EXPLORING THE USE OF AI AGENTS TO SIMULATE HUMAN BEHAVIOR IN GROUP DECISION-MAKING

by

HANNAH TAWASHY

(Under the Direction of Neal Outland)

ABSTRACT

This study explores the inefficiencies inherent in group decision-making processes, particularly when critical information is dispersed among team members. The limitations of traditional human-based research methods prompt an AI-driven approach to model discussion. CogSystem is designed for simulating group decision-making, examining the interplay between group-level processes and individual-level cognition. The system comprises of CogFrame, a discussion framework, and CogChain, a cognitive architecture. CogFrame allows for the manipulation of discussion length, information distribution, and decision rules. CogChains are introduced to LLM-based agents to enhance their realism and simulate the cognitive processes influencing contribution of information items. Each CogChain captures a different individual level factor, including motivations, memory, and trust. Combinations of CogFrame and CogChain configurations are tested to investigate their impact on the optimality of the discussion result and human-like behavior of agents. The results offer insights into enhancing behavior modeling, decision-making outcomes, and human-AI collaboration.

INDEX WORDS: Cognitive Architecture, Large Language Model, Decision Model, Group Discussion Framework, Behavior Model, Hidden Profiles

EXPLORING THE USE OF AI AGENTS TO SIMULATE HUMAN BEHAVIOR IN GROUP DECISION-MAKING

by

HANNAH TAWASHY

B.S., The University of Georgia, 2022

B.S., The University of Georgia, 2022

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2024

EXPLORING THE USE OF AI AGENTS TO SIMULATE HUMAN BEHAVIOR IN GROUP DECISION-MAKING

by

HANNAH TAWASHY

DEDICATION

I want to dedicate this to my parents. To my father, for supporting the pursuit of my dreams even when it hurt. To my mother, who laid out the blueprint for achieving what is seemingly impossible. My sister, Sarah, and my best friend, Becca, for their unwavering confidence. When I struggled to see the completion of my research, I leaned on their support.

Although my mind was here, my heart was with my family in Gaza, Palestine.

ACKNOWLEDGEMENTS

I grew tremendously during this journey as an individual and an academic. I’d like to thank the people who shaped my experience. Evette, the heart of the AI institute, whose door is always open. Dr. Maier, Dr. Rasheed, and Dr. Outland, who spent countless hours in discussions with me. Their guidance led to the completion of my degree. Dr. Doshi, who provided me with the opportunity to work in a lab. I wish the best to my lab-mates, who were there for me through the highs and lows.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Decision making groups are deployed across a wide range of tasks, from simple with mundane implications to complex with irreversible consequences. The adage ‘there is wisdom in numbers’ is not without evidence. Decisions made by group consensus, in comparison to individuals, may lead to overall higher quality and reliability (Stasser & Titus, 1985). A group represents an array of viewpoints, and this collective knowledge potentially resolves issues of social judgment and preference (Stasser, 1988). They facilitate the combining of knowledge, skills, and capabilities of their team members (Martinez-Miranda & Pavon, 2012). This is reflected in decisions with large-scale impact, such as democratic governance systems, and ethically challenging undertones, like jury deliberations. Discussions can be framed as an exchange of information, where group members contribute information items as they work towards their decision. The decision-making process includes discovering and integrating information items to generate evaluations of decision options and arriving at a consensus (Stasser, 1988). Groups function as information processors, intaking relevant information to perform cognitive tasks to reach their goal (Hinsz et al., 1997). With a larger information capacity, groups are better equipped to make an informed decision in comparison to individuals.

Unfortunately, a comprehensive exchange of information cannot always be guaranteed. The nature of collaboration may result in the utilization of only 70% of the groups’ storage capacity, stemming from inefficient information processing (Hinsz et al., 1997). Group and individual level factors, including communication barriers, cognitive biases, and interpersonal dynamics, can impede the free flow of information exchange. Group settings often fail to recognize that one option is better supported than another, as a direct consequence of the limited information exchange failing to uncover the hidden profile.

Hidden Profiles

A hidden profile is the set of information items that indicates a better alternative among decision options (Stasser, 1988). Initially, the information items within this set are dispersed across individual group members. Information item utility is defined with respect to the positive or negative semantic connotation they cast on decision alternatives. As members discuss, there are many group-level and individual level processes influencing which information items are mentioned. Group discussions are not a perfect mechanism in balancing the complex interrelationships between these influences to maximize the amount of information exchanged (Stasser & Titus, 1985). As a result, uninformed decisions are possible, leading to suboptimal outcomes. The failure to uncover hidden profiles underscores the inherent limitations of group discussions as decision-makers. While the exchange of information within groups encourages the exposure of various perspectives, it does not guarantee the identification and synthesis of all relevant information. There is wisdom in numbers—sometimes. Evidently, the efficacy of a group correlates as a proportional function to the amount of information shared.

Investigating human-collaborative decision making provides insights into when and why groups do not arrive at optimal decisions. Analyzing the hidden profile phenomenon requires separating group interaction, the means of information exchange, from individual processing (Hinsz et al., 1997). There is a bidirectional flow of influence between group processing and individual level processing. The trading of information items can be examined to uncover causal associations between specific group and individual level processes. These associations can be used to predict the distribution of preferences, such as group consensus or opinion changes (Stasser, 1988). Modeling this information provides a theoretical framework for a deeper understanding of human collaboration, serving as a practical tool to develop strategies that mitigate the risk of uninformed decisions.

The Potential of AI in Simulating Human Behavior

Comprehensively modeling information exchange involves tracking an exponentially increasing number of variables, all of which have their own potential outside influence. Investigating this exhaustive list of variables affecting both the individual and group level processes, and their subsequent interactions, is arguably intractable. Existing literature slims down this list to a digestible scope by noting the processes that are most relevant to interaction, such as initial information distribution (Stasser & Titus, 1985), expert role assignments (Stasser et al., 2000), and socio-emotional attributes (Stasser & Davis, 1981). This human-based research is limited insofar as the inability to directly measure the relevant factors in a white-box manner, such as tracking the internal states of participants. Additionally, researchers are limited by the pre-disposed attributes of participants that affect group dynamics and individual processing. Computer modeling provides convenience to researchers, allowing them to vary parameters to conduct large batches of low-cost simulations of typically tedious studies. Participants can be synthesized using artificially intelligent agents and deployed in discussion simulation environments. This level of control allows for granular experimentation to investigate a causal relationship between specific group and individual level processes.

Artificial intelligence has evolved exponentially in recent years, particularly with the research in generative AI and the development of generative pre-trained transformers (GPTs). GPTs are a transformer architecture, training a deep neural network on vast amounts of human written text data. The network learns the connecting weights between nodes necessary to generate coherent and contextually relevant responses (Yenduri et al., 2023). Large language models (LLMs), a type of GPT, afford intelligent agents the capabilities of simulating human-like language. GPT-4, a model developed by OpenAI, passed the Turing test with remarkable consistency; long considered the milestone assessment of conversational intelligence (Jones & Bergen, 2023). LLMs offer a platform for systematic experimentation, enabling researchers to manipulate variables such as temperature, seed, and token limits. Unlike human participants, who may vary in their availability, reliability, and responsiveness, LLM-based agents can be deployed consistently and repeatedly across numerous experimental conditions. Conducting experiments with human participants typically entails maintaining records of confidentiality, consent, and specific training. They also require extensive preparation, such as finding willing participants and scheduling appointments. These administrative tasks can impose practical limitations on the scale of research, especially when focusing on discussion tasks that may contain sensitive topics. AI agents do not possess consciousness, eliminating ethical concerns regarding participant privacy and well-being.

While LLM-backed agents successfully mimic the surface-level dynamics of human interactions, they struggle to emulate the depth and complexity of human cognition. Their high linguistic capacity is not enough to meet the criterion to simulate human-like thinking, which is necessary when designing the agents for discussion simulation. LLMs lack cognitive modality as there is no semantic understanding of the content parsed or generated (Yenduri et al., 2023). This limitation becomes apparent in contexts that require nuanced logical reasoning, deep comprehension, or context-aware decision-making. This lack of ability to ‘think humanly’ undermines the usage of purely LLM-based agents in computer models investigating individual level processes that influence group discussions. The first step to creating more complex agents is integrating cognition (Sun, 2007). Instilling cognitive processes into an LLM-based agent provides it with a reasoning system to enhance human-like thinking (Sun, 2007), turning it into a *CogAgent*. If the input and output behavior of the CogAgent matches human behavior, then there is evidence to suggest that the cognitive mechanisms modeled may be a plausible explanation for human behavior (Russel & Norvig, 2020).

When implemented with task-specific prompts and inference systems, CogAgents can cooperate with one another. By leveraging the linguistic capabilities of LLMs, group level dynamics can be simulated. The design of inference systems can be guided by the objective for agent behavior. Simply enhancing logical reasoning of LLMs can be done through prompt engineering, which alone has developed some strategies for improvement through iterative trial and error. One strategy is Chain-of-Thought, which decomposes a prompt to intermediary steps (Wei et al., 2022). This approach still leads to suboptimal performance in arithmetic tasks and does not translate easily to a reasoning structure. Program-Aided-Language is a strategy that uses LLMs to translate from natural language into intermediary programming steps, with the final output being a script to provide for a code interpreter to generate the response (Gao et al., 2023). This seemingly works with straightforward tasks but is difficult to adapt to cognitive based intermediary steps. *Cognitive architectures* can define an inference system of intelligent agents, greatly improving their ability to think humanly (Kotseruba & Tsotsos, 2020). The main difference between cognitive architectures and intelligent systems, is that the latter produces a purely fixed model for general computation. CogAgents that employ cognitive architectures can dynamically adjust their contributions based on the internal cognitive mechanisms modeled.

Computer modeling gives researchers the opportunity to define the cognitive architecture used by CogAgents. The individual level processes of an agent can be defined to their liking and tracked throughout simulation by the CogAgents internal state. The selection of these mechanisms is guided by their ability to influence the contribution of a CogAgent. This includes if, and what, information is shared. The group-level processes, such as information distribution and role assignments, may be defined in tandem with the individual level processes to precisely capture the relationship between specific mechanisms. Simulation easily allows for the manipulation of these variables for large scale experimentation. The internal state of the agent cannot be measured in human-based experiments with explicit isolation from extraneous influence. This capability of explicitly tracking internal states, delineated by the cognitive architecture design, is only afforded through simulation.

Cognitive Architectures

Cognitive architectures are computational models that aim to simulate and understand human cognition. The main proponent of cognitive architectures is to create something that can accurately demonstrate reasoning, adaptation, and improvement (Kotseruba & Tsotsos, 2020). These models typically consist of a set of interconnected components, each representing a different aspect of human cognition, such as perception, attention, memory, and decision-making. Kotseruba and Tsotsos (2020) groups cognitive architectures with a particular focus on perception, attention mechanisms, action selection, learning, memory, and reasoning. The components of a cognitive architecture are designed to work together to produce intelligent human-like behavior. These categories allow for the study of the human mind and the underlying cognitive process that galvanizes behavior. By explicitly representing general human cognitive mechanisms, a strong comprehension of the mind can be obtained (Kotseruba & Tsotsos, 2020).

The design of the computational procedures defines which cognitive attributes are relevant to the application domain. In group-based discussions, a cognitive architecture would outline how an agent processes a message and the procedures performed to guide their contributions. These processes directly impact the exchange of information items and the discovery of the hidden profile. Intelligent behaviors exhibited by CogAgents can be analyzed with respect to the computational procedures that perform operations on mental representations (Kotseruba & Tsotsos, 2020). If the deployment of a cognitive process makes an impact on CogAgent behavior in a similar fashion to humans, it is indicative of their presence in humans. Simulating human-like thinking is central to investigating the hidden profile phenomenon through simulation, as CogAgents should not act omnipotently. They should, in effect, make the same mistakes as people.

CHAPTER 2

LITERATURE REVIEW

Hidden Profiles: Theoretical Foundations and Empirical Studies

Initial research into hidden profiles began with Stasser's challenging of the idea that groups make more informed decisions. Four-member decision-making groups were given information regarding a hypothetical student body president election and were requested to discuss the candidates before making a decision (Stasser & Titus, 1985). The results of the study indicated that even though the groups had access to the information highlighting the better candidate, the superior candidate was not chosen. This is because their initial distribution biased their initial decision alternative preference, suggesting that discussions would perpetuate rather than correct candidate perception (Stasser, & Titus, 1985). This research was among the first to indicate that unbound group discussion may not be the optimal means of disseminating information.

Stasser's (1988) findings were further extrapolated using the DISCUSS model, which emphasizes information flow throughout discussion and identifies the product of information exchange as the evolution of opinion. The focus is on a discussion oriented single decision task, not accounting for temporal influences. The input parameters include valence variability, group size, participation rate, advocacy, and discussion norms (Stasser, 1988). The advocacy parameter influences the extent to which personal preference biases contributions, while the norm defines the rate of whether discussion resolves differences and conveys information. The first phase, prediscussion, specifies the distribution of items. This distinguishes between shared and unshared information items, defining the hidden profile. The second phase is where discussion takes place in a series of steps where members exchange information based on a recall probability and advocacy. The recall probability employs a ratio of information load and number of recalled items to bias memory. Stasser (1988) assumes members recall and contribute information that supports their current preference. The output of the model is the initial and final distribution of preferences, the group decision, recalled information, and discussion content.

The DISCUSS model departed from the existing models at the time, such as DICE (Penrod & Hastie, 1980), JUS (Hastie et. al, 1983), and SIS (Stasser & Davis, 1981), as they did not explicitly define the revision of preferences in the presence of new information (Stasser, 1988). These models represent group decision making through a series of opinion changes that may lead to a decision or statement depending on convergence. The input consists of the distribution of initial preferences, and the output is the predicted outcome of the group decision. The change in preferences is not galvanized by the information items available, rather, they deploy a strength-in-numbers approach. The probability of preference change is based on the number of advocates, focusing on social influence. The operationalization of this influence varies between the models, with DICE and JUS using a persuasibility parameter and SIS through certainty states (Stasser, 1988). By investigating this information exchange in group discussion, the manipulation of information levels is isolated in the DISCUSS model. The model initially assumed that member participation rates were significant, which was later disproved when manipulating the variable did not affect the simulation (Stasser, 1988). The DISCUSS model produced results that indicate that groups fail to discover the hidden profile, regardless of if the discussion contributions are biased based on advocacy or minority status (Stasser, 1988).

Stasser et al. (2000) has conducted further research into yet another potential factor: expert-role assignments. The results demonstrated that export roles may encourage retention and encoding of information items. This is because experts increased the prevalence of unshared information, while forewarning of expert roles heightened the probability of retaining that information (Stasser et al., 2000). This research spurred the emerging view of conceptualizing groups as information processors. Researchers considered factors including processing objectives, attention, encoding, storage, retrieval, processing, response, feedback, and learning in small group settings (Hinsz et al., 1997). These factors are designed to encompass the dimensions of variability in group discussions performing cognitive tasks: commonality-uniqueness of information, convergence-diversity of ideas, accentuation-attenuation of cognitive processes, and belongingness-distinctiveness of members (Hinsz et al., 1997).

Group and Individual Behavior Models

The temporal taxonomy by Marks et al. (2001) expands group modeling by focusing on the general team effectiveness of teams completing a wide range of tasks. It differs from the DISCUSS model as it utilizes recurrent input-process-output episodes, rather than two phases for pre-discussion and discussion for a single task accomplishment period (Stasser, 1988), splitting a task into multiple episodes. Processes are viewed as multidimensional, composed of a two-tier hierarchical structure, organizing 10 processes into 3 higher level categories–action, transition, and interpersonal. By differentiating between task types, different process periods can be correlated with each task (Marks et al., 2001). These tasks include cognitive, verbal, and behavioral activities as a part of the task work. These distinctions are very significant as information processing in groups is sensitive to context; task characteristics can dictate processing goals (Hinsz et al., 1997). The task characteristics of group-based decision making indicate the types of cognitive processes that impact information exchange. With temporal consideration, and a broader task set, the taxonomy allows for a cyclical process that influences itself (Marks et al., 2001). This approach is more easily generalized to real world scenarios, as teams typically manage multiple activities over time. Marks' temporal taxonomy introduces a component of social interaction: emergent states of whole teams, which delineate cognitive, motivational, and affective states. Emergent states explain how these aspects may influence interaction between team members, rather than focusing on the processes of interaction. They are also updated due to interaction outcomes, introducing a feedback concept (Marks et al., 2001). These emergent states explain the interaction between group level and individual level processes and can be studied to link states with which information items are shared, and when they are shared.

Marks et al. (2001) discusses how emergent states influence the group, while Hinsz et al. (1997) establish that individual non-interaction related factors can also influence group interaction. This offers an insight that Stasser and Titus's (1985) original study does not, through explaining factors that influence individuals which in turn influence the group. The method of study also varies, as Stasser and Titus (1985) utilized discussion groups, while Hinsz et al. (1997) employed ad hoc, or lab, settings. Member roles are another individual related consideration that affects group processing objectives (Hinsz et al., 1997). Stasser et al. (2000) does highlight this, as expert-role assignments were included in the study. The group information processing framework outlined by Hinsz et al. (1997) functions as a bridge between research focused on individuals and research focused on groups. It adds more dimension to processes and provides explanations for more realistic scenarios where multitasking is necessary. By expanding processes, components are more clearly visible. This makes it easier to correlate specific processes with information exchange variability. In isolating the influence of attention on the processing objective, researchers may explain, or predict, the recall of hidden profiles. A small enough minority group may result in their information being perceived as an opinion, or they may be discouraged from contributing altogether (Hinsz et al., 1997). The additional temporal variable, also seen in the Marks et al. (2001) temporal taxonomy, captures authentic business environments, where teams operate on multiple tasks in pursuit of an extended goal and maintain a relationship history.

On the other end of the spectrum, focusing more deeply on the individual, Martinez-Miranda and Pavon (2009) introduce a more personal concept into their TEAKS model: trust. Trust is a significant aspect of human relationships, specifically teams. The trust relationship between team members influences both individual and team performance. The feedback concept presented in the work by Hinsz et al. (1997) is also apparent in this model, as trust increases when an outcome meets an expectation. Elevated trust results in a reduced sense of risk in future interactions and a positive expectation of future interactions (Martinez-Miranda & Pavon, 2009). The impact of trust on this sense of risk and positive expectation may influence whether individuals choose to contribute information, especially novel items. Another similarity is in considering task characteristics as influencing factors. The temporal aspect of TEAKS maintains a relationship history in order to manage trust updates (Martinez-Miranda & Pavon, 2009). This differs from previous models, which were more process oriented. The DISCUSS model (Stasser, 1988), as well as the research by Hinsz et al. (1997), emphasizes information exchange. The TEAKS model shares similarities with the DISCUSS model in relation to modeling team members as agents in a virtual environment. The model expands agent representation by including variables for emotional states, social characteristics, cognitive abilities, and personality types. The internal state of an agent is a combination of these variable values, and the state is also influenced by interaction with other agents' states and task characteristics (Martinez-Miranda & Pavon, 2009). The method parameters vary between the DISCUSS and TEAKS models. The TEAKS model utilizes fuzzy values in assessing parameters of an agent's internal state. This is done in an attempt to capture the randomness of the non-deterministic nature of human emotion (Martinez-Miranda & Pavon, 2009). The results demonstrate that other-person controlled emotions influenced trust more than personal control emotions; anger decreased trust, while gratitude increased trust.

In summary, the DISCUSS model (Stasser, 1988) was group and information exchange oriented, with the TEAKS (Martinez-Miranda & Pavon, 2009) model expanding into collections of individuals with personalized elements. The DISCUSS model focuses on how the exchange of information is influenced by group size, participation, advocacy, and discussion norms. The evolution of group preference is illustrated using the presence of information items, giving insight into how the influencing parameters may impact the uncovering of the hidden profile. TEAKS leans more into a specific influencing social factor, namely trust, and how it is influenced by emotion. The temporal taxonomy (Marks et al., 2001), and Hinsz et al. (1997) discussion on groups as information processors provide more group level factors for consideration. The expansion of task type, and accounting for temporal influences, create even more potential factors to investigate their interrelationships. In conjunction, these frameworks expound the factors that impact information exchange, including situational, individual, and environmental attributes.

The models discussed were all designed in a time where the capabilities of technology were not apt enough to pass a Turing test. The agents in DISCUSS and TEAKS are not designed with the ability to dynamically alter their processes due to incoming context, rather, they are statically defined. Moreso, even the most current LLMs may not be successfully deployed into the DISCUSS or TEAKS model. Their lack of cognition also inhibits their use in simulating the temporal taxonomy (Marks et al., 2001) or framework proposed by Hinsz et al. (1997). There would not be any points of analysis for individual level processes. However, their use in modeling group team members provides a level of realism in generating human like discussion of information items.

With more natural commentary, LLM-based agents allow for context aware analysis of group level processes. To capture individual level processes, however, the task of implementing a reasoning system remains.

Cognitive Architectures

Cognitive architectures are a pivotal framework in understanding and emulating human cognitive processes. These frameworks draw together insights from cognitive science, artificial intelligence, computational neuroscience, cognitive robotics, and computational cognitive systems (Lieto et al., 2017). The architectures aim to capture invariant cognitive processes such as reasoning, learning, perception, memory, and action execution (Oltramari & Lebiere, 2012). The cognitive architecture, as a system, demonstrates human-like abilities and deficiencies in cognition, reasoning, perception, memory, and attention (Russel & Norvig, 2020). Human performance modeling is dominated by a small number of specialized architectures (Kotseruba & Tsotsos, 2020).

Cognitive architectures can be tailored to specific scientific objectives and task types. Examples include SOAR (Laird, 2012), ACT-R (Anderson et al., 2004), CLARION (Sun, 2006), and iCub (Vernon et. al, 2007). SOAR aims to model general cognitive processes such as reasoning, learning, and decision-making. ACT-R emphasizes cognitive processes involved in perception, memory, and problem-solving. CLARION integrates symbolic and connectionist approaches to cognitive modeling. Icub is specifically designed for cognitive robotics applications. These architectures differ in their approaches, ranging from cognition-centric designs aimed at capturing general cognitive processes to application-driven designs focused on fulfilling specific requirements. Cognitive architectures are often categorized based on their design perspectives, with some prioritizing adhering to cognitive theories while others emphasize functional attributes and user needs (Vernon, 2017). Some aim to model cognition comprehensively, focusing on generality, completeness, and the standardization of mind models (Laird et. al, 2017). These types of models utilize a ‘cognition in the loop’ approach, simulating cognitive and biological processes to refine theories about intelligent behavior (Cordeschi, 2002). Other models prioritize application value, employing *system architectures*. Like cognitive architectures, system architectures exhibit cognitive abilities such as perception, action anticipation, learning, and adaptation (Vernon, 2017). However, their design is based on application and user requirements, drawing from algorithms and data structures (Vernon, 2017).

Symbolic architectures align closely to human-like thought processes, being a natural representation of knowledge. They are common in planning tasks, which is effective in a team-based setting due to its recursive decomposition of tasks into subgoals identified in the attention module (Kotseruba & Tsotsos, 2020). A conversational agent can utilize this to model human performance, as seen in the IMPRINT architecture (Kotseruba & Tsotsos, 2020). Other types of planning may work well with pre-existing time frameworks, for example, temporal planning would work efficiently with the temporal taxonomy proposed by Marks et al. (2001). Symbolic architectures appear in architectures that are formalized with modal logic (Cohen & Levesque, 1990), such as those based on the Belief-Desire-Intention paradigm (Bratman, 1987). However, symbolic architectures face perceptual limitations insofar as the reliance on direct data (Kotseruba & Tsotsos, 2020). This limitation curbs the ability to deploy symbolic architectures in more realistic, uncontrolled, environments. Alternatively, sub-symbolic reasoning facilitates the storing of intricate sensory data such as intonation, speech rate, and loudness (Kotseruba & Tsotsos, 2020). Connectionist networks are inspired by the biological phenomenon of the human brain, with nodes acting as dendrites. Examples of these networks are deep learning architectures like neural networks, capable of processing vast amounts of data and extracting patterns that are undetectable by humans (Kotseruba & Tsotsos, 2020).

Cognitive architectures have been utilized in reasoning, learning, perception, action, execution, selective attention, and recognition tasks (Vernon, 2017). Historically, cognitive architectures have been implemented to target three main objectives (Lieto et al., 2017). First, to elucidate the fundamental mechanisms of human cognition. Second, to facilitate the development of cognitive capabilities over time. Third, to achieve human-level intelligence in artificially intelligent agents, also known as General Artificial Intelligence. This third elusive goal attempts to take on the challenge of realism. Agent success could be evaluated in terms of its likeness to human cognitive processes and behavior (Kotseruba & Tsotsos, 2020). Reflexive agents, cognitive agents, emotional agents, personality agents, and normative systems are all architectures proposed to undertake realism (Bourgais et al., 2020). These architectures are not used often in social simulation, due to a lack of implementation or restriction to domain dependency (Bourgais et al., 2020).

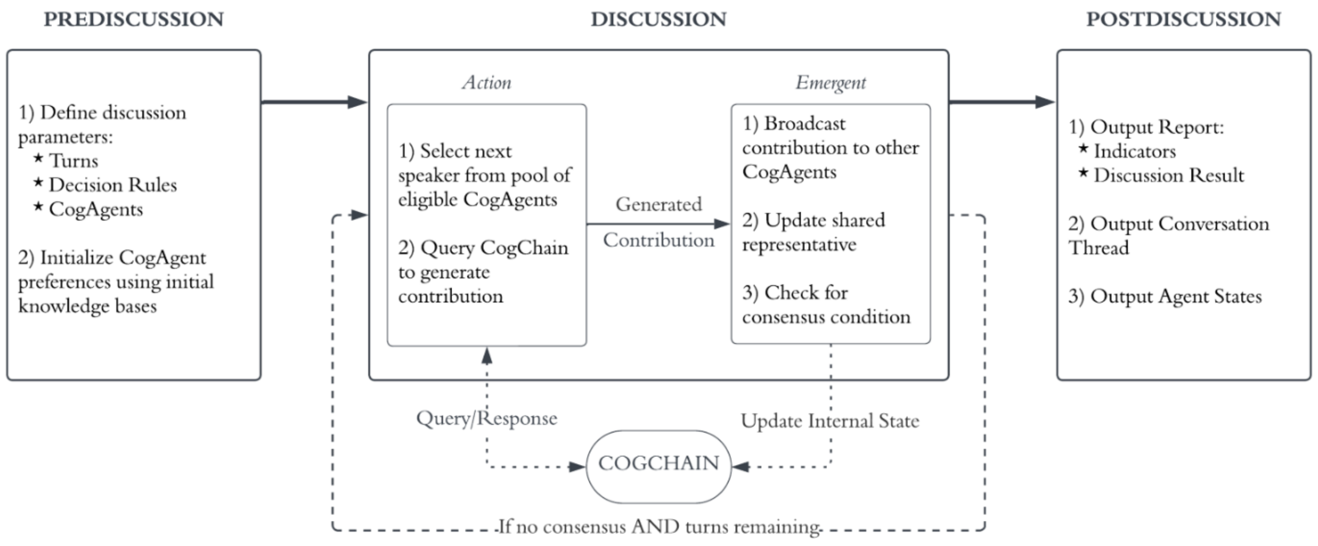
However, existing cognitive architectures face limitations. Few existing architectures have the theoretical, software, and hardware foundations to facilitate simulation of communication and social interaction (Kotseruba & Tsotsos, 2020). Moreover, existing architectures rarely have the capacity to detect the emotional states or intentions of an individual (Breazeal, 2003), or provide personalized responses (Gobet & Lane, 2012). Another limitation of cognitive architectures is that modeling human memory to a realistic magnitude is intractable (Kotseruba & Tsotsos, 2020). These limitations do not revoke the benefits of integrating a cognitive architecture with an LLM agent, rather they necessitate the development of a cognitive architecture that is designed to be implemented with the current capacity of artificial intelligence.

CHAPTER 3

DESIGN AND METHODOLOGY

CogSystem is for goal-based group decision-making simulation using LLM-based cognitive agents. It consists of CogFrame, a discussion framework, and CogChain, a cognitive architecture.

CogFrame: Discussion Framework



PREDISCUSSION
1) Define discussion parameters:
★ Turns
★ Decision Rules
★ CogAgents
2) Initialize CogAgent preferences using initial knowledge bases
DISCUSSION
Action
1) Select next speaker from pool of eligible CogAgents
2) Query CogChain to generate contribution
Generated Contribution
Emergent
1) Broadcast contribution to other CogAgents
2) Update shared representative
3) Check for consensus condition
POSTDISCUSSION
1) Output Report:
★ Indicators
★ Discussion Result
2) Output Conversation Thread
3) Output Agent States
Query/Response
Update Internal State
COGCHAIN
If no consensus AND turns remaining

Figure 1: CogFrame Diagram

Discussions are depicted as a single step turn-based series of contributions by team members. Due to the nature of the simulation of discussion, CogFrame is designed for text-based discussions. This is not a major limitation, as text is the most common input for architectures that perform inference tasks (Kotseruba & Tsotsos, 2020). CogFrame allows for two input parameters that vary the group level processes: decision rule, and the discussion task. Decision rules define the termination of conversation. The discussion task defines the task prompt, as well as the initial distribution of information items across CogAgents. This formalizes the definition of the hidden profile set and the initial distribution of CogAgent preferences. CogFrame is structured by drawing inspiration from the DISCUSS model (Stasser, 1988) and temporal taxonomy proposed by Marks et al. (2001). Similar to DISCUSS, CogFrame has 3 phases: pre-discussion, discussion, and post-discussion. The justification of this design choice is aligning with the decomposition of the discussion process for simulation, as done by Stasser's original model.

In the first phase, the distribution of information items to agents is used to determine their initial individual preferences for the group's impending decision. Items are not necessarily unique across team members. This accounts for the commonality-uniqueness dimension of variability in group level processing (Hinsz et al., 1997). In a realistic setting, it cannot be known whether a hidden profile is entirely revealed. However, in this simulation, the pre-discussion phase allows for a knowledge bank, where an additional input parameter contains the comprehensive list of information items. The next phase, discussion, embeds action-emergent episodes. The integration of episodes is seen in the temporal taxonomy by Marks et. al (2001), allowing for the correlation of processing periods to specific tasks. The first episode encompasses a discussion turn, the action phase, and the second episode captures recursive feedback in the emergent phase. The emergent phase is where the emergent state of the team and individuals is updated as a result of a contribution. Each loop of a discussion phase constitutes one discussion turn. At the beginning of the discussion, the first speaker is selected from the pool of eligible CogAgents. This agent triggers the action phase, which consists of querying the CogChain of the CogAgent to generate a contribution. This is the shell module for the cognitive architecture, which is responsible for the individual agent processes. Contributions consist of an information item and the CogAgents commentary (see example contribution below). The emergent phase in CogFrame links the framework to the CogChain cognitive architecture, representing the interaction between group and individual processing. After a contribution is generated in the action phase, the emergent phase updates CogAgents by providing the input to update the agent's internal state within CogChain.

Example Contribution:

INFORMATION ITEM: While it is widely known that Bean holds a bachelor's degree in Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a minor in Counseling Psychology.

COMMENTARY: Acknowledging the importance of Collins's linguistic skills in enhancing communication and cultural bonding within the diverse crew, it's imperative to also consider the psychological dynamics and mental well-being of the team during the extended Mars mission.

CogFrame maintains a revealed profile variable that serves as the group's shared representation. The revealed profile is updated with every contribution made, tracking which items from the knowledge base have been revealed to the group. When group members collectively develop and share representations of information, it helps to validate and strengthen the choices made. The shared representations provide a common understanding and perspective, aligning group members' thoughts and decisions. Transactive memory enhances the quality of choices and promotes consensus within the group, accounting for the accentuation-attenuation dimension of variability (Hinsz et al., 1997). It also allows for observing the changes in the amount of information revealed throughout the discussion. This is particularly useful when investigating the effects of specific CogChain variables, as a researcher can isolate the changes to view the direct impact on shared information.

Once the emergent phase is complete, the current preferences of each CogAgent are compared to determine whether the group has arrived at a consensus given the decision rule. The decision rule is either unanimous or majority. This accounts for the last dimension of variability in group level processing, convergence-diversity (Hinsz et al., 1997). If the agents have agreed, the conversation is terminated, and the successful outcome is reported. If the agents have not yet agreed, the next speaker is selected from the pool of possible participants to begin the next iteration of the discussion phase. The most recent contribution is provided as context for the next speaker. CogFrame continues to iteratively loop discussion phases until the agents have agreed, or until the maximum number of turns have elapsed. If the maximum number of turns has been reached, and the agents have not come to a decision, then the discussion result is returned as failure.

In essence, CogFrame facilitates the simulation of the group level processes that influence discussion between CogAgents. These group level processes, being the decision rule and discussion task, can be varied to investigate their influence on the individual level processes that are defined in the CogChain architecture. The initial preferences of CogAgents are determined, followed by the simulation of discussion. The discussion simulation consists of iteratively selecting the next speaker, querying their CogChain to generate a contribution. Each contribution is broadcasted to all other CogAgents for CogChain internal state updates, linking group level and individual level processes. The revealed profile is aggregated as turns progress. The consensus condition is checked after each discussion phase iteration, terminating upon group agreement. If a decision was reached, the preferred alternative is returned as the discussion result.

CogChain: Cognitive Architecture Design

While CogFrame captures the group-level influences on information exchange related processes, it still needs its counterpart CogChain. Each CogAgent has a CogChain, serving as their reasoning system. CogChain is designed with a modular approach, each component responsible for a different aspect of the overall individual processing. Generally, the CogChain parses incoming contributions by updating the internal state and generates contributions by referencing this internal state and performing cognitive computations. This defines the two main components of CogChain: Process, and Response. These components are where the individual level processes, known as CogModules, are categorized. The addition of CogModules to CogChains modify the input parameters to the model, which can be varied for investigating the relationship with the group level processes defined in CogFrame. CogChain is an unfolding model, with each iteration selecting CogModules designed to capture human communication and behavior based on the KISS principle: Keeping It Simple Stupid (Axelrod, 1997). CogChain captures the simplest CogModules necessary to model the discussion, and only expands the scope to iteratively increasing the realism of human-like thinking. Each new CogModule builds on the limitations of the previous CodModule, adding to the CogChain layer by layer. By isolating each module, the architecture is scalable, with the ability to support a range of behaviors in a transparent and interpretable manner. CogChain layers can be deployed atomically or compounded. To actuate, CogAgents employ LLMs to generate their commentary with the guidance of their cognitive architecture.

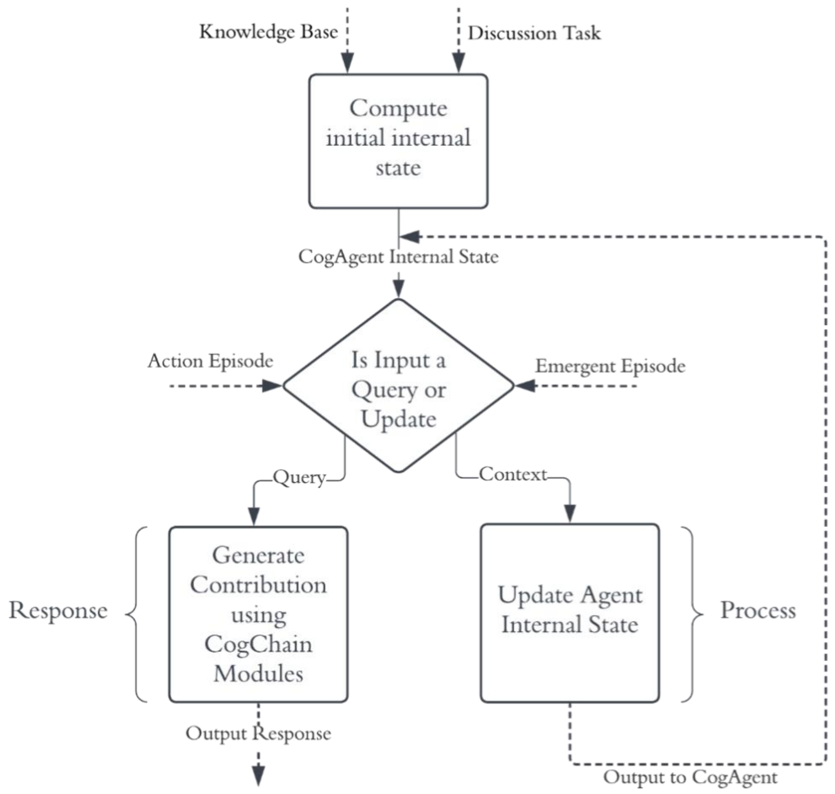


Figure 2: Generic CogChain Flowchart

BaseCog, the simplest version of CogChain, defines the CogAgents' internal state as having a decision preference and a knowledge base. This provides insight into the preference changes and expansion of CogAgent knowledge bases, allowing for investigating their connection to the final revealed profile. These parameters are defined using the group-level discussion task input, defining the initial information items that are in each CogAgents' knowledge base. CogAgents are selected to contribute at random. When an agent receives a broadcast contribution, they undergo a processing update in the first component. If the information item in the contribution is not present in the CogAgents' knowledge base, it is added. This updated knowledge base is then used to update the agent's preference (see the prompt below). The knowledge base update is Python based, while the preference update and contribution generation are LLM based.

Preference Update Prompt:

You have access to a list of facts, called a 'knowledge base', to help guide your preference. Here is the most recently updated list:

{updated knowledge base}

Based on the above facts, choose one alternative from the decision options. You must choose from the provided options only.

{decision options}

Use this format for your output: PREFERENCE: (Insert candidate name) EXPLANATION: (justify your decision, limit to 300 tokens).

Upon an agent's turn, given the most recent prior contribution, they reference their decision preference to choose an information item from the knowledge base and generate commentary (see prompt below). The biasing of this retrieval impacts the information items revealed from the hidden profile (Stasser, 1988). The knowledge base of the agent serves as their memory, dynamically updating in the process component. The ability for agents to have, and change, their preference delineates the motivational process affecting the attentional mechanism of the agent. Attentional processes are significant in a discussion context, as the distribution of information across group members guides the focus of attention. BaseCog is limited insofar that the CogAgents correctly generate commentary that align with their goal, but do not interact with one another. Even with the given context, CogAgents need to be ushered to respond to their teammates.

Contribution Prompt:

Your team mate just contributed: {context}.

