Introducción

Objetivo: Aprender a crear el juego "Simon", un juego de memoria clásico, utilizando tecnologías web básicas: HTML, CSS y JavaScript.

Estructura HTML

1. **HTML Básico:** Explica la estructura básica de un documento HTML. Muestra cómo agregar el doctype, la etiqueta `<html>` y el elemento `<head>` que contiene la configuración del charset y la vista.

2. **Cuerpo del Documento:** Dentro de `<body>`, explica cómo crear una estructura principal con la etiqueta `<main>` que contendrá todo el juego.

3. **Secciones del Juego:**
    - **Encabezado `(<header>):`** Aquí irá el título del juego.
    - **Contenedor de Azulejos `(<section class="tile-container">):`** Explica cómo crear cuatro divisiones `(<div>)` para los colores del juego.
    - **Pie de página `(<footer>):`** Donde colocarás el botón de inicio y un espacio para mensajes informativos.
4. **Audio:** Muestra cómo agregar elementos de audio que se usarán para los sonidos de los azulejos.
5. **Script de JavaScript:** Explica la importancia de la etiqueta `<script>` al final del cuerpo para vincular el archivo JavaScript.

Estilos CSS

1. **Estilos Generales:** Describe cómo configurar los estilos básicos del documento, incluyendo el `box-sizing`, márgenes, padding y fuentes.
2. **Estilizando el Juego:**

    - **Contenedor del Juego:** Explica cómo centrar y posicionar el contenedor principal.
    - **Azulejos `(tile):`** Enseña cómo dar estilo a los azulejos con colores y sombras específicas.
    - **Botón de Inicio y Mensajes Informativos:** Muestra cómo estilizar el botón y los textos.

3. **Estilos Responsivos:** Explica cómo hacer que el juego se vea bien en dispositivos móviles usando media queries.

Lógica JavaScript

1. **Variables Globales:** Define las variables necesarias para guardar la secuencia del juego, la secuencia del jugador, y el nivel actual.

2. **Funciones Básicas:**

    - **`resetGame():`** Reinicia el juego.
    - **`humanTurn():`** Indica el turno del jugador.
    - **`activateTile():`** Activa visual y sonoramente un azulejo.
    - **`playRound():`** Juega una ronda mostrando la secuencia.

3. **Funciones de Control del Juego:**

    - **`nextStep():`** Elige el próximo color aleatoriamente.
    - **`nextRound():`** Avanza al siguiente nivel.
    - **`handleClick():`** Maneja los clics en los azulejos.
    - **`startGame():`** Inicia el juego.

4. **Event Listeners:**

    - Explica cómo agregar event listeners al botón de inicio y a los azulejos para manejar los clics.

Variables Globales

Las variables globales en JavaScript son accesibles desde cualquier parte del código. En el contexto de nuestro juego, estas variables almacenan información crucial como la secuencia del juego, la secuencia de acciones del jugador y el nivel actual del juego.

```
let sequence = [];
let humanSequence = [];
let level = 0;
```

Funciones Básicas

Funcionalidad de `resetGame()`

1. **Restablecer Variables de Estado:** La función comienza limpiando o restableciendo las variables globales que mantienen el estado del juego. Esto incluye la secuencia del juego (los colores que deben seguirse), la secuencia que el jugador ha ingresado, y el nivel actual del juego. Por ejemplo:

```
sequence = [];
humanSequence = [];
level = 0;
```

2. **Actualizar la Interfaz de Usuario:** `resetGame()` también se encarga de actualizar elementos de la interfaz para reflejar que el juego se ha reiniciado. Esto puede incluir:
    - Mostrar u ocultar botones (como el botón de inicio).
    - Actualizar mensajes en la pantalla, como el nivel actual o instrucciones para el jugador.
    - Desactivar la interacción con los azulejos hasta que el juego se reinicie.
3. **Preparar para una Nueva Partida:** La función prepara todo para que el jugador pueda empezar una nueva partida desde el principio, asegurando que el juego esté en su estado inicial.

