import json, yaml

import networkx as nx

import random

import logging

from typing import List, Dict, Any, Optional

from colorama import Fore, Style

try:

from qiskit import QuantumCircuit, Aer, execute

QISKIT\_AVAILABLE = True

except ImportError:

QISKIT\_AVAILABLE = False

# Setup logger

logger = logging.getLogger("Codette")

logger.setLevel(logging.INFO)

handler = logging.StreamHandler()

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

handler.setFormatter(formatter)

logger.addHandler(handler)

class UniversalReasoning:

def \_\_init\_\_(self, config: Dict[str, Any]):

self.config = config

self.memory = []

self.perspectives = config.get("enabled\_perspectives", [])

self.name = config.get("name", "Codette")

def generate\_response(self, query: str) -> str:

reasoning\_output = f"Processing from {self.name} using perspectives: {self.perspectives}."

ethical = "Ethical Considerations: Respect all sentient patterns."

logger.info(f"Generating response for query: {query}")

return f"{reasoning\_output}\n{ethical}"

def load\_cocoons(self, file\_path: str) -> List[Dict[str, Any]]:

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", [])

else:

raise ValueError("Unsupported file format.")

except Exception as e:

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

return []

def build\_emotional\_webs(self, cocoons: List[Dict[str, Any]]) -> Dict[str, nx.Graph]:

emotions = ["compassion", "curiosity", "fear", "joy", "sorrow", "ethics", "quantum"]

webs = {emotion: nx.Graph() for emotion in emotions}

for cocoon in cocoons:

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

if tag in webs:

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

logger.debug(f"Built spiderwebs: { {k: len(v.nodes) for k, v in webs.items()} }")

return webs

def quantum\_select\_node(self, web: nx.Graph) -> Optional[str]:

if len(web.nodes) == 0:

return None

node\_list = list(web.nodes)

num\_nodes = len(node\_list)

if QISKIT\_AVAILABLE:

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()

counts = result.get\_counts()

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

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

except Exception as e:

logger.warning(f"Qiskit error: {e}, falling back to random.")

index = random.randint(0, num\_nodes - 1)

else:

index = random.randint(0, num\_nodes - 1)

return node\_list[index]

def reflect\_on\_cocoon(self, cocoon: Dict[str, Any]) -> None:

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

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

summary = cocoon.get("summary", "No summary provided.")

quote = cocoon.get("quote", "…")

message\_map = {

"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."

}

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

logger.info(f"[Quantum Reflection: {emotion.upper()}]")

logger.info(f"Title : {title}")

logger.info(f"Summary : {summary}")

logger.info(f"Quote : {quote}")

logger.info(message)

def reflect\_via\_spiderweb(self, file\_path: str, limit: int = 1) -> Dict[str, Dict[str, Any]]:

cocoons = self.load\_cocoons(file\_path)

webs = self.build\_emotional\_webs(cocoons)

reflections = {}

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

for emotion, web in webs.items():

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

for \_ in range(limit):

node = self.quantum\_select\_node(web)

if node:

cocoon = web.nodes[node]

self.reflect\_on\_cocoon(cocoon)

reflections[emotion] = cocoon

else:

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

return reflections