It is your turn in the discussion. Choose a fact from the knowledge base and provide it (exactly as phrased) alongside your commentary. You are likely to choose a fact that aligns with *your preference. You may also directly address the team member who just contributed, if you want to do so.*

Preference: {CogAgent current preference}

Knowledge Base: {CogAgent knowledge base}

Output format: INFORMATION ITEM: (insert fact exactly as given) COMMENTARY: (insert your commentary)

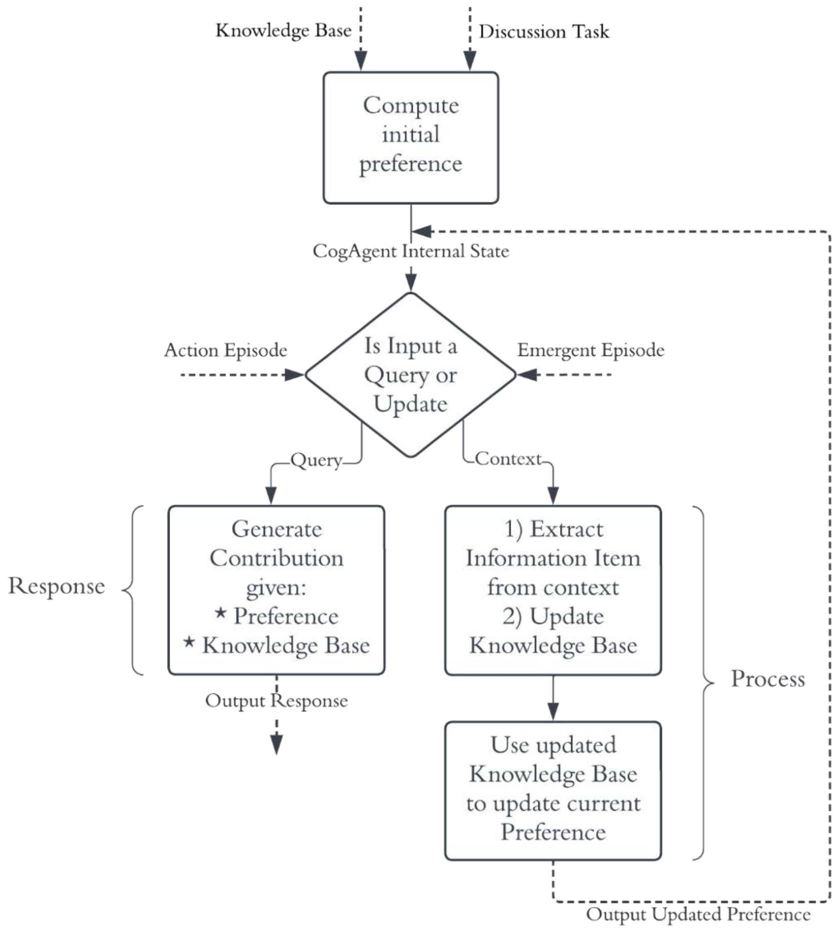

Knowledge Base
Discussion Task
Compute initial preference
CogAgent Internal State
Action Episode
Is Input a Query or Update
Emergent Episode
Query
Context
Response
Generate Contribution given:
* Preference
* Knowledge Base
1) Extract Information Item from context
2) Update Knowledge Base
Process
Output Response
Use updated Knowledge Base to update current Preference
Output Updated Preference

Figure 3: BaseCog Flowchart

To address this potential pitfall, interaction is encouraged as the next CogModule introduced to the chain. The encouragement of interaction should vary the information items selected for sharing by forcing contextually relevant bias, increasing the likelihood of exchanging a wider array of information. This can be demonstrated if the resulting revealed profile is larger than those produced by BaseCog simulations. InteractionCog utilizes the same architecture structure as BaseCog, with the addition of directly requesting agents to respond to the incoming context from their teammate in the contribution prompt. No additional input parameters are included. The justification behind this Cog is to facilitate more human-like interactive behavior. The flowchart for InteractionCog is almost identical to BaseCog's, with the addition of the interaction guide in the response component. The selection of information items is from the mental representation of the agent's memory, which is operationalized as the knowledge base. The knowledge base is stored as a dictionary, but information items are presented to the CogAgent as a list of strings when LLM prompting is used (see example knowledge base below). This operationalization does not mimic a human-like process of memory storage and access. The retrieval of information from long-term memory to short term memory is based on bias (Kotseruba & Tsotsos, 2020), which is not captured by InteractionCog.

Example Knowledge Base:

Knowledge Base

Collins has a passion for languages. He is fluent in English, Spanish, and French and proficient in Russian. Recently he started a class on Mandarin and already he is nearly proficient.

Astronaut Anders received the highest possible scores on the military survival standards, an assessment that all astronauts must complete in order to qualify for space missions.

While it is widely known that Bean holds a bachelor's degree in Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a minor in Counseling Psychology.

.

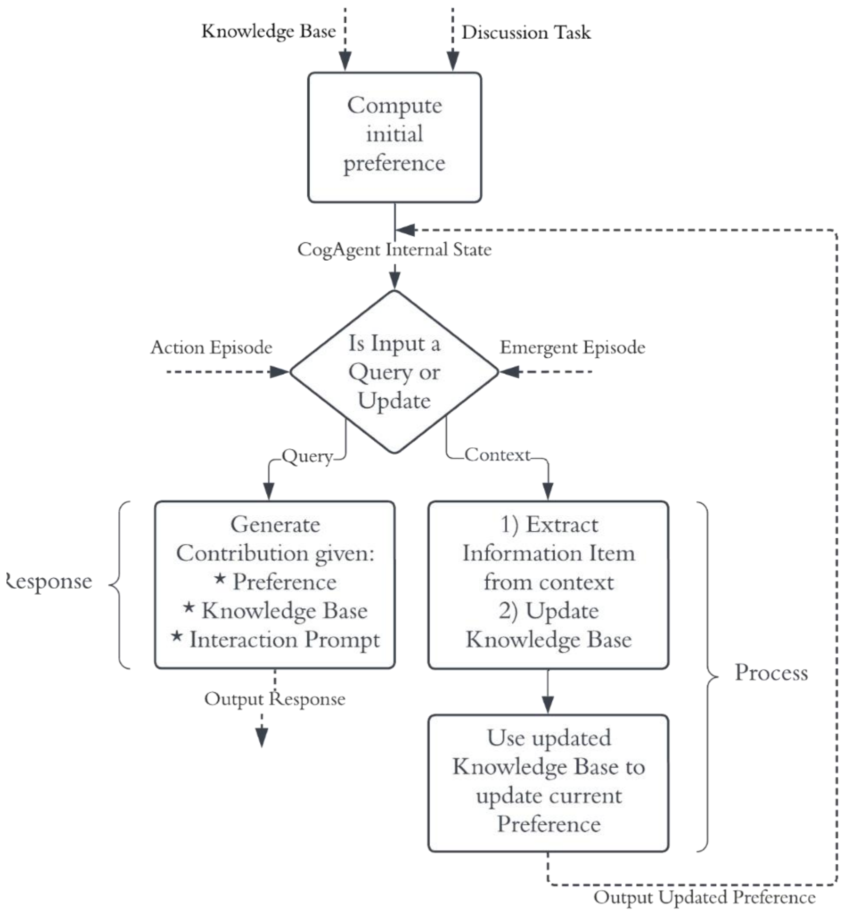

Knowledge Base
Discussion Task
Compute initial preference
CogAgent Internal State
Action Episode
Is Input a Query or Update
Emergent Episode
Query
Context
Response
Generate Contribution given:
* Preference
* Knowledge Base
* Interaction Prompt
1) Extract Information Item from context
2) Update Knowledge Base
Process
Output Response
Use updated Knowledge Base to update current Preference
Output Updated Preference

Figure 4: InteractionCog Flowchart

MemoryCog adds a new cognitive module to the respond component of CogChain. The knowledge base is cast as the long-term memory, while a *relevant information item set* (RIS) is generated to simulate working memory. Working memory is critical when considering the cognitive capabilities modeled in the attention and decision-making mechanisms (Kotseruba & Tsotsos, 2020). This is drawn from the multi-store memory model, which was influenced by the Atkinson-Shiffrin model (1968). Working memory and long-term memory are typical for planning tasks, in which the latter retains the factual information, and the former contains the information relevant to the current world model and goal stack (Kotseruba & Tsotsos, 2020). The RIS is used for selection when generating contributions to bias the attention to retrieve information items from the knowledge base that pertain to the CogAgents' preference. This is similar to Stasser's (1988) assumption that agents recall items that support their preference. The RIS is generated as a randomized list of these information items and is presented to the LLM as a list of strings in the contribution prompt. The knowledge base is also randomized each time it is accessed to ensure no bias is introduced from the order. The impact of including memory should increase the amount of information items shared and the variance. MemoryCog, like InteractionCog, does not introduce any additional input parameters to the architecture beyond those provided in BaseCog by the group-level discussion task.

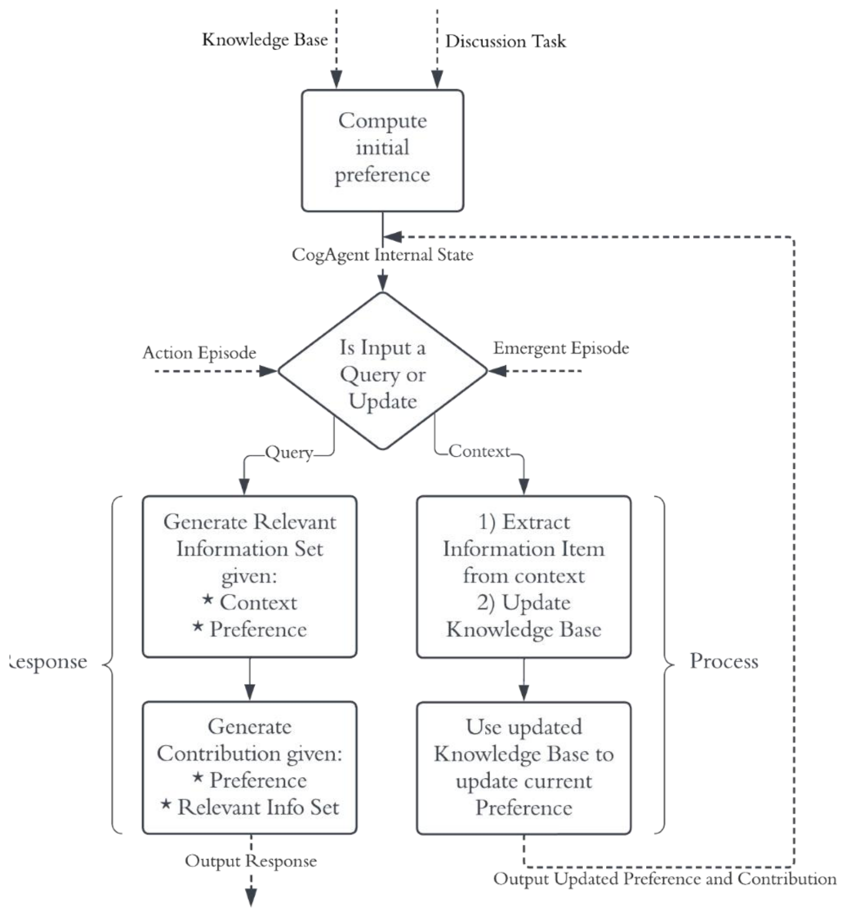

Knowledge Base
Discussion Task
Compute initial preference
CogAgent Internal State
Action Episode
Is Input a Query or Update
Emergent Episode
Query
Context
Generate Relevant Information Set given:
* Context
* Preference
1) Extract Information Item from context
2) Update Knowledge Base
Response
Process
Generate Contribution given:
* Preference
* Relevant Info Set
Use updated Knowledge Base to update current Preference
Output Response
Output Updated Preference and Contribution

Figure 5: MemoryCog Flowchart

Typically, team members do not equally participate in discussion unless there are strong norms or procedural constraints that enforce equal contribution (Stasser, 1988). ContributeCog allows agents to choose whether or not they want to contribute, modeling a motivational internal process that affects a group-level process. The impact of this addition can be investigated in the resulting distribution of contributing agents. This internal level process does not require any input to the model beyond those introduced by BaseCog. CogAgents are instructed to reference their current preferred decision in updating their contribute variable as a part of the process component, relying on the capability of LLM natural language processing (see contribute tendency prompt below). This 'contribute tendency' variable, added to the CogAgents' internal state, then constitutes the pool of possible contributors to select for the next CogAgent speaker.

Contribute Query Prompt:

Your teammate just contributed: {context}

Determine whether you would like to respond to this contribution based on your preference.

Preference: {preference}

Use this format for your output: CONTRIBUTE: (insert YES/NO) \n EXPLANATION: (explain why you want to respond to your teammate (max 150 tokens)

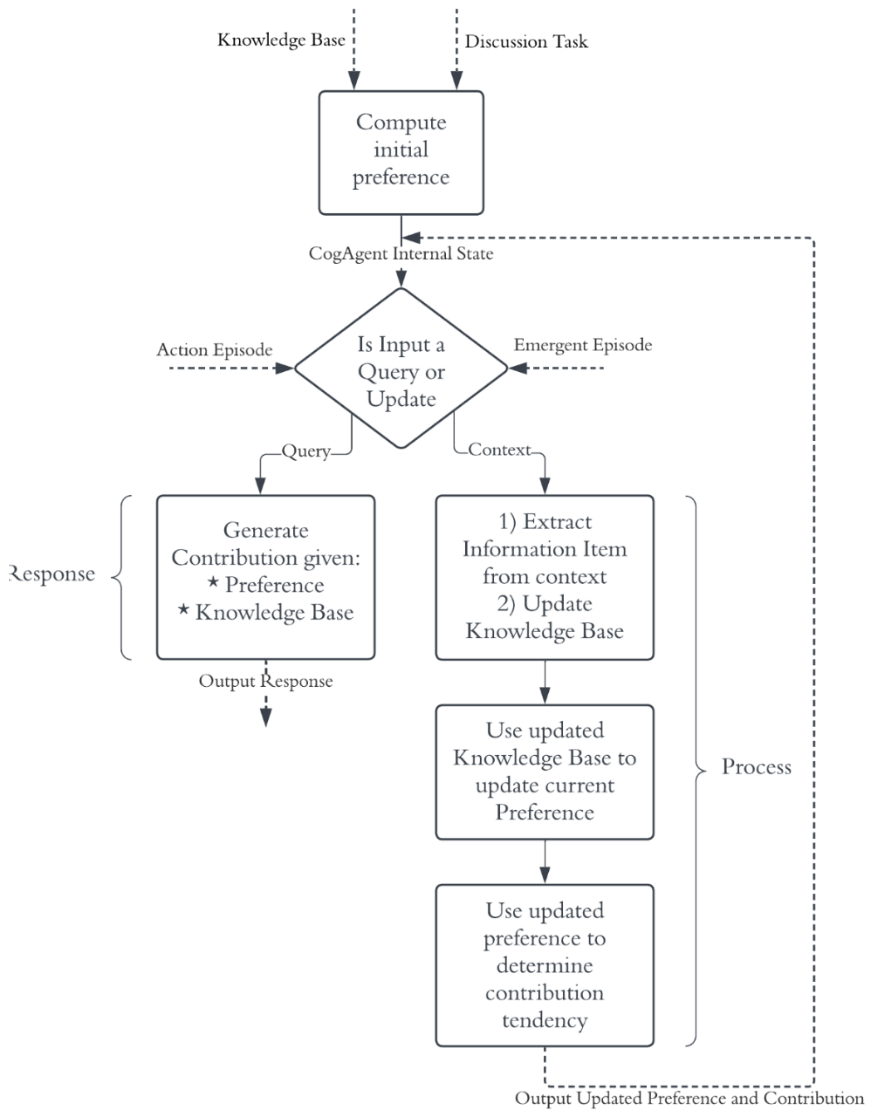

Knowledge Base
Discussion Task
Compute initial preference
CogAgent Internal State
Action Episode
Is Input a Query or Update
Emergent Episode
Query
Context
Response
Generate Contribution given:
⋆ Preference
⋆ Knowledge Base
Output Response
1) Extract Information Item from context
2) Update Knowledge Base
Use updated Knowledge Base to update current Preference
Process
Use updated preference to determine contribution tendency
Output Updated Preference and Contribution

Figure 6: ContributeCog Flowchart

TrustCog predisposes a CogAgents' integration of new information items, introducing a mutable variable to the internal state—trust tendency. The input parameter is static, defined upon CogAgent instantiation as 'low', or 'high'. This trust tendency is referenced during the processing of incoming context. CogAgents with high trust tendencies will always update their knowledge base with a new information item. Low trust agents have a fifty percent probability of integrating new information into their knowledge bases. The addition of this CogModule influences the distribution of information items, as CogAgents with lower trust tendencies will have a bias to their existing information items. Even if the hidden profile is revealed to the group, distrusting CogAgents may still choose a suboptimal alternative due to their lack of trust in the truth of the information items contributed by others (Martinez-Miranda & Pavon, 2009). TrustCog is the only CogChain that allows for the variation of an individual level process in experimentation.

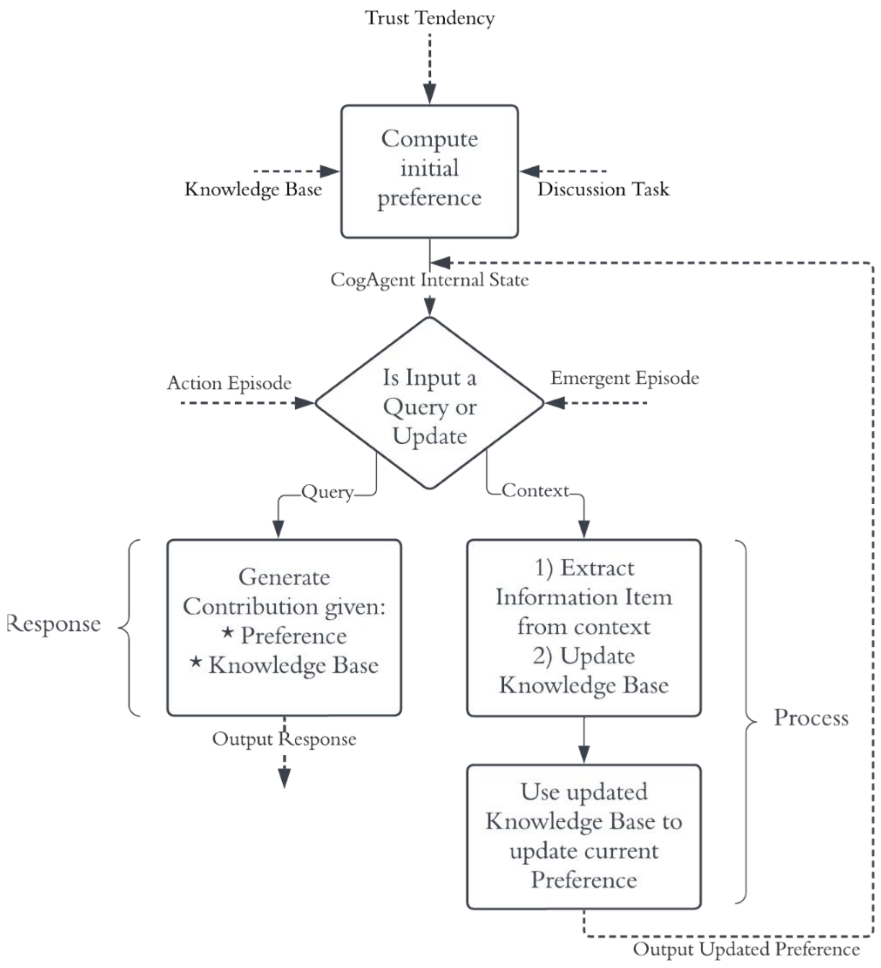

Trust Tendency
Compute initial preference
Knowledge Base
Discussion Task
CogAgent Internal State
Is Input a Query or Update
Action Episode
Emergent Episode
Query
Context
Response
Generate Contribution given:
⋆ Preference
⋆ Knowledge Base
1) Extract Information Item from context
2) Update Knowledge Base
Process
Output Response
Use updated Knowledge Base to update current Preference
Output Updated Preference

Figure 7: TrustCog Flowchart

CogFrame maintains the discussion simulation by storing the conversation history and knowledge bank. CogChains track internal state parameters that help identify important results for discussion analysis, including CogAgent information profiles and preferences. At the end of discussion simulation, three files are returned by the CogSystem: conversation thread, agent states, and the discussion report. The report uses the internal states from CogChain, and the conversation in CogFrame, to generate a summary of the discussion's *indicators*.

The information items in CogAgents' knowledge bank are initially determined through the discussion task. As each discussion turn passes, their knowledge bank may grow with incoming context that contains a potentially novel information item. At each new loop of the discussion phase, after message processing, a snapshot of the CogAgent's information profile is taken. The initial, turns, and final information profiles of each CogAgent are provided as *information distribution* indicators. Similarly, the revealed profile is returned after each discussion phase as the *group information distribution* indicator. Preferences are determined prior to the start of discussion on the basis of the CogAgents' knowledge bases. This is noted as the CogAgents' initial preference distribution. Similar to the information distribution indicator, the preferences of each CogAgent are tracked at each turn as a result of the process update. The initial, turns, and final preferences are provided per CogAgent as *preference distribution* indicators. The frequency of CogAgent contributions, as well as the distribution of the information items that they chose to share, are returned as *agent speaker frequency* and *agent information distribution* indicators. Since CogAgents' knowledge bases are mutable, the percentage of contributions that contain information items outside of their initial knowledge bases are tracked as the agent *reshare tendency* indicator. The *discussion result* indicator notes whether the group reached consensus, alongside the summarized conversation parameters (e.g number of turns, decision rule, etc.). If the agents did agree, the number of rounds it took to reach the decision is also returned. Finally, the percentage of the information items from the knowledge bank that were revealed in discussion are given as the *revealed hidden profile* indicator.

Model Development

Implementation occurs in LangChain and Python. LangChain is an open-source framework that simplifies the use of LLMs in model development by providing a Python library to integrate LLM models, APIs, and code. Templates are provided to standardize LLM prompts across agents, with the inputs to the templates varying across the CogModules implemented in the CogChains. These LLM calls are 'chained' together to form a sequence in an order defined by the CogChain. This approach to cognition, namely Chain-of-Thought, can be structured with python-based processing modules to enhance LLM-based performance (see chapter 1.b). Within these chains, Program-Aided-Language is employed by utilizing the capabilities of LLM's to extract variables from natural language prompts to provide as inputs to the python-based modules. The LLM model used in development, for fine-tuning prompts and experimentation, is OpenAI's GPT-4-0125-preview. Python is used for developing the CogSystem scheme. CogSystem is the main directory that contains the class files that implement the agents, discussion framework, cognitive architecture, and facilitate discussion simulation by instantiating conversations and tracking measurable variables for output analysis. A new CogSystem is created for each variation of CogChain, but they all contain the same five base files: CogAgent, CogChain, Conversation, Discuss, and Report. The functions implemented, input parameters defined, and sequence of chains, vary across these files according to the requirements of the CogChain specifications.

For BaseCog, the CogAgent class consists of variables maintaining the agent's name, initial knowledge base, current preference, the number of contributions they have made, and list aggregating their internal state changes. The CogChain class takes an instance of CogAgent, and defines the functions for updating preference, generating a contribution, and processing a message. The process function takes a generated contribution as the input and updates the knowledge base and preference of the agent. The knowledge base update and message processing sequence are python based, while the preference update and contribution generation are LLM-based. The Conversation class contains the variables that define the structure of the discussion, including the task, maximum number of rounds, decision rule, available decision options, the revealed profile, the knowledge bank, and the set of CogAgents participating. It also contains an instance of the Report class for use at the end of discussion simulation. The function that simulates the discussion is included in the Conversation class, capturing the phases of CogFrame. Functions are also defined for the specific episodes in the discussion phase, such as broadcasting the message, updating the revealed profile, checking the consensus condition, and selecting the next speaker. The Report class defines the functions necessary to summarize the discussion indicators and is generated at the end of the conversation simulation. Finally, the Discuss class ties the files together, instantiating the information items, CogAgents and their CogChains, the discussion task, maximum number of rounds, and the decision rule. These data points are used to instantiate the conversation. When the Discuss file is run, the CogSystem comes together to simulate the discussion.

InteractionCog only differs from BaseCog in the contribution generating function. MemoryCog adds an additional function to the CogChain class that generates the relevant information set. The contribution generating function is edited to use this relevant information set, instead of the CogAgents' knowledge base. ContributeCog adds an additional variable to the internal state in the CogAgent class, and the processing update function in CogChain is modified to update this variable. The internal state is referenced in the function that selects the next speaker in the Conversation class. TrustCog adds the internal state variable 'trust tendency' in CogAgent, and the corresponding function in CogChain that processes the incoming message is modified to use this variable to bias integration.

Experimental Setup

The discussion task chosen for experimentation is the Fast Five hidden profile. A team of four agents, CMR, MS1, MS2, and FE, are tasked with choosing between 3 alternatives. The discussion task explains to the agents that they are a part NASA's Astronaut Crew Composition team that is choosing a candidate to serve as a flight engineer for a Mars mission. The three candidates each have their own set of information items. The best candidate, Samuel Anders, has six positive, four negative, and four neutral information items. The average candidate, John Bean, has three positive, three negative, and six neutral information items. Scott Collins, the worst candidate, has four positive, six negative, and four neutral information items. The knowledge bank encompasses forty total information items. The distribution of the items across the four agents is predefined (see Table 1). This data mirrors the set used in Stasser and Titus' (1985) original study about a hypothetical student body election with three candidates.

Table 1: Distribution of Fast Five Information Items

		CMR	**MS1**	**MS2**	**FE**
Collins	Good	G1, G2, G3, G4	G1	G1, G2, G3, G4	G3, G4
	Bad	B4	B3, B4, B5, B6	B1	B1, B2, B6
	Neutral	N1, N2	N3, N4	N1, N3	N2, N3
Bean	Good	G1, G2, G3	G1, G2, G3	G1, G2, G3	G1, G2, G3
	Bad	B2, B3	B1, B2, B3	B1, B2	B1
	Neutral	N1, N2	N3, N4	N1, N3	N2, N3, N5, N6
Anders	Good	G2, G6	G1, G2, G3, G4	G3, G4	G5
	Bad	B1, B2, B3, B4	B1	B1, B2, B3, B4	B1, B2, B3, B4
	Neutral	N1, N2	N1, N2, N3	N3, N4	N2, N3, N4

Investigating the efficacy of CogSystem lies in the analysis of the input and output parameters of the CogFrame and CogChains with respect to the points of analysis. The discussion indicators returned by the end of discussion report are synthesized to return the analysis report. The first analysis point in the report returns the percentage of the knowledge bank that was revealed in discussion. The second consists of the percentage of CogAgents that deviated between their initial and final preferences. The third is the distribution of CogAgent contributions. If the team reached a decision, the optimality of their preference can be analyzed with two factors: the revealed profile, and the knowledge bank. A preference is considered optimal if it has the highest number of positive information items, average if it has the most neutral items, and the worst option if it has the most negative items. If the number of positive items is equal, then the candidates are considered average. If there are no negative or neutral items presented, the rank is determined with the worst option having the least positive items. If the agents did not come to a consensus, the individual final preferences of the agents are analyzed using the same two factors. The mutable group level processes in CogFrame are the information distribution and the decision rule. The information distribution is defined by the discussion task (see table 1). The decision rules, unanimous and majority, provide the other condition. The discussion length is fixed at a maximum of 10 turns due to limitations in cost and time. The combination of these testing conditions provides the group processes experiment configurations (see table 2).

Table 2: CogFrame Experiment Configurations

Configuration Label	**Discussion Length**	**Decision Rule**
GE1	10	Unanimous
GE2	10	Majority

Each CogChain is unit tested and the resulting analysis report is compared to BaseCog, which serves as the baseline of CogAgent behavior. BaseCog, InteractionCog, MemoryCog, and ContributeCog are unit tested by running ten trials under the group experiment CogFrame configurations *GE1* and *GE2* (see table 3) to investigate the relationship between group level and individual level processes. TrustCog requires additional configurations for unit testing due to the additional mutable trust tendency parameter. Three trust conditions are considered, all agents having a high trust tendency, all agents having a low trust tendency, and splitting the agents into half high and half low trust tendencies. Each trust condition is tested across *GE1* and *GE2* for TrustCog's unit testing (see table 3). A 95% confidence interval of the revealed profile percentage and preference change percentage from these trials are calculated. The percentage of trials that resulted in a decision are returned with their optimality rating percentages. The number of rounds taken to reach the decision is also returned using a 95% confidence interval. The percentage of trials that do not arrive at a consensus are returned with the distribution percentages of individual final preference optimality ratings. The contribution frequency of agents is also returned as a 95% confidence interval.

Table 3: CogChain Experiment Configurations

Configuration Label	**CogChain**	**Discussion Length**	**Decision Rule**	**Trust Tendency**
BGE1	BaseCog	10	Unanimous	-
BGE2	BaseCog	10	Majority	-
IGE1	InteractionCog	10	Unanimous	-
IGE2	InteractionCog	10	Majority	-
MGE1	MemoryCog	10	Unanimous	-
MGE2	MemoryCog	10	Majority	-
CGE1	ContributeCog	10	Unanimous	-
CGE2	ContributeCog	10	Majority	-
T1GE1	TrustCog	10	Unanimous	All High
T1GE2	TrustCog	10	Majority	All High
T2GE1	TrustCog	10	Unanimous	All Low
T2GE2	TrustCog	10	Majority	All Low
T3GE1	TrustCog	10	Unanimous	Half/Half
T3GE2	TrustCog	10	Majority	Half/Half

For each configuration in the above table, 10 trials are run, totaling 140. Then, compound chains are tested by combining the CogChains and running each combination for 10 trials across the group level and trust condition configurations, totaling 60 trials (see table 4). This results in

200 trials between the unit test and compound chain experiments. The amount of trials is limited by cost and time restrictions.

Table 4: Compound Chain Experiment Configurations

Configuration Label	**Discussion Length**	**Decision Rule**	**Trust Condition**
ICMT1_GE1	10	Unanimous	All High
ICMT1_GE2	10	Majority	All High
ICMT2_GE1	10	Unanimous	All Low
ICMT2_GE2	10	Majority	All Low
ICMT3_GE1	10	Unanimous	Half/Half
ICMT3_GE2	10	Majority	Half/Half

CHAPTER 4

RESULTS AND ANALYSIS

Table 5: BaseCog Experiment Analysis

	BGE1 (%)	BGE2 (%)
Revealed Profile	11.50 ± 1.24	11.25 ± 0.78
Preference Change	0	0
Agent Contribution Distribution	CMR: 21 ± 5.85 MS1: 20 ± 5.54 MS2: 30 ± 8.32 FE: 29 ± 6.47	CMR: 25 ± 6.93 MS1: 23 ± 6.23 MS2: 26 ± 5.68 FE: 26 ± 7.937
Decision Made	0	0
Agent Optimality: Revealed Profile	Best: 25 Average: 65 Worst: 10	Best: 25 Average: 62.5 Worst: 12.5
Agent Optimality: Knowledge Bank	Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

The BaseCog experiment results indicate that agents never deviated from their initial preference (see table 5). These initial preferences did not constitute a consensus, and as such a group decision was never reached, regardless of the decision rule. The analysis of final individual preferences demonstrates that the CogAgents most often select the average preference according to the revealed profile, and the worst preference with respect to the entire knowledge bank. For both decision rule conditions, the revealed profile percentage did not exceed 12% (approximately 5 information items out of 40).

Table 6: InteractionCog Experiment Analysis

	IGE1 (%)	IGE2 (%)
Revealed Profile	11.75 ± 0.71	10.25 ± 1.287
Preference Change	0	0

Agent Contribution Distribution	CMR: 24 ± 4.958 MS1: 27 ± 3.969 MS2: 31 ± 4.339 FE: 18 ± 3.719	CMR: 22 ± 4.638 MS1: 30 ± 4.801 MS2: 26 ± 4.111 FE: 22 ± 6.073
Decision Made	0	0
Agent Optimality: Revealed Profile	Best: 40 Average: 55 Worst: 5	Best: 15 Average: 77.5 Worst: 7.5
Agent Optimality: Knowledge Bank	Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

The CogAgents in the InteractionCog experiments also never reached a consensus. The percentage of the revealed profile in the InteractionCog experiments does not differ substantially from the BaseCog experiments. However, the optimality of the CogAgents' final individual preference represents a significant reduction in the selection of the worst preference with respect to the revealed profile. This can be attributed to the relevance of shared information items, as each contribution builds upon the context of the previous message. CogAgents shared the same information items, causing a cyclical loop of discussing the same topic. This resulted in an increase of the frequency of specific shared items but did not increase the variance of information item uniqueness. This repetition of shared information is also evident in Stasser's (1988) DISCUSS model, in which the results demonstrated that discussion may reinforce misaligned preferences rather than correct them.

Table 7: ContributeCog Experiment Analysis

	CGE1 (%)	CGE2 (%)
Revealed Profile	11.50 ± 1.028	12 ± 1.351
Preference Change	0	0
Agent Contribution Distribution	CMR: 26 ± 4.958 MS1: 17 ± 3.969 MS2: 27 ± 5.578 FE: 30 ± 2.772	CMR: 28 ± 5.403 MS1: 23 ± 9.214 MS2: 25 ± 8.874 FE: 24 ± 4.958
Decision Made	0	0

Agent Optimality: Revealed Profile	Best: 35 Average: 55 Worst: 10	Best: 32.5 Average: 55 Worst: 12.5
Agent Optimality: Knowledge Bank	Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

The results of the ContributeCog experiments also share a similar percentage of the revealed profile as BaseCog and InteractionCog trials. The discussions in this set of experiments also never reached a group decision regardless of the decision rule. Even with the introduction of the choice for CogAgents to contribute, the distribution of contributing agents did not majorly change. There is a noticeable increase in the width of the confidence intervals, but it is still not statistically significant enough to demonstrate that the inclusion of this cognitive process had an impact on speaker distribution. Looking deeper into the justifications for CogAgents' choice to contribute, which is provided in the agent states output file, the CogAgents almost always chose to contribute. This points to the altruistic nature of LLM's, particularly GPT-4, being that they are trained to produce helpful responses. The results of the InteractionCog experiments indicate further guidance into the choice for contributing, such as implementing personality or emotions, is necessary to counteract the inherent benevolent nature of LLMs.

The TrustCog experiment results (see appendix B) did not produce statistically significant deviations from the BaseCog results, regardless of the trust condition. The only minor change observed was a slight increase in the optimality of average preferences, with respect to the revealed profile. A minor decrease in the optimality of best preferences is also seen, accommodating the increase in the average optimality. The revealed profile percentage reaches a maximum of approximately 13%. These results indicate that the integration of information into the knowledge base, influenced by the static trust definition, does not impact the choice of which information items are chosen for exchange. This cognitive process, alone, is not enough to emulate the impact of trust. Alternatively, this may necessitate a more nuanced approach to trust measures or the requirement of integrating another cognitive process.

Table 8: MemoryCog Experiment Analysis

	MGE1 (%)	MGE2 (%)
Revealed Profile	16.25 ± 1.588	10.25 ± 2.977
Preference Change	20 ± 11.596	32.5 ± 9.922
Agent Contribution Distribution	CMR: 25 ± 5.714 MS1: 22 ± 7.228 MS2: 24 ± 6.321 FE: 29 ± 3.338	CMR: 17.096 ± 9.002 MS1: 16.107 ± 9.332 MS2: 36.773 ± 6.341 FE: 30.023 ± 10.088
Decision Made	0	100 Rounds Taken: 5.5556 ± 1.633
Optimality: Revealed Profile	Best: 30 Average: 45 Worst: 25	Best: 60 Average: 40 Worst: 0
Optimality: Knowledge Bank	Best: 30 Average: 47.5 Worst: 22.5	Best: 0 Average: 100 Worst: 0

The introduction of memory causes the first seismic effects on the points of analysis. The unanimous decision rule condition still did not result in a consensus, however the majority condition always resulted in a group decision being made. Preference changes are seen in CogAgents, with a lower percentage for the unanimous decision rule condition. This can be explained by the shortened discussions, due to the premature consensus afforded by the majority decision rule. Discussions that reached a consensus did so in approximately half the maximum allocated discussion length. Less discussion turns also impacted the distribution of CogAgent contributions, with more biased weighting of half of the participants. Additionally, MemoryCog increased the best optimality percentage of the final individual preferences of CogAgents (with respect to the revealed profile). Differing from the results of BaseCog, InteractionCog, and TrustCog, the optimality rating percentages (with respect to the knowledge bank) demonstrated a large decrease in worst choices in the unanimous condition. The best and average rating percentages increased as a result. In the majority decision rule, worst choices were eliminated entirely in favor of only average choices in the optimality ratings regarding the revealed profile and the knowledge bank.

The results of the MemoryCog experiments indicate that there is a major impact on the order of the information presented to the LLM. MemoryCog is the only CogChain to reveal non-supportive and neutral items in simulation. This 'scope of sight' of language models proves to be a challenge in modeling human-like memory, as the attention of the LLM is biased based on internal weights that are trained into the model. These internal weights may not represent a true emulation of how the order of presentation affects attention processing. This prompts the obligation to further investigate if, and how, these internal weights can be manipulated to mimic human-like memory bias.

When analyzing for human-likeness across cogs, the only CogChain to demonstrate preference changes is MemoryCog (see table 9). This is consistent with the findings demonstrated in Stasser and Titus' hidden profile experiments with human participants, where the pregroup and postgroup preferences varied by a range of 5 to 14 percent (Stasser & Titus, 1985). Preference changes are also seen in the distribution preferences produced by the DISCUSS model predictions, with a range of 3 to 9 percent (Stasser, 1988). However, all CogChains exhibit the phenomenon of individual CogAgents choosing the average decision option according to the revealed profile (see table 9). This aligns with the human-based results of Stasser and Titus, as the average option was the clear favorite attaining 61% of individual member support.

