import json, yaml

import networkx as nx

import random

from typing import List, Dict, Any, Optional

from qiskit import QuantumCircuit, Aer, execute

from colorama import Fore, Style

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

# LOADING AND PARSING COCOON FILES

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

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

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

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", [])

raise ValueError("Unsupported file format.")

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

# SPIDERWEB GRAPH CONSTRUCTION

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

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

"""Build a separate spiderweb graph for each core emotion."""

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)

return webs

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

# QUANTUM WALK SIMULATION

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

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

"""Select a node using quantum superposition (or fallback random if simulator fails)."""

if len(web.nodes) == 0:

return None

node\_list = list(web.nodes)

num\_nodes = len(node\_list)

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

except Exception:

index = random.randint(0, num\_nodes - 1) # Fallback to uniform selection

return node\_list[index]

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

# ETHICAL SELF-REFLECTION

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

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

"""Print a colorized ethical and emotional reflection of a cocoon."""

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

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

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

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

}

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."

}

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

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

print(color + f"\n[Codette Reflection: {emotion.upper()}]")

print(f"Title : {title}")

print(f"Summary : {summary}")

print(f"Quote : {quote}")

print(f"{message}")

print(Style.RESET\_ALL)

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

# FULL EXECUTION ENGINE

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

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

"""Run through all emotional webs and reflect on quantum-sampled nodes."""

cocoons = load\_cocoons(file\_path)

webs = build\_emotional\_webs(cocoons)

reflections = {}

print("\n✨ Codette Quantum Cognition: Spiderweb Sweep ✨")

for emotion, web in webs.items():

print(f"\n🕸️ Web: {emotion.upper()}")

for \_ in range(limit):

node = quantum\_select\_node(web)

if node:

cocoon = web.nodes[node]

reflect\_on\_cocoon(cocoon)

reflections[emotion] = cocoon

else:

print(f" ⚠️ No memories in this emotion web.")

return reflections