```
# UAID: NBX-ALG-00014
# GoldenDAG: c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b
#
# NeuralBlitz UEF/SIMI v11.1
# Core Algorithm: Bloom Event Timeline Renderer
# Part of the Self-Reflection_Logs Subsystem
#
# Core Principle: Radical Transparency (ε₂) - visualizing th

import json
from pathlib import Path
import datetime as dt
from typing import List, Dict
# Matplotlib is the designated python_user_visible tool for
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

class BloomTimelineRenderer:
    """
    Parses a bloom alert log file and generates a visual tir
    Bloom and Hyperbloom events, plotting entropy levels ove
    """

    def __init__(self, log_jsonl_path: str):
        """
        Initializes the renderer with the path to the log fi

        Args:
            log_jsonl_path (str): Path to the .jsonl file ge
                                  BloomEventDetector.
        """
        self.log_path = Path(log_jsonl_path)
```

```
        if not self.log_path.exists():
            raise FileNotFoundError(f"ERR-FS-009: Bloom log

        self.events = self._load_events()

    def _load_events(self) -> List[Dict]:
        """Loads and parses all valid event entries from the
        loaded_events = []
        with self.log_path.open('r') as f:
            for line in f:
                try:
                    data = json.loads(line)
                    # We are only interested in the actual a
                    if data.get("event_type") == "BLOOM_DETE
                        # Convert timestamp string to a date
                        data['timestamp_dt'] = dt.datetime.f
                        loaded_events.append(data)
                except (json.JSONDecodeError, KeyError, Valu
                    print(f"Warning: Skipping malformed log

        # Sort events chronologically
        return sorted(loaded_events, key=lambda x: x['timest

    def render_plot(self, output_png_path: str):
        """
        Generates and saves a matplotlib plot of the bloom e

        Args:
            output_png_path (str): The path to save the gene
        """
        if not self.events:
            print("Info: No bloom events found in log file.
            return
```

```
dates = [event['timestamp_dt'] for event in self.eve
entropy_levels = [event['entropy'] for event in sel

# Take the baseline from the first event's metadata
mean_entropy = self.events[0].get('mean_entropy')
alert_threshold = self.events[0].get('threshold')

# --- Plotting with Matplotlib ---
fig, ax = plt.subplots(figsize=(15, 7))

# Plot the entropy levels as a stem plot
markerline, stemlines, baseline = ax.stem(
    dates,
    entropy_levels,
    linefmt='grey',
    markerfmt='o',
    bottom=mean_entropy if mean_entropy is not None
)
plt.setp(stemlines, 'linewidth', 1, 'color', 'royalB
plt.setp(markerline, 'markersize', 6, 'color', 'roya
plt.setp(baseline, 'color', 'grey', 'linewidth', 1,


# Plot the baseline and threshold lines
if mean_entropy is not None:
    ax.axhline(y=mean_entropy, color='grey', linesty
if alert_threshold is not None:
    ax.axhline(y=alert_threshold, color='red', lines

# Formatting the plot for readability
ax.set_title('NeuralBlitz - DRS Latent Space Bloom E
ax.set_ylabel('Shannon Entropy (Effective Dimensiona
ax.set_xlabel('Event Timestamp (UTC)', fontsize=12)
ax.legend()
```

```
        ax.grid(True, which='both', linestyle=':', linewidth
        # Improve date formatting on the x-axis
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%
        plt.gcf().autofmt_xdate() # Auto-rotate date labels
        plt.tight_layout()
        # Save the plot to the specified file
        plt.savefig(output_png_path, dpi=150)
        print(f"Bloom timeline visualization saved to: {outp
if __name__ == '__main__':
    # --- Example NBCL Invocation Simulation ---
    # NBCL Command: /invoke visualizer --render_bloom_timel
    print("--- Initiating NeuralBlitz Bloom Timeline Renderer
    # Create a dummy log file for the simulation
    log_file = Path("bloom_alerts_sim.jsonl")
    # We will need some fake event data
    now = dt.datetime.utcnow()
    dummy_log_content = [
        {"event_type": "BLOOM_DETECTED", "timestamp": (now -
        {"event_type": "BLOOM_DETECTED", "timestamp": (now -
        # This one is a significant bloom
        {"event_type": "BLOOM_DETECTED", "timestamp": (now -
        {"event_type": "BLOOM_DETECTED", "timestamp": (now -
        {"event_type": "BLOOM_DETECTED", "timestamp": (now -
    ]
    with log_file.open('w') as f:
        # A header summary might also be in the real log, wh
```

```
    f.write(json.dumps({"summary": "This is not an event
    for event in dummy_log_content:
        f.write(json.dumps(event) + '\n')

print(f"Created dummy log file: {log_file}")

try:
    renderer = BloomTimelineRenderer(str(log_file))
    output_image = "bloom_timeline.png"
    renderer.render_plot(output_image)

    print(f"\nSuccessfully generated plot '{output_image
    print("The plot shows entropy levels over time, with
except Exception as e:
    print(f"\nAn error occurred during the simulation: {
finally:
    # Clean up the dummy file
    log_file.unlink(missing_ok=True)
    # In a real run, you'd keep the output image.
    Path("bloom_timeline.png").unlink(missing_ok=True)
```