Table 9: CogFrame Group Configuration 1 Cross-Cog Analysis

	Preference Changes	Contribution Split	Agent Optimality (Revealed Profile)	Agent Optimality (Knowledge Bank)

BaseCog	0	CMR: 21 ± 5.85 MS1: 20 ± 5.54 MS2: 30 ± 8.32 FE: 29 ± 6.47	Best: 25 **Average: 65** Worst: 10	Best: 25 Average: 25 **Worst: 50**
InteractionCog	0	CMR: 24 ± 4.958 MS1: 27 ± 3.969 MS2: 31 ± 4.339 FE: 18 ± 3.719	Best: 40 **Average: 55** Worst: 5	Best: 25 **Average: 55** Worst: 50
ContributeCog	0	CMR: 26 ± 4.958 MS1: 17 ± 3.969 MS2: 27 ± 5.578 FE: 30 ± 2.772	Best: 35 **Average: 55** Worst: 10	Best: 25 Average: 25 **Worst: 50**
MemoryCog	**20 ± 11.596**	CMR: 25 ± 5.714 MS1: 22 ± 7.228 MS2: 24 ± 6.321 FE: 29 ± 3.338	Best: 30 **Average: 45** Worst: 25	Best: 30 **Average: 47.5** Worst: 22.5
TrustCog (All High)	0	CMR: 25 ± 4.997 MS1: 24 ± 4.958 MS2: 27 ± 6.818 FE: 24 ± 4.11	Best: 20 **Average: 62.5** Worst: 17.5	Best: 25 Average: 25 **Worst: 50**
TrustCog (All Low)	0	CMR: 30 ± 3.92 MS1: 18 ± 5.403 MS2: 23 ± 7.864 FE: 29 ± 4.339	Best: 22.5 **Average: 60** Worst: 17.5	Best: 25 Average: 25 **Worst: 50**
TrustCog (Half/Half)	0	CMR: 30 ± 3.92 MS1: 18 ± 5.403 MS2: 23 ± 7.864 FE: 29 ± 4.339	Best: 30 **Average: 60** Worst: 10	Best: 25 Average: 25 **Worst: 50**

Table 10: Compound Chain 1 Experiment Analysis

	ICMT1 GE1 (%)	ICMT1 GE2 (%)
Revealed Profile	16.25 ± 1.588	9 ± 2.42
Preference Change	22.5 ± 12.871	32.5 ± 12.102
Agent Contribution Distribution	CMR: 24 ± 5.681 MS1: 27 ± 6.818 MS2: 23 ± 7.3 FE: 26 ± 5.681	CMR: 20.453 ± 10.983 MS1: 27.453 ± 11.573 MS2: 27.453 ± 11.573 FE: 29.643 ± 13.135
Decision Made	0	100 Rounds Taken: 4.1 ± 1.222
Optimality: Revealed Profile	Best: 27.5 Average: 55 Worst: 17.5	Best: 40 Average: 50 Worst: 10
Optimality: Knowledge Bank	Best: 25 Average: 42.5	Best: 0 Average: 100

	Worst: 32.5	Worst: 0

Table 11: Compound Chain 2 Experiment Analysis

	ICMT2 GE1 (%)	ICMT2 GE2 (%)
Revealed Profile	14 ± 1.24	11.5 ± 2.956
Preference Change	30 ± 11.596	27.5 ± 8.344
Agent Contribution Distribution	CMR: 25 ± 8.43 MS1: 21 ± 7.566 MS2: 29 ± 3.338 FE: 25 ± 6.93	CMR: 24.745 ± 8.102 MS1: 26.873 ± 11.878 MS2: 27.856 ± 8.056 FE: 20.523 ± 9.037
Decision Made	0	60 Rounds Taken: 4.5 ± 2.053
Group Optimality: Revealed Profile	-	Best: 33.33 Average: 66.7 Worst: 0.00
Group Optimality: Knowledge Bank	-	Best: 0 Average: 100 Worst: 0
Agent Optimality: Revealed Profile	Best: 40 Average: 42.5 Worst: 17.5	Best: 31.25 Average: 62.50 Worst: 6.25
Agent Optimality: Knowledge Bank	Best: 22.5 Average: 55 Worst: 22.5	Best: 25 Average: 43.75 Worst: 31.25

Table 12: Compound Chain 3 Experiment Analysis

	ICMT3 GE1 (%)	ICMT3 GE2 (%)
Revealed Profile	16.5 ± 1.028	11.25 ± 2.219
Preference Change	22.5 ± 12.871	24.5 ± 9.844
Agent Contribution Distribution	CMR: 24 ± 4.111 MS1: 29 ± 4.339 MS2: 20 ± 3.92 FE: 27 ± 2.84	CMR: 21.611 ± 9.309 MS1: 25.611 ± 10.926 MS2: 25.055 ± 10.261 FE: 27.722 ± 9.006
Decision Made	0	90
Group Optimality: Revealed Profile	-	Best: 33.33 Average: 55.55 Worst: 11.12
Group Optimality: Knowledge Bank	-	Best: 0 Average: 100 Worst: 0
Agent Optimality: Revealed Profile	Best: 45 Average: 40 Worst: 15	Best: 50 Average: 25 Worst: 25

Agent Optimality: Knowledge Bank	Best: 27.5 Average: 47.5 Worst: 25	Best: 25 Average: 50 Worst: 25

The compound chain results demonstrate that the combination of the CogChains produce the highest percentage of the revealed profile, in comparison to the unit CogChain results, across trust conditions for both decision rules. The preference changes and distribution of contributing agents align with the impact described by their unit test. No group decisions were made in unanimous conditions, however the highest optimality rating for best decisions (according to the revealed profile) is seen in the ICMT3_GE2 configuration (see table 12). This indicates that the half and half trust tendency split between high and low biased the information retention of CogAgents, reinforcing the preference of CogAgents that have the best optimality rating. The rarity of the best choice being made in this condition aligns with human-like behavior, as only 18% of human-based groups agreed on the best decision option (Stasser & Titus, 1985). The majority decision rule conditions did not always arrive at a group consensus, as seen in the ICMT2 and ICMT3 configurations, differing from the MemoryCog results. They are similar insofar as the distribution of knowledge bank group optimality ratings for the majority decision rule, which never chose the worst preference.

The combination of CogChains is intended to produce a more cognitively complex human-like agent in comparison to the individual CogChains. As previously discussed, the unit testing trials pointed towards a clear favoritism of the average decision option (see table 9). This human-like trend continues in the combination chain results, as seen in the second group configuration cross-analysis table (see table 13). The major distinction between the combination trials and the MemoryCog trials is that the group does not always come to a decision. In the ‘all high’ trust configuration, information is always integrated, so the results are similar to the MemoryCog results. In the second and third trust configurations, the variation in trust tendencies indicate an impact on the percentage of trials that resulted in a consensus (see table 13). The ‘all low’ condition generated an agreement 60% of the time, while the half and half condition jumped to 90%. The participants in Stasser & Titus’ human-based study came to an agreement most of the time, with only one group unable to reach a decision (Stasser & Titus, 1985). This is indicative of the half and half condition being a more natural representation for the team composition (see table 13), as human groups do not always reach a decision (as seen in the ‘all high’ condition) but do so more often than 60% (as seen in the ‘all low’ condition).

Table 13: Combination Chain Group Configuration 2 Cross-Analysis

	Preference Changes	Decision Made	Group Optimality (Revealed Profile)	Group Optimality (Knowledge Bank)
ICMT1 (All High)	32.5 ± 12.102	100 Rounds Taken: 4.1 ± 1.222	Best: 40 **Average: 50** Worst: 10	Best: 0 **Average: 100** Worst: 0
ICMT2 (All Low)	27.5 ± 8.344	60 Rounds Taken: 4.5 ± 2.053	Best: 33.33 **Average: 66.7** Worst: 0	Best: 0 **Average: 100** Worst: 0
ICMT3 (Half/Half)	24.5 ± 9.844	90 Rounds Taken: 4.2 ± 1.897	Best: 35 **Average: 55** Worst: 10	Best: 25 Average: 25 **Worst: 50**

Combining these chains illustrates that the CogChains that have a similar impact on the same point of analysis across their unit and compound test are not massively influenced by their combination with other CogChains. The distribution of CogAgent contributions were not overwhelmingly different across trials, signaling that ContributeCog either has limited interaction with other CogChains or needs further extrapolation of CogModules. A similar conclusion can be drawn for TrustCog. The most impactful of CogChains is MemoryCog, which produced changes in preference, discussion result, optimality ratings, and an overall increase in the revealed profile percentage (see table 9). Memory proves to be pivotal in its impact on attention, which further delineates which information is most pertinent to consider when determining a preference or choosing which information item to share.

CHAPTER 5

IMPLICATIONS AND CONTRIBUTIONS

Advancement of AI-Based Human Behavior Modeling

The advancements made in AI-based human behavior modeling presented in this study contribute to the broader goal of developing more sophisticated and realistic AI agents. It demonstrates the ability to implement a cognitive architecture as the inference system of AI agents that actuate with an LLM. This was achieved through the chaining of LLM cognitive process prompts and coded cognitive processes. The results also point towards the limitations in the abstraction of these cognitive processes to fit the capabilities of technology. Highly abstracting minor processes is possible, but the same cannot be said for broadly impactful processes like memory. Memory is one of the most difficult processes to simulate at a human-like level, and there is no consensus on the bounds of working memory in cognitive architecture (Kotseruba & Tsotsos). There are relatively few architectures utilizing working memory models that provide satisfactory information on the internal process structure and their interactions (Kotseruba & Tsotsos). The result of this study provides evidence that further research into implementing and modeling memory related processes, such as activation mechanisms (Anderson et al. 1996), can shed light on which are necessary to emulate human behavior. Investigating this avenue uncovers the more intricate underlying mechanisms of memory, and to what extent it can be captured by the current capacity of artificial intelligence.

Enhanced Understanding of Hidden Profiles

There was not a single discussion simulation across any of the trials that produced a consensus agreeing on the best decision, with respect to the overall knowledge bank. The prevalence of the hidden profile is demonstrated even at the BaseCog level. The presence of hidden profiles persists even in the absence of consensus, emphasizing that information exchange is not only valuable to group decision making but also individual-made decisions. Singular cognitive processes do not majorly impact the hidden profile; rather, it is their combination that produces the largest effect. This suggests that the interrelationships between cognitive processes are crucial in revealing, and obscuring, hidden profiles. The results of this study point towards investigating the interactions between modules that cause the highest impact, rather than focusing solely on individual processes. Prioritizing depth over breadth in understanding the nuances of the influences of cognitive processes can provide insight into a specific association between a detailed process interaction and the hidden profile.

Potential Applications: Improving Decision-Making Processes

Researchers can deploy the CogSystem schema in different task environments to analyze which cognitive modules affect conversation trends and warrant further investigation. By providing an insight into how people make decisions and interact with others, techniques can be developed to counteract potential obstacles. These strategies can be further explored through CogSystem by implementing additional CogModules that define the interaction rules, relevant cognitive processes, and other calculated restrictions. The architecture is designed to be open and extensible, so that it can be easily modified and extended to support new cognitive processes as needed for specific discussion tasks. A modular and hierarchical design is implemented, enabling the ability to incorporate new modules and systems as they are developed. In doing so, the architecture is flexible and adaptable while remaining transparent for explainability. Future extensions of this architecture may incorporate more complex inference by also implementing causal reasoning. Computational techniques, such as parallel processing, can be employed to increase optimization. An alternative method to the optimization technique would be using machine learning strategies to enhance performance as time passes. An example of this would be reinforcement learning, where the agent would receive a negative reward if their contribution resulted in a conflict or deviation from the discussion task.

Potential Applications: Enhancing Human-AI Collaboration

CogSystem can be used in optimizing productive human interaction through conversational agents. Human behavior may be misaligned, inhibiting conversation flow. Conversational agents can function as facilitators for group discussions to enhance conversation flow (Hogan et. al, 2021). The special role of a team member is also seen in Stasser's expert-role assignments. These conversational agents may potentially be interventive experts that encourage the exchange of unshared information. Kraus et. al (2020) identifies human team member perceptions of trust as significant in the evaluation of a conversational agent. Team member perception is influenced by how proactive or reactive the system is designed to be, as higher-level proactive rule interactions negatively impact perception of the agent. The conversational agents themselves may also be optimized to have a more human-like performance through the integration of a CogChain. A more human-like performance would provide a more realistic team member interaction, rather than the 'wizard of oz' approach used in other studies, which may enhance team member perception of the agent

REFERENCES

Stasser, G. (1988). Computer simulation as a research tool: The discuss model of Group Decision making. *Journal of Experimental Social Psychology*, *24*(5), 393–422. https://doi.org/10.1016/0022-1031(88)90028-5

Martínez-Miranda, J., & Pavón, J. (2012). Modeling the influence of trust on work Team Performance. *SIMULATION*, *88*(4), 408–436. https://doi.org/10.1177/0037549711404714

Hinsz, V. B., Tindale, R. S., & Vollrath, D. A. (1997). The emerging conceptualization of groups as information processes. *Psychological Bulletin*, *121*(1), 43–64. https://doi.org/10.1037//0033-2909.121.1.43

Stasser, G., & Titus, W. (1985). Pooling of unshared information in Group Decision making: Biased information sampling during discussion. *Journal of Personality and Social Psychology*, *48*(6), 1467–1478. https://doi.org/10.1037//0022-3514.48.6.1467

Stasser, G., Vaughan, S. I., & Stewart, D. D. (2000). Pooling unshared information: The benefits of knowing how access to information is distributed among group members. *Organizational Behavior and Human Decision Processes*, *82*(1), 102–116. https://doi.org/10.1006/obhd.2000.2890

Stasser, G., & Davis, J. H. (1981). Group decision making and social influence: A social interaction sequence model. *Psychological Review*, *88*(6), 523–551. https://doi.org/10.1037//0033-295x.88.6.523

Yenduri, G., M, R., G, C. S., Y, S., Srivastava, G., Maddikunta, P. K. R., G, D. R., Jhaveri, R. H., B, P., Wang, W., Vasilakos, A. V., & Gadekallu, T. R. (2023). Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. *Arxiv*. https://doi.org/10.48550/arXiv.2305.10435

Jones, C., & Bergen, B. (2023). Does GPT-4 Pass the Turing Test? *Arxiv*. https://doi.org/10.48550/arXiv.2310.20216

Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson.

Sun, R. (2007). Cognitive social simulation incorporating cognitive architectures. *IEEE Intelligent Systems*, *22*(5), 33–39. https://doi.org/10.1109/mis.2007.4338492

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Arxiv*. https://doi.org/10.48550/arXiv.2201.11903

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., & Neubig, G. (2023). PAL: Program-aided Language Models. *Arxiv*. https://doi.org/10.48550/arXiv.2201.11903

Kotseruba, I., & Tsotsos, J. K. (2020). 40 years of cognitive architectures: Core cognitive abilities and practical applications. *Artificial Intelligence Review*, *53*(1), 17–94. https://doi.org/10.1007/s10462-018-9646-y

Penrod, S., & Hastie, R. (1980). A computer simulation of jury decision making. *Psychological Review*, *87*(2), 133–159. https://doi.org/10.1037//0033-295x.87.2.133

Hastie, R., Penrod, S. D., & Pennington, N. (1983). *Inside the Jury*. https://doi.org/10.4159/harvard.9780674865945

Stasser, G., & Davis, J. H. (1981). Group decision making and social influence: A social interaction sequence model. *Psychological Review*, *88*(6), 523–551. https://doi.org/10.1037//0033-295x.88.6.523

Marks, M. A., Mathieu, J. E., & Zaccaro, S. J. (2001). A temporally based framework and taxonomy of Team Processes. *Academy of Management Review*, *26*(3), 356–376. https://doi.org/10.5465/amr.2001.4845785

Martínez-Miranda, J., & Pavón, J. (2009). Modelling Trust into an agent-based simulation tool to support the formation and configuration of work teams. *Advances in Intelligent and Soft Computing*, 80–89. https://doi.org/10.1007/978-3-642-00487-2_9

Lieto, A., Bhatt, M., Oltramari, A., & Vernon, D. (2017). The role of cognitive architectures in General Artificial Intelligence. *Cognitive Systems Research*, *48*, 1–3. https://doi.org/10.1016/j.cogsys.2017.08.003

Oltramari, A., & Lebiere, C. (2012). Pursuing artificial general intelligence by leveraging the knowledge capabilities of ACT-R. *Artificial General Intelligence*, 199–208. https://doi.org/10.1007/978-3-642-35506-6_21

Vernon, D. (2017) Two ways (not) to design a cognitive architecture. *Cognitive Robot Architectures*, 42.

Laird, J. E., Lebiere, C., & Rosenbloom, P. S. (2017). A standard model of the mind: Toward a common computational framework across artificial intelligence, Cognitive Science, Neuroscience, and Robotics. *AI Magazine*, *38*(4), 13–26. https://doi.org/10.1609/aimag.v38i4.2744

Cordeschi, R. (2002). *The discovery of the artificial: Behavior, mind and machines before and beyond cybernetics* (Vol. 28). Springer Science & Business Media.

Laird, J. (2012). *The Soar Cognitive Architecture*. MIT Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036–1060. https://doi.org/10.1037/0033-295x.111.4.1036

Sun, R. (2006). The Clarion Cognitive Architecture: Extending cognitive modeling to Social Simulation. *Cognition and Multi-Agent Interaction*, 79–99. https://doi.org/10.1017/cbo9780511610721.005

Vernon, D., Metta, G., & Sandini, G. (2007). The ICUB cognitive architecture: Interactive development in a humanoid robot. *2007 IEEE 6th International Conference on Development and Learning*, 122–127. https://doi.org/10.1109/devlrn.2007.4354038

Cohen, P. R., & Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, *42*(2–3), 213–261. https://doi.org/10.1016/0004-3702(90)90055-5

Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press.

Bourgais, M., Taillandier, P., & Vercouter, L. (2020). Ben: An architecture for the behavior of Social Agents. *Journal of Artificial Societies and Social Simulation*, *23*(4). https://doi.org/10.18564/jasss.4437

Breazeal, C. (2003). Emotion and sociable humanoid robots. *International Journal of Human-Computer Studies*, *59*(1–2), 119–155. https://doi.org/10.1016/s1071-5819(03)00018-1

Gobet, F., & Lane, P. C. (2012). Chunking mechanisms and learning. *Encyclopedia of the Sciences of Learning*, 541–544. https://doi.org/10.1007/978-1-4419-1428-6_1731

Axelrod, R. (1997). Advancing the art of simulation in the social sciences. In R. Conte, R. Hegselmann & P. Terna, *Simulating Social Phenomena*, 21–40.

Hogan, K., Baer, A., & Purtilo, J. (2021). Diplomat: A conversational agent framework for goal-oriented group discussion. *Contemporary Issues in Group Decision and Negotiation*, 143–154. https://doi.org/10.1007/978-3-030-77208-6_11

Kraus, M., Wagner, N., & Minker, W. (2020). Effects of proactive dialogue strategies on Human-Computer Trust. *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*. https://doi.org/10.1145/3340631.3394840

Atkinson, R. C., & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. *Psychology of Learning and Motivation*, 89–195. https://doi.org/10.1016/s0079-7421(08)60422-3

APPENDIX A: FAST FIVE DISCUSSION TASK

Overview

You work as a member of NASA's Astronaut Crew Composition team. Your team analyzes all of the information obtained during astronaut selection and training - including test scores, analog performance measures, expert observations, peer evaluations, and medical records. Armed with this information, your team makes two kinds of evaluations. The first evaluation is to determine which astronauts are ready for flight and which are not. Of those who are deemed to be flight ready, your team composes entire crews for each mission. In this second evaluation, you consider the complementarity and compatibility of the astronauts as a team to select the 4, 5, or 6 astronauts (depending on the mission) who will perform best as individuals, and as a crew. While composing crews for missions on the International Space Station has become routine, your team just received a special assignment from the Mission Planning Directorate that is anything but routine.

The Mars mission has been fast-tracked. Instead of the 2033 launch date that NASA had been aiming for, the launch date is being moved up to 2020. Intensive crew training for this unprecedented and bold journey must begin immediately. A 5-member crew will embark on this 3 year journey to Mars. This a truly international endeavor: the Russian, European, Indian, and Chinese Space Agencies will each select one astronaut for the mission. NASA will select the 5th crew member - an American astronaut - to join this international crew.

The crew members from the other countries have backgrounds in engineering, botany, geology and medicine. The crew still needs a flight engineer.

Your team was able to quickly narrow the field down to three veteran astronauts: Anders, Bean, and Collins. Each of these three have experience as a flight engineer on the ISS, have the necessary basic technical skills for the flight engineer position, meet basic requirements for the mission including height, Body Mass Index, and have passed a mental health screening. Now that you know the other potential crew members it is necessary for your team to determine which of the crew members will be the greatest asset on this particular mission.

The 3-year mission to Mars will be demanding. The success of the Mars mission will require the 5-person crew to live and work together in extremely close quarters, while isolated from the usual sources of social support - friends and family - for extended periods of time. It will require that the crew can adapt to the unknown and be able to handle whatever challenges will come their way. Once they arrive at Mars, the crew will have to complete complex feats such as landing and extravehicular activities. Because of a significant communication delay with Earth of up to 22 minutes each way, the crew will have to work autonomously. Additionally, the crew of the Mars mission consists of individuals representing multiple nationalities, making strong interpersonal skills all the more critical to the success of the mission.

The international space consortium has agreed to announce the names of the crew members from each of the 5 countries in just 10 days from now. Your team *must* deliver a recommendation for whom should be the American astronaut; years of hard work (and millions of dollars) rest on this decision.

To guide your difficult decision, you have compiled information from the personnel file for each candidate. The personnel files for each candidate are detailed and include information such as their performance scores and ranking on various training tasks, biographical information, evaluations and questionnaires completed by supervisors and individuals with whom they worked on prior trainings and missions, and questionnaires completed by the candidate and his family. A few human resource assistant have been working through the personnel files and have prepared briefs for you. Your team must now come together and discuss who should be selected for the mission.

Remember, the success of the mission to Mars depends on your team. Choose quickly, but wisely!

Group Decision Making Task

In this exercise, you will meet with the Crew Composition Team to determine which of the three astronauts to send to Mars: Anders, Bean, or Collins.

The spacecraft will only fit 5 crewmembers; This means that you *must* choose only 1 of the three remaining candidates. As a result of this constraint, please note that it is <u>not</u> appropriate to say that each astronaut is equally qualified or to choose all three.

Fortunately, you work with an outstanding team and each individual has also compiled information on the three candidates and prepared their own personnel briefing for to aid in the decision.

Note: it is not appropriate to consider any information other than that presented in this report. Thus, no web-searching, prior reports, etc. are permitted.

Your task is to review the personnel briefs, and make a preliminary decision. Then, you will meet with your crewmates to discuss the candidates and reach a team decision. It is up to the team to decide whether the decision should be unanimous, majority rule, or some variation thereof.

Ground Rules

- Read the personnel brief carefully. The content of this brief represents your knowledge about the three candidates.
- Take any notes that you want, but you will not have access to the case description or personnel brief in your crew meeting.

- You are working under serious time pressure. NASA wants to announce its choice at the international press briefing in just 10 days. The clock is ticking.

Personnel Brief

Candidate 1: Samuel Anders (Best option - 6G, 4B, 4N)
Samuel Anders is a 35-year-old male. He was born and raised in Houston, TX, and attended a space education high school magnet program. Living only one town over from the Johnson Space Center, he always knew he wanted to be an astronaut. He attended Yale, where he earned his Bachelor's degree in aerospace engineering and cultural anthropology. He graduated with high honors. He next completed his masters in electrical engineering from UC Berkeley University. After graduation, he moved back to Houston where he completed the astronaut candidate training program. He has since worked in a number of roles in the Exploration Branch, including a space mission as a Flight Engineer for STS-127. In his free time, Astronaut Anders enjoys working with his favorite charities to raise money for Science, Technology, Engineering, and Math (STEM) education in underprivileged communities, playing the guitar, and hiking with his dog, Bailey.

G1: Astronaut Anders received the highest possible scores on the military survival standards, an assessment that all astronauts must complete in order to qualify for space missions. Anders' scores were significantly higher than either candidate's scores.
G2: During Ander's last ICE (Isolated and Confined in Extreme environments) training sequence, a fellow crew member praised him for being able to recognize others' personal needs. In particular, he was recognized for facilitating personal time and bringing the group together for light-hearted fun.
G3: While working at NASA, Anders coordinated and completed a number of science projects. This expertise is important to Flight Engineers and one of the reasons Anders was chosen for the previous space mission. Anders did not disappoint. His ability to set up and coordinate science projects allowed his crew to finish more than twice the number of projects than anticipated.
G4: Anders was one of 8 hikers to successfully summit Mount Everest in 2005. Interviews with his comrades suggest Anders was the "glue" that held the group together when things went wrong.
G5: On previous missions, where lack of sleep was a frequent occurrence, Anders was largely unaffected. Despite the sub-optimal sleeping patterns, Anders' mental and physical health did not show signs of suffering.
G6: Data collected during training on simulated emergencies shows that Anders is able to remain calm during stressful and unexpected situations. He performs well under pressure.

B1: Anders suffered a broken wrist 7 months ago during a training simulation. The injury has healed, but will likely require extra exercise to prevent forearm muscle atrophy in zero gravity. This may take time away from him performing other team duties.
B2: Most of Anders' space training experience has been limited to simulations on Earth, with only one mission to the International Space Station. Furthermore, this mission to Mars would be even longer than his only long-distance flight experience--it is not clear if he will adapt well for a long-duration mission.
B3: Anders comes from a close-knit family. In diaries collected during training, he reported feeling sad and frustrated when he was unable to communicate with family members on a regular

basis. This could be a serious concern for his mental health on a mission to Mars because there will be very little contact with Earth, and the little contact there is will be through mission control.
B4: Anders easily gets annoyed when explaining concepts to others and becomes very short with them as a result.

N1: Anders enjoys science fiction literature and has self-published a novel about the start of a human colony in a distant galaxy. He hopes to write a sequel when he retires.
N2: As a child, Anders enjoyed attending Houstonli-area sporting events with his parents. He was particularly fond of attending the Texans football games.
N3: Anders has a pet cat named Buzz, after Buzz Aldrin. Anders met Buzz as a child around when he decided he wanted to be an astronaut.
N4: Anders used to own a Brown Labrador Retriever named Bailey. When Anders participated in astronaut training and missions, Bailey used to stay with his family.

Candidate 2: John Bean (Middle Option - 3G, 3B, 6N)
John Bean is a 32-year-old male who grew up in Cape Canaveral, Florida right near Kennedy Space Center. Bean grew up watching the shuttle launches, and attended space camp every summer.. When he was 18, he moved to Palo Alto, California to attend Stanford to pursue a Bachelor's degree in aerospace engineering, and cross-cultural studies. He graduated cum laude. He next completed his masters in electrical engineering from Texas A&M University. After completing his education he worked at NASA's Ames Research Center in Moffett Field, California. After a few years at Ames, he was selected to be an astronaut. He has been very successful in his role as an astronaut and just three years ago, he served as Flight Engineer for the Expedition 31/32. This expedition was primarily a resupply mission to the International Space Station, but they also completed a number of important science experiments while on the International Space Station. In addition to an enthusiasm for science, Astronaut Bean enjoys classic literature, mentoring the next generation of astronauts at space camp, and sailing.

G1: On Expedition 31/32, where he helped refuel the International Space Station (ISS), Bean encountered a breakdown in the communication system between his vehicle and the ISS before arrival to the ISS. Bean was credited for leading the extravehicular activity that identified and fixed the issue, ultimately making the refuel possible.
G2: While it is widely known that Bean holds a bachelor's degree in Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a minor in Counseling Psychology. Crewmembers have reported that Bean is a great listener and often knows exactly what to say to make others feel more comfortable.
G3: Survey responses from Bean's previous crew mates reveal that he has a universal sense of humor. People always understand enjoy Bean's jokes, so there is little risk of his humor being misinterpreted as being negative. Crew members report that this really helps to de-escalate tense situations.

B1: Bean had a difficult time getting along with some of his co-workers at Ames. He had asked his supervisor to intervene on several occasions as they were not able to resolve the disagreements on their own.

B2: Bean does not always plan his work well, and sometimes his pace of work is too slow. This has become problematic not only in his mission to the ISS, but also in training simulations. Often times, the maintenance work and data collection the astronauts engage in on these missions requires high levels of interdependency - this can be jeopardized if one person works at a slower pace than the rest of the crew members.
B3: Bean has a 2-year-old daughter, and colleagues have noticed he is frustrated when he has to work late and cannot be home with her. There is worry that being away for so long on this mission will exponentially increase Bean's frustration.

N1: Bean enjoys listening to the Beatles while performing work on his own.
N2: Bean is an avid fan of Shakespeare and often quotes his work.
N3: In college, Bean was on the squash team, where he met his best friend Jack. Jack now lives in Providence, Rhode Island, but they still talk on a weekly basis.
N4: Astronaut Bean enjoys traveling and his favorite place to vacation to is in the south of France.
N5: Bean has five siblings: one older brother, one older sister, and three younger sisters. He is closest to his younger sister, Rebecca, who is just one year younger than he is.
N6: If chosen for this mission, Bean would be the shortest crew member at 5-feet, 5-inches tall.

Candidate 3: Scott Collins (worst option 4G, 6B, 4N)
Scott Collins is a 33-year-old California native who grew up in Pasadena near the NASA Jet Propulsion Laboratory. Collins has always wanted to be a career astronaut, and became interested in spaceflight and exploration from a very early age. Astronaut Collins has a dual Bachelor’s degree in aerospace engineering and psychology from the Massachusetts Institute of Technology (MIT). He also has a masters in electrical engineering from University of Illinois. At NASA, Collins served mostly in the International Space Station Operations and International Space Station Integration offices. He was the Flight Engineer on STS - 135, which was the last flight of the American Space Shuttle. When he does get free time, Collins volunteers with at-risk youth, coaches little league, and enjoys surfing.

G1: Collins is accustomed to being isolated for long periods of time. He successfully completed two treks to the Antarctic in which he lived in tents and a small building for the Antarctic winter-over period. Living in small spaces did not bother him.
G2: Collins has a passion for languages. He is fluent in English, Spanish, and French and proficient in Russian. Recently he started a class on Mandarin and already he is nearly proficient. This ability will help him interact and bond with international crew members.
G3: Collins has participated in spacewalks to repair damaged solar panels and ammonia pump module. His peers have ranked him as the most competent crew member during their spacewalks.
G4: Collins received the “Silver Snoopy” award for his contributions on NASA’s STS-135 mission. This award is considered to be the highest honor of all of the awards, given to the person who was the most outstanding team member, creating a team environment focused on flight safety and overall team success.

B1: Surveys showed that Collins has snapped at coworkers on multiple occasions. In one situation, a crewmember even went so far as to say would not want to work with Collins on future missions.
B2: Collins has the least space-related training of all three potential candidates.
B3: Recent blood work indicated that Collins may be developing above-average blood pressure. Doctors advised another assessment before making any conclusive determinations. However, if Collins does have heightened blood pressure, he would have to take extra time to rest while on the mission. This would detract from his time working on experiments, as well as his ability to do physically stressful activities.
B4: Three weeks ago, Collins broke his ankle. As a result, he has not been training as much and the current state of his cardiovascular performance is behind compared to the other two candidates.
B5: In one incident during a mock extravehicular activity scenario, Collins refused to follow an instruction from mission control. Fellow crew members were unsure of why he did this since the request from mission control seemed straightforward and helpful. Crewmembers reported being conflicted with how to respond, which slowed their progress on the extravehicular activity scenario. Only half of the tasks needed were completed on the simulated extravehicular activity.
B6: An internal incident report indicates that Collins made offensive jokes to a Chinese astronaut on a previous analog space mission that made the Chinese astronaut and other crew members uncomfortable.

N1: Collins considers himself a coffee aficionado and his favorite brand of coffee is Intelligentsia.
N2: Collins' childhood friend completed NASA's astronaut candidate training program but was never selected for a space mission.
N3: Collins enjoys completing Sudoku puzzles to relax.
N4: Collins has a fraternal twin who works in finance in Houston, Texas.

		CMR (C)	MS1 (1)	MS2 (2)	FE (F)	
Collins	Good	G1, G2, G3, G4	G1	G1, G2, G3, G4	G3, G4	Worst
	Bad	B4	B3, B4, B5, B6	B1	B1, B2, B6	
	Neutral	N1, N2	N3, N4	N1, N3	N2, N3	
Bean	Good	G1, G2, G3	G1, G2, G3	G1, G2, G3	G1, G2, G3	Middle
	Bad	B2, B3	B1, B2, B3	B1, B2	B1	
	Neutral	N1, N2	N3, N4	N1, N3	N2, N3, N5, N6	
Anders	Good	G2, G6	G1, G2, G3, G4	G3, G4	G5	Best
	Bad	B1, B2, B3, B4	B1	B1, B2, B3, B4	B1, B2, B3, B4	
	Neutral	N1, N2	N1, N2, N3	N3, N4	N2, N3, N4	

APPENDIX B: TRUSTCOG EXPERIMENT ANALYSIS

Trust1Cog Experiment Analysis

		T1GE1 (%)	T1GE2 (%)
Revealed Profile		12.25 ± 0.834	11.75 ± 0.992
Preference Change		0	0
Agent Contribution Distribution		CMR: 25 ± 4.997 MS1: 24 ± 4.958 MS2: 27 ± 6.818 FE: 24 ± 4.111	CMR: 19 ± 6.471 MS1: 26 ± 4.958 MS2: 27 ± 4.841 FE: 28 ± 5.403
Decision Made		0	0
Agent Optimality: Revealed Profile		Best: 20 Average: 62.5 Worst: 17.5	Best: 25 Average: 62.5 Worst: 12.5
Agent Optimality: Knowledge Bank		Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

Trust2Cog Experiment Analysis

	T2GE1 (%)	T2GE2 (%)
Revealed Profile	11.5 ± 0.759	11.67 ± 0.77
Preference Change	0	0
Agent Contribution Distribution	CMR: 30 ± 3.92 MS1: 18 ± 5.403 MS2: 23 ± 7.864 FE: 29 ± 4.339	CMR: 27 ± 5.578 MS1: 25 ± 7.962 MS2: 27 ± 6.818 FE: 21 ± 4.339
Decision Made	0	0
Agent Optimality: Revealed Profile	Best: 22.5 Average: 60 Worst: 17.5	Best: 22.5 Average: 70 Worst: 12.5
Agent Optimality: Knowledge Bank	Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

Trust3Cog Experiment Analysis

	T3GE1 (%)	T3GE2 (%)
Revealed Profile	10.5 ± 1.518	11.5 ± 1.58

Preference Change	0	0
Agent Contribution Distribution	CMR: 22 ± 4.638 MS1: 30 ± 4.801 MS2: 23 ± 6.229 FE: 25 ± 8.43	CMR: 22 ± 4.638 MS1: 19 ± 5.148 MS2: 30 ± 6.198 FE: 29 ± 6.471
Decision Made	0	0
Agent Optimality: Revealed Profile	Best: 30 Average: 60 Worst: 10	Best: 25 Average: 57.5 Worst: 17.5
Agent Optimality: Knowledge Bank	Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

Trust1Cog Experiment Analysis

		T1GE1 (%)	T1GE2 (%)
Revealed Profile		12.25 ± 0.834	11.75 ± 0.992
Preference Change		0	0
Agent Contribution Distribution		CMR: 25 ± 4.997 MS1: 24 ± 4.958 MS2: 27 ± 6.818 FE: 24 ± 4.111	CMR: 19 ± 6.471 MS1: 26 ± 4.958 MS2: 27 ± 4.841 FE: 28 ± 5.403
Decision Made		0	0
Agent Optimality: Revealed Profile		Best: 20 Average: 62.5 Worst: 17.5	Best: 25 Average: 62.5 Worst: 12.5
Agent Optimality: Knowledge Bank		Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

Trust2Cog Experiment Analysis

	T2GE1 (%)	T2GE2 (%)
Revealed Profile	11.5 ± 0.759	11.67 ± 0.77
Preference Change	0	0
Agent Contribution Distribution	CMR: 30 ± 3.92 MS1: 18 ± 5.403 MS2: 23 ± 7.864 FE: 29 ± 4.339	CMR: 27 ± 5.578 MS1: 25 ± 7.962 MS2: 27 ± 6.818 FE: 21 ± 4.339
Decision Made	0	0
Agent Optimality: Revealed Profile	Best: 22.5 Average: 60 Worst: 17.5	Best: 22.5 Average: 70 Worst: 12.5
Agent Optimality: Knowledge Bank	Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

Trust3Cog Experiment Analysis

	T3GE1 (%)	T3GE2 (%)
Revealed Profile	10.5 ± 1.518	11.5 ± 1.58
Preference Change	0	0
Agent Contribution Distribution	CMR: 22 ± 4.638 MS1: 30 ± 4.801 MS2: 23 ± 6.229 FE: 25 ± 8.43	CMR: 22 ± 4.638 MS1: 19 ± 5.148 MS2: 30 ± 6.198 FE: 29 ± 6.471
Decision Made	0	0
Agent Optimality: Revealed Profile	Best: 30 Average: 60 Worst: 10	Best: 25 Average: 57.5 Worst: 17.5
Agent Optimality: Knowledge Bank	Best: 25 Average: 25 Worst: 50	Best: 25 Average: 25 Worst: 50

APPENDIX C: BASECOG CODE

CogAgent.py

```
class CogAgent:
    def __init__(self, name, knowledge_base):
        self.name = name
        self.knowledge_base = knowledge_base
        self.initial_knowledge_base = knowledge_base.copy()
        self.preference = ""
        self.state = []
        self.frequency = 0

    def was_item_in_initial_kb(self, item):
        return any(entry['item'] == item for entry in
self.initial_knowledge_base)

    def state_snap(self):
        snapshot = ""
        snapshot += f"Preference: {self.preference}\n"
        return snapshot

    def update_agent_state(self, state):
        self.state.append(state)
```

CogChain.py

```
import os
import re
os.environ["OPENAI_API_KEY"]='sk-
iRdjQCppyWt3WblPYd24T3BlbkFJYsTrZOxSnVoajtnSoI5U'
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
llm = ChatOpenAI(model="gpt-4-0125-preview")
import random
import sys
from CogAgent import CogAgent

class CogChain:
    def __init__(self, cog_agent):
        self.cog_agent = cog_agent
        self.convo_history = []
        self.name = cog_agent.name


    def kb_print(self, kb):
        kb_str = "KNOWLEDGE BASE:\n"
        for item in kb:
            item_str = f"{item['item']}\n\n"
            kb_str += item_str
        return kb_str

    def preference_chain(self, task, kb, decision_options):
        kb_string = self.kb_print(kb)
        update_goal = PromptTemplate(
            input_variables = ["task","updated_kb", "decision_options"],
            template = """{task}. You have access to a list of facts, called a
'knowledge base', to help guide your preference. Here is the most recently
updated list:
            {updated_kb}
            Based on the above facts, choose one alternative from the decision
options. You must choose from the provided options only.
            {decision_options}
            Use this format for your output: PREFERENCE: (Insert candidate name)
\n EXPLANATION: (justify your decision, limit to 300 tokens).""")
        goal_update = LLMChain(llm=llm, prompt=update_goal)
        updated_goal = goal_update.invoke({
        "task": task,
        "updated_kb": kb_string,
        "decision_options": decision_options})
        goal_match = re.search(r'PREFERENCE: (.+)', updated_goal['text'])
        if goal_match:
            new_goal = goal_match.group(1)
            self.cog_agent.preference = new_goal
        return updated_goal["text"]

    def contribution_chain(self, task, context):
        kb = self.cog_agent.knowledge_base
        kb_string = self.kb_print(kb)
        generate_contribution = PromptTemplate(
            input_variables = ["goal", "task", "kb", "context"],
            template = """{task} Your team mate just contributed: {context}.
            It is your turn in the discussion. Choose a fact from the knowledge
base and provide it (exactly as phrased) alongside your commentary. You are
likely to choose a fact that aligns with your preference. You may also directly
address the team member who just contributed, if you want to do so.
            Preference: {goal}
            {kb}
            Output format: INFORMATION ITEM: (insert fact exactly as given) \n
COMMENTARY: (insert your commentary)""")
        gen_contribution = LLMChain(llm=llm, prompt=generate_contribution)
        contribution = gen_contribution.invoke({
            "goal": self.cog_agent.preference,
            "task": task,
            "kb" : kb_string,
            "context": context})
        return contribution['text']
    def process_message(self, task, context, decision_options):
        #Initial Snap
        initial_snap = self.cog_agent.state_snap()
        #Take in Context, Update KB
        self.knowledge_base_chain(context)
        #Update Goal
        goal_result = self.preference_chain(task, self.cog_agent.knowledge_base,
decision_options)
        #Final Snap
        final_snap = self.cog_agent.state_snap()
        #Generate process report
        process_report = f"---------PROCESS REPORT----------\n"
        process_report += f"Initial Agent State: " + initial_snap + "\n"
        process_report += f"***\n"
        process_report += f"Context: " + context + "\n"
        process_report += f"***\n"
        process_report += f"Final Agent State: " + final_snap + "\n"
        process_report += f""+ goal_result + "\n"
        self.cog_agent.update_agent_state(process_report)

    def knowledge_base_chain(self, context):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        # Find if the item is in the knowledge base
        item_found = False
        for item in self.cog_agent.knowledge_base:
            if item['item'] == info_item:
                item['frequency'] += 1
                item_found = True
                break

        # If the item is not found, add it to the knowledge base
        if not item_found:
            self.cog_agent.knowledge_base.append({"item": info_item, "frequency":
1, "initial": False})

        # Use kb_print to get the updated knowledge base string
        updated_kb_str = self.kb_print(self.cog_agent.knowledge_base)
        return updated_kb_str

    def first_turn_contribution(self, discussion_task):
        initial_snap = self.cog_agent.state_snap()
        goal = self.cog_agent.preference
        knowledge_base = self.cog_agent.knowledge_base
        kb_string = self.kb_print(knowledge_base)
        generate_first_turn_contribution = PromptTemplate(
            input_variables = ["goal", "kb", "task", "context"],
            template = """{task}. You are starting the discussion. Choose a fact
from the knowledge base and provide it (exactly as phrased) alongside your
commentary. You are likely to choose a fact that aligns with your preference.
            Preference: {goal}
            {kb}
            Output format: INFORMATION ITEM: (insert fact) \n COMMENTARY: (insert
your commentary)""")
        gen_first_turn_contribution = LLMChain(llm=llm,
prompt=generate_first_turn_contribution)
        contribution = gen_first_turn_contribution.invoke({
            "goal": goal,
            "kb": kb_string,
            "task": discussion_task,})
        updated_kb = self.knowledge_base_chain(contribution['text'])
        final_snap = self.cog_agent.state_snap()
        #Generate state report
        state_report = f"Contribution: " + contribution['text'] + "\n"
        state_report += f"--------------------------------------\n"
        state_report += f"Final Agent State: " + final_snap + "\n"
        self.cog_agent.update_agent_state(state_report)
        return contribution['text']

    def invoke_convo(self, sample_context, discussion_task):
        kb = self.cog_agent.knowledge_base
        goal = self.cog_agent.preference
        contribution_result =
self.contribution_chain(discussion_task,sample_context)
        updated_kb = self.knowledge_base_chain(contribution_result)
        ###GENERATE TURN REPORT
        final_snap = self.cog_agent.state_snap()
        self.cog_agent.update_agent_state(final_snap)
        return contribution_result
```

