**1) Estructura del proyecto (qué archivos y por qué)**

Tú tienes exactamente tres piezas:

* index.html — la **estructura HTML**: botones de colores, inputs (tiempos), botón “Comenzar” y dos indicadores (mejor racha y progreso).
* simon.css — los **estilos**: cómo se ven los botones y qué clase .active hace que un botón se "ilumine".
* simon.js — la **lógica del juego** y la interacción con la página (DOM). Aquí está el “motor” y los eventListener.

El index.html incluye el <script src="simon.js"></script> al final del <body>, así el HTML está cargado antes de que JS intente encontrar elementos. Esa ordenación es importante para que document.getElementById() funcione.

**2) HTML: los elementos importantes y qué significan**

Abre index.html y fíjate en estos elementos (los usarás desde JS):

* Los botones de colores tienen id: "verde", "rojo", "amarillo", "azul". Ej: <button id="verde" class="color"></button>.

→ El id es la forma más directa que tiene JS para encontrar un elemento con document.getElementById("verde").

* Inputs para tiempos:
  + #tiempoOn → tiempo que cada luz se mantiene encendida (valor numérico).
  + #tiempoOff → tiempo de espera entre luces.
* Botón de control: #btnComenzar (inicio del juego).
* Indicadores:
  + #mejorRacha (muestra la mejor puntuación).
  + #rachaActual (muestra progreso tipo “x de y”).

Todos estos id permiten al JS leer/actualizar valores y controlar comportamiento del juego.

**3) CSS: cómo se “iluminan” los botones**

En simon.css tienes reglas como:

.color { background-color: #999; /\* color por defecto \*/ }

#verde.active { background-color: limegreen; }

Cuando JS añade la clase .active a un botón (element.classList.add("active")), el CSS cambia su color — eso crea el efecto de luz encendida. La clase se quita con classList.remove("active") para volver al estado normal.

**4) JavaScript: visión general de simon.js**

(a) El archivo está dividido en dos partes principales:

1. **Closure / módulo simon** — una función autoinvocada que encapsula toda la lógica del juego y expone solo funciones públicas (iniciarJuego, establecerColorPulsado, obtenerEstado, etc.). Esto evita “ensuciar” variables globales.
2. **Interfaz** (fuera del closure) — código que conecta los elementos del DOM (botones, inputs) con las funciones públicas del módulo simon (pone listeners y actualiza indicadores).

(b) Variables clave dentro del módulo:

* secuencia — array con la secuencia actual de colores (p. ej. ["rojo","azul"]).
* posicion — índice de la posición que el jugador debe pulsar ahora.
* mejorRacha — número máximo conseguido hasta ahora.
* estado — "parado" | "mostrando" | "jugando" (controla cuándo aceptar clics).

**5) Flujo: qué ocurre cuando pulsas Comenzar**

1. Interfaz (handler del botón) llama a simon.iniciarJuego() (en la sección pública del módulo).
2. iniciarJuego() dentro del módulo limpia secuencia y llama a nuevaRonda().
3. nuevaRonda() añade un color aleatorio a secuencia, pone estado = "mostrando" y llama a mostrarSecuencia() (privada).
4. mostrarSecuencia() lee los inputs #tiempoOn y #tiempoOff del DOM usando document.getElementById(...).value para saber cuánto encender y esperar.
5. mostrarSecuencia() recorre la secuencia con un pequeño bucle asíncrono (función siguiente() que se llama recursivamente con setTimeout) y para cada color:
   * obtiene el botón con document.getElementById(color),
   * añade la clase .active (btn.classList.add("active")),
   * programa quitarla pasado tiempoOn (setTimeout(..., tiempoOn)).  
     Esto crea la animación de encendido y apagado controlada por temporizadores.
6. Al terminar la reproducción de la secuencia, mostrarSecuencia() cambia estado = "jugando". Desde ese momento las pulsaciones del jugador se aceptan.

**6) Qué ocurre cuando haces clic en un botón de color**

* Los botones tienen este listener (en la parte interfaz):

botones.forEach(btn => {

btn.addEventListener("click", () => {

if (simon.obtenerEstado() !== "jugando") return;

const color = btn.id;

iluminarVisual(color); // feedback inmediato (clase .active)

simon.establecerColorPulsado(color); // lógica del juego

// actualizar progreso y mejor racha en DOM

});

});

* iluminarVisual(color) simplemente añade .active y la quita con setTimeout, para que el clic también tenga efecto visual.
* simon.establecerColorPulsado(color) comprueba si ese color coincide con secuencia[posicion]. Si es correcto incrementa posicion. Si posicion == secuencia.length entonces el jugador completó la ronda: mejorRacha se actualiza y se programa nuevaRonda() tras un delay. Si es incorrecto se hace alert() y se reinicia la secuencia.

