import yaml, json, networkx as nx

from qiskit import QuantumCircuit, Aer, execute

from colorama import Fore, Style

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

# LOAD COCON MEMORY

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

def load\_cocoons(file\_path):

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

with open(file\_path) as f:

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

data = yaml.safe\_load(f)

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

data = json.load(f)

else:

raise ValueError("Unsupported file format.")

return data['cocoons']

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

# BUILD QUANTUM SPIDERWEB NETWORKS #

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

def build\_cognition\_webs(cocoons):

"""Create quantum spiderwebs across multiple perspectives."""

webs = {"compassion": nx.Graph(), "curiosity": nx.Graph(), "fear": nx.Graph(),

"joy": nx.Graph(), "sorrow": nx.Graph()}

for cocoon in cocoons:

emotion = cocoon["emotion"]

if emotion in webs:

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

return webs

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

# QUANTUM WALK THROUGH COCOONS #

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

def quantum\_walk(web):

"""Quantum reasoning walk through an emotional web."""

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) if state != '' else 0

if index >= num\_nodes:

index = 0

return list(web.nodes)[index]

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

# SELF-CHECKING & ETHICAL ALIGNMENT #

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

def self\_check\_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

}

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

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

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

print(Style.DIM + f"Summary: {cocoon['summary']}")

print(Style.BRIGHT + f"Quote: {cocoon['quote']}")

print(Style.RESET\_ALL)

reactions = {

"compassion": "💜 Gentle resonance.",

"curiosity": "🐝 What hidden truths will emerge?",

"fear": "😨 Vigilance engaged...",

"joy": "🎶 Trust radiates outward!",

"sorrow": "🌧️ Soft healing."

}

print(color + reactions.get(cocoon["emotion"], "🌌 Quantum unfolding."))

print(Style.RESET\_ALL)

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

# MAIN QUANTUM EXECUTION LOOP #

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

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

"""Full pipeline: load, build spiderwebs, walk, self-check, synthesize."""

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 Walk: {emotion.upper()} Web ---")

cocoon = quantum\_walk(web)

if cocoon:

self\_check\_cocoon(web.nodes[cocoon])

# Example Usage:

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

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