```
#!/usr/bin/env python3
"""
═══════════════════════════════════════════════════════════════════════════
SISTEMA HUNTER - ORACLE MONOLÍTICO UNIFICADO
Diseñado para Google Colab con integración Drive
═══════════════════════════════════════════════════════════════════════════

Filosofía operativa:
1. Observar antes de interpretar
2. Respetar señal cruda
3. Buscar cambios estructurales, no valores absolutos
4. Filtrar con honestidad extrema (E-Veto)
5. Generar evidencia auditable
6. Proteger con gobernanza
7. Trazar todo (eventos y no-eventos)

Autor: Sistema Hunter / Validación Giovanni
Fecha: 2025-12-22
"""

import os
import sys
import time
import json
import math
import hashlib
import smtplib
import ssl
from datetime import datetime, timezone, timedelta
from collections import deque
from dataclasses import dataclass, asdict
from typing import Optional, Dict, List, Tuple, Any

import numpy as np
import pandas as pd
import requests
import ephem
from scipy.stats import linregress, entropy
from scipy.signal import hilbert
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication

# ═══════════════════════════════════════════════════════════════════════════
# PASO 0: MONTAJE DE GOOGLE DRIVE
# ═══════════════════════════════════════════════════════════════════════════

try:
    from google.colab import drive
    drive.mount('/content/drive', force_remount=False)
    WORKSPACE = "/content/drive/MyDrive/HUNTER_WORKSPACE"
    print(f"✓ Drive montado en: {WORKSPACE}")
except:
    WORKSPACE = "./HUNTER_LOCAL"
    print(f"⚠ Modo local: {WORKSPACE}")

# Crear estructura de carpetas
FOLDERS = {
    "ingesta": f"{WORKSPACE}/01_INGESTA",
    "memoria": f"{WORKSPACE}/02_MEMORIA",
```

```
    "metrica": f"{WORKSPACE}/03_METRICA",
    "eveto": f"{WORKSPACE}/04_EVETO",
    "candidatos": f"{WORKSPACE}/05_CANDIDATOS",
    "oracle": f"{WORKSPACE}/06_ORACLE",
    "registro": f"{WORKSPACE}/07_REGISTRO",
    "config": f"{WORKSPACE}/CONFIG"
}

for folder in FOLDERS.values():
    os.makedirs(folder, exist_ok=True)

# ═══════════════════════════════════════════════════════════════════════════
# CONFIGURACIÓN GLOBAL
# ═══════════════════════════════════════════════════════════════════════════

CONFIG = {
    # Umbrales físicos (calibrados)
    "dH_threshold": -0.2,      # Caída de entropía crítica
    "LI_threshold": 0.9,       # Linealidad mínima
    "R2_threshold": 0.95,      # Ajuste del modelo
    "RMSE_threshold": 0.1,     # Error residual máximo
    "coherence_min": 0.7,      # Coherencia vectorial

    # Persistencia temporal
    "persistence_cycles": 3,    # Ciclos mínimos de confirmación
    "dissipation_minutes": 30,  # Tiempo de disipación de falsos positivos
    "memory_window_minutes": 60, # Ventana deslizante

    # Vetos contextuales
    "kp_veto": 5,              # Bloquea si tormenta geomagnética
    "lunar_stress_min": 0.3,   # Estrés tidal mínimo
    "volcanic_radius_km": 50,  # Radio de exclusión volcánica
    "z_score_threshold": 2.0,  # Anomalía estadística regional

    # Operación
    "poll_interval_seconds": 300,  # 5 minutos
    "usgs_lookback_hours": 2,
    "event_magnitude_min": 4.0,

    # Email
    "smtp_server": "smtp.gmail.com",
    "smtp_port": 587,
    "email_from": "tu_email@gmail.com",
    "email_to": ["destinatario@example.com"],
    "email_cooldown_minutes": 60
}

# Guardar configuración
with open(f"{FOLDERS['config']}/hunter_config.json", "w") as f:
    json.dump(CONFIG, f, indent=2)

# ═══════════════════════════════════════════════════════════════════════════
# ESTRUCTURAS DE DATOS
# ═══════════════════════════════════════════════════════════════════════════

@dataclass
class SealResult:
    """Resultado del Doble Sello de validación"""
    passed: bool
    score: float
    reasons: List[str]
    snapshot: Dict[str, Any]
```

```
@dataclass
class CandidateEvent:
    """Expediente de candidato a precursor"""
    event_id: str
    timestamp_utc: str
    latitude: float
    longitude: float
    magnitude: float
    region: str

    # Métricas físicas
    dH: float
    LI: float
    R2: float
    RMSE: float
    coherence: float

    # Contexto
    kp_index: float
    lunar_phase: float
    lunar_stress: float
    z_score: float

    # Estado
    seal_passed: bool
    seal_score: float
    persistence_count: int
    first_detection: str

    def to_dict(self):
        return asdict(self)

# ═══════════════════════════════════════════════════════════════════════════
# 1) CARPETA INGESTA - Observación Pura
# ═══════════════════════════════════════════════════════════════════════════

class IngestaPublica:
    """
    Escucha el mundo sin interpretar.