Conversation.py

```
import os
import re
import random
import sys
from Report import Report
from CogAgent import CogAgent
from CogChain import CogChain

class Conversation:
    def __init__(self, cog_agents, discussion_task, rounds, decision_options,
decision_rule, knowledge_bank):
        self.cog_agents = cog_agents
        self.discussion_task = discussion_task
        self.rounds = rounds
        self.convo_history = []
        self.decision_options = decision_options
        self.decision_rule = decision_rule
        self.knowledge_bank = knowledge_bank
        self.report = Report(cog_agents, knowledge_bank)
        self.revealed_profile = []

    def broadcast_message(self, cog_agents, context, agent_in):
        for agent in cog_agents:
            if agent.name != agent_in:
                agent.process_message(self.discussion_task, context,
self.decision_options)

    def update_revealed_profile(self, context, current_agent, round):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        rp = [{"item": info_item,
            "agent": current_agent.name,
            "round": round}]

        info_item_label = self.report.kb_map(rp, self.knowledge_bank)
        # Add the info_item to the revealed_profile
        self.revealed_profile.append({
            "item": info_item_label[0],
            "agent": current_agent.name,
            "round": round})
        return self.revealed_profile

    def consensus(self):
        # Check for consensus based on the decision_rule
        if self.decision_rule == 'unanimous':
            # Check if all agents have the same goal
            goals = {agent.cog_agent.preference for agent in self.cog_agents}
            if len(goals) == 1:
                return goals.pop()  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        elif self.decision_rule == 'majority':
            # Check if there is a majority agreement on the goal
            goal_counts = {agent.cog_agent.preference: 0 for agent in
self.cog_agents}
            for agent in self.cog_agents:
                goal_counts[agent.cog_agent.preference] += 1

            max_goal = max(goal_counts, key=goal_counts.get)

            if goal_counts[max_goal] > len(self.cog_agents) / 2:
                return max_goal  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        else:
            return 'NO CONSENSUS'

    def next_speaker(self, current_agent, broadcast_agents):
        next_speakers = [agent for agent in broadcast_agents]
        if next_speakers:
            # If more than 2+ agents, choose randomly
            next_speaker = random.choice(next_speakers)
        return next_speaker

    def run_convo(self):
        #initialize agent preferences
        for agent in self.cog_agents:
            # Call preference_chain for each agent
            preference = agent.preference_chain(
                self.discussion_task, agent.cog_agent.knowledge_base,
self.decision_options)
            agent.cog_agent.update_agent_state(preference)

        # Store Initial Profiles
        self.report.initial_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        # Store Initial Preferences
        self.report.initial_preferences =
self.report.retrieve_preferences(self.cog_agents)

        # Initial Turn Contribution
        agent1 = random.choice(self.cog_agents)
        contribution1 = agent1.first_turn_contribution(self.discussion_task)
        agent1.cog_agent.frequency += 1

        # Add first contribution to chat history, ADD ITEM TO REVEALED_PROFILE
        self.convo_history.append(contribution1)
        self.update_revealed_profile(contribution1, agent1, 0)

        # Broadcast first contribution to other agents
        broadcast_agents = [agent for agent in self.cog_agents if agent !=
agent1]
        self.broadcast_message(broadcast_agents, contribution1, agent1.name)

        # Store Turn Preference
        self.report.turn_preferences[0] =
self.report.retrieve_preferences(self.cog_agents)
        # Store Turn Info Profiles
        self.report.turn_info_profiles[0] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        print(agent1.name)
        print(contribution1)
        current_agent = self.next_speaker(agent1, broadcast_agents)
        for round_num in range(1, self.rounds):
            context = self.convo_history[-1]
            contribution = current_agent.invoke_convo(context,
self.discussion_task)
            current_agent.cog_agent.frequency += 1
            print(current_agent.name)
            print(contribution)
            self.convo_history.append(contribution)
            #ADD ITEM TO REVEALED PROFILE
            self.update_revealed_profile(contribution, current_agent, round_num)
            # Broadcast to other agents
            broadcast_agents = [agent for agent in self.cog_agents if agent !=
current_agent]
            self.broadcast_message(broadcast_agents, contribution,
current_agent.name)
            # Store Turn Preference
            self.report.turn_preferences[round_num] =
self.report.retrieve_preferences(self.cog_agents)
             # Store Turn Info Profiles
            self.report.turn_info_profiles[round_num] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
            decision_result = self.consensus()
            if decision_result != 'NO CONSENSUS':
                print(f"Consensus reached! Agreed upon goal: {decision_result}")
                break  # Exit the loop if consensus is reached
            #IF agents do not agree, select next speaker
            current_agent = self.next_speaker(current_agent, broadcast_agents)
        decision_result = self.consensus()
        # Store Final Preferences
        self.report.final_preferences =
self.report.retrieve_preferences(self.cog_agents)
        # Store Final Info Profiles
        self.report.final_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
        # Store Agent Speaker Frequencies
        self.report.speaker_frequency =
self.report.retrieve_agent_frequency(self.cog_agents)
        self.report.revealed_profile = self.revealed_profile
        # Calc & Store non-initial contribution %
        non_initial_percentages = {}
        for agent in self.cog_agents:
            agent_name = agent.cog_agent.name
            non_initial_percentage =
self.report.calculate_non_initial_contributions_percentage(agent_name,
self.revealed_profile)

            if non_initial_percentage is not None:
                non_initial_percentages[agent_name] = non_initial_percentage
        self.report.non_initial_contributions_percentages =
non_initial_percentages
        # Generate Report
        self.report.generate_preference_report(decision_result, self.rounds, 
self.decision_rule)
        self.report.generate_analysis_report(decision_result)
        with open("conversation_log.txt", "w") as log_file:
            log_file.write("----- Conversation Thread -----\n")
            for i, entry in enumerate(self.convo_history, start=1):
                log_file.write(f"Turn {i}:\n{entry}\n")
```

Discuss.py

```
from Conversation import Conversation
from CogAgent import CogAgent
from Report import Report
from CogChain import CogChain

####INFO PROFILE####
# Anders
AG1 = """Astronaut Anders received the highest possible scores on the military 
survival standards, an assessment that all astronauts must complete in order to 
qualify for space missions. Anders' scores were significantly higher than either 
candidate’s scores."""
AG2 = """During Ander’s last ICE (Isolated and Confined in Extreme environments) 
training sequence, a fellow crew member praised him for being able to recognize 
others’ personal needs. In particular, he was recognized for facilitating 
personal time and bringing the group together for light-hearted fun."""
AG3 = """While working at NASA, Anders coordinated and completed a number of 
science projects. This expertise is important to Flight Engineers and one of the 
reasons Anders was chosen for the previous space mission. Anders did not 
disappoint. His ability to set up and coordinate science projects allowed his 
crew to finish more than twice the number of projects than anticipated."""
AG4 = """Anders was one of 8 hikers to successfully summit Mount Everest in 2005. 
Interviews with his comrades suggest Anders was the “glue” that held the group 
together when things went wrong."""
AG5 = """On previous missions, where lack of sleep was a frequent occurrence, 
Anders was largely unaffected. Despite the sub-optimal sleeping patterns, Anders' 
mental and physical health did not show signs of suffering."""
AG6 = """Data collected during training on simulated emergencies shows that 
Anders is able to remain calm during stressful and unexpected situations. He 
performs well under pressure."""
AB1 = """Anders suffered a broken wrist 7 months ago during a training 
simulation. The injury has healed, but will likely require extra exercise to 
prevent forearm muscle atrophy in zero gravity. This may take time away from him 
performing other team duties."""
AB2 = """Most of Anders' space training experience has been limited to 
simulations on Earth, with only one mission to the International Space Station. 
Furthermore, this mission to Mars would be even longer than his only long-
distance flight experience--it is not clear if he will adapt well for a long-
duration mission."""
AB3 = """Anders comes from a close-knit family. In diaries collected during 
training, he reported feeling sad and frustrated when he was unable to 
communicate with family members on a regular basis. This could be a serious
```

```
concern for his mental health on a mission to Mars because there will be very
little contact with Earth, and the little contact there is will be through
mission control."""
AB4 = """Anders easily gets annoyed when explaining concepts to others and
becomes very short with them as a result."""
AN1 = """Anders enjoys science fiction literature and has self-published a novel
about the start of a human colony in a distant galaxy. He hopes to write a sequel
when he retires."""
AN2 = """As a child, Anders enjoyed attending Houstonli-area sporting events with
his parents. He was particularly fond of attending the Texans football games."""
AN3 = """Anders has a pet cat named Buzz, after Buzz Aldrin. Anders met Buzz as a
child around when he decided he wanted to be an astronaut."""
AN4 = """Anders used to own a Brown Labrador Retriever named Bailey. When Anders
participated in astronaut training and missions, Bailey used to stay with his
family."""

# Bean
BG1 = """On Expedition 31/32, where he helped refuel the International Space
Station (ISS), Bean encountered a breakdown in the communication system between
his vehicle and the ISS before arrival to the ISS. Bean was credited for leading
the extravehicular activity that identified and fixed the issue, ultimately
making the refuel possible."""
BG2 = """While it is widely known that Bean holds a bachelor’s degree in
Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a
minor in Counseling Psychology. Crewmembers have reported that Bean is a great
listener and often knows exactly what to say to make others feel more
comfortable."""
BG3 = """Survey responses from Bean’s previous crew mates reveal that he has a
universal sense of humor. People always understand and enjoy Bean’s jokes, so
there is little risk of his humor being misinterpreted as being negative. Crew
members report that this really helps to de-escalate tense situations."""
BB1 = """Bean had a difficult time getting along with some of his co-workers at
Ames. He had asked his supervisor to intervene on several occasions as they were
not able to resolve the disagreements on their own."""
BB2 = """Bean does not always plan his work well, and sometimes his pace of work
is too slow. This has become problematic not only in his mission to the ISS, but
also in training simulations. Often times, the maintenance work and data
collection the astronauts engage in on these missions requires high levels of
interdependency - this can be jeopardized if one person works at a slower pace
than the rest of the crew members."""
BB3 = """Bean has a 2-year-old daughter, and colleagues have noticed he is
frustrated when he has to work late and cannot be home with her. There is worry
that being away for so long on this mission will exponentially increase Bean's
frustration."""
BN1 = """Bean enjoys listening to the Beatles while performing work on his
own."""
BN2 = """Bean is an avid fan of Shakespeare and often quotes his work."""
BN3 = """In college, Bean was on the squash team, where he met his best friend
Jack. Jack now lives in Providence, Rhode Island, but they still talk on a weekly
basis."""
BN4 = """Astronaut Bean enjoys traveling and his favorite place to vacation to is
in the south of France."""
```

```
BN5 = """Bean has five siblings: one older brother, one older sister, and three
younger sisters. He is closest to his younger sister, Rebecca, who is just one
year younger than he is."""
BN6 = """If chosen for this mission, Bean would be the shortest crew member at 5-
feet, 5-inches tall."""

# Collins
CG1 = """Collins is accustomed to being isolated for long periods of time. He
successfully completed two treks to the Antarctic in which he lived in tents and
a small building for the Antarctic winter-over period. Living in small spaces did
not bother him."""
CG2 = """Collins has a passion for languages. He is fluent in English, Spanish,
and French and proficient in Russian. Recently he started a class on Mandarin and
already he is nearly proficient. This ability will help him interact and bond
with international crew members."""
CG3 = """Collins has participated in spacewalks to repair damaged solar panels
and ammonia pump module. His peers have ranked him as the most competent crew
member during their spacewalks."""
CG4 = """Collins received the “Silver Snoopy” award for his contributions on
NASA’s STS-135 mission. This award is considered to be the highest honor of all
of the awards, given to the person who was the most outstanding team member,
creating a team environment focused on flight safety and overall team success."""
CB1 = """Surveys showed that Collins has snapped at coworkers on multiple
occasions. In one situation, a crewmember even went so far as to say would not
want to work with Collins on future missions."""
CB2 = """Collins has the least space-related training of all three potential
candidates."""
CB3 = """Recent blood work indicated that Collins may be developing above-average
blood pressure. Doctors advised another assessment before making any conclusive
determinations. However, if Collins does have heightened blood pressure, he would
have to take extra time to rest while on the mission. This would detract from his
time working on experiments, as well as his ability to do physically stressful
activities."""
CB4 = """Three weeks ago, Collins broke his ankle. As a result, he has not been
training as much and the current state of his cardiovascular performance is
behind compared to the other two candidates."""
CB5 = """In one incident during a mock extravehicular activity scenario, Collins
refused to follow an instruction from mission control. Fellow crew members were
unsure of why he did this since the request from mission control seemed
straightforward and helpful. Crewmembers reported being conflicted with how to
respond, which slowed their progress on the extravehicular activity scenario.
Only half of the tasks needed were completed on the simulated extravehicular
activity."""
CB6 = """An internal incident report indicates that Collins made offensive jokes
to a Chinese astronaut on a previous analog space mission that made the Chinese
astronaut and other crew members uncomfortable."""
CN1 = """Collins considers himself a coffee aficionado and his favorite brand of
coffee is Intelligentsia."""
CN2 = """Collins’ childhood friend completed NASA’s astronaut candidate training
program but was never selected for a space mission."""
CN3 = """Collins enjoys completing Sudoku puzzles to relax"""
CN4 = """Collins has a fraternal twin who works in finance in Houston, Texas."""
```

```
knowledge_bank = [
    {"label": "AG1", "item": """Astronaut Anders received the highest possible
scores on the military survival standards, an assessment that all astronauts must
complete in order to qualify for space missions. Anders' scores were
significantly higher than either candidate’s scores."""},
    {"label": "AG2", "item": """During Ander’s last ICE (Isolated and Confined in
Extreme environments) training sequence, a fellow crew member praised him for
being able to recognize others’ personal needs. In particular, he was recognized
for facilitating personal time and bringing the group together for light-hearted
fun."""},
    {"label": "AG3", "item": """While working at NASA, Anders coordinated and
completed a number of science projects. This expertise is important to Flight
Engineers and one of the reasons Anders was chosen for the previous space
mission. Anders did not disappoint. His ability to set up and coordinate science
projects allowed his crew to finish more than twice the number of projects than
anticipated."""},
    {"label": "AG4", "item": """Anders was one of 8 hikers to successfully summit
Mount Everest in 2005. Interviews with his comrades suggest Anders was the “glue”
that held the group together when things went wrong."""},
    {"label": "AG5", "item": """On previous missions, where lack of sleep was a
frequent occurrence, Anders was largely unaffected. Despite the sub-optimal
sleeping patterns, Anders' mental and physical health did not show signs of
suffering."""},
    {"label": "AG6", "item": """Data collected during training on simulated
emergencies shows that Anders is able to remain calm during stressful and
unexpected situations. He performs well under pressure."""},
    {"label": "AB1", "item": """Anders suffered a broken wrist 7 months ago
during a training simulation. The injury has healed, but will likely require
extra exercise to prevent forearm muscle atrophy in zero gravity. This may take
time away from him performing other team duties."""},
    {"label": "AB2", "item": """Most of Anders' space training experience has
been limited to simulations on Earth, with only one mission to the International
Space Station. Furthermore, this mission to Mars would be even longer than his
only long-distance flight experience--it is not clear if he will adapt well for a
long-duration mission."""},
    {"label": "AB3", "item": """Anders comes from a close-knit family. In diaries
collected during training, he reported feeling sad and frustrated when he was
unable to communicate with family members on a regular basis. This could be a
serious concern for his mental health on a mission to Mars because there will be
very little contact with Earth, and the little contact there is will be through
mission control."""},
    {"label": "AB4", "item": """Anders easily gets annoyed when explaining
concepts to others and becomes very short with them as a result."""},
    {"label": "AN1", "item": """Anders enjoys science fiction literature and has
self-published a novel about the start of a human colony in a distant galaxy. He
hopes to write a sequel when he retires."""},
    {"label": "AN2", "item": """As a child, Anders enjoyed attending Houstonli-
area sporting events with his parents. He was particularly fond of attending the
Texans football games."""},
    {"label": "AN3", "item": """Anders has a pet cat named Buzz, after Buzz
Aldrin. Anders met Buzz as a child around when he decided he wanted to be an
astronaut."""},
    {"label": "AN4", "item": """Anders used to own a Brown Labrador Retriever
named Bailey. When Anders participated in astronaut training and missions, Bailey
used to stay with his family."""},
    {"label": "BG1", "item": """On Expedition 31/32, where he helped refuel the
International Space Station (ISS), Bean encountered a breakdown in the
communication system between his vehicle and the ISS before arrival to the ISS.
Bean was credited for leading the extravehicular activity that identified and
fixed the issue, ultimately making the refuel possible."""},
    {"label": "BG2", "item": """While it is widely known that Bean holds a
bachelor’s degree in Aerospace Engineering and Cross-Cultural studies, few know
that he also pursued a minor in Counseling Psychology. Crewmembers have reported
that Bean is a great listener and often knows exactly what to say to make others
feel more comfortable."""},
    {"label": "BG3", "item": """Survey responses from Bean’s previous crew mates
reveal that he has a universal sense of humor. People always understand and enjoy
Bean’s jokes, so there is little risk of his humor being misinterpreted as being
negative. Crew members report that this really helps to de-escalate tense
situations."""},
    {"label": "BB1", "item": """Bean had a difficult time getting along with some
of his co-workers at Ames. He had asked his supervisor to intervene on several
occasions as they were not able to resolve the disagreements on their own."""},
    {"label": "BB2", "item": """Bean does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "BB3", "item": """Bean has a 2-year-old daughter, and colleagues
have noticed he is frustrated when he has to work late and cannot be home with
her. There is worry that being away for so long on this mission will
exponentially increase Bean's frustration."""},
    {"label": "BN1", "item": """Bean enjoys listening to the Beatles while
performing work on his own."""},
    {"label": "BN2", "item": """Bean is an avid fan of Shakespeare and often
quotes his work."""},
    {"label": "BN3", "item": """In college, Bean was on the squash team, where he
met his best friend Jack. Jack now lives in Providence, Rhode Island, but they
still talk on a weekly basis."""},
    {"label": "BN4", "item": """Astronaut Bean enjoys traveling and his favorite
place to vacation to is in the south of France."""},
    {"label": "BN5", "item": """Bean has five siblings: one older brother, one
older sister, and three younger sisters. He is closest to his younger sister,
Rebecca, who is just one year younger than he is."""},
    {"label": "BN6", "item": """If chosen for this mission, Bean would be the
shortest crew member at 5-feet, 5-inches tall."""},
    {"label": "CG1", "item": """Collins is accustomed to being isolated for long
periods of time. He successfully completed two treks to the Antarctic in which he
lived in tents and a small building for the Antarctic winter-over period. Living
in small spaces did not bother him."""},
    {"label": "CG2", "item": """Collins has a passion for languages. He is fluent
in English, Spanish, and French and proficient in Russian. Recently he started a
class on Mandarin and already he is nearly proficient. This ability will help him
interact and bond with international crew members."""},
    {"label": "CG3", "item": """Collins has participated in spacewalks to repair
damaged solar panels and ammonia pump module. His peers have ranked him as the
most competent crew member during their spacewalks."""},
    {"label": "CG4", "item": """Collins received the “Silver Snoopy” award for
his contributions on NASA’s STS-135 mission. This award is considered to be the
highest honor of all of the awards, given to the person who was the most
outstanding team member, creating a team environment focused on flight safety and
overall team success."""},
    {"label": "CB1", "item": """Surveys showed that Collins has snapped at
coworkers on multiple occasions. In one situation, a crewmember even went so far
as to say would not want to work with Collins on future missions."""},
    {"label": "CB2", "item": """Collins does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "CB3", "item": """Recent blood work indicated that Collins may be
developing above-average blood pressure. Doctors advised another assessment
before making any conclusive determinations. However, if Collins does have
heightened blood pressure, he would have to take extra time to rest while on the
mission. This would detract from his time working on experiments, as well as his
ability to do physically stressful activities."""},
    {"label": "CB4", "item": """Three weeks ago, Collins broke his ankle. As a
result, he has not been training as much and the current state of his
cardiovascular performance is behind compared to the other two candidates."""},
    {"label": "CB5", "item": """In one incident during a mock extravehicular
activity scenario, Collins refused to follow an instruction from mission control.
Fellow crew members were unsure of why he did this since the request from mission
control seemed straightforward and helpful. Crewmembers reported being conflicted
with how to respond, which slowed their progress on the extravehicular activity
scenario. Only half of the tasks needed were completed on the simulated
extravehicular activity."""},
    {"label": "CB6", "item": """An internal incident report indicates that
Collins made offensive jokes to a Chinese astronaut on a previous analog space
mission that made the Chinese astronaut and other crew members
uncomfortable."""},
    {"label": "CN1", "item": """Collins considers himself a coffee aficionado and
his favorite brand of coffee is Intelligentsia."""},
    {"label": "CN2", "item": """Collins’ childhood friend completed NASA’s
astronaut candidate training program but was never selected for a space
mission."""},
    {"label": "CN3", "item": """Collins enjoys completing Sudoku puzzles to
relax"""},
    {"label": "CN4", "item": """Collins has a fraternal twin who works in finance
in Houston, Texas."""}
]
```

```
#####AGENT 1: CMR######
cmr_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG6, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True}
]
cmr = CogAgent("CMR", cmr_KB)
cog_cmr = CogChain(cmr)

#####AGENT 2: MS1#####
ms1_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CB3, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CB5, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": CN4, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN4, "frequency" : 0 ,"initial": True},
    {"item": AG1, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True}
]
ms1 = CogAgent("MS1", ms1_KB)
cog_ms1 = CogChain(ms1)

#####AGENT 3: MS2#####
ms2_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
ms2 = CogAgent("MS2", ms2_KB)
cog_ms2 = CogChain(ms2)

#####AGENT 4: FE######
fe_KB = [
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CB2, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN5, "frequency" : 0 ,"initial": True},
    {"item": BN6, "frequency" : 0 ,"initial": True},
    {"item": AG5, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
fe = CogAgent("FE", fe_KB)
cog_fe = CogChain(fe)

cog_agents = [cog_cmr, cog_ms1, cog_ms2, cog_fe]
rounds = 10
decision_options = """Candidate 1: Samuel Anders
Candidate 2: John Bean
Candidate 3: Scott Collins"""
decision_rule = 'majority'
discussion_task = """ DISCUSSION TASK: You work as a member of NASA's Astronaut
Crew Composition team. Your team analyzes all of the information obtained during
astronaut selection and training - including test scores, analog performance
measures, expert observations, peer evaluations, and medical  records. Armed with
this information, your team makes two kinds of evaluations. The first evaluation
is to determine which astronauts are ready for flight and which are not. Of those
who are deemed to be flight ready, your team composes entire crews for each
mission. In this second evaluation, you consider the complementarity and
compatibility of the astronauts as a team to select the 4, 5, or 6  astronauts
(depending on the mission) who will perform best as individuals, and as a crew.
While composing crews for missions on the International Space Station has become
routine, your team just received a special assignment from the Mission Planning
Directorate that is anything but routine.
The Mars mission has been fast-tracked. Instead of the 2033 launch date that NASA
had been aiming for, the launch date is being moved up to 2020. Intensive crew
training for this unprecedented and bold journey must begin immediately. A 5-
member crew will embark on this 3 year journey to Mars. This a truly
international endeavor: the Russian, European, Indian, and Chinese Space Agencies
will each select one astronaut for the mission. NASA will select the 5th crew
member - an American astronaut - to join this international crew.
The crew members from the other countries have backgrounds in engineering,
botany, geology and medicine. The crew still needs a flight engineer.
Your team was able to quickly narrow the field down to three veteran astronauts:
Anders, Bean, and Collins. Each of these three have experience as a flight
engineer on the ISS, have the necessary basic technical skills for the flight
engineer position, meet basic requirements for the mission including height, Body
Mass Index, and have passed a mental health screening. Now that you know the
other potential crew members it is necessary for your team to determine which of
the crew members will be the greatest asset on this particular mission.
The 3-year mission to Mars will be demanding. The success of the Mars mission
will require the 5-person crew to live and work together in extremely close
quarters, while isolated from the usual sources of social support - friends and
family - for extended periods of time. It will require that the crew can adapt to
the unknown and be able to handle whatever challenges will come their way. Once
they arrive at Mars, the crew will have to complete complex feats such as landing
and extravehicular activities. Because of a significant communication delay with
```

```
Earth of up to 22 minutes each way, the crew will have to work autonomously.
Additionally, the crew of the Mars mission consists of individuals representing
multiple nationalities, making strong interpersonal skills all the more critical
to the success of the mission.
The international space consortium has agreed to announce the names of the crew
members from each of the 5 countries in just 10 days from now. Your team must
deliver a recommendation for whom should be the American astronaut; years of hard
work (and millions of dollars) rest on this decision.
To guide your difficult decision, you have compiled information from the
personnel file for each candidate. The personnel files for each candidate are
detailed and include information such as their performance scores and ranking on
various training tasks, biographical information, evaluations and questionnaires
completed by  supervisors and individuals with whom they worked on prior
trainings and missions, and questionnaires completed by the candidate and his
family. A few human resource assistant have been working through the personnel
files and have prepared briefs for you. Your team must now come together and
discuss who should be selected for the mission.
Remember, the success of the mission to Mars depends on your team. Choose
quickly, but wisely!
"""
astronaut_candidate = Conversation(cog_agents, discussion_task, rounds,
decision_options, decision_rule, knowledge_bank)

astronaut_candidate.run_convo()

with open("agent_states.txt", "w") as states_file:
    for agent in astronaut_candidate.cog_agents:
        states_file.write(f"----- {agent.cog_agent.name} States -----\n")
        for element in agent.cog_agent.state:
            states_file.write(f"{element}\n")
        states_file.write("\n")  # Add a newline after each agent's states
```

Report.py

```
from CogAgent import CogAgent
from CogChain import CogChain

class Report:
    def __init__(self, cog_agents, knowledge_bank):
        self.initial_preferences = []
        self.turn_preferences = {}
        self.final_preferences = []
        self.turn_info_profiles = {}
        self.initial_info_profiles = []
        self.final_info_profiles = []
        self.speaker_frequency = []
        self.revealed_profile = []
        self.non_initial_contributions_percentages = []
        self.cog_agents = cog_agents
        self.knowledge_bank = knowledge_bank

    def retrieve_preferences(self, cog_agents):
        preferences = [f"{agent.cog_agent.name}: {agent.cog_agent.preference}"
for agent in cog_agents]
        return preferences

    def aggregate_round_profile(self, revealed_profile):
        aggregated_profile = {}
        all_shared_items = set()  # To keep track of all items shared in the
rounds
        item_frequencies = {}  # To track the frequency of each item

        for entry in revealed_profile:
            round_num = entry["round"]
            item = entry["item"]
            agent = entry["agent"]  # Include the contributing agent name

            if round_num not in aggregated_profile:
                aggregated_profile[round_num] = [(item, agent)]
            else:
                aggregated_profile[round_num].append((item, agent))

            all_shared_items.add(item)  # Add the item to the set

            # Update the frequency of the item
            item_frequencies[item] = item_frequencies.get(item, 0) + 1

        # Prepare a string to be used in the report
        result_str = "\n----- Turn Group Revealed Profile -----\n"
        for round_num, items_agents in sorted(aggregated_profile.items()):
            result_str += f"Round {round_num}:\n"
            for item, agent in items_agents:
                result_str += f"Info Item: {item}\nContributing Agent: {agent}\n"
            result_str += "\n"
        result_str += "\n----- Final Group Revealed Profile -----\n"
        # Include a list of all items shared in the rounds with frequency
        for item in all_shared_items:
            result_str += f"{item} (Frequency: {item_frequencies[item]})\n"
        return result_str

    def retrieve_agent_frequency(self, cog_agents):
        speaker_frequency = [f"{agent.cog_agent.name}:
{agent.cog_agent.frequency}" for agent in cog_agents]
        return speaker_frequency

    def generate_analysis_report(self, discussion_result):
        with open("analysis_report.txt", "w") as analysis_file:
            analysis_file.write("********************************************\n"
)
            analysis_file.write("ANALYSIS REPORT\n")
            analysis_file.write("********************************************\n"
)
```

```
            # 1. Calculate percentage of revealed profile items that are in the
knowledge bank
            all_shared_items = self.extract_all_shared_items()
            revealed_profile_percentage =
self.calculate_revealed_profile_percentage(all_shared_items)
            analysis_file.write("REVEALED PROFILE:\n")
            analysis_file.write(f"{revealed_profile_percentage:.2f}%\n\n")

            # 2. Calculate percentage of preference changes
            preference_change_percentage = self.calculate_preference_changes()
            analysis_file.write("PREFERENCE CHANGES:\n")
            analysis_file.write(f"{preference_change_percentage:.2f}%\n\n")

            # 3. Speaker distribution
            speaker_distribution = self.calculate_speaker_distribution()
            analysis_file.write("SPEAKER DISTRIBUTION:\n")
            for agent, percentage in speaker_distribution.items():
                analysis_file.write(f"{agent}: {percentage:.2f}%\n")
            analysis_file.write("\n")

            # 4. Decision made and group optimality rating
            if discussion_result != 'NO CONSENSUS':
                # decision_made, group_optimality =
self.calculate_decision_optimality(all_shared_items)
                group_optimality_revealed_profile =
self.calculate_revealed_profile_optimality(all_shared_items, discussion_result)
                group_optimality_knowledge_bank =
self.calculate_knowledge_bank_optimality(discussion_result)
                analysis_file.write("DECISION MADE:\n")
                analysis_file.write(f"Decision: {discussion_result}\n")
                analysis_file.write(f"Group Optimality (Revealed Profile):
{group_optimality_revealed_profile}\n")
                analysis_file.write(f"Group Optimality (Knowledge Bank):
{group_optimality['knowledge_bank']}\n")

                # Corrected individual optimality ratings

            else:
                analysis_file.write("No group decision made.\n")
                analysis_file.write("\nIndividual Optimality:\n")
                for agent_pref in self.final_preferences:
                    agent_name, preference = agent_pref.split(": ")
                    revealed_profile_optimality =
self.calculate_revealed_profile_optimality(all_shared_items, preference)
                    knowledge_bank_optimality =
self.calculate_knowledge_bank_optimality(preference)
                    analysis_file.write(f"{agent_name}:\n")
                    analysis_file.write(f"Revealed Profile:
{revealed_profile_optimality}\n")
                    analysis_file.write(f"Knowledge Bank:
{knowledge_bank_optimality}\n")
```

```
    def extract_all_shared_items(self):
        all_shared_items = set()
        for entry in self.revealed_profile:
            label = entry['item']
            for item in self.knowledge_bank:
                if item['label'] == label:
                    all_shared_items.add(item['label'])
                    break  # Stop searching once the item is found
        return all_shared_items

    def calculate_revealed_profile_percentage(self, all_shared_items):
    # Calculate the percentage of unique items in the revealed profile also in
the knowledge bank
        unique_revealed_items = set(entry['item'] for entry in
self.revealed_profile)
        total_revealed_items = len(unique_revealed_items)

        if total_revealed_items == 0:
            return 0.0

        kb_items = [entry['label'] for entry in self.knowledge_bank]
        kb_length = len(kb_items)

        percentage = (total_revealed_items / kb_length) * 100
        return percentage

    def calculate_preference_changes(self):
        initial_preferences = [agent.split(": ")[1] for agent in
self.initial_preferences]
        final_preferences = [agent.split(": ")[1] for agent in
self.final_preferences]
        changed_agents = [initial != final for initial, final in
zip(initial_preferences, final_preferences)]
        return (sum(changed_agents) / len(self.cog_agents)) * 100

    def calculate_speaker_distribution(self):
        total_turns = sum([int(freq.split(": ")[1]) for freq in
self.speaker_frequency])
        speaker_distribution = {freq.split(": ")[0]: (int(freq.split(": ")[1]) /
total_turns) * 100 for freq in self.speaker_frequency}
        return speaker_distribution

    def calculate_revealed_profile_optimality(self, all_shared_items,
agent_preference):
        ####FIX
        preference_counts = {'C': 0, 'A': 0, 'B': 0}
        for item in all_shared_items:
        # Check if the second item letter is 'G'
            if item[1] == 'G':
                # Determine the preference of the first item and update
preference counts
                if item[0] == 'C':
                    preference_counts['C'] += 1
                elif item[0] == 'A':
                    preference_counts['A'] += 1
                elif item[0] == 'B':
                    preference_counts['B'] += 1
        preference_ranking = {'C': 0, 'A': 0, 'B': 0}
        for preference, count in preference_counts.items():
            preference_ranking[preference] = count
        sorted_preferences = sorted(preference_ranking.items(), key=lambda x:
x[1], reverse=True)
        max_count = sorted_preferences[0][1]
        max_count_preferences = sum(1 for pref, count in sorted_preferences if
count == max_count)
        ranks = {}
        if max_count_preferences == 1:  # Only one preference has the maximum
count
            for preference, count in sorted_preferences:
                ranks[preference] = 'BEST' if count == max_count else 'WORST'
        else:  # Multiple preferences have the maximum count
            for preference, count in sorted_preferences:
                ranks[preference] = 'AVERAGE' if count == max_count else 'WORST'

            # Calculate second highest count
        second_highest_count = sorted_preferences[1][1]
            # Calculate how many preferences have the second highest count
        second_highest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == second_highest_count)

            # Assign 'AVERAGE' rank to preferences with the second highest count
        if second_highest_count_preferences > 1:
            for preference, count in sorted_preferences:
                if count == second_highest_count:
                    ranks[preference] = 'AVERAGE'

            # Calculate lowest count
        lowest_count = sorted_preferences[-1][1]

            # If lowest count is different from second highest count, assign
'WORST' rank
        if lowest_count != second_highest_count:
            lowest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == lowest_count)
            if lowest_count_preferences > 1:
                for preference, count in sorted_preferences:
                    if count == lowest_count:
                        ranks[preference] = 'WORST'

            # Extract agent preference label
        agent_preference_label = agent_preference.split()[1][0]

            # Assign the agent's preference ranking
        agent_preference_ranking = ranks.get(agent_preference_label, None)
        return agent_preference_ranking

    def calculate_knowledge_bank_optimality(self, preference):
        # Define the fixed knowledge bank optimality ratings
        knowledge_bank_ratings = {'Samuel Anders': 'BEST', 'John Bean':
'AVERAGE', 'Scott Collins': 'WORST'}
        return knowledge_bank_ratings.get(preference, 'UNKNOWN')

    def generate_preference_report(self, discussion_result, rounds,
decision_rule):
        with open("report.txt", "w") as report_file:
            report_file.write("********************************************\n")
            report_file.write("DISCUSSION REPORT\n")
            report_file.write("********************************************\n")
            report_file.write(f"Number of Rounds: {rounds}\n")
            report_file.write(f"Decision Rule: {decision_rule}\n")
            report_file.write
            if discussion_result == 'NO CONSENSUS':
                report_file.write(f"Discussion Result: Failure to reach consensus
in the specified number of rounds.\n")
            else:
                report_file.write(f"Discussion Result: Consensus reached in
{rounds} rounds!\n")
                report_file.write(f"Agreed upon goal: {discussion_result}\n")
            report_file.write("\n********************************************\n"
)
            report_file.write("PREFERENCE INDICATORS\n")
            report_file.write("********************************************\n")
            report_file.write("----- Initial Preference Distribution -----\n")
            for i, entry in enumerate(self.initial_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("----- Turn Preference Distribution -----\n")
            for round_num, preferences in self.turn_preferences.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(preferences, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Preference Distribution -----\n")
            for i, entry in enumerate(self.final_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n********************************************\n"
)
            report_file.write("INFO ITEM INDICATORS\n")
            report_file.write("********************************************\n")
            report_file.write("\n----- Initial Info Profiles -----\n")
            for i, entry in enumerate(self.initial_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n----- Turn Info Profiles -----\n")
            for round_num, info_profile in self.turn_info_profiles.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(info_profile, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Info Profiles -----\n")
            for i, entry in enumerate(self.final_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            aggregated_profile_str =
self.aggregate_round_profile(self.revealed_profile)
            report_file.write(f"{aggregated_profile_str}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("CONVO INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("-----Speaker Frequency -----\n")
            for i, entry in enumerate(self.speaker_frequency, start=1):
                report_file.write(f"{entry}\n")
            report_file.write("\n----- Non-Initial Contributions-----\n")
            for agent_name, percentage in
self.non_initial_contributions_percentages.items():
                report_file.write(f"{agent_name}: {percentage}%\n")

    def kb_map(self, knowledge_base, knowledge_bank):
        labels = []
        for kb_entry in knowledge_base:
            kb_item = kb_entry["item"]
            for bank_entry in knowledge_bank:
                bank_item = bank_entry["item"]
                if kb_item == bank_item:
                    labels.append(bank_entry["label"])
                    break  # Stop searching once a match is found
        return labels

    def retrieve_info_profiles(self, cog_agents, knowledge_bank):
        profiles = []
        for agent in cog_agents:
            knowledge_base = agent.cog_agent.knowledge_base
            profile_labels = self.kb_map(knowledge_base, knowledge_bank)
            profiles += [f"{agent.cog_agent.name}: {profile_labels}"]
        return profiles

    def calculate_non_initial_contributions_percentage(self, agent_name,
revealed_profile):
        agent_contributions = [entry for entry in revealed_profile if
entry['agent'] == agent_name]
        total_contributions = len(agent_contributions)

        if total_contributions == 0:
            return None  # Return None for agents who did not contribute

        non_initial_contributions = sum(
            1 for entry in agent_contributions if not
self.is_item_in_initial_kb(agent_name, entry['item']))

        if total_contributions == non_initial_contributions:
            return 0.0  # All contributions were non-initial

        percentage = (non_initial_contributions / (total_contributions -
non_initial_contributions)) * 100
        return round(percentage, 2)

    def is_item_in_initial_kb(self, agent_name, item):
        for agent in self.cog_agents:
            if agent.cog_agent.name == agent_name:
                return agent.cog_agent.was_item_in_initial_kb(item)
        return False
```