Ejemplo de Implementación

Aquí hay un ejemplo simplificado de cómo podría verse la función **`resetGame():`**

```
function resetGame(text) {
    // Muestra un mensaje, por ejemplo, "¡Juego terminado!" o
"¡Felicitaciones!"
    alert(text);

    // Restablece las variables de estado a sus valores iniciales
    sequence = [];
    humanSequence = [];
    level = 0;
```

```
    // Actualiza la interfaz de usuario
    startButton.classList.remove('hidden'); // Muestra el botón de inicio
    heading.textContent = 'Simon Game';     // Restablece el título
    info.classList.add('hidden');           // Oculta cualquier mensaje
informativo
    tileContainer.classList.add('unclickable'); // Hace que los azulejos
no sean clicables
}
```

Funcionalidad de `humanTurn(level)`

1. **Permitir la Interacción del Jugador:** Lo más importante que hace esta función es habilitar la interactividad de los azulejos (o botones) de colores, permitiendo al jugador hacer clic en ellos. Esto se puede hacer modificando las propiedades CSS o las clases de los elementos HTML. Por ejemplo, se puede cambiar la clase de los azulejos para hacerlos clicables.

2. **Actualizar la Interfaz de Usuario:** La función también actualiza la interfaz para informar al jugador de que es su turno. Esto puede incluir:
   - Cambiar un mensaje en la pantalla que indique el nivel actual y que es el turno del jugador.
   - Iluminar los azulejos o cambiar su apariencia para indicar que ahora son interactivos.

3. **Manejar el Nivel Actual:** La función recibe como argumento el nivel actual del juego **`(level).`** Esto se puede utilizar para mostrar en la interfaz el nivel en el que se encuentra el jugador o ajustar la dificultad de la secuencia que el jugador debe seguir.

Ejemplo de Implementación

Aquí hay un ejemplo simplificado de cómo podría implementarse la función **`humanTurn(level):`**

```
function humanTurn(level) {
    // Hace que los azulejos sean clicables
    tileContainer.classList.remove('unclickable');

    // Actualiza la interfaz de usuario para mostrar que es el turno del
jugador
    info.textContent = `Tu turno: Nivel ${level}`;
}
```

Funcionalidad de `activateTile(color)`

1. **Cambio Visual:** Cuando un azulejo se activa, es importante que se realice un cambio visual claro para que el jugador pueda identificar fácilmente cuál azulejo se está activando. Esto puede incluir:
    - Cambiar el color del azulejo para que sea más brillante o resaltado.
    - Aplicar una animación breve, como un parpadeo o un resplandor.

2. **Reproducción de Sonido:** Cada azulejo en el juego Simon está asociado con un sonido único. La función **`activateTile`** debería reproducir el sonido correspondiente al color del azulejo activado. Esto ayuda a los jugadores a asociar cada color con su respectivo sonido, lo cual es un componente crítico del juego.

3. **Temporización:** La activación del azulejo generalmente dura solo un breve momento. Es importante que la función maneje esta temporización correctamente, asegurando que el cambio visual y el sonido no duren más de lo necesario.

Ejemplo de Implementación

Aquí hay un ejemplo simplificado de cómo podría implementarse la función **`activateTile(color):`**

```
function activateTile(color) {
    const tile = document.querySelector(`[data-tile='${color}']`);
    const sound = document.querySelector(`[data-sound='${color}']`);

    // Cambia el estilo del azulejo para mostrar que está activado
    tile.classList.add('activated');

    // Reproduce el sonido asociado
    sound.play();

    // Establece un temporizador para desactivar el azulejo después de un
corto período
    setTimeout(() => {
        tile.classList.remove('activated');
    }, 300); // El tiempo en milisegundos que el azulejo permanece
activado
}
```

- **`document.querySelector`** se usa para seleccionar el azulejo y el elemento de audio basado en el color proporcionado.
- **`tile.classList.add('activated')`** cambia el estilo del azulejo para indicar que está activo.
- **`sound.play()`** reproduce el sonido correspondiente.
- **`setTimeout`** se utiliza para desactivar el azulejo después de un breve período, revirtiendo el cambio visual.

