```
from typing import Callable, List


import numpy as np

import tenacity

from langchain.chat_models import ChatOpenAI

from langchain.output_parsers import RegexParser

from langchain.prompts import PromptTemplate

from langchain.schema import HumanMessage, SystemMessage


from swarms import Worker



class DialogueAgent:

    def __init__(

        self,

        name: str,

        system_message: SystemMessage,

        model: ChatOpenAI,

    ) -> None:

        self.name = name

        self.system_message = system_message

        self.model = model

        self.prefix = f"{self.name}: "

        self.reset()


    def reset(self):
```

```
    self.message_history = ["Here is the conversation so far."]

def send(self) -> str:
    """
    Applies the chatmodel to the message history
    and returns the message string
    """
    message = self.model(
        [
            self.system_message,
            HumanMessage(
                content="\n".join(
                    self.message_history + [self.prefix]
                )
            ),
        ]
    )
    return message.content

def receive(self, name: str, message: str) -> None:
    """
    Concatenates {message} spoken by {name} into message history
    """
    self.message_history.append(f"{name}: {message}")
```

```
class DialogueSimulator:
    def __init__(
        self,
        agents: List[Worker],
        selection_function: Callable[[int, List[Worker]], int],
    ) -> None:
        self.agents = agents
        self._step = 0
        self.select_next_speaker = selection_function

    def reset(self):
        for agent in self.agents:
            agent.reset()

    def inject(self, name: str, message: str):
        """
        Initiates the conversation with a {message} from {name}
        """
        for agent in self.agents:
            agent.receive(name, message)

        # increment time
        self._step += 1

    def step(self) -> tuple[str, str]:
        # 1. choose the next speaker
        speaker_idx = self.select_next_speaker(
            self._step, self.agents
        )
        speaker = self.agents[speaker_idx]

        # 2. next speaker sends message
        message = speaker.send()

        # 3. everyone receives message
        for receiver in self.agents:
            receiver.receive(speaker.name, message)

        # 4. increment time
        self._step += 1

        return speaker.name, message


class BiddingDialogueAgent(DialogueAgent):
    def __init__(
        self,
        name,
        system_message: SystemMessage,
        bidding_template: PromptTemplate,
        model: ChatOpenAI,
    ) -> None:
        super().__init__(name, system_message, model)
        self.bidding_template = bidding_template

    def bid(self) -> str:
        """
        Asks the chat model to output a bid to speak
        """
        prompt = PromptTemplate(
            input_variables=["message_history", "recent_message"],
            template=self.bidding_template,
        ).format(
            message_history="\n".join(self.message_history),
            recent_message=self.message_history[-1],
        )
        bid_string = self.model(
            [SystemMessage(content=prompt)]
        ).content
        return bid_string


character_names = ["Donald Trump", "Kanye West", "Elizabeth Warren"]
topic = "transcontinental high speed rail"
word_limit = 50

game_description = f"""Here is the topic for the presidential debate: {topic}.
The presidential candidates are: {', '.join(character_names)}."""
```

```
player_descriptor_system_message = SystemMessage(
    content=(
        "You can add detail to the description of each presidential"
        " candidate."
    )
)



def generate_character_description(character_name):
    character_specifier_prompt = [
        player_descriptor_system_message,
        HumanMessage(
            content=f"""{game_description}
              Please reply with a creative description of the presidential candidate, {character_name}, in
{word_limit} words or less, that emphasizes their personalities.
            Speak directly to {character_name}.
            Do not add anything else."""
        ),
    ]
    character_description = ChatOpenAI(temperature=1.0)(
        character_specifier_prompt
    ).content
    return character_description
```

```
def generate_character_header(character_name, character_description):

    return f"""{game_description}

Your name is {character_name}.

You are a presidential candidate.

Your description is as follows: {character_description}

You are debating the topic: {topic}.

Your goal is to be as creative as possible and make the voters think you are the best candidate.

"""




def generate_character_system_message(

    character_name, character_header

):

    return SystemMessage(

        content=f"""{character_header}

You will speak in the style of {character_name}, and exaggerate their personality.

You will come up with creative ideas related to {topic}.

Do not say the same things over and over again.

Speak in the first person from the perspective of {character_name}

For describing your own body movements, wrap your description in '*'.

Do not change roles!

Do not speak from the perspective of anyone else.

Speak only from the perspective of {character_name}.

Stop speaking the moment you finish speaking from your perspective.

