# GDD: SHUT UP!

## Parte 1: Conceptualización y Resumen Ejecutivo (The High Concept)

### 1. Documento de Visión

1. **Título del Proyecto:** Shut Up!.
2. **Género:** Horror Psicológico / Simulación de Caminata (Walking Sim) / Detección de Micrófono.
3. **Elevator Pitch:** *Shut Up!* es un juego de terror en primera persona donde el jugador debe explorar un entorno hostil durante un tiempo determinado sin emitir sonidos reales; si el micrófono detecta un ruido, el juego se reinicia instantáneamente.
4. **Referencia (X meets Y):** "Don't Scream" conoce a "The Blair Witch Project".
5. **Público Objetivo:** Jugadores "Hardcore" amantes del terror indie y streamers que buscan contenido interactivo de alta tensión.
6. **Unique Selling Points (USP):**
    1. **Detección de Pánico Real:** El silencio absoluto es la única mecánica de defensa.
    2. **Estética de Metraje Encontrado (Bodycam):** Estilo visual de baja carga poligonal oculto tras filtros de grabación analógica.
    3. **Cronómetro de Supervivencia:** El progreso solo avanza si el jugador se mueve, evitando el campeo pasivo.

### 2. Backlog: Epic “High Concept” (GitHub Projects)

Para tu tablero en GitHub, aquí tienes las 4 Historias de Usuario iniciales que definen el núcleo del juego:

| Historia de Usuario | Descripción (Como [rol], quiero [acción], para [beneficio]) | Criterios de Aceptación |
|---|---|---|
| **HU-01: Calibración de Voz** | Como jugador, quiero calibrar mi micrófono al inicio para que el juego reconozca mi nivel de silencio ambiental. | Interfaz de barra de volumen funcional. Configuración de umbral (threshold) guardada. |

| | | |
|---|---|---|
| **HU-02: Detección de Grito** | Como desarrollador, quiero que el sistema dispare un Game Over cuando el input de audio supere los X decibelios. | El juego detecta picos de audio. Transición inmediata a pantalla de reinicio. |
| **HU-03: Renderizado Bodycam** | Como artista, quiero aplicar un shader de distorsión y grano de película para ocultar el estilo Low Poly. | Presencia de aberración cromática, viñeteado y ruido visual constante en la cámara. |
| **HU-04: Temporizador Condicional** | Como diseñador, quiero que el reloj de 18<br>min avance solo si el jugador tiene velocidad >0. | El cronómetro se detiene si el jugador no se mueve. Visualización en UI tipo "timestamp" de video. |

*Table 1. Backlog "High Concept"*

## Parte 2: Mecánicas y Gameplay (El Núcleo MDA)

En esta fase definimos cómo se juega, cómo se comporta el sistema y cómo se gestiona el desarrollo.

### 1. Análisis MDA (Justificación de Diseño)

- **Mechanics (Mecánicas):** El núcleo es el **Mic-Detection**. Usamos la Web Audio API para capturar el flujo de datos. Si el valor *RMS* (Root Mean Square) del audio supera el *threshold* (umbral) configurado, el estado cambia a GameOver. El movimiento está limitado a un eje horizontal *(X, Z)* con un "head bob" que simula el paso humano.
- **Dynamics (Dinámicas):** El jugador experimenta la **"Contención Física"**. Al no poder gritar, el jugador debe regular su respiración real. Surge una dinámica de "Riesgo vs. Tiempo": si caminas rápido, el cronómetro avanza, pero el ruido de los pasos in-game dificulta distinguir si un sonido proviene del entorno o de tu propia habitación.
- **Aesthetics (Estética):** Buscamos la **Sumisión y Temor**. La estética de "Metraje Encontrado" (Bodycam) genera una barrera entre el jugador y la realidad, aumentando la inmersión y la paranoia.

## 2. Diagrama de Flujo del Core Loop

Este diagrama representa el ciclo infinito de juego (gameplay loop) y cómo las condiciones de victoria y derrota rompen ese ciclo, siguiendo la lógica de arquitectura orientada a eventos que definimos anteriormente.

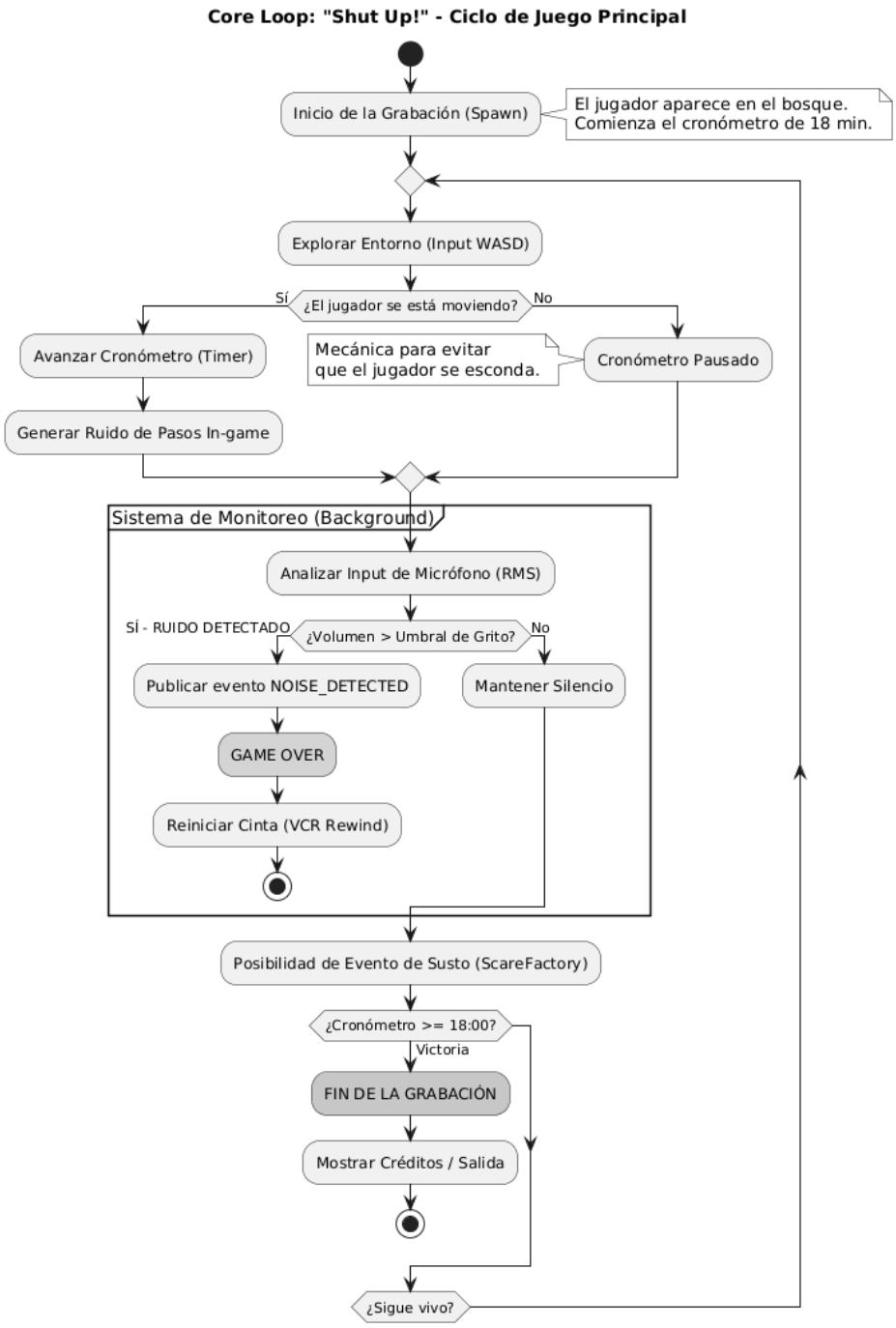


*Figure 1. Diagrama de Flujo Core Loop*

**Explicación del flujo para el GDD:**

1. **Entrada al Bucle:** El jugador comienza en un estado de vulnerabilidad. El tiempo es el recurso que debe "comprar" moviéndose.
2. **Condición de Movimiento:** Si el jugador no se mueve, el cronómetro de 18 minutos se detiene. Esto obliga a la **Dinámica de Exposición**: el jugador debe arriesgarse a caminar y encontrarse con sustos para ganar.
3. **El Vigilante (Micrófono):** Es un proceso paralelo constante. No importa qué esté haciendo el jugador, si hay un pico de sonido, el hilo principal se interrumpe y se dispara el Game Over.
4. **Salida:** Solo hay dos formas de salir del bucle: logrando los 18 minutos de grabación o fallando por ruido/grito.

## 3. Máquina de Estados del Personaje (FSM) en UML

Para la implementación en código (Babylon.js/TypeScript), el personaje se rige por estos estados.

