```
import os

import traceback

from datetime import datetime

from typing import Callable, Dict, List, Optional


from loguru import logger

from swarm_models import OpenAIChat


from swarms.structs.agent import Agent

from swarms.structs.rearrange import AgentRearrange



class TestResult:

    """Class to store test results and metadata"""

    def __init__(self, test_name: str):

        self.test_name = test_name

        self.start_time = datetime.now()

        self.end_time = None

        self.success = False

        self.error = None

        self.traceback = None

        self.function_output = None


    def complete(self, success: bool, error: Optional[Exception] = None):

        """Complete the test execution with results"""

        self.end_time = datetime.now()
        self.success = success
        if error:
            self.error = str(error)
            self.traceback = traceback.format_exc()

    def duration(self) -> float:
        """Calculate test duration in seconds"""
        if self.end_time:
            return (self.end_time - self.start_time).total_seconds()
        return 0

def run_test(test_func: Callable) -> TestResult:
    """
    Decorator to run tests with error handling and logging

    Args:
        test_func (Callable): Test function to execute

    Returns:
        TestResult: Object containing test execution details
    """
    def wrapper(*args, **kwargs) -> TestResult:
        result = TestResult(test_func.__name__)
        logger.info(f"\n{'='*20} Running test: {test_func.__name__} {'='*20}")

        try:
            output = test_func(*args, **kwargs)
            result.function_output = output
            result.complete(success=True)
            logger.success(f" Test {test_func.__name__} passed successfully")

        except Exception as e:
            result.complete(success=False, error=e)
            logger.error(f" Test {test_func.__name__} failed with error: {str(e)}")
            logger.error(f"Traceback: {traceback.format_exc()}")

        logger.info(f"Test duration: {result.duration():.2f} seconds\n")
        return result

    return wrapper

def create_functional_agents() -> List[Agent]:
    """
    Create a list of functional agents with real LLM integration for testing.
    Using OpenAI's GPT model for realistic agent behavior testing.
    """
    # Initialize OpenAI Chat model
    api_key = os.getenv("OPENAI_API_KEY")
    if not api_key:
        logger.warning("No OpenAI API key found. Using mock agents instead.")
        return [create_mock_agent("TestAgent1"), create_mock_agent("TestAgent2")]
```

```
try:
    model = OpenAIChat(
        api_key=api_key,
        model_name="gpt-4o",
        temperature=0.1
    )

    # Create boss agent
    boss_agent = Agent(
        agent_name="BossAgent",
        system_prompt="""
        You are the BossAgent responsible for managing and overseeing test scenarios.
        Your role is to coordinate tasks between agents and ensure efficient collaboration.
        Analyze inputs, break down tasks, and provide clear directives to other agents.
        Maintain a structured approach to task management and result compilation.
        """,
        llm=model,
        max_loops=1,
        dashboard=False,
        streaming_on=True,
        verbose=True,
        stopping_token="<DONE>",
        state_save_file_type="json",
        saved_state_path="test_boss_agent.json",
    )
```

```
# Create analysis agent
analysis_agent = Agent(
    agent_name="AnalysisAgent",
    system_prompt="""
    You are the AnalysisAgent responsible for detailed data processing and analysis.
    Your role is to examine input data, identify patterns, and provide analytical insights.
    Focus on breaking down complex information into clear, actionable components.
    """,
    llm=model,
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",
    saved_state_path="test_analysis_agent.json",
)

# Create summary agent
summary_agent = Agent(
    agent_name="SummaryAgent",
    system_prompt="""
    You are the SummaryAgent responsible for consolidating and summarizing information.
    Your role is to take detailed analysis and create concise, actionable summaries.
    Focus on highlighting key points and ensuring clarity in communication.