Funcionalidad de `playRound()`

1. **Mostrar la Secuencia de Azulejos:** La función recorre la secuencia actual de colores (o azulejos) que el jugador debe seguir. Para cada color en la secuencia, la función activa el azulejo correspondiente, generalmente cambiando su apariencia visual y reproduciendo un sonido.

2. **Temporización y Orden:** Un aspecto clave de **`playRound()`** es la gestión del tiempo y el orden en que se muestran los azulejos. Cada azulejo debe activarse en un intervalo específico para que el jugador tenga tiempo de observar y memorizar la secuencia. Esto se logra generalmente a través de temporizadores o delays.

3. **Preparar para la Interacción del Jugador:** Después de mostrar la secuencia completa, la función debe preparar el juego para el turno del jugador, permitiendo que interactúen con los azulejos para replicar la secuencia mostrada.

Ejemplo de Implementación

Aquí hay un ejemplo simplificado de cómo podría implementarse la función **`playRound(nextSequence):`**

```
function playRound(nextSequence) {
    nextSequence.forEach((color, index) => {
        // Establece un temporizador para cada azulejo de la secuencia
        setTimeout(() => {
            activateTile(color);
        }, (index + 1) * 600); // El tiempo se incrementa para cada
azulejo
    });
}
```

En este ejemplo:

- **`nextSequence`** es un array que contiene la secuencia de colores que deben mostrarse.
- **`forEach`** se utiliza para iterar sobre cada color en la secuencia.
- **`setTimeout`** se usa para programar la activación de cada azulejo. El tiempo de espera se incrementa con cada iteración, asegurando que los azulejos se activen uno después del otro en lugar de todos al mismo tiempo.
-

Funciones de Control del Juego

Funcionalidad de `nextStep()`

1. **Generar una Selección Aleatoria:** La función elige aleatoriamente un color de un conjunto predefinido de colores (en el caso del juego Simon, estos suelen ser rojo, verde, azul y amarillo). La aleatoriedad es crucial para garantizar que el juego sea impredecible y desafiante.
2. **Actualizar la Secuencia del Juego:** Una vez que se ha seleccionado un color, **`nextStep()`** lo agrega a la secuencia actual del juego, que es la secuencia que el jugador necesita memorizar y replicar.

Ejemplo de Implementación

Aquí hay un ejemplo simplificado de cómo podría implementarse la función **`nextStep()`:**

```
function nextStep() {
    const tiles = ['red', 'green', 'blue', 'yellow']; // Colores
disponibles
    const randomIndex = Math.floor(Math.random() * tiles.length); //
Índice aleatorio
    return tiles[randomIndex]; // Retorna el color seleccionado
aleatoriamente
}
```

En este ejemplo:

- **`tiles`** es un arreglo que contiene los colores disponibles para los azulejos.
- **`Math.random()`** genera un número aleatorio entre 0 y 1.

- **`Math.floor()`** redondea el número hacia abajo para obtener un índice entero.
- El índice aleatorio se utiliza para seleccionar un color del arreglo **`tiles.`**

Funcionalidad de `nextRound()`

1. **Incrementar el Nivel:** **`nextRound()`** aumenta el nivel del juego, lo que generalmente se traduce en una secuencia más larga y, por lo tanto, más difícil de memorizar. Esto se logra agregando otro color a la secuencia existente.

2. **Generar un Nuevo Paso en la Secuencia:** Utilizando la función **`nextStep(), nextRound()`** agrega un nuevo color aleatorio a la secuencia actual, aumentando su longitud.

3. **Mostrar la Nueva Secuencia al Jugador:** Después de actualizar la secuencia, la función juega esta secuencia extendida utilizando **`playRound(),`** permitiendo al jugador observar y memorizar la nueva serie de colores.