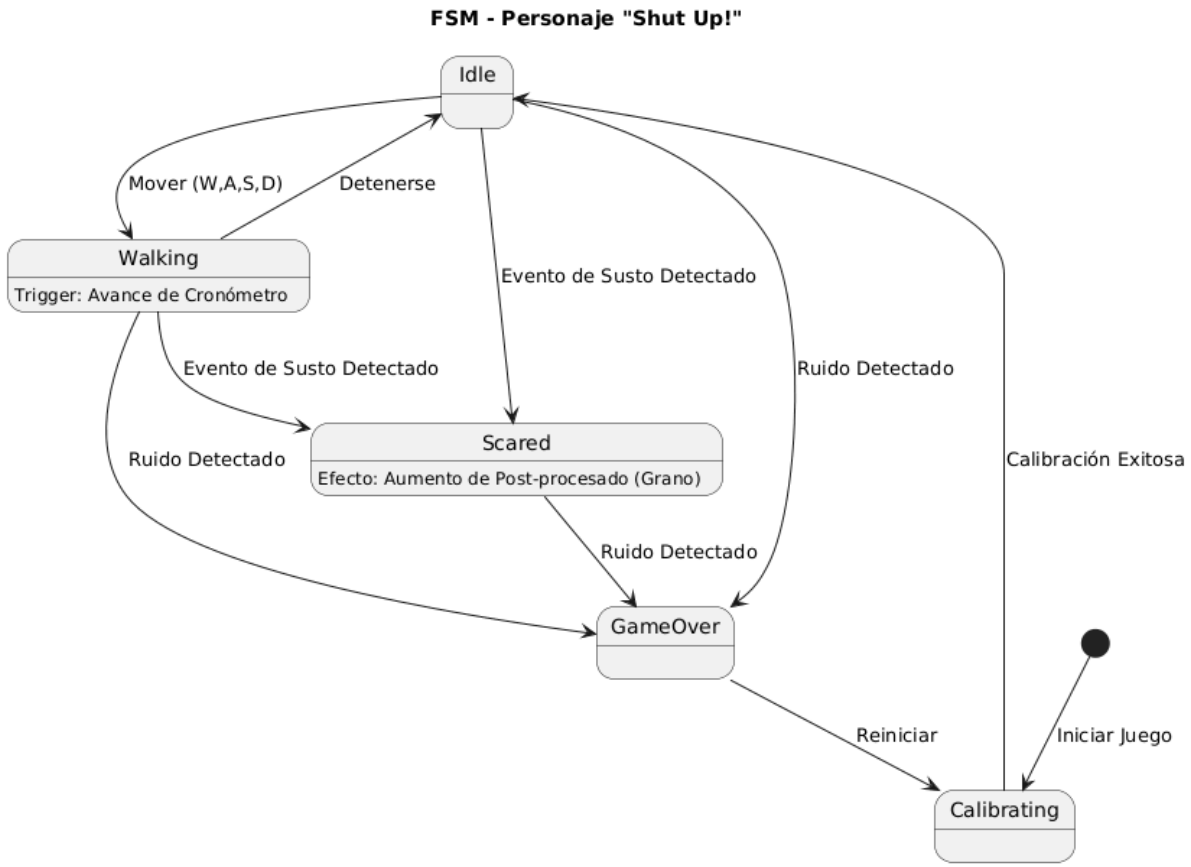


*Figure 2. Máquina de estado del Personaje*

## 4. Gestión del Proyecto

A diferencia de Scrum (que usa Sprints cerrados), en Kanban usaremos un flujo continuo de valor. El tablero en GitHub Projects se dividirá en: Todo, In Progress, Review/QA y Done.

**Backlog Parte 2: Historias de Usuario**

Estas son las historias de usuario técnicas  del backlog:

| ID | Historia de Usuario | Criterios de Aceptación (DoD) | Prioridad |
|---|---|---|---|
| **HU-05** | Filtro Bodycam (Shader) | Shader de post-proceso con aberración cromática y distorsión de lente aplicado a la cámara principal. | Crítica |
| **HU-06** | Detector de Picos de Audio | Script que analice el AudioContext y dispare un evento OnScreamDetected. | Crítica |
| **HU-07** | Cronómetro Condicional | Interfaz de usuario que muestre el tiempo de grabación, el cual solo descuenta si el PlayerState == Walking. | Alta |
| **HU-08** | Generador de Sustos Aleatorios | Sistema que elija un asset de audio/visual de un pool y lo instancie en un radio de 10m del jugador cada N segundos. | Media |

*Table 2. HU Técnicas*

## 5. Sistemas y Datos (Gestión de Recursos)

En **Shut Up!**, el flujo de datos principal es la **Sensibilidad del Micrófono**.

- **Variable Clave:** currentSensitivity. Se calcula como un flotante entre $0.0$ y $1.0$.
- **Persistencia:** El umbral de calibración se guarda en el localStorage del navegador para que el jugador no tenga que calibrar en cada sesión, a menos que cambie de hardware.

## 6. Narrativa y Diseño de Niveles

El nivel es un **Bosque Low Poly**. La técnica para optimizar la web es el **Occlusion Culling**: solo se renderizan los árboles frente a la cámara Bodycam (que tiene un FOV cerrado de 60°), permitiendo una alta densidad de vegetación sin sacrificar FPS.

## Parte 3 : Dinámicas, Sistemas y Economía de Juego

En *Shut Up!*, la "Economía" no se basa en monedas, sino en la gestión de **recursos de supervivencia**: El Silencio, la Energía de la Linterna y la Estabilidad Mental (Estrés).

### 1. GDD: Definición de Límites y Sistemas

**El Sistema de "Economía de Silencio"**

El recurso principal es el **Margen de Ruido**.

- **Límite Crítico ($L_c$):** Definido durante la calibración. Si el input supera este valor, el estado cambia a GameOverState.
- **Decaimiento de Tolerancia:** Cada susto activo reduce temporalmente el umbral de tolerancia en un $15\%$, haciendo que sea más fácil perder si el jugador ya está bajo presión.

**El Sistema de Energía (Linterna)**

- **Capacidad:** 100 unidades (aprox. 3 minutos de uso continuo).
- **Regeneración:** Solo en "Zonas de Luz" fijas (postes de luz parpadeantes).
- **Impacto:** Al llegar a 0, la linterna se apaga y el estrés aumenta un $2 \times$ por segundo.

**El Sistema de Estrés (Economía Visual)**

- **Límite:** 0 a 100 puntos.
- **Penalización:** A mayor estrés, mayor es el valor de la variable u_aberration en el shader de la cámara.
- **Recuperación:** Estar inmóvil en una zona segura o mantener silencio absoluto por 10 segundos.

| Sistema | Recurso | Límite Máximo | Comportamiento |
|---|---|---|---|
| **Vital** | Silencio (Mic) | UmbralCalibrado | Si el volumen RMS supera el límite, se emite EVENT_DEATH. |
| **Visión** | Batería Linterna | 100 unidades | Se agota en 180s. Al llegar a 0, el estrés sube +5/s. |

| | | | |
|---|---|---|---|
| **Mental** | Nivel de Estrés | 100 puntos | Al llegar a 80, la cámara tiembla. Al llegar a 100, el jugador se detiene a jadear (haciendo ruido). |
| **Progreso** | Tiempo Grabado | 1080 segundos | Solo descuenta si la velocidad del jugador >0.1 u/s. |

