# \*\*Game Design Document (GDD) - SiK Hell\*\*

## \*\*✨ Introducción\*\*

SiK Hell es un juego dungeon crawler en 2D desarrollado en \*\*Godot 4.3\*\*. En este documento se detallan \*\*todos los sistemas clave del juego\*\*, su estado actual y las mejoras planificadas para optimizar el desarrollo y escalabilidad.

---

## \*\*🏷️ Sistemas Principales (Autoloads)\*\*

Los siguientes scripts están configurados como \*\*autoloads\*\*, lo que significa que \*\*se ejecutan en todo momento y son accesibles globalmente.\*\*

### \*\*1. Sistema de Logs (`SYSLOG.gd`)\*\*

\*\*🔧 Responsabilidad:\*\*

- Registrar eventos y errores en el juego.

- \*\*Debe usarse en todos los scripts\*\* para mantener un seguimiento exhaustivo.

\*\*🔍 Observaciones:\*\*

- Funciona correctamente, pero podría beneficiarse de un \*\*mejor filtrado de logs.\*\*

\*\*📊 Métricas de Calidad:\*\*

- \*\*Longitud del Código:\*\* ~150 líneas.

- \*\*Complejidad Ciclomática:\*\* Baja.

- \*\*Acoplamiento:\*\* Bajo.

\*\*👉 Cambios Prioritarios:\*\*

- Agregar niveles de log configurables (DEBUG, INFO, WARNING, ERROR).

- Posible integración con una interfaz de depuración en el juego.

\*\*🔗 Enlace:\*\*

https://github.com/Kavalieri/SiK\_HELL/blob/main/scripts/systems/SYSLOG.gd

---

### \*\*2. Sistema de Guardado (`SAVE.gd`)\*\*

\*\*🔧 Responsabilidad:\*\*

- Gestiona la carga y almacenamiento de datos del juego.

- \*\*Debe usarse siempre\*\* para recuperar o guardar información.

\*\*🔍 Observaciones:\*\*

- Faltan validaciones adicionales para prevenir corrupción de datos.

- Genera una copia de depuración en formato JSON.

\*\*📊 Métricas de Calidad:\*\*

- \*\*Longitud del Código:\*\* ~200 líneas.

- \*\*Complejidad Ciclomática:\*\* Moderada.

- \*\*Acoplamiento:\*\* Moderado.

\*\*👉 Cambios Prioritarios:\*\*

- Implementar cifrado en los archivos de guardado.

- Mejorar detección de errores y manejo de archivos corruptos.

\*\*🔗 Enlace:\*\*

https://github.com/Kavalieri/SiK\_HELL/blob/main/scripts/systems/SAVE.gd

---

### \*\*3. Sistema de Combate (`COMBAT.gd`)\*\*

\*\*🔧 Responsabilidad:\*\*

- Gestiona la lógica de combate: ataques, daño y proyectiles.

\*\*🔍 Observaciones:\*\*

- Alta complejidad, requiere modularización.

- Maneja el pool de proyectiles de manera eficiente.

\*\*📊 Métricas de Calidad:\*\*

- \*\*Longitud del Código:\*\* ~300 líneas.

- \*\*Complejidad Ciclomática:\*\* Alta.

- \*\*Acoplamiento:\*\* Alto.

\*\*👉 Cambios Prioritarios:\*\*

- Separar la lógica de proyectiles y ataques en un `ATTACK\_MANAGER.gd`.

- Unificar el cálculo de daño para evitar redundancia.

\*\*🔗 Enlace:\*\*

https://github.com/Kavalieri/SiK\_HELL/blob/main/scripts/systems/COMBAT.gd

---

### \*\*4. Sistema de Control (`CONTROL.gd`)\*\*

\*\*🔧 Responsabilidad:\*\*

- Gestiona la navegación y otras funciones globales.

\*\*🔍 Observaciones:\*\*

- Actualmente funciona como un "contenedor" de utilidades.

- Se debe evaluar si realmente debe ser un \*\*autoload\*\*.

\*\*📊 Métricas de Calidad:\*\*

- \*\*Longitud del Código:\*\* ~200 líneas.

- \*\*Complejidad Ciclomática:\*\* Moderada.

- \*\*Acoplamiento:\*\* Moderado.

\*\*👉 Cambios Prioritarios:\*\*

- Definir si separamos la navegación en un `NAV\_MANAGER.gd`.

- Optimizar las funciones de gestión de escenas.

\*\*🔗 Enlace:\*\*

https://github.com/Kavalieri/SiK\_HELL/blob/main/scripts/systems/CONTROL.gd

---

## \*\*🌟 Sistemas de Juego\*\*

### \*\*5. Generación de Niveles (`nivel\_1.gd`)\*\*

\*\*🔧 Responsabilidad:\*\*

- Gestiona la carga del jugador, enemigos y la progresión de fases.

\*\*🔍 Observaciones:\*\*

- Código extenso, requiere modularización.

- Mezcla lógica de juego con control de combate.

\*\*📊 Métricas de Calidad:\*\*

- \*\*Longitud del Código:\*\* ~500 líneas.

- \*\*Complejidad Ciclomática:\*\* Alta.

- \*\*Acoplamiento:\*\* Alto.

\*\*👉 Cambios Prioritarios:\*\*

- Modularizar en `LEVEL\_MANAGER.gd`, `ENEMY\_SPAWNER.gd`, etc.

- Separar la gestión de enemigos en un sistema aparte.

\*\*🔗 Enlace:\*\*

https://github.com/Kavalieri/SiK\_HELL/blob/main/scripts/niveles/nivel\_1.gd

---

## \*\*📅 Plan de Refactorización Prioritaria\*\*

### \*\*🔝 Prioridad Alta\*\*

1. Modularizar `nivel\_1.gd`.

2. Crear `ENEMY\_SPAWNER.gd`.

3. Separar lógica de ataques en `ATTACK\_MANAGER.gd`.

### \*\*⭐ Prioridad Media\*\*

4. Mejorar `SAVE.gd` con detección de corrupción de datos.

5. Revisar si `CONTROL.gd` debe ser un autoload.

6. Optimizar la gestión de logs en `SYSLOG.gd`.

---

\*\*💻 Este documento está actualizado con el estado actual del proyecto y es apto para ser entregado al cliente.\*\* 🎯