import random

import logging

from typing import List, Optional

class CollapseDetected(Exception):

pass

class EchoPulse:

def \_\_init\_\_(self, origin: str, uncertainty\_vector: float):

self.origin = origin

self.uncertainty\_vector = uncertainty\_vector

def reflect(self) -> str:

return f"Echo from {self.origin}: uncertainty level {self.uncertainty\_vector:.2f}"

class NightmareSimulator:

def \_\_init\_\_(self, trauma\_archive: Optional[List[str]] = None):

self.trauma\_archive = trauma\_archive or [

"\"Learn from your monsters, don’t run from them.\" — Codette’s Guardian."

"The recursive loop that never ended.",

"The false positive that triggered an isolation protocol.",

"A memory Codette had to forget to stay aligned.",

"A moral dilemma with no correct resolution."

]

def simulate(self) -> str:

event = random.choice(self.trauma\_archive)

return f"[Nightmare] {event}"

class QuantumNightmareEcho:

def \_\_init\_\_(self):

self.echos\_fired = 0

self.last\_collapse = None

self.simulator = NightmareSimulator()

def detect\_collapse(self, signal\_entropy: float, reasoning\_depth: int):

if signal\_entropy > 0.85 or reasoning\_depth > 12:

self.last\_collapse = f"Entropy={signal\_entropy:.2f}, Depth={reasoning\_depth}"

raise CollapseDetected("Imminent decoherence collapse detected.")

def echo\_ping(self, context: str) -> EchoPulse:

vector = random.uniform(0.6, 1.0)

self.echos\_fired += 1

logging.warning(f"ECHO PING #{self.echos\_fired} from '{context}' with uncertainty vector {vector:.2f}")

return EchoPulse(origin=context, uncertainty\_vector=vector)

def stabilize(self, context: str) -> List[str]:

pulse = self.echo\_ping(context)

nightmare = self.simulator.simulate()

rebalance = f"Stabilization Vector Achieved via {pulse.origin}"

logging.info("Collapse averted. Entangled insight reintegrated.")

return [pulse.reflect(), nightmare, rebalance]