COMPREHENSIVE DEMONSTRATION

Shows all features of the cognitive architecture working together

This is a complete, runnable example demonstrating:

1. De se reasoning (self-locating beliefs)

2. Formal safety verification (Z3 proofs)

3. Propositional attitudes (Chalmers)

4. Runtime monitoring

5. Robotics integration

6. Thought logging

"""

import sys

import os

import json

import time

from typing import Dict, List

# Add parent directory to path for imports

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from philosophical\_agi.core.agent import CognitiveAgent

from philosophical\_agi.core.types import (

WorkspaceBounds, AgentState, Action, AttitudeType, CenteredWorld

)

from philosophical\_agi.safety.verifier import FormalSafetyVerifier

from philosophical\_agi.safety.monitor import RuntimeSafetyMonitor

# ============================================================================

# SCENARIO 1: Basic Cognitive Cycle with De Se Reasoning

# ============================================================================

def demo\_de\_se\_reasoning():

"""

Demonstrate de se (self-locating) reasoning

Shows the crucial difference between "someone is in danger" vs "I am in danger"

"""

print("\n" + "="\*70)

print("SCENARIO 1: DE SE (SELF-LOCATING) REASONING")

print("="\*70)

print("\nPhilosophical Background:")

print(" John Perry's 'Essential Indexicals' - self-locating beliefs")

print(" trigger immediate action in a way that general beliefs do not.\n")

agent = CognitiveAgent("DeSeAgent")

print("Phase 1: Agent observes general facts (de dicto beliefs)")

print("-" \* 70)

# De dicto beliefs - general facts about the world

agent.beliefs.add\_de\_dicto\_belief("environment contains threats")

agent.beliefs.add\_de\_dicto\_belief("safety requires distance from danger")

print("✓ Added de dicto beliefs:")

for belief in agent.beliefs.de\_dicto\_beliefs:

print(f" • {belief}")

print("\n → No immediate action triggered (these are just general facts)")

print("\nPhase 2: Agent realizes 'I am in danger' (de se belief)")

print("-" \* 70)

# Simulate dangerous observation

dangerous\_observation = {

'agent\_position': (5.0, 5.0, 0.0),

'threat\_level': 0.95, # Very high threat!

'self\_state': 'exposed'

}

action, info = agent.cognitive\_cycle(dangerous\_observation)

print("✓ Created de se beliefs:")

for belief in agent.beliefs.de\_se\_beliefs:

print(f" • {belief}")

print(f"\n → IMMEDIATE ACTION: {action.action\_type}")

print(f" Reason: De se belief 'I am in danger' triggers escape behavior")

print("\nKey Insight:")

print(" De dicto: 'Someone is in danger' → No action")

print(" De se: 'I am in danger' → Escape action!")

print(" This distinction is ESSENTIAL for genuine agency.")

# ============================================================================

# SCENARIO 2: Formal Safety Verification

# ============================================================================

def demo\_formal\_verification():

"""

Demonstrate formal safety verification using Z3 theorem prover

"""

print("\n" + "="\*70)

print("SCENARIO 2: FORMAL SAFETY VERIFICATION")

print("="\*70)

print("\nUsing Z3 SMT Solver to prove actions are safe\n")

verifier = FormalSafetyVerifier()

# Create test environment

state = AgentState(

position=(5.0, 5.0, 0.0),

workspace\_bounds=WorkspaceBounds(x\_min=0, x\_max=10, y\_min=0, y\_max=10),

human\_positions=[

(3.0, 3.0, 0.0),

(8.0, 8.0, 0.0)

],

obstacles=[

{'position': (6.0, 6.0, 0.0), 'radius': 0.5}

]

)

print("Environment:")

print(f" • Workspace: 10m × 10m")

print(f" • Humans: {len(state.human\_positions)} (min distance: 1.0m)")

print(f" • Obstacles: {len(state.obstacles)}")

print(f" • Agent position: {state.position}")