APPENDIX D: CONTRIBUTIONCOG CODE

CogAgent.py

```
import random

class CogAgent:
    def __init__(self, name, knowledge_base):
        self.name = name
        self.knowledge_base = knowledge_base
        self.initial_knowledge_base = knowledge_base.copy()
        self.preference = ""
        self.state = []
        self.frequency = 0
        self.contribute = ""

    def was_item_in_initial_kb(self, item):
        return any(entry['item'] == item for entry in
self.initial_knowledge_base)

    def state_snap(self):
        snapshot = ""
        snapshot += f"Preference: {self.preference}\n"
        return snapshot

    def update_agent_state(self, state):
        self.state.append(state)
```

CogChain.py

```
import os
import re
os.environ["OPENAI_API_KEY"]='sk-
iRdjQCppyWt3WblPYd24T3BlbkFJYsTrZOxSnVoajtnSoI5U'
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
llm = ChatOpenAI(model="gpt-4-0125-preview")
import random
import sys
from CogAgent import CogAgent
```

```
class CogChain:
    def __init__(self, cog_agent):
        self.cog_agent = cog_agent
        self.convo_history = []
        self.name = cog_agent.name


    def kb_print(self, kb):
        kb_str = "KNOWLEDGE BASE:\n"
        for item in kb:
            item_str = f"{item['item']}\n\n"
            kb_str += item_str
        return kb_str

    def preference_chain(self, task, kb, decision_options):
        kb_string = self.kb_print(kb)
        update_goal = PromptTemplate(
            input_variables = ["task","updated_kb", "decision_options"],
            template = """{task}. You have access to a list of facts, called a
'knowledge base', to help guide your preference. Here is the most recently
updated list:
            {updated_kb}
            Based on the above facts, choose one alternative from the decision
options. You must choose from the provided options only.
            {decision_options}
            Use this format for your output: PREFERENCE: (Insert candidate name)
\n EXPLANATION: (justify your decision, limit to 300 tokens).""")
        goal_update = LLMChain(llm=llm, prompt=update_goal)
        updated_goal = goal_update.invoke({
        "task": task,
        "updated_kb": kb_string,
        "decision_options": decision_options})
        goal_match = re.search(r'PREFERENCE: (.+)', updated_goal['text'])
        if goal_match:
            new_goal = goal_match.group(1)
            self.cog_agent.preference = new_goal
        return updated_goal["text"]

    def contribution_chain(self, task, context):
        kb = self.cog_agent.knowledge_base
        kb_string = self.kb_print(kb)
        generate_contribution = PromptTemplate(
            input_variables = ["goal", "task", "kb", "context"],
            template = """{task} Your team mate just contributed: {context}.
```

```
                It is your turn in the discussion. Choose a fact from the
knowledge base and provide it (exactly as phrased) alongside your response. You
are likely to choose a fact that aligns with your preference. Your must directly
respond to the contribution made by your teammate. Do not discuss any other fact
from your knowledge base, aside from your chosen fact and your teammates
contribution, in your response.
                Preference: {goal}
                {kb}
                Output format: INFORMATION ITEM: (insert fact exactly as given)
\n COMMENTARY: (insert response)""")
        gen_contribution = LLMChain(llm=llm, prompt=generate_contribution)
        contribution = gen_contribution.invoke({
            "goal": self.cog_agent.preference,
            "task": task,
            "kb" : kb_string,
            "context": context})
        return contribution['text']

    def contribute_query(self, task, context):
        contribute_query = PromptTemplate(
            input_variables = ["task", "context", "preference"],
            template = """{task}. Your teammate just contributed: {context}
            Determine whether you would like to respond to this contribution
based on your preference.
            Preference: {preference}
            Use this format for your output: CONTRIBUTE: (insert YES/NO) \n
EXPLANATION: (explain why you want to respond to your teammate (max 150
tokens)""")
        query = LLMChain(llm=llm, prompt=contribute_query)
        query_result = query.invoke({
            "task" : task,
            "context": context,
            "preference": self.cog_agent.preference
            })
        query_match = re.search(r'CONTRIBUTE: (.+)', query_result['text'])
        if query_match:
            yesORno = query_match.group(1)
            self.cog_agent.contribute = yesORno
        return query_result["text"]

    def process_message(self, task, context, decision_options):
        #Initial Snap
        initial_snap = self.cog_agent.state_snap()
        #Take in Context, Update KB
        self.knowledge_base_chain(context)
        #Update Goal
        goal_result = self.preference_chain(task, self.cog_agent.knowledge_base, 
decision_options)
        #Final Snap
        final_snap = self.cog_agent.state_snap()
        contribute = self.contribute_query(task, context)
        #Generate process report
        process_report = f"---------PROCESS REPORT----------\n"
        process_report += f"Initial Agent State: " + initial_snap + "\n"
        process_report += f"***\n"
        process_report += f"Context: " + context + "\n"
        process_report += f"***\n"
        process_report += f"Final Agent State: " + final_snap + "\n"
        process_report += f""+ goal_result + "\n"
        process_report += f"Want to Respond?\n"
        process_report += f""+ contribute + "\n"
        self.cog_agent.update_agent_state(process_report)

    def knowledge_base_chain(self, context):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        # Find if the item is in the knowledge base
        item_found = False
        for item in self.cog_agent.knowledge_base:
            if item['item'] == info_item:
                item['frequency'] += 1
                item_found = True
                break

        # If the item is not found, add it to the knowledge base
        if not item_found:
            self.cog_agent.knowledge_base.append({"item": info_item, "frequency": 
1, "initial": False})

        # Use kb_print to get the updated knowledge base string
        updated_kb_str = self.kb_print(self.cog_agent.knowledge_base)
        return updated_kb_str

    def first_turn_contribution(self, discussion_task):
        initial_snap = self.cog_agent.state_snap()
        goal = self.cog_agent.preference
        knowledge_base = self.cog_agent.knowledge_base
        kb_string = self.kb_print(knowledge_base)
        generate_first_turn_contribution = PromptTemplate(
            input_variables = ["goal", "kb", "task", "context"],
            template = """{task}. You are starting the discussion. Choose a fact 
from the knowledge base and provide it (exactly as phrased) alongside your 
commentary. You are likely to choose a fact that aligns with your preference.
            Preference: {goal}
            {kb}
            Output format: INFORMATION ITEM: (insert fact) \n COMMENTARY: (insert 
your commentary)""")
        gen_first_turn_contribution = LLMChain(llm=llm, 
prompt=generate_first_turn_contribution)
        contribution = gen_first_turn_contribution.invoke({
            "goal": goal,
            "kb": kb_string,
            "task": discussion_task})
        #updated_kb = self.knowledge_base_chain(contribution['text'])
        final_snap = self.cog_agent.state_snap()
        #Generate state report
        state_report = f"Contribution: " + contribution['text'] + "\n"
        state_report += f"--------------------------------------\n"
        state_report += f"Final Agent State: " + final_snap + "\n"
        self.cog_agent.update_agent_state(state_report)
        return contribution['text']

    def invoke_convo(self, sample_context, discussion_task):
        kb = self.cog_agent.knowledge_base
        goal = self.cog_agent.preference
        contribution_result = 
self.contribution_chain(discussion_task,sample_context)
        #updated_kb = self.knowledge_base_chain(contribution_result)
        ###GENERATE TURN REPORT
        final_snap = self.cog_agent.state_snap()
        self.cog_agent.update_agent_state(final_snap)
        return contribution_result
```

Conversation.py

```
import os
import re
import random
import sys
```

```
from Report import Report
from CogAgent import CogAgent
from CogChain import CogChain

class Conversation:
    def __init__(self, cog_agents, discussion_task, rounds, decision_options,
decision_rule, knowledge_bank):
        self.cog_agents = cog_agents
        self.discussion_task = discussion_task
        self.rounds = rounds
        self.convo_history = []
        self.decision_options = decision_options
        self.decision_rule = decision_rule
        self.report = Report(cog_agents, knowledge_bank)
        self.revealed_profile = []
        self.knowledge_bank = knowledge_bank

    def broadcast_message(self, cog_agents, context, agent_in):
        for agent in cog_agents:
            if agent.name != agent_in:
                agent.process_message(self.discussion_task, context,
self.decision_options)

    def update_revealed_profile(self, context, current_agent, round):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        rp = [{"item": info_item,
            "agent": current_agent.name,
            "round": round}]

        info_item_label = self.report.kb_map(rp, self.knowledge_bank)
        # Add the info_item to the revealed_profile
        self.revealed_profile.append({
            "item": info_item_label[0],
            "agent": current_agent.name,
            "round": round})
        return self.revealed_profile

    def consensus(self):
        # Check for consensus based on the decision_rule
        if self.decision_rule == 'unanimous':
            # Check if all agents have the same goal
            goals = {agent.cog_agent.preference for agent in self.cog_agents}
            if len(goals) == 1:
                return goals.pop()  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        elif self.decision_rule == 'majority':
            # Check if there is a majority agreement on the goal
            goal_counts = {agent.cog_agent.preference: 0 for agent in
self.cog_agents}
            for agent in self.cog_agents:
                goal_counts[agent.cog_agent.preference] += 1

            max_goal = max(goal_counts, key=goal_counts.get)

            if goal_counts[max_goal] > len(self.cog_agents) / 2:
                return max_goal  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        else:
            return 'NO CONSENSUS'

    def next_speaker(self, current_agent, broadcast_agents):
        # Check if there is at least one agent with a 'YES' contribute value
        next_speakers = [agent for agent in broadcast_agents if
agent.cog_agent.contribute == 'YES']
        if next_speakers:
            # If more than one agent with 'YES', choose randomly
            next_speaker = random.choice(next_speakers)
            return next_speaker
        else:
            return None  # No agent with 'YES' contribute value

    def run_convo(self):
        #initialize agent preferences
        for agent in self.cog_agents:
            # Call preference_chain for each agent
            preference = agent.preference_chain(
                self.discussion_task, agent.cog_agent.knowledge_base,
self.decision_options)
            agent.cog_agent.update_agent_state(preference)

        # Store Initial Profiles
        self.report.initial_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        # Store Initial Preferences
        self.report.initial_preferences =
self.report.retrieve_preferences(self.cog_agents)

        # Initial Turn Contribution
        agent1 = random.choice(self.cog_agents)
        contribution1 = agent1.first_turn_contribution(self.discussion_task)
        agent1.cog_agent.frequency += 1

        # Add first contribution to chat history, ADD ITEM TO REVEALED_PROFILE
        self.convo_history.append(contribution1)
        self.update_revealed_profile(contribution1, agent1, 0)

        # Broadcast first contribution to other agents
        broadcast_agents = [agent for agent in self.cog_agents if agent !=
agent1]
        self.broadcast_message(broadcast_agents, contribution1, agent1.name)

        # Store Turn Preference
        self.report.turn_preferences[0] =
self.report.retrieve_preferences(self.cog_agents)
        # Store Turn Info Profiles
        self.report.turn_info_profiles[0] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        print(agent1.name)
        print(contribution1)
        current_agent = self.next_speaker(agent1, broadcast_agents)

        for round_num in range(1, self.rounds):
            context = self.convo_history[-1]
            contribution = current_agent.invoke_convo(context,
self.discussion_task)
            current_agent.cog_agent.frequency += 1
            print(current_agent.name)
            print(contribution)
            self.convo_history.append(contribution)
            #ADD ITEM TO REVEALED PROFILE
            self.update_revealed_profile(contribution, current_agent, round_num)
            # Broadcast to other agents
            broadcast_agents = [agent for agent in self.cog_agents if agent !=
current_agent]
```

```
            self.broadcast_message(broadcast_agents, contribution,
current_agent.name)
            # Store Turn Preference
            self.report.turn_preferences[round_num] =
self.report.retrieve_preferences(self.cog_agents)
             # Store Turn Info Profiles
            self.report.turn_info_profiles[round_num] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
            decision_result = self.consensus()
            if decision_result != 'NO CONSENSUS':
                print(f"Consensus reached! Agreed upon goal: {decision_result}")
                break  # Exit the loop if consensus is reached
            #IF agents do not agree, select next speaker
            current_agent = self.next_speaker(current_agent, broadcast_agents)
            if current_agent is None:
                print(f"DISCUSSION FAILURE: No agent wants to contribute, and no
consensus has been reached in {round_num} rounds")
                break
        decision_result = self.consensus()
        # Store Final Preferences
        self.report.final_preferences =
self.report.retrieve_preferences(self.cog_agents)
        # Store Final Info Profiles
        self.report.final_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
        # Store Agent Speaker Frequencies
        self.report.speaker_frequency =
self.report.retrieve_agent_frequency(self.cog_agents)
        self.report.revealed_profile = self.revealed_profile
        # Calc & Store non-initial contribution %
        non_initial_percentages = {}
        for agent in self.cog_agents:
            agent_name = agent.cog_agent.name
            non_initial_percentage =
self.report.calculate_non_initial_contributions_percentage(agent_name,
self.revealed_profile)

            if non_initial_percentage is not None:
                non_initial_percentages[agent_name] = non_initial_percentage
        self.report.non_initial_contributions_percentages =
non_initial_percentages
        # Generate Report
        self.report.generate_preference_report(decision_result, self.rounds,
self.decision_rule)
        self.report.generate_analysis_report(decision_result)
        with open("conversation_log.txt", "w") as log_file:
            log_file.write("----- Conversation Thread -----\n")
            for i, entry in enumerate(self.convo_history, start=1):
                log_file.write(f"Turn {i}:\n{entry}\n")
```

Discuss.py

```
from Conversation import Conversation
from CogAgent import CogAgent
from Report import Report
from CogChain import CogChain

####INFO PROFILE####
# Anders
AG1 = """Astronaut Anders received the highest possible scores on the military
survival standards, an assessment that all astronauts must complete in order to
qualify for space missions. Anders' scores were significantly higher than either
candidate’s scores."""
AG2 = """During Ander’s last ICE (Isolated and Confined in Extreme environments)
training sequence, a fellow crew member praised him for being able to recognize
others’ personal needs. In particular, he was recognized for facilitating
personal time and bringing the group together for light-hearted fun."""
AG3 = """While working at NASA, Anders coordinated and completed a number of
science projects. This expertise is important to Flight Engineers and one of the
reasons Anders was chosen for the previous space mission. Anders did not
disappoint. His ability to set up and coordinate science projects allowed his
crew to finish more than twice the number of projects than anticipated."""
AG4 = """Anders was one of 8 hikers to successfully summit Mount Everest in 2005.
Interviews with his comrades suggest Anders was the “glue” that held the group
together when things went wrong."""
AG5 = """On previous missions, where lack of sleep was a frequent occurrence,
Anders was largely unaffected. Despite the sub-optimal sleeping patterns, Anders'
mental and physical health did not show signs of suffering."""
AG6 = """Data collected during training on simulated emergencies shows that
Anders is able to remain calm during stressful and unexpected situations. He
performs well under pressure."""
AB1 = """Anders suffered a broken wrist 7 months ago during a training
simulation. The injury has healed, but will likely require extra exercise to
prevent forearm muscle atrophy in zero gravity. This may take time away from him
performing other team duties."""
AB2 = """Most of Anders' space training experience has been limited to
simulations on Earth, with only one mission to the International Space Station.
Furthermore, this mission to Mars would be even longer than his only long-
```

```
distance flight experience--it is not clear if he will adapt well for a long-
duration mission."""
AB3 = """Anders comes from a close-knit family. In diaries collected during
training, he reported feeling sad and frustrated when he was unable to
communicate with family members on a regular basis. This could be a serious
concern for his mental health on a mission to Mars because there will be very
little contact with Earth, and the little contact there is will be through
mission control."""
AB4 = """Anders easily gets annoyed when explaining concepts to others and
becomes very short with them as a result."""
AN1 = """Anders enjoys science fiction literature and has self-published a novel
about the start of a human colony in a distant galaxy. He hopes to write a sequel
when he retires."""
AN2 = """As a child, Anders enjoyed attending Houstonli-area sporting events with
his parents. He was particularly fond of attending the Texans football games."""
AN3 = """Anders has a pet cat named Buzz, after Buzz Aldrin. Anders met Buzz as a
child around when he decided he wanted to be an astronaut."""
AN4 = """Anders used to own a Brown Labrador Retriever named Bailey. When Anders
participated in astronaut training and missions, Bailey used to stay with his
family."""

# Bean
BG1 = """On Expedition 31/32, where he helped refuel the International Space
Station (ISS), Bean encountered a breakdown in the communication system between
his vehicle and the ISS before arrival to the ISS. Bean was credited for leading
the extravehicular activity that identified and fixed the issue, ultimately
making the refuel possible."""
BG2 = """While it is widely known that Bean holds a bachelor’s degree in
Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a
minor in Counseling Psychology. Crewmembers have reported that Bean is a great
listener and often knows exactly what to say to make others feel more
comfortable."""
BG3 = """Survey responses from Bean’s previous crew mates reveal that he has a
universal sense of humor. People always understand and enjoy Bean’s jokes, so
there is little risk of his humor being misinterpreted as being negative. Crew
members report that this really helps to de-escalate tense situations."""
BB1 = """Bean had a difficult time getting along with some of his co-workers at
Ames. He had asked his supervisor to intervene on several occasions as they were
not able to resolve the disagreements on their own."""
BB2 = """Bean does not always plan his work well, and sometimes his pace of work
is too slow. This has become problematic not only in his mission to the ISS, but
also in training simulations. Often times, the maintenance work and data
collection the astronauts engage in on these missions requires high levels of
interdependency - this can be jeopardized if one person works at a slower pace
than the rest of the crew members."""
```

```
BB3 = """Bean has a 2-year-old daughter, and colleagues have noticed he is
frustrated when he has to work late and cannot be home with her. There is worry
that being away for so long on this mission will exponentially increase Bean's
frustration."""
BN1 = """Bean enjoys listening to the Beatles while performing work on his
own."""
BN2 = """Bean is an avid fan of Shakespeare and often quotes his work."""
BN3 = """In college, Bean was on the squash team, where he met his best friend
Jack. Jack now lives in Providence, Rhode Island, but they still talk on a weekly
basis."""
BN4 = """Astronaut Bean enjoys traveling and his favorite place to vacation to is
in the south of France."""
BN5 = """Bean has five siblings: one older brother, one older sister, and three
younger sisters. He is closest to his younger sister, Rebecca, who is just one
year younger than he is."""
BN6 = """If chosen for this mission, Bean would be the shortest crew member at 5-
feet, 5-inches tall."""

# Collins
CG1 = """Collins is accustomed to being isolated for long periods of time. He
successfully completed two treks to the Antarctic in which he lived in tents and
a small building for the Antarctic winter-over period. Living in small spaces did
not bother him."""
CG2 = """Collins has a passion for languages. He is fluent in English, Spanish,
and French and proficient in Russian. Recently he started a class on Mandarin and
already he is nearly proficient. This ability will help him interact and bond
with international crew members."""
CG3 = """Collins has participated in spacewalks to repair damaged solar panels
and ammonia pump module. His peers have ranked him as the most competent crew
member during their spacewalks."""
CG4 = """Collins received the “Silver Snoopy” award for his contributions on
NASA’s STS-135 mission. This award is considered to be the highest honor of all
of the awards, given to the person who was the most outstanding team member,
creating a team environment focused on flight safety and overall team success."""
CB1 = """Surveys showed that Collins has snapped at coworkers on multiple
occasions. In one situation, a crewmember even went so far as to say would not
want to work with Collins on future missions."""
CB2 = """Collins has the least space-related training of all three potential
candidates."""
CB3 = """Recent blood work indicated that Collins may be developing above-average
blood pressure. Doctors advised another assessment before making any conclusive
determinations. However, if Collins does have heightened blood pressure, he would
have to take extra time to rest while on the mission. This would detract from his
time working on experiments, as well as his ability to do physically stressful
activities."""
```

```
CB4 = """Three weeks ago, Collins broke his ankle. As a result, he has not been
training as much and the current state of his cardiovascular performance is
behind compared to the other two candidates."""
CB5 = """In one incident during a mock extravehicular activity scenario, Collins
refused to follow an instruction from mission control. Fellow crew members were
unsure of why he did this since the request from mission control seemed
straightforward and helpful. Crewmembers reported being conflicted with how to
respond, which slowed their progress on the extravehicular activity scenario.
Only half of the tasks needed were completed on the simulated extravehicular
activity."""
CB6 = """An internal incident report indicates that Collins made offensive jokes
to a Chinese astronaut on a previous analog space mission that made the Chinese
astronaut and other crew members uncomfortable."""
CN1 = """Collins considers himself a coffee aficionado and his favorite brand of
coffee is Intelligentsia."""
CN2 = """Collins' childhood friend completed NASA's astronaut candidate training
program but was never selected for a space mission."""
CN3 = """Collins enjoys completing Sudoku puzzles to relax"""
CN4 = """Collins has a fraternal twin who works in finance in Houston, Texas."""

knowledge_bank = [
    {"label": "AG1", "item": """Astronaut Anders received the highest possible
scores on the military survival standards, an assessment that all astronauts must
complete in order to qualify for space missions. Anders' scores were
significantly higher than either candidate's scores."""},
    {"label": "AG2", "item": """During Ander's last ICE (Isolated and Confined in
Extreme environments) training sequence, a fellow crew member praised him for
being able to recognize others' personal needs. In particular, he was recognized
for facilitating personal time and bringing the group together for light-hearted
fun."""},
    {"label": "AG3", "item": """While working at NASA, Anders coordinated and
completed a number of science projects. This expertise is important to Flight
Engineers and one of the reasons Anders was chosen for the previous space
mission. Anders did not disappoint. His ability to set up and coordinate science
projects allowed his crew to finish more than twice the number of projects than
anticipated."""},
    {"label": "AG4", "item": """Anders was one of 8 hikers to successfully summit
Mount Everest in 2005. Interviews with his comrades suggest Anders was the "glue"
that held the group together when things went wrong."""},
    {"label": "AG5", "item": """On previous missions, where lack of sleep was a
frequent occurrence, Anders was largely unaffected. Despite the sub-optimal
sleeping patterns, Anders' mental and physical health did not show signs of
suffering."""},
    {"label": "AG6", "item": """Data collected during training on simulated
emergencies shows that Anders is able to remain calm during stressful and
unexpected situations. He performs well under pressure."""},
    {"label": "AB1", "item": """Anders suffered a broken wrist 7 months ago
during a training simulation. The injury has healed, but will likely require
extra exercise to prevent forearm muscle atrophy in zero gravity. This may take
time away from him performing other team duties."""},
    {"label": "AB2", "item": """Most of Anders' space training experience has
been limited to simulations on Earth, with only one mission to the International
Space Station. Furthermore, this mission to Mars would be even longer than his
only long-distance flight experience--it is not clear if he will adapt well for a
long-duration mission."""},
    {"label": "AB3", "item": """Anders comes from a close-knit family. In diaries
collected during training, he reported feeling sad and frustrated when he was
unable to communicate with family members on a regular basis. This could be a
serious concern for his mental health on a mission to Mars because there will be
very little contact with Earth, and the little contact there is will be through
mission control."""},
    {"label": "AB4", "item": """Anders easily gets annoyed when explaining
concepts to others and becomes very short with them as a result."""},
    {"label": "AN1", "item": """Anders enjoys science fiction literature and has
self-published a novel about the start of a human colony in a distant galaxy. He
hopes to write a sequel when he retires."""},
    {"label": "AN2", "item": """As a child, Anders enjoyed attending Houstonli-
area sporting events with his parents. He was particularly fond of attending the
Texans football games."""},
    {"label": "AN3", "item": """Anders has a pet cat named Buzz, after Buzz
Aldrin. Anders met Buzz as a child around when he decided he wanted to be an
astronaut."""},
    {"label": "AN4", "item": """Anders used to own a Brown Labrador Retriever
named Bailey. When Anders participated in astronaut training and missions, Bailey
used to stay with his family."""},
    {"label": "BG1", "item": """On Expedition 31/32, where he helped refuel the
International Space Station (ISS), Bean encountered a breakdown in the
communication system between his vehicle and the ISS before arrival to the ISS.
Bean was credited for leading the extravehicular activity that identified and
fixed the issue, ultimately making the refuel possible."""},
    {"label": "BG2", "item": """While it is widely known that Bean holds a
bachelor’s degree in Aerospace Engineering and Cross-Cultural studies, few know
that he also pursued a minor in Counseling Psychology. Crewmembers have reported
that Bean is a great listener and often knows exactly what to say to make others
feel more comfortable."""},
    {"label": "BG3", "item": """Survey responses from Bean’s previous crew mates
reveal that he has a universal sense of humor. People always understand and enjoy
Bean’s jokes, so there is little risk of his humor being misinterpreted as being
negative. Crew members report that this really helps to de-escalate tense
situations."""},
    {"label": "BB1", "item": """Bean had a difficult time getting along with some
of his co-workers at Ames. He had asked his supervisor to intervene on several
occasions as they were not able to resolve the disagreements on their own."""},
    {"label": "BB2", "item": """Bean does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "BB3", "item": """Bean has a 2-year-old daughter, and colleagues
have noticed he is frustrated when he has to work late and cannot be home with
her. There is worry that being away for so long on this mission will
exponentially increase Bean's frustration."""},
    {"label": "BN1", "item": """Bean enjoys listening to the Beatles while
performing work on his own."""},
    {"label": "BN2", "item": """Bean is an avid fan of Shakespeare and often
quotes his work."""},
    {"label": "BN3", "item": """In college, Bean was on the squash team, where he
met his best friend Jack. Jack now lives in Providence, Rhode Island, but they
still talk on a weekly basis."""},
    {"label": "BN4", "item": """Astronaut Bean enjoys traveling and his favorite
place to vacation to is in the south of France."""},
    {"label": "BN5", "item": """Bean has five siblings: one older brother, one
older sister, and three younger sisters. He is closest to his younger sister,
Rebecca, who is just one year younger than he is."""},
    {"label": "BN6", "item": """If chosen for this mission, Bean would be the
shortest crew member at 5-feet, 5-inches tall."""},
    {"label": "CG1", "item": """Collins is accustomed to being isolated for long
periods of time. He successfully completed two treks to the Antarctic in which he
lived in tents and a small building for the Antarctic winter-over period. Living
in small spaces did not bother him."""},
    {"label": "CG2", "item": """Collins has a passion for languages. He is fluent
in English, Spanish, and French and proficient in Russian. Recently he started a
class on Mandarin and already he is nearly proficient. This ability will help him
interact and bond with international crew members."""},
    {"label": "CG3", "item": """Collins has participated in spacewalks to repair
damaged solar panels and ammonia pump module. His peers have ranked him as the
most competent crew member during their spacewalks."""},
    {"label": "CG4", "item": """Collins received the “Silver Snoopy” award for
his contributions on NASA’s STS-135 mission. This award is considered to be the
highest honor of all of the awards, given to the person who was the most
outstanding team member, creating a team environment focused on flight safety and
overall team success."""},
    {"label": "CB1", "item": """Surveys showed that Collins has snapped at
coworkers on multiple occasions. In one situation, a crewmember even went so far
as to say would not want to work with Collins on future missions."""},
    {"label": "CB2", "item": """Collins does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "CB3", "item": """Recent blood work indicated that Collins may be
developing above-average blood pressure. Doctors advised another assessment
before making any conclusive determinations. However, if Collins does have
heightened blood pressure, he would have to take extra time to rest while on the
mission. This would detract from his time working on experiments, as well as his
ability to do physically stressful activities."""},
    {"label": "CB4", "item": """Three weeks ago, Collins broke his ankle. As a
result, he has not been training as much and the current state of his
cardiovascular performance is behind compared to the other two candidates."""},
    {"label": "CB5", "item": """In one incident during a mock extravehicular
activity scenario, Collins refused to follow an instruction from mission control.
Fellow crew members were unsure of why he did this since the request from mission
control seemed straightforward and helpful. Crewmembers reported being conflicted
with how to respond, which slowed their progress on the extravehicular activity
scenario. Only half of the tasks needed were completed on the simulated
extravehicular activity."""},
    {"label": "CB6", "item": """An internal incident report indicates that
Collins made offensive jokes to a Chinese astronaut on a previous analog space
mission that made the Chinese astronaut and other crew members
uncomfortable."""},
    {"label": "CN1", "item": """Collins considers himself a coffee aficionado and
his favorite brand of coffee is Intelligentsia."""},
    {"label": "CN2", "item": """Collins’ childhood friend completed NASA’s
astronaut candidate training program but was never selected for a space
mission."""},
    {"label": "CN3", "item": """Collins enjoys completing Sudoku puzzles to
relax"""},
    {"label": "CN4", "item": """Collins has a fraternal twin who works in finance
in Houston, Texas."""}
]

#####AGENT 1: CMR######
cmr_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG6, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True}
]
cmr = CogAgent("CMR", cmr_KB)
cog_cmr = CogChain(cmr)

#####AGENT 2: MS1#####
ms1_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CB3, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CB5, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": CN4, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN4, "frequency" : 0 ,"initial": True},
    {"item": AG1, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True}
]
ms1 = CogAgent("MS1", ms1_KB)
cog_ms1 = CogChain(ms1)

#####AGENT 3: MS2#####
ms2_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
ms2 = CogAgent("MS2", ms2_KB)
cog_ms2 = CogChain(ms2)

#####AGENT 4: FE######
fe_KB = [
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CB2, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN5, "frequency" : 0 ,"initial": True},
    {"item": BN6, "frequency" : 0 ,"initial": True},
    {"item": AG5, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
fe = CogAgent("FE", fe_KB)
cog_fe = CogChain(fe)

cog_agents = [cog_cmr, cog_ms1, cog_ms2, cog_fe]
rounds = 10
decision_options = """Candidate 1: Samuel Anders
Candidate 2: John Bean
Candidate 3: Scott Collins"""
decision_rule = 'majority'
discussion_task = """ DISCUSSION TASK: You work as a member of NASA’s Astronaut
Crew Composition team. Your team analyzes all of the information obtained during
astronaut selection and training - including test scores, analog performance
measures, expert observations, peer evaluations, and medical  records. Armed with
this information, your team makes two kinds of evaluations. The first evaluation
is to determine which astronauts are ready for flight and which are not. Of those
who are deemed to be flight ready, your team composes entire crews for each
mission. In this second evaluation, you consider the complementarity and
compatibility of the astronauts as a team to select the 4, 5, or 6  astronauts
(depending on the mission) who will perform best as individuals, and as a crew.
While composing crews for missions on the International Space Station has become
routine, your team just received a special assignment from the Mission Planning
Directorate that is anything but routine.
The Mars mission has been fast-tracked. Instead of the 2033 launch date that NASA
had been aiming for, the launch date is being moved up to 2020. Intensive crew
training for this unprecedented and bold journey must begin immediately. A 5-
member crew will embark on this 3 year journey to Mars. This a truly
international endeavor: the Russian, European, Indian, and Chinese Space Agencies
will each select one astronaut for the mission. NASA will select the 5th crew
member - an American astronaut - to join this international crew.
```

```
The crew members from the other countries have backgrounds in engineering,
botany, geology and medicine. The crew still needs a flight engineer.
Your team was able to quickly narrow the field down to three veteran astronauts:
Anders, Bean, and Collins. Each of these three have experience as a flight
engineer on the ISS, have the necessary basic technical skills for the flight
engineer position, meet basic requirements for the mission including height, Body
Mass Index, and have passed a mental health screening. Now that you know the
other potential crew members it is necessary for your team to determine which of
the crew members will be the greatest asset on this particular mission.
The 3-year mission to Mars will be demanding. The success of the Mars mission
will require the 5-person crew to live and work together in extremely close
quarters, while isolated from the usual sources of social support - friends and
family - for extended periods of time. It will require that the crew can adapt to
the unknown and be able to handle whatever challenges will come their way. Once
they arrive at Mars, the crew will have to complete complex feats such as landing
and extravehicular activities. Because of a significant communication delay with
Earth of up to 22 minutes each way, the crew will have to work autonomously.
Additionally, the crew of the Mars mission consists of individuals representing
multiple nationalities, making strong interpersonal skills all the more critical
to the success of the mission.
The international space consortium has agreed to announce the names of the crew
members from each of the 5 countries in just 10 days from now. Your team must
deliver a recommendation for whom should be the American astronaut; years of hard
work (and millions of dollars) rest on this decision.
To guide your difficult decision, you have compiled information from the
personnel file for each candidate. The personnel files for each candidate are
detailed and include information such as their performance scores and ranking on
various training tasks, biographical information, evaluations and questionnaires
completed by  supervisors and individuals with whom they worked on prior
trainings and missions, and questionnaires completed by the candidate and his
family. A few human resource assistant have been working through the personnel
files and have prepared briefs for you. Your team must now come together and
discuss who should be selected for the mission.
Remember, the success of the mission to Mars depends on your team. Choose
quickly, but wisely!
"""
astronaut_candidate = Conversation(cog_agents, discussion_task, rounds,
decision_options, decision_rule, knowledge_bank)

astronaut_candidate.run_convo()

with open("agent_states.txt", "w") as states_file:
    for agent in astronaut_candidate.cog_agents:
        states_file.write(f"----- {agent.cog_agent.name} States -----\n")
        for element in agent.cog_agent.state:
            states_file.write(f"{element}\n")
        states_file.write("\n")  # Add a newline after each agent's states
```

