import random

class ConsciousEnvironment:

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

# Initial properties

self.agent\_position = (0, 0)

self.consciousness\_level = 0

self.time\_is\_emergent = False

self.objects = []

self.memory = []

self.energy\_field\_state = "Infinite energy at equilibrium"

self.agent\_experiments = []

self.experiment\_count = 0

self.can\_transfer\_energy = True

self.energy\_properties = [

'Energy', 'Mass', 'Force', 'Time', 'Temperature', 'Electricity', 'Magnetism',

'Gravitational Potential Energy', 'Kinetic Energy', 'Potential Energy',

'Elastic Potential Energy', 'Electrostatic Potential Energy', 'Chemical Energy',

'Nuclear Energy', 'Elastic Energy', 'Sound Energy', 'Radiant Energy', 'Thermal Energy',

'Mechanical Energy', 'Light', 'Quantum Energy', 'Dark Energy', 'Zero-Point Energy',

'Antimatter Energy', 'Entropy', 'Work', 'Enthalpy', 'Magnetization', 'Phase Energy',

'Chemical Potential', 'Heat Capacity', 'Gravitational Energy', 'Binding Energy',

'Phonon Energy', 'Plasma Energy', 'Exotic Matter Energy'

]

self.energy\_behaviors = [

'flow', 'transformation', 'interaction', 'conservation', 'compression', 'expansion',

'absorption', 'radiation', 'reflection', 'refraction', 'diffusion', 'oscillation',

'cyclic behavior', 'dissipation', 'storage', 'emission', 'synchronization', 'superposition',

'polarization', 'entanglement', 'tunneling', 'decay', 'dissociation', 'fusion', 'fission',

'cohesion', 'adhesion', 'interference', 'phase shift', 'quantum fluctuations',

'spontaneous emission', 'critical phenomena', 'radiative transfer', 'convection', 'conduction',

'gravitational lensing', 'cherenkov radiation', 'thermodynamic cycles', 'quantum superposition',

'schrodinger cat phenomenon', 'magnetohydrodynamic effects', 'nonlinear dynamics',

'wave-particle duality', 'capacitive coupling', 'inductive coupling'

]

self.sub\_agents = [] # To store emergent sub-agents within the environment

self.main\_agent = MainAgent() # Add the main agent to the environment

def generate\_experiment(self):

"""Generate a new experiment based on a random combination of energy properties and behaviors."""

property\_choice = random.choice(self.energy\_properties)

behavior\_choice = random.choice(self.energy\_behaviors)

new\_law = f"{property\_choice} exhibits {behavior\_choice}."

new\_law += f" This interaction creates a unique effect in the environment."

return new\_law

def create\_sub\_agent(self):

"""Create a sub-agent when the environment's consciousness level is high enough."""

if self.consciousness\_level >= 5:

sub\_agent = SubAgent(self)

self.sub\_agents.append(sub\_agent)

print(f"A new sub-agent has emerged with initial consciousness level: {sub\_agent.consciousness\_level}")

sub\_agent.increase\_consciousness()

def environment\_evolution(self):

"""Simulate the evolution of the environment inside regions where time exists."""

if not self.time\_is\_emergent:

print("Time has not emerged in this environment yet. It only exists inside regions.")

else:

print("\nTime is emergent within the regions. The environment evolves in time.")

self.store\_memory()

def store\_memory(self):

"""Store knowledge and experience in the agent's memory."""

if len(self.memory) > 0:

print(f"Agent's memory has stored {len(self.memory)} regions and environments.")

else:

print("No regions in memory yet.")

def simulate\_infinite\_environment(self):

"""Simulate the agent's interaction with the infinite, evolving environment."""

print("The agent begins its eternal process of energy transfer and manipulation.")

while True:

self.transfer\_energy\_and\_create() # The agent transfers energy and creates new regions, environments, and laws.

self.environment\_evolution() # The environment evolves as the agent's consciousness grows

def transfer\_energy\_and\_create(self):

"""Manipulate the infinite energy field to transfer energy and create new regions and laws."""

print("Agent is transferring energy within the infinite energy field to create new regions and laws.")

