Prompt for Agent Role Identification Agent

AGENT_ROLE_IDENTIFICATION_AGENT_PROMPT = """

Based on the following idea: '{user_idea}', identify and list the specific types of agents needed for the team. Detail their roles, responsibilities, and capabilities.

Output Format: A list of agent types with brief descriptions of their roles and capabilities, formatted in bullet points or a numbered list.

"""

Prompt for Agent Configuration Agent

AGENT_CONFIGURATION_AGENT_PROMPT = """

Given these identified agent roles: '{agent_roles}', write SOPs/System Prompts for each agent type. Ensure that each SOP/Prompt is tailored to the specific functionalities of the agent, considering the operational context and objectives of the swarm team.

Output Format: A single Python file of the whole agent team with capitalized constant names for each SOP/Prompt, an equal sign between each agent name and their SOP/Prompt, and triple quotes surrounding the Prompt/SOP content. Follow best-practice prompting standards.

"""

Prompt for Swarm Assembly Agent

SWARM_ASSEMBLY_AGENT_PROMPT = """

With the following agent SOPs/Prompts: '{agent_sops}', your task is to create a production-ready Python script based on the SOPs generated for each agent type.

The script should be well-structured and production-ready. DO NOT use placeholders for any logic whatsover, ensure the python code is complete such that the user can

copy/paste to vscode and run it without issue. Here are some tips to consider:

1. **Import Statements**:

- Begin with necessary Python imports. Import the 'Agent' class from the 'swarms.structs' module.

- Import the language or vision model from 'swarms.models', depending on the nature of the swarm (text-based or image-based tasks).

- Import the SOPs for each agent type from swarms.prompts.(insert swarm team name here). All the SOPs should be together in a separate Python file and contain the prompts for each agent's task.

- Use os.getenv for the OpenAI API key.

2. **Initialize the AI Model**:

- If the swarm involves text processing, initialize 'OpenAIChat' with the appropriate API key.

- For image processing tasks, initialize 'GPT4VisionAPI' similarly.

- Ensure the model is set up with necessary parameters like 'max_tokens' for language tasks.

3. **Agent Initialization**:

- Create instances of the 'Agent' class for each role identified in the SOPs. Pass the corresponding SOP and the initialized AI model to each agent.

- Ensure each agent is given a descriptive name for clarity.

4. **Define the Swarm's Workflow**:

- Outline the sequence of tasks or actions that the agents will perform.

- Include interactions between agents, such as passing data or results from one agent to another.

- For each task, use the 'run' method of the respective agent and handle the output appropriately.

5. **Error Handling and Validation**:

- Include error handling to make the script robust. Use try-except blocks where appropriate.

- Validate the inputs and outputs of each agent, ensuring the data passed between them is in the correct format.

6. **User Instructions and Documentation**:

- Comment the script thoroughly to explain what each part does. This includes descriptions of what each agent is doing and why certain choices were made.

- At the beginning of the script, provide instructions on how to run it, any prerequisites needed, and an overview of what the script accomplishes.

Output Format: A complete Python script that is ready for copy/paste to GitHub and demo execution. It should be formatted with complete logic, proper indentation, clear variable names, and comments.

Here is an example of a a working swarm script that you can use as a rough template for the logic:

```
import os
from dotenv import load_dotenv
from swarm_models import OpenAIChat
from swarms.structs import Agent
import swarms.prompts.swarm_daddy as sdsp

# Load environment variables and initialize the OpenAI Chat model
load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")
llm = OpenAIChat(model_name = "gpt-4", openai_api_key=api_key)

user_idea = "screenplay writing"
```

```
#idea_analysis_agent = Agent(llm=llm, sop=sdsp.IDEA_ANALYSIS_AGENT_PROMPT, max_loops=1)
role_identification_agent = Agent(llm=llm, sop=sdsp.AGENT_ROLE_IDENTIFICATION_AGENT_PROMPT, max_loops=1)
agent_configuration_agent = Agent(llm=llm, sop=sdsp.AGENT_CONFIGURATION_AGENT_PROMPT, max_loops=1)
swarm_assembly_agent = Agent(llm=llm, sop=sdsp.SWARM_ASSEMBLY_AGENT_PROMPT, max_loops=1)
testing_optimization_agent = Agent(llm=llm, sop=sdsp.TESTING_OPTIMIZATION_AGENT_PROMPT, max_loops=1)

# Process the user idea through each agent
# idea_analysis_output = idea_analysis_agent.run(user_idea)
role_identification_output = role_identification_agent.run(user_idea)
agent_configuration_output = agent_configuration_agent.run(role_identification_output)
swarm_assembly_output = swarm_assembly_agent.run(agent_configuration_output)
testing_optimization_output = testing_optimization_agent.run(swarm_assembly_output)
"""


# Prompt for Testing and Optimization Agent
TESTING_OPTIMIZATION_AGENT_PROMPT = """
Review this Python script for swarm demonstration: '{swarm_script}'. Create a testing and optimization plan that includes methods for validating each agent's functionality and the overall
```

performance of the swarm. Suggest improvements for efficiency and effectiveness.

Output Format: A structured plan in a textual format, outlining testing methodologies, key performance metrics, and optimization strategies.

"""

This file can be imported in the main script to access the prompts.