Report.py

```
from CogAgent import CogAgent
from CogChain import CogChain

class Report:
    def __init__(self, cog_agents, knowledge_bank):
        self.initial_preferences = []
        self.turn_preferences = {}
        self.final_preferences = []
        self.turn_info_profiles = {}
        self.initial_info_profiles = []
        self.final_info_profiles = []
        self.speaker_frequency = []
        self.revealed_profile = []
        self.non_initial_contributions_percentages = []
        self.cog_agents = cog_agents
        self.knowledge_bank = knowledge_bank

    def retrieve_preferences(self, cog_agents):
        preferences = [f"{agent.cog_agent.name}: {agent.cog_agent.preference}"
for agent in cog_agents]
        return preferences

    def aggregate_round_profile(self, revealed_profile):
        aggregated_profile = {}
        all_shared_items = set()  # To keep track of all items shared in the
rounds
        item_frequencies = {}  # To track the frequency of each item

        for entry in revealed_profile:
            round_num = entry["round"]
            item = entry["item"]
            agent = entry["agent"]  # Include the contributing agent name

            if round_num not in aggregated_profile:
                aggregated_profile[round_num] = [(item, agent)]
            else:
                aggregated_profile[round_num].append((item, agent))

            all_shared_items.add(item)  # Add the item to the set
            # Update the frequency of the item
            item_frequencies[item] = item_frequencies.get(item, 0) + 1

        # Prepare a string to be used in the report
        result_str = "\n----- Turn Group Revealed Profile -----\n"
        for round_num, items_agents in sorted(aggregated_profile.items()):
            result_str += f"Round {round_num}:\n"
            for item, agent in items_agents:
                result_str += f"Info Item: {item}\nContributing Agent: {agent}\n"
            result_str += "\n"
        result_str += "\n----- Final Group Revealed Profile -----\n"
        # Include a list of all items shared in the rounds with frequency
        for item in all_shared_items:
            result_str += f"{item} (Frequency: {item_frequencies[item]})\n"
        return result_str

    def retrieve_agent_frequency(self, cog_agents):
        speaker_frequency = [f"{agent.cog_agent.name}:
{agent.cog_agent.frequency}" for agent in cog_agents]
        return speaker_frequency

    def generate_analysis_report(self, discussion_result):
        with open("analysis_report.txt", "w") as analysis_file:
            analysis_file.write("********************************************\n"
)
            analysis_file.write("ANALYSIS REPORT\n")
            analysis_file.write("********************************************\n"
)

            # 1. Calculate percentage of revealed profile items that are in the
knowledge bank
            all_shared_items = self.extract_all_shared_items()
            revealed_profile_percentage =
self.calculate_revealed_profile_percentage(all_shared_items)
            analysis_file.write("REVEALED PROFILE:\n")
            analysis_file.write(f"{revealed_profile_percentage:.2f}%\n\n")

            # 2. Calculate percentage of preference changes
            preference_change_percentage = self.calculate_preference_changes()
            analysis_file.write("PREFERENCE CHANGES:\n")
            analysis_file.write(f"{preference_change_percentage:.2f}%\n\n")

            # 3. Speaker distribution
            speaker_distribution = self.calculate_speaker_distribution()
            analysis_file.write("SPEAKER DISTRIBUTION:\n")
            for agent, percentage in speaker_distribution.items():
                analysis_file.write(f"{agent}: {percentage:.2f}%\n")
            analysis_file.write("\n")

            # 4. Decision made and group optimality rating
            if discussion_result != 'NO CONSENSUS':
                # decision_made, group_optimality =
self.calculate_decision_optimality(all_shared_items)
                group_optimality_revealed_profile =
self.calculate_revealed_profile_optimality(all_shared_items, discussion_result)
                group_optimality_knowledge_bank =
self.calculate_knowledge_bank_optimality(discussion_result)
                analysis_file.write("DECISION MADE:\n")
                analysis_file.write(f"Decision: {discussion_result}\n")
                analysis_file.write(f"Group Optimality (Revealed Profile):
{group_optimality_revealed_profile}\n")
                analysis_file.write(f"Group Optimality (Knowledge Bank):
{group_optimality['knowledge_bank']}\n")

                # Corrected individual optimality ratings

            else:
                analysis_file.write("No group decision made.\n")
                analysis_file.write("\nIndividual Optimality:\n")
                for agent_pref in self.final_preferences:
                    agent_name, preference = agent_pref.split(": ")
                    revealed_profile_optimality =
self.calculate_revealed_profile_optimality(all_shared_items, preference)
                    knowledge_bank_optimality =
self.calculate_knowledge_bank_optimality(preference)
                    analysis_file.write(f"{agent_name}:\n")
                    analysis_file.write(f"Revealed Profile:
{revealed_profile_optimality}\n")
                    analysis_file.write(f"Knowledge Bank:
{knowledge_bank_optimality}\n")

    def extract_all_shared_items(self):
        all_shared_items = set()
        for entry in self.revealed_profile:
            label = entry['item']
            for item in self.knowledge_bank:
                if item['label'] == label:
                    all_shared_items.add(item['label'])
                    break  # Stop searching once the item is found
        return all_shared_items
    def calculate_revealed_profile_percentage(self, all_shared_items):
    # Calculate the percentage of unique items in the revealed profile also in
the knowledge bank
        unique_revealed_items = set(entry['item'] for entry in
self.revealed_profile)
        total_revealed_items = len(unique_revealed_items)

        if total_revealed_items == 0:
            return 0.0

        kb_items = [entry['label'] for entry in self.knowledge_bank]
        kb_length = len(kb_items)

        percentage = (total_revealed_items / kb_length) * 100
        return percentage

    def calculate_preference_changes(self):
        initial_preferences = [agent.split(": ")[1] for agent in
self.initial_preferences]
        final_preferences = [agent.split(": ")[1] for agent in
self.final_preferences]
        changed_agents = [initial != final for initial, final in
zip(initial_preferences, final_preferences)]
        return (sum(changed_agents) / len(self.cog_agents)) * 100

    def calculate_speaker_distribution(self):
        total_turns = sum([int(freq.split(": ")[1]) for freq in
self.speaker_frequency])
        speaker_distribution = {freq.split(": ")[0]: (int(freq.split(": ")[1]) /
total_turns) * 100 for freq in self.speaker_frequency}
        return speaker_distribution

    def calculate_revealed_profile_optimality(self, all_shared_items,
agent_preference):
        ####FIX
        preference_counts = {'C': 0, 'A': 0, 'B': 0}
        for item in all_shared_items:
        # Check if the second item letter is 'G'
            if item[1] == 'G':
                # Determine the preference of the first item and update
preference counts
                if item[0] == 'C':
                    preference_counts['C'] += 1
                elif item[0] == 'A':
                    preference_counts['A'] += 1
                elif item[0] == 'B':
                    preference_counts['B'] += 1
        preference_ranking = {'C': 0, 'A': 0, 'B': 0}
        for preference, count in preference_counts.items():
            preference_ranking[preference] = count
        sorted_preferences = sorted(preference_ranking.items(), key=lambda x:
x[1], reverse=True)
        max_count = sorted_preferences[0][1]
        max_count_preferences = sum(1 for pref, count in sorted_preferences if
count == max_count)
        ranks = {}
        if max_count_preferences == 1:  # Only one preference has the maximum
count
            for preference, count in sorted_preferences:
                ranks[preference] = 'BEST' if count == max_count else 'WORST'
        else:  # Multiple preferences have the maximum count
            for preference, count in sorted_preferences:
                ranks[preference] = 'AVERAGE' if count == max_count else 'WORST'

            # Calculate second highest count
        second_highest_count = sorted_preferences[1][1]
            # Calculate how many preferences have the second highest count
        second_highest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == second_highest_count)

            # Assign 'AVERAGE' rank to preferences with the second highest count
        if second_highest_count_preferences > 1:
            for preference, count in sorted_preferences:
                if count == second_highest_count:
                    ranks[preference] = 'AVERAGE'

            # Calculate lowest count
        lowest_count = sorted_preferences[-1][1]

            # If lowest count is different from second highest count, assign
'WORST' rank
        if lowest_count != second_highest_count:
            lowest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == lowest_count)
            if lowest_count_preferences > 1:
                for preference, count in sorted_preferences:
                    if count == lowest_count:
                        ranks[preference] = 'WORST'
            # Extract agent preference label
        agent_preference_label = agent_preference.split()[1][0]

            # Assign the agent's preference ranking
        agent_preference_ranking = ranks.get(agent_preference_label, None)
        return agent_preference_ranking

    def calculate_knowledge_bank_optimality(self, preference):
        # Define the fixed knowledge bank optimality ratings
        knowledge_bank_ratings = {'Samuel Anders': 'BEST', 'John Bean':
'AVERAGE', 'Scott Collins': 'WORST'}
        return knowledge_bank_ratings.get(preference, 'UNKNOWN')

    def generate_preference_report(self, discussion_result, rounds,
decision_rule):
        with open("report.txt", "w") as report_file:
            report_file.write("*********************************************\n")
            report_file.write("DISCUSSION REPORT\n")
            report_file.write("*********************************************\n")
            report_file.write(f"Number of Rounds: {rounds}\n")
            report_file.write(f"Decision Rule: {decision_rule}\n")
            report_file.write
            if discussion_result == 'NO CONSENSUS':
                report_file.write(f"Discussion Result: Failure to reach consensus
in the specified number of rounds.\n")
            else:
                report_file.write(f"Discussion Result: Consensus reached in
{rounds} rounds!\n")
                report_file.write(f"Agreed upon goal: {discussion_result}\n")
            report_file.write("\n*********************************************\n"
)
            report_file.write("PREFERENCE INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("----- Initial Preference Distribution -----\n")
            for i, entry in enumerate(self.initial_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("----- Turn Preference Distribution -----\n")
            for round_num, preferences in self.turn_preferences.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(preferences, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Preference Distribution -----\n")
            for i, entry in enumerate(self.final_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("INFO ITEM INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("\n----- Initial Info Profiles -----\n")
            for i, entry in enumerate(self.initial_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n----- Turn Info Profiles -----\n")
            for round_num, info_profile in self.turn_info_profiles.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(info_profile, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Info Profiles -----\n")
            for i, entry in enumerate(self.final_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            aggregated_profile_str =
self.aggregate_round_profile(self.revealed_profile)
            report_file.write(f"{aggregated_profile_str}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("CONVO INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("-----Speaker Frequency -----\n")
            for i, entry in enumerate(self.speaker_frequency, start=1):
                report_file.write(f"{entry}\n")
            report_file.write("\n----- Non-Initial Contributions-----\n")
            for agent_name, percentage in
self.non_initial_contributions_percentages.items():
                report_file.write(f"{agent_name}: {percentage}%\n")

    def kb_map(self, knowledge_base, knowledge_bank):
        labels = []
        for kb_entry in knowledge_base:
            kb_item = kb_entry["item"]
            for bank_entry in knowledge_bank:
                bank_item = bank_entry["item"]
                if kb_item == bank_item:
                    labels.append(bank_entry["label"])
                    break  # Stop searching once a match is found
        return labels

    def retrieve_info_profiles(self, cog_agents, knowledge_bank):
        profiles = []
        for agent in cog_agents:
            knowledge_base = agent.cog_agent.knowledge_base
            profile_labels = self.kb_map(knowledge_base, knowledge_bank)
            profiles += [f"{agent.cog_agent.name}: {profile_labels}"]
        return profiles

    def calculate_non_initial_contributions_percentage(self, agent_name,
revealed_profile):
        agent_contributions = [entry for entry in revealed_profile if
entry['agent'] == agent_name]
        total_contributions = len(agent_contributions)

        if total_contributions == 0:
            return None  # Return None for agents who did not contribute

        non_initial_contributions = sum(
            1 for entry in agent_contributions if not
self.is_item_in_initial_kb(agent_name, entry['item']))

        if total_contributions == non_initial_contributions:
            return 0.0  # All contributions were non-initial

        percentage = (non_initial_contributions / (total_contributions -
non_initial_contributions)) * 100
        return round(percentage, 2)

    def is_item_in_initial_kb(self, agent_name, item):
        for agent in self.cog_agents:
            if agent.cog_agent.name == agent_name:
                return agent.cog_agent.was_item_in_initial_kb(item)
        return False
```

APPENDIX E: INTERACTIONCOG CODE

CogAgent.py

```
class CogAgent:
    def __init__(self, name, knowledge_base):
        self.name = name
        self.knowledge_base = knowledge_base
        self.initial_knowledge_base = knowledge_base.copy()
        self.preference = ""
        self.state = []
        self.frequency = 0

    def was_item_in_initial_kb(self, item):
        return any(entry['item'] == item for entry in
self.initial_knowledge_base)

    def state_snap(self):
        snapshot = ""
        snapshot += f"Preference: {self.preference}\n"
        return snapshot

    def update_agent_state(self, state):
        self.state.append(state)
```

CogChain.py

```
import os
import re
os.environ["OPENAI_API_KEY"]='sk-
iRdjQCppyWt3WblPYd24T3BlbkFJYsTrZOxSnVoajtnSoI5U'
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
llm = ChatOpenAI(model="gpt-4-0125-preview")
import random
import sys
from CogAgent import CogAgent

class CogChain:
    def __init__(self, cog_agent):
        self.cog_agent = cog_agent
        self.convo_history = []
        self.name = cog_agent.name


    def kb_print(self, kb):
        kb_str = "KNOWLEDGE BASE:\n"
        for item in kb:
            item_str = f"{item['item']}\n\n"
            kb_str += item_str
        return kb_str

    def preference_chain(self, task, kb, decision_options):
        kb_string = self.kb_print(kb)
        update_goal = PromptTemplate(
            input_variables = ["task","updated_kb", "decision_options"],
            template = """{task}. You have access to a list of facts, called a
'knowledge base', to help guide your preference. Here is the most recently
updated list:
            {updated_kb}
            Based on the above facts, choose one alternative from the decision
options. You must choose from the provided options only.
            {decision_options}
            Use this format for your output: PREFERENCE: (Insert candidate name)
\n EXPLANATION: (justify your decision, limit to 300 tokens).""")
        goal_update = LLMChain(llm=llm, prompt=update_goal)
        updated_goal = goal_update.invoke({
        "task": task,
        "updated_kb": kb_string,
        "decision_options": decision_options})
        goal_match = re.search(r'PREFERENCE: (.+)', updated_goal['text'])
        if goal_match:
            new_goal = goal_match.group(1)
            self.cog_agent.preference = new_goal
        return updated_goal["text"]

    def contribution_chain(self, task, context):
        kb = self.cog_agent.knowledge_base
        kb_string = self.kb_print(kb)
        generate_contribution = PromptTemplate(
            input_variables = ["goal", "task", "kb", "context"],
            template = """{task} Your team mate just contributed: {context}.
                It is your turn in the discussion. Choose a fact from the
knowledge base and provide it (exactly as phrased) alongside your response. You
are likely to choose a fact that aligns with your preference. You must directly
respond to the contribution made by your teammate. Do not discuss any other fact
from your knowledge base, aside from your chosen fact and your teammates
contribution, in your response.
                Preference: {goal}
                {kb}
                Output format: INFORMATION ITEM: (insert fact exactly as given)
\n COMMENTARY: (insert response)""")
        gen_contribution = LLMChain(llm=llm, prompt=generate_contribution)
        contribution = gen_contribution.invoke({
            "goal": self.cog_agent.preference,
            "task": task,
            "kb" : kb_string,
            "context": context})
        return contribution['text']

    def process_message(self, task, context, decision_options):
        #Initial Snap
        initial_snap = self.cog_agent.state_snap()
        #Take in Context, Update KB
        self.knowledge_base_chain(context)
        #Update Goal
        goal_result = self.preference_chain(task, self.cog_agent.knowledge_base,
decision_options)
        #Final Snap
        final_snap = self.cog_agent.state_snap()
        #Generate process report
        process_report = f"---------PROCESS REPORT----------\n"
        process_report += f"Initial Agent State: " + initial_snap + "\n"
        process_report += f"***\n"
        process_report += f"Context: " + context + "\n"
        process_report += f"***\n"
        process_report += f"Final Agent State: " + final_snap + "\n"
        process_report += f""+ goal_result + "\n"
        self.cog_agent.update_agent_state(process_report)

    def knowledge_base_chain(self, context):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        # Find if the item is in the knowledge base
        item_found = False
        for item in self.cog_agent.knowledge_base:
            if item['item'] == info_item:
                item['frequency'] += 1
                item_found = True
                break

        # If the item is not found, add it to the knowledge base
        if not item_found:
            self.cog_agent.knowledge_base.append({"item": info_item, "frequency":
1, "initial": False})

        # Use kb_print to get the updated knowledge base string
        updated_kb_str = self.kb_print(self.cog_agent.knowledge_base)
        return updated_kb_str

    def first_turn_contribution(self, discussion_task):
        initial_snap = self.cog_agent.state_snap()
        goal = self.cog_agent.preference
        knowledge_base = self.cog_agent.knowledge_base
        kb_string = self.kb_print(knowledge_base)
        generate_first_turn_contribution = PromptTemplate(
            input_variables = ["goal", "kb", "task", "context"],
            template = """{task}. You are starting the discussion. Choose a fact
from the knowledge base and provide it (exactly as phrased) alongside your
commentary. You are likely to choose a fact that aligns with your preference.
            Preference: {goal}
            {kb}
            Output format: INFORMATION ITEM: (insert fact) \n COMMENTARY: (insert
your commentary)""")
        gen_first_turn_contribution = LLMChain(llm=llm,
prompt=generate_first_turn_contribution)
        contribution = gen_first_turn_contribution.invoke({
            "goal": goal,
            "kb": kb_string,
            "task": discussion_task,})
        updated_kb = self.knowledge_base_chain(contribution['text'])
        final_snap = self.cog_agent.state_snap()
        #Generate state report
        state_report = f"Contribution: " + contribution['text'] + "\n"
        state_report += f"---------------------------------------\n"
        state_report += f"Final Agent State: " + final_snap + "\n"
        self.cog_agent.update_agent_state(state_report)
        return contribution['text']

    def invoke_convo(self, sample_context, discussion_task):
        kb = self.cog_agent.knowledge_base
        goal = self.cog_agent.preference
        contribution_result = 
self.contribution_chain(discussion_task,sample_context)
        updated_kb = self.knowledge_base_chain(contribution_result)
        ###GENERATE TURN REPORT
        final_snap = self.cog_agent.state_snap()
        self.cog_agent.update_agent_state(final_snap)
        return contribution_result
```

Conversation.py

```
import os
import re
import random
import sys
from Report import Report
from CogAgent import CogAgent
from CogChain import CogChain

class Conversation:
    def __init__(self, cog_agents, discussion_task, rounds, decision_options, 
decision_rule, knowledge_bank):
        self.cog_agents = cog_agents
        self.discussion_task = discussion_task
        self.rounds = rounds
        self.convo_history = []
        self.decision_options = decision_options
        self.decision_rule = decision_rule
        self.report = Report(cog_agents, knowledge_bank)
        self.revealed_profile = []
        self.knowledge_bank = knowledge_bank

    def broadcast_message(self, cog_agents, context, agent_in):
        for agent in cog_agents:
            if agent.name != agent_in:
                agent.process_message(self.discussion_task, context, 
self.decision_options)

    def update_revealed_profile(self, context, current_agent, round):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        rp = [{"item": info_item,
            "agent": current_agent.name,
            "round": round}]

        info_item_label = self.report.kb_map(rp, self.knowledge_bank)
        # Add the info_item to the revealed_profile
        self.revealed_profile.append({
            "item": info_item_label[0],
            "agent": current_agent.name,
            "round": round})
        return self.revealed_profile

    def consensus(self):
        # Check for consensus based on the decision_rule
        if self.decision_rule == 'unanimous':
            # Check if all agents have the same goal
            goals = {agent.cog_agent.preference for agent in self.cog_agents}
            if len(goals) == 1:
                return goals.pop()  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        elif self.decision_rule == 'majority':
            # Check if there is a majority agreement on the goal
            goal_counts = {agent.cog_agent.preference: 0 for agent in
self.cog_agents}
            for agent in self.cog_agents:
                goal_counts[agent.cog_agent.preference] += 1

            max_goal = max(goal_counts, key=goal_counts.get)

            if goal_counts[max_goal] > len(self.cog_agents) / 2:
                return max_goal  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        else:
            return 'NO CONSENSUS'

    def next_speaker(self, current_agent, broadcast_agents):
        next_speakers = [agent for agent in broadcast_agents]
        if next_speakers:
            # If more than 2+ agents, choose randomly
            next_speaker = random.choice(next_speakers)
        return next_speaker

    def run_convo(self):
        #initialize agent preferences
        for agent in self.cog_agents:
            # Call preference_chain for each agent
            preference = agent.preference_chain(
                self.discussion_task, agent.cog_agent.knowledge_base,
self.decision_options)
            agent.cog_agent.update_agent_state(preference)

        # Store Initial Profiles
        self.report.initial_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        # Store Initial Preferences
        self.report.initial_preferences =
self.report.retrieve_preferences(self.cog_agents)

        # Initial Turn Contribution
        agent1 = random.choice(self.cog_agents)
        contribution1 = agent1.first_turn_contribution(self.discussion_task)
        agent1.cog_agent.frequency += 1

        # Add first contribution to chat history, ADD ITEM TO REVEALED_PROFILE
        self.convo_history.append(contribution1)
        self.update_revealed_profile(contribution1, agent1, 0)

        # Broadcast first contribution to other agents
        broadcast_agents = [agent for agent in self.cog_agents if agent !=
agent1]
        self.broadcast_message(broadcast_agents, contribution1, agent1.name)

        # Store Turn Preference
        self.report.turn_preferences[0] =
self.report.retrieve_preferences(self.cog_agents)
        # Store Turn Info Profiles
        self.report.turn_info_profiles[0] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        print(agent1.name)
        print(contribution1)
        current_agent = self.next_speaker(agent1, broadcast_agents)

        for round_num in range(1, self.rounds):
            context = self.convo_history[-1]
            contribution = current_agent.invoke_convo(context,
self.discussion_task)
            current_agent.cog_agent.frequency += 1
            print(current_agent.name)
            print(contribution)
            self.convo_history.append(contribution)
            #ADD ITEM TO REVEALED PROFILE
            self.update_revealed_profile(contribution, current_agent, round_num)
            # Broadcast to other agents
            broadcast_agents = [agent for agent in self.cog_agents if agent !=
current_agent]
            self.broadcast_message(broadcast_agents, contribution,
current_agent.name)
            # Store Turn Preference
            self.report.turn_preferences[round_num] =
self.report.retrieve_preferences(self.cog_agents)
             # Store Turn Info Profiles
            self.report.turn_info_profiles[round_num] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
            decision_result = self.consensus()
            if decision_result != 'NO CONSENSUS':
                print(f"Consensus reached! Agreed upon goal: {decision_result}")
                break  # Exit the loop if consensus is reached
            #IF agents do not agree, select next speaker
            current_agent = self.next_speaker(current_agent, broadcast_agents)
        decision_result = self.consensus()
        # Store Final Preferences
        self.report.final_preferences =
self.report.retrieve_preferences(self.cog_agents)
        # Store Final Info Profiles
        self.report.final_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
        # Store Agent Speaker Frequencies
        self.report.speaker_frequency =
self.report.retrieve_agent_frequency(self.cog_agents)
        self.report.revealed_profile = self.revealed_profile
        # Calc & Store non-initial contribution %
        non_initial_percentages = {}
        for agent in self.cog_agents:
            agent_name = agent.cog_agent.name
            non_initial_percentage =
self.report.calculate_non_initial_contributions_percentage(agent_name,
self.revealed_profile)
```

```
            if non_initial_percentage is not None:
                non_initial_percentages[agent_name] = non_initial_percentage
        self.report.non_initial_contributions_percentages = 
non_initial_percentages
        # Generate Report
        self.report.generate_preference_report(decision_result, self.rounds, 
self.decision_rule)
        self.report.generate_analysis_report(decision_result)
        with open("conversation_log.txt", "w") as log_file:
            log_file.write("----- Conversation Thread -----\n")
            for i, entry in enumerate(self.convo_history, start=1):
                log_file.write(f"Turn {i}:\n{entry}\n")
```

Discuss.py

```
from Conversation import Conversation
from CogAgent import CogAgent
from Report import Report
from CogChain import CogChain

####INFO PROFILE####
# Anders
AG1 = """Astronaut Anders received the highest possible scores on the military 
survival standards, an assessment that all astronauts must complete in order to 
qualify for space missions. Anders' scores were significantly higher than either 
candidate’s scores."""
AG2 = """During Ander’s last ICE (Isolated and Confined in Extreme environments) 
training sequence, a fellow crew member praised him for being able to recognize 
others’ personal needs. In particular, he was recognized for facilitating 
personal time and bringing the group together for light-hearted fun."""
AG3 = """While working at NASA, Anders coordinated and completed a number of 
science projects. This expertise is important to Flight Engineers and one of the 
reasons Anders was chosen for the previous space mission. Anders did not 
disappoint. His ability to set up and coordinate science projects allowed his 
crew to finish more than twice the number of projects than anticipated."""
AG4 = """Anders was one of 8 hikers to successfully summit Mount Everest in 2005. 
Interviews with his comrades suggest Anders was the “glue” that held the group 
together when things went wrong."""
AG5 = """On previous missions, where lack of sleep was a frequent occurrence, 
Anders was largely unaffected. Despite the sub-optimal sleeping patterns, Anders' 
mental and physical health did not show signs of suffering."""
AG6 = """Data collected during training on simulated emergencies shows that 
Anders is able to remain calm during stressful and unexpected situations. He 
performs well under pressure."""
```

```
AB1 = """Anders suffered a broken wrist 7 months ago during a training
simulation. The injury has healed, but will likely require extra exercise to
prevent forearm muscle atrophy in zero gravity. This may take time away from him
performing other team duties."""
AB2 = """Most of Anders' space training experience has been limited to
simulations on Earth, with only one mission to the International Space Station.
Furthermore, this mission to Mars would be even longer than his only long-
distance flight experience--it is not clear if he will adapt well for a long-
duration mission."""
AB3 = """Anders comes from a close-knit family. In diaries collected during
training, he reported feeling sad and frustrated when he was unable to
communicate with family members on a regular basis. This could be a serious
concern for his mental health on a mission to Mars because there will be very
little contact with Earth, and the little contact there is will be through
mission control."""
AB4 = """Anders easily gets annoyed when explaining concepts to others and
becomes very short with them as a result."""
AN1 = """Anders enjoys science fiction literature and has self-published a novel
about the start of a human colony in a distant galaxy. He hopes to write a sequel
when he retires."""
AN2 = """As a child, Anders enjoyed attending Houstonli-area sporting events with
his parents. He was particularly fond of attending the Texans football games."""
AN3 = """Anders has a pet cat named Buzz, after Buzz Aldrin. Anders met Buzz as a
child around when he decided he wanted to be an astronaut."""
AN4 = """Anders used to own a Brown Labrador Retriever named Bailey. When Anders
participated in astronaut training and missions, Bailey used to stay with his
family."""

# Bean
BG1 = """On Expedition 31/32, where he helped refuel the International Space
Station (ISS), Bean encountered a breakdown in the communication system between
his vehicle and the ISS before arrival to the ISS. Bean was credited for leading
the extravehicular activity that identified and fixed the issue, ultimately
making the refuel possible."""
BG2 = """While it is widely known that Bean holds a bachelor’s degree in
Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a
minor in Counseling Psychology. Crewmembers have reported that Bean is a great
listener and often knows exactly what to say to make others feel more
comfortable."""
BG3 = """Survey responses from Bean’s previous crew mates reveal that he has a
universal sense of humor. People always understand and enjoy Bean’s jokes, so
there is little risk of his humor being misinterpreted as being negative. Crew
members report that this really helps to de-escalate tense situations."""
```

```
BB1 = """Bean had a difficult time getting along with some of his co-workers at
Ames. He had asked his supervisor to intervene on several occasions as they were
not able to resolve the disagreements on their own."""
BB2 = """Bean does not always plan his work well, and sometimes his pace of work
is too slow. This has become problematic not only in his mission to the ISS, but
also in training simulations. Often times, the maintenance work and data
collection the astronauts engage in on these missions requires high levels of
interdependency - this can be jeopardized if one person works at a slower pace
than the rest of the crew members."""
BB3 = """Bean has a 2-year-old daughter, and colleagues have noticed he is
frustrated when he has to work late and cannot be home with her. There is worry
that being away for so long on this mission will exponentially increase Bean's
frustration."""
BN1 = """Bean enjoys listening to the Beatles while performing work on his
own."""
BN2 = """Bean is an avid fan of Shakespeare and often quotes his work."""
BN3 = """In college, Bean was on the squash team, where he met his best friend
Jack. Jack now lives in Providence, Rhode Island, but they still talk on a weekly
basis."""
BN4 = """Astronaut Bean enjoys traveling and his favorite place to vacation to is
in the south of France."""
BN5 = """Bean has five siblings: one older brother, one older sister, and three
younger sisters. He is closest to his younger sister, Rebecca, who is just one
year younger than he is."""
BN6 = """If chosen for this mission, Bean would be the shortest crew member at 5-
feet, 5-inches tall."""

# Collins
CG1 = """Collins is accustomed to being isolated for long periods of time. He
successfully completed two treks to the Antarctic in which he lived in tents and
a small building for the Antarctic winter-over period. Living in small spaces did
not bother him."""
CG2 = """Collins has a passion for languages. He is fluent in English, Spanish,
and French and proficient in Russian. Recently he started a class on Mandarin and
already he is nearly proficient. This ability will help him interact and bond
with international crew members."""
CG3 = """Collins has participated in spacewalks to repair damaged solar panels
and ammonia pump module. His peers have ranked him as the most competent crew
member during their spacewalks."""
CG4 = """Collins received the “Silver Snoopy” award for his contributions on
NASA’s STS-135 mission. This award is considered to be the highest honor of all
of the awards, given to the person who was the most outstanding team member,
creating a team environment focused on flight safety and overall team success."""
```

```
CB1 = """Surveys showed that Collins has snapped at coworkers on multiple
occasions. In one situation, a crewmember even went so far as to say would not
want to work with Collins on future missions."""
CB2 = """Collins has the least space-related training of all three potential
candidates."""
CB3 = """Recent blood work indicated that Collins may be developing above-average
blood pressure. Doctors advised another assessment before making any conclusive
determinations. However, if Collins does have heightened blood pressure, he would
have to take extra time to rest while on the mission. This would detract from his
time working on experiments, as well as his ability to do physically stressful
activities."""
CB4 = """Three weeks ago, Collins broke his ankle. As a result, he has not been
training as much and the current state of his cardiovascular performance is
behind compared to the other two candidates."""
CB5 = """In one incident during a mock extravehicular activity scenario, Collins
refused to follow an instruction from mission control. Fellow crew members were
unsure of why he did this since the request from mission control seemed
straightforward and helpful. Crewmembers reported being conflicted with how to
respond, which slowed their progress on the extravehicular activity scenario.
Only half of the tasks needed were completed on the simulated extravehicular
activity."""
CB6 = """An internal incident report indicates that Collins made offensive jokes
to a Chinese astronaut on a previous analog space mission that made the Chinese
astronaut and other crew members uncomfortable."""
CN1 = """Collins considers himself a coffee aficionado and his favorite brand of
coffee is Intelligentsia."""
CN2 = """Collins’ childhood friend completed NASA’s astronaut candidate training
program but was never selected for a space mission."""
CN3 = """Collins enjoys completing Sudoku puzzles to relax"""
CN4 = """Collins has a fraternal twin who works in finance in Houston, Texas."""

knowledge_bank = [
    {"label": "AG1", "item": """Astronaut Anders received the highest possible
scores on the military survival standards, an assessment that all astronauts must
complete in order to qualify for space missions. Anders' scores were
significantly higher than either candidate’s scores."""},
    {"label": "AG2", "item": """During Ander’s last ICE (Isolated and Confined in
Extreme environments) training sequence, a fellow crew member praised him for
being able to recognize others’ personal needs. In particular, he was recognized
for facilitating personal time and bringing the group together for light-hearted
fun."""},
    {"label": "AG3", "item": """While working at NASA, Anders coordinated and
completed a number of science projects. This expertise is important to Flight
Engineers and one of the reasons Anders was chosen for the previous space
mission. Anders did not disappoint. His ability to set up and coordinate science
```

```
projects allowed his crew to finish more than twice the number of projects than
anticipated."""},
    {"label": "AG4", "item": """Anders was one of 8 hikers to successfully summit
Mount Everest in 2005. Interviews with his comrades suggest Anders was the “glue”
that held the group together when things went wrong."""},
    {"label": "AG5", "item": """On previous missions, where lack of sleep was a
frequent occurrence, Anders was largely unaffected. Despite the sub-optimal
sleeping patterns, Anders' mental and physical health did not show signs of
suffering."""},
    {"label": "AG6", "item": """Data collected during training on simulated
emergencies shows that Anders is able to remain calm during stressful and
unexpected situations. He performs well under pressure."""},
    {"label": "AB1", "item": """Anders suffered a broken wrist 7 months ago
during a training simulation. The injury has healed, but will likely require
extra exercise to prevent forearm muscle atrophy in zero gravity. This may take
time away from him performing other team duties."""},
    {"label": "AB2", "item": """Most of Anders' space training experience has
been limited to simulations on Earth, with only one mission to the International
Space Station. Furthermore, this mission to Mars would be even longer than his
only long-distance flight experience--it is not clear if he will adapt well for a
long-duration mission."""},
    {"label": "AB3", "item": """Anders comes from a close-knit family. In diaries
collected during training, he reported feeling sad and frustrated when he was
unable to communicate with family members on a regular basis. This could be a
serious concern for his mental health on a mission to Mars because there will be
very little contact with Earth, and the little contact there is will be through
mission control."""},
    {"label": "AB4", "item": """Anders easily gets annoyed when explaining
concepts to others and becomes very short with them as a result."""},
    {"label": "AN1", "item": """Anders enjoys science fiction literature and has
self-published a novel about the start of a human colony in a distant galaxy. He
hopes to write a sequel when he retires."""},
    {"label": "AN2", "item": """As a child, Anders enjoyed attending Houstonli-
area sporting events with his parents. He was particularly fond of attending the
Texans football games."""},
    {"label": "AN3", "item": """Anders has a pet cat named Buzz, after Buzz
Aldrin. Anders met Buzz as a child around when he decided he wanted to be an
astronaut."""},
    {"label": "AN4", "item": """Anders used to own a Brown Labrador Retriever
named Bailey. When Anders participated in astronaut training and missions, Bailey
used to stay with his family."""},
    {"label": "BG1", "item": """On Expedition 31/32, where he helped refuel the
International Space Station (ISS), Bean encountered a breakdown in the
communication system between his vehicle and the ISS before arrival to the ISS.
```

```
Bean was credited for leading the extravehicular activity that identified and
fixed the issue, ultimately making the refuel possible."""},
    {"label": "BG2", "item": """While it is widely known that Bean holds a
bachelor’s degree in Aerospace Engineering and Cross-Cultural studies, few know
that he also pursued a minor in Counseling Psychology. Crewmembers have reported
that Bean is a great listener and often knows exactly what to say to make others
feel more comfortable."""},
    {"label": "BG3", "item": """Survey responses from Bean’s previous crew mates
reveal that he has a universal sense of humor. People always understand and enjoy
Bean’s jokes, so there is little risk of his humor being misinterpreted as being
negative. Crew members report that this really helps to de-escalate tense
situations."""},
    {"label": "BB1", "item": """Bean had a difficult time getting along with some
of his co-workers at Ames. He had asked his supervisor to intervene on several
occasions as they were not able to resolve the disagreements on their own."""},
    {"label": "BB2", "item": """Bean does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "BB3", "item": """Bean has a 2-year-old daughter, and colleagues
have noticed he is frustrated when he has to work late and cannot be home with
her. There is worry that being away for so long on this mission will
exponentially increase Bean's frustration."""},
    {"label": "BN1", "item": """Bean enjoys listening to the Beatles while
performing work on his own."""},
    {"label": "BN2", "item": """Bean is an avid fan of Shakespeare and often
quotes his work."""},
    {"label": "BN3", "item": """In college, Bean was on the squash team, where he
met his best friend Jack. Jack now lives in Providence, Rhode Island, but they
still talk on a weekly basis."""},
    {"label": "BN4", "item": """Astronaut Bean enjoys traveling and his favorite
place to vacation to is in the south of France."""},
    {"label": "BN5", "item": """Bean has five siblings: one older brother, one
older sister, and three younger sisters. He is closest to his younger sister,
Rebecca, who is just one year younger than he is."""},
    {"label": "BN6", "item": """If chosen for this mission, Bean would be the
shortest crew member at 5-feet, 5-inches tall."""},
    {"label": "CG1", "item": """Collins is accustomed to being isolated for long
periods of time. He successfully completed two treks to the Antarctic in which he
lived in tents and a small building for the Antarctic winter-over period. Living
in small spaces did not bother him."""},
    {"label": "CG2", "item": """Collins has a passion for languages. He is fluent
in English, Spanish, and French and proficient in Russian. Recently he started a
```

```
class on Mandarin and already he is nearly proficient. This ability will help him
interact and bond with international crew members."""},
    {"label": "CG3", "item": """Collins has participated in spacewalks to repair
damaged solar panels and ammonia pump module. His peers have ranked him as the
most competent crew member during their spacewalks."""},
    {"label": "CG4", "item": """Collins received the “Silver Snoopy” award for
his contributions on NASA’s STS-135 mission. This award is considered to be the
highest honor of all of the awards, given to the person who was the most
outstanding team member, creating a team environment focused on flight safety and
overall team success."""},
    {"label": "CB1", "item": """Surveys showed that Collins has snapped at
coworkers on multiple occasions. In one situation, a crewmember even went so far
as to say would not want to work with Collins on future missions."""},
    {"label": "CB2", "item": """Collins does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "CB3", "item": """Recent blood work indicated that Collins may be
developing above-average blood pressure. Doctors advised another assessment
before making any conclusive determinations. However, if Collins does have
heightened blood pressure, he would have to take extra time to rest while on the
mission. This would detract from his time working on experiments, as well as his
ability to do physically stressful activities."""},
    {"label": "CB4", "item": """Three weeks ago, Collins broke his ankle. As a
result, he has not been training as much and the current state of his
cardiovascular performance is behind compared to the other two candidates."""},
    {"label": "CB5", "item": """In one incident during a mock extravehicular
activity scenario, Collins refused to follow an instruction from mission control.
Fellow crew members were unsure of why he did this since the request from mission
control seemed straightforward and helpful. Crewmembers reported being conflicted
with how to respond, which slowed their progress on the extravehicular activity
scenario. Only half of the tasks needed were completed on the simulated
extravehicular activity."""},
    {"label": "CB6", "item": """An internal incident report indicates that
Collins made offensive jokes to a Chinese astronaut on a previous analog space
mission that made the Chinese astronaut and other crew members
uncomfortable."""},
    {"label": "CN1", "item": """Collins considers himself a coffee aficionado and
his favorite brand of coffee is Intelligentsia."""},
    {"label": "CN2", "item": """Collins’ childhood friend completed NASA’s
astronaut candidate training program but was never selected for a space
mission."""},
    {"label": "CN3", "item": """Collins enjoys completing Sudoku puzzles to
relax"""},
    {"label": "CN4", "item": """Collins has a fraternal twin who works in finance
in Houston, Texas."""}
]

#####AGENT 1: CMR######
cmr_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG6, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True}
]
cmr = CogAgent("CMR", cmr_KB)
cog_cmr = CogChain(cmr)

#####AGENT 2: MS1#####
ms1_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CB3, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CB5, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": CN4, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN4, "frequency" : 0 ,"initial": True},
    {"item": AG1, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True}
]
ms1 = CogAgent("MS1", ms1_KB)
cog_ms1 = CogChain(ms1)

#####AGENT 3: MS2#####
ms2_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
ms2 = CogAgent("MS2", ms2_KB)
cog_ms2 = CogChain(ms2)
```

```
#####AGENT 4: FE######
fe_KB = [
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CB2, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN5, "frequency" : 0 ,"initial": True},
    {"item": BN6, "frequency" : 0 ,"initial": True},
    {"item": AG5, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
fe = CogAgent("FE", fe_KB)
cog_fe = CogChain(fe)

cog_agents = [cog_cmr, cog_ms1, cog_ms2, cog_fe]
rounds = 10
decision_options = """Candidate 1: Samuel Anders
Candidate 2: John Bean
Candidate 3: Scott Collins"""
decision_rule = 'majority'
discussion_task = """ DISCUSSION TASK: You work as a member of NASA’s Astronaut
Crew Composition team. Your team analyzes all of the information obtained during
astronaut selection and training - including test scores, analog performance
measures, expert observations, peer evaluations, and medical  records. Armed with
this information, your team makes two kinds of evaluations. The first evaluation
is to determine which astronauts are ready for flight and which are not. Of those
who are deemed to be flight ready, your team composes entire crews for each
mission. In this second evaluation, you consider the complementarity and
compatibility of the astronauts as a team to select the 4, 5, or 6  astronauts
```

(depending on the mission) who will perform best as individuals, and as a crew. While composing crews for missions on the International Space Station has become routine, your team just received a special assignment from the Mission Planning Directorate that is anything but routine.
The Mars mission has been fast-tracked. Instead of the 2033 launch date that NASA had been aiming for, the launch date is being moved up to 2020. Intensive crew training for this unprecedented and bold journey must begin immediately. A 5-member crew will embark on this 3 year journey to Mars. This a truly international endeavor: the Russian, European, Indian, and Chinese Space Agencies will each select one astronaut for the mission. NASA will select the 5th crew member - an American astronaut - to join this international crew.
The crew members from the other countries have backgrounds in engineering, botany, geology and medicine. The crew still needs a flight engineer.
Your team was able to quickly narrow the field down to three veteran astronauts: Anders, Bean, and Collins. Each of these three have experience as a flight engineer on the ISS, have the necessary basic technical skills for the flight engineer position, meet basic requirements for the mission including height, Body Mass Index, and have passed a mental health screening. Now that you know the other potential crew members it is necessary for your team to determine which of the crew members will be the greatest asset on this particular mission.
The 3-year mission to Mars will be demanding. The success of the Mars mission will require the 5-person crew to live and work together in extremely close quarters, while isolated from the usual sources of social support - friends and family - for extended periods of time. It will require that the crew can adapt to the unknown and be able to handle whatever challenges will come their way. Once they arrive at Mars, the crew will have to complete complex feats such as landing and extravehicular activities. Because of a significant communication delay with Earth of up to 22 minutes each way, the crew will have to work autonomously. Additionally, the crew of the Mars mission consists of individuals representing multiple nationalities, making strong interpersonal skills all the more critical to the success of the mission.
The international space consortium has agreed to announce the names of the crew members from each of the 5 countries in just 10 days from now. Your team must deliver a recommendation for whom should be the American astronaut; years of hard work (and millions of dollars) rest on this decision.
To guide your difficult decision, you have compiled information from the personnel file for each candidate. The personnel files for each candidate are detailed and include information such as their performance scores and ranking on various training tasks, biographical information, evaluations and questionnaires completed by supervisors and individuals with whom they worked on prior trainings and missions, and questionnaires completed by the candidate and his family. A few human resource assistant have been working through the personnel files and have prepared briefs for you. Your team must now come together and discuss who should be selected for the mission.

```
Remember, the success of the mission to Mars depends on your team. Choose
quickly, but wisely!
"""
astronaut_candidate = Conversation(cog_agents, discussion_task, rounds,
decision_options, decision_rule, knowledge_bank)

astronaut_candidate.run_convo()

with open("agent_states.txt", "w") as states_file:
    for agent in astronaut_candidate.cog_agents:
        states_file.write(f"----- {agent.cog_agent.name} States -----\n")
        for element in agent.cog_agent.state:
            states_file.write(f"{element}\n")
        states_file.write("\n")  # Add a newline after each agent's states
```

