Documentación Técnica: Simulador de Red Nacional de Energía

Johan Barajas, Miguel Ortegón,Nicolas Fuentes

9 de diciembre de 2025

Resumen

Este documento presenta una descripción técnica del funcionamiento y arquitectura del software "Pikachu Simulator". El sistema es una simulación interactiva de una red eléctrica nacional (basada en la topología de Colombia) que modela la generación, transmisión y consumo de energía en tiempo real. Se detalla la estructura modular del código, las relaciones entre componentes y los algoritmos principales que gobiernan la estabilidad de la red.

Índice

1. Introducción

El proyecto tiene como objetivo simular una red eléctrica a gran escala visualizada sobre un mapa geográfico. El sistema permite al usuario interactuar con la red mediante controles de estrés (picos de demanda, fallas de línea) y observar la respuesta del sistema en términos de flujo de carga y estabilidad. La arquitectura está basada en un ciclo de simulación continuo (Game Loop) que actualiza los cálculos físicos y renderiza el estado en un elemento *Canvas* de HTML5.

2. Arquitectura del Sistema

El código está estructurado bajo un patrón modular utilizando ES6 Modules. La arquitectura se divide en tres capas principales:

1. **Capa de Datos y Recursos:** Responsable de la configuración estática y la carga de activos (imágenes, datos geográficos GeoJSON, topología de la red).
2. **Capa de Simulación (Lógica de Negocio):** Contiene las clases que modelan el comportamiento físico de la red eléctrica (nodos, líneas, flujo de potencia).
3. **Capa de Presentación y Control:** Gestiona el renderizado gráfico, la cámara (zoom/pan), la interfaz de usuario (UI) y el bucle principal de ejecución.

3. Descripción de Módulos

A continuación, se desglosa el funcionamiento de cada módulo y su rol dentro del sistema.

3.1. Módulo de Configuración (`config.js`)

Actúa como la fuente de verdad para las constantes globales del sistema. Define parámetros críticos como:

- **Física del Mundo:** Dimensiones del mundo virtual y factores de escala.
- **Visualización:** Paleta de colores para estados de la red (normal, crítico, desconectado) y estilos de la grilla.
- **Interfaz:** Dimensiones de los paneles laterales y consola.

Su función es centralizar los "números mágicos"para facilitar el ajuste y balanceo de la simulación sin modificar la lógica interna.

3.2. Gestor de Recursos (`AssetLoader.js`)

Este módulo se encarga de la inicialización asíncrona. Antes de arrancar la simulación, realiza dos tareas fundamentales:

1. Cargar y decodificar los iconos (generadores y cargas) en formato SVG/Base64.
2. Descargar datos geográficos externos (GeoJSON) para dibujar el contorno del país con alta precisión.

Utiliza promesas (*Promises*) para asegurar que todos los activos estén disponibles antes de que el motor de renderizado intente dibujarlos.

3.3. Núcleo de Simulación (`PowerGridSimulation.js`)

Es el "cerebro" del sistema. Esta clase orquesta toda la lógica de la red eléctrica. Sus responsabilidades incluyen:

- **Construcción de Topología:** Convierte los datos crudos en objetos interactivos. Implementa un algoritmo de mallado que conecta nodos generadores con municipios y crea una çolumna vertebral"(backbone) para unir el país, además de redes locales vecinales.
- **Cálculo de Flujo de Potencia:** Ejecuta en cada ciclo el algoritmo que determina cuánta energía fluye por cada línea. Utiliza un modelo simplificado basado en "voltaje virtualτ diferencias de potencial para calcular flujos.
- **Manejo de Eventos:** Procesa acciones del usuario como "Pico Nacional"(aumento de demanda) o "Falla de Línea"(desconexión forzada), recalculando el estado de la red instantáneamente.
- **Protecciones:** Monitorea el estrés térmico de las líneas y desconecta aquellas que superan su capacidad máxima por un tiempo prolongado.

3.4. Entidades de la Red (`PowerNode.js` y `TransmissionLine`)

Estas clases representan los objetos individuales de la simulación:

- **PowerNode:** Modela tanto generadores como cargas (municipios). Mantiene su estado (potencia generada/consumida, voltaje virtual, posición geográfica) y maneja su propia lógica de dibujo básico, ajustando su apariencia según el nivel de zoom.
- **TransmissionLine:** Representa los cables físicos. Calcula su propia carga actual (MVA) basada en la diferencia de estado entre los nodos que conecta. Posee lógica interna para simular calentamiento y disparo de protecciones.

3.5. Sistema de Cámara (`Camera.js`)

Permite la navegación por el mapa. Transforma las coordenadas del "Mundo"(simulación) a coordenadas de "Pantalla"(canvas).

- **Zoom Dinámico:** Implementa un zoom que se centra en la posición del puntero del mouse.
- **Panning:** Permite arrastrar el mapa para explorar diferentes regiones geográficas.

3.6. Bucle de Juego (`GameLoop.js`)

Gestiona el tiempo de la aplicación. Desacopla la lógica de simulación de la tasa de refresco de la pantalla:

- Calcula el *delta time* (tiempo transcurrido entre cuadros).
- Llama a la función de actualización de la simulación.
- Solicita el siguiente cuadro de animación al navegador.

4. Funcionamiento Algorítmico Macro

La interacción entre módulos sigue un flujo cíclico constante:

4.1. Inicialización

Al cargar la página (`index.html`), el punto de entrada (`main.js`) invoca al `AssetLoader`. Una vez cargados los datos, se instancia la `PowerGridSimulation`, la cual proyecta las coordenadas geográficas (latitud/longitud) al espacio 2D del canvas y construye el grafo de conexiones (nodos y líneas).

4.2. Ciclo de Actualización (Update Loop)

En cada iteración del `GameLoop`:

1. **Balance Oferta/Demanda:** La simulación suma toda la demanda de los nodos de carga y la capacidad de los generadores. Calcula un "factor de utilización" global.
2. **Distribución de Carga:** Se ajusta la producción de los generadores según el factor calculado. Si hay un arranque en frío, se aplica un "Soft Start" para evitar picos bruscos.
3. **Física de Líneas:** Se calcula el flujo a través de cada línea basado en la diferencia de estado de los nodos conectados y la impedancia.
4. **Verificación de Estado:** Se actualizan las métricas de estrés del sistema y se verifica si alguna línea debe desconectarse por sobrecarga.

4.3. Ciclo de Renderizado

Paralelamente, el sistema gráfico limpia el canvas, aplica las transformaciones de la `Camera`, dibuja el mapa base (GeoJSON) y luego itera sobre la lista de líneas y nodos para dibujarlos en sus nuevas posiciones y estados (colores de alerta o normalidad). Finalmente, la interfaz de usuario (DOM) se actualiza con métricas numéricas.

5. Conclusión

El código presenta una arquitectura robusta para la simulación de sistemas complejos en tiempo real web. La separación de responsabilidades entre la lógica física (`PowerGridSimulation`), la representación de datos (`PowerNode`) y la visualización (`Camera`, UI) permite escalar el proyecto, facilitando la adición de nuevas reglas físicas o elementos visuales sin comprometer la estabilidad del sistema.