Análisis Lineal del Código: `global_state.py`

El archivo `global_state.py` representa un patrón de diseño que, aunque a veces controvertido, es muy útil en aplicaciones complejas como esta. Su propósito principal es actuar como un repositorio centralizado para el estado compartido, es decir, para todas las variables y datos que necesitan ser accedidos y/o modificados por diferentes módulos del proyecto. El uso de un archivo como este ayuda a resolver un problema común en programas grandes: las dependencias circulares (cuando el módulo A necesita importar a B, y el módulo B a su vez necesita importar a A). Al tener un tercer módulo neutro (`global_state`) que ambos pueden importar, se rompe este círculo.

El script comienza importando `threading`, que es esencial porque una de las variables más importantes que gestionará, los datos del hardware, necesita un cerrojo (`threading.Lock`) para ser accedida de forma segura por múltiples hilos simultáneamente.

A continuación, se definen las variables de estado, agrupadas por función.

En la sección "Estado de la Conexión con Arduino", se declaran `arduino_serial_object` y `arduino_esta_conectado`. La primera guardará el objeto de conexión real de PySerial, mientras que la segunda es una simple bandera (booleana) para que cualquier módulo pueda verificar rápidamente si la conexión está activa.

La sección "Datos del Hardware y Lock" es quizás la más crítica. `datos_hardware_compartidos` es un diccionario que contiene los últimos valores leídos de todos los sensores del Arduino (distancias, interruptores, RFID). Junto a él, se define `lock_datos_hardware`, un `Lock` que debe ser usado siempre que se lea o escriba en este diccionario. Esto es vital porque el hilo del listener de Arduino estará escribiendo en este diccionario constantemente, mientras que la GUI y la máquina de estados estarán leyéndolo. El `Lock` previene que se lean datos corruptos o a medio escribir.

Las siguientes secciones muestran decisiones de diseño interesantes. El estado actual de la máquina de estados y las variables de reportes están comentados. Esto indica que el desarrollador consideró ponerlos aquí, pero decidió que era mejor que permanecieran dentro de sus respectivos módulos (`state_machine_logic.py` y `reporting_logging.py`). Esta es una buena práctica de encapsulación, ya que evita que módulos externos modifiquen directamente un estado que no les "pertenece", promoviendo un código más limpio y menos propenso a errores.

Sin embargo, sí se incluyen aquí variables que, aunque son gestionadas por la máquina de estados, son leídas frecuentemente por la GUI, como `protocolo_seleccionado_info` (que indica si se usa RFID, QR, etc.) y `estado_validacion_secuencial_actual` (el "borrador" para validaciones de múltiples pasos).

La variable `encodings_faciales_globales` se declara aquí para servir como la memoria caché central de los encodings faciales. El módulo `facial_recognition_utils` será el encargado de *poblar* esta lista al inicio, y la máquina de estados será la encargada de *leerla* durante el proceso de validación facial.

La variable `app_gui_instancia` es una referencia global a la interfaz gráfica. Esto permite que módulos que no importan directamente a `gui_manager` (para evitar dependencias circulares) puedan, por ejemplo, mostrar una ventana de mensaje de error. Se usa con precaución, pero es muy práctico.

Finalmente, la sección más importante para el control del programa es "Flags de Control de Hilos". Las variables `hilo_listener_arduino_debe_correr` y `hilo_maquina_estados_debe_correr` son las "llaves de apagado" de los hilos de fondo. Cuando el usuario decide cerrar la aplicación o desconectar el Arduino, la GUI o el script principal pondrán estas variables en `False`. Los bucles `while` dentro de los hilos revisan el estado de estas banderas en cada ciclo. Al ver que han cambiado a `False`, los bucles terminan y los hilos se apagan de manera ordenada y segura.

El archivo concluye con un par de funciones de ejemplo, `set_app_gui_instance` y `get_app_gui_instance`, que son métodos "setter" y "getter" para manejar la referencia a la GUI de una manera un poco más estructurada que modificando la variable global directamente.