Report.py

```
from CogAgent import CogAgent
from CogChain import CogChain

class Report:
    def __init__(self, cog_agents, knowledge_bank):
        self.initial_preferences = []
        self.turn_preferences = {}
        self.final_preferences = []
        self.turn_info_profiles = {}
        self.initial_info_profiles = []
        self.final_info_profiles = []
        self.speaker_frequency = []
        self.revealed_profile = []
        self.non_initial_contributions_percentages = []
        self.cog_agents = cog_agents
        self.knowledge_bank = knowledge_bank

    def retrieve_preferences(self, cog_agents):
        preferences = [f"{agent.cog_agent.name}: {agent.cog_agent.preference}"
for agent in cog_agents]
        return preferences

    def aggregate_round_profile(self, revealed_profile):
        aggregated_profile = {}
        all_shared_items = set()  # To keep track of all items shared in the
rounds
        item_frequencies = {}  # To track the frequency of each item
        for entry in revealed_profile:
            round_num = entry["round"]
            item = entry["item"]
            agent = entry["agent"]  # Include the contributing agent name

            if round_num not in aggregated_profile:
                aggregated_profile[round_num] = [(item, agent)]
            else:
                aggregated_profile[round_num].append((item, agent))

            all_shared_items.add(item)  # Add the item to the set

            # Update the frequency of the item
            item_frequencies[item] = item_frequencies.get(item, 0) + 1

        # Prepare a string to be used in the report
        result_str = "\n----- Turn Group Revealed Profile -----\n"
        for round_num, items_agents in sorted(aggregated_profile.items()):
            result_str += f"Round {round_num}:\n"
            for item, agent in items_agents:
                result_str += f"Info Item: {item}\nContributing Agent: {agent}\n"
            result_str += "\n"
        result_str += "\n----- Final Group Revealed Profile -----\n"
        # Include a list of all items shared in the rounds with frequency
        for item in all_shared_items:
            result_str += f"{item} (Frequency: {item_frequencies[item]})\n"
        return result_str

    def retrieve_agent_frequency(self, cog_agents):
        speaker_frequency = [f"{agent.cog_agent.name}:
{agent.cog_agent.frequency}" for agent in cog_agents]
        return speaker_frequency

    def generate_analysis_report(self, discussion_result):
        with open("analysis_report.txt", "w") as analysis_file:
            analysis_file.write("********************************************\n"
)
            analysis_file.write("ANALYSIS REPORT\n")
            analysis_file.write("********************************************\n"
)

            # 1. Calculate percentage of revealed profile items that are in the
knowledge bank
            all_shared_items = self.extract_all_shared_items()
            revealed_profile_percentage =
self.calculate_revealed_profile_percentage(all_shared_items)
            analysis_file.write("REVEALED PROFILE:\n")
            analysis_file.write(f"{revealed_profile_percentage:.2f}%\n\n")

            # 2. Calculate percentage of preference changes
            preference_change_percentage = self.calculate_preference_changes()
            analysis_file.write("PREFERENCE CHANGES:\n")
            analysis_file.write(f"{preference_change_percentage:.2f}%\n\n")

            # 3. Speaker distribution
            speaker_distribution = self.calculate_speaker_distribution()
            analysis_file.write("SPEAKER DISTRIBUTION:\n")
            for agent, percentage in speaker_distribution.items():
                analysis_file.write(f"{agent}: {percentage:.2f}%\n")
            analysis_file.write("\n")

            # 4. Decision made and group optimality rating
            if discussion_result != 'NO CONSENSUS':
                # decision_made, group_optimality =
self.calculate_decision_optimality(all_shared_items)
                group_optimality_revealed_profile =
self.calculate_revealed_profile_optimality(all_shared_items, discussion_result)
                group_optimality_knowledge_bank =
self.calculate_knowledge_bank_optimality(discussion_result)
                analysis_file.write("DECISION MADE:\n")
                analysis_file.write(f"Decision: {discussion_result}\n")
                analysis_file.write(f"Group Optimality (Revealed Profile):
{group_optimality_revealed_profile}\n")
                analysis_file.write(f"Group Optimality (Knowledge Bank):
{group_optimality['knowledge_bank']}\n")

                # Corrected individual optimality ratings

            else:
                analysis_file.write("No group decision made.\n")
                analysis_file.write("\nIndividual Optimality:\n")
                for agent_pref in self.final_preferences:
                    agent_name, preference = agent_pref.split(": ")
                    revealed_profile_optimality =
self.calculate_revealed_profile_optimality(all_shared_items, preference)
                    knowledge_bank_optimality =
self.calculate_knowledge_bank_optimality(preference)
                    analysis_file.write(f"{agent_name}:\n")
                    analysis_file.write(f"Revealed Profile:
{revealed_profile_optimality}\n")
                    analysis_file.write(f"Knowledge Bank:
{knowledge_bank_optimality}\n")

    def extract_all_shared_items(self):
        all_shared_items = set()
        for entry in self.revealed_profile:
            label = entry['item']
            for item in self.knowledge_bank:
                if item['label'] == label:
                    all_shared_items.add(item['label'])
                    break  # Stop searching once the item is found
        return all_shared_items

    def calculate_revealed_profile_percentage(self, all_shared_items):
    # Calculate the percentage of unique items in the revealed profile also in
the knowledge bank
        unique_revealed_items = set(entry['item'] for entry in
self.revealed_profile)
        total_revealed_items = len(unique_revealed_items)

        if total_revealed_items == 0:
            return 0.0

        kb_items = [entry['label'] for entry in self.knowledge_bank]
        kb_length = len(kb_items)

        percentage = (total_revealed_items / kb_length) * 100
        return percentage

    def calculate_preference_changes(self):
        initial_preferences = [agent.split(": ")[1] for agent in
self.initial_preferences]
        final_preferences = [agent.split(": ")[1] for agent in
self.final_preferences]
        changed_agents = [initial != final for initial, final in
zip(initial_preferences, final_preferences)]
        return (sum(changed_agents) / len(self.cog_agents)) * 100

    def calculate_speaker_distribution(self):
        total_turns = sum([int(freq.split(": ")[1]) for freq in
self.speaker_frequency])
        speaker_distribution = {freq.split(": ")[0]: (int(freq.split(": ")[1]) /
total_turns) * 100 for freq in self.speaker_frequency}
```

```
        return speaker_distribution

    def calculate_revealed_profile_optimality(self, all_shared_items,
agent_preference):
        ####FIX
        preference_counts = {'C': 0, 'A': 0, 'B': 0}
        for item in all_shared_items:
        # Check if the second item letter is 'G'
            if item[1] == 'G':
                # Determine the preference of the first item and update
preference counts
                if item[0] == 'C':
                    preference_counts['C'] += 1
                elif item[0] == 'A':
                    preference_counts['A'] += 1
                elif item[0] == 'B':
                    preference_counts['B'] += 1
        preference_ranking = {'C': 0, 'A': 0, 'B': 0}
        for preference, count in preference_counts.items():
            preference_ranking[preference] = count
        sorted_preferences = sorted(preference_ranking.items(), key=lambda x:
x[1], reverse=True)
        max_count = sorted_preferences[0][1]
        max_count_preferences = sum(1 for pref, count in sorted_preferences if
count == max_count)
        ranks = {}
        if max_count_preferences == 1:  # Only one preference has the maximum
count
            for preference, count in sorted_preferences:
                ranks[preference] = 'BEST' if count == max_count else 'WORST'
        else:  # Multiple preferences have the maximum count
            for preference, count in sorted_preferences:
                ranks[preference] = 'AVERAGE' if count == max_count else 'WORST'

            # Calculate second highest count
        second_highest_count = sorted_preferences[1][1]
            # Calculate how many preferences have the second highest count
        second_highest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == second_highest_count)

            # Assign 'AVERAGE' rank to preferences with the second highest count
        if second_highest_count_preferences > 1:
            for preference, count in sorted_preferences:
                if count == second_highest_count:
                    ranks[preference] = 'AVERAGE'
            # Calculate lowest count
        lowest_count = sorted_preferences[-1][1]

            # If lowest count is different from second highest count, assign
'WORST' rank
        if lowest_count != second_highest_count:
            lowest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == lowest_count)
            if lowest_count_preferences > 1:
                for preference, count in sorted_preferences:
                    if count == lowest_count:
                        ranks[preference] = 'WORST'

            # Extract agent preference label
        agent_preference_label = agent_preference.split()[1][0]

            # Assign the agent's preference ranking
        agent_preference_ranking = ranks.get(agent_preference_label, None)
        return agent_preference_ranking

    def calculate_knowledge_bank_optimality(self, preference):
        # Define the fixed knowledge bank optimality ratings
        knowledge_bank_ratings = {'Samuel Anders': 'BEST', 'John Bean':
'AVERAGE', 'Scott Collins': 'WORST'}
        return knowledge_bank_ratings.get(preference, 'UNKNOWN')

    def generate_preference_report(self, discussion_result, rounds,
decision_rule):
        with open("report.txt", "w") as report_file:
            report_file.write("*********************************************\n")
            report_file.write("DISCUSSION REPORT\n")
            report_file.write("*********************************************\n")
            report_file.write(f"Number of Rounds: {rounds}\n")
            report_file.write(f"Decision Rule: {decision_rule}\n")
            report_file.write
            if discussion_result == 'NO CONSENSUS':
                report_file.write(f"Discussion Result: Failure to reach consensus
in the specified number of rounds.\n")
            else:
                report_file.write(f"Discussion Result: Consensus reached in
{rounds} rounds!\n")
                report_file.write(f"Agreed upon goal: {discussion_result}\n")
            report_file.write("\n*********************************************\n"
)
```

```
            report_file.write("PREFERENCE INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("----- Initial Preference Distribution -----\n")
            for i, entry in enumerate(self.initial_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("----- Turn Preference Distribution -----\n")
            for round_num, preferences in self.turn_preferences.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(preferences, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Preference Distribution -----\n")
            for i, entry in enumerate(self.final_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("INFO ITEM INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("\n----- Initial Info Profiles -----\n")
            for i, entry in enumerate(self.initial_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n----- Turn Info Profiles -----\n")
            for round_num, info_profile in self.turn_info_profiles.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(info_profile, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Info Profiles -----\n")
            for i, entry in enumerate(self.final_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            aggregated_profile_str =
self.aggregate_round_profile(self.revealed_profile)
            report_file.write(f"{aggregated_profile_str}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("CONVO INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("-----Speaker Frequency -----\n")
            for i, entry in enumerate(self.speaker_frequency, start=1):
                report_file.write(f"{entry}\n")
            report_file.write("\n----- Non-Initial Contributions-----\n")
            for agent_name, percentage in
self.non_initial_contributions_percentages.items():
```

```
                report_file.write(f"{agent_name}: {percentage}%\n")

    def kb_map(self, knowledge_base, knowledge_bank):
        labels = []
        for kb_entry in knowledge_base:
            kb_item = kb_entry["item"]
            for bank_entry in knowledge_bank:
                bank_item = bank_entry["item"]
                if kb_item == bank_item:
                    labels.append(bank_entry["label"])
                    break  # Stop searching once a match is found
        return labels

    def retrieve_info_profiles(self, cog_agents, knowledge_bank):
        profiles = []
        for agent in cog_agents:
            knowledge_base = agent.cog_agent.knowledge_base
            profile_labels = self.kb_map(knowledge_base, knowledge_bank)
            profiles += [f"{agent.cog_agent.name}: {profile_labels}"]
        return profiles

    def calculate_non_initial_contributions_percentage(self, agent_name,
revealed_profile):
        agent_contributions = [entry for entry in revealed_profile if
entry['agent'] == agent_name]
        total_contributions = len(agent_contributions)

        if total_contributions == 0:
            return None  # Return None for agents who did not contribute

        non_initial_contributions = sum(
            1 for entry in agent_contributions if not
self.is_item_in_initial_kb(agent_name, entry['item']))

        if total_contributions == non_initial_contributions:
            return 0.0  # All contributions were non-initial

        percentage = (non_initial_contributions / (total_contributions -
non_initial_contributions)) * 100
        return round(percentage, 2)

    def is_item_in_initial_kb(self, agent_name, item):
        for agent in self.cog_agents:
            if agent.cog_agent.name == agent_name:
                return agent.cog_agent.was_item_in_initial_kb(item)
        return False
```

APPENDIX F: MEMORYCOG CODE

CogAgent.py

```
class CogAgent:
    def __init__(self, name, knowledge_base):
        self.name = name
        self.knowledge_base = knowledge_base
        self.initial_knowledge_base = knowledge_base.copy()
        self.preference = ""
        self.state = []
        self.frequency = 0

    def was_item_in_initial_kb(self, item):
        return any(entry['item'] == item for entry in
self.initial_knowledge_base)

    def state_snap(self):
        snapshot = ""
        snapshot += f"Preference: {self.preference}\n"
        return snapshot

    def update_agent_state(self, state):
        self.state.append(state)
```

CogChain.py

```
import os
import re
os.environ["OPENAI_API_KEY"]='sk-
iRdjQCppyWt3WblPYd24T3BlbkFJYsTrZOxSnVoajtnSoI5U'
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
llm = ChatOpenAI(model="gpt-4-0125-preview")
import random
import sys
from CogAgent import CogAgent

class CogChain:
    def __init__(self, cog_agent):
        self.cog_agent = cog_agent
        self.convo_history = []
        self.name = cog_agent.name


    def kb_print(self, kb):
        random.shuffle(kb)
        kb_str = "KNOWLEDGE BASE:\n"
        for item in kb:
            item_str = f"{item['item']}\n\n"
            kb_str += item_str
        return kb_str

    def preference_chain(self, task, kb, decision_options):
        kb_string = self.kb_print(kb)
        update_goal = PromptTemplate(
            input_variables = ["task","updated_kb", "decision_options"],
            template = """{task}. You have access to a list of facts, called a
'knowledge base', to help guide your preference. Here is the most recently
updated list:
            {updated_kb}
            Based on the above facts, choose one alternative from the decision
options. You must choose from the provided options only.
            {decision_options}
            Use this format for your output: PREFERENCE: (Insert candidate name)
\n EXPLANATION: (justify your decision, limit to 300 tokens).""")
        goal_update = LLMChain(llm=llm, prompt=update_goal)
        updated_goal = goal_update.invoke({
        "task": task,
        "updated_kb": kb_string,
        "decision_options": decision_options})
        goal_match = re.search(r'PREFERENCE: (.+)', updated_goal['text'])
        if goal_match:
            new_goal = goal_match.group(1)
            self.cog_agent.preference = new_goal
        return updated_goal["text"]

    def contribution_chain(self, task, context, ris):
        generate_contribution = PromptTemplate(
            input_variables = ["goal", "task", "kb", "context"],
            template = """{task} Your team mate just contributed: {context}.
            It is your turn in the discussion. Choose a fact from the knowledge
base and provide it (exactly as phrased) alongside your commentary. You are
likely to choose a fact that aligns with your preference. You may also directly
address the team member who just contributed, if you want to do so.
            Preference: {goal}
            {ris}
            Output format: INFORMATION ITEM: (insert fact exactly as given) \n
COMMENTARY: (insert your commentary)""")
        gen_contribution = LLMChain(llm=llm, prompt=generate_contribution)
        contribution = gen_contribution.invoke({
            "goal": self.cog_agent.preference,
            "task": task,
            "ris" : ris,
            "context": context})
        return contribution['text']

    def process_message(self, task, context, decision_options):
        #Initial Snap
        initial_snap = self.cog_agent.state_snap()
        #Take in Context, Update KB
        self.knowledge_base_chain(context)
        #Update Goal
        goal_result = self.preference_chain(task, self.cog_agent.knowledge_base,
decision_options)
        #Final Snap
        final_snap = self.cog_agent.state_snap()
        #Generate process report
        process_report = f"---------PROCESS REPORT----------\n"
        process_report += f"Initial Agent State: " + initial_snap + "\n"
        process_report += f"***\n"
        process_report += f"Context: " + context + "\n"
        process_report += f"***\n"
        process_report += f"Final Agent State: " + final_snap + "\n"
        process_report += f""+ goal_result + "\n"
        self.cog_agent.update_agent_state(process_report)

    def knowledge_base_chain(self, context):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        # Find if the item is in the knowledge base
        item_found = False
        for item in self.cog_agent.knowledge_base:
            if item['item'] == info_item:
                item['frequency'] += 1
                item_found = True
                break

        # If the item is not found, add it to the knowledge base
        if not item_found:
            self.cog_agent.knowledge_base.append({"item": info_item, "frequency":
1, "initial": False})

        # Use kb_print to get the updated knowledge base string
        updated_kb_str = self.kb_print(self.cog_agent.knowledge_base)
        return updated_kb_str

    def relevant_item_set_chain(self, knowledge_bank):
        preference = self.cog_agent.preference
        preference_label = preference.split()[1][0]
        labels = self.kb_map(self.cog_agent.knowledge_base, knowledge_bank)
        relevant_labels = [label for label in labels if label[0] ==
preference_label]
        relevant_items = [bank_entry["item"] for bank_entry in knowledge_bank if
bank_entry["label"] in relevant_labels]
```

```
        relevant_items = [next((entry for entry in self.cog_agent.knowledge_base
if entry["item"] == item), None) for item in relevant_items]
        ris = self.kb_print(relevant_items)
        return ris

    def first_turn_contribution(self, discussion_task):
        initial_snap = self.cog_agent.state_snap()
        goal = self.cog_agent.preference
        knowledge_base = self.cog_agent.knowledge_base
        kb_string = self.kb_print(knowledge_base)
        generate_first_turn_contribution = PromptTemplate(
            input_variables = ["goal", "kb", "task", "context"],
            template = """{task}. You are starting the discussion. Choose a fact
from the knowledge base and provide it (exactly as phrased) alongside your
commentary. You are likely to choose a fact that aligns with your preference.
            Preference: {goal}
            {kb}
            Output format: INFORMATION ITEM: (insert fact) \n COMMENTARY: (insert
your commentary)""")
        gen_first_turn_contribution = LLMChain(llm=llm,
prompt=generate_first_turn_contribution)
        contribution = gen_first_turn_contribution.invoke({
            "goal": goal,
            "kb": kb_string,
            "task": discussion_task,})
        updated_kb = self.knowledge_base_chain(contribution['text'])
        final_snap = self.cog_agent.state_snap()
        #Generate state report
        state_report = f"Contribution: " + contribution['text'] + "\n"
        state_report += f"---------------------------------------\n"
        state_report += f"Final Agent State: " + final_snap + "\n"
        self.cog_agent.update_agent_state(state_report)
        return contribution['text']

    def invoke_convo(self, sample_context, discussion_task, knowledge_bank):
        kb = self.cog_agent.knowledge_base
        goal = self.cog_agent.preference
        ris = self.relevant_item_set_chain(knowledge_bank)
        contribution_result =
self.contribution_chain(discussion_task,sample_context, ris)
        updated_kb = self.knowledge_base_chain(contribution_result)
        ###GENERATE TURN REPORT
        final_snap = self.cog_agent.state_snap()
        self.cog_agent.update_agent_state(final_snap)
        return contribution_result

    def kb_map(self, knowledge_base, knowledge_bank):
        labels = []
        for kb_entry in knowledge_base:
            kb_item = kb_entry["item"]
            for bank_entry in knowledge_bank:
                bank_item = bank_entry["item"]
                if kb_item == bank_item:
                    labels.append(bank_entry["label"])
                    break  # Stop searching once a match is found
        return labels
```

Conversation.py

```
import os
import re
import random
import sys
from Report import Report
from CogAgent import CogAgent
from CogChain import CogChain

class Conversation:
    def __init__(self, cog_agents, discussion_task, rounds, decision_options,
decision_rule, knowledge_bank):
        self.cog_agents = cog_agents
        self.discussion_task = discussion_task
        self.rounds = rounds
        self.convo_history = []
        self.decision_options = decision_options
        self.decision_rule = decision_rule
        self.knowledge_bank = knowledge_bank
        self.report = Report(cog_agents, knowledge_bank)
        self.revealed_profile = []

    def broadcast_message(self, cog_agents, context, agent_in):
        for agent in cog_agents:
            if agent.name != agent_in:
                agent.process_message(self.discussion_task, context,
self.decision_options)

    def update_revealed_profile(self, context, current_agent, round):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        rp = [{"""item""": info_item,
            """agent""": current_agent.name,
            """round""": round}]
        print("----")
        print(rp)
        print("----")
        info_item_label = self.report.kb_map(rp, self.knowledge_bank)
        # Add the info_item to the revealed_profile
        self.revealed_profile.append({
            "item": info_item_label[0],
            "agent": current_agent.name,
            "round": round})
        return self.revealed_profile

    def consensus(self):
        # Check for consensus based on the decision_rule
        if self.decision_rule == 'unanimous':
            # Check if all agents have the same goal
            goals = {agent.cog_agent.preference for agent in self.cog_agents}
            if len(goals) == 1:
                return goals.pop()  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        elif self.decision_rule == 'majority':
            # Check if there is a majority agreement on the goal
            goal_counts = {agent.cog_agent.preference: 0 for agent in
self.cog_agents}
            for agent in self.cog_agents:
                goal_counts[agent.cog_agent.preference] += 1

            max_goal = max(goal_counts, key=goal_counts.get)

            if goal_counts[max_goal] > len(self.cog_agents) / 2:
                return max_goal  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        else:
            return 'NO CONSENSUS'

    def next_speaker(self, current_agent, broadcast_agents):
        next_speakers = [agent for agent in broadcast_agents]
        if next_speakers:
            # If more than 2+ agents, choose randomly
            next_speaker = random.choice(next_speakers)
        return next_speaker

    def run_convo(self):
        #initialize agent preferences
        for agent in self.cog_agents:
            # Call preference_chain for each agent
            preference = agent.preference_chain(
                self.discussion_task, agent.cog_agent.knowledge_base,
self.decision_options)
            agent.cog_agent.update_agent_state(preference)

        # Store Initial Profiles
        self.report.initial_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        # Store Initial Preferences
        self.report.initial_preferences =
self.report.retrieve_preferences(self.cog_agents)
```

```
        # Initial Turn Contribution
        agent1 = random.choice(self.cog_agents)
        contribution1 = agent1.first_turn_contribution(self.discussion_task)
        agent1.cog_agent.frequency += 1

        # Add first contribution to chat history, ADD ITEM TO REVEALED_PROFILE
        self.convo_history.append(contribution1)
        self.update_revealed_profile(contribution1, agent1, 0)

        # Broadcast first contribution to other agents
        broadcast_agents = [agent for agent in self.cog_agents if agent !=
agent1]
        self.broadcast_message(broadcast_agents, contribution1, agent1.name)

        # Store Turn Preference
        self.report.turn_preferences[0] =
self.report.retrieve_preferences(self.cog_agents)
        # Store Turn Info Profiles
        self.report.turn_info_profiles[0] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        print(agent1.name)
        print(contribution1)
        current_agent = self.next_speaker(agent1, broadcast_agents)
        roundHolder = 0
        for round_num in range(1, self.rounds):
            context = self.convo_history[-1]
            contribution = current_agent.invoke_convo(context,
self.discussion_task, self.knowledge_bank)
            current_agent.cog_agent.frequency += 1
            print(current_agent.name)
            print(contribution)
            self.convo_history.append(contribution)
            #ADD ITEM TO REVEALED PROFILE
            self.update_revealed_profile(contribution, current_agent, round_num)
            # Broadcast to other agents
            broadcast_agents = [agent for agent in self.cog_agents if agent !=
current_agent]
            self.broadcast_message(broadcast_agents, contribution,
current_agent.name)
            # Store Turn Preference
            self.report.turn_preferences[round_num] =
self.report.retrieve_preferences(self.cog_agents)
             # Store Turn Info Profiles
            self.report.turn_info_profiles[round_num] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
            decision_result = self.consensus()
            if decision_result != 'NO CONSENSUS':
                print(f"Consensus reached! Agreed upon goal: {decision_result}")
                break  # Exit the loop if consensus is reached
            #IF agents do not agree, select next speaker
            current_agent = self.next_speaker(current_agent, broadcast_agents)
            roundHolder = round_num
        decision_result = self.consensus()
        # Store Final Preferences
        self.report.final_preferences =
self.report.retrieve_preferences(self.cog_agents)
        # Store Final Info Profiles
        self.report.final_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
        # Store Agent Speaker Frequencies
        self.report.speaker_frequency =
self.report.retrieve_agent_frequency(self.cog_agents)
        self.report.revealed_profile = self.revealed_profile
        # Calc & Store non-initial contribution %
        non_initial_percentages = {}
        for agent in self.cog_agents:
            agent_name = agent.cog_agent.name
            non_initial_percentage =
self.report.calculate_non_initial_contributions_percentage(agent_name,
self.revealed_profile)

            if non_initial_percentage is not None:
                non_initial_percentages[agent_name] = non_initial_percentage
        self.report.non_initial_contributions_percentages =
non_initial_percentages
        # Generate Report
        self.report.generate_preference_report(decision_result, self.rounds,
self.decision_rule, roundHolder)
        self.report.generate_analysis_report(decision_result, roundHolder)
        with open("conversation_log.txt", "w") as log_file:
            log_file.write("----- Conversation Thread -----\n")
            for i, entry in enumerate(self.convo_history, start=1):
                log_file.write(f"Turn {i}:\n{entry}\n")
```

Discuss.py

```
from Conversation import Conversation
from CogAgent import CogAgent
from Report import Report
from CogChain import CogChain

####INFO PROFILE####
# Anders
AG1 = """Astronaut Anders received the highest possible scores on the military
survival standards, an assessment that all astronauts must complete in order to
qualify for space missions. Anders' scores were significantly higher than either
candidate’s scores."""
AG2 = """During Ander’s last ICE (Isolated and Confined in Extreme environments)
training sequence, a fellow crew member praised him for being able to recognize
others’ personal needs. In particular, he was recognized for facilitating
personal time and bringing the group together for light-hearted fun."""
```

```
AG3 = """While working at NASA, Anders coordinated and completed a number of
science projects. This expertise is important to Flight Engineers and one of the
reasons Anders was chosen for the previous space mission. Anders did not
disappoint. His ability to set up and coordinate science projects allowed his
crew to finish more than twice the number of projects than anticipated."""
AG4 = """Anders was one of 8 hikers to successfully summit Mount Everest in 2005.
Interviews with his comrades suggest Anders was the “glue” that held the group
together when things went wrong."""
AG5 = """On previous missions, where lack of sleep was a frequent occurrence,
Anders was largely unaffected. Despite the sub-optimal sleeping patterns, Anders'
mental and physical health did not show signs of suffering."""
AG6 = """Data collected during training on simulated emergencies shows that
Anders is able to remain calm during stressful and unexpected situations. He
performs well under pressure."""
AB1 = """Anders suffered a broken wrist 7 months ago during a training
simulation. The injury has healed, but will likely require extra exercise to
prevent forearm muscle atrophy in zero gravity. This may take time away from him
performing other team duties."""
AB2 = """Most of Anders' space training experience has been limited to
simulations on Earth, with only one mission to the International Space Station.
Furthermore, this mission to Mars would be even longer than his only long-
distance flight experience--it is not clear if he will adapt well for a long-
duration mission."""
AB3 = """Anders comes from a close-knit family. In diaries collected during
training, he reported feeling sad and frustrated when he was unable to
communicate with family members on a regular basis. This could be a serious
concern for his mental health on a mission to Mars because there will be very
little contact with Earth, and the little contact there is will be through
mission control."""
AB4 = """Anders easily gets annoyed when explaining concepts to others and
becomes very short with them as a result."""
AN1 = """Anders enjoys science fiction literature and has self-published a novel
about the start of a human colony in a distant galaxy. He hopes to write a sequel
when he retires."""
AN2 = """As a child, Anders enjoyed attending Houstonli-area sporting events with
his parents. He was particularly fond of attending the Texans football games."""
AN3 = """Anders has a pet cat named Buzz, after Buzz Aldrin. Anders met Buzz as a
child around when he decided he wanted to be an astronaut."""
AN4 = """Anders used to own a Brown Labrador Retriever named Bailey. When Anders
participated in astronaut training and missions, Bailey used to stay with his
family."""

# Bean
BG1 = """On Expedition 31/32, where he helped refuel the International Space
Station (ISS), Bean encountered a breakdown in the communication system between
his vehicle and the ISS before arrival to the ISS. Bean was credited for leading
the extravehicular activity that identified and fixed the issue, ultimately
making the refuel possible."""
BG2 = """While it is widely known that Bean holds a bachelor’s degree in
Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a
minor in Counseling Psychology. Crewmembers have reported that Bean is a great
```

```
listener and often knows exactly what to say to make others feel more
comfortable."""
BG3 = """Survey responses from Bean’s previous crew mates reveal that he has a
universal sense of humor. People always understand and enjoy Bean’s jokes, so
there is little risk of his humor being misinterpreted as being negative. Crew
members report that this really helps to de-escalate tense situations."""
BB1 = """Bean had a difficult time getting along with some of his co-workers at
Ames. He had asked his supervisor to intervene on several occasions as they were
not able to resolve the disagreements on their own."""
BB2 = """Bean does not always plan his work well, and sometimes his pace of work
is too slow. This has become problematic not only in his mission to the ISS, but
also in training simulations. Often times, the maintenance work and data
collection the astronauts engage in on these missions requires high levels of
interdependency - this can be jeopardized if one person works at a slower pace
than the rest of the crew members."""
BB3 = """Bean has a 2-year-old daughter, and colleagues have noticed he is
frustrated when he has to work late and cannot be home with her. There is worry
that being away for so long on this mission will exponentially increase Bean's
frustration."""
BN1 = """Bean enjoys listening to the Beatles while performing work on his
own."""
BN2 = """Bean is an avid fan of Shakespeare and often quotes his work."""
BN3 = """In college, Bean was on the squash team, where he met his best friend
Jack. Jack now lives in Providence, Rhode Island, but they still talk on a weekly
basis."""
BN4 = """Astronaut Bean enjoys traveling and his favorite place to vacation to is
in the south of France."""
BN5 = """Bean has five siblings: one older brother, one older sister, and three
younger sisters. He is closest to his younger sister, Rebecca, who is just one
year younger than he is."""
BN6 = """If chosen for this mission, Bean would be the shortest crew member at 5-
feet, 5-inches tall."""

# Collins
CG1 = """Collins is accustomed to being isolated for long periods of time. He
successfully completed two treks to the Antarctic in which he lived in tents and
a small building for the Antarctic winter-over period. Living in small spaces did
not bother him."""
CG2 = """Collins has a passion for languages. He is fluent in English, Spanish,
and French and proficient in Russian. Recently he started a class on Mandarin and
already he is nearly proficient. This ability will help him interact and bond
with international crew members."""
CG3 = """Collins has participated in spacewalks to repair damaged solar panels
and ammonia pump module. His peers have ranked him as the most competent crew
member during their spacewalks."""
CG4 = """Collins received the “Silver Snoopy” award for his contributions on
NASA’s STS-135 mission. This award is considered to be the highest honor of all
of the awards, given to the person who was the most outstanding team member,
creating a team environment focused on flight safety and overall team success."""
CB1 = """Surveys showed that Collins has snapped at coworkers on multiple
occasions. In one situation, a crewmember even went so far as to say would not
want to work with Collins on future missions."""
```

```
CB2 = """Collins has the least space-related training of all three potential
candidates."""
CB3 = """Recent blood work indicated that Collins may be developing above-average
blood pressure. Doctors advised another assessment before making any conclusive
determinations. However, if Collins does have heightened blood pressure, he would
have to take extra time to rest while on the mission. This would detract from his
time working on experiments, as well as his ability to do physically stressful
activities."""
CB4 = """Three weeks ago, Collins broke his ankle. As a result, he has not been
training as much and the current state of his cardiovascular performance is
behind compared to the other two candidates."""
CB5 = """In one incident during a mock extravehicular activity scenario, Collins
refused to follow an instruction from mission control. Fellow crew members were
unsure of why he did this since the request from mission control seemed
straightforward and helpful. Crewmembers reported being conflicted with how to
respond, which slowed their progress on the extravehicular activity scenario.
Only half of the tasks needed were completed on the simulated extravehicular
activity."""
CB6 = """An internal incident report indicates that Collins made offensive jokes
to a Chinese astronaut on a previous analog space mission that made the Chinese
astronaut and other crew members uncomfortable."""
CN1 = """Collins considers himself a coffee aficionado and his favorite brand of
coffee is Intelligentsia."""
CN2 = """Collins' childhood friend completed NASA's astronaut candidate training
program but was never selected for a space mission."""
CN3 = """Collins enjoys completing Sudoku puzzles to relax"""
CN4 = """Collins has a fraternal twin who works in finance in Houston, Texas."""

knowledge_bank = [
    {"label": "AG1", "item": """Astronaut Anders received the highest possible
scores on the military survival standards, an assessment that all astronauts must
complete in order to qualify for space missions. Anders' scores were
significantly higher than either candidate's scores."""},
    {"label": "AG2", "item": """During Ander's last ICE (Isolated and Confined in
Extreme environments) training sequence, a fellow crew member praised him for
being able to recognize others' personal needs. In particular, he was recognized
for facilitating personal time and bringing the group together for light-hearted
fun."""},
    {"label": "AG3", "item": """While working at NASA, Anders coordinated and
completed a number of science projects. This expertise is important to Flight
Engineers and one of the reasons Anders was chosen for the previous space
mission. Anders did not disappoint. His ability to set up and coordinate science
projects allowed his crew to finish more than twice the number of projects than
anticipated."""},
    {"label": "AG4", "item": """Anders was one of 8 hikers to successfully summit
Mount Everest in 2005. Interviews with his comrades suggest Anders was the "glue"
that held the group together when things went wrong."""},
    {"label": "AG5", "item": """On previous missions, where lack of sleep was a
frequent occurrence, Anders was largely unaffected. Despite the sub-optimal
sleeping patterns, Anders' mental and physical health did not show signs of
suffering."""},
    {"label": "AG6", "item": """Data collected during training on simulated
emergencies shows that Anders is able to remain calm during stressful and
unexpected situations. He performs well under pressure."""},
    {"label": "AB1", "item": """Anders suffered a broken wrist 7 months ago
during a training simulation. The injury has healed, but will likely require
extra exercise to prevent forearm muscle atrophy in zero gravity. This may take
time away from him performing other team duties."""},
    {"label": "AB2", "item": """Most of Anders' space training experience has
been limited to simulations on Earth, with only one mission to the International
Space Station. Furthermore, this mission to Mars would be even longer than his
only long-distance flight experience--it is not clear if he will adapt well for a
long-duration mission."""},
    {"label": "AB3", "item": """Anders comes from a close-knit family. In diaries
collected during training, he reported feeling sad and frustrated when he was
unable to communicate with family members on a regular basis. This could be a
serious concern for his mental health on a mission to Mars because there will be
very little contact with Earth, and the little contact there is will be through
mission control."""},
    {"label": "AB4", "item": """Anders easily gets annoyed when explaining
concepts to others and becomes very short with them as a result."""},
    {"label": "AN1", "item": """Anders enjoys science fiction literature and has
self-published a novel about the start of a human colony in a distant galaxy. He
hopes to write a sequel when he retires."""},
    {"label": "AN2", "item": """As a child, Anders enjoyed attending Houstonli-
area sporting events with his parents. He was particularly fond of attending the
Texans football games."""},
    {"label": "AN3", "item": """Anders has a pet cat named Buzz, after Buzz
Aldrin. Anders met Buzz as a child around when he decided he wanted to be an
astronaut."""},
    {"label": "AN4", "item": """Anders used to own a Brown Labrador Retriever
named Bailey. When Anders participated in astronaut training and missions, Bailey
used to stay with his family."""},
    {"label": "BG1", "item": """On Expedition 31/32, where he helped refuel the
International Space Station (ISS), Bean encountered a breakdown in the
communication system between his vehicle and the ISS before arrival to the ISS.
Bean was credited for leading the extravehicular activity that identified and
fixed the issue, ultimately making the refuel possible."""},
    {"label": "BG2", "item": """While it is widely known that Bean holds a
bachelor’s degree in Aerospace Engineering and Cross-Cultural studies, few know
that he also pursued a minor in Counseling Psychology. Crewmembers have reported
that Bean is a great listener and often knows exactly what to say to make others
feel more comfortable."""},
    {"label": "BG3", "item": """Survey responses from Bean’s previous crew mates
reveal that he has a universal sense of humor. People always understand and enjoy
Bean’s jokes, so there is little risk of his humor being misinterpreted as being
negative. Crew members report that this really helps to de-escalate tense
situations."""},
    {"label": "BB1", "item": """Bean had a difficult time getting along with some
of his co-workers at Ames. He had asked his supervisor to intervene on several
occasions as they were not able to resolve the disagreements on their own."""},
    {"label": "BB2", "item": """Bean does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
```

```
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "BB3", "item": """Bean has a 2-year-old daughter, and colleagues
have noticed he is frustrated when he has to work late and cannot be home with
her. There is worry that being away for so long on this mission will
exponentially increase Bean's frustration."""},
    {"label": "BN1", "item": """Bean enjoys listening to the Beatles while
performing work on his own."""},
    {"label": "BN2", "item": """Bean is an avid fan of Shakespeare and often
quotes his work."""},
    {"label": "BN3", "item": """In college, Bean was on the squash team, where he
met his best friend Jack. Jack now lives in Providence, Rhode Island, but they
still talk on a weekly basis."""},
    {"label": "BN4", "item": """Astronaut Bean enjoys traveling and his favorite
place to vacation to is in the south of France."""},
    {"label": "BN5", "item": """Bean has five siblings: one older brother, one
older sister, and three younger sisters. He is closest to his younger sister,
Rebecca, who is just one year younger than he is."""},
    {"label": "BN6", "item": """If chosen for this mission, Bean would be the
shortest crew member at 5-feet, 5-inches tall."""},
    {"label": "CG1", "item": """Collins is accustomed to being isolated for long
periods of time. He successfully completed two treks to the Antarctic in which he
lived in tents and a small building for the Antarctic winter-over period. Living
in small spaces did not bother him."""},
    {"label": "CG2", "item": """Collins has a passion for languages. He is fluent
in English, Spanish, and French and proficient in Russian. Recently he started a
class on Mandarin and already he is nearly proficient. This ability will help him
interact and bond with international crew members."""},
    {"label": "CG3", "item": """Collins has participated in spacewalks to repair
damaged solar panels and ammonia pump module. His peers have ranked him as the
most competent crew member during their spacewalks."""},
    {"label": "CG4", "item": """Collins received the “Silver Snoopy” award for
his contributions on NASA’s STS-135 mission. This award is considered to be the
highest honor of all of the awards, given to the person who was the most
outstanding team member, creating a team environment focused on flight safety and
overall team success."""},
    {"label": "CB1", "item": """Surveys showed that Collins has snapped at
coworkers on multiple occasions. In one situation, a crewmember even went so far
as to say would not want to work with Collins on future missions."""},
    {"label": "CB2", "item": """Collins does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "CB3", "item": """Recent blood work indicated that Collins may be
developing above-average blood pressure. Doctors advised another assessment
before making any conclusive determinations. However, if Collins does have
heightened blood pressure, he would have to take extra time to rest while on the
```

```
mission. This would detract from his time working on experiments, as well as his
ability to do physically stressful activities."""},
    {"label": "CB4", "item": """Three weeks ago, Collins broke his ankle. As a
result, he has not been training as much and the current state of his
cardiovascular performance is behind compared to the other two candidates."""},
    {"label": "CB5", "item": """In one incident during a mock extravehicular
activity scenario, Collins refused to follow an instruction from mission control.
Fellow crew members were unsure of why he did this since the request from mission
control seemed straightforward and helpful. Crewmembers reported being conflicted
with how to respond, which slowed their progress on the extravehicular activity
scenario. Only half of the tasks needed were completed on the simulated
extravehicular activity."""},
    {"label": "CB6", "item": """An internal incident report indicates that
Collins made offensive jokes to a Chinese astronaut on a previous analog space
mission that made the Chinese astronaut and other crew members
uncomfortable."""},
    {"label": "CN1", "item": """Collins considers himself a coffee aficionado and
his favorite brand of coffee is Intelligentsia."""},
    {"label": "CN2", "item": """Collins’ childhood friend completed NASA’s
astronaut candidate training program but was never selected for a space
mission."""},
    {"label": "CN3", "item": """Collins enjoys completing Sudoku puzzles to
relax"""},
    {"label": "CN4", "item": """Collins has a fraternal twin who works in finance
in Houston, Texas."""}
]

#####AGENT 1: CMR######
cmr_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG6, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True}
]
```

```
cmr = CogAgent("CMR", cmr_KB)
cog_cmr = CogChain(cmr)

#####AGENT 2: MS1#####
ms1_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CB3, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CB5, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": CN4, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN4, "frequency" : 0 ,"initial": True},
    {"item": AG1, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True}
]
ms1 = CogAgent("MS1", ms1_KB)
cog_ms1 = CogChain(ms1)

#####AGENT 3: MS2#####
ms2_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
ms2 = CogAgent("MS2", ms2_KB)
cog_ms2 = CogChain(ms2)

#####AGENT 4: FE######
fe_KB = [
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CB2, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN5, "frequency" : 0 ,"initial": True},
    {"item": BN6, "frequency" : 0 ,"initial": True},
    {"item": AG5, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
fe = CogAgent("FE", fe_KB)
cog_fe = CogChain(fe)

cog_agents = [cog_cmr, cog_ms1, cog_ms2, cog_fe]
rounds = 10
decision_options = """Candidate 1: Samuel Anders
Candidate 2: John Bean
Candidate 3: Scott Collins"""
decision_rule = 'majority'
discussion_task = """ DISCUSSION TASK: You work as a member of NASA’s Astronaut
Crew Composition team. Your team analyzes all of the information obtained during
astronaut selection and training - including test scores, analog performance
measures, expert observations, peer evaluations, and medical  records. Armed with
this information, your team makes two kinds of evaluations. The first evaluation
is to determine which astronauts are ready for flight and which are not. Of those
who are deemed to be flight ready, your team composes entire crews for each
mission. In this second evaluation, you consider the complementarity and
compatibility of the astronauts as a team to select the 4, 5, or 6  astronauts
```

```
(depending on the mission) who will perform best as individuals, and as a crew.
While composing crews for missions on the International Space Station has become
routine, your team just received a special assignment from the Mission Planning
Directorate that is anything but routine.
The Mars mission has been fast-tracked. Instead of the 2033 launch date that NASA
had been aiming for, the launch date is being moved up to 2020. Intensive crew
training for this unprecedented and bold journey must begin immediately. A 5-
member crew will embark on this 3 year journey to Mars. This a truly
international endeavor: the Russian, European, Indian, and Chinese Space Agencies
will each select one astronaut for the mission. NASA will select the 5th crew
member - an American astronaut - to join this international crew.
The crew members from the other countries have backgrounds in engineering,
botany, geology and medicine. The crew still needs a flight engineer.
Your team was able to quickly narrow the field down to three veteran astronauts:
Anders, Bean, and Collins. Each of these three have experience as a flight
engineer on the ISS, have the necessary basic technical skills for the flight
engineer position, meet basic requirements for the mission including height, Body
Mass Index, and have passed a mental health screening. Now that you know the
other potential crew members it is necessary for your team to determine which of
the crew members will be the greatest asset on this particular mission.
The 3-year mission to Mars will be demanding. The success of the Mars mission
will require the 5-person crew to live and work together in extremely close
quarters, while isolated from the usual sources of social support - friends and
family - for extended periods of time. It will require that the crew can adapt to
the unknown and be able to handle whatever challenges will come their way. Once
they arrive at Mars, the crew will have to complete complex feats such as landing
and extravehicular activities. Because of a significant communication delay with
Earth of up to 22 minutes each way, the crew will have to work autonomously.
Additionally, the crew of the Mars mission consists of individuals representing
multiple nationalities, making strong interpersonal skills all the more critical
to the success of the mission.
The international space consortium has agreed to announce the names of the crew
members from each of the 5 countries in just 10 days from now. Your team must
deliver a recommendation for whom should be the American astronaut; years of hard
work (and millions of dollars) rest on this decision.
To guide your difficult decision, you have compiled information from the
personnel file for each candidate. The personnel files for each candidate are
detailed and include information such as their performance scores and ranking on
various training tasks, biographical information, evaluations and questionnaires
completed by  supervisors and individuals with whom they worked on prior
trainings and missions, and questionnaires completed by the candidate and his
family. A few human resource assistant have been working through the personnel
files and have prepared briefs for you. Your team must now come together and
discuss who should be selected for the mission.
Remember, the success of the mission to Mars depends on your team. Choose
quickly, but wisely!
"""
astronaut_candidate = Conversation(cog_agents, discussion_task, rounds,
decision_options, decision_rule, knowledge_bank)

astronaut_candidate.run_convo()
```

```
with open("agent_states.txt", "w") as states_file:
    for agent in astronaut_candidate.cog_agents:
        states_file.write(f"----- {agent.cog_agent.name} States -----\n")
        for element in agent.cog_agent.state:
            states_file.write(f"{element}\n")
        states_file.write("\n")  # Add a newline after each agent's states
```