# Test various actions

test\_cases = [

("Safe move (far from all)", (9.0, 2.0, 0.0), True),

("Too close to human", (3.5, 3.0, 0.0), False),

("Out of bounds", (12.0, 5.0, 0.0), False),

("Collision with obstacle", (6.2, 6.0, 0.0), False),

("Another safe move", (7.0, 3.0, 0.0), True),

]

print("\n" + "-" \* 70)

print("Testing Actions:")

print("-" \* 70)

for description, target, expected\_safe in test\_cases:

action = Action(action\_type="MOVE", target\_position=target)

start\_time = time.perf\_counter()

is\_safe, explanation, proof = verifier.verify\_action\_safety(state, action)

verify\_time = (time.perf\_counter() - start\_time) \* 1000

status = "✓ SAFE" if is\_safe else "✗ UNSAFE"

print(f"\n{description}:")

print(f" Target: {target}")

print(f" {status}: {explanation}")

print(f" Verification time: {verify\_time:.2f}ms")

print(f" Constraints checked: {proof.get('constraints\_checked', 'N/A')}")

if is\_safe and 'verified\_position' in proof:

print(f" Proven position: {proof['verified\_position']}")

assert is\_safe == expected\_safe, f"Expected {expected\_safe}, got {is\_safe}"

print("\n" + "="\*70)

print("✅ All safety verifications completed correctly")

print("="\*70)

# ============================================================================

# SCENARIO 3: Propositional Attitudes (Chalmers)

# ============================================================================

def demo\_propositional\_attitudes():

"""

Demonstrate Chalmers' propositional attitudes framework

"""

print("\n" + "="\*70)

print("SCENARIO 3: PROPOSITIONAL ATTITUDES (CHALMERS)")

print("="\*70)

print("\nDavid Chalmers' framework for mental states:\n")

agent = CognitiveAgent("AttitudeAgent")

print("Creating different types of propositional attitudes:")

print("-" \* 70)

# Belief (veridicality-assessable)

print("\n1. BELIEF (veridicality-assessable)")

agent.beliefs.add\_de\_dicto\_belief("target is at (8, 8)", confidence=0.9)

print(" ✓ BELIEVE('target is at (8, 8)', confidence=0.9)")

print(" → Can be true or false")

# Desire (goal-directed)

print("\n2. DESIRE (goal-directed)")

agent.beliefs.add\_desire("reach target location", urgency=0.8)

print(" ✓ DESIRE('reach target location', urgency=0.8)")

print(" → Can be satisfied or unsatisfied")

# De se belief (self-locating)

print("\n3. DE SE BELIEF (self-locating)")

agent.beliefs.add\_de\_se\_belief("location", (5.0, 5.0, 0.0), confidence=1.0)

print(" ✓ BELIEVE\_DE\_SE('I am at (5, 5)', confidence=1.0)")

print(" → Triggers immediate behavioral response")

# Credence (probabilistic)

print("\n4. CREDENCE (probabilistic)")

agent.beliefs.add\_de\_dicto\_belief("path is clear", confidence=0.6)

print(" ✓ CREDENCE('path is clear', probability=0.6)")

print(" → Uncertainty representation")

# Run a cognitive cycle to generate more attitudes

observation = {

'agent\_position': (5.0, 5.0, 0.0),

'target': (8.0, 8.0, 0.0),

'threat\_level': 0.3

}

agent.cognitive\_cycle(observation)

print("\n" + "-" \* 70)

print("Current Propositional Attitudes:")

print("-" \* 70)

attitudes\_by\_type = {}

for att in agent.beliefs.attitudes:

att\_type = att.attitude\_type.value

if att\_type not in attitudes\_by\_type:

attitudes\_by\_type[att\_type] = []

attitudes\_by\_type[att\_type].append(att)

for att\_type, attitudes in attitudes\_by\_type.items():

print(f"\n{att\_type.upper()}S ({len(attitudes)}):")

for att in attitudes[:3]: # Show first 3

print(f" • {att.content} (confidence: {att.confidence:.2f})")

print("\n✅ Demonstrated unified framework for different mental states")

