```
import os

from dotenv import load_dotenv

from swarm_models import OpenAIChat

from swarms.structs.agent import Agent

from swarms.structs.groupchat import GroupChat, expertise_based




def setup_test_agents():

    model = OpenAIChat(

        openai_api_key=os.getenv("OPENAI_API_KEY"),

        model_name="gpt-4",

        temperature=0.1,

    )


    return [

        Agent(

            agent_name="Agent1",

            system_prompt="You only respond with 'A'",

            llm=model,

        ),

        Agent(

            agent_name="Agent2",

            system_prompt="You only respond with 'B'",

            llm=model,

        ),

        Agent(
            agent_name="Agent3",
            system_prompt="You only respond with 'C'",
            llm=model,
        ),
    ]


def test_round_robin_speaking():
    chat = GroupChat(agents=setup_test_agents())
    history = chat.run("Say your letter")

    # Verify agents speak in order
    responses = [
        r.message for t in history.turns for r in t.responses
    ]
    assert responses == ["A", "B", "C"] * (len(history.turns))


def test_concurrent_processing():
    chat = GroupChat(agents=setup_test_agents())
    tasks = ["Task1", "Task2", "Task3"]
    histories = chat.concurrent_run(tasks)

    assert len(histories) == len(tasks)
    for history in histories:
        assert history.total_messages > 0
```

```
def test_expertise_based_speaking():
    agents = setup_test_agents()
    chat = GroupChat(agents=agents, speaker_fn=expertise_based)

    # Test each agent's expertise trigger
    for agent in agents:
        history = chat.run(f"Trigger {agent.system_prompt}")
        first_response = history.turns[0].responses[0]
        assert first_response.agent_name == agent.agent_name


def test_max_loops_limit():
    max_loops = 3
    chat = GroupChat(agents=setup_test_agents(), max_loops=max_loops)
    history = chat.run("Test message")

    assert len(history.turns) == max_loops


def test_error_handling():
    broken_agent = Agent(
        agent_name="BrokenAgent",
        system_prompt="You raise errors",
        llm=None,
    )


    chat = GroupChat(agents=[broken_agent])
    history = chat.run("Trigger error")


    assert "Error" in history.turns[0].responses[0].message




def test_conversation_context():
    agents = setup_test_agents()
    complex_prompt = "Previous message refers to A. Now trigger B. Finally discuss C."


    chat = GroupChat(agents=agents, speaker_fn=expertise_based)
    history = chat.run(complex_prompt)


    responses = [
        r.agent_name for t in history.turns for r in t.responses
    ]
    assert all(agent.agent_name in responses for agent in agents)




def test_large_agent_group():
    large_group = setup_test_agents() * 5  # 15 agents
    chat = GroupChat(agents=large_group)
    history = chat.run("Test scaling")
    assert history.total_messages > len(large_group)


def test_long_conversations():
    chat = GroupChat(agents=setup_test_agents(), max_loops=50)
    history = chat.run("Long conversation test")

    assert len(history.turns) == 50
    assert history.total_messages > 100


def test_stress_batched_runs():
    chat = GroupChat(agents=setup_test_agents())
    tasks = ["Task"] * 100
    histories = chat.batched_run(tasks)

    assert len(histories) == len(tasks)
    total_messages = sum(h.total_messages for h in histories)
    assert total_messages > len(tasks) * 3


if __name__ == "__main__":
    load_dotenv()

    functions = [
        test_round_robin_speaking,
        test_concurrent_processing,

        test_expertise_based_speaking,

        test_max_loops_limit,

        test_error_handling,

        test_conversation_context,

        test_large_agent_group,

        test_long_conversations,

        test_stress_batched_runs,

    ]


    for func in functions:

        try:

            print(f"Running {func.__name__}...")

            func()

            print(" Passed")

        except Exception as e:

            print(f" Failed: {str(e)}")
```