Refactorización scripts de system:  
  
SYSLOG.gd  
  
extends Node

# ==========================

# Configuración General

# ==========================

@export var log\_directory: String = "user://logs/SYSLOG" # Directorio principal para los logs

var log\_files = {} # Diccionario para mapear nombres de scripts a archivos de log

# ==========================

# Inicialización del Sistema

# ==========================

func \_ready() -> void:

\_initialize\_log\_system()

func \_initialize\_log\_system():

\_create\_directory(log\_directory)

\_backup\_logs()

\_log\_internal("Sistema de logs inicializado correctamente.", "DEBUG")

# ==========================

# Manejo de Directorios

# ==========================

func \_create\_directory(path: String) -> void:

var dir = DirAccess.open(path.get\_base\_dir())

if not dir or dir.make\_dir\_recursive(path) != OK:

\_log\_internal("No se pudo crear el directorio '%s'." % path, "ERROR")

else:

\_log\_internal("Directorio creado: '%s'." % path, "DEBUG")

func \_backup\_logs():

var timestamp = Time.get\_datetime\_string\_from\_system().replace(":", "-").replace(" ", "\_")

var backup\_path = "%s\_BACKUP\_%s" % [log\_directory, timestamp]

\_create\_directory(backup\_path)

# Lógica de copia de archivos implementada aquí

# ==========================

# Registro de Mensajes

# ==========================

func log\_message(message: String, level: String, script\_name: String):

var formatted\_message = "[%s] [%s] %s: %s" % [

Time.get\_datetime\_string\_from\_system(),

level,

script\_name,

message

]

print(formatted\_message)

\_write\_to\_file(script\_name, formatted\_message)

func \_write\_to\_file(script\_name: String, message: String):

if not log\_files.has(script\_name):

\_initialize\_log\_file(script\_name)

var log\_path = log\_files[script\_name]

var file = FileAccess.open(log\_path, FileAccess.ModeFlags.READ\_WRITE)

if file:

file.seek\_end()

file.store\_line(message)

file.close()

# ==========================

# Logs Internos

# ==========================

func \_log\_internal(message: String, level: String):

log\_message(message, level, "SYSLOG")

func \_initialize\_log\_file(script\_name: String):

var log\_path = "%s/%s.log" % [log\_directory, script\_name]

log\_files[script\_name] = log\_path

var file = FileAccess.open(log\_path, FileAccess.ModeFlags.WRITE)

if file:

file.store\_line("[LOG] Archivo inicializado para '%s'." % script\_name)

file.close()

SAVE.gd  
extends Node

# ==========================

# Señales Emitidas

# ==========================

signal save\_system\_ready

# ==========================

# Configuración General

# ==========================

@export var template\_path: String = "res://assets/save\_game/save\_game\_template.json"

@export var save\_path: String = "user://saves/save\_game.dat"

var game\_data: Dictionary = {}

# ==========================

# Inicialización del Sistema

# ==========================

func \_ready() -> void:

\_initialize\_save\_system()

func \_initialize\_save\_system():

load\_template\_data()

ensure\_save\_directory()

SYSLOG.log\_message("Sistema de guardado inicializado correctamente.", "DEBUG", "SAVE")

# ==========================

# Manejo de Directorios

# ==========================

func ensure\_save\_directory():

var dir\_path = save\_path.get\_base\_dir()

if not DirAccess.file\_exists(dir\_path):

var dir = DirAccess.open("user://")

if dir and dir.make\_dir\_recursive(dir\_path) == OK:

SYSLOG.log\_message("Directorio creado: '%s'." % dir\_path, "DEBUG", "SAVE")

else:

SYSLOG.log\_message("No se pudo crear el directorio: '%s'." % dir\_path, "ERROR", "SAVE")