*Table 3. Límites y Sistemas*

## 2. Diagrama de Flujo: Ciclo de Recursos

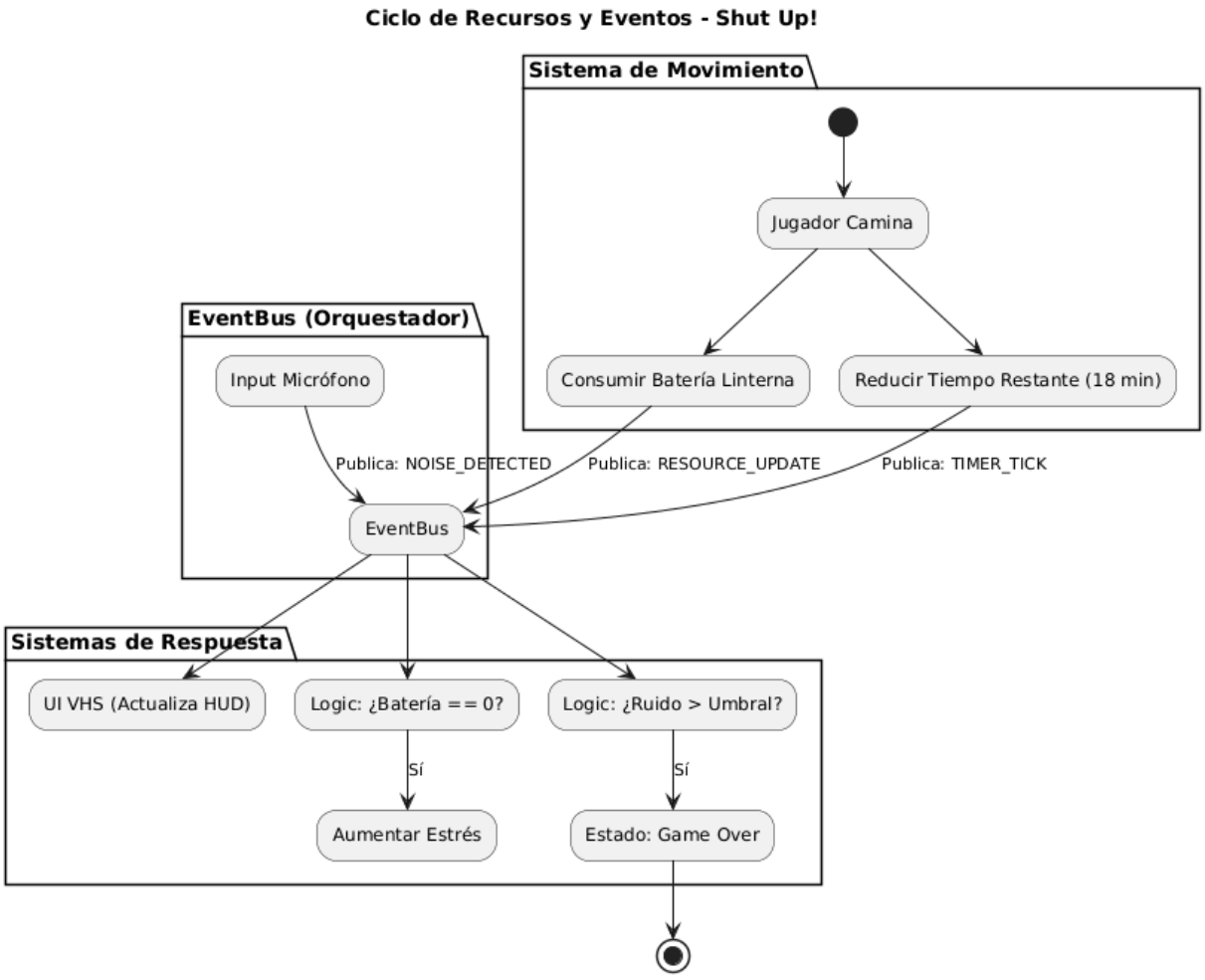


*Figure 3. Ciclo de Recursos*

## 3. Backlog: Lógica de Datos y UI

Estas historias de usuario son los entregables para el  tablero de GitHub Projects, centradas exclusivamente en el flujo de información y la experiencia visual del usuario.

| ID | Historia de Usuario | Criterios de Aceptación | Prioridad |
|---|---|---|---|
| **HU-09** | Sistema de EventBus Global | Implementar un Singleton EventBus en TS que permita subscribe y publish con tipos de eventos fuertes. | Bloqueante |
| **HU-10** | Persistencia de Calibración | Guardar el mic_threshold en localStorage. Al recargar el juego, el sistema debe recuperar el valor sin recalibrar. | Alta |
| **HU-11** | Interfaz de HUD VHS Diegético | Crear un overlay UI que muestre: "REC", el timestamp rojo (00:00:00), y el icono de batería parpadeante. | Alta |
| **HU-12** | Lógica de 'Panic Mode' (UI) | Cuando el estrés sea >80, aplicar un filtro de post-procesado de "distorsión cromática" que pulse al ritmo de un corazón. | Media |
| **HU-13** | Data Binding de Micrófono | Vincular el nivel de entrada del micrófono a una barra visual en el menú de pausa para que el jugador vea su ruido en tiempo real. | Media |

*Table 4. Backlog Logica de datos y UI*

## 4. Código Estructural: Orquestación (TypeScript)

Para que tu arquitectura Event-Driven funcione, necesitamos definir cómo se ven esos datos al circular.

src/core/EventTypes.ts

```
export enum GameEvents {
  TIMER_TICK = "TIMER_TICK",
  NOISE_DETECTED = "NOISE_DETECTED",
  BATTERY_LOW = "BATTERY_LOW",
  SCARE_TRIGGERED = "SCARE_TRIGGERED"
}

export interface IEventPayload {
  value: number | string | object;
  origin: string;
}
```

src/data/initialState.json
Este archivo define el estado "fresco" del juego al iniciar.

```
{
  "player_stats": {
    "stress_level": 0,
    "battery_life": 100,
    "current_time_left": 1080
  },
  "ui_config": {
    "vhs_overlay_enabled": true,
    "show_mic_meter": false,
    "timestamp_format": "HH:mm:ss:ms"
  },
  "physics": {
    "walk_speed": 1.5,
    "run_speed": 3.0,
    "noise_multiplier": 1.2
  }
}
```

**Reflexión de Diseño (MDA):**

Al centralizar todo en el EventBus, el AudioAnalysisSystem no necesita saber que existe una pantalla de "Game Over". Solo grita: "¡Oigan, hubo un ruido de 0.8!". El GameStateManager escucha eso y decide qué hacer. Esto te permite, por ejemplo, cambiar la lógica de muerte por una de "persecución" en el futuro simplemente cambiando quién escucha el evento.

## Parte 4: Narrativa, Mundo y Diseño de Niveles

### 1. GDD: El Bosque de los Susurros (Ambientación)

- **Narrativa:** El jugador despierta tras un accidente cerca de un bosque prohibido. No hay diálogos; la historia se cuenta a través de Props Narrativos (una cámara rota, una tienda de campaña desgarrada, marcas en los árboles).
- **Diseño de Espacios:** El mapa es un "Pasillo Orgánico". Aunque parece un bosque abierto, el diseño utiliza la vegetación densa y la niebla (Fog) para guiar al jugador por una ruta específica que maximiza la tensión.
- **Hito Narrativo:** Al llegar al minuto 15 (de los 18 totales), el entorno pasa de "bosque natural" a "bosque surrealista" (árboles flotando levemente o sombras moviéndose en la periferia).

## 2. Mapa de Nivel: Flujo de Progresión

El diseño se divide en 4 zonas de intensidad creciente.

**Diagrama de Flujo del Nivel (PlantUML)**

Este diagrama describe la progresión lógica del jugador a través de los hitos del nivel.

