Skill Tree Visualizer

Documentación del Código y Algoritmos

Geor Sebastián Gómez Correa
Tomás Camilo García López
Eduardo Castellanos Márquez

UNIVERSIDAD NACIONAL DE COLOMBIA

Documentación Técnica:

Tecnologías: HTML5, CSS3 (Tailwind + Custom), Vanilla JavaScript (ES6+), Python (ETL).
Arquitectura: Single Page Application (SPA) con Carga Modular Dinámica.

---

1. Visión General del Proyecto

Esta aplicación es una plataforma web interactiva diseñada para visualizar, planificar y compartir "Builds" (configuraciones de habilidades) de videojuegos complejos. Actualmente soporta tres módulos:

1. **Cyberpunk 2077:** Estilo futurista/neón con conexiones lógicas por atributos.
2. **The Elder Scrolls V: Skyrim:** Estilo constelación 3D con navegación rotativa.
3. **Assassin's Creed Origins:** Estilo papiro egipcio con un lienzo de desplazamiento infinito.

El núcleo del sistema funciona como un "lanzador" (Launcher) que inyecta dinámicamente el código, estilos y datos del juego seleccionado, manteniendo el rendimiento optimizado al no cargar todo al inicio.

---

2. Estructura de Directorios

El proyecto sigue una estructura jerárquica clara para separar el núcleo (Core) de los módulos de juego (Games).

Plaintext

```
Python
/SkillTree-Visualizer
│
├── index.html              # Punto de entrada (Shell).
Contiene el menú principal.
├── main.js                 # Orquestador. Maneja el carrusel
y la inyección de módulos.
├── global_styles.css       # Estilos globales (fuentes,
carrusel 3D, scrollbars).
│
└── /games                  # Carpeta contenedora de módulos
    ├── /cp2077             # MÓDULO 1: Cyberpunk
    │   ├── cp2077.js           # Lógica encapsulada del juego
(IIFE).
    │   ├── cp2077.css          # Estilos específicos
(scoped).
    │   ├── skills_data.js      # Base de datos de habilidades
(JSON).
    │   ├── layout_config.js    # Coordenadas X/Y de los
nodos.
    │   ├── presets_data.js     # Builds predefinidas
(Netrunner, Solo, etc.).
    │   ├── process_data.py     # Script ETL (Excel -> JS).
    │   └── /assets             # Imágenes e iconos.
    │
    ├── /skyrim             # MÓDULO 2: Skyrim (Estructura
idéntica al anterior)
    │   ├── skyrim.js
    │   ├── ...
    │
    └── /acorigins          # MÓDULO 3: AC Origins (Estructura
idéntica al anterior)
        ├── acorigins.js
        ├── ...
```

---

3. Arquitectura del Núcleo (Core)

A. Interfaz Principal (`index.html` y `global_styles.css`)

- **Diseño:** Utiliza **Tailwind CSS** para la estructura rápida y CSS personalizado (`global_styles.css`) para efectos avanzados como el **Carrusel 3D** (`transform-style: preserve-3d`) y las fuentes *Orbitron/Inter*.
- **Responsabilidad:** Solo muestra el menú de selección. No contiene lógica de ningún juego específico.

B. El Orquestador (`main.js`)

Es el cerebro de la aplicación. Sus funciones principales son:

1. **`GAMES_DB`:** Un array de configuración que define qué juegos existen, sus imágenes de portada y colores temáticos.
2. **`init3DCarousel()`:** Genera las tarjetas en el DOM y calcula su rotación 3D basándose en la cantidad de juegos.
3. **`loadGameModule(gameId)`:** La función crítica.
    - Oculta el menú principal (`display: none`).
    - Crea dinámicamente etiquetas `<link rel="stylesheet">` para cargar el CSS del juego.
    - Utiliza `loadScriptsSequentially` para cargar en orden estricto: `Data` -> `Layout` -> `Presets` -> `Logic`.
    - Ejecuta el método `init()` del módulo cargado.
4. **`restoreMainMenu()`:** Función global expuesta. Permite que los módulos "hijos" se cierren y devuelvan el control al menú principal.