    Respeta la señal cruda en su secuencia temporal.
    """

    def __init__(self):
        self.cache = {}
        self.cache_ttl = 300  # 5 minutos

    def fetch_usgs_events(self, lookback_hours: int = 2, min_mag: float = 4.0) -> List[Dict]:
        """Obtiene eventos recientes de USGS"""
        cache_key = f"usgs_{lookback_hours}_{min_mag}"

        # Check cache
        if cache_key in self.cache:
            cached_time, cached_data = self.cache[cache_key]
            if time.time() - cached_time < self.cache_ttl:
                return cached_data

        try:
            end_time = datetime.now(timezone.utc)
            start_time = end_time - timedelta(hours=lookback_hours)

            url = (
                f"https://earthquake.usgs.gov/fdsnws/event/1/query?"
                f"format=geojson&"
```

```
            f"starttime={start_time.isoformat()}&"
            f"endtime={end_time.isoformat()}&"
            f"minmagnitude={min_mag}"
        )

        response = requests.get(url, timeout=30)
        response.raise_for_status()
        data = response.json()

        events = []
        for feature in data.get("features", []):
            props = feature["properties"]
            coords = feature["geometry"]["coordinates"]

            events.append({
                "event_id": feature["id"],
                "time_utc": datetime.fromtimestamp(props["time"]/1000, timezone.utc).isoformat(),
                "latitude": coords[1],
                "longitude": coords[0],
                "depth_km": coords[2],
                "magnitude": props["mag"],
                "region": props.get("place", "Unknown"),
                "source": "USGS"
            })

        # Save to ingesta folder
        timestamp = datetime.now(timezone.utc).strftime("%Y%m%d_%H%M%S")
        filepath = f"{FOLDERS['ingesta']}/usgs_{timestamp}.json"
        with open(filepath, "w") as f:
            json.dump(events, f, indent=2)

        self.cache[cache_key] = (time.time(), events)
        return events

    except Exception as e:
        print(f"✗ Error ingesta USGS: {e}")
        return []

def fetch_kp_index(self) -> float:
    """Índice geomagnético Kp actual"""
    try:
        url = "https://services.swpc.noaa.gov/products/noaa-planetary-k-index.json"
        r = requests.get(url, timeout=10)
        data = r.json()
        # Última fila, columna Kp
        kp = float(data[-1][1])
        return kp
    except:
        return 0.0  # Neutro si falla

def calculate_lunar_context(self, lat: float, lon: float) -> Dict:
    """Fase y estrés lunar para coordenadas específicas"""
    try:
        observer = ephem.Observer()
        observer.lat = str(lat)
        observer.lon = str(lon)
        observer.date = datetime.now(timezone.utc)

        moon = ephem.Moon(observer)
        phase = moon.phase  # 0-100
        altitude = float(moon.alt) * 180 / np.pi  # Radianes a grados

        # Estrés heurístico: combinación fase + altitud
```

```
            stress = (phase / 100) * (1 + abs(altitude) / 90)

            return {
                "phase": phase,
                "altitude": altitude,
                "stress": stress
            }
        except:
            return {"phase": 0, "altitude": 0, "stress": 0}

# ═══════════════════════════════════════════════════════════════════════
# 2) CARPETA MEMORIA - Ventanas Deslizantes
# ═══════════════════════════════════════════════════════════════════════

class MemoriaOperativa:
    """
    Mantiene ventanas deslizantes de señal.
    Pregunta: ¿Cómo está cambiando lo que pasa ahora?
    """

    def __init__(self, window_minutes: int = 60):
        self.window_size = window_minutes
        self.pressure_buffer = deque(maxlen=window_minutes)
        self.accel_buffer = deque(maxlen=window_minutes)
        self.event_buffer = deque(maxlen=100)

    def add_pressure_sample(self, value: float):
        """Agrega muestra de presión (simulada o real)"""
        self.pressure_buffer.append({
            "timestamp": datetime.now(timezone.utc).isoformat(),
            "value": value
        })

    def add_accel_sample(self, x: float, y: float, z: float):
        """Agrega muestra de aceleración 3D"""
        self.accel_buffer.append({
            "timestamp": datetime.now(timezone.utc).isoformat(),
            "x": x, "y": y, "z": z
        })

    def add_event(self, event: Dict):
        """Registra evento en memoria"""
        self.event_buffer.append(event)

    def get_pressure_series(self) -> np.ndarray:
        """Retorna serie temporal de presión"""
        if not self.pressure_buffer:
            return np.array([])
        return np.array([s["value"] for s in self.pressure_buffer])

    def get_accel_matrix(self) -> np.ndarray:
        """Retorna matriz [N x 3] de aceleración"""
        if not self.accel_buffer:
            return np.array([])
        return np.array([[s["x"], s["y"], s["z"]] for s in self.accel_buffer])

    def save_state(self):
        """Persiste memoria en disco"""
        state = {
            "pressure": list(self.pressure_buffer),
            "accel": list(self.accel_buffer),
            "events": list(self.event_buffer)
        }
```

```
        timestamp = datetime.now(timezone.utc).strftime("%Y%m%d_%H%M%S")
        filepath = f"{FOLDERS['memoria']}/state_{timestamp}.json"
        with open(filepath, "w") as f:
            json.dump(state, f)

# ═══════════════════════════════════════════════════════════════════════════
# 3) CARPETA MÉTRICA - Entropía y Coherencia
# ═══════════════════════════════════════════════════════════════════════════

class MetricaFisica:
    """
    Calcula cambios estructurales:
    - Entropía (dispersión)
    - Linealidad (organización temporal)
    - Coherencia (organización espacial)

    Busca: caídas de entropía + aumento de orden.
