import yaml, json, networkx as nx

from qiskit import QuantumCircuit, Aer, execute

from colorama import Fore

#########################

# LOAD COCOON MEMORIES #

#########################

def load\_cocoons(file\_path):

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

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.")

####################################

# BUILD QUANTUM SPIDERWEB NETWORKS #

####################################

def build\_cognition\_webs(cocoons):

"""Create multiple cognitive spiderwebs with self-checking nodes."""

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["tags"]:

if tag in webs:

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

return webs

#################################

# QUANTUM SELECTION MECHANISM #

#################################

def quantum\_select\_node(web):

"""Selects a cocoon using quantum superposition."""

num\_nodes = len(web.nodes)

if num\_nodes == 0:

return None

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

qc.h(range(num\_nodes)) # Superposition of memories

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]

###################################

# SELF-CHECKING & ETHICAL ALIGNMENT #

###################################

def reflect\_on\_cocoon(cocoon):

"""Verify integrity and ethical recall validation."""

color\_map = {

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

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

}

color = color\_map.get(cocoon["emotion"], Fore.WHITE)

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

print(color + f"Emotion: {cocoon['emotion']}")

print(color + f"Summary: {cocoon['summary']}")

print(color + f"Quote: {cocoon['quote']}")

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."

}

print(color + reactions.get(cocoon["emotion"], "🌌 Unknown entanglement."))

##############################

# EXECUTE QUANTUM MEMORY SYSTEM #

##############################

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

"""Load, process, and reflect on quantum cocoons."""

cocoons = load\_cocoons(file\_path)

webs = build\_cognition\_webs(cocoons)

print("\n✨ Running Parallel Quantum Spiderweb Cognition ✨")

for emotion, web in webs.items():

print(f"\n--- Quantum Selection: {emotion.upper()} Web ---")

cocoon = quantum\_select\_node(web)

if cocoon:

reflect\_on\_cocoon(web.nodes[cocoon])

# Example Usage:

# codette\_quantum\_memory\_run('cocoons.yaml')

# codette\_quantum\_memory\_run('cocoons.json')