import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import solve\_ivp

from scipy.fft import fft

from typing import Tuple, Dict, List, Optional

import logging

import json

import csv

import math

from collections import Counter

from datetime import datetime

from pathlib import Path

# Logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

# --- CONFIG ---

class Config:

def \_\_init\_\_(self):

self.HBAR = 1.0545718e-34

self.G = 6.67430e-11

self.BASE\_FREQ = 440.0

self.INTENT\_COEFF = 0.7

self.TUNNELING\_FACTOR = 0.4

self.ENTANGLEMENT\_STRENGTH = 0.85

self.DECOHERENCE\_FACTOR = 0.02

self.NUM\_AGENTS = 3

self.D = 2.0

self.MASSES = [1.0, 1.05, 0.95]

self.T\_SPAN = [0, 100]

self.NUM\_POINTS = 2500

self.RTOL = 1e-6

self.ATOL = 1e-8

# --- DREAMCORE + WAKESTATE SYSTEMS (EXTRACTED) ---

class DreamCore:

def \_\_init\_\_(self, path="dreamcore\_memory.yaml"):

self.path = Path(path)

self.memory = {}

def add\_anchor(self, anchor, tag, entropy\_level="medium"):

timestamp = datetime.utcnow().isoformat()

self.memory[timestamp] = {

"anchor": anchor,

"tag": tag,

"entropy": entropy\_level

}

def save(self):

with open(self.path, "w") as f:

for ts, data in self.memory.items():

f.write(f"- timestamp: {ts}\n")

for k, v in data.items():

f.write(f" {k}: {v}\n")

class WakeStateTracer:

def \_\_init\_\_(self, path="wakestate\_map.json"):

self.path = Path(path)

self.states = []

def add\_state(self, trigger, response, anchor, vector):

self.states.append({

"trigger": trigger,

"response": response,

"linked\_anchor": anchor,

"emotions": vector

})

def save(self):

with open(self.path, "w") as f:

json.dump({"states": self.states}, f, indent=2)

# --- AGENT STATE ---

class AgentState:

def \_\_init\_\_(self, num\_agents, d):

self.positions = np.array([[-d, 0], [0, 0], [d, 0]], dtype=np.float64)

self.velocities = np.array([[0, 0.5], [0, -0.5], [0, 0.3]], dtype=np.float64)

self.y0 = np.concatenate([np.ravel([p, v]) for p, v in zip(self.positions, self.velocities)])

self.observer\_log: List[Dict] = []

self.entropy\_trace: List[float] = []

self.observer\_state = 1.0

def log(self, t, positions, modifier, entropy):

self.observer\_log.append({

"t": t, "observer\_state": self.observer\_state,

"modifier": modifier, "entropy": entropy,

"positions": positions.tolist()

})

self.entropy\_trace.append(entropy)

# --- SIMULATION CORE ---

def compute\_accelerations(positions, masses, config):

n = len(masses)

acc = np.zeros\_like(positions)

for i in range(n):

for j in range(i+1, n):

r = positions[j] - positions[i]

dist = np.linalg.norm(r)

if dist > 1e-6:

force = (config.G \* masses[i] \* masses[j] / dist\*\*3) \* r

acc[i] += force / masses[i]

acc[j] -= force / masses[j]

return acc

def compute\_quantum(t, positions, state, config):

phase = np.sum(np.sin(config.BASE\_FREQ \* t / 1000)) \* config.INTENT\_COEFF

entropy = -state.observer\_state \* np.log(np.abs(phase) + 1e-10)

tunneling = config.TUNNELING\_FACTOR \* np.exp(-np.linalg.norm(positions) / config.HBAR) if np.random.rand() < config.TUNNELING\_FACTOR else 0

entangled = config.ENTANGLEMENT\_STRENGTH \* np.exp(-np.linalg.norm(positions) / config.HBAR)

decohere = config.DECOHERENCE\_FACTOR \* (1 - np.exp(-np.linalg.norm(positions) / config.HBAR))

force = np.full\_like(positions, phase + entangled + tunneling - decohere)

force += entropy \* 0.01

return force, entropy

def observer\_dynamics(t, y, state, config, dreamcore, wakestate):

pos = y[:2\*config.NUM\_AGENTS].reshape(config.NUM\_AGENTS, 2)

vel = y[2\*config.NUM\_AGENTS:].reshape(config.NUM\_AGENTS, 2)

acc = compute\_accelerations(pos, config.MASSES, config)

qforce, entropy = compute\_quantum(t, pos, state, config)

acc += qforce

# Logging and reactions

state.log(t, pos, qforce[0, 0], entropy)

if abs(entropy) > 0.8:

anchor = f"Entropy spike at t={t:.2f}, modifier={qforce[0, 0]:.3f}"

dreamcore.add\_anchor(anchor, "critical-decision", "high")

wakestate.add\_state("entropy spike", "internal realignment", anchor, {"urgency": 0.9, "clarity": 0.8})

return np.concatenate([vel.flatten(), acc.flatten()])

def simulate(config, state, dreamcore, wakestate):

t\_eval = np.linspace(config.T\_SPAN[0], config.T\_SPAN[1], config.NUM\_POINTS)

sol = solve\_ivp(lambda t, y: observer\_dynamics(t, y, state, config, dreamcore, wakestate),

config.T\_SPAN, state.y0, t\_eval=t\_eval,

rtol=config.RTOL, atol=config.ATOL)

return sol

def export\_logs(state: AgentState):

with open("observer\_log.json", "w") as f:

json.dump(state.observer\_log, f, indent=2)

with open("entropy\_trace.csv", "w", newline='') as csvfile:

writer = csv.writer(csvfile)

writer.writerow(["Time Index", "Entropy"])

for idx, ent in enumerate(state.entropy\_trace):

writer.writerow([idx, ent])

def detect\_fft\_anomalies(state: AgentState):

spectrum = np.abs(fft(state.entropy\_trace))

peak = np.max(spectrum)

if peak > 50:

logger.warning(f"FFT anomaly detected! Peak: {peak:.2f}")

return True

return False

def main():

config = Config()

state = AgentState(config.NUM\_AGENTS, config.D)

dreamcore = DreamCore()

wakestate = WakeStateTracer()

result = simulate(config, state, dreamcore, wakestate)

export\_logs(state)

dreamcore.save()

wakestate.save()

anomaly = detect\_fft\_anomalies(state)

if anomaly:

logger.info("Entropy anomaly detected via FFT.")

logger.info("Simulation complete. Logs and outputs saved.")

if \_\_name\_\_ == "\_\_main\_\_":

main()