    """

    @staticmethod
    def compute_delta_H(series: np.ndarray) -> float:
        """
        ΔH: Cambio de entropía relativa
        Negativo = señal se ordena (posible precursor)
        """
        if len(series) < 10:
            return 0.0

        # Dividir en dos mitades
        mid = len(series) // 2
        half1 = series[:mid]
        half2 = series[mid:]

        # Entropía gaussiana aproximada
        var1 = np.var(half1) if np.var(half1) > 0 else 1e-6
        var2 = np.var(half2) if np.var(half2) > 0 else 1e-6

        H1 = 0.5 * np.log(2 * np.pi * np.e * var1)
        H2 = 0.5 * np.log(2 * np.pi * np.e * var2)

        return H2 - H1  # Negativo si H disminuye

    @staticmethod
    def compute_LI(series: np.ndarray) -> Tuple[float, float, float]:
        """
        LI: Índice de linealidad basado en R²
        Retorna: (LI, R², RMSE)
        """
        if len(series) < 3:
            return 0.0, 0.0, 1.0

        x = np.arange(len(series))
        y = series

        try:
            slope, intercept, r_value, _, _ = linregress(x, y)
            R2 = r_value ** 2

            y_pred = slope * x + intercept
            rmse = np.sqrt(np.mean((y - y_pred) ** 2))

            # LI = R² ajustado por RMSE
            LI = R2 if rmse < 0.1 else R2 * (0.1 / (rmse + 1e-6))
```

```
            return LI, R2, rmse
        except:
            return 0.0, 0.0, 1.0

    @staticmethod
    def compute_coherence(accel_matrix: np.ndarray) -> float:
        """
        Coherencia vectorial: ¿Movimiento direccional o caótico?
        1.0 = orden perfecto, 0.0 = ruido isotrópico
        """
        if len(accel_matrix) < 5:
            return 0.0

        try:
            # Varianza por eje
            var_x = np.var(accel_matrix[:, 0])
            var_y = np.var(accel_matrix[:, 1])
            var_z = np.var(accel_matrix[:, 2])

            total_var = var_x + var_y + var_z
            if total_var < 1e-6:
                return 0.0

            # Dominancia del eje principal
            max_var = max(var_x, var_y, var_z)
            coherence = max_var / total_var

            return coherence
        except:
            return 0.0

    def analizar_evento(self, memoria: MemoriaOperativa) -> Dict[str, float]:
        """Pipeline completo de métricas"""
        pressure = memoria.get_pressure_series()
        accel = memoria.get_accel_matrix()

        dH = self.compute_delta_H(pressure) if len(pressure) > 0 else 0.0
        LI, R2, RMSE = self.compute_LI(pressure) if len(pressure) > 0 else (0.0, 0.0, 1.0)
        coherence = self.compute_coherence(accel) if len(accel) > 0 else 0.0

        metricas = {
            "dH": dH,
            "LI": LI,
            "R2": R2,
            "RMSE": RMSE,
            "coherence": coherence,
            "timestamp": datetime.now(timezone.utc).isoformat()
        }

        # Guardar en carpeta métrica
        timestamp = datetime.now(timezone.utc).strftime("%Y%m%d_%H%M%S")
        filepath = f"{FOLDERS['metrica']}/metrics_{timestamp}.json"
        with open(filepath, "w") as f:
            json.dump(metricas, f, indent=2)

        return metricas

# ═══════════════════════════════════════════════════════════════════════
# 4) CARPETA E-VETO - Filtro de Honestidad
# ═══════════════════════════════════════════════════════════════════════

class EVetoEngine:
```

```
"""
Filtro estricto: aprende a callar.
Mejor perder 100 señales que aceptar 1 falsa.

Doble Sello:
1) E-Veto: dH <= threshold
2) Σ-Metrics: LI, R², RMSE, coherence
"""

def __init__(self, config: Dict):
    self.config = config

def apply_double_seal(self, metricas: Dict) -> SealResult:
    """
    Doble Sello canónico:
    Solo pasa si TODAS las condiciones se cumplen.