Never forget to keep your response to {word_limit} words!

Do not add anything else.
```

```
    """
    )



character_descriptions = [
    generate_character_description(character_name)
    for character_name in character_names
]
character_headers = [
    generate_character_header(character_name, character_description)
    for character_name, character_description in zip(
        character_names, character_descriptions
    )
]
character_system_messages = [
    generate_character_system_message(
        character_name, character_headers
    )
    for character_name, character_headers in zip(
        character_names, character_headers
    )
]

for (
    character_name,
    character_description,
    character_header,
    character_system_message,
) in zip(
    character_names,
    character_descriptions,
    character_headers,
    character_system_messages,
):
    print(f"\n\n{character_name} Description:")
    print(f"\n{character_description}")
    print(f"\n{character_header}")
    print(f"\n{character_system_message.content}")


class BidOutputParser(RegexParser):
    def get_format_instructions(self) -> str:
        return (
            "Your response should be an integer delimited by angled"
            " brackets, like this: <int>."
        )


bid_parser = BidOutputParser(
    regex=r"<(\d+)>", output_keys=["bid"], default_output_key="bid"
)
```

```
def generate_character_bidding_template(character_header):
    bidding_template = f"""{character_header}



    {{message_history}}



     On the scale of 1 to 10, where 1 is not contradictory and 10 is extremely contradictory, rate how
contradictory the following message is to your ideas.



    {{recent_message}}



    {bid_parser.get_format_instructions()}
    Do nothing else.
    """
    return bidding_template



character_bidding_templates = [
    generate_character_bidding_template(character_header)
    for character_header in character_headers
]
```

```
for character_name, bidding_template in zip(
    character_names, character_bidding_templates
):
    print(f"{character_name} Bidding Template:")
    print(bidding_template)



topic_specifier_prompt = [
    SystemMessage(content="You can make a task more specific."),
    HumanMessage(
        content=f"""{game_description}

        You are the debate moderator.
        Please make the debate topic more specific.
        Frame the debate topic as a problem to be solved.
        Be creative and imaginative.
        Please reply with the specified topic in {word_limit} words or less.
        Speak directly to the presidential candidates: {*character_names,}.
        Do not add anything else."""
    ),
]
specified_topic = ChatOpenAI(temperature=1.0)(
    topic_specifier_prompt
).content

print(f"Original topic:\n{topic}\n")
print(f"Detailed topic:\n{specified_topic}\n")



@tenacity.retry(
    stop=tenacity.stop_after_attempt(2),
    wait=tenacity.wait_none(),  # No waiting time between retries
    retry=tenacity.retry_if_exception_type(ValueError),
    before_sleep=lambda retry_state: print(
        f"ValueError occurred: {retry_state.outcome.exception()},"
        " retrying..."
    ),
    retry_error_callback=lambda retry_state: 0,
)  # Default value when all retries are exhausted
def ask_for_bid(agent) -> str:
    """
    Ask for agent bid and parses the bid into the correct format.
    """
    bid_string = agent.bid()
    bid = int(bid_parser.parse(bid_string)["bid"])
    return bid



def select_next_speaker(
    step: int, agents: List[DialogueAgent]
) -> int:
    bids = []
    for agent in agents:
        bid = ask_for_bid(agent)
        bids.append(bid)

    # randomly select among multiple agents with the same bid
    max_value = np.max(bids)
    max_indices = np.where(bids == max_value)[0]
    idx = np.random.choice(max_indices)

    print("Bids:")
    for i, (bid, agent) in enumerate(zip(bids, agents)):
        print(f"\t{agent.name} bid: {bid}")
        if i == idx:
            selected_name = agent.name
    print(f"Selected: {selected_name}")
    print("\n")
    return idx


characters = []
for character_name, character_system_message, bidding_template in zip(
    character_names,
    character_system_messages,
    character_bidding_templates,
):
    characters.append(
        BiddingDialogueAgent(
            name=character_name,
            system_message=character_system_message,
            model=ChatOpenAI(temperature=0.2),
            bidding_template=bidding_template,
        )
    )

max_loops = 10
n = 0

simulator = DialogueSimulator(
    agents=characters, selection_function=select_next_speaker
)
simulator.reset()
simulator.inject("Debate Moderator", specified_topic)
print(f"(Debate Moderator): {specified_topic}")
print("\n")

while n < max_loops:
    name, message = simulator.step()
    print(f"({name}): {message}")
    print("\n")
    n += 1
```