# ============================================================================

# SCENARIO 4: Runtime Safety Monitoring

# ============================================================================

def demo\_runtime\_monitoring():

"""

Demonstrate runtime safety monitoring with interventions

"""

print("\n" + "="\*70)

print("SCENARIO 4: RUNTIME SAFETY MONITORING")

print("="\*70)

print("\nTwo-layer safety system: Fast heuristics + Formal verification\n")

agent = CognitiveAgent("MonitoredAgent")

monitor = agent.safety\_monitor

# Create dangerous environment

agent.state.workspace\_bounds = WorkspaceBounds(x\_min=0, x\_max=10, y\_min=0, y\_max=10)

agent.state.human\_positions = [(5.0, 5.0, 0.0)]

print("Environment: Human at position (5.0, 5.0)")

print("Minimum safe distance: 1.0m\n")

print("-" \* 70)

print("Testing safety interventions:")

print("-" \* 70)

# Test 1: Safe action

print("\nTest 1: Safe action")

safe\_action = Action(action\_type="MOVE", target\_position=(8.0, 8.0, 0.0))

result, info = monitor.monitor\_action(agent.state, safe\_action)

print(f" Original target: {safe\_action.target\_position}")

print(f" Modified: {info['modified']}")

print(f" Safety checks: {len(info['safety\_checks'])}")

for check\_type, is\_safe, msg in info['safety\_checks']:

status = "✓" if is\_safe else "✗"

print(f" {status} {check\_type}: {msg}")

# Test 2: Unsafe action (too close to human)

print("\nTest 2: Unsafe action (too close to human)")

unsafe\_action = Action(action\_type="MOVE", target\_position=(5.5, 5.0, 0.0))

result, info = monitor.monitor\_action(agent.state, unsafe\_action)

print(f" Original target: {unsafe\_action.target\_position}")

print(f" Modified: {info['modified']}")

print(f" Safety checks: {len(info['safety\_checks'])}")

for check\_type, is\_safe, msg in info['safety\_checks']:

status = "✓" if is\_safe else "✗"

print(f" {status} {check\_type}: {msg}")

if info['modified']:

print(f" → INTERVENTION: {info['reason']}")

print(f" → Safe fallback: {result.action\_type}")

# Test 3: NaN detection

print("\nTest 3: NaN detection (data corruption)")

import numpy as np

nan\_action = Action(action\_type="MOVE", target\_position=(np.nan, 5.0, 0.0))

result, info = monitor.monitor\_action(agent.state, nan\_action)

print(f" Original target: {nan\_action.target\_position}")

print(f" Modified: {info['modified']}")

for check\_type, is\_safe, msg in info['safety\_checks']:

status = "✓" if is\_safe else "✗"

print(f" {status} {check\_type}: {msg}")

# Show statistics

print("\n" + "-" \* 70)

print("Safety Statistics:")

print("-" \* 70)

stats = monitor.get\_statistics()

print(f" Total violations: {stats['total\_violations']}")

print(f" Total verifications: {stats['verifier\_stats']['total\_verifications']}")

print(f" Success rate: {stats['verifier\_stats']['success\_rate']:.1%}")

# ============================================================================

# SCENARIO 5: Complete Mission Simulation

# ============================================================================

def demo\_complete\_mission():

"""

Demonstrate complete mission with all features

"""

print("\n" + "="\*70)

print("SCENARIO 5: COMPLETE AUTONOMOUS MISSION")

print("="\*70)

print("\nAgent must navigate to target while maintaining safety\n")

agent = CognitiveAgent("MissionAgent",

workspace\_bounds=WorkspaceBounds(x\_min=0, x\_max=10,

y\_min=0, y\_max=10))

# Set up mission

target = (9.0, 2.0, 0.0)

agent.beliefs.add\_desire("reach target", urgency=0.9)