    """
    reasons = []
    score = 0.0

    # 1) E-Veto: Entropía
    if metricas["dH"] > self.config["dH_threshold"]:
        reasons.append(f"E-Veto FAIL: dH={metricas['dH']:.3f} > {self.config['dH_threshold']}")
    else:
        score += 25
        reasons.append(f"E-Veto PASS: dH={metricas['dH']:.3f}")

    # 2) Σ-Metrics: Linealidad
    if metricas["LI"] < self.config["LI_threshold"]:
        reasons.append(f"LI FAIL: {metricas['LI']:.3f} < {self.config['LI_threshold']}")
    else:
        score += 25
        reasons.append(f"LI PASS: {metricas['LI']:.3f}")

    # 3) Σ-Metrics: R²
    if metricas["R2"] < self.config["R2_threshold"]:
        reasons.append(f"R² FAIL: {metricas['R2']:.3f} < {self.config['R2_threshold']}")
    else:
        score += 25

    # 4) Σ-Metrics: RMSE
    if metricas["RMSE"] > self.config["RMSE_threshold"]:
        reasons.append(f"RMSE FAIL: {metricas['RMSE']:.3f} > {self.config['RMSE_threshold']}")
    else:
        score += 15

    # 5) Coherencia espacial
    if metricas["coherence"] < self.config["coherence_min"]:
        reasons.append(f"Coherence FAIL: {metricas['coherence']:.3f} < {self.config['coherence_min']}")
    else:
        score += 10

    passed = (score >= 75)  # 75/100 puntos mínimo

    result = SealResult(
        passed=passed,
        score=score,
        reasons=reasons,
        snapshot=metricas.copy()
    )

    # Guardar decisión
    timestamp = datetime.now(timezone.utc).strftime("%Y%m%d_%H%M%S")
```

```
        filepath = f"{FOLDERS['eveto']}/seal_{timestamp}.json"
        with open(filepath, "w") as f:
            json.dump({
                "passed": passed,
                "score": score,
                "reasons": reasons,
                "metrics": metricas
            }, f, indent=2)

        return result

    def test_shuffle_significance(self, LI_original: float, pressure_data: np.ndarray, trials: int = 100) -> bool:
        """
        Test adversarial: ¿Es el orden una casualidad?
        Baraja los datos y recalcula LI. Si el original es significativamente mejor, pasa.
        """
        if len(pressure_data) < 10:
            return False

        fake_LIs = []
        for _ in range(trials):
            shuffled = pressure_data.copy()
            np.random.shuffle(shuffled)
            fake_LI, _, _ = MetricaFisica.compute_LI(shuffled)
            fake_LIs.append(fake_LI)

        threshold = np.percentile(fake_LIs, 95)  # 95º percentil del ruido
        margin = 0.1

        passed = LI_original > (threshold + margin)

        print(f"  Shuffle Test: LI_real={LI_original:.3f}, threshold_95={threshold:.3f}, passed={passed}")
        return passed

# ═══════════════════════════════════════════════════════════════════════════
# 5) CARPETA CANDIDATOS - Generación de Expedientes
# ═══════════════════════════════════════════════════════════════════════════

class GeneradorCandidatos:
    """
    Solo si E-Veto pasa: genera expediente auditable.
    No es predicción, es evidencia empaquetada.
    """

    def __init__(self, ingesta: IngestaPublica):
        self.ingesta = ingesta
        self.persistence_tracker = {}  # {event_id: count}

    def create_candidate(self, event: Dict, metricas: Dict, seal: SealResult) -> Optional[CandidateEvent]:
        """Construye expediente completo"""

        # Contexto cósmico
        kp = self.ingesta.fetch_kp_index()
        lunar = self.ingesta.calculate_lunar_context(event["latitude"], event["longitude"])

        # Z-score regional (simplificado - necesitaría histórico real)
        z_score = 0.0  # Placeholder

        event_id = event["event_id"]

        # Persistencia
        if event_id not in self.persistence_tracker:
            self.persistence_tracker[event_id] = {
```

```
                "count": 1,
                "first_seen": datetime.now(timezone.utc).isoformat()
            }
        else:
            self.persistence_tracker[event_id]["count"] += 1

        persistence_count = self.persistence_tracker[event_id]["count"]
        first_detection = self.persistence_tracker[event_id]["first_seen"]

        candidate = CandidateEvent(
            event_id=event_id,
            timestamp_utc=event["time_utc"],
            latitude=event["latitude"],
            longitude=event["longitude"],
            magnitude=event["magnitude"],
            region=event["region"],

            dH=metricas["dH"],
            LI=metricas["LI"],
            R2=metricas["R2"],
            RMSE=metricas["RMSE"],
            coherence=metricas["coherence"],

            kp_index=kp,
            lunar_phase=lunar["phase"],
            lunar_stress=lunar["stress"],
            z_score=z_score,

            seal_passed=seal.passed,
            seal_score=seal.score,
            persistence_count=persistence_count,
            first_detection=first_detection
        )

        # Guardar expediente
        timestamp = datetime.now(timezone.utc).strftime("%Y%m%d_%H%M%S")
        filepath = f"{FOLDERS['candidatos']}/{event_id}_{timestamp}.json"
        with open(filepath, "w") as f:
            json.dump(candidate.to_dict(), f, indent=2)

        return candidate

# ═══════════════════════════════════════════════════════════════════════════
# 6) CARPETA ORACLE - Gobernanza
# ═══════════════════════════════════════════════════════════════════════════

class OracleGovernance:
    """
    Decisión final: ¿Merece atención humana/institucional?