# ==========================

# Gestión de Datos

# ==========================

func load\_template\_data():

if FileAccess.file\_exists(template\_path):

var file = FileAccess.open(template\_path, FileAccess.READ)

if file:

var json\_data = file.get\_as\_text()

file.close()

var result = JSON.parse(json\_data)

if result.error == OK:

game\_data = result.result

SYSLOG.log\_message("Datos cargados desde el template '%s'." % template\_path, "DEBUG", "SAVE")

else:

SYSLOG.log\_message("Error al parsear JSON '%s': %s." % [template\_path, result.error\_string], "ERROR", "SAVE")

else:

SYSLOG.log\_message("El template '%s' no existe." % template\_path, "ERROR", "SAVE")

func save\_game():

ensure\_save\_directory()

var file = FileAccess.open(save\_path, FileAccess.WRITE)

if file:

file.store\_var(game\_data)

file.close()

\_create\_debug\_copy()

SYSLOG.log\_message("Datos guardados en '%s'." % save\_path, "DEBUG", "SAVE")

func load\_game():

if FileAccess.file\_exists(save\_path):

var file = FileAccess.open(save\_path, FileAccess.READ)

if file:

game\_data = file.get\_var()

file.close()

SYSLOG.log\_message("Datos cargados desde '%s'." % save\_path, "DEBUG", "SAVE")

# ==========================

# Depuración

# ==========================

func \_create\_debug\_copy():

var debug\_path = "user://saves/save\_game.json"

var file = FileAccess.open(debug\_path, FileAccess.WRITE)

if file:

file.store\_string(JSON.print(game\_data))

file.close()

CONTROL.gd  
extends Node

# ==========================

# Señales Emitidas

# ==========================

signal control\_exit

# ==========================

# Variables

# ==========================

var savegame\_value: String = null

var pj\_value: String = null

var attack\_value: String = null

# ==========================

# Inicialización

# ==========================

func \_ready() -> void:

\_initialize\_control\_system()

func \_initialize\_control\_system():

get\_tree().connect("tree\_changed", Callable(self, "\_on\_scene\_changed"))

SYSLOG.log\_message("Configurado para detectar cambios de escena.", "DEBUG", "CONTROL")

SYSLOG.log\_message("Script CONTROL inicializado correctamente.", "DEBUG", "CONTROL")

# ==========================

# Gestión de Escenas

# ==========================

func \_on\_scene\_changed():

if get\_tree() and get\_tree().current\_scene:

var current\_scene = get\_tree().current\_scene

SYSLOG.log\_message("Escena actual: %s" % current\_scene.name, "DEBUG", "CONTROL")

else:

SYSLOG.log\_message("No hay escena actual o el árbol ya no está disponible.", "DEBUG", "CONTROL")

func print\_scene\_tree():

if get\_tree() and get\_tree().root:

\_debug\_tree(get\_tree().root)

else:

SYSLOG.log\_message("El árbol de nodos no está disponible.", "DEBUG", "CONTROL")

func \_debug\_tree(node: Node, depth: int = 0):

var indentation = " " \* depth

print(indentation + node.name)

for child in node.get\_children():

\_debug\_tree(child, depth + 1)

# ==========================

# Manejo del Cierre de Ventana

# ==========================

func \_notification(what: int) -> void:

if what == NOTIFICATION\_WM\_CLOSE\_REQUEST:

\_on\_window\_close\_request()

func \_on\_window\_close\_request():

SYSLOG.log\_message("---VENTANA CERRADA---", "DEBUG", "CONTROL")

SAVE.save\_game()

emit\_signal("control\_exit")

get\_tree().quit()

# ==========================

# Gestión de Señales Dinámicas

# ==========================

func connect\_signals(scene\_node: Node):

if not scene\_node:

SYSLOG.log\_message("Nodo no válido para conexión de señales.", "ERROR", "CONTROL")

return

\_connect\_signal(scene\_node, "savegame", "\_on\_savegame\_signal")

\_connect\_signal(scene\_node, "pj", "\_on\_pj\_signal")

\_connect\_signal(scene\_node, "attack", "\_on\_attack\_signal")

func \_connect\_signal(node: Node, signal\_name: String, method\_name: String):

if not node.is\_connected(signal\_name, Callable(self, method\_name)):

node.connect(signal\_name, Callable(self, method\_name))

