electron tetris

Informe técnico  
Autor: María Sol Rico

Informe Técnico: Desarrollo de una Aplicación de Escritorio Multiplataforma - Tetris en Electron JS

# Introducción

Este informe detalla el diseño, la implementación y los resultados de una aplicación de escritorio multiplataforma desarrollada con Electron JS. El objetivo del proyecto fue crear una versión funcional del clásico juego de Tetris para demostrar la capacidad de Electron para combinar tecnologías web modernas (HTML, CSS, JavaScript) con APIs nativas del sistema operativo, resultando en una experiencia de usuario robusta y completa. El proyecto cumple con los requisitos del BTS SIO, centrándose en la arquitectura de procesos y la interacción con el entorno nativo.

# Arquitectura de la Aplicación

La arquitectura de la aplicación se basa en un modelo de dos procesos de Electron y una estructura de interfaz de usuario de tres pantallas:

* **Proceso Principal (main.js):** Actúa como el "backend" de la aplicación. Su función principal es gestionar la ventana de la aplicación, el ciclo de vida y la interacción con el sistema operativo.
  + **Gestión de Ventanas:** Se encarga de crear y controlar la ventana principal (BrowserWindow).
  + **APIs Nativas:** Implementa el menú de la aplicación, que permite acciones como "Reiniciar" y "Salir", y gestiona las notificaciones del sistema a través del módulo Notification.
  + **Comunicación Inter-Procesos (IPC):** Utiliza ipcMain para escuchar eventos enviados desde el proceso de renderizado, como la solicitud para mostrar una notificación.
* **Proceso de Renderizado (renderer.js e index.html):** Funciona como el "frontend" de la aplicación.
  + **Interfaz de Usuario:** El archivo index.html sirve como la interfaz visual del juego. Se ha estructurado en tres pantallas discretas (start-screen, game-screen y game-over-screen), que se controlan dinámicamente con CSS y JavaScript para asegurar que solo una sea visible a la vez.
  + **Lógica del Juego:** El archivo renderer.js contiene toda la lógica del juego de Tetris. Se encarga de la gestión del estado, la representación del tablero como una matriz 2D, la lógica de movimiento, la detección de colisiones, la eliminación de líneas y el cálculo de la puntuación.
  + **Interacción con APIs Nativas:** Usa ipcRenderer para enviar mensajes al proceso principal, como cuando la partida ha terminado y se debe mostrar una notificación.

# Desafíos y Soluciones

El desarrollo de la aplicación presentó varios desafíos, los cuales se abordaron con las siguientes soluciones:

* **Detección de Colisiones:** El principal desafío fue garantizar que las piezas no se superpusieran con otras o se salieran de los límites del tablero. Esto se resolvió con la función isColliding(), que verifica la posición de cada celda de la pieza antes de permitir cualquier movimiento.
* **Rotación de Piezas:** La rotación de una matriz 2D requería una transformación matemática precisa. La solución fue transponer la matriz y luego invertir las filas, un método eficaz que funciona para cualquier forma de pieza.
* **Eliminación de Líneas:** El problema de que las líneas se saltaran la revisión después de ser eliminadas se resolvió ajustando el índice del bucle (row++) cada vez que se borraba una fila.
* **Gestión del Fin del Juego:** Para evitar que el juego continuara y enviara múltiples notificaciones de "Game Over", se implementó una variable de estado (isGameOver) y se detiene explícitamente el bucle de animación con cancelAnimationFrame() tan pronto como se detecta el fin de la partida. Esto asegura que la lógica de "Game Over" se ejecute solo una vez.
* **Puntuaciones y Mensajes Personalizados:** Se creó un sistema de puntuaciones altas persistente utilizando la API localStorage. Adicionalmente, se implementó un sistema de mensajes dinámicos que muestra un comentario personalizado en la pantalla de "Fin de la Partida" basado en la puntuación del jugador.
* **Arquitectura de Pantallas:** La visibilidad de las tres pantallas (inicio, juego, fin del juego) se gestiona a través de un sistema de clases CSS (screen-visible, screen-hidden) y funciones dedicadas en renderer.js. Cada función de pantalla oculta las demás de manera explícita para evitar la superposición de la interfaz.

# Integración de APIs Nativas

La integración de APIs nativas de Electron mejoró la funcionalidad de la aplicación de las siguientes maneras:

* **API Menu:** Se creó un menú de aplicación que permite a los usuarios "Reiniciar" el juego, proporcionando una forma familiar y accesible de empezar una nueva partida.
* **API Notification:** Se utilizaron notificaciones del sistema para enviar mensajes al jugador al inicio y al final de la partida, lo que resulta en una experiencia más interactiva y profesional que los diálogos alert() tradicionales.
* **API Dialog:** Se integró en la opción "Acerca de" del menú. Esto permite mostrar información sobre la aplicación a través de un cuadro de diálogo nativo del sistema, en lugar de un cuadro de alerta básico.
* **API localStorage:** Aunque no es una API nativa de Electron, su uso para guardar y cargar puntuaciones demuestra la capacidad de **Electron** para integrar **APIs web del navegador** en una aplicación de escritorio, creando una experiencia completa sin la necesidad de un backend.

# Empaquetado y Conclusión

La aplicación fue empaquetada con Electron-Builder, lo que permitió generar ejecutables para Windows, macOS y Linux. Este proceso valida la capacidad de la aplicación para ser distribuida a usuarios sin necesidad de que estos instalen dependencias de desarrollo.

En conclusión, este proyecto demuestra el potencial de Electron JS para crear software de escritorio robusto y funcional a partir de una base de conocimientos en desarrollo web. La aplicación de Tetris es un ejemplo claro de cómo la integración de tecnologías web con APIs nativas puede resultar en un producto final de alta calidad y completamente adaptado al entorno de escritorio.