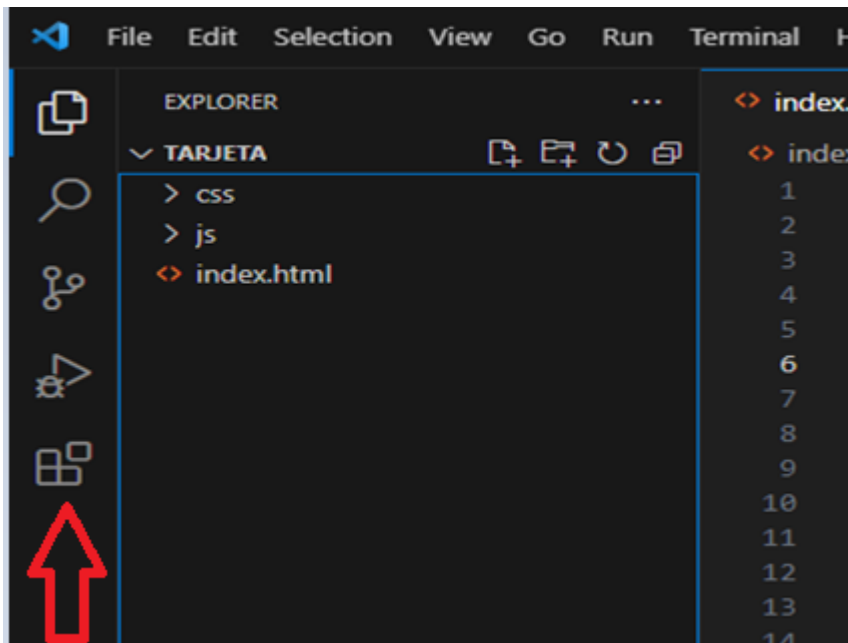
**Paso 13:** Ahora llamamos la función que acabamos de crear.

```
28   boardContainer.innerHTML = html.join("");
29 }
30 //Llamar funcion "renderBoard"
31 renderBoard();
32
```

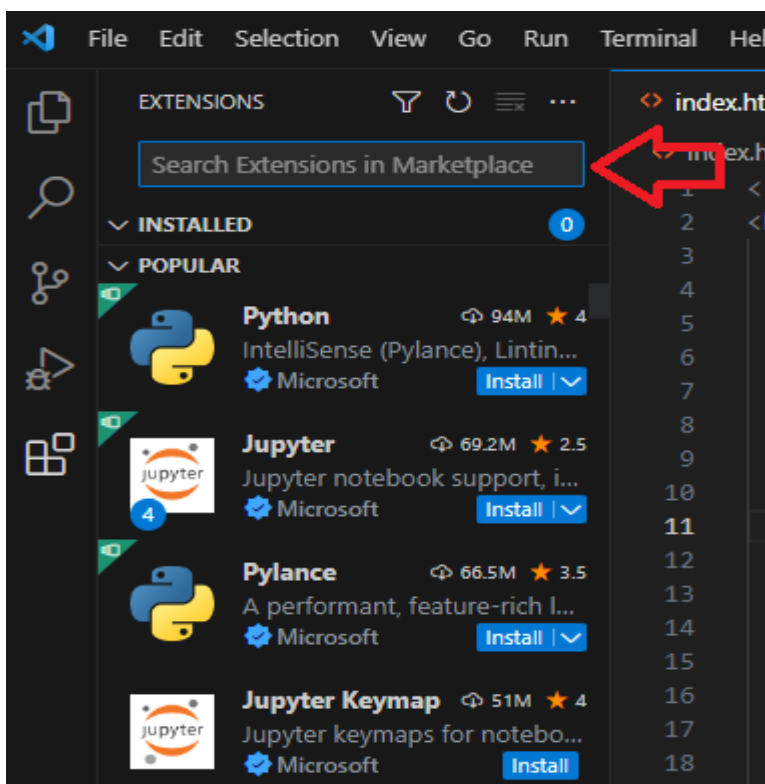
**Paso 14:** Vamos a visualizar cómo está quedando para esto necesitamos ir a la opción de abajo a la izquierda para instalar una extensión del visual studio.



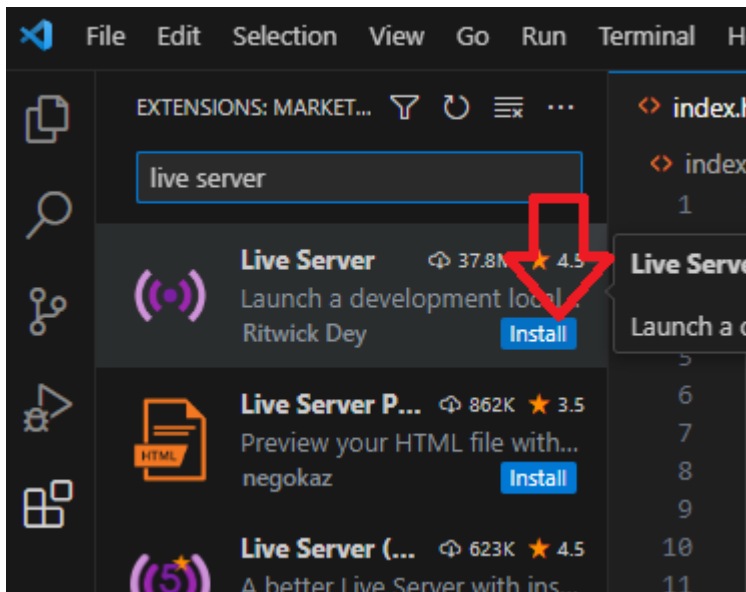
**Paso 15:** Ahora para poder visualizar como se esta viendo nuestro proyecto le daremos click al icono de abajo a la izquierda.