    Puede: archivar, observar, escalar, cerrar.
    Protege contra sobreinterpretación.
    """

    def __init__(self, config: Dict):
        self.config = config
        self.last_email_time = {}

    def evaluate_candidate(self, candidate: CandidateEvent) -> Dict[str, Any]:
        """
        Juicio del Oracle: decisión gobernada
        """
        verdict = {
```

```
            "event_id": candidate.event_id,
            "timestamp": datetime.now(timezone.utc).isoformat(),
            "decision": "OBSERVE",  # Default
            "confidence": 0,
            "actions": [],
            "vetos": []
        }

        # Veto 1: Kp geomagnético
        if candidate.kp_index >= self.config["kp_veto"]:
            verdict["vetos"].append(f"KP_VETO: Kp={candidate.kp_index:.1f} >= {self.config['kp_veto']}")
            verdict["decision"] = "ARCHIVE"
            verdict["confidence"] = 0
            return verdict

        # Veto 2: Estrés lunar insuficiente
        if candidate.lunar_stress < self.config["lunar_stress_min"]:
            verdict["vetos"].append(f"LUNAR_VETO: stress={candidate.lunar_stress:.2f} <
{self.config['lunar_stress_min']}")

        # Veto 3: No pasó doble sello
        if not candidate.seal_passed:
            verdict["vetos"].append("SEAL_VETO: Double seal failed")
            verdict["decision"] = "ARCHIVE"
            return verdict

        # Veto 4: Persistencia insuficiente
        if candidate.persistence_count < self.config["persistence_cycles"]:
            verdict["vetos"].append(f"PERSISTENCE_VETO: {candidate.persistence_count} <
{self.config['persistence_cycles']} cycles")
            verdict["decision"] = "OBSERVE"
            verdict["confidence"] = 30
            return verdict

        # ✓ Pasó todos los vetos
        verdict["decision"] = "ESCALATE"
        verdict["confidence"] = min(95, candidate.seal_score + 20)
        verdict["actions"].append("NOTIFY_HUMAN")
        verdict["actions"].append("GENERATE_REPORT")

        # Guardar decisión
        filepath = f"{FOLDERS['oracle']}/{candidate.event_id}_verdict.json"
        with open(filepath, "w") as f:
            json.dump(verdict, f, indent=2)

        return verdict

    def should_send_email(self, event_id: str) -> bool:
        """Control de cooldown para emails"""
        now = time.time()
        cooldown = self.config["email_cooldown_minutes"] * 60

        if event_id in self.last_email_time:
            elapsed = now - self.last_email_time[event_id]
            if elapsed < cooldown:
                return False

        self.last_email_time[event_id] = now
        return True

    def send_alert_email(self, candidate: CandidateEvent, verdict: Dict):
        """Envía alerta por email (si configurado)"""
```

```
        if not self.should_send_email(candidate.event_id):
            print(f"  Email cooldown activo para {candidate.event_id}")
            return

        try:
            # Cargar credenciales
            cred_path = f"{FOLDERS['config']}/email_credentials.json"
            if not os.path.exists(cred_path):
                print(f"  ⚠ No se encontró {cred_path}")
                return

            with open(cred_path) as f:
                creds = json.load(f)

            msg = MIMEMultipart()
            msg["From"] = creds["email_from"]
            msg["To"] = ", ".join(creds["email_to"])
            msg["Subject"] = f"🚨 HUNTER ALERT: {candidate.region} M{candidate.magnitude}"

            body = f"""
SISTEMA HUNTER - DETECCIÓN DE PRECURSOR
═══════════════════════════════════════════════════════════

EVENTO: {candidate.event_id}
Región: {candidate.region}
Magnitud: M{candidate.magnitude}
Coordenadas: {candidate.latitude:.3f}°N, {candidate.longitude:.3f}°E
Tiempo UTC: {candidate.timestamp_utc}

MÉTRICAS FÍSICAS
─────────────────────────────────────────────────────────
ΔH (Entropía):        {candidate.dH:.4f} {'✓' if candidate.dH <= -0.2 else '✗'}
LI (Linealidad):      {candidate.LI:.4f} {'✓' if candidate.LI >= 0.9 else '✗'}
R² (Ajuste):          {candidate.R2:.4f} {'✓' if candidate.R2 >= 0.95 else '✗'}
RMSE:                 {candidate.RMSE:.4f} {'✓' if candidate.RMSE <= 0.1 else '✗'}
Coherencia:           {candidate.coherence:.4f}

CONTEXTO CÓSMICO
─────────────────────────────────────────────────────────
Kp Index:             {candidate.kp_index:.1f} {'⚠ TORMENTA' if candidate.kp_index >= 5 else '✓'}
Fase Lunar:           {candidate.lunar_phase:.1f}%
Estrés Lunar:         {candidate.lunar_stress:.3f}

VALIDACIÓN
─────────────────────────────────────────────────────────
Doble Sello:          {'✓ PASSED' if candidate.seal_passed else '✗ FAILED'}
Score:                {candidate.seal_score}/100
Persistencia:         {candidate.persistence_count} ciclos
Primera Detección:    {candidate.first_detection}

DECISIÓN ORACLE
─────────────────────────────────────────────────────────
Veredicto:            {verdict['decision']}
Confianza:            {verdict['confidence']}%
Acciones:             {', '.join(verdict['actions']) if verdict['actions'] else 'Ninguna'}
Vetos:                {', '.join(verdict['vetos']) if verdict['vetos'] else 'Ninguno'}

═══════════════════════════════════════════════════════════
Este mensaje fue generado automáticamente por el Sistema Hunter.
