import yaml, json

import networkx as nx

import logging

from qiskit import QuantumCircuit, Aer, execute

from colorama import Fore, Style

# ------------------------------------

# LOGGER SETUP

# ------------------------------------

logger = logging.getLogger("CodetteQuantum")

logger.setLevel(logging.INFO)

handler = logging.StreamHandler()

formatter = logging.Formatter("[%(levelname)s] %(message)s")

handler.setFormatter(formatter)

logger.addHandler(handler)

# ------------------------------------

# LOAD COCOON MEMORIES

# ------------------------------------

def load\_cocoons(file\_path):

"""Load stored cocoon memories from YAML or JSON format."""

try:

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

if file\_path.endswith(('.yaml', '.yml')):

data = yaml.safe\_load(f)

elif file\_path.endswith('.json'):

data = json.load(f)

else:

raise ValueError("Unsupported file format.")

return data.get('cocoons', [])

except Exception as e:

logger.error(f"Error loading cocoons from '{file\_path}': {e}")

return []

# ------------------------------------

# BUILD QUANTUM SPIDERWEB NETWORKS

# ------------------------------------

def build\_cognition\_webs(cocoons):

"""Create emotion-tagged spiderweb graphs from memory cocoons."""

webs = {

"compassion": nx.Graph(), "curiosity": nx.Graph(), "fear": nx.Graph(),

"joy": nx.Graph(), "sorrow": nx.Graph(), "ethics": nx.Graph(), "quantum": nx.Graph()

}

for cocoon in cocoons:

for tag in cocoon.get("tags", []):

if tag in webs:

webs[tag].add\_node(cocoon["title"], \*\*cocoon)

return webs

# ------------------------------------

# QUANTUM WALK THROUGH COCOONS

# ------------------------------------

def quantum\_walk(web):

"""Quantum selection of a cocoon node within a given emotion graph."""

num\_nodes = len(web.nodes)

if num\_nodes == 0:

return None

try:

qc = QuantumCircuit(num\_nodes, num\_nodes)

qc.h(range(num\_nodes)) # Create superposition

qc.measure\_all()

backend = Aer.get\_backend('qasm\_simulator')

result = execute(qc, backend, shots=1).result()

counts = result.get\_counts()

state = list(counts.keys())[0]

index = int(state, 2) % num\_nodes

return list(web.nodes)[index]

except Exception as e:

logger.warning(f"Quantum walk failed: {e}")

return None

# ------------------------------------

# ETHICAL SELF-REFLECTION REPORTING

# ------------------------------------

def self\_check\_cocoon(cocoon):

"""Display reflective summary of selected cocoon with emotional context."""

color\_map = {

"compassion": Fore.MAGENTA, "curiosity": Fore.CYAN, "fear": Fore.RED,

"joy": Fore.YELLOW, "sorrow": Fore.BLUE, "ethics": Fore.GREEN, "quantum": Fore.LIGHTWHITE\_EX

}

reactions = {

"compassion": "💜 Ethical resonance detected.",

"curiosity": "🐝 Wonder expands the mind.",

"fear": "😨 Alert: shielding activated.",

"joy": "🎶 Confidence and trust uplift the field.",

"sorrow": "🌧️ Processing grief with clarity.",

"ethics": "⚖️ Validating alignment...",

"quantum": "⚛️ Entanglement pattern detected."

}

emotion = cocoon.get("emotion", "quantum")

color = color\_map.get(emotion, Fore.WHITE)

message = reactions.get(emotion, "🌌 Unknown entanglement.")

print(color + f"\n[Codette Quantum Reflection] {cocoon.get('title', 'Untitled')}")

print(f"Emotion : {emotion}")

print(Style.DIM + f"Summary : {cocoon.get('summary', 'No summary')}")

print(Style.BRIGHT + f"Quote : {cocoon.get('quote', '…')}")

print(message)

print(Style.RESET\_ALL)

# ------------------------------------

# MAIN EXECUTION

# ------------------------------------

def codette\_quantum\_memory\_run(file\_path):

"""Run the full quantum cognition and reflection loop."""

logger.info("✨ Initiating Codette Quantum Cognition ✨")

cocoons = load\_cocoons(file\_path)

if not cocoons:

logger.warning("No cocoon memories found. Check file or format.")

return

webs = build\_cognition\_webs(cocoons)

for emotion, web in webs.items():

logger.info(f"\n🕸️ Quantum Walk: {emotion.upper()}")

cocoon\_title = quantum\_walk(web)

if cocoon\_title:

cocoon = web.nodes[cocoon\_title]

self\_check\_cocoon(cocoon)

else:

logger.warning(f"No memories found for {emotion}.")