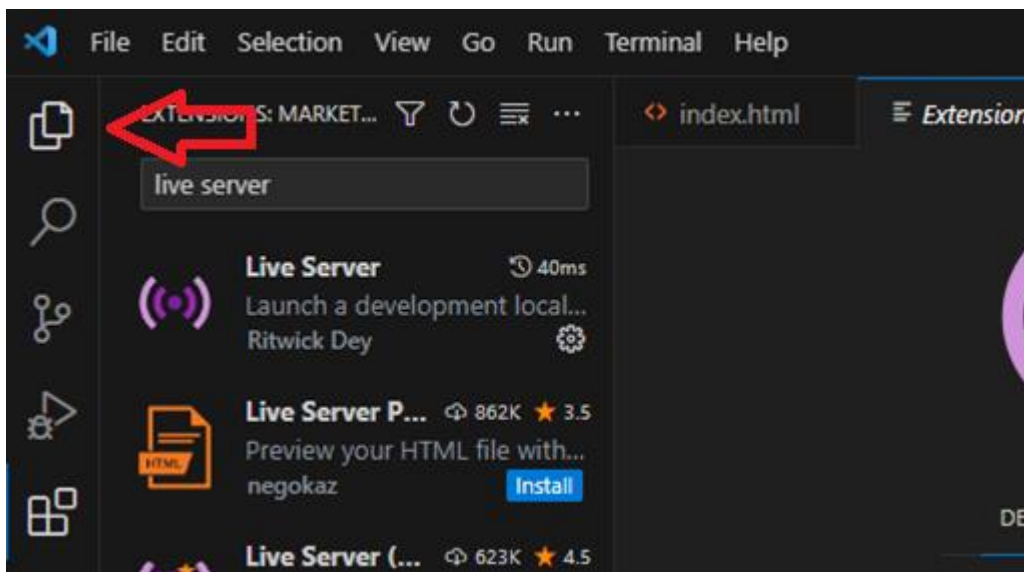
**Paso 16:** Nos saldrá esta pestaña y buscaremos live server, en la barra de búsqueda



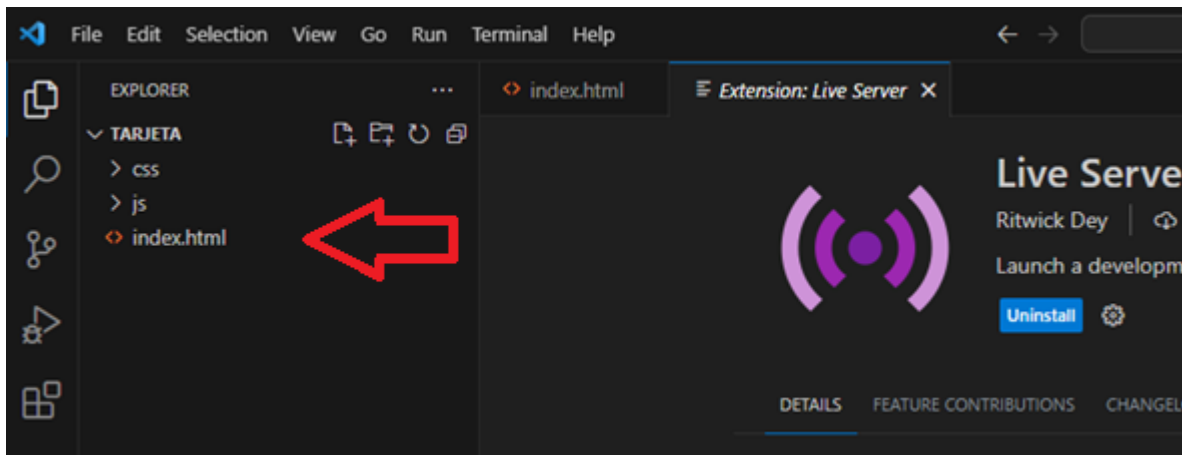
**Paso 17:** Ahora que encontramos lo que necesitamos, le damos click a “install”



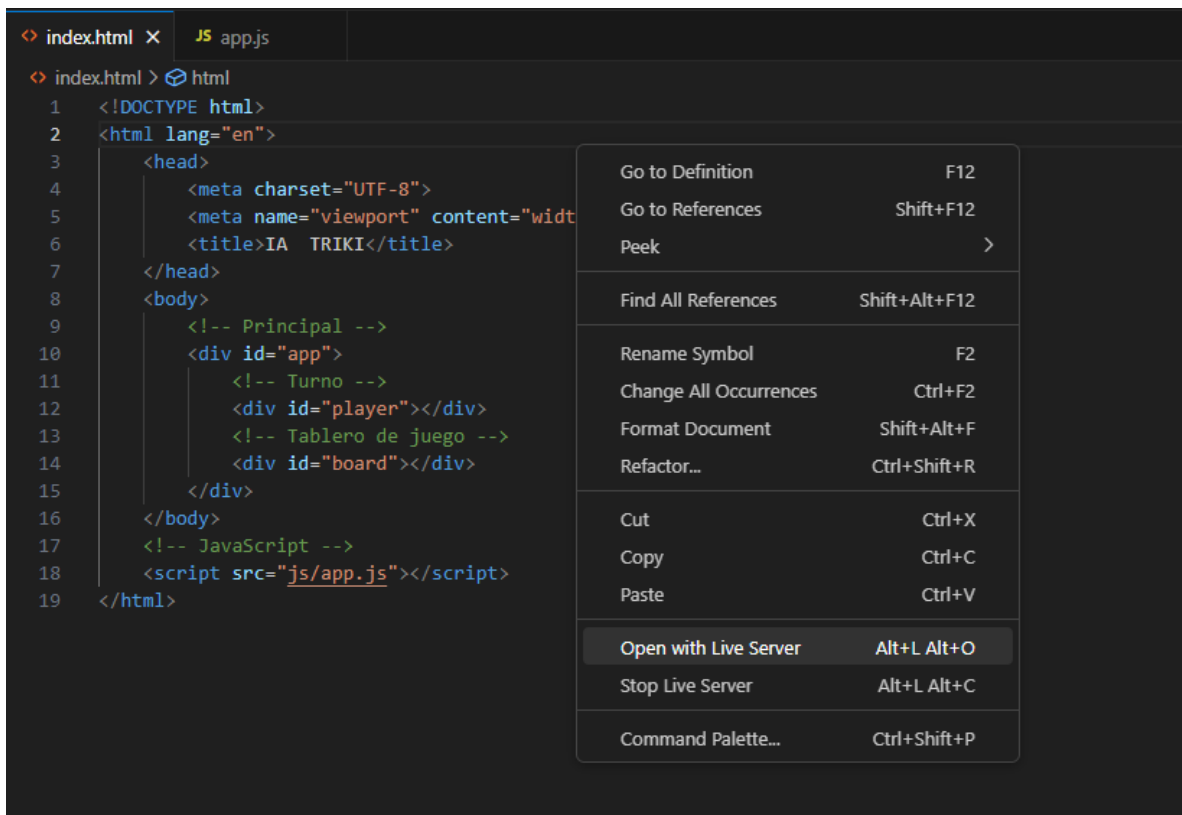
**Paso 18:** Ahora que tenemos instalado el live server nos debería de aparecer algo así donde podremos volver a nuestro proyecto dando click al explorador.