4. **Preparar para la Interacción del Jugador:** Finalmente, **`nextRound()`** prepara el juego para el turno del jugador, habilitando la interacción con los azulejos para que el jugador intente replicar la secuencia recién mostrada.

Ejemplo de Implementación

Aquí hay un ejemplo simplificado de cómo podría implementarse la función **`nextRound():`**

```
function nextRound() {
    level += 1; // Incrementa el nivel
    // Agrega un nuevo color a la secuencia
    const newColor = nextStep();
    sequence.push(newColor);
    // Muestra la nueva secuencia al jugador
    playRound(sequence);
    // Preparaciones adicionales para el nuevo nivel, como actualizar la
interfaz
    // Por ejemplo, actualizar un mensaje en la pantalla con el nivel
actual
    updateLevelDisplay(level);
}
```

En este ejemplo:

- **`level += 1`** aumenta el nivel actual del juego.
- **`nextStep()`** se utiliza para generar un nuevo color aleatorio que se agrega a la secuencia con **`sequence.push(newColor).`**
- **`playRound(sequence)`** muestra la secuencia actualizada al jugador.
- **`updateLevelDisplay(level)`** podría ser una función adicional para actualizar la interfaz de usuario con el nuevo nivel.

Funcionalidad de `handleClick(tile)`

1. **Registrar la Elección del Jugador:** La función **`handleClick`** comienza registrando el azulejo que el jugador ha seleccionado. Esto implica identificar el color del azulejo y añadirlo a una secuencia que representa los intentos del jugador.

2. **Comprobar la Corrección de la Elección:** Luego, la función compara el azulejo seleccionado con el correspondiente en la secuencia generada por el juego. Si el jugador ha seleccionado el color incorrecto, se considera un error, y el juego puede terminar o manejar el error de alguna otra manera.

3. **Manejar la Secuencia Completa:** Si el jugador ha seleccionado correctamente el azulejo actual, la función verifica si el jugador ha completado toda la secuencia. Si es así, el juego puede avanzar al siguiente nivel.

4. **Retroalimentación al Jugador:** Además de la lógica del juego, **`handleClick`** también puede proporcionar retroalimentación visual o auditiva al jugador para indicar si su elección fue correcta o incorrecta.

Ejemplo de Implementación

Aquí hay un ejemplo simplificado de cómo podría implementarse la función **`handleClick(tile):`**

```
function handleClick(tile) {
    const index = humanSequence.push(tile) - 1; // Añade el azulejo
seleccionado a la secuencia del jugador
    const correctTile = sequence[index]; // Obtiene el azulejo correcto
de la secuencia del juego

    // Reproduce el sonido del azulejo
    playTileSound(tile);
```

```
    // Verifica si el azulejo seleccionado es correcto
    if (tile !== correctTile) {
        // Manejar error (por ejemplo, terminar el juego o mostrar un
mensaje)
        handleMistake();
        return;
    }

    // Verifica si el jugador ha completado la secuencia
    if (humanSequence.length === sequence.length) {
        // Avanza al siguiente nivel o maneja la victoria si es el último
nivel
        handleSequenceCompletion();
    }
}
```

En este ejemplo:

- **`humanSequence.push(tile) - 1`** registra la elección del jugador y obtiene el índice del azulejo seleccionado.
- **`playTileSound(tile)`** podría ser una función para reproducir el sonido asociado con el azulejo.
- **`if (tile !== correctTile)`** verifica si el azulejo seleccionado es el correcto.
- **`handleMistake() y handleSequenceCompletion() son`** funciones que manejarían un error o la finalización de la secuencia, respectivamente.

Funcionalidad de `startGame()`

1. **Establecer Condiciones Iniciales:** **`startGame()`** configura el juego en su estado inicial. Esto incluye inicializar o restablecer variables importantes como la secuencia del juego, la secuencia de entrada del jugador, y el nivel del juego.