Para más información, consulte el expediente completo en Drive.
═══════════════════════════════════════════════════════════
"""
```

```
            msg.attach(MIMEText(body, "plain"))

            # Adjuntar expediente JSON
            json_data = json.dumps(candidate.to_dict(), indent=2)
            attachment = MIMEApplication(json_data.encode(), _subtype="json")
            attachment.add_header("Content-Disposition", "attachment", filename=f"{candidate.event_id}.json")
            msg.attach(attachment)

            # Enviar
            context = ssl.create_default_context()
            with smtplib.SMTP(creds["smtp_server"], creds["smtp_port"]) as server:
                server.starttls(context=context)
                server.login(creds["email_from"], creds["password"])
                server.send_message(msg)

            print(f"  ✓ Email enviado para {candidate.event_id}")

        except Exception as e:
            print(f"  ✗ Error enviando email: {e}")

# ═══════════════════════════════════════════════════════════════════════════
# 7) CARPETA REGISTRO - Trazabilidad Total
# ═══════════════════════════════════════════════════════════════════════════

class RegistroTrazable:
    """
    Registra TODO: eventos, no-eventos, decisiones, silencio.
    La ausencia de señal también es información.
    """

    def __init__(self):
        self.log_path = f"{FOLDERS['registro']}/hunter_master.jsonl"

    def log_event(self, event_type: str, data: Dict):
        """Añade entrada al log maestro"""
        entry = {
            "timestamp": datetime.now(timezone.utc).isoformat(),
            "event_type": event_type,
            "data": data
        }

        with open(self.log_path, "a") as f:
            f.write(json.dumps(entry) + "\n")

    def log_cycle(self, cycle_num: int, events_found: int, candidates_generated: int, alerts_sent: int):
        """Registra ciclo de ejecución"""
        self.log_event("CYCLE", {
            "cycle_number": cycle_num,
            "events_found": events_found,
            "candidates_generated": candidates_generated,
            "alerts_sent": alerts_sent
        })

    def log_no_activity(self):
        """Importante: registrar cuando NO hay actividad"""
        self.log_event("NO_ACTIVITY", {
            "message": "Sistema operativo, sin eventos significativos"
        })

    def generate_summary_report(self, days: int = 7) -> Dict:
        """Genera reporte de los últimos N días"""
        if not os.path.exists(self.log_path):
            return {"error": "No log file found"}
```

```
        cutoff = datetime.now(timezone.utc) - timedelta(days=days)

        stats = {
            "total_cycles": 0,
            "total_events": 0,
            "total_candidates": 0,
            "total_alerts": 0,
            "event_types": {}
        }

        with open(self.log_path, "r") as f:
            for line in f:
                try:
                    entry = json.loads(line.strip())
                    entry_time = datetime.fromisoformat(entry["timestamp"])

                    if entry_time < cutoff:
                        continue

                    event_type = entry["event_type"]
                    stats["event_types"][event_type] = stats["event_types"].get(event_type, 0) + 1

                    if event_type == "CYCLE":
                        stats["total_cycles"] += 1
                        stats["total_events"] += entry["data"].get("events_found", 0)
                        stats["total_candidates"] += entry["data"].get("candidates_generated", 0)
                        stats["total_alerts"] += entry["data"].get("alerts_sent", 0)

                except:
                    continue

        return stats

# ═══════════════════════════════════════════════════════════════════════════
# 8) LOOP PRINCIPAL - Integración Completa
# ═══════════════════════════════════════════════════════════════════════════

class HunterOracle:
    """
    Sistema Hunter completo - Oracle Monolítico

    Flujo operativo:
    1. INGESTA → Observar el mundo
    2. MEMORIA → Ventanas deslizantes
    3. MÉTRICA → Calcular ΔH, LI, coherencia
    4. E-VETO → Filtrar con honestidad
    5. CANDIDATOS → Generar expedientes
    6. ORACLE → Decidir gobernanza
    7. REGISTRO → Trazar todo
    """

    def __init__(self, config: Dict):
        self.config = config

        # Inicializar componentes
        self.ingesta = IngestaPublica()
        self.memoria = MemoriaOperativa(window_minutes=config["memory_window_minutes"])
        self.metrica = MetricaFisica()
        self.eveto = EVetoEngine(config)
        self.generador = GeneradorCandidatos(self.ingesta)
        self.oracle = OracleGovernance(config)
        self.registro = RegistroTrazable()
```

```
        self.cycle_count = 0
        self.running = True

        print("""
╔═══════════════════════════════════════════════════════════════════════════╗
║                    SISTEMA HUNTER INICIADO                                ║
║                    Oracle Monolítico v1.0                                 ║
╚═══════════════════════════════════════════════════════════════════════════╝
""")
        print(f"Workspace: {WORKSPACE}")
        print(f"Intervalo: {config['poll_interval_seconds']}s")
        print(f"Lookback: {config['usgs_lookback_hours']}h")
        print(f"Magnitud mínima: M{config['event_magnitude_min']}")
        print()

    def simulate_sensor_data(self):
        """
        Genera datos sintéticos de sensores para demostración.
