```
import json
import os
from unittest.mock import Mock

import pytest

from swarms import Agent
from swarm_models import OpenAIChat
from swarms.structs.multi_agent_collab import MultiAgentCollaboration

# Initialize the director agent

director = Agent(
    agent_name="Director",
    system_prompt="Directs the tasks for the workers",
    llm=OpenAIChat(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",
    saved_state_path="director.json",
)
```

```
# Initialize worker 1

worker1 = Agent(
    agent_name="Worker1",
    system_prompt="Generates a transcript for a youtube video on what swarms are",
    llm=OpenAIChat(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",
    saved_state_path="worker1.json",
)


# Initialize worker 2
worker2 = Agent(
    agent_name="Worker2",
    system_prompt="Summarizes the transcript generated by Worker1",
    llm=OpenAIChat(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
```

```
    state_save_file_type="json",
    saved_state_path="worker2.json",
)


# Create a list of agents
agents = [director, worker1, worker2]


@pytest.fixture
def collaboration():
    return MultiAgentCollaboration(agents)


def test_collaboration_initialization(collaboration):
    assert len(collaboration.agents) == 2
    assert callable(collaboration.select_next_speaker)
    assert collaboration.max_loops == 10
    assert collaboration.results == []
    assert collaboration.logging is True


def test_reset(collaboration):
    collaboration.reset()
    for agent in collaboration.agents:
        assert agent.step == 0
```

```
def test_inject(collaboration):
    collaboration.inject("TestName", "TestMessage")
    for agent in collaboration.agents:
        assert "TestName" in agent.history[-1]
        assert "TestMessage" in agent.history[-1]


def test_inject_agent(collaboration):
    agent3 = Agent(llm=OpenAIChat(), max_loops=2)
    collaboration.inject_agent(agent3)
    assert len(collaboration.agents) == 3
    assert agent3 in collaboration.agents


def test_step(collaboration):
    collaboration.step()
    for agent in collaboration.agents:
        assert agent.step == 1


def test_ask_for_bid(collaboration):
    agent = Mock()
    agent.bid.return_value = "<5>"
    bid = collaboration.ask_for_bid(agent)
```

```
    assert bid == 5



def test_select_next_speaker(collaboration):
    collaboration.select_next_speaker = Mock(return_value=0)
    idx = collaboration.select_next_speaker(1, collaboration.agents)
    assert idx == 0



def test_run(collaboration):
    collaboration.run()
    for agent in collaboration.agents:
        assert agent.step == collaboration.max_loops



def test_format_results(collaboration):
    collaboration.results = [
        {"agent": "Agent1", "response": "Response1"}
    ]
    formatted_results = collaboration.format_results(
        collaboration.results
    )
    assert "Agent1 responded: Response1" in formatted_results



def test_save_and_load(collaboration):
```

```
    collaboration.save()
    loaded_state = collaboration.load()
    assert loaded_state["_step"] == collaboration._step
    assert loaded_state["results"] == collaboration.results


def test_performance(collaboration):
    performance_data = collaboration.performance()
    for agent in collaboration.agents:
        assert agent.name in performance_data
        assert "metrics" in performance_data[agent.name]


def test_set_interaction_rules(collaboration):
    rules = {"rule1": "action1", "rule2": "action2"}
    collaboration.set_interaction_rules(rules)
    assert hasattr(collaboration, "interaction_rules")
    assert collaboration.interaction_rules == rules


def test_repr(collaboration):
    repr_str = repr(collaboration)
    assert isinstance(repr_str, str)
    assert "MultiAgentCollaboration" in repr_str
```

```
def test_load(collaboration):
    state = {
        "step": 5,
        "results": [{"agent": "Agent1", "response": "Response1"}],
    }
    with open(collaboration.saved_file_path_name, "w") as file:
        json.dump(state, file)

    loaded_state = collaboration.load()
    assert loaded_state["_step"] == state["step"]
    assert loaded_state["results"] == state["results"]


def test_save(collaboration, tmp_path):
    collaboration.saved_file_path_name = tmp_path / "test_save.json"
    collaboration.save()

    with open(collaboration.saved_file_path_name) as file:
        saved_data = json.load(file)

    assert saved_data["_step"] == collaboration._step
    assert saved_data["results"] == collaboration.results


# Add more tests here...
```

```
# Add more parameterized tests for different scenarios...



# Example of exception testing
def test_exception_handling(collaboration):
    agent = Mock()
    agent.bid.side_effect = ValueError("Invalid bid")
    with pytest.raises(ValueError):
        collaboration.ask_for_bid(agent)



# Add more exception testing...



# Example of environment variable testing (if applicable)
@pytest.mark.parametrize("env_var", ["ENV_VAR_1", "ENV_VAR_2"])
def test_environment_variables(collaboration, monkeypatch, env_var):
    monkeypatch.setenv(env_var, "test_value")
    assert os.getenv(env_var) == "test_value"
```