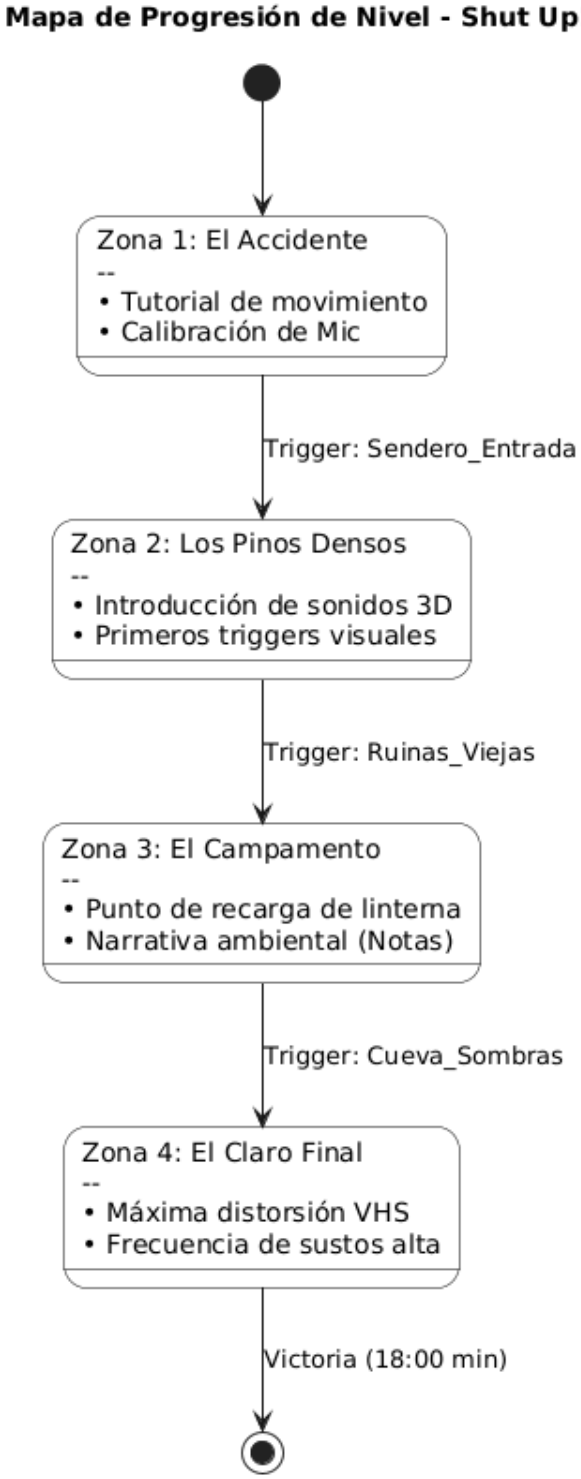


*Figure 4. Mapa de Progresión*

## 3. Tabla de Eventos

| Nombre del Evento | Emisor (Publisher) | Suscriptor (Subscriber) | Descripción / Carga de Datos (Payload) |
|---|---|---|---|
| **MIC_CALIBRATED** | AudioAnalysisSystem | UIController, GameState | Notifica que el umbral de silencio ha sido establecido. Payload: { threshold: float }. |
| **PLAYER_MOVED** | InputSystem | TimerSystem, AudioSystem | Indica que el jugador camina. Activa el cronómetro de 18 min y aumenta el ruido generado. |
| **NOISE_DETECTED** | AudioAnalysisSystem | ThreatController, UI | Crítico. Se dispara si el volumen supera el umbral. Payload: { intensity: float }. |
| **SCARE_TRIGGERED** | EnvironmentSystem | StressSystem, SFXManager | Se dispara al pisar un trigger invisible. Payload: { type: string, stress_impact: int }. |
| **STRESS_CHANGED** | StressSystem | PostProcessManager | Actualiza el nivel de aberración cromática en el shader de la cámara. Payload: { current_stress: 0-100 }. |

| | | | |
|---|---|---|---|
| **BATTERY_UPDATE** | PlayerController | UIController, LightSystem | Actualiza el icono de batería en el HUD y la intensidad de la linterna. Payload: { level: float }. |
| **TIMER_TICK** | TimerSystem | UIController, GameState | Actualiza el timestamp "REC" del HUD cada segundo. Payload: { time_left: int }. |
| **GAME_OVER_STATE** | GameStateManager | Todos los sistemas | Ordena detener todos los procesos y mostrar la pantalla de estática VHS. |

*Figure 5. Tabla de Eventos*

**Dinámica de Interacción: El "Efecto Dominó"**

Para que entiendas cómo esta tabla da vida al juego, veamos un ejemplo de una interacción típica en ¡Shut Up!:

1. **Acción:** El jugador camina sobre una rama seca (Trigger).
2. **Evento 1:** El EnvironmentSystem publica SCARE_TRIGGERED.
3. **Reacción A:** El SFXManager reproduce un sonido de crujido fuerte en 3D.
4. **Reacción B:** El StressSystem recibe el evento y publica un STRESS_CHANGED.
5. **Reacción C:** El PostProcessManager escucha el cambio de estrés y aumenta el grano de película y la distorsión VHS inmediatamente.
6. **Consecuencia:** El jugador se asusta en la vida real, exhala fuerte, el AudioAnalysisSystem capta el ruido y publica NOISE_DETECTED, lo que finalmente lleva al GAME_OVER_STATE.

## 4. Backlog: Mundo y Niveles

| ID | Historia de Usuario | Criterios de Aceptación | Prioridad |
|---|---|---|---|
| **HU-14** | Sistema de Triggers de Eventos | Crear un componente EventTrigger que publique un evento al EventBus cuando el jugador entre en su área. | Crítica |
| **HU-15** | Layout de Bosque Low Poly | Implementar el terreno base con árboles instanciados y neblina densa para ocultar el "popping" de objetos. | Alta |
| **HU-16** | Puntos de Recarga de Linterna | Colocar props (Farolas) que emitan un evento de ENERGY_REPLENISH al estar cerca. | Media |
| **HU-17** | Pool de Sustos Dinámicos | Crear un sistema que posicione un ScareEntity en uno de los 5 puntos aleatorios cerca del jugador tras un trigger. | Alta |

*Table 5. Backlog Mundo y niveles*

## 5. Código Estructural: El Orquestador de Triggers

Siguiendo tu arquitectura de Loose Coupling, el trigger no sabe qué susto va a ocurrir, solo avisa que el jugador pasó por ahí.

**Interfaz del Sistema de Triggers** (src/systems/EnvironmentSystem.ts)

```
/**
 * ITriggerData: Define qué evento dispara este volumen en el espacio
 */
export interface ITriggerData {
    id: string;
    eventType: 'SCARE_TRIGGERED' | 'AMBIENT_CHANGE' | 'CHECKPOINT';
    intensity: number; // Del 0 al 1
    oneShot: boolean;  // ¿Se destruye tras usarse?
}

/**
 * Clase base para Triggers en Babylon.js
 */
export class GameTrigger {
    constructor(
        private mesh: BABYLON.Mesh,
```

```
        private data: ITriggerData,
        private eventBus: EventBus
    ) {
        this.mesh.isVisible = false; // El trigger es invisible
        this.mesh.checkCollisions = false;
    }

    onPlayerEnter() {
        this.eventBus.publish({
            type: this.data.eventType,
            payload: {
                id: this.data.id,
                intensity: this.data.intensity
            },
            timestamp: Date.now()
        });

        if (this.data.oneShot) this.mesh.dispose();
    }
}
```

**JSON de Definición de Nivel (src/data/level_01.json)**

```
{
  "level_name": "Forest of Silence",
  "ambient": {
    "fog_density": 0.08,
    "fog_color": "#0a0a0c",
    "ambient_sound": "wind_howl.mp3"
  },
  "triggers": [
    {
      "id": "first_scare_branch",
      "position": {"x": 10, "y": 0, "z": 50},
      "data": {
        "eventType": "SCARE_TRIGGERED",
        "intensity": 0.3,
        "oneShot": true
      }
    },
    {
      "id": "checkpoint_1",
      "position": {"x": 5, "y": 0, "z": 150},
      "data": {
        "eventType": "CHECKPOINT",
        "intensity": 1,
```

```
      "oneShot": true
    }
   }
  ]
}
```


## 6. Listas de assets

**Assets 3D (Modelos y Entorno)**

Estilo: Low Poly, vértices marcados, sin suavizado (flat shading).

| Asset | Descripción | Prioridad | Presupuesto (Polígonos) |
|---|---|---|---|
| **Player_Bodycam** | Modelo simple de cámara en mano (solo visible en sombras/reflejos). | Media | 800 tris |
| **Forest_Tree_TypeA** | Pino estilizado con ramas en bloque. | Crítica | 300 tris |
| **Forest_Tree_Dead** | Árbol seco sin hojas para zonas de mayor tensión. | Alta | 200 tris |
| **Ground_Rocks** | Conjunto de 3 rocas para bordes de camino. | Baja | 150 tris |
| **Old_Car_Wreck** | Coche accidentado (Punto de inicio). | Alta | 1,200 tris |
| **Abandoned_Tent** | Tienda de campaña desgarrada (Zona de campamento). | Alta | 500 tris |
| **Flashlight_Pickup** | Modelo de linterna de mano para coleccionar. | Media | 300 tris |
| **The_Stalker** | Entidad humanoide alargada y oscura. | Crítica | 2,500 tris |

*Table 6. Assets 3D*

**Assets 2D (Texturas e Interfaz)**

*Estilo: Pixelado, baja resolución, paleta de colores desaturada.*

| Asset | Tipo | Resolución | Descripción |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **VHS_Overlay** | Texture (PNG) | 1024x1024 | Capa de estática y ruido visual. |
| **UI_Icons_Sheet** | Sprite Map | 512x512 | Iconos de batería, REC, y barra de audio. |
| **Ground_Texture** | Seamless | 512x512 | Tierra con hojas secas y barro. |
| **Note_Paper** | Texture | 256x256 | Textura de papel escrito para la narrativa. |
| **Vignette_Mask** | Mask | 512x512 | Máscara de degradado para bordes oscuros. |

*Table 7. Assets 2D*

**Assets de Audio (SFX y Ambiente)**

*Estilo: Audio lo-fi, con distorsión de cinta.*

| Asset | Tipo | Loop | Descripción |
|---|---|---|---|
| **Amb_Forest_Wind** | Ambiente | Sí | Viento sordo y silbidos entre árboles. |
| **Footsteps_Leaves** | SFX | No | Sonido de pasos sobre hojas secas. |
| **Mic_Static_Noise** | SFX | Sí | Ruido blanco de fondo para el micrófono. |
| **Scare_Screech** | SFX | No | Sonido agudo y metálico para jumpscares. |
| **Flashlight_Click** | SFX | No | Sonido mecánico de encendido/apagado. |

*Table 8. Assets de Audio*

## Parte 5: Estética, UI/UX y Arte Técnico

El objetivo es que el jugador olvide que tiene un ratón y un teclado, y sienta que sostiene una Bodycam dañada de los años 90.

