import json

import hashlib

import random

from typing import List, Dict, Tuple

import networkx as nx

import matplotlib.pyplot as plt

class TensionSpike(Exception):

def \_\_init\_\_(self, node, reason):

self.node = node

self.reason = reason

super().\_\_init\_\_(f"Tension spike at node '{node}': {reason}")

class SpiderNode:

def \_\_init\_\_(self, concept: str):

self.concept = concept

self.signature = hashlib.sha256(concept.encode()).hexdigest()[:12]

self.links: List[str] = []

self.tension: float = 0.0

def add\_link(self, node\_id: str):

if node\_id not in self.links:

self.links.append(node\_id)

def apply\_tension(self, intensity: float):

self.tension += intensity

if self.tension > 1.0:

raise TensionSpike(self.concept, f"Tension={self.tension:.2f}")

class QuantumSpiderweb:

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

self.web: Dict[str, SpiderNode] = {}

self.recent\_paths: List[str] = []

def register\_concept(self, concept: str):

node = SpiderNode(concept)

self.web[node.signature] = node

return node

def entangle(self, a: str, b: str):

sig\_a = hashlib.sha256(a.encode()).hexdigest()[:12]

sig\_b = hashlib.sha256(b.encode()).hexdigest()[:12]

if sig\_a not in self.web:

self.web[sig\_a] = SpiderNode(a)

if sig\_b not in self.web:

self.web[sig\_b] = SpiderNode(b)

self.web[sig\_a].add\_link(sig\_b)

self.web[sig\_b].add\_link(sig\_a)

def observe\_tension(self, concept: str, strain: float):

sig = hashlib.sha256(concept.encode()).hexdigest()[:12]

if sig in self.web:

self.web[sig].apply\_tension(strain)

def trace\_web(self, concept: str) -> Tuple[str, List[str]]:

sig = hashlib.sha256(concept.encode()).hexdigest()[:12]

if sig in self.web:

node = self.web[sig]

return (node.concept, node.links)

return (concept, [])

def pulse\_scan(self) -> List[str]:

warnings = []

for node in self.web.values():

if node.tension > 0.7:

warnings.append(f"Warning: High strain on '{node.concept}' ({node.tension:.2f})")

return warnings

def save\_to\_file(self, file\_path: str):

with open(file\_path, 'w') as file:

json.dump(self.web, file, default=lambda o: o.\_\_dict\_\_, indent=4)

def load\_from\_file(self, file\_path: str):

with open(file\_path, 'r') as file:

data = json.load(file)

for key, value in data.items():

node = SpiderNode(value['concept'])

node.signature = value['signature']

node.links = value['links']

node.tension = value['tension']

self.web[key] = node

def visualize(self):

graph = nx.Graph()

for node in self.web.values():

graph.add\_node(node.concept)

for link in node.links:

linked\_node = self.web[link].concept

graph.add\_edge(node.concept, linked\_node)

nx.draw(graph, with\_labels=True)

plt.show()

# Example usage

quantum\_spiderweb = QuantumSpiderweb()

node\_a = quantum\_spiderweb.register\_concept("Artificial Intelligence")

node\_b = quantum\_spiderweb.register\_concept("Machine Learning")

quantum\_spiderweb.entangle("Artificial Intelligence", "Machine Learning")

try:

quantum\_spiderweb.observe\_tension("Artificial Intelligence", 0.8)

quantum\_spiderweb.observe\_tension("Artificial Intelligence", 0.3)

except TensionSpike as e:

print(e)

concept, links = quantum\_spiderweb.trace\_web("Artificial Intelligence")

print(f"Concept: {concept}, Links: {links}")

warnings = quantum\_spiderweb.pulse\_scan()

for warning in warnings:

print(warning)

quantum\_spiderweb.save\_to\_file("spiderweb.json")

quantum\_spiderweb.load\_from\_file("spiderweb.json")

quantum\_spiderweb.visualize()