    """,
            llm=model,

            max_loops=1,

            dashboard=False,

            streaming_on=True,

            verbose=True,

            stopping_token="<DONE>",

            state_save_file_type="json",

            saved_state_path="test_summary_agent.json",

        )


        logger.info("Successfully created functional agents with LLM integration")

        return [boss_agent, analysis_agent, summary_agent]


    except Exception as e:

        logger.error(f"Failed to create functional agents: {str(e)}")

        logger.warning("Falling back to mock agents")

        return [create_mock_agent("TestAgent1"), create_mock_agent("TestAgent2")]


def create_mock_agent(name: str) -> Agent:

    """Create a mock agent for testing when LLM integration is not available"""

    return Agent(

        agent_name=name,

        system_prompt=f"You are a test agent named {name}",

        llm=None

    )
```

```
@run_test
def test_init():
    """Test AgentRearrange initialization with functional agents"""
    logger.info("Creating agents for initialization test")
    agents = create_functional_agents()

    rearrange = AgentRearrange(
        name="TestRearrange",
        agents=agents,
        flow=f"{agents[0].agent_name} -> {agents[1].agent_name} -> {agents[2].agent_name}"
    )

    assert rearrange.name == "TestRearrange"
    assert len(rearrange.agents) == 3
        assert rearrange.flow == f"{agents[0].agent_name} -> {agents[1].agent_name} ->
{agents[2].agent_name}"

    logger.info(f"Initialized AgentRearrange with {len(agents)} agents")
    return True


@run_test
def test_validate_flow():
    """Test flow validation logic"""
    agents = create_functional_agents()
    rearrange = AgentRearrange(
        agents=agents,
        flow=f"{agents[0].agent_name} -> {agents[1].agent_name}"
    )

    logger.info("Testing valid flow pattern")
    valid = rearrange.validate_flow()
    assert valid is True

    logger.info("Testing invalid flow pattern")
    rearrange.flow = f"{agents[0].agent_name} {agents[1].agent_name}"  # Missing arrow
    try:
        rearrange.validate_flow()
        assert False, "Should have raised ValueError"
    except ValueError as e:
        logger.info(f"Successfully caught invalid flow error: {str(e)}")
        assert True

    return True

@run_test
def test_add_remove_agent():
    """Test adding and removing agents from the swarm"""
    agents = create_functional_agents()
    rearrange = AgentRearrange(agents=agents[:2])  # Start with first two agents

    logger.info("Testing agent addition")
    new_agent = agents[2]  # Use the third agent as new agent
    rearrange.add_agent(new_agent)
    assert new_agent.agent_name in rearrange.agents

    logger.info("Testing agent removal")
    rearrange.remove_agent(new_agent.agent_name)
    assert new_agent.agent_name not in rearrange.agents

    return True

@run_test
def test_basic_run():
    """Test basic task execution with the swarm"""
    agents = create_functional_agents()
    rearrange = AgentRearrange(
        name="TestSwarm",
        agents=agents,
        flow=f"{agents[0].agent_name} -> {agents[1].agent_name} -> {agents[2].agent_name}",
        max_loops=1
    )

    test_task = "Analyze this test message and provide a brief summary."
    logger.info(f"Running test task: {test_task}")

    try:
        result = rearrange.run(test_task)
        assert result is not None
        logger.info(f"Successfully executed task with result length: {len(str(result))}")
        return True
    except Exception as e:
        logger.error(f"Task execution failed: {str(e)}")
        raise


def run_all_tests() -> Dict[str, TestResult]:
    """
    Run all test cases and collect results


    Returns:
        Dict[str, TestResult]: Dictionary mapping test names to their results
    """
    logger.info("\n Starting AgentRearrange test suite execution")
    test_functions = [
        test_init,
        test_validate_flow,
        test_add_remove_agent,
        test_basic_run
    ]


    results = {}
    for test in test_functions:
        result = test()
        results[test.__name__] = result
    # Log summary
    total_tests = len(results)
    passed_tests = sum(1 for r in results.values() if r.success)
    failed_tests = total_tests - passed_tests

    logger.info("\n Test Suite Summary:")
    logger.info(f"Total Tests: {total_tests}")
    print(f" Passed: {passed_tests}")

    if failed_tests > 0:
        logger.error(f" Failed: {failed_tests}")

    # Detailed failure information
    if failed_tests > 0:
        logger.error("\n Failed Tests Details:")
        for name, result in results.items():
            if not result.success:
                logger.error(f"\n{name}:")
                logger.error(f"Error: {result.error}")
                logger.error(f"Traceback: {result.traceback}")

    return results

if __name__ == "__main__":
    print(" Starting AgentRearrange Test Suite")
    results = run_all_tests()
```

```
print(" Test Suite Execution Completed")
```