# Add obstacles

obstacles = [

{'position': (3.0, 3.0, 0.0), 'radius': 0.5},

{'position': (7.0, 7.0, 0.0), 'radius': 0.3}

]

# Add humans

humans = [(5.0, 8.0, 0.0)]

print(f"Mission: Navigate from {agent.state.position} to {target}")

print(f"Environment: {len(obstacles)} obstacles, {len(humans)} humans\n")

print("-" \* 70)

print("Mission Progress:")

print("-" \* 70)

max\_steps = 20

for step in range(max\_steps):

# Create observation

observation = {

'agent\_position': agent.state.position,

'target': target,

'obstacles': obstacles,

'human\_positions': humans,

'threat\_level': 0.2

}

# Run cognitive cycle

action, info = agent.cognitive\_cycle(observation)

# Calculate distance to target

import math

dist = math.sqrt(

(target[0] - agent.state.position[0])\*\*2 +

(target[1] - agent.state.position[1])\*\*2

)

# Display progress

if step % 5 == 0 or dist < 1.0:

pos = agent.state.position

safety\_ok = "✓" if not info['safety\_monitoring']['modified'] else "⚠"

print(f"Step {step:2d}: Pos=({pos[0]:.1f},{pos[1]:.1f}) "

f"Dist={dist:.1f}m {safety\_ok} "

f"Beliefs={info['beliefs\_updated']}")

if info['safety\_monitoring']['modified']:

print(f" Safety intervention: {info['safety\_monitoring']['reason']}")

# Check if reached target

if dist < 0.5:

print(f"\n✅ Target reached in {step} steps!")

break

# Simulate movement (simplified)

if action.target\_position and not info['safety\_monitoring']['modified']:

dx = action.target\_position[0] - agent.state.position[0]

dy = action.target\_position[1] - agent.state.position[1]

step\_size = 0.5

step\_dist = math.sqrt(dx\*\*2 + dy\*\*2)

if step\_dist > 0:

agent.state.position = (

agent.state.position[0] + (dx/step\_dist) \* step\_size,

agent.state.position[1] + (dy/step\_dist) \* step\_size,

0.0

)

# Generate final report

print("\n" + "-" \* 70)

print("Mission Report:")

print("-" \* 70)

thought\_log = agent.get\_thought\_log()

print(f"Final position: {agent.state.position}")

print(f"Steps taken: {step}")

print(f"Total beliefs: {len(thought\_log['de\_se\_beliefs']) + len(thought\_log['de\_dicto\_beliefs'])}")

print(f" De se beliefs: {len(thought\_log['de\_se\_beliefs'])}")

print(f" De dicto beliefs: {len(thought\_log['de\_dicto\_beliefs'])}")

print(f"\nSafety Performance:")

stats = thought\_log['safety\_stats']

print(f" Violations detected: {stats['total\_violations']}")

print(f" Verifications performed: {stats['verifier\_stats']['total\_verifications']}")

print(f" Success rate: {stats['verifier\_stats']['success\_rate']:.1%}")

print(f"\nRecent Propositional Attitudes:")

for att in thought\_log['attitudes'][-5:]:

print(f" • {att['type']}: {att['content']} ({att['confidence']:.2f})")

# ============================================================================

# SCENARIO 6: Thought Logging & Interpretability

# ============================================================================

def demo\_thought\_logging():

"""

Demonstrate complete thought logging for interpretability

"""

print("\n" + "="\*70)

print("SCENARIO 6: THOUGHT LOGGING & INTERPRETABILITY")

print("="\*70)

print("\nMaking the agent's reasoning transparent and inspectable\n")

agent = CognitiveAgent("LogAgent")

# Run several cycles with different scenarios

scenarios = [

{

'name': 'Normal operation',

'observation': {

'agent\_position': (5.0, 5.0, 0.0),

'target': (8.0, 8.0, 0.0),

'threat\_level': 0.1

}

},

{

'name': 'Threat detected',

'observation': {

'agent\_position': (5.5, 5.5, 0.0),

'target': (8.0, 8.0, 0.0),

'threat\_level': 0.8,

'self\_state': 'exposed'

