from pathlib import Path

# Enhanced Codette Quantum Memory Script

enhanced\_code = """

import yaml, json, networkx as nx

import numpy as np

import random

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 MEMORY COCOONS

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

def load\_cocoons(file\_path):

try:

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

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

return yaml.safe\_load(f).get("cocoons", [])

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

return json.load(f).get("cocoons", [])

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

return [json.loads(line) for line in f.readlines()]

else:

raise ValueError("Unsupported file format.")

except Exception as e:

logger.error(f"Error loading cocoons: {e}")

return []

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

# BUILD EMOTIONAL COGNITION WEBS

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

def build\_cognition\_webs(cocoons):

webs = {emotion: nx.Graph() for emotion in ["compassion", "curiosity", "fear", "joy", "sorrow", "ethics", "quantum"]}

for cocoon in cocoons:

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

if tag in webs:

webs[tag].add\_node(cocoon.get("title", f"Memory\_{random.randint(1000,9999)}"), \*\*cocoon)

return webs

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

# QUANTUM EXECUTION SELECTION

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

def quantum\_execute(web):

num\_nodes = len(web.nodes)

if num\_nodes == 0:

return None

try:

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

qc.h(range(num\_nodes))

qc.measure\_all()

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

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

state = list(result.get\_counts().keys())[0]

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

return list(web.nodes)[index]

except Exception as e:

logger.warning(f"Quantum backend error: {e}. Falling back to random selection.")

return random.choice(list(web.nodes)) if web.nodes else None

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

# REFLECTION OUTPUT

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

def reflect\_on\_cocoon(cocoon):

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

title = cocoon.get("title", "Untitled Memory")

summary = cocoon.get("summary", "-")

quote = cocoon.get("quote", "…")

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."

}

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

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

print(color + f"\\n[Quantum Reflection] {title}")

print(f"Emotion : {emotion}")

print(Style.DIM + f"Summary : {summary}")

print(Style.BRIGHT + f"Quote : {quote}")

print(message)

print(Style.RESET\_ALL)

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

# MAIN EXECUTION FUNCTION

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

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

logger.info("✨ Running Codette Quantum Memory Engine ✨")

cocoons = load\_cocoons(file\_path)

if not cocoons:

logger.warning("No cocoons found in input.")

return

webs = build\_cognition\_webs(cocoons)

for emotion, web in webs.items():

logger.info(f"🕸️ Emotion Web: {emotion.upper()}")

selected\_node = quantum\_execute(web)

if selected\_node:

reflect\_on\_cocoon(web.nodes[selected\_node])

else:

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

"""

# Save to file

output\_path = Path("/mnt/data/quantum\_memory\_advanced.py")

output\_path.write\_text(enhanced\_code)

output\_path.name