Sequência Lógica e Funcionamento do `app.js` (Para Iniciantes)

Olá! Este documento foi criado para te guiar passo a passo pela lógica do arquivo `app.js` do GeoDesafio. O JavaScript é a "inteligência" do seu jogo, e entender a ordem em que ele executa as tarefas é fundamental.

1. A Sequência Lógica de Execução

O JavaScript executa o código de cima para baixo, mas as **funções** (blocos de código que fazem uma tarefa específica) só são executadas quando são chamadas.

A sequência lógica do nosso jogo é a seguinte:

Passo	Bloco de Código	O que Acontece
1. Preparação	Seções 1, 2 e 3 (Variáveis, DOM, Estado)	O código **declara** as regras (`CONFIG`), **encontra** os elementos do HTML (DOM) e **define** o estado inicial do jogo (pontuação, desafio atual). *Nenhuma ação visível ainda.*
2. Início	Seção 8 (Inicialização) -> `loadChallenges()`	A última linha do arquivo (`loadChallenges()`) é a primeira a ser executada. Ela **chama** a função principal de carregamento de dados.
3. Carregamento de Dados	`loadChallenges()`	O JavaScript **solicita** o arquivo `data/images.json` (usando `fetch`), **converte** os dados para um formato que ele entende, e **embaralha** a lista de desafios.
4. Configuração Inicial	`initializeGame()`	Após carregar os dados, esta função é chamada. Ela **atualiza** textos no HTML com base nas regras (`CONFIG`), **carrega** a pontuação salva (`loadLeaderboard()`) e **chama** a função para mostrar o primeiro desafio.
5. Exibição do Desafio	`loadChallenge(index)`	Esta função **seleciona** o desafio atual, **reseta** o contador de tentativas, **marca** o tempo de início (`startTime`), **configura** a imagem e a descrição na tela e **aplica** o desfoque inicial. O jogo está pronto para o palpite.
6. Espera por Ação	`guessForm.addEventListener('submit', ...)`	O código **para** de executar a sequência principal e **fica esperando** que o usuário interaja com o formulário de palpite.
		Quando o usuário clica em

7. Interação do Usuário	`guessForm.addEventListener('submit', ...)`	"Palpitar", o código dentro do `addEventListener` é **ativado**. Ele **compara** o palpite com a resposta correta e **calcula** a pontuação e o bônus de velocidade.
8. Feedback e Pontuação	Bloco `if` (Acerto) ou `else` (Erro)	Se acertar, a imagem é **revelada**, a pontuação é **atualizada** (`updateLeaderboard()`) e o **mock do viaCEP** é chamado. Se errar, o contador de tentativas é **incrementado** e, se atingir o limite, uma **penalidade** é aplicada.
9. Próximo Desafio	`nextChallengeBtn.addEventListener('click', ...)`	Quando o usuário clica em "Próximo Desafio", o código **incrementa** o índice do desafio (`currentChallengeIndex++`) e **volta** ao **Passo 5** (`loadChallenge()`) para carregar o próximo item.

2. O que Cada Bloco de Instrução Faz

O código está dividido em seções lógicas para facilitar a organização.

Seções 1, 2 e 3: Definições (Variáveis, DOM e Estado)

- **`const CONFIG = { ... }`** : Define as **constantes** (valores fixos) do jogo, como pontuações e nomes de pastas.
- **`const challengeImage = document.getElementById('challenge-image');`** : Esta linha é crucial. Ela usa o **DOM** para criar uma **ponte** entre o JavaScript e o elemento HTML que tem o `id="challenge-image"` . Tudo o que você fizer com a variável `challengeImage` afetará a imagem na tela.
- **`let score = 0;`** : Define as **variáveis de estado** (valores que mudam), como a pontuação atual.

Seção 4: Funções de Inicialização

- `async function loadChallenges() { ... }` : A função que usa `fetch` para **ler** o arquivo `images.json`. O `async/await` é usado para dizer ao JavaScript: "Espere a internet trazer o arquivo antes de continuar".
- `function shuffleArray(array) { ... }` : Função auxiliar que **reorganiza** a ordem dos desafios de forma aleatória.
- `function initializeGame() { ... }` : Função que **prepara** a tela e chama o carregamento do primeiro desafio.

Seção 5: Lógica do Jogo

- `function getBasePoints(difficulty) { ... }` : Uma função de **decisão** que usa `switch` para retornar um valor diferente de pontuação dependendo da dificuldade do desafio.
- `async function mockViaCEP(cep) { ... }` : Simula a **integração com uma API externa**. Em vez de fazer uma chamada real, ela mostra como o resultado de um CEP seria usado para dar informações extras ao jogador.
- `guessForm.addEventListener('submit', async (event) => { ... });` : Este é um **manipulador de eventos**. Ele "escuta" o evento de submissão do formulário. É o coração da lógica do jogo, onde o palpite é **validado** e a **pontuação** é calculada.

Seção 6: Gamificação (Leaderboard)

- `function loadLeaderboard() { ... }` : Função que **lê** os dados de pontuação salvos no `localStorage` do navegador.
- `function updateLeaderboard(reason, points) { ... }` : Função que **salva** a pontuação atualizada de volta no `localStorage`. Isso garante que a pontuação não seja perdida quando o jogador fechar a página.

Seção 8: Inicialização

- `loadChallenges();` : Esta é a **chamada de função** que dá o pontapé inicial em todo o processo. Sem ela, o código seria apenas um conjunto de definições que nunca seriam executadas.

Ao entender essa sequência, você percebe que o JavaScript é como um maestro: ele define os instrumentos (variáveis), ensaia as músicas (funções) e, por fim, dá o sinal para a orquestra começar (chamada de inicialização).