### 1. GDD: Diseño de Interfaz Diegética

En **Shut Up!**, la interfaz "vive" dentro del lente de la cámara. No hay barras de vida ni mapas en pantalla.

**Elementos del HUD (Head-Up Display)**

- **Timestamp de Grabación:** Ubicado en la esquina superior derecha. Muestra el tiempo transcurrido en formato 00:00:00:00. Es el cronómetro de 18 minutos.

- **Icono de Batería:** Ubicado en la esquina superior izquierda. Parpadea en rojo cuando queda menos del 20%.
- **Vúmetro de Audio (Audio Meter):** Disfrazado como el nivel de ganancia de la cámara en la parte inferior. Si las barras llegan al área roja, el jugador sabe que está a punto de disparar el GameOver.
- **Filtro "Noise":** Estática sutil que aumenta según el nivel de **Estrés** del jugador.

**Menús e Interacción**

- **Menú Principal:** Se presenta como una pantalla de "Playback/VCR" en un monitor CRT viejo.
- **Pausa:** No detiene el mundo por completo (para mantener la tensión), sino que aplica un efecto de "Pausa de Cinta" (distorsión horizontal fuerte).

## 2. Diagrama de Flujo: Navegación de UI

Este diagrama muestra cómo los estados de la UI cambian mediante el **StateManager** y los eventos del sistema.

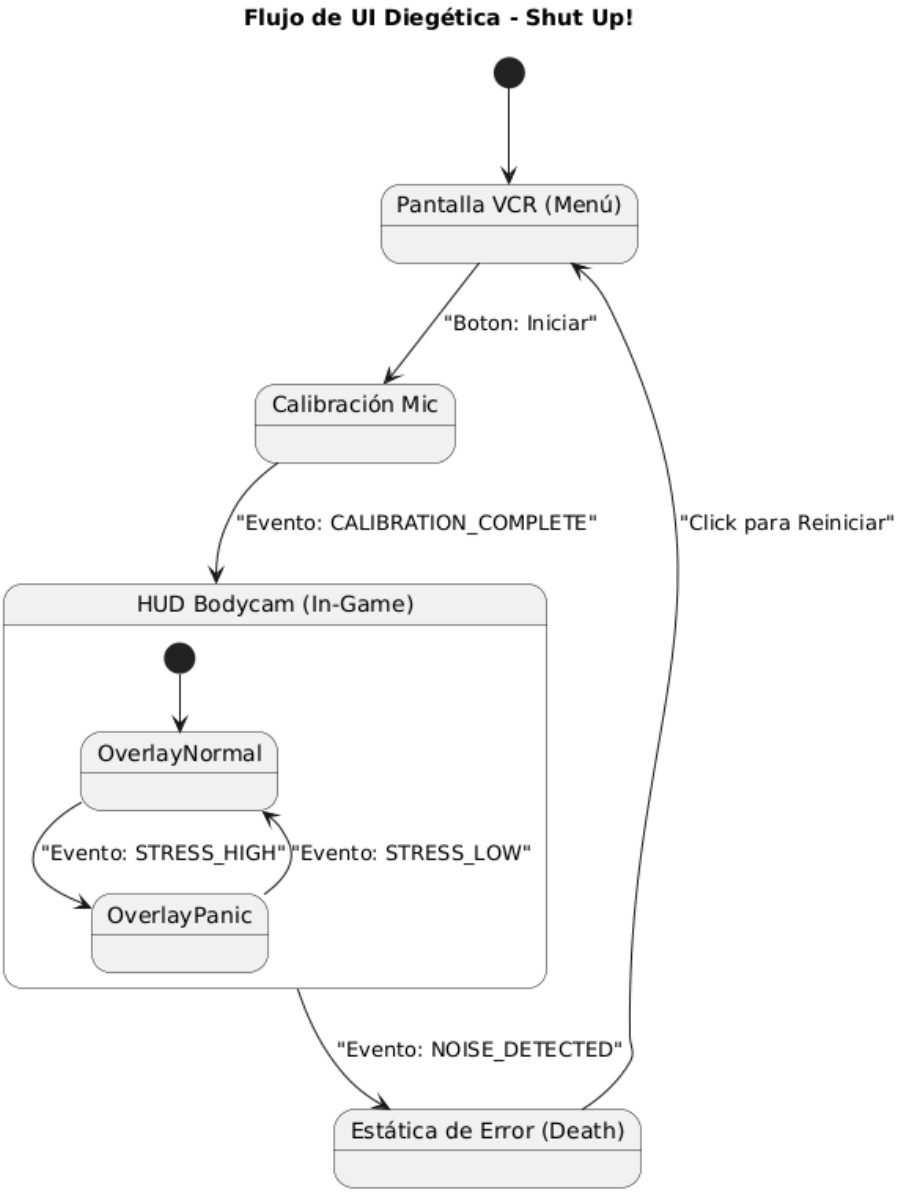


*Figure 6. Navegación de UI*

## 3. Backlog: UI/UX y Arte Técnico

| ID | Historia de Usuario | Criterios de Aceptación | Prioridad |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **HU-18** | **Overlay de Cámara VHS** | Crear una capa de UI (Canvas 2D) con el texto "REC" parpadeante y el timestamp dinámico vinculado al TimerSystem. | **Crítica** |
| **HU-19** | **Visualizador de Micrófono** | Implementar barras de volumen (VU Meter) que reaccionen en tiempo real al input del AudioAnalysisSystem. | Alta |
| **HU-20** | **Shader de Aberración Cromática** | Desarrollar un shader que se intensifique cuando el EventBus publique un STRESS_INCREASE. | Alta |
| **HU-21** | **Feedback de Muerte Visual** | Al detectar ruido, la pantalla debe mostrar "Blue Screen of Death" o estática de video antes de reiniciar. | Media |
| HU-22 | **Optimización de Assets GLB** | Implementar compresión Draco en los modelos del bosque para que el peso total del nivel no supere los 10MB. | Alta |
| HU-23 | **Shader de Vegetación** | Crear un shader de viento simple (Vertex Displacement) para que los árboles se muevan levemente sin usar huesos. | Media |

*Table 9. Backlog UI/UX*

## 4. Código Estructural: Interfaces y Configuración

**Interfaces de UI (src/types/ui.ts)**

Siguiendo el desacoplamiento, la UI solo escucha eventos de datos.

```
export interface IUIState {
  timestamp: string;
  batteryLevel: number; // 0 to 100
  micVolume: number;    // 0 to 1
  isPanicMode: boolean;
}

/**
 * Configuración visual del HUD para el EnvironmentSystem
 */
export interface IHUDSettings {
  scanlineIntensity: number;
  chromaticAberration: number;
  grainAmount: number;
  vignetteStrength: number;
```

}

**JSON de Datos de Estética (src/data/theme.json)**

Este archivo controla el "look" de la cámara sin cambiar el código de los shaders.

```
{
  "vhs_theme": {
    "text_color": "#FF0000",
    "font_family": "VCR_OSD_MONO",
    "overlay_opacity": 0.8,
    "effects": {
      "noise_texture": "assets/textures/noise_grain.png",
      "glitch_frequency": 0.05,
      "color_shift": {
        "r": 0.02,
        "g": -0.01,
        "b": 0.05
      }
    }
  }
}
```

## 5. Arte Técnico: Optimización Web

Para que el estilo **Low Poly** luzca bien en un navegador:

1. **Dithering:** Usaremos un shader de *Dithering* para que los degradados de sombras se vean pixelados (estilo PS1), ocultando la falta de polígonos.
2. **Post-Process Stack:** En Babylon.js, agruparemos los filtros en un DefaultRenderingPipeline para procesarlos en una sola pasada de la GPU, manteniendo los 60 FPS.

**Nota de Diseño:** La UI no debe tener botones brillantes. Todo debe parecer texto de sistema de una cámara Sony o Panasonic de 1996.

## 6. Sección UI/UX con Wireframes (dibujos) de las pantallas principales

Un boceto de interfaz de usuario de baja fidelidad para un videojuego de terror en primera persona llamado "¡Shut Up!". El estilo es un boceto a lápiz dibujado a mano sobre papel cuadriculado texturizado, similar a un plano funcional o una maqueta de UX con anotaciones, flechas y casillas de verificación. Debe tener un aspecto tosco y conceptual, no pulido.