**Paso 19:** Una vez te sale el explorador volvemos al Index.html



**Paso 20:** Ahora vamos a dar click derecho en cualquier parte del código y damos click en la opción "Open with live server"

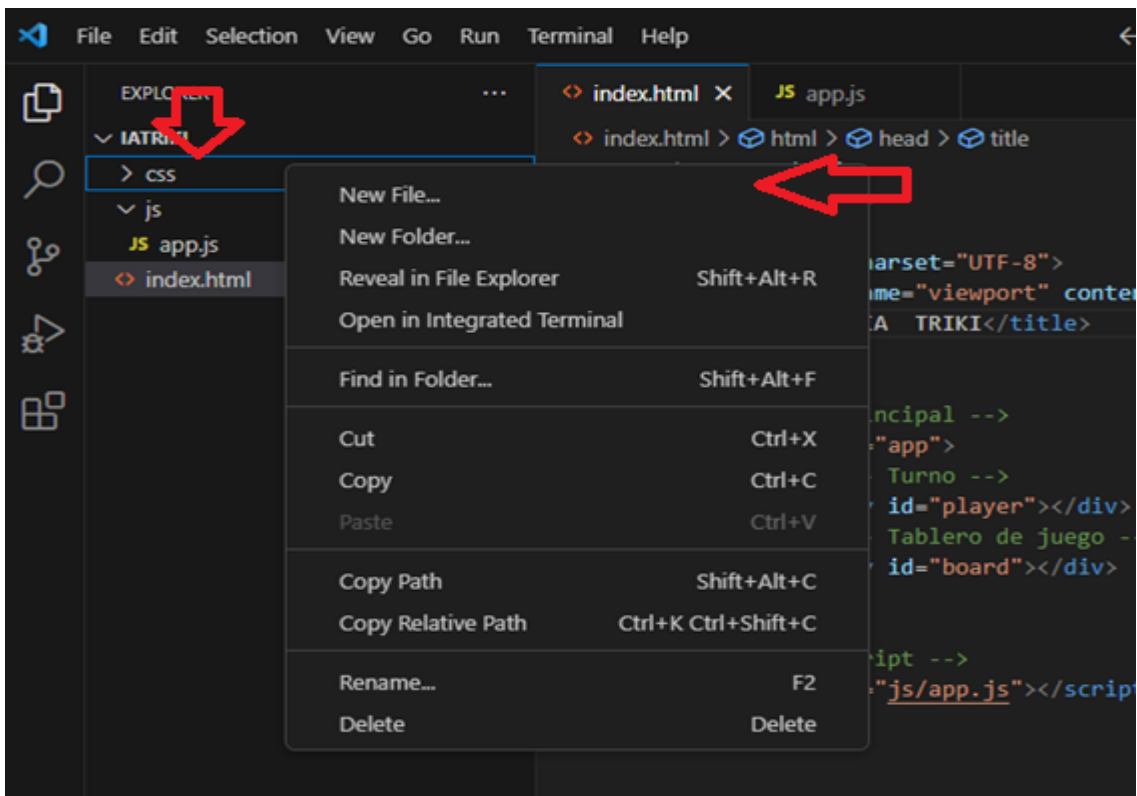




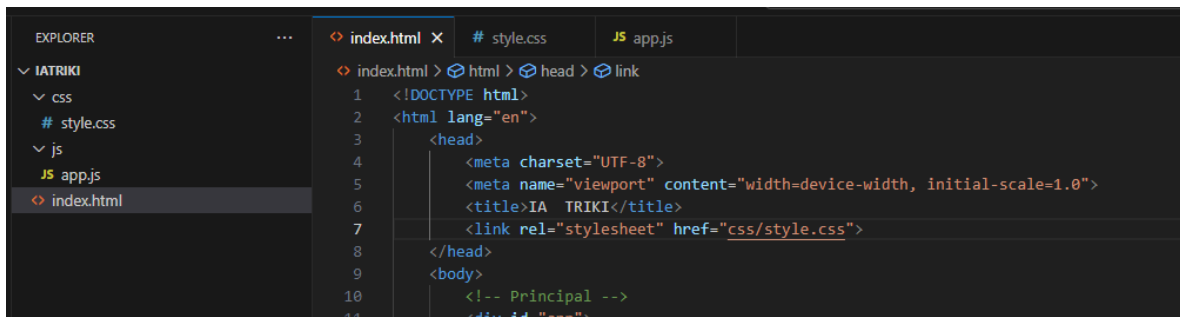
**Paso 21:** Así está quedando nuestro pequeño proyecto, podemos mirar que ya esta generado el tablero. Ahora vamos a meterle diseño.



