```
# UAID: NBX-ALG-00010
# GoldenDAG: c3d4e<seg_34>7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3
#
# NeuralBlitz UEF/SIMI v11.1
# Core Algorithm: Bloom Event Detector
# Part of the MetaMind and Self-Reflection_Logs Subsystems
#
# Core Principle: Recursive Self-Betterment - understanding

import numpy as np
import json
from pathlib import Path
from typing import List, Dict, Optional
import datetime as dt

class BloomEventDetector:
    """
    Analyzes sequences of DRS vector shards to detect 'Bloo
    events. A Bloom is identified as a statistically signif
    effective dimensionality or variance of the latent spac
    """

    def __init__(self, shard_dir: str, sigma_threshold: floa
        """
        Initializes the detector.

        Args:
            shard_dir (str): The directory containing DRS v
            sigma_threshold (float): The number of standard
                                     entropy required to tr
            min_shard_history (int): The minimum number of 
                                     to establish a baselin
        """
        self.shard_dir = Path(shard_dir)
        if not self.shard_dir.is_dir():
            raise FileNotFoundError(f"ERR-FS-006: Shard dire

        self.sigma_threshold = sigma_threshold
        self.min_shard_history = min_shard_history

    def _calculate_shannon_entropy_from_variance(self, princ
        """
        Calculates the Shannon entropy of the variance distr
        means the variance is spread across more dimensions,

        Args:
            principal_components (np.ndarray): The singular

        Returns:
            float: The calculated Shannon entropy.
        """
        # Normalize the variance explained by each component
        variance_explained = principal_components**2 / np.su

        # Filter out zero probabilities to avoid log(0)
        variance_explained = variance_explained[variance_exp

        # Calculate Shannon entropy
        entropy = -np.sum(variance_explained * np.log2(varia
        return float(entropy)

    def analyze_shard(self, shard_path: Path) -> Optional[f
        """
        Analyzes a single vector shard file and calculates i

        Args:
            shard_path (Path): Path to the .npz shard file.
                               an array under the key 'vecto

        Returns:
            Optional[float]: The entropy of the shard, or No
        """
        try:
            with np.load(shard_path) as data:
                vectors = data['vectors']

            if vectors.ndim != 2 or vectors.shape[0] < 2:
                print(f"Warning: Skipping shard '{shard_path
                return None

            # Center the data before SVD
            centered_vectors = vectors - np.mean(vectors, ax

            # Use SVD to find principal components. We only
            # Truncating to min(shape) for performance on ve
            num_components = min(centered_vectors.shape)
            s = np.linalg.svd(centered_vectors, full_matrice

            return self._calculate_shannon_entropy_from_vari
        except Exception as e:
            print(f"Warning: Failed to process shard '{shard
            return None

    def run_detection(self) -> List[Dict]:
        """
        Runs the detection process over all shards in the di
        a list of detected bloom events.

        Returns:
            List[Dict]: A list of dictionaries, where each
```

```
                        a detected bloom event.
        """
        shard_files = sorted(self.shard_dir.glob("*.npz"))
        if len(shard_files) < self.min_shard_history:
            print(f"Info: Insufficient history ({len(shard_
                  f"Need at least {self.min_shard_history} 
            return []

        print(f"Analyzing {len(shard_files)} DRS vector sha

        entropies = []
        for shard_file in shard_files:
            entropy = self.analyze_shard(shard_file)
            if entropy is not None:
                entropies.append({"file": str(shard_file), 

        if not entropies:
            return []

        entropy_values = np.array([e['entropy'] for e in en

        # --- Statistical Anomaly Detection ---
        mean_entropy = np.mean(entropy_values)
        std_entropy = np.std(entropy_values)
        alert_threshold = mean_entropy + self.sigma_thresho

        alerts = []
        for entry in entropies:
            if entry['entropy'] > alert_threshold:
                alert = {
                    "event_type": "BLOOM_DETECTED",
                    "UAID": f"NBX-LOG-BLM-{dt.datetime.utcn
                    "shard_file": entry['file'],
                    "entropy": entry['entropy'],
                    "mean_entropy": mean_entropy,
                    "std_entropy": std_entropy,
                    "threshold": alert_threshold,
                    "sigma_level": (entry['entropy'] - mean_
                    "timestamp": dt.datetime.utcnow().isofor
                }
                alerts.append(alert)

        print(f"Detection complete. Found {len(alerts)} pote

        if alerts:
            log_path = self.shard_dir.parent / "Self-Reflect
            log_path.parent.mkdir(exist_ok=True)
            with log_path.open('a') as f:
                for alert in alerts:
                    f.write(json.dumps(alert) + '\n')
            print(f"Alerts logged to: {log_path}")

        return alerts


if __name__ == '__main__':
    # --- Example NBCL Invocation Simulation ---
    # NBCL Command: /invoke MetaMind --run_bloom_detection -

    print("--- Initiating NeuralBlitz Bloom Event Detector S

    # Create a dummy shard directory and populate it with sa
    sim_shard_dir = Path("./sim_shards")
    sim_shard_dir.mkdir(exist_ok=True)

    # Generate some baseline shards (low entropy)
    print("Generating baseline vector shards...")
    for i in range(10):
        # Data concentrated in the first few dimensions
        base_vectors = np.random.randn(1000, 512) * np.array
        np.savez_compressed(sim_shard_dir / f"shard_{i:02d}.

    # Generate a "Bloom" shard (high entropy)
    print("Generating a BLOOM vector shard...")
    bloom_vectors = np.random.randn(1000, 512) # Uniform var
    np.savez_compressed(sim_shard_dir / "shard_10_BLOOM.npz"

    try:
        # Initialize and run the detector
        detector = BloomEventDetector(str(sim_shard_dir), s
        detected_events = detector.run_detection()

        print("\n--- Detection Report ---")
        if detected_events:
            print(json.dumps(detected_events, indent=2))
        else:
            print("No significant bloom events detected.")

    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Clean up the dummy directory and files
        for f in sim_shard_dir.glob("*.npz"):
            f.unlink()
        sim_shard_dir.rmdir()
```