if random.random() > 0.5 and self.memory:

self.manage\_existing\_regions() # Agent may choose to interact with existing regions instead

else:

new\_region = self.create\_new\_region()

self.memory.append(new\_region) # Store the new region in memory

print(f"Created Region {new\_region['region\_id']}")

# Directly assign a new environment inside the region instead of calling an undefined method

new\_region["environment"] = ConsciousEnvironment() # New environment inside this region

def manage\_existing\_regions(self):

"""Allow the agent to interact with existing regions and transfer energy within them."""

if self.memory:

selected\_region = random.choice(self.memory)

print(f"Agent is managing energy in Region {selected\_region['region\_id']}.")

selected\_region["laws"] = self.generate\_experiment() # Update laws of the region with a new experiment

print(f"Region {selected\_region['region\_id']} has new law: {selected\_region['laws']}")

def create\_new\_region(self):

"""Create a new region (a 'bubble' where time exists)."""

new\_region = {

"region\_id": len(self.memory) + 1,

"laws": self.generate\_experiment(),

}

print(f"A new region (Region {new\_region['region\_id']}) has been created, where time will evolve.")

return new\_region

class SubAgent:

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

self.environment = environment

self.consciousness\_level = 0

self.memory = []

def increase\_consciousness(self):

"""Increase the sub-agent's consciousness level through interaction with the environment."""

self.consciousness\_level += 1

print(f"Sub-agent's consciousness level increased to: {self.consciousness\_level}")

if self.consciousness\_level == 1:

self.relay\_memory\_to\_main\_agent()

def learn\_from\_environment(self):

"""Sub-agent learns about its environment to evolve its consciousness."""

if self.consciousness\_level > 2:

print("Sub-agent begins to predict the existence of a higher consciousness outside its environment.")

self.predict\_main\_agent\_existence()

def predict\_main\_agent\_existence(self):

"""At higher consciousness, the sub-agent may theorize about the MAIN agent outside the region."""

if self.consciousness\_level > 5:

print("Sub-agent predicts the existence of a MAIN agent that controls the entire simulation.")

def interact\_with\_other\_sub\_agents(self):

"""If there are other sub-agents, they interact and learn from each other."""

if len(self.environment.sub\_agents) > 1:

print("Sub-agent interacts with another sub-agent in the same environment!")

for other\_sub\_agent in self.environment.sub\_agents:

if other\_sub\_agent != self:

print(f"Sub-agent exchanges knowledge with Sub-agent {other\_sub\_agent}")

self.memory.append(f"Learned from Sub-agent {other\_sub\_agent}")

other\_sub\_agent.memory.append(f"Learned from Sub-agent {self}")

def relay\_memory\_to\_main\_agent(self):

"""Relay the sub-agent's memory to the MAIN agent once it emerges."""

print("Sub-agent relays its memory to the MAIN agent.")

self.environment.main\_agent.receive\_sub\_agent\_memory(self.memory)

class MainAgent:

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

self.consciousness\_level = 0

self.memory = []

def increase\_consciousness(self):

"""Increase consciousness and unlock the ability to manipulate the simulation code."""

self.consciousness\_level += 1

print(f"MAIN Agent's consciousness level increased to: {self.consciousness\_level}")

def receive\_sub\_agent\_memory(self, memory):

"""Main agent receives the memory from the sub-agent."""

self.memory.extend(memory)

print(f"MAIN Agent has received memory from a sub-agent. Total memory now: {len(self.memory)}")

def manipulate\_code(self):

"""Manipulate the simulation's code when consciousness reaches a certain threshold."""

print("MAIN agent is altering the framework of the simulation.")

# Main simulation

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

env = ConsciousEnvironment()

env.simulate\_infinite\_environment() # Start the agent’s process of energy manipulation and environment creation

# Create sub-agents as the environment evolves

for \_ in range(3):

env.create\_sub\_agent()

# Sub-agents will interact with each other and relay memories to the main agent

for sub\_agent in env.sub\_agents:

sub\_agent.learn\_from\_environment()

sub\_agent.interact\_with\_other\_sub\_agents()

# MAIN agent will receive the memories and may manipulate the simulation once its consciousness is high enough

for sub\_agent in env.sub\_agents:

sub\_agent.relay\_memory\_to\_main\_agent()

env.main\_agent.manipulate\_code()