Análisis del Script `mobile.js`

Este archivo es la interfaz del público. Su propósito es ofrecer a cualquier persona con un teléfono móvil una manera simple e intuitiva de interactuar con la visualización principal. La característica más importante de este script es su naturaleza de "camaleón": no tiene una única interfaz, sino que transforma su apariencia y funcionalidad para que coincida con el estado visual que el `Controller` ha activado.

Utiliza tecnologías web estándar:

1. JavaScript (puro): Para manipular los elementos de la página (crear botones, añadir texto, etc.) y gestionar la lógica de la interacción.
2. CSS: Para el estilo visual de los elementos (definido en el archivo `index.html`).
3. Socket.IO: Para recibir comandos del servidor (principalmente el cambio de estado) y enviar las acciones del usuario.

El Cerebro de la Aplicación: La Función `renderUIForState(estado)`

A diferencia de los otros scripts, la lógica aquí no se encuentra en un bucle `draw()` o en una serie de "escuchadores" de eventos dispersos. Casi toda la inteligencia del `mobile.js` reside en una única función: `renderUIForState(estado)`.

- Propósito: Es una función "constructora" de interfaces. Su trabajo es:
    i. Borrar completamente cualquier elemento que exista en la pantalla (`appContainer.innerHTML = '';`).
    ii. Leer el número del `estado` actual.
    iii. Basándose en ese número, construir desde cero la interfaz de usuario (UI) específica para esa interacción.
- ¿Cuándo se Llama?:
    i. Una vez al inicio, para dibujar la interfaz del estado por defecto.
    ii. Cada vez que se recibe un mensaje `'cambio_estado'` del servidor. Este es el disparador que le dice a la aplicación móvil que debe transformarse.

Análisis de la Interfaz y Lógica por Estado

Dentro de `renderUIForState`, una serie de `if / else if` determina qué interfaz construir.

Si `estado === 1` (Cascada)

- UI Construida: Un texto de instrucción ("Mantén presionado...") y un círculo (`.touch-ellipse`).
- Interacción: La lógica se centra en eventos de "presión sostenida".
    - Cuando el usuario toca y mantiene el círculo (`mousedown` o `touchstart`), se añade la clase CSS `.active` para dar feedback visual (el círculo brilla) y se envía el mensaje `socket.emit('state1_start_press');`.
    - Cuando el usuario suelta el círculo (`mouseup` o `touchend`), se quita la clase `.active` y se envía `socket.emit('state1_end_press');`.
- Propósito: Crear una interacción continua. El servidor puede contar cuántos usuarios están presionando simultáneamente para hacer que el visual reaccione a la cantidad de participantes.

Si `estado === 2` (Partículas)

- UI Construida: Un texto de instrucción y un botón circular grande (`.particle-button`) con una capa de "relleno" (`.fill`) para el efecto de recarga.
- Interacción: Un simple toque o clic en el botón.
- Lógica del Cliente (Cooldown): Esta interfaz tiene su propia lógica interna para evitar que los usuarios envíen eventos de forma masiva.
    i. Una variable `isReady` controla si se puede pulsar el botón.
    ii. Al hacer clic, si `isReady` es `true`, se envía inmediatamente `socket.emit('state2_create_particles');`.
    iii. Acto seguido, `isReady` se pone en `false` y se activa una animación CSS que vacía y rellena lentamente el botón.
    iv. Un temporizador (`setTimeout`) espera a que termine la animación de recarga (ej. 3 segundos) para volver a poner `isReady` en `true`.
- Propósito: Permitir al público generar eventos discretos e impactantes (una "explosión" por toque) de una manera controlada.

Si `estado === 3` (Puntos desde Imagen)

- UI Construida: Un texto de instrucción y una cuadrícula de botones, cada uno con una palabra (que corresponde a un nombre de archivo de imagen).
- Interacción: El usuario selecciona una de las palabras.
- Lógica del Cliente (Uso Único):

    i. Cuando se hace clic en un botón, se envía el mensaje `socket.emit('image_selection', 'PALABRA.png');`.
    ii. Inmediatamente después, el código deshabilita todos los botones y cambia el texto de instrucción a "¡Gracias por tu selección!". Esto le da al usuario un feedback claro de que su acción fue registrada y que ya no puede interactuar. La opción `{ once: true }` en el `addEventListener` asegura que cada botón solo pueda ser presionado una vez.
- Propósito: Dar al público la capacidad de influir directamente en el contenido visual, eligiendo la siguiente imagen que se mostrará.

Si `estado === 4` (Túnel)

- UI Construida: Un texto de instrucción y un contenedor invisible (`.pulse-container`) que ocupa toda la pantalla.
- Interacción: El usuario puede tocar en cualquier parte de la pantalla.
- Lógica del Cliente (Feedback Visual):
    i. Al detectar un toque, se envía el mensaje `socket.emit('state4_send_pulse');`.
    ii. Simultáneamente, se crea un nuevo elemento `div` (`.pulse-wave`) en el punto exacto donde el usuario tocó.
    iii. Una animación CSS hace que este `div` se expanda y se desvanezca, creando una onda visual. Este efecto es puramente local y sirve para dar al usuario una respuesta inmediata y satisfactoria a su acción.
- Propósito: Ofrecer una interacción libre y espacial que se siente como si el usuario estuviera "enviando pulsos" o "tocando" el túnel directamente.

Inicialización y Sincronización

Las últimas líneas del archivo son las que ponen todo en marcha:

```
renderUIForState(estadoActual);

socket.on('cambio_estado', (nuevoEstado) => {
    estadoActual = nuevoEstado;
    renderUIForState(estadoActual);
});
```

1. La primera llamada a `renderUIForState` dibuja la interfaz inicial (en este caso, la del Estado 4, ya que `estadoActual` empieza en 4).
2. El `socket.on('cambio_estado', ...)` es el "escuchador" que espera el comando del servidor. Cuando llega, actualiza la variable `estadoActual` local y vuelve a

llamar a `renderUIForState` para reconstruir la interfaz, asegurando que el móvil siempre esté sincronizado con la experiencia visual principal.