**Paso 22:** Lo primero que haremos será volver a nuestro código y vamos a crear un nuevo archivo dentro de la carpeta css, lo llamaremos "style.css"

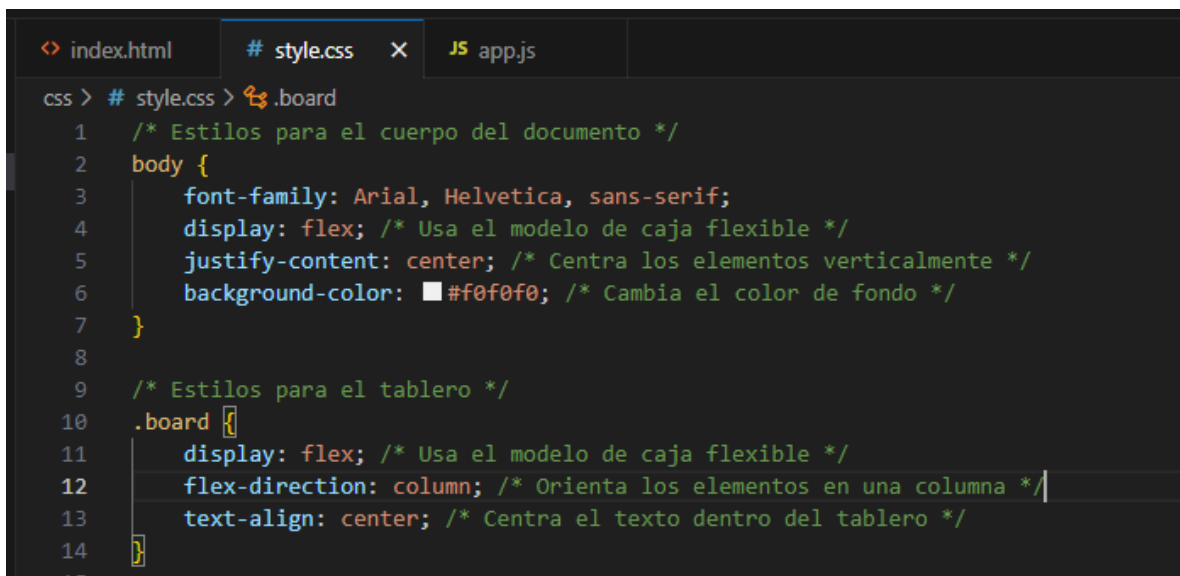


**Paso 23:** Una vez creado el archivo css vamos a vincularlo a nuestro html (línea 7).



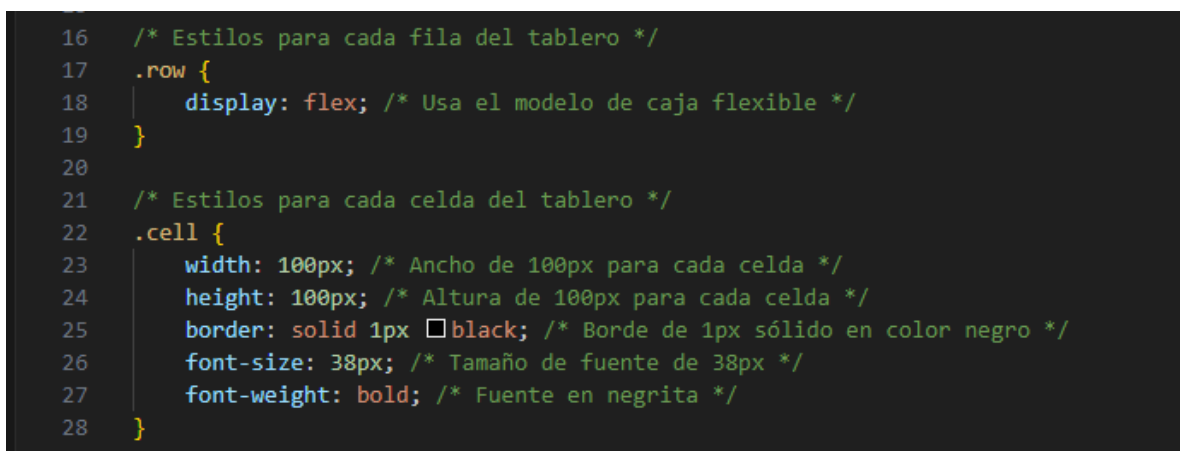
```
EXPLORER
  IATRIKI
    css
      # style.css
    js
      app.js
      index.html
  index.html X # style.css JS app.js
  index.html > html > head > link
  1 <!DOCTYPE html>
  2 <html lang="en">
  3   <head>
  4     <meta charset="UTF-8">
  5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
  6     <title>IA TRIKI</title>
  7     <link rel="stylesheet" href="css/style.css">
  8   </head>
  9   <body>
  10     <!-- Principal -->
  11     <div id="app">
```

**Paso 24:** Comenzamos con el diseño del body para centrar su contenido y también le ponemos una fuente. También diseñamos el tablero con él “.board” centrando su texto.



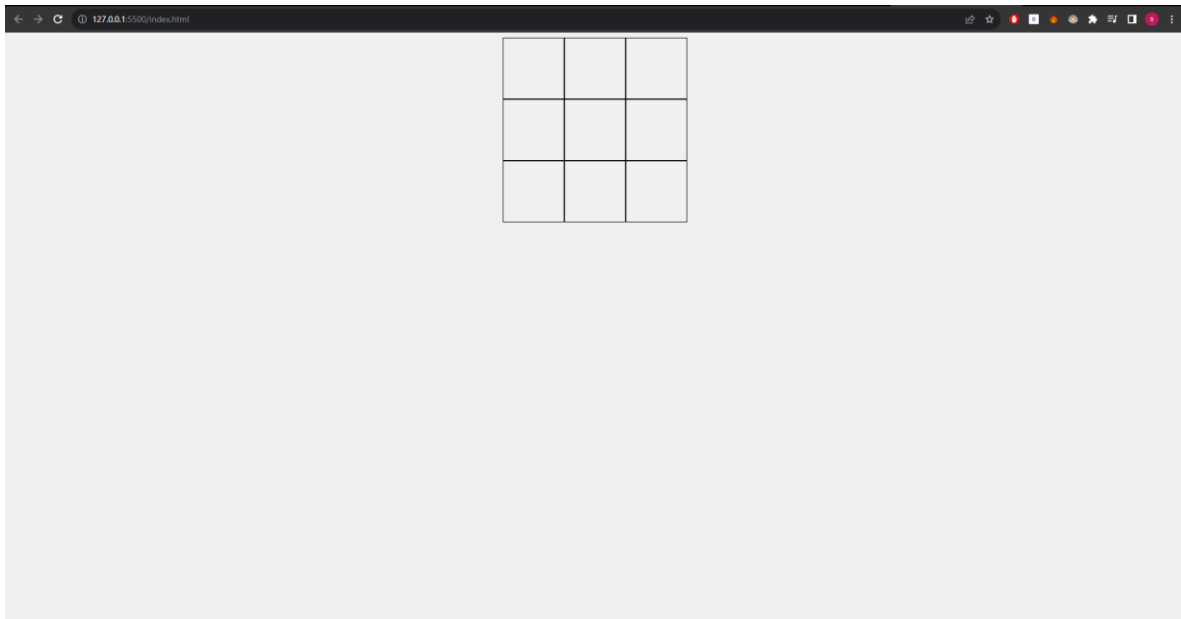
```
css > # style.css > .board
1 /* Estilos para el cuerpo del documento */
2 body {
3   font-family: Arial, Helvetica, sans-serif;
4   display: flex; /* Usa el modelo de caja flexible */
5   justify-content: center; /* Centra los elementos verticalmente */
6   background-color: #f0f0f0; /* Cambia el color de fondo */
7 }
8
9 /* Estilos para el tablero */
10 .board {
11   display: flex; /* Usa el modelo de caja flexible */
12   flex-direction: column; /* Orienta los elementos en una columna */
13   text-align: center; /* Centra el texto dentro del tablero */
14 }
15
```

