```
import pytest

from unittest.mock import MagicMock

from swarms import AgentRearrange




class MockAgent:

    def __init__(self, name):

        self.name = name


    def run(self, task, img=None, *args, **kwargs):

        return f"{self.name} processed {task}"




@pytest.fixture

def mock_agents():

    return [

        MockAgent(name="Agent1"),

        MockAgent(name="Agent2"),

        MockAgent(name="Agent3"),

    ]




@pytest.fixture

def agent_rearrange(mock_agents):

    return AgentRearrange(

        agents=mock_agents, flow="Agent1 -> Agent2 -> Agent3"
    )


def test_initialization(mock_agents):
    agent_rearrange = AgentRearrange(
        agents=mock_agents, flow="Agent1 -> Agent2 -> Agent3"
    )
    assert len(agent_rearrange.agents) == 3
    assert agent_rearrange.flow == "Agent1 -> Agent2 -> Agent3"


def test_add_agent(agent_rearrange):
    new_agent = MockAgent(name="Agent4")
    agent_rearrange.add_agent(new_agent)
    assert "Agent4" in agent_rearrange.agents


def test_remove_agent(agent_rearrange):
    agent_rearrange.remove_agent("Agent2")
    assert "Agent2" not in agent_rearrange.agents


def test_add_agents(agent_rearrange):
    new_agents = [MockAgent(name="Agent4"), MockAgent(name="Agent5")]
    agent_rearrange.add_agents(new_agents)
    assert "Agent4" in agent_rearrange.agents
    assert "Agent5" in agent_rearrange.agents


def test_validate_flow_valid(agent_rearrange):
    assert agent_rearrange.validate_flow() is True


def test_validate_flow_invalid(agent_rearrange):
    agent_rearrange.flow = "Agent1 -> Agent4"
    with pytest.raises(ValueError):
        agent_rearrange.validate_flow()


def test_run(agent_rearrange):
    result = agent_rearrange.run("Test Task")
    assert (
        result
            == "Agent1 processed Test Task; Agent2 processed Agent1 processed Test Task; Agent3
processed Agent2 processed Agent1 processed Test Task"
    )


def test_run_with_custom_tasks(agent_rearrange):
    custom_tasks = {"Agent2": "Custom Task"}
    result = agent_rearrange.run(
        "Test Task", custom_tasks=custom_tasks
    )
    assert (
        result
         == "Agent1 processed Test Task; Agent2 processed Custom Task; Agent3 processed Agent2 processed Custom Task"
    )


def test_run_with_human_intervention(agent_rearrange):
    agent_rearrange.human_in_the_loop = True
    agent_rearrange.custom_human_in_the_loop = MagicMock(
        return_value="Human processed Task"
    )
    agent_rearrange.flow = "Agent1 -> H -> Agent3"
    result = agent_rearrange.run("Test Task")
    assert (
        result
            == "Agent1 processed Test Task; Human processed Task; Agent3 processed Human processed Task"
    )


def test_run_sub_swarm(agent_rearrange):
    sub_swarm_flow = "Agent1 -> Agent3"
    agent_rearrange.add_sub_swarm("SubSwarm1", sub_swarm_flow)
    result = agent_rearrange.run_sub_swarm(
        "SubSwarm1", "Sub Task", None
    )
    assert (
        result
        == "Agent1 processed Sub Task; Agent3 processed Agent1 processed Sub Task"
    )


def test_process_agent_or_swarm(agent_rearrange):
    result = agent_rearrange.process_agent_or_swarm(
        "Agent1", "Process Task", None
    )
    assert result == "Agent1 processed Process Task"


def test_track_history(agent_rearrange):
    agent_rearrange.track_history("Agent1", "Task Result")
    assert agent_rearrange.swarm_history["Agent1"] == ["Task Result"]


def test_human_intervention(agent_rearrange):
    agent_rearrange.human_in_the_loop = True
    agent_rearrange.custom_human_in_the_loop = MagicMock(
        return_value="Human processed Task"
    )
    result = agent_rearrange.human_intervention("Task")
```

```
assert result == "Human processed Task"
```