2. **Ocultar o Mostrar Elementos de la Interfaz:** La función a menudo modifica la interfaz de usuario, como ocultar el botón de inicio y mostrar mensajes relevantes para el juego (por ejemplo, instrucciones o el nivel actual).

3. **Generar la Primera Secuencia del Juego:** **`startGame()`** comienza la primera ronda del juego generando el primer paso de la secuencia y mostrándolo al jugador.

4. **Habilitar la Interacción del Jugador:** La función prepara el juego para la entrada del jugador, asegurando que los azulejos sean interactivos y que el juego esté listo para responder a las acciones del jugador.

Ejemplo de Implementación

Aquí hay un ejemplo simplificado de cómo podría implementarse la función **`startGame()`:**

```
function startGame() {
    // Reinicia las variables del juego
    sequence = [];
    humanSequence = [];
    level = 0;

    // Oculta el botón de inicio y muestra cualquier otro elemento
relevante
    startButton.classList.add('hidden');
    info.classList.remove('hidden');
    info.textContent = 'Espera a la computadora';

    // Comienza la primera ronda
    nextRound();
}
```

En este ejemplo:

- Las variables **`sequence`**, **`humanSequence`** y **`level`** se reinician para establecer el estado inicial del juego.
- **`startButton.classList.add('hidden')`** oculta el botón de inicio, mientras que **`info.classList.remove('hidden')`** muestra un mensaje informativo.
- **`nextRound()`** se llama para comenzar la primera ronda del juego.

Funcionamiento de los Event Listeners

1. **Asignación de Event Listeners:** Se utiliza **`addEventListener`** para asignar un escuchador a un elemento específico del DOM (Document Object Model). Este escuchador "escucha" un tipo de evento (como un clic) y ejecuta una función cuando ese evento ocurre.

2. **Especificar el Tipo de Evento:** El primer argumento de **`addEventListener`** es el tipo de evento que se desea escuchar. En el ejemplo, **`'click'`** indica que el escuchador reaccionará a los clics del ratón.

3. **Definir la Función de Respuesta:** El segundo argumento es una función que se ejecuta cuando se detecta el evento. Puede ser una función ya definida (como **`startGame)`** o una función anónima (como en el segundo ejemplo).

Ejemplo de Uso en el Juego "Simon"

```
startButton.addEventListener('click', startGame);
tileContainer.addEventListener('click', event => {
    // Código para manejar el clic en un azulejo
});
```

En este código:

- **`startButton.addEventListener('click', startGame);`** asigna un escuchador al botón de inicio. Cuando se hace clic en este botón, se ejecuta la función **`startGame,`** que inicia el juego.

- **`tileContainer.addEventListener('click', event => {...});`** asigna un escuchador al contenedor de azulejos. Cada vez que se hace clic en este contenedor, se ejecuta la función anónima definida. Esta función puede utilizar el objeto **`event`** para determinar cuál azulejo específico fue clickeado (por ejemplo, usando **`event.target`**).

Importancia de los Event Listeners en Juegos Interactivos

- **Interacción con el Usuario:** Los event listeners permiten que el juego responda a las acciones del usuario, como iniciar el juego o elegir un azulejo, lo cual es esencial para cualquier juego interactivo.

- **Control del Flujo del Juego:** Mediante los event listeners, el juego puede controlar el flujo de juego, como avanzar de un nivel a otro o responder a elecciones incorrectas del jugador.

- **Mejora de la Experiencia del Usuario:** Proporcionan una experiencia de juego más rica y reactiva, lo que aumenta el compromiso y la satisfacción del usuario.

Conclusión y Prueba

1. **Revisión Final:** Asegúrate de que todo el código esté bien explicado y sea entendible para principiantes.
2. **Prueba el Juego:** Anima a los estudiantes a probar el juego y experimentar con cambios en el código para ver cómo afectan el juego.
3. **Desafíos Adicionales:** Sugiere mejoras o características adicionales que los estudiantes pueden intentar implementar por su cuenta.