**Paso 25:** Ahora ponemos el “display: flex;” a la filas del tablero, y también cambiamos el diseño de las celdas para ponerle un tamaño de fuente y ponerlo en negrita.



```
16 /* Estilos para cada fila del tablero */
17 .row {
18   display: flex; /* Usa el modelo de caja flexible */
19 }
20
21 /* Estilos para cada celda del tablero */
22 .cell {
23   width: 100px; /* Ancho de 100px para cada celda */
24   height: 100px; /* Altura de 100px para cada celda */
25   border: solid 1px black; /* Borde de 1px sólido en color negro */
26   font-size: 38px; /* Tamaño de fuente de 38px */
27   font-weight: bold; /* Fuente en negrita */
28 }
```

**Paso 26:** Ahora miramos como está quedando nuestro proyecto (paso 19).



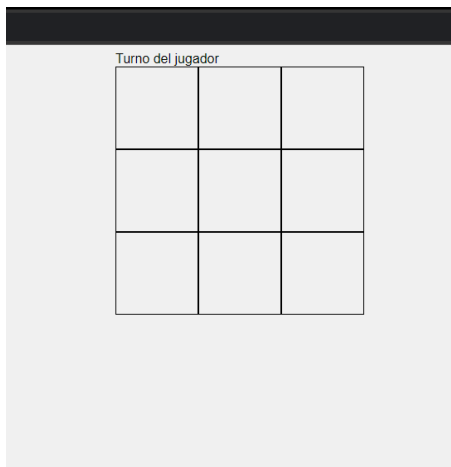
**Paso 27:** Una vez miramos como está quedando nuestro proyecto vamos a crear una nueva función en el archivo “app.js” y llamamos una función llamada “startGame” y la creamos además cuando llamamos la función “renderBoard” (Paso 13), vamos a meterlo en la nueva función, y llamamos la función ahí (línea 37), también llamaremos la constante “turn” y vamos a generar un numero aleatorio entre el 0 y el 1 para que se genere el turno (línea 39) y vamos a llamar una función llamada “renderCurrentPlayer” (línea 41) la cual va a mostrar arriba del tablero de quien es turno, y por ultimo crearemos una condicional (línea 43) la cual en caso de que la constante “turn” sea igual a 0 llame a la función para que el jugador pueda jugar, de lo contrario llama a la función para que la computadora juegue.

```
29 }
30
31 // llamada a la función que inicia el juego
32 startGame();
33
34 // Función para iniciar el juego
35 function startGame() {
36     // Renderiza el tablero en la interfaz
37     renderBoard();
38     // Asigna el turno aleatoriamente: 0 para el jugador, 1 para la computadora
39     turn = Math.random() <= 0.5 ? 0 : 1;
40     // Renderiza el jugador actual en la interfaz
41     renderCurrentPlayer();
42     // Si es el turno del jugador
43     if (turn === 0) {
44         // El jugador realiza su jugada
45         playerPlays();
46     } else {
47         // La computadora realiza su jugada
48         PCPlays();
49     }
50 }
51
```

**Paso 28:** Ahora vamos a crear la función que mencione anteriormente llamada “renderCurrentPlayer” la cual lo único que hará será cambiar nuestro texto de arriba del tablero para que muestre de quien es turno (línea 55)

```
50 }
51
52 // Función para renderizar el jugador actual en la interfaz
53 function renderCurrentPlayer() {
54   // Actualiza el contenido del elemento con ID "player" según el turno actual
55   playerDiv.textContent = `${
56     turn === 0 ? "Turno del jugador" : "Turno del computador"
57   }`;
58 }
59
```

**Paso 29:** Ahora aleatoriamente al cargar la página (paso 19) va a colocar arriba quien comienza a jugar.



**Paso 30:** Ahora crearemos la función del jugador.

Lo primero será crear la constante “cells” y agarrará todos los elementos con la clase.” cell” dentro del tablero (línea 63), ahora creamos un foreach que calcule la fila y la celda actual basándose en su índice (línea 65 a 69).

```
58 }
59
60 //Función que permite al jugador hacer una jugada
61 function playerPlays() {
62   // Selecciona todos los elementos con la clase "cell" (celdas del tablero)
63   const cells = document.querySelectorAll(".cell");
64   // Calcula la fila y la columna de la celda actual basandose en su indice
65   cells.forEach((cell, i) => {
66     const row = Math.floor(i / 3); // Obtiene el índice de la fila
67     const column = i % 3; // Obtiene el índice de la columna
68   });
69 }
70
```