**7) DOM: métodos que usas y qué hacen exactamente**

Aquí los métodos básicos que empleas —explicados con ejemplo:

* document.getElementById("verde")  
  → devuelve el elemento con id "verde". Si no existe devuelve null. (Usado en mostrarSecuencia() y iluminarVisual.)
* document.querySelectorAll(".color")  
  → devuelve una lista (NodeList) con los elementos que tienen la clase .color. Útil para iterar y añadir listeners.
* element.addEventListener("click", handler)  
  → registra un manejador para el evento click. No sobrescribe otros handlers si hay más. (Usado para los botones y para btnComenzar.)
* element.classList.add("active") / element.classList.remove("active")  
  → añade/quita clases CSS, lo que provoca cambio visual (CSS .active hace que se ilumine el botón).
* input.value  
  → lee el valor del campo <input id="tiempoOn">. Number(input.value) convierte a número.
* element.textContent = "..."  
  → muestra texto dentro de un elemento (se usa para actualizar #mejorRacha y #rachaActual).
* setTimeout(fn, ms)  
  → programa fn para ejecutarse después de ms milisegundos. Se usa para:
  + encender/apagar botones (efecto visual),
  + espaciar cada paso de la secuencia,
  + demorar la siguiente ronda.

**8) Temporizadores y el “bucle” de la secuencia**

La parte posiblemente más confusa si eres nuevo: mostrarSecuencia() no usa un for tradicional porque necesita esperar entre cada iluminación. En vez de eso:

* Se define una función siguiente().
* Al final de cada paso, setTimeout(siguiente, tiempoOn + tiempoOff) programa el siguiente ítem.  
  Es la forma clásica de “simular” un bucle temporizado sin bloquear la página.

**9) Estado (muy importante)**

Usas la variable estado con valores:

* "parado" → no se está jugando ni mostrando; cualquier clic se ignora.
* "mostrando" → el juego está reproduciendo la secuencia; ignorar clics.
* "jugando" → el jugador puede pulsar botones y la lógica compara.

Controlando el estado evitas que el usuario haga clic mientras el juego está mostrando la secuencia y rompe la sincronía.

**10) Errores comunes y cómo depurarlos**

Si algo no funciona, revisa en este orden:

1. **Consola del navegador** (F12 → Console). Ahí verás errores JS (ej. Uncaught ReferenceError).
2. **IDs coinciden**: si en HTML tu botón es id="green" y en JS buscas getElementById("verde"), no funcionará. Asegúrate que los id en HTML coinciden con los usados en JS.
3. **Orden de carga**: <script src="simon.js"></script> debe estar colocado **después** de los elementos HTML (al final del body) — así getElementById encuentra elementos.
4. **Errores de tiempo**: si los tiempoOn/tiempoOff son muy pequeños puede parecer que no se iluminan; aumenta a 700/300 para probar.

Para depurar paso a paso: pon console.log() en puntos clave (nuevaRonda(), mostrarSecuencia() y establecerColorPulsado()), recarga la página y observa la consola. Tu simon.js ya tiene algunos console.log útiles.

**11) Pequeños ejemplos prácticos para que practiques DOM (toca y verás)**

Copia estas líneas en la consola del navegador con la página abierta e inspecciona efectos:

* Seleccionar un botón y cambiar su color (usa la clase .active):

const b = document.getElementById("verde");

b.classList.add("active"); // se ilumina

setTimeout(()=> b.classList.remove("active"), 500); // vuelve

* Cambiar texto del indicador:

document.getElementById("mejorRacha").textContent = "5";

* Leer el tiempoOn:

Number(document.getElementById("tiempoOn").value);

* Añadir un handler extra a los botones:

document.getElementById("verde").addEventListener("click", () => console.log("Has pulsado verde"));

**12) Sugerencias de mejora / ejercicios para seguir aprendiendo**

* Añade un panel visual para los console.log (como te propuse antes) para no depender de la consola.
* Haz que el juego reproduzca un sonido por color (usar Audio o AudioContext).
* Añade un botón “Reset” que ponga mejorRacha = 0 y actualice el DOM.
* Practica creando elementos con document.createElement() y appendChild() para mostrar el historial de rondas.

**13) Resumen rápido (qué debes recordar)**

* HTML define elementos con id que JS usa con getElementById o querySelectorAll.

index

* CSS .active controla la apariencia cuando JS añade esa clase.
* JS contiene la lógica del juego en un **módulo** (closure) y la **interfaz** que manipula el DOM.
* setTimeout y classList son claves para la animación secuencial y el feedback visual.