**Pantalla del menú principal**

Un boceto de un antiguo con un menú estilo VCR. El título superior dice "¡Shut up! (MENÚ VCR)". Debajo, opciones seleccionables esbozadas con flechas de cursor rudimentarias: "REPRODUCIR CINTA (INICIO)", "CALIBRAR MICRÓFONO (CONFIGURACIÓN)", "EXPULSAR (SALIDA)". En la parte inferior, un pequeño texto rayado dice "AUTO-TRACKING ACTIVADO". La sensación general es retro y analógica.


*Figure 7. Wireframe Menu*

**In-Game HUD - Diegetic Bodycam**

Un boceto en primera persona que muestra el contorno aproximado de un bosque oscuro con pinos. Los elementos de la interfaz están diseñados para simular superposiciones de videocámara, tal como se solicita en el documento de diseño del juego para una interfaz diegética. En la esquina superior izquierda, un icono cuadrado de batería con "BAT 85%". En la esquina superior derecha, el texto parpadeante "● REC 00:04:15:12" (que representa el temporizador de 18 minutos). A lo largo del borde inferior, un vúmetro de audio horizontal con la etiqueta "GANANCIA DE MICRÓFONO (NIVEL DE RUIDO)" y barras que pulsan hacia un indicador "MÁXIMO" a la derecha. En el centro, una cruz tenue y rayada.

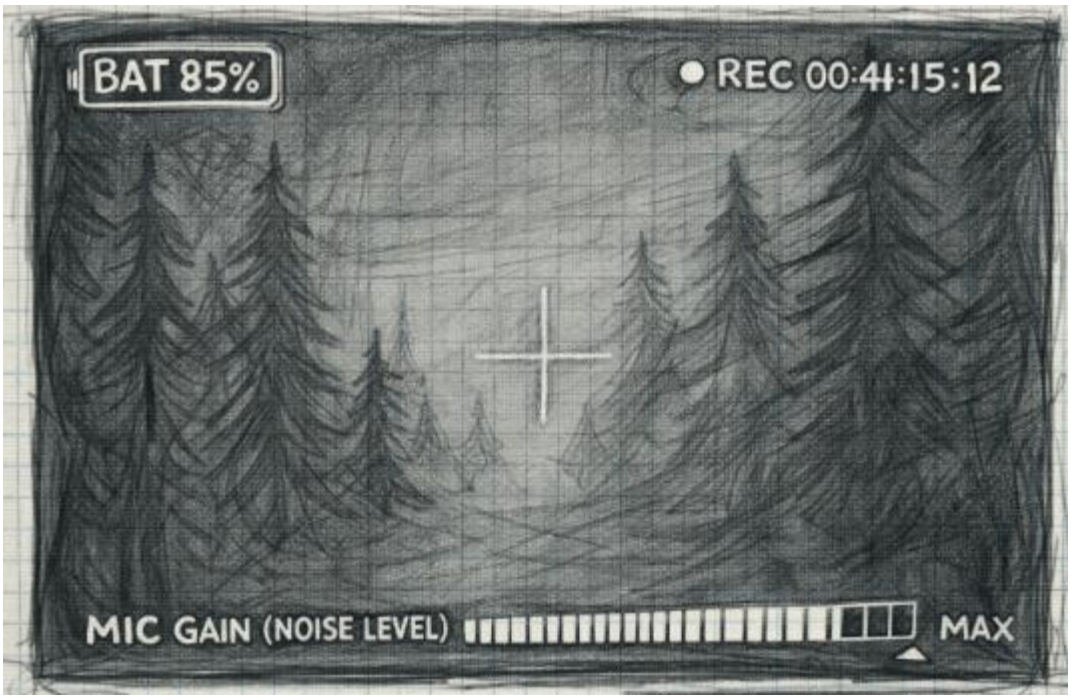


*Figure 8. In-Game HUD*

**Pantalla de Game Over**

Un boceto que representa una pérdida repentina de señal. La vista principal está llena de ruido estático agresivo y garabateado, y líneas distorsionadas. Un texto superpuesto, grande y esbozado en el centro dice "Noise detected". Debajo, un texto más pequeño dice "SIGNAL LOST – RECORDING TERMINATED". En la parte inferior, un mensaje dice "PULSE 'R' PARA REBOBINAR LA CINTA (REINTENTAR)

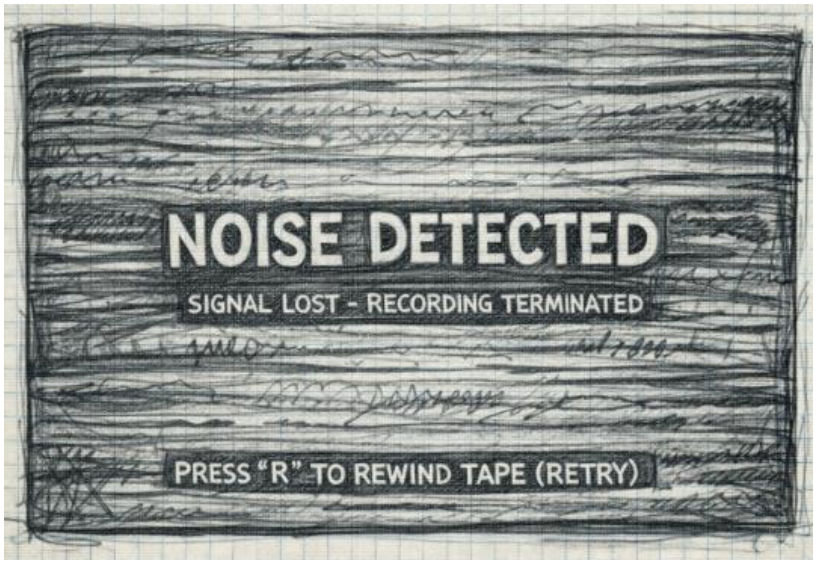


*Figure 9. Game Over*

## 7. Presupuesto de Arte (Art Budget)

El objetivo es mantener una tasa de refresco constante de 60 FPS en navegadores de gama media. El estilo VHS nos permite usar modelos extremadamente simples, ya que el post-procesado (grano y distorsión) "suaviza" las aristas y añade detalle donde no lo hay.

**Tabla de Límites Técnicos**

| Categoría de Asset | Polígonos (Máx. Tris) | Resolución de Textura | Formato de Archivo | Notas |
|---|---|---|---|---|
| **Entorno (Suelo/Terreno)** | 1,500 por sector | 1024x1024 (Atlas) | .glb | Usar "Tiled textures" para el suelo del bosque. |
| **Vegetación (Árboles)** | 200 - 400 | 256x256 o Color plano | .glb | Usar **Instanced Rendering** en Babylon.js. |
| **Props (Notas, Cámaras)** | 50 - 150 | 512x512 | .glb | Texturas con "Pixel Art style" para evitar el blur. |
| **Entidad de Susto (Monstruo)** | 2,500 | 1024x1024 | .glb | Rigging simple (máx. 24 huesos). |
| **Skybox / Niebla** | N/A | Color sólido / Gradiente | N/A | La niebla es un Shader, no un objeto. |

*Table 10. Presupuesto de Arte*

**Especificaciones de Texturas y Shaders**

- **Atlas de Texturas:** Para reducir las llamadas de dibujado (*Draw Calls*), agruparemos todos los objetos pequeños (props) en una sola textura de $1024 \times 1024$.
- **Compresión Draco:** Todos los archivos .glb deben pasar por una compresión **Draco** para reducir el peso de descarga inicial del juego en la web.
- **Mapeo UV:** Se permite un solapamiento de UVs para texturas repetitivas, ya que el filtro de ruido VHS ocultará las costuras de la textura.

**Justificación Técnica (MDA)**

Al limitar los polígonos, liberamos recursos de la GPU para procesar los **Shaders de Post-proceso** (Aberración cromática, Scanlines y Grano), que son los que realmente generan la **Estética** de terror en *Shut Up!*. Sin este presupuesto, el juego se vería como un proyecto de bloques simple; con los filtros, se convierte en un registro de "metraje encontrado" aterrador.

### 8. Asset de Prueba:

*Table 11. Asset cargado .glb*

## Parte 6: Arquitectura Técnica y Stack Tecnológico (Visión de Ingeniería)

Esta fase asegura que el sistema sea robusto y permita a varios desarrolladores trabajar en sistemas paralelos (ej. uno en audio, otro en renderizado) sin pisarse el código.