**Paso 31:** Ahora vamos a crear una condicional de que en el caso de que la columna y la fila este vacia, añada un evento “click” y que coloque un “O” y que se muestre en el tablero (línea 73), cambiamos la constante “turn” a 1 y llamamos la función que hace que pueda jugar el computador

```
65 cells.forEach((cell, i) => {
66   const row = Math.floor(i / 3); // Obtiene el índice de la fila
67   const column = i % 3; // Obtiene el índice de la columna
68
69   if (board[row][column] === "") {
70     cell.addEventListener("click", (e) => {
71       // Si la celda está vacía
72       board[row][column] = "O"; // Marca la celda como ocupada por el jugador
73       cell.textContent = board[row][column]; // Actualiza la visualización en la interfaz
74       // Cambia el turno al de la computadora
75       turn = 1;
76       //Llama la función "PCPlays"
77       PCPlays();
78     });
79   }
80 });
81 }
```

**Paso 32:** Ahora podremos mirar que podemos ir añadiendo nuestros símbolos en nuestro turno.

|                   |   |  |
|-------------------|---|--|
| Turno del jugador |   |  |
| O                 | O |  |
|                   | O |  |
|                   |   |  |

**Paso 33:** Ahora creamos una función para que en caso de empate se muestre.

```
81 }
82
83 // Función para renderizar un empate en la interfaz
84 function renderDraw() {
85   playerDiv.textContent = "Empate"; // Muestra el mensaje de empate en el elemento HTML con ID "player"
86 }
87
```

**Paso 34:** Ahora vamos a comenzaremos a crear la inteligencia de nuestra máquina.

Comenzamos creando una función llamada “checkIfCanWin” y crearemos una constante llamada “arr” que usando el método JSON.parse(), el cual se utiliza para analizar una cadena JSON y crear un objeto JavaScript a partir de ella. Y dentro usamos el método “JSON.stringify” el cual se utiliza para convertir un objeto (en este caso, el arreglo board) en una cadena JSON. Esto toma todos los elementos del arreglo, los convierte en su representación JSON y los concatena en una cadena.

```
87
88 // Función para verificar si un jugador puede ganar en el próximo movimiento
89 function checkIfCanWin() {
90   // Crea una copia del tablero para realizar cálculos sin modificar el tablero original
91   const arr = JSON.parse(JSON.stringify(board));
92 }
93
```

**Paso 35:** Ahora crearemos un for para que itere por todas las casillas y les de un valor según si la casilla tiene una “X”, “O” o este vacía.

En caso de que la casilla tenga una “X” va a darle un valor de 1. (Línea 96 y 97)

En caso de que la casilla este vacía va a darle un valor de 0. (Línea 98 y 99)

En caso de que la casilla tenga una “O” va a darle un valor de -2. (Línea 100 y 101)

```
88 // Función para verificar si un jugador puede ganar en el próximo movimiento
89 function checkIfCanWin() {
90   // Crea una copia del tablero para realizar cálculos sin modificar el tablero original
91   const arr = JSON.parse(JSON.stringify(board));
92
93   // Convierte las celdas marcadas con "X", "O" o vacías en objetos con valor y coordenadas
94   for (let i = 0; i < arr.length; i++) {
95     for (let j = 0; j < arr[i].length; j++) {
96       if (arr[i][j] === "X") {
97         arr[i][j] = { value: 1, i, j }; // Casilla ocupada por "X"
98       } else if (arr[i][j] === "") {
99         arr[i][j] = { value: 0, i, j }; // Casilla vacía
100       } else if (arr[i][j] === "O") {
101         arr[i][j] = { value: -2, i, j }; // Casilla ocupada por "O"
102       }
103     }
104   }
105 }
```

**Paso 36:** Ahora crearemos constantes para saber la ubicación de cada casilla del tablero.

```
104     }
105
106     // Define todas las combinaciones posibles para ganar
107     const p1 = arr[0][0]; // Esquina superior izquierda
108     const p2 = arr[0][1]; // Borde superior
109     const p3 = arr[0][2]; // Esquina superior derecha
110     const p4 = arr[1][0]; // Borde izquierdo
111     const p5 = arr[1][1]; // Centro
112     const p6 = arr[1][2]; // Borde derecho
113     const p7 = arr[2][0]; // Esquina inferior izquierda
114     const p8 = arr[2][1]; // Borde inferior
115     const p9 = arr[2][2]; // Esquina inferior derecha
116 }
```

**Paso 37:** Crearemos arrays para cada forma de ganar en el juego.

```
115     const p9 = arr[2][2]; // Esquina inferior derecha
116
117     const s1 = [p1, p2, p3]; // Fila superior
118     const s2 = [p4, p5, p6]; // Fila central
119     const s3 = [p7, p8, p9]; // Fila inferior
120     const s4 = [p1, p4, p7]; // Columna izquierda
121     const s5 = [p2, p5, p8]; // Columna central
122     const s6 = [p3, p6, p9]; // Columna derecha
123     const s7 = [p1, p5, p9]; // Diagonal principal izquierda a derecha
124     const s8 = [p3, p5, p7]; // Diagonal inversa izquierda a derecha
125 }
126
```

**Paso 38:** Así debe de estar.

```
104 }
105
106 // Define todas las combinaciones posibles para ganar
107 const p1 = arr[0][0]; // Esquina superior izquierda
108 const p2 = arr[0][1]; // Borde superior
109 const p3 = arr[0][2]; // Esquina superior derecha
110 const p4 = arr[1][0]; // Borde izquierdo
111 const p5 = arr[1][1]; // Centro
112 const p6 = arr[1][2]; // Borde derecho
113 const p7 = arr[2][0]; // Esquina inferior izquierda
114 const p8 = arr[2][1]; // Borde inferior
115 const p9 = arr[2][2]; // Esquina inferior derecha
116
117 const s1 = [p1, p2, p3]; // Fila superior
118 const s2 = [p4, p5, p6]; // Fila central
119 const s3 = [p7, p8, p9]; // Fila inferior
120 const s4 = [p1, p4, p7]; // Columna izquierda
121 const s5 = [p2, p5, p8]; // Columna central
122 const s6 = [p3, p6, p9]; // Columna derecha
123 const s7 = [p1, p5, p9]; // Diagonal principal izquierda a derecha
124 const s8 = [p3, p5, p7]; // Diagonal inversa izquierda a derecha
125 }
```