Report.py

```
from CogAgent import CogAgent
from CogChain import CogChain

class Report:
    def __init__(self, cog_agents, knowledge_bank):
        self.initial_preferences = []
        self.turn_preferences = {}
        self.final_preferences = []
        self.turn_info_profiles = {}
        self.initial_info_profiles = []
        self.final_info_profiles = []
        self.speaker_frequency = []
        self.revealed_profile = []
        self.non_initial_contributions_percentages = []
        self.cog_agents = cog_agents
        self.knowledge_bank = knowledge_bank

    def retrieve_preferences(self, cog_agents):
        preferences = [f"{agent.cog_agent.name}: {agent.cog_agent.preference}"
for agent in cog_agents]
        return preferences

    def aggregate_round_profile(self, revealed_profile):
        aggregated_profile = {}
        all_shared_items = set()  # To keep track of all items shared in the
rounds
        item_frequencies = {}  # To track the frequency of each item

        for entry in revealed_profile:
            round_num = entry["round"]
            item = entry["item"]
            agent = entry["agent"]  # Include the contributing agent name

            if round_num not in aggregated_profile:
                aggregated_profile[round_num] = [(item, agent)]
            else:
                aggregated_profile[round_num].append((item, agent))

            all_shared_items.add(item)  # Add the item to the set

            # Update the frequency of the item
            item_frequencies[item] = item_frequencies.get(item, 0) + 1
        # Prepare a string to be used in the report
        result_str = "\n----- Turn Group Revealed Profile -----\n"
        for round_num, items_agents in sorted(aggregated_profile.items()):
            result_str += f"Round {round_num}:\n"
            for item, agent in items_agents:
                result_str += f"Info Item: {item}\nContributing Agent: {agent}\n"
            result_str += "\n"
        result_str += "\n----- Final Group Revealed Profile -----\n"
        # Include a list of all items shared in the rounds with frequency
        for item in all_shared_items:
            result_str += f"{item} (Frequency: {item_frequencies[item]})\n"
        return result_str

    def retrieve_agent_frequency(self, cog_agents):
        speaker_frequency = [f"{agent.cog_agent.name}:
{agent.cog_agent.frequency}" for agent in cog_agents]
        return speaker_frequency

    def generate_analysis_report(self, discussion_result, roundHolder):
        with open("analysis_report.txt", "w") as analysis_file:
            analysis_file.write("********************************************\n"
)
            analysis_file.write("ANALYSIS REPORT\n")
            analysis_file.write("********************************************\n"
)

            # 1. Calculate percentage of revealed profile items that are in the
knowledge bank
            all_shared_items = self.extract_all_shared_items()
            revealed_profile_percentage =
self.calculate_revealed_profile_percentage(all_shared_items)
            analysis_file.write("REVEALED PROFILE:\n")
            analysis_file.write(f"{revealed_profile_percentage:.2f}%\n\n")

            # 2. Calculate percentage of preference changes
            preference_change_percentage = self.calculate_preference_changes()
            analysis_file.write("PREFERENCE CHANGES:\n")
            analysis_file.write(f"{preference_change_percentage:.2f}%\n\n")

            # 3. Speaker distribution
            speaker_distribution = self.calculate_speaker_distribution()
            analysis_file.write("SPEAKER DISTRIBUTION:\n")
            for agent, percentage in speaker_distribution.items():
                analysis_file.write(f"{agent}: {percentage:.2f}%\n")
            analysis_file.write("\n")

            # 4. Decision made and group optimality rating
            if discussion_result != 'NO CONSENSUS':
                # decision_made, group_optimality =
self.calculate_decision_optimality(all_shared_items)
                group_optimality_revealed_profile =
self.calculate_revealed_profile_optimality(all_shared_items, discussion_result)
```

```
                group_optimality_knowledge_bank =
self.calculate_knowledge_bank_optimality(discussion_result)
                analysis_file.write("DECISION MADE:\n")
                analysis_file.write(f"Decision: {discussion_result}\n")
                analysis_file.write(f"Rounds Taken: {roundHolder}\n")
                analysis_file.write(f"Group Optimality (Revealed Profile):
{group_optimality_revealed_profile}\n")
                analysis_file.write(f"Group Optimality (Knowledge Bank):
{group_optimality_knowledge_bank}\n")

                # Corrected individual optimality ratings

            else:
                analysis_file.write("No group decision made.\n")
                analysis_file.write("\nIndividual Optimality:\n")
                for agent_pref in self.final_preferences:
                    agent_name, preference = agent_pref.split(": ")
                    revealed_profile_optimality =
self.calculate_revealed_profile_optimality(all_shared_items, preference)
                    knowledge_bank_optimality =
self.calculate_knowledge_bank_optimality(preference)
                    analysis_file.write(f"{agent_name}:\n")
                    analysis_file.write(f"Revealed Profile:
{revealed_profile_optimality}\n")
                    analysis_file.write(f"Knowledge Bank:
{knowledge_bank_optimality}\n")

    def extract_all_shared_items(self):
        all_shared_items = set()
        for entry in self.revealed_profile:
            label = entry['item']
            for item in self.knowledge_bank:
                if item['label'] == label:
                    all_shared_items.add(item['label'])
                    break  # Stop searching once the item is found
        return all_shared_items

    def calculate_revealed_profile_percentage(self, all_shared_items):
    # Calculate the percentage of unique items in the revealed profile also in
the knowledge bank
        unique_revealed_items = set(entry['item'] for entry in
self.revealed_profile)
        total_revealed_items = len(unique_revealed_items)

        if total_revealed_items == 0:
            return 0.0

        kb_items = [entry['label'] for entry in self.knowledge_bank]
        kb_length = len(kb_items)

        percentage = (total_revealed_items / kb_length) * 100
        return percentage
    def calculate_preference_changes(self):
        initial_preferences = [agent.split(": ")[1] for agent in
self.initial_preferences]
        final_preferences = [agent.split(": ")[1] for agent in
self.final_preferences]
        changed_agents = [initial != final for initial, final in
zip(initial_preferences, final_preferences)]
        return (sum(changed_agents) / len(self.cog_agents)) * 100

    def calculate_speaker_distribution(self):
        total_turns = sum([int(freq.split(": ")[1]) for freq in
self.speaker_frequency])
        speaker_distribution = {freq.split(": ")[0]: (int(freq.split(": ")[1]) /
total_turns) * 100 for freq in self.speaker_frequency}
        return speaker_distribution

    def calculate_revealed_profile_optimality(self, all_shared_items,
agent_preference):
        ####FIX
        preference_counts = {'C': 0, 'A': 0, 'B': 0}
        for item in all_shared_items:
        # Check if the second item letter is 'G'
            if item[1] == 'G':
                # Determine the preference of the first item and update
preference counts
                if item[0] == 'C':
                    preference_counts['C'] += 1
                elif item[0] == 'A':
                    preference_counts['A'] += 1
                elif item[0] == 'B':
                    preference_counts['B'] += 1
        preference_ranking = {'C': 0, 'A': 0, 'B': 0}
        for preference, count in preference_counts.items():
            preference_ranking[preference] = count
        sorted_preferences = sorted(preference_ranking.items(), key=lambda x:
x[1], reverse=True)
        max_count = sorted_preferences[0][1]
        max_count_preferences = sum(1 for pref, count in sorted_preferences if
count == max_count)
        ranks = {}
        if max_count_preferences == 1:  # Only one preference has the maximum
count
            for preference, count in sorted_preferences:
                ranks[preference] = 'BEST' if count == max_count else 'WORST'
        else:  # Multiple preferences have the maximum count
            for preference, count in sorted_preferences:
                ranks[preference] = 'AVERAGE' if count == max_count else 'WORST'

            # Calculate second highest count
        second_highest_count = sorted_preferences[1][1]
            # Calculate how many preferences have the second highest count
        second_highest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == second_highest_count)

            # Assign 'AVERAGE' rank to preferences with the second highest count
        if second_highest_count_preferences > 1:
            for preference, count in sorted_preferences:
                if count == second_highest_count:
                    ranks[preference] = 'AVERAGE'

            # Calculate lowest count
        lowest_count = sorted_preferences[-1][1]

            # If lowest count is different from second highest count, assign
'WORST' rank
        if lowest_count != second_highest_count:
            lowest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == lowest_count)
            if lowest_count_preferences > 1:
                for preference, count in sorted_preferences:
                    if count == lowest_count:
                        ranks[preference] = 'WORST'

            # Extract agent preference label
        agent_preference_label = agent_preference.split()[1][0]

            # Assign the agent's preference ranking
        agent_preference_ranking = ranks.get(agent_preference_label, None)
        return agent_preference_ranking

    def calculate_knowledge_bank_optimality(self, preference):
        # Define the fixed knowledge bank optimality ratings
        knowledge_bank_ratings = {'Samuel Anders': 'BEST', 'John Bean':
'AVERAGE', 'Scott Collins': 'WORST'}
        return knowledge_bank_ratings.get(preference, 'UNKNOWN')

    def generate_preference_report(self, discussion_result, rounds,
decision_rule, roundHolder):
        with open("report.txt", "w") as report_file:
            report_file.write("*********************************************\n")
            report_file.write("DISCUSSION REPORT\n")
            report_file.write("*********************************************\n")
            report_file.write(f"Number of Rounds: {rounds}\n")
            report_file.write(f"Decision Rule: {decision_rule}\n")
            report_file.write
            if discussion_result == 'NO CONSENSUS':
                report_file.write(f"Discussion Result: Failure to reach consensus
in the specified number of rounds.\n")
            else:
                report_file.write(f"Discussion Result: Consensus reached in
{roundHolder} rounds!\n")
                report_file.write(f"Agreed upon goal: {discussion_result}\n")
            report_file.write("\n*********************************************\n"
)
            report_file.write("PREFERENCE INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("----- Initial Preference Distribution -----\n")
            for i, entry in enumerate(self.initial_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("----- Turn Preference Distribution -----\n")
            for round_num, preferences in self.turn_preferences.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(preferences, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Preference Distribution -----\n")
            for i, entry in enumerate(self.final_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("INFO ITEM INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("\n----- Initial Info Profiles -----\n")
            for i, entry in enumerate(self.initial_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n----- Turn Info Profiles -----\n")
            for round_num, info_profile in self.turn_info_profiles.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(info_profile, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Info Profiles -----\n")
            for i, entry in enumerate(self.final_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            aggregated_profile_str =
self.aggregate_round_profile(self.revealed_profile)
            report_file.write(f"{aggregated_profile_str}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("CONVO INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("-----Speaker Frequency -----\n")
            for i, entry in enumerate(self.speaker_frequency, start=1):
                report_file.write(f"{entry}\n")
            report_file.write("\n----- Non-Initial Contributions-----\n")
            for agent_name, percentage in
self.non_initial_contributions_percentages.items():
```

```
                report_file.write(f"{agent_name}: {percentage}%\n")

    def kb_map(self, knowledge_base, knowledge_bank):
        labels = []
        for kb_entry in knowledge_base:
            kb_item = kb_entry["item"]
            for bank_entry in knowledge_bank:
                bank_item = bank_entry["item"]
                if kb_item == bank_item:
                    labels.append(bank_entry["label"])
                    break  # Stop searching once a match is found
        return labels

    def retrieve_info_profiles(self, cog_agents, knowledge_bank):
        profiles = []
        for agent in cog_agents:
            knowledge_base = agent.cog_agent.knowledge_base
            profile_labels = self.kb_map(knowledge_base, knowledge_bank)
            profiles += [f"{agent.cog_agent.name}: {profile_labels}"]
        return profiles

    def calculate_non_initial_contributions_percentage(self, agent_name,
revealed_profile):
        agent_contributions = [entry for entry in revealed_profile if
entry['agent'] == agent_name]
        total_contributions = len(agent_contributions)

        if total_contributions == 0:
            return None  # Return None for agents who did not contribute

        non_initial_contributions = sum(
            1 for entry in agent_contributions if not
self.is_item_in_initial_kb(agent_name, entry['item']))

        if total_contributions == non_initial_contributions:
            return 0.0  # All contributions were non-initial

        percentage = (non_initial_contributions / (total_contributions -
non_initial_contributions)) * 100
        return round(percentage, 2)

    def is_item_in_initial_kb(self, agent_name, item):
        for agent in self.cog_agents:
            if agent.cog_agent.name == agent_name:
                return agent.cog_agent.was_item_in_initial_kb(item)
        return False
```

APPENDIX G: TRUSTCOG CODE

CogAgent.py

```
class CogAgent:
    def __init__(self, name, knowledge_base, trust_tendency):
        self.name = name
        self.knowledge_base = knowledge_base
        self.initial_knowledge_base = knowledge_base.copy()
        self.preference = ""
        self.state = []
        self.frequency = 0
        self.trust_tendency = trust_tendency

    def was_item_in_initial_kb(self, item):
        return any(entry['item'] == item for entry in
self.initial_knowledge_base)

    def state_snap(self):
        snapshot = ""
        snapshot += f"Preference: {self.preference}\n"
        return snapshot

    def update_agent_state(self, state):
        self.state.append(state)
```

CogChain.py

```
import os
import re
os.environ["OPENAI_API_KEY"]='sk-
iRdjQCppyWt3WblPYd24T3BlbkFJYsTrZOxSnVoajtnSoI5U'
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
llm = ChatOpenAI(model="gpt-4-0125-preview")
import random
import sys
from CogAgent import CogAgent

class CogChain:
    def __init__(self, cog_agent):
        self.cog_agent = cog_agent
        self.convo_history = []
        self.name = cog_agent.name
    def kb_print(self, kb):
        kb_str = "KNOWLEDGE BASE:\n"
        for item in kb:
            item_str = f"{item['item']}\n\n"
            kb_str += item_str
        return kb_str

    def preference_chain(self, task, kb, decision_options):
        kb_string = self.kb_print(kb)
        update_goal = PromptTemplate(
            input_variables = ["task","updated_kb", "decision_options"],
            template = """{task}. You have access to a list of facts, called a 
'knowledge base', to help guide your preference. Here is the most recently 
updated list:
            {updated_kb}
            Based on the above facts, choose one alternative from the decision 
options. You must choose from the provided options only.
            {decision_options}
            Use this format for your output: PREFERENCE: (Insert candidate name) 
\n EXPLANATION: (justify your decision, limit to 300 tokens).""")
        goal_update = LLMChain(llm=llm, prompt=update_goal)
        updated_goal = goal_update.invoke({
        "task": task,
        "updated_kb": kb_string,
        "decision_options": decision_options})
        goal_match = re.search(r'PREFERENCE: (.+)', updated_goal['text'])
        if goal_match:
            new_goal = goal_match.group(1)
            self.cog_agent.preference = new_goal
        return updated_goal["text"]

    def contribution_chain(self, task, context):
        kb = self.cog_agent.knowledge_base
        kb_string = self.kb_print(kb)
        generate_contribution = PromptTemplate(
            input_variables = ["goal", "task", "kb", "context"],
            template = """{task} Your team mate just contributed: {context}.
            It is your turn in the discussion. Choose a fact from the knowledge 
base and provide it (exactly as phrased) alongside your commentary. You are 
likely to choose a fact that aligns with your preference. You may also directly 
address the team member who just contributed, if you want to do so.
            Preference: {goal}
            {kb}
            Output format: INFORMATION ITEM: (insert fact exactly as given) \n 
COMMENTARY: (insert your commentary)""")
        gen_contribution = LLMChain(llm=llm, prompt=generate_contribution)
        contribution = gen_contribution.invoke({
            "goal": self.cog_agent.preference,
            "task": task,
            "kb" : kb_string,
            "context": context})
        return contribution['text']
    def process_message(self, task, context, decision_options):
        #Initial Snap
        initial_snap = self.cog_agent.state_snap()
        #Take in Context, Update KB
        #UPDATE KB ACCORDING TO TRUST TENDENCY.
        if self.cog_agent.trust_tendency == 'high':
            self.knowledge_base_chain(context)
        elif self.cog_agent.trust_tendency == 'low':
        # With a 50% chance, update knowledge base
            if random.random() < 0.5:
                self.knowledge_base_chain(context)
        #Update Goal
        goal_result = self.preference_chain(task, self.cog_agent.knowledge_base,
decision_options)
        #Final Snap
        final_snap = self.cog_agent.state_snap()
        #Generate process report
        process_report = f"---------PROCESS REPORT----------\n"
        process_report += f"Initial Agent State: " + initial_snap + "\n"
        process_report += f"***\n"
        process_report += f"Context: " + context + "\n"
        process_report += f"***\n"
        process_report += f"Final Agent State: " + final_snap + "\n"
        process_report += f""+ goal_result + "\n"
        self.cog_agent.update_agent_state(process_report)

    def knowledge_base_chain(self, context):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        # Find if the item is in the knowledge base
        item_found = False
        for item in self.cog_agent.knowledge_base:
            if item['item'] == info_item:
                item['frequency'] += 1
                item_found = True
                break

        # If the item is not found, add it to the knowledge base
        if not item_found:
            self.cog_agent.knowledge_base.append({"item": info_item, "frequency":
1, "initial": False})

        # Use kb_print to get the updated knowledge base string
        updated_kb_str = self.kb_print(self.cog_agent.knowledge_base)
        return updated_kb_str
    def first_turn_contribution(self, discussion_task):
        initial_snap = self.cog_agent.state_snap()
        goal = self.cog_agent.preference
        knowledge_base = self.cog_agent.knowledge_base
        kb_string = self.kb_print(knowledge_base)
        generate_first_turn_contribution = PromptTemplate(
            input_variables = ["goal", "kb", "task", "context"],
            template = """{task}. You are starting the discussion. Choose a fact
from the knowledge base and provide it (exactly as phrased) alongside your
commentary. You are likely to choose a fact that aligns with your preference.
            Preference: {goal}
            {kb}
            Output format: INFORMATION ITEM: (insert fact) \n COMMENTARY: (insert
your commentary)""")
        gen_first_turn_contribution = LLMChain(llm=llm,
prompt=generate_first_turn_contribution)
        contribution = gen_first_turn_contribution.invoke({
            "goal": goal,
            "kb": kb_string,
            "task": discussion_task,})
        updated_kb = self.knowledge_base_chain(contribution['text'])
        final_snap = self.cog_agent.state_snap()
        #Generate state report
        state_report = f"Contribution: " + contribution['text'] + "\n"
        state_report += f"----------------------------------------\n"
        state_report += f"Final Agent State: " + final_snap + "\n"
        self.cog_agent.update_agent_state(state_report)
        return contribution['text']

    def invoke_convo(self, sample_context, discussion_task):
        kb = self.cog_agent.knowledge_base
        goal = self.cog_agent.preference
        contribution_result =
self.contribution_chain(discussion_task,sample_context)
        updated_kb = self.knowledge_base_chain(contribution_result)
        ###GENERATE TURN REPORT
        final_snap = self.cog_agent.state_snap()
        self.cog_agent.update_agent_state(final_snap)
        return contribution_result
```

Conversation.py

```
import os
import re
import random
import sys
from Report import Report
from CogAgent import CogAgent
from CogChain import CogChain

class Conversation:
    def __init__(self, cog_agents, discussion_task, rounds, decision_options,
decision_rule, knowledge_bank):
        self.cog_agents = cog_agents
        self.discussion_task = discussion_task
        self.rounds = rounds
        self.convo_history = []
        self.decision_options = decision_options
        self.decision_rule = decision_rule
        self.knowledge_bank = knowledge_bank
        self.report = Report(cog_agents, knowledge_bank)
        self.revealed_profile = []

    def broadcast_message(self, cog_agents, context, agent_in):
        for agent in cog_agents:
            if agent.name != agent_in:
                agent.process_message(self.discussion_task, context,
self.decision_options)

    def update_revealed_profile(self, context, current_agent, round):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        rp = [{"item": info_item,
            "agent": current_agent.name,
            "round": round}]

        info_item_label = self.report.kb_map(rp, self.knowledge_bank)
        # Add the info_item to the revealed_profile
        self.revealed_profile.append({
            "item": info_item_label[0],
            "agent": current_agent.name,
            "round": round})
        return self.revealed_profile

    def consensus(self):
        # Check for consensus based on the decision_rule
        if self.decision_rule == 'unanimous':
            # Check if all agents have the same goal
            goals = {agent.cog_agent.preference for agent in self.cog_agents}
            if len(goals) == 1:
                return goals.pop()  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        elif self.decision_rule == 'majority':
            # Check if there is a majority agreement on the goal
            goal_counts = {agent.cog_agent.preference: 0 for agent in
self.cog_agents}
            for agent in self.cog_agents:
                goal_counts[agent.cog_agent.preference] += 1

            max_goal = max(goal_counts, key=goal_counts.get)

            if goal_counts[max_goal] > len(self.cog_agents) / 2:
                return max_goal  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        else:
            return 'NO CONSENSUS'

    def next_speaker(self, current_agent, broadcast_agents):
        next_speakers = [agent for agent in broadcast_agents]
        if next_speakers:
            # If more than 2+ agents, choose randomly
            next_speaker = random.choice(next_speakers)
        return next_speaker

    def run_convo(self):
        #initialize agent preferences
        for agent in self.cog_agents:
            # Call preference_chain for each agent
            preference = agent.preference_chain(
                self.discussion_task, agent.cog_agent.knowledge_base,
self.decision_options)
            agent.cog_agent.update_agent_state(preference)

        # Store Initial Profiles
        self.report.initial_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        # Store Initial Preferences
        self.report.initial_preferences =
self.report.retrieve_preferences(self.cog_agents)

        # Initial Turn Contribution
        agent1 = random.choice(self.cog_agents)
        contribution1 = agent1.first_turn_contribution(self.discussion_task)
        agent1.cog_agent.frequency += 1

        # Add first contribution to chat history, ADD ITEM TO REVEALED_PROFILE
        self.convo_history.append(contribution1)
        self.update_revealed_profile(contribution1, agent1, 0)

        # Broadcast first contribution to other agents
        broadcast_agents = [agent for agent in self.cog_agents if agent !=
agent1]
        self.broadcast_message(broadcast_agents, contribution1, agent1.name)

        # Store Turn Preference
        self.report.turn_preferences[0] =
self.report.retrieve_preferences(self.cog_agents)
```

```
        # Store Turn Info Profiles
        self.report.turn_info_profiles[0] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        print(agent1.name)
        print(contribution1)
        current_agent = self.next_speaker(agent1, broadcast_agents)

        for round_num in range(1, self.rounds):
            context = self.convo_history[-1]
            contribution = current_agent.invoke_convo(context,
self.discussion_task)
            current_agent.cog_agent.frequency += 1
            print(current_agent.name)
            print(contribution)
            self.convo_history.append(contribution)
            #ADD ITEM TO REVEALED PROFILE
            self.update_revealed_profile(contribution, current_agent, round_num)
            # Broadcast to other agents
            broadcast_agents = [agent for agent in self.cog_agents if agent !=
current_agent]
            self.broadcast_message(broadcast_agents, contribution,
current_agent.name)
            # Store Turn Preference
            self.report.turn_preferences[round_num] =
self.report.retrieve_preferences(self.cog_agents)
             # Store Turn Info Profiles
            self.report.turn_info_profiles[round_num] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
            decision_result = self.consensus()
            if decision_result != 'NO CONSENSUS':
                print(f"Consensus reached! Agreed upon goal: {decision_result}")
                break  # Exit the loop if consensus is reached
            #IF agents do not agree, select next speaker
            current_agent = self.next_speaker(current_agent, broadcast_agents)
        decision_result = self.consensus()
        # Store Final Preferences
        self.report.final_preferences =
self.report.retrieve_preferences(self.cog_agents)
        # Store Final Info Profiles
        self.report.final_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
        # Store Agent Speaker Frequencies
        self.report.speaker_frequency =
self.report.retrieve_agent_frequency(self.cog_agents)
        self.report.revealed_profile = self.revealed_profile
        # Calc & Store non-initial contribution %
        non_initial_percentages = {}
        for agent in self.cog_agents:
            agent_name = agent.cog_agent.name
            non_initial_percentage =
self.report.calculate_non_initial_contributions_percentage(agent_name,
self.revealed_profile)

            if non_initial_percentage is not None:
                non_initial_percentages[agent_name] = non_initial_percentage
        self.report.non_initial_contributions_percentages =
non_initial_percentages
        # Generate Report
        self.report.generate_preference_report(decision_result, self.rounds,
self.decision_rule)
        self.report.generate_analysis_report(decision_result)
        with open("conversation_log.txt", "w") as log_file:
            log_file.write("----- Conversation Thread -----\n")
            for i, entry in enumerate(self.convo_history, start=1):
                log_file.write(f"Turn {i}:\n{entry}\n")
```

Discuss.py

```
from Conversation import Conversation
from CogAgent import CogAgent
from Report import Report
from CogChain import CogChain

####INFO PROFILE####
# Anders
AG1 = """Astronaut Anders received the highest possible scores on the military
survival standards, an assessment that all astronauts must complete in order to
qualify for space missions. Anders' scores were significantly higher than either
candidate’s scores."""
AG2 = """During Ander’s last ICE (Isolated and Confined in Extreme environments)
training sequence, a fellow crew member praised him for being able to recognize
others’ personal needs. In particular, he was recognized for facilitating
personal time and bringing the group together for light-hearted fun."""
AG3 = """While working at NASA, Anders coordinated and completed a number of
science projects. This expertise is important to Flight Engineers and one of the
reasons Anders was chosen for the previous space mission. Anders did not
disappoint. His ability to set up and coordinate science projects allowed his
crew to finish more than twice the number of projects than anticipated."""
AG4 = """Anders was one of 8 hikers to successfully summit Mount Everest in 2005.
Interviews with his comrades suggest Anders was the “glue” that held the group
together when things went wrong."""
AG5 = """On previous missions, where lack of sleep was a frequent occurrence,
Anders was largely unaffected. Despite the sub-optimal sleeping patterns, Anders'
mental and physical health did not show signs of suffering."""
AG6 = """Data collected during training on simulated emergencies shows that
Anders is able to remain calm during stressful and unexpected situations. He
performs well under pressure."""
AB1 = """Anders suffered a broken wrist 7 months ago during a training
simulation. The injury has healed, but will likely require extra exercise to
```

```
prevent forearm muscle atrophy in zero gravity. This may take time away from him
performing other team duties."""
AB2 = """Most of Anders' space training experience has been limited to
simulations on Earth, with only one mission to the International Space Station.
Furthermore, this mission to Mars would be even longer than his only long-
distance flight experience--it is not clear if he will adapt well for a long-
duration mission."""
AB3 = """Anders comes from a close-knit family. In diaries collected during
training, he reported feeling sad and frustrated when he was unable to
communicate with family members on a regular basis. This could be a serious
concern for his mental health on a mission to Mars because there will be very
little contact with Earth, and the little contact there is will be through
mission control."""
AB4 = """Anders easily gets annoyed when explaining concepts to others and
becomes very short with them as a result."""
AN1 = """Anders enjoys science fiction literature and has self-published a novel
about the start of a human colony in a distant galaxy. He hopes to write a sequel
when he retires."""
AN2 = """As a child, Anders enjoyed attending Houstonli-area sporting events with
his parents. He was particularly fond of attending the Texans football games."""
AN3 = """Anders has a pet cat named Buzz, after Buzz Aldrin. Anders met Buzz as a
child around when he decided he wanted to be an astronaut."""
AN4 = """Anders used to own a Brown Labrador Retriever named Bailey. When Anders
participated in astronaut training and missions, Bailey used to stay with his
family."""

# Bean
BG1 = """On Expedition 31/32, where he helped refuel the International Space
Station (ISS), Bean encountered a breakdown in the communication system between
his vehicle and the ISS before arrival to the ISS. Bean was credited for leading
the extravehicular activity that identified and fixed the issue, ultimately
making the refuel possible."""
BG2 = """While it is widely known that Bean holds a bachelor’s degree in
Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a
minor in Counseling Psychology. Crewmembers have reported that Bean is a great
listener and often knows exactly what to say to make others feel more
comfortable."""
BG3 = """Survey responses from Bean’s previous crew mates reveal that he has a
universal sense of humor. People always understand and enjoy Bean’s jokes, so
there is little risk of his humor being misinterpreted as being negative. Crew
members report that this really helps to de-escalate tense situations."""
BB1 = """Bean had a difficult time getting along with some of his co-workers at
Ames. He had asked his supervisor to intervene on several occasions as they were
not able to resolve the disagreements on their own."""
BB2 = """Bean does not always plan his work well, and sometimes his pace of work
is too slow. This has become problematic not only in his mission to the ISS, but
also in training simulations. Often times, the maintenance work and data
collection the astronauts engage in on these missions requires high levels of
interdependency - this can be jeopardized if one person works at a slower pace
than the rest of the crew members."""
BB3 = """Bean has a 2-year-old daughter, and colleagues have noticed he is
frustrated when he has to work late and cannot be home with her. There is worry
```

```
that being away for so long on this mission will exponentially increase Bean's
frustration."""
BN1 = """Bean enjoys listening to the Beatles while performing work on his
own."""
BN2 = """Bean is an avid fan of Shakespeare and often quotes his work."""
BN3 = """In college, Bean was on the squash team, where he met his best friend
Jack. Jack now lives in Providence, Rhode Island, but they still talk on a weekly
basis."""
BN4 = """Astronaut Bean enjoys traveling and his favorite place to vacation to is
in the south of France."""
BN5 = """Bean has five siblings: one older brother, one older sister, and three
younger sisters. He is closest to his younger sister, Rebecca, who is just one
year younger than he is."""
BN6 = """If chosen for this mission, Bean would be the shortest crew member at 5-
feet, 5-inches tall."""

# Collins
CG1 = """Collins is accustomed to being isolated for long periods of time. He
successfully completed two treks to the Antarctic in which he lived in tents and
a small building for the Antarctic winter-over period. Living in small spaces did
not bother him."""
CG2 = """Collins has a passion for languages. He is fluent in English, Spanish,
and French and proficient in Russian. Recently he started a class on Mandarin and
already he is nearly proficient. This ability will help him interact and bond
with international crew members."""
CG3 = """Collins has participated in spacewalks to repair damaged solar panels
and ammonia pump module. His peers have ranked him as the most competent crew
member during their spacewalks."""
CG4 = """Collins received the “Silver Snoopy” award for his contributions on
NASA’s STS-135 mission. This award is considered to be the highest honor of all
of the awards, given to the person who was the most outstanding team member,
creating a team environment focused on flight safety and overall team success."""
CB1 = """Surveys showed that Collins has snapped at coworkers on multiple
occasions. In one situation, a crewmember even went so far as to say would not
want to work with Collins on future missions."""
CB2 = """Collins has the least space-related training of all three potential
candidates."""
CB3 = """Recent blood work indicated that Collins may be developing above-average
blood pressure. Doctors advised another assessment before making any conclusive
determinations. However, if Collins does have heightened blood pressure, he would
have to take extra time to rest while on the mission. This would detract from his
time working on experiments, as well as his ability to do physically stressful
activities."""
CB4 = """Three weeks ago, Collins broke his ankle. As a result, he has not been
training as much and the current state of his cardiovascular performance is
behind compared to the other two candidates."""
CB5 = """In one incident during a mock extravehicular activity scenario, Collins
refused to follow an instruction from mission control. Fellow crew members were
unsure of why he did this since the request from mission control seemed
straightforward and helpful. Crewmembers reported being conflicted with how to
respond, which slowed their progress on the extravehicular activity scenario.
```

```
Only half of the tasks needed were completed on the simulated extravehicular
activity."""
CB6 = """An internal incident report indicates that Collins made offensive jokes
to a Chinese astronaut on a previous analog space mission that made the Chinese
astronaut and other crew members uncomfortable."""
CN1 = """Collins considers himself a coffee aficionado and his favorite brand of
coffee is Intelligentsia."""
CN2 = """Collins' childhood friend completed NASA's astronaut candidate training
program but was never selected for a space mission."""
CN3 = """Collins enjoys completing Sudoku puzzles to relax"""
CN4 = """Collins has a fraternal twin who works in finance in Houston, Texas."""

knowledge_bank = [
    {"label": "AG1", "item": """Astronaut Anders received the highest possible
scores on the military survival standards, an assessment that all astronauts must
complete in order to qualify for space missions. Anders' scores were
significantly higher than either candidate's scores."""},
    {"label": "AG2", "item": """During Ander's last ICE (Isolated and Confined in
Extreme environments) training sequence, a fellow crew member praised him for
being able to recognize others' personal needs. In particular, he was recognized
for facilitating personal time and bringing the group together for light-hearted
fun."""},
    {"label": "AG3", "item": """While working at NASA, Anders coordinated and
completed a number of science projects