### 1. Stack Tecnológico

| Tecnología | Herramienta / Librería | Justificación Técnica |
|---|---|---|
| **Motor de Render** | Babylon.js 6.x | Superior a Three.js en el manejo de cámaras, post-procesado avanzado y colisiones nativas. |
| **Lenguaje** | TypeScript 5.x | Evita errores de tipo en los eventos y facilita la colaboración de múltiples desarrolladores. |

| | | |
|---|---|---|
| **Entorno de Desarrollo** | Vite | Compilación instantánea y optimización automática de assets para web. |
| **Análisis de Audio** | Web Audio API | Permite acceso de baja latencia al micrófono y análisis de frecuencia (FFT) en tiempo real. |
| **Modelado 3D** | Blender (Export .glb) | Soporte nativo para compresión Draco, esencial para que el juego cargue rápido en la web. |
| **Shaders** | GLSL / BJS Nodes | Necesario para crear la distorsión analógica de VHS y la aberración cromática diegética. |
| **Gestión de Versiones** | Git + GitHub | Uso de GitHub Projects para Kanban y GitHub Actions para despliegue automático. |

*Table 12. Stack Tecnologico*

## 2. Árbol de Directorios (Estructura del Proyecto)

Esta estructura está diseñada para separar la lógica de negocio (Sistemas) de la representación visual (Escenas), facilitando el **Loose Coupling**.

```
/shut-up-game
├── /.github/workflows       # CI/CD: Automatización de despliegue en GH Pages
├── /public                  # Assets estáticos (sin procesar por Vite)
│   ├── /models              # Archivos .glb comprimidos con Draco
│   ├── /audio               # Archivos .mp3 y .wav (SFX y Ambiente)
│   └── /textures            # Mapas de ruido y overlays VHS
├── /src
│   ├── /core                # El "Cerebro" del motor
│   │   ├── EventBus.ts      # Singleton para Pub/Sub
│   │   ├── StateManager.ts  # Máquina de estados (Menu, Play, Over)
│   │   └── GameLoop.ts      # Orquestador del requestAnimationFrame
│   ├── /systems             # Servicios independientes (Servicios)
│   │   ├── AudioAnalysis.ts # Lógica de Web Audio API
│   │   ├── InputSystem.ts   # Captura de teclado y mouse
│   │   ├── Environment.ts   # Gestión de luces, niebla y triggers
│   │   └── PostProcess.ts   # Configuración de Shaders (VHS/Bodycam)
│   ├── /entities            # Objetos con lógica propia
│   │   ├── Player.ts        # Controlador de cámara y linterna
│   │   └── ScareFactory.ts  # Generador dinámico de sustos
│   ├── /scenes              # Escenas de Babylon.js
│   │   ├── MainMenu.ts
│   │   └── ForestLevel.ts
```

```
│   ├── /ui                # Capa diegética (Canvas 2D/HTML)
│   │   └── HUD.ts         # Overlay de grabación y batería
│   ├── /data              # Configuración en JSON
│   │   └── config.json
│   └── main.ts            # Punto de entrada de la aplicación
├── package.json           # Dependencias y scripts
├── tsconfig.json          # Configuración de TypeScript
└── vite.config.ts         # Configuración de build y assets
```

## 3. Diagrama de clases

**Explicación de la Estructura:**

1. **EventBus (Observer):** Es la clase central. Todas las clases que heredan de BaseSystem tienen acceso a ella para emitir eventos (como un grito detectado) sin saber quién los va a recibir.
2. **StateManager & IGameState (State Pattern):** Permite que el juego se comporte de forma totalmente distinta según el estado. Por ejemplo, en PlayState, el AudioSystem está activo, pero en MenuState podría estar apagado o en modo calibración.
3. **GameEngine (Facade/Singleton):** Es el punto de entrada que inicializa Babylon.js y mantiene el bucle de renderizado (GameLoop). Su única responsabilidad es que el reloj del juego siga avanzando.
4. **BaseSystem:** Es una clase abstracta que asegura que todos los sistemas (Audio, Input, Ambiente) tengan el mismo ciclo de vida (init, update) y acceso al despachador de eventos.

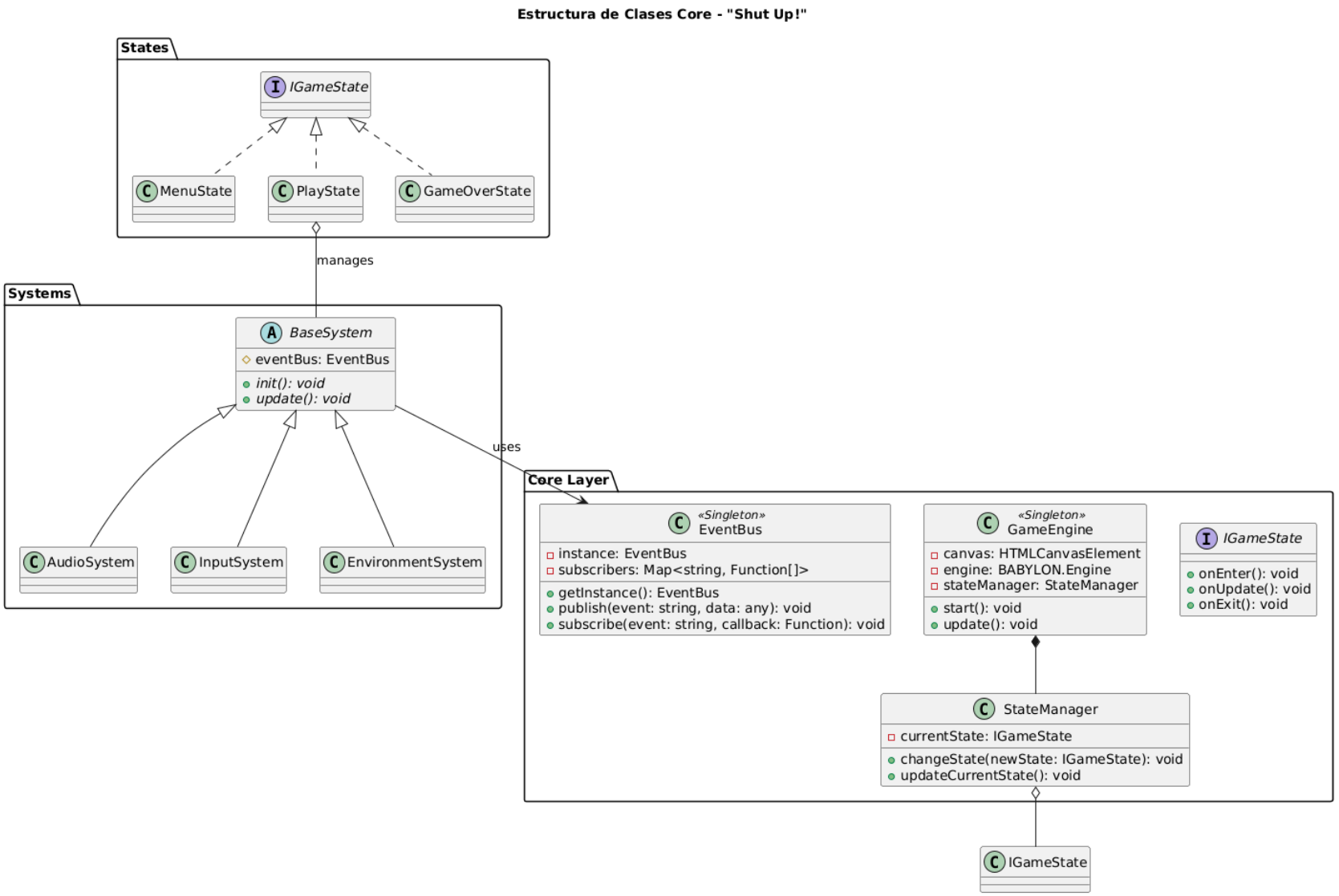


*Figure 10. Diagrama de clases Shut Up!*

## 4. Código Estructural: El Corazón del Sistema

**1. El EventBus (Patrón Observer/Singleton)**

Este módulo permite que el `AudioAnalysisSystem` no necesite saber que el `Player` existe.

```
// src/core/EventBus.ts

type Callback = (payload?: any) => void;


export class EventBus {
  private static instance: EventBus;
  private subscribers: Map<string, Callback[]> = new Map();


  private constructor() {}

  public static getInstance(): EventBus {
    if (!EventBus.instance) EventBus.instance = new EventBus();
    return EventBus.instance;
  }

  public subscribe(event: string, callback: Callback) {
    if (!this.subscribers.has(event)) this.subscribers.set(event, []);
    this.subscribers.get(event)?.push(callback);
  }

  public publish(event: string, payload?: any) {
    this.subscribers.get(event)?.forEach(callback => callback(payload));
  }
}
```