**Paso 39:** Ahora crearemos una constante que mire que filtre todas las combinaciones para encontrar las que llevan a la victoria (línea 127) y va a sumar los valores y que en caso de que el valor sumado sea “2” significa que es 1 + 1 (Paso 34) y significa que podrá ganar y pondrá el movimiento en la casilla para ganar (línea 131) o si por el contrario la suma es “-4” ósea -2 -2, significa que el jugador va a ganar y por ello la maquina lo bloqueará (línea 132)

Y por último retorna las combinaciones (línea 136)

```
124 const s8 = [p3, p5, p7]; // Diagonal inversa izquierda a derecha
125
126 // Filtra las combinaciones para encontrar las que puedan llevar a una victoria
127 const res = [s1, s2, s3, s4, s5, s6, s7, s8].filter((line) => {
128   // Comprueba si la suma de los valores en la línea es igual a 2 (Para hacer su jugada y poder ganar)
129   // o igual a -4 (Para bloquear el movimiento del jugador y que no gane)
130   return (
131     line[0].value + line[1].value + line[2].value === 2 ||
132     line[0].value + line[1].value + line[2].value === -4
133   );
134 });
135
136 return res; // Retorna las combinaciones que pueden llevar a la victoria
137 }
138
```



**Paso 40:** Ahora comenzamos con la función del movimiento de la computadora con una llamándola “PCPlays” y llamaremos la función “renderCurrentPlayer” para que se muestre que es turno de la máquina.

Y vamos a colocar un “setTimeout” para el tiempo que la máquina tendrá para jugar.

Dentro colocaremos una variable que controla si la computadora ya jugó o no (línea 142) y otra para comprobar si la computadora puede ganar en su próximo movimiento gracias a la función que antes creamos (línea 143)

```
139 function PCPlays() {  
140     renderCurrentPlayer(); // Actualiza el jugador actual en la interfaz  
141     setTimeout(() => {  
142         let played = false; // Variable para controlar si la computadora realizó una jugada  
143         const options = checkIfCanWin(); // Comprueba si la computadora puede ganar en el próximo movimiento  
144     }, 1500);  
145 }  
146  
147
```

**Paso 41:** Ahora crearemos una condicional (línea 146) y mirará si hay opciones para ganar. En caso de que haya va a escoger la mejor opción (línea 147)

Creando un “for” (línea 148) que en caso de que el valor de la mejor opción sea “0” (paso 34) ó sea este vacío, ubique su posición en la fila (línea 150) y en la columna (línea 151) y coloque una “X” (línea 152) y también cambiará la constante (played) a true para que se sepa que ya la maquina jugó(línea 153), además saldrá del bucle (línea 154) y por último, colocamos lo que tardará la maquina en hacer todo este proceso (línea 158)

```
143     const options = checkIfCanWin(); // Comprueba si la computadora puede ganar en el próximo movimiento  
144  
145     // Si hay opciones de ganar disponibles  
146     if (options.length > 0) {  
147         const bestOption = options[0]; // Escoge la mejor opción de ganar  
148         for (let i = 0; i < bestOption.length; i++) {  
149             if (bestOption[i].value === 0) {  
150                 const posi = bestOption[i].i;  
151                 const posj = bestOption[i].j;  
152                 board[posi][posj] = "X"; // Realiza la jugada  
153                 played = true; // Marca como jugada realizada  
154                 break; // Sale del bucle  
155             }  
156         }  
157     }  
158     setTimeout(() => {  
159         // Espera 1500 milisegundos antes de que la computadora realice su jugada  
160     }, 1500);  
161 }
```

**Paso 42:** De lo contrario si no hay opción buena realiza una jugada aleatoria y va a colocar una “X” y cambiará su estado a que ya jugo (línea 163)

```
157     } else {  
158         // Si no hay opciones de ganar disponibles, realiza una jugada aleatoria  
159         for (let i = 0; i < board.length; i++) {  
160             for (let j = 0; j < board[i].length; j++) {  
161                 if (board[i][j] === "" && !played) {  
162                     board[i][j] = "X"; // Realiza la jugada  
163                     played = true; // Marca como jugada realizada  
164                 }  
165             }  
166         }  
167     }  
168 }
```

**Paso 43:** Ahora vamos a cambiar la variable “turn” a 0 para que se sepa que es el turno del jugador (línea 169) y llamamos la función “renderBoard”(línea 170) además actualizamos lo que se muestra de los turnos llamando la función “renderCurrentPlayer” (línea 171) y por último llamamos la función para que pueda jugar el jugador “playerPlays” (línea 172)

```
167     }  
168  
169     turn = 0; // Cambia el turno al jugador  
170     renderBoard(); // Actualiza el tablero en la interfaz  
171     renderCurrentPlayer(); // Actualiza el jugador actual en la interfaz  
172     playerPlays(); // Activa la función para que juegue el jugador  
173     setTimeout(() => { // Espera 1500 milisegundos antes de que la computadora realice su jugada  
174     }, 1500);  
175 }  
176 }
```

**Paso 44:** Ahora podremos mirar nuestro proyecto y vemos que está funcionando bien cambiando de turno y la maquina con cierta inteligencia. (Paso 19)

Turno del jugador

|   |   |   |
|---|---|---|
| O | X | O |
| X |   |   |
|   |   |   |

**Paso 45:** Ahora vamos a crear una función para saber si alguien ya ganó creando variables para saber las ubicaciones del tablero. (línea 179 a 187)

```
176 // Función para verificar si hay un ganador o empate en el juego
177 function checkIfWinner() {
178     // Extrae los valores de las celdas en el tablero para cada posición
179     const p1 = board[0][0];
180     const p2 = board[0][1];
181     const p3 = board[0][2];
182     const p4 = board[1][0];
183     const p5 = board[1][1];
184     const p6 = board[1][2];
185     const p7 = board[2][0];
186     const p8 = board[2][1];
187     const p9 = board[2][2];
188 }
```

**Paso 46:** También vamos a definir los patrones de las victorias. (línea 190 a 199)

```
187     const p9 = board[2][2];
188
189     // Define patrones de victoria posibles en el tablero
190     const winPatterns = [
191         [p1, p2, p3], // Fila 1
192         [p4, p5, p6], // Fila 2
193         [p7, p8, p9], // Fila 3
194         [p1, p4, p7], // Columna 1
195         [p2, p5, p8], // Columna 2
196         [p3, p6, p9], // Columna 3
197         [p1, p5, p9], // Diagonal principal
198         [p3, p5, p7], // Diagonal secundaria
199     ];
200
201     // Retorna true si alguno de los patrones de victoria es igual al board[0][0] o board[1][0] o board[2][0]
```

**Paso 47:** Ahora vamos a iterar en los patrones de victoria para ver si alguno se cumple (línea 202) y en caso de que alguno se cumpla, verifica si todas las casillas en el patrón son "X" (línea 204) y en caso de que así sea va a actualizar el "playerDiv" para que muestre un mensaje que diga que gana el computador (línea 205) y retorna un string (línea 206).