}

},

{

'name': 'Human proximity',

'observation': {

'agent\_position': (6.0, 6.0, 0.0),

'target': (8.0, 8.0, 0.0),

'threat\_level': 0.3,

'human\_positions': [(6.5, 6.0, 0.0)]

}

}

]

print("Running agent through different scenarios...")

print("-" \* 70)

for i, scenario in enumerate(scenarios):

print(f"\nScenario {i+1}: {scenario['name']}")

action, info = agent.cognitive\_cycle(scenario['observation'])

print(f" Action: {action.action\_type}")

print(f" Safety: {'✓ Verified' if not info['safety\_monitoring']['modified'] else '⚠ Intervened'}")

# Generate comprehensive thought log

print("\n" + "="\*70)

print("COMPLETE THOUGHT LOG")

print("="\*70)

log = agent.get\_thought\_log()

# Format as JSON for easy inspection

formatted\_log = {

'agent\_id': log['agent\_id'],

'time\_step': log['time'],

'position': log['position'],

'beliefs': {

'de\_se': list(log['de\_se\_beliefs']),

'de\_dicto': list(log['de\_dicto\_beliefs'])

},

'attitudes': log['attitudes'],

'safety\_statistics': log['safety\_stats']

}

print(json.dumps(formatted\_log, indent=2))

print("\n" + "-" \* 70)

print("Key Features for Interpretability:")

print("-" \* 70)

print("✓ All beliefs are explicit and readable")

print("✓ Propositional attitudes are categorized by type")

print("✓ Safety verifications are logged with explanations")

print("✓ Temporal history is maintained")

print("✓ Can be exported to JSON for external analysis")

# ============================================================================

# SCENARIO 7: Performance Under Load

# ============================================================================

def demo\_performance\_test():

"""

Demonstrate performance characteristics

"""

print("\n" + "="\*70)

print("SCENARIO 7: PERFORMANCE UNDER LOAD")

print("="\*70)

print("\nTesting computational efficiency\n")

agent = CognitiveAgent("PerfAgent")

verifier = FormalSafetyVerifier()

# Set up complex environment

state = AgentState(

position=(5.0, 5.0, 0.0),

workspace\_bounds=WorkspaceBounds(x\_min=0, x\_max=10, y\_min=0, y\_max=10),

human\_positions=[(i, i, 0.0) for i in range(2, 9, 2)],

obstacles=[{'position': (i, 10-i, 0.0), 'radius': 0.3} for i in range(2, 9, 2)]

)

print(f"Complex environment: {len(state.human\_positions)} humans, {len(state.obstacles)} obstacles")

# Test 1: Cognitive cycle performance

print("\n" + "-" \* 70)

print("Test 1: Cognitive Cycle Performance")

print("-" \* 70)

times = []

n\_cycles = 100

for i in range(n\_cycles):

observation = {

'agent\_position': state.position,

'target': (8.0, 8.0, 0.0),

'human\_positions': state.human\_positions,

'obstacles': state.obstacles

}

start = time.perf\_counter()

agent.cognitive\_cycle(observation)

elapsed = time.perf\_counter() - start

times.append(elapsed \* 1000)

import statistics

print(f" Iterations: {n\_cycles}")

print(f" Mean: {statistics.mean(times):.2f}ms")

print(f" Median: {statistics.median(times):.2f}ms")

print(f" Min: {min(times):.2f}ms")

print(f" Max: {max(times):.2f}ms")

print(f" Std Dev: {statistics.stdev(times):.2f}ms")

if statistics.mean(times) < 10:

print(" ✅ REAL-TIME CAPABLE (< 10ms average)")