SYSLOG.log\_message("Señal '%s' conectada." % signal\_name, "DEBUG", "CONTROL")

# ==========================

# Manejadores de Señales

# ==========================

func \_on\_savegame\_signal(selected\_save: String):

savegame\_value = selected\_save

SYSLOG.log\_message("Señal 'savegame' recibida: '%s'." % savegame\_value, "DEBUG", "CONTROL")

func \_on\_pj\_signal(selected\_pj: String):

pj\_value = selected\_pj

SYSLOG.log\_message("Señal 'pj' recibida: '%s'." % pj\_value, "DEBUG", "CONTROL")

func \_on\_attack\_signal(selected\_attack: String):

attack\_value = selected\_attack

SYSLOG.log\_message("Señal 'attack' recibida: '%s'." % attack\_value, "DEBUG", "CONTROL")

Explicación detallada:  
**1. SYSLOG.gd**

**Cambios Realizados**

1. **Inicialización del Sistema:**
   * Consolidé la creación de directorios y las copias de seguridad en \_initialize\_log\_system() para reducir redundancia.
   * Centralicé la creación de directorios en \_create\_directory(path).
2. **Copias de Seguridad:**
   * Simplifiqué la lógica de respaldo en \_backup\_logs, permitiendo modularizar la copia de archivos en una función separada.
3. **Registro de Mensajes:**
   * Simplifiqué el manejo de mensajes utilizando una única función log\_message, que ahora gestiona todos los niveles de log.
4. **Modularización:**
   * Creé \_log\_internal para mensajes relacionados con el sistema SYSLOG, separándolos del resto de logs.

**Beneficios**

* **Reducción de Redundancia:** Funciones consolidadas para crear directorios y registrar mensajes.
* **Claridad:** Bloques funcionales claramente separados.
* **Facilidad de Extensión:** Más sencillo agregar nuevos niveles de log o modificar la lógica de registro.

**2. SAVE.gd**

**Cambios Realizados**

1. **Inicialización:**
   * Centralicé la carga de plantillas y la verificación de directorios en \_initialize\_save\_system().
2. **Validación de Directorios:**
   * Simplifiqué la lógica de verificación y creación de directorios en ensure\_save\_directory().
3. **Gestión de Datos:**
   * Modularicé la carga de la plantilla con validaciones más claras en load\_template\_data().
4. **Depuración:**
   * Separé la creación de copias de depuración en \_create\_debug\_copy() para evitar mezclar responsabilidades en save\_game().

**Beneficios**

* **Legibilidad:** Funciones separadas y organizadas para plantillas, directorios y depuración.
* **Trazabilidad Mejorada:** Registros detallados en SYSLOG para todas las operaciones críticas.
* **Preparación para Producción:** La copia de depuración ahora está claramente aislada, lista para ser eliminada antes del lanzamiento.

**3. CONTROL.gd**

**Cambios Realizados**

1. **Conexión de Señales:**
   * Consolidé la lógica de conexión de señales en \_connect\_signal, eliminando redundancia en connect\_signals.
2. **Gestión de Escenas:**
   * Agrupé funciones relacionadas con cambios de escena en un bloque separado, como \_on\_scene\_changed y print\_scene\_tree.
3. **Manejo del Cierre:**
   * Clarifiqué la secuencia de cierre en \_on\_window\_close\_request, asegurando que todas las operaciones críticas se registren en SYSLOG.
4. **Modularidad:**
   * Separé claramente los manejadores de señales (\_on\_savegame\_signal, \_on\_pj\_signal, \_on\_attack\_signal) en un bloque dedicado.

**Beneficios**

* **Flexibilidad:** Es fácil agregar nuevas señales o cambiarlas.
* **Legibilidad:** Organización clara en bloques funcionales.
* **Resiliencia:** Validaciones adicionales para conexiones y nodos no válidos.