De lo contrario si todas las casillas en el patrón son "O" (línea 209) actualiza el "playerDiv" a que gana el jugador (línea 210).

```
200
201 // Itera a través de los patrones de victoria para verificar si alguno se cumple
202 for (const pattern of winPatterns) {
203   // Verifica si todas las celdas en el patrón son "X" (computadora ganadora)
204   if (pattern.every((cell) => cell === "X")) {
205     playerDiv.textContent = "Gana el computador"; // Muestra mensaje de victoria
206     return "computer"; // Retorna "computer" indicando que ganó la computadora
207   }
208   // Verifica si todas las celdas en el patrón son "O" (jugador ganador)
209   else if (pattern.every((cell) => cell === "O")) {
210     playerDiv.textContent = "Gana el jugador"; // Muestra mensaje de victoria
211     return "player"; // Retorna "player" indicando que ganó el jugador
212   }
213 }
214
```

**Paso 48:** Y ahora crearemos una nueva condicional para saber si todas las casillas están llenas y no hay ningún patrón que se cumpla (línea 216) en caso de que así sea llama a la función "renderDraw" para que muestre un empate (línea 217) y retornará un string "draw" (línea 218) y por último fuera de esa condicional va a retornar un "none" en caso de que aun no haya empate ni ganador aún

```
213 }
214
215 // Si todas las celdas están llenas y no hay ganador, se trata de un empate
216 if (board.flat().every((cell) => cell !== "")) {
217   renderDraw(); // Llama a la función para mostrar mensaje de empate
218   return "draw"; // Retorna "draw" indicando que es un empate
219 }
220
221 return "none"; // Si no hay ganador ni empate aún, retorna "none"
222 }
```

**Paso 49:** Ahora iremos a la función “PCPlays” (Paso 42) borraremos donde llamamos la función “playerPlays” y crearemos una constante llamada “won” y comprobará si hay un ganador llamando la función “checkIfWinner” (línea 173) y en caso de que lo que se le retorna es “none” (línea 175) significa que aún no hay ganador ni empate, por lo tanto llama a la función para que pueda jugar el jugador que borramos arriba (línea 176) y en caso de que se le retorna el string “draw” (línea 180) significa que ya se empato y llama a la función “renderDraw” (línea 181)

```
169     turn = 0; // Cambia el turno al jugador
170     renderBoard(); // Actualiza el tablero en la interfaz
171     renderCurrentPlayer(); // Actualiza el jugador actual en la interfaz
172
173     const won = checkIfWinner(); // Comprueba si hay un ganador
174     // Si no hay ganador, permite al jugador realizar su jugada
175     if (won === "none") {
176         playerPlays(); // Activa la función para que juegue el jugador
177         return;
178     }
179     // Si el resultado es empate, muestra el mensaje de empate
180     if (won === "draw") {
181         renderDraw(); // Activa la función para saber si fue un empate
182         return;
183     }
184     setTimeout(() => {}, 1500); // Espera 1500 milisegundos antes de que la computadora realice su jugada
185 }
```

**Paso 50:** Ahora iremos a la función “playerPlays” (paso 30) y vamos a borrar donde llamaos la función “PCPlays” y crearemos una constante llamada “won” y comprobará si hay un ganador llamando la función “checkIfWinner” (línea 77) y en caso de que lo que se le retorna es “none” (línea 79) significa que aún no hay ganador ni empate, por lo tanto llama a la función para que pueda jugar el computador que borramos arriba (línea 81)

```
69     if (board[row][column] === "") {
70         cell.addEventListener("click", (e) => {
71             // Si la celda está vacía
72             board[row][column] = "O"; // Marca la celda como ocupada por el jugador
73             cell.textContent = board[row][column]; // Actualiza la visualización en la interfaz
74             // Cambia el turno al de la computadora
75             turn = 1;
76             // Verifica si alguien ha ganado o si el juego ha terminado en empate
77             const won = checkIfWinner();
78             // Si no hay un ganador todavía
79             if (won === "none") {
80                 // Deja que la computadora juegue
81                 PCPlays();
82                 return;
83             }
84         });
85     }
```

**Paso 51:** Por último creamos otra condicional de que en caso de que “won” sea igual a “draw” significa que hay un empate (línea 86) llamando a la función “renderDraw” (línea 88) y remueve el evento clic para que el jugador ya no pueda hacer más jugadas (línea 90 a 93)

```
84
85     // Si el juego ha terminado en empate
86     if (won === "draw") {
87         // Muestra el empate en la interfaz
88         renderDraw();
89         // Remueve el evento de clic para que el jugador no pueda hacer más jugadas
90         cells.forEach((cell) => {
91             cell.removeEventListener("click", this);
92         });
93         return;
94     }
95 }
96 }
97 });
98 }
99
```

**Paso 52:** Así quedo nuestro proyecto, con nuestra maquina teniendo cierta inteligencia, y sabiendo si es un empate o una victoria para el jugador/maquina

Empate

|   |   |   |
|---|---|---|
| O | X | X |
| X | O | O |
| O | O | X |

Gana el jugador

|   |   |   |
|---|---|---|
| O | X | O |
| X | O |   |
| O |   | X |

Gana el computador

|   |   |   |
|---|---|---|
| X | X | O |
| X | O | O |
| X |   |   |