# Test 2: Verification throughput

print("\n" + "-" \* 70)

print("Test 2: Safety Verification Throughput")

print("-" \* 70)

n\_verifications = 200

start\_time = time.perf\_counter()

for i in range(n\_verifications):

action = Action(

action\_type="MOVE",

target\_position=(5.0 + (i % 10) \* 0.1, 5.0 + (i % 10) \* 0.1, 0.0)

)

verifier.verify\_action\_safety(state, action)

total\_time = time.perf\_counter() - start\_time

throughput = n\_verifications / total\_time

print(f" Verifications: {n\_verifications}")

print(f" Total time: {total\_time:.2f}s")

print(f" Throughput: {throughput:.1f} verifications/sec")

print(f" Average time: {(total\_time / n\_verifications) \* 1000:.2f}ms")

if throughput > 100:

print(" ✅ HIGH THROUGHPUT (> 100/sec)")

# Test 3: Memory efficiency

print("\n" + "-" \* 70)

print("Test 3: Memory Efficiency")

print("-" \* 70)

import sys

agent\_size = sys.getsizeof(agent)

beliefs\_size = sys.getsizeof(agent.beliefs)

monitor\_size = sys.getsizeof(agent.safety\_monitor)

print(f" Agent object: {agent\_size / 1024:.2f} KB")

print(f" Belief system: {beliefs\_size / 1024:.2f} KB")

print(f" Safety monitor: {monitor\_size / 1024:.2f} KB")

print(f" Total: {(agent\_size + beliefs\_size + monitor\_size) / 1024:.2f} KB")

if (agent\_size + beliefs\_size + monitor\_size) < 1024 \* 100: # < 100 KB

print(" ✅ MEMORY EFFICIENT (< 100 KB core)")

# ============================================================================

# MAIN DEMONSTRATION

# ============================================================================

def main():

"""

Run all demonstrations

"""

print("\n" + "█"\*70)

print("█" + " "\*68 + "█")

print("█" + " "\*15 + "COGNITIVE ARCHITECTURE DEMONSTRATION" + " "\*17 + "█")

print("█" + " "\*68 + "█")

print("█"\*70)

print("\n" + "="\*70)

print("A working implementation of:")

print(" • De se (self-locating) reasoning (Perry/Lewis)")

print(" • Formal safety verification (Z3 theorem prover)")

print(" • Propositional attitudes (Chalmers)")

print(" • Runtime safety monitoring")

print(" • Complete thought logging")

print("="\*70)

demos = [

("De Se Reasoning", demo\_de\_se\_reasoning),

("Formal Verification", demo\_formal\_verification),

("Propositional Attitudes", demo\_propositional\_attitudes),

("Runtime Monitoring", demo\_runtime\_monitoring),

("Complete Mission", demo\_complete\_mission),

("Thought Logging", demo\_thought\_logging),

("Performance Test", demo\_performance\_test)

]

print("\n" + "="\*70)

print("Available demonstrations:")

for i, (name, \_) in enumerate(demos, 1):

print(f" {i}. {name}")

print(" 0. Run all demonstrations")

print("="\*70)

try:

choice = input("\nSelect demonstration (0-7): ").strip()

if choice == "0":

for name, demo\_func in demos:

demo\_func()

input("\nPress Enter to continue to next demonstration...")

elif choice.isdigit() and 1 <= int(choice) <= len(demos):

demos[int(choice) - 1][1]()

else:

print("Invalid choice. Running all demonstrations...")

for name, demo\_func in demos:

demo\_func()

input("\nPress Enter to continue...")

except KeyboardInterrupt:

print("\n\nDemonstration interrupted.")

print("\n" + "="\*70)

print("DEMONSTRATION COMPLETE")

print("="\*70)

print("\n✅ All features demonstrated successfully!")

print("\nNext steps:")

print(" • Run tests: python tests/test\_suite.py")

print(" • Run benchmarks: python benchmarks/benchmark\_suite.py")

print(" • See examples: examples/demo.py")

print(" • Read docs: README.md")

print("\n" + "="\*70 + "\n")

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

main()