---

4. Arquitectura de los Módulos de Juego

Cada juego (CP2077, Skyrim, AC) funciona como un **módulo independiente** utilizando el patrón de diseño **IIFE (Immediately Invoked Function Expression)**. Esto evita conflictos de variables entre juegos.

Patrón Estándar de un Módulo (Ej. `cp2077.js`)

JavaScript

```
JavaScript
const Game_Module = (function() {
```

```
    // 1. Estado Privado
    let container, skillsData;
    let points = 50;

    // 2. Método Init (Constructor)
    function init() {
        // Inyecta el HTML específico del juego
        // Dibuja los nodos basados en skills_data.js
        // Dibuja las conexiones SVG
    }

    // 3. Lógica del Árbol
    function handleNodeClick(node) {
        // Verifica requisitos (padres desbloqueados, puntos
suficientes)
        // Actualiza el estado visual (CSS classes)
    }

    // 4. Método Destroy (Destructor)
    function destroy() {
        // Elimina el contenedor HTML del juego
        // Elimina el CSS específico del <head>
        // Llama a window.restoreMainMenu()
    }

    // API Pública
    return { init, destroy };
})();
```

Componentes de Datos

1. **`skills_data.js` (La Base de Datos):**
    - Contiene la información lógica: ID, Nombre, Descripción, Requisitos (Padres), Niveles Máximos.
    - Es generado automáticamente. **No se edita a mano.**
2. **`layout_config.js` (La Vista):**
    - Contiene puramente coordenadas visuales: `{ id: "1", x: 500, y: 300 }`.

    - Separar esto permite cambiar el diseño visual sin tocar la lógica de datos.
3. **`process_data.py` (El Backend ETL):**
    - Script de Python utilizando `pandas`.
    - Lee archivos Excel (`.xlsx`) fáciles de editar para humanos.
    - Transforma filas y columnas en una estructura JSON jerárquica (Padres/Hijos).
    - Exporta el archivo `skills_data.js` listo para usar.

---

5. Guía de Desarrollo y Mantenimiento

¿Cómo agregar un nuevo juego?

1. Crea una carpeta en `/games/nuevo_juego`.
2. Crea el archivo Excel con las habilidades.
3. Ejecuta el script de Python para generar `skills_data.js`.
4. Crea `nuevo_juego.js` siguiendo la plantilla IIFE (`init` y `destroy`).
5. Registra el juego en `GAMES_DB` dentro de `main.js`.
6. Añade el bloque `else if (gameId === 'nuevo_juego')` en `loadGameModule` en `main.js`.

¿Cómo editar las posiciones de los nodos?

1. El sistema incluye un **Modo Edición** (interno en los JS).
2. Al activarlo, los nodos se vuelven "arrastrables" (Drag & Drop).
3. Se incluye un botón "Exportar Layout" que imprime el JSON de coordenadas en la consola.
4. Copia ese JSON y pégalo en `layout_config.js`.

Manejo de Errores

El sistema es robusto ante fallos de carga.

- Si un script falla al cargar (error 404), `main.js` captura el evento `onerror`, alerta al usuario y llama automáticamente a `restoreMainMenu()` para evitar que la pantalla se quede en negro.

---

6. Flujo de Usuario (User Journey)

1. **Inicio:** El usuario ve el título animado y el carrusel de juegos.
2. **Selección:** Al hacer clic en "Cyberpunk 2077", el menú se desvanece.

3. **Carga:** Se descargan los scripts específicos de Cyberpunk.
4. **Juego:**
    - Aparece la interfaz de Cyberpunk (pestañas de atributos, fondo, nodos).
    - El usuario gasta puntos, las líneas se iluminan (SVG).
    - Puede guardar "Presets" o reiniciar.
5. **Salida:** Al hacer clic en "Salir", el módulo Cyberpunk se autodestruye (limpieza de DOM) y reaparece el menú principal instantáneamente.