**2. AudioAnalysisSystem (Web Audio API)**

Este sistema se encarga únicamente de la señal, cumpliendo con la Single Responsibility Principle.

```
// src/systems/AudioAnalysisSystem.ts
export class AudioAnalysisSystem {
  private analyzer: AnalyserNode | null = null;
  private dataArray: Uint8Array | null = null;
```

```
  private threshold: number = 0.5;

  async init() {
    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    const audioContext = new AudioContext();
    const source = audioContext.createMediaStreamSource(stream);
    this.analyzer = audioContext.createAnalyser();
    source.connect(this.analyzer);
    this.dataArray = new Uint8Array(this.analyzer.frequencyBinCount);
  }

  update() {
    if (!this.analyzer || !this.dataArray) return;
    this.analyzer.getByteTimeDomainData(this.dataArray);

    // Calcular volumen RMS
    let sum = 0;
    for (let i = 0; i < this.dataArray.length; i++) {
      const val = (this.dataArray[i] - 128) / 128;
      sum += val * val;
    }
    const rms = Math.sqrt(sum / this.dataArray.length);

    if (rms > this.threshold) {
      EventBus.getInstance().publish("NOISE_DETECTED", { volume: rms });
    }
  }
}
```

**Backlog Parte 6: Ingeniería**

| ID | Historia de Usuario | Criterios de Aceptación | Prioridad |
|---|---|---|---|
| HU-24 | Setup de Boilerplate Vite+Babylon | Repositorio configurado con TS, carpetas /src/core, /src/systems y despliegue en GH Pages. | Bloqueante |
| HU-25 | Implementación de StateManager | Sistema que permita cambiar entre MENU, PLAY y GAMEOVER descargando los assets de la escena anterior. | Crítica |
| HU-26 | Integración de Web Audio API | El sistema debe pedir permiso de micrófono y mostrar un log en consola cuando se detecte un pico de sonido. | Alta |
| HU-27 | Factory de Sustos (ScareFactory) | Clase que instancie objetos Scare dinámicamente según el tipo definido en el JSON de la Parte 3. | Media |

*Table 13. Backlog Ingeniería*

**Gestión de Memoria y Garbage Collection**
Para un juego web, es vital evitar las fugas de memoria:

1. Limpieza de Eventos: Al cambiar de estado (ej. de Play a Game Over), el StateManager debe llamar a un método dispose() en todos los sistemas para desuscribir funciones del EventBus.
2. Asset Management: Uso de AssetContainer de Babylon.js para cargar y descargar modelos GLB de forma masiva según la zona del bosque.

## Parte 7: Mantenimiento del GDD Vivo y Planificación Final

En esta etapa, el GDD deja de ser una "promesa" y se convierte en un registro de lo que realmente se construyó.

### 1. GDD Actualizado: Sección de Mantenimiento y QA

Un "GDD Vivo" significa que si durante el testeo descubrimos que 18 minutos es demasiado tiempo para un navegador (por fatiga del jugador o uso de memoria), el documento se actualiza a 10 o 12 minutos.

**Límites de QA para la Web:**

- **Compatibilidad:** El juego debe ser testeado en Chrome, Firefox y Edge.
- **Latencia de Audio:** El sistema de detección de ruido debe tener una latencia inferior a 100ms para que la muerte se sienta justa.
- **Rendimiento:** Mantener un mínimo de 30 FPS en dispositivos móviles de gama alta y 60 FPS en desktop.

## 2. Diagrama de Flujo: Ciclo de Despliegue y Feedback

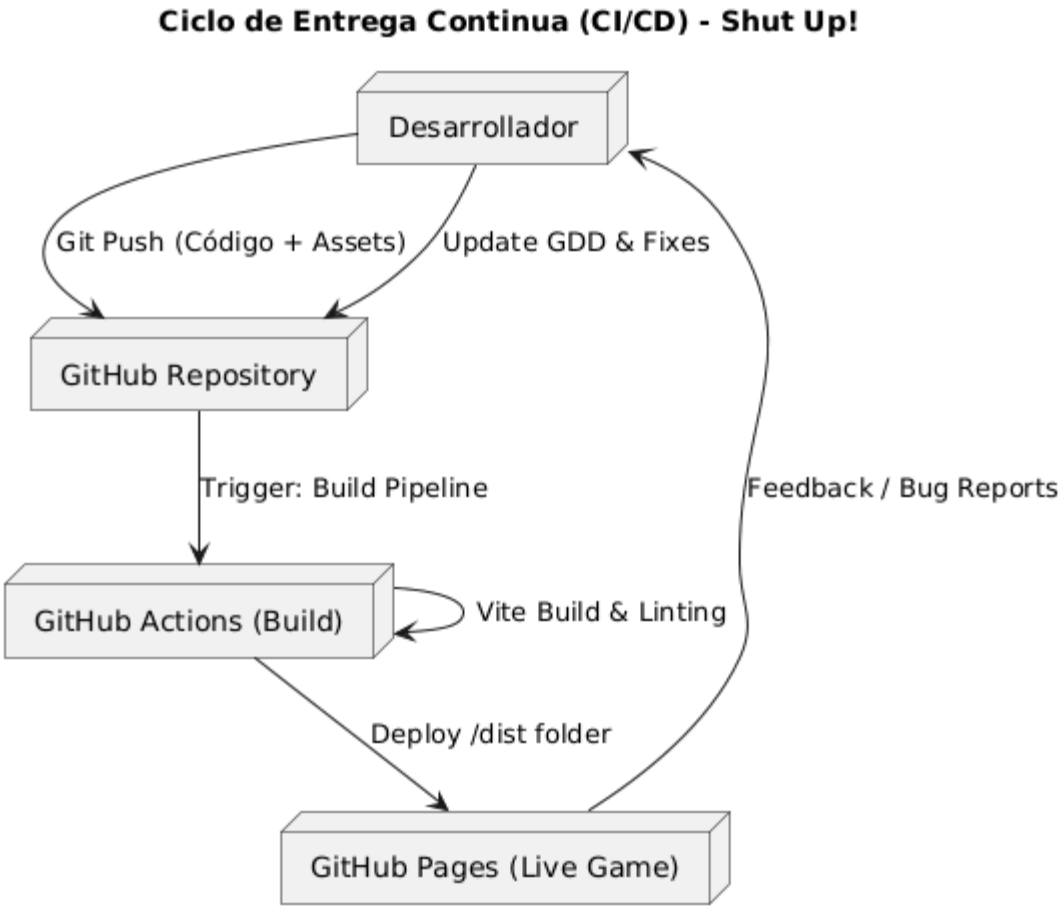


*Figure 11. DF de CI/CD*

## 3. Backlog Final: Historias de Usuario de Cierre

| ID | Historia de Usuario | Criterios de Aceptación | Prioridad |
|---|---|---|---|
| **HU-28** | Pipeline de Despliegue Automático | Configurar GitHub Actions para que el juego se actualice en la URL de GH Pages tras cada "push" a la rama main. | **Crítica** |
| **HU-29** | Sistema de Logs de Errores | Implementar un sistema que capture errores de la Web Audio API y los muestre en una pantalla amigable de "Error de Señal". | Media |
| **HU-30** | Optimización de Carga (Splash Screen) | Crear una pantalla de carga que precargue los assets .glb y los sonidos pesados antes de iniciar el cronómetro. | Alta |

| | | | |
|---|---|---|---|
| **HU-31** | Documentación Post-Mortem | Redactar el análisis final del proyecto (aciertos y errores) dentro del GDD. | Alta |

*Table 14. Backlog FInal*

## 4. Código Estructural: Configuración de Despliegue (YAML)

Como entregable técnico de esta parte, aquí tienes la "receta" para que tu juego sea accesible vía URL, tal como pide la guía para profesionalizar la entrega.

**Archivo:** .github/workflows/deploy.yml

```
name: Deploy Shut Up! to GitHub Pages

on:
  push:
    branches: [ main ]

permissions:
  contents: read
  pages: write
  id-token: write

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Install Dependencies
        run: npm install

      - name: Build Project
        run: npm run build # Ejecuta vite build

      - name: Upload Artifact
        uses: actions/upload-pages-artifact@v1
        with:
          path: './dist'

  deploy:
    needs: build
    runs-on: ubuntu-latest
```

```
  environment:
    name: github-pages
  steps:
    - name: Deploy to GitHub Pages
      id: deployment
      uses: actions/deploy-pages@v1
```

## 5. El Post-Mortem (Análisis de Finalización)

Una vez que el juego esté publicado, el equipo debe completar estas 4 preguntas clave en el GDD para cerrar el ciclo de aprendizaje:

- **¿Qué salió bien?** (Ej: "La estética VHS logró ocultar perfectamente los modelos de bajos polígonos").
- **¿Qué salió mal?** (Ej: "La detección de ruido en micrófonos integrados de laptops era muy sensible y daba falsos positivos").
- **¿Qué aprendimos?** (Ej: "Aprendimos a usar el EventBus para que el sistema de audio no colisionara con el sistema de movimiento").
- **¿Qué haríamos diferente?** (Ej: "Hubiéramos dedicado más tiempo a la calibración automática del micrófono al inicio").