        En producción: reemplazar con datos reales de hardware/APIs.
        """
        # Presión: simulación de precursor (caída gradual)
        base = 1013.25  # hPa
        noise = np.random.normal(0, 0.5)
        trend = -0.1 * (self.cycle_count % 20)  # Caída cada 20 ciclos
        pressure = base + trend + noise

        # Aceleración: movimiento con algo de orden direccional
        if self.cycle_count % 10 < 5:
            # Fase "ordenada"
            accel_x = np.random.normal(0.5, 0.1)
            accel_y = np.random.normal(0.2, 0.1)
            accel_z = np.random.normal(-0.3, 0.1)
        else:
            # Fase "caótica"
            accel_x = np.random.normal(0, 0.5)
            accel_y = np.random.normal(0, 0.5)
            accel_z = np.random.normal(0, 0.5)

        self.memoria.add_pressure_sample(pressure)
        self.memoria.add_accel_sample(accel_x, accel_y, accel_z)

    def process_cycle(self):
        """Ejecuta un ciclo completo del sistema"""
        self.cycle_count += 1
        print(f"\n{'='*70}")
        print(f"CICLO #{self.cycle_count} - {datetime.now(timezone.utc).strftime('%Y-%m-%d %H:%M:%S UTC')}")
        print(f"{'='*70}")

        # 1) INGESTA: Obtener eventos
        print("\n[1/7] INGESTA - Observando el mundo...")
        events = self.ingesta.fetch_usgs_events(
            lookback_hours=self.config["usgs_lookback_hours"],
            min_mag=self.config["event_magnitude_min"]
        )
        print(f"  → {len(events)} eventos encontrados")

        if not events:
            self.registro.log_no_activity()
            self.registro.log_cycle(self.cycle_count, 0, 0, 0)
            return

        # 2) MEMORIA: Actualizar con datos de sensores
```

```
print("\n[2/7] MEMORIA - Actualizando ventanas...")
self.simulate_sensor_data()  # En producción: datos reales
self.memoria.save_state()
print(f"  → Ventana: {len(self.memoria.pressure_buffer)} muestras de presión")

# 3) MÉTRICA: Calcular cambios estructurales
print("\n[3/7] MÉTRICA - Calculando ΔH, LI, coherencia...")
metricas = self.metrica.analizar_evento(self.memoria)
print(f"  → ΔH = {metricas['dH']:.4f} {'✓' if metricas['dH'] <= self.config['dH_threshold'] else '✗'}")
print(f"  → LI = {metricas['LI']:.4f} {'✓' if metricas['LI'] >= self.config['LI_threshold'] else '✗'}")
print(f"  → R² = {metricas['R2']:.4f}")
print(f"  → Coherencia = {metricas['coherence']:.4f}")

candidates_generated = 0
alerts_sent = 0

for event in events:
    print(f"\n  Procesando: {event['event_id']} - {event['region']} M{event['magnitude']}")

    # 4) E-VETO: Aplicar doble sello
    print("\n[4/7] E-VETO - Filtro de honestidad...")
    seal = self.eveto.apply_double_seal(metricas)
    print(f"  → Doble Sello: {'✓ PASSED' if seal.passed else '✗ FAILED'} (Score: {seal.score}/100)")

    if not seal.passed:
        print(f"  → RECHAZADO: {', '.join(seal.reasons)}")
        self.registro.log_event("SEAL_REJECTED", {
            "event_id": event["event_id"],
            "reasons": seal.reasons
        })
        continue

    # Test de significancia
    pressure_series = self.memoria.get_pressure_series()
    if len(pressure_series) > 10:
        shuffle_pass = self.eveto.test_shuffle_significance(
            metricas["LI"],
            pressure_series
        )
        if not shuffle_pass:
            print(f"  → RECHAZADO: Falló test de shuffle (no significativo)")
            continue

    # 5) CANDIDATOS: Generar expediente
    print("\n[5/7] CANDIDATOS - Generando expediente...")
    candidate = self.generador.create_candidate(event, metricas, seal)

    if candidate:
        candidates_generated += 1
        print(f"  → Expediente creado: {candidate.event_id}")
        print(f"  → Persistencia: {candidate.persistence_count}/{self.config['persistence_cycles']} ciclos")

        # 6) ORACLE: Decisión de gobernanza
        print("\n[6/7] ORACLE - Evaluando gobernanza...")
        verdict = self.oracle.evaluate_candidate(candidate)
        print(f"  → Decisión: {verdict['decision']}")
        print(f"  → Confianza: {verdict['confidence']}%")

        if verdict['vetos']:
            print(f"  → Vetos activos: {', '.join(verdict['vetos'])}")

        if verdict['decision'] == "ESCALATE":
            print(f"  → 🚨 ALERTA ESCALADA - Notificando...")
```

```
                    self.oracle.send_alert_email(candidate, verdict)
                    alerts_sent += 1

                # 7) REGISTRO: Trazar decisión
                self.registro.log_event("CANDIDATE_EVALUATED", {
                    "event_id": candidate.event_id,
                    "verdict": verdict,
                    "candidate": candidate.to_dict()
                })

        # Resumen del ciclo
        self.registro.log_cycle(self.cycle_count, len(events), candidates_generated, alerts_sent)

        print(f"\n{'='*70}")
        print(f"RESUMEN CICLO #{self.cycle_count}")
        print(f"  Eventos procesados:     {len(events)}")
        print(f"  Candidatos generados:   {candidates_generated}")
        print(f"  Alertas enviadas:       {alerts_sent}")
        print(f"{'='*70}\n")

    def run_continuous(self):
        """Loop continuo del sistema"""
        print("Iniciando monitoreo continuo...")
        print("Presiona Ctrl+C para detener\n")

        try:
            while self.running:
                self.process_cycle()
                time.sleep(self.config["poll_interval_seconds"])

        except KeyboardInterrupt:
            print("\n\n⚠ Deteniendo sistema...")
            self.shutdown()

    def run_single_cycle(self):
        """Ejecuta un solo ciclo (útil para testing)"""
        self.process_cycle()

    def shutdown(self):
        """Cierre ordenado del sistema"""
        print("\n[SHUTDOWN] Guardando estado final...")
        self.memoria.save_state()

        # Generar reporte final
        summary = self.registro.generate_summary_report(days=7)
        print(f"\nRESUMEN ÚLTIMOS 7 DÍAS:")
        print(f"  Total ciclos:     {summary['total_cycles']}")
        print(f"  Total eventos:    {summary['total_events']}")
        print(f"  Total candidatos: {summary['total_candidates']}")
        print(f"  Total alertas:    {summary['total_alerts']}")

        print("\n✓ Sistema Hunter detenido correctamente")
        print(f"✓ Datos persistidos en: {WORKSPACE}\n")

# ═══════════════════════════════════════════════════════════════════════════
# MAIN - Punto de Entrada
# ═══════════════════════════════════════════════════════════════════════════

def main():
    """
    Punto de entrada principal.

    Uso en Colab:
```

```
    1. Ejecuta esta celda
    2. El sistema iniciará monitoreo continuo
    3. Revisa Drive para ver los resultados

    Para testing:
    - Usa run_single_cycle() para ejecutar solo un ciclo
    """

    # Crear sistema
    hunter = HunterOracle(CONFIG)

    # Opción 1: Modo continuo (producción)
    # hunter.run_continuous()

    # Opción 2: Modo single-shot (testing/demo)
    hunter.run_single_cycle()

    # Mostrar ubicación de archivos
    print(f"\n📁 UBICACIÓN DE ARCHIVOS:")
    print(f"   Drive: {WORKSPACE}")
    print(f"   Candidatos: {FOLDERS['candidatos']}")
    print(f"   Registro: {FOLDERS['registro']}")
    print(f"\n💡 Para modo continuo, descomenta: hunter.run_continuous()")

# ═══════════════════════════════════════════════════════════════════════════
# CONFIGURACIÓN DE CREDENCIALES (Ejecutar primero en Colab)
# ═══════════════════════════════════════════════════════════════════════════

def setup_email_credentials():
    """
    Función helper para configurar credenciales de email.
    Ejecutar ANTES de main() si quieres recibir alertas por email.
    """

    print("Configuración de Email para Alertas")
    print("="*50)

    email_from = input("Tu email (Gmail): ")
    password = input("Contraseña de aplicación (no tu password normal): ")
    email_to = input("Email destino (separados por coma si son varios): ")

    email_to_list = [e.strip() for e in email_to.split(",")]

    creds = {
        "smtp_server": "smtp.gmail.com",
        "smtp_port": 587,
        "email_from": email_from,
        "password": password,
        "email_to": email_to_list
    }

    cred_path = f"{FOLDERS['config']}/email_credentials.json"
    with open(cred_path, "w") as f:
        json.dump(creds, f, indent=2)

    print(f"\n✓ Credenciales guardadas en: {cred_path}")
    print("\n⚠ IMPORTANTE: Para Gmail, necesitas una 'App Password':")
    print("   1. Ve a https://myaccount.google.com/security")
    print("   2. Activa 'Verificación en 2 pasos'")
    print("   3. Ve a 'Contraseñas de aplicaciones'")
    print("   4. Genera una contraseña para 'Mail'")
    print("   5. Usa esa contraseña aquí (no tu password normal)\n")
```

```
# ═══════════════════════════════════════════════════════════════════════════
# EJECUTAR
# ═══════════════════════════════════════════════════════════════════════════

if __name__ == "__main__":
    # Paso 1: Configurar email (opcional, comentar si no quieres alertas)
    # setup_email_credentials()

    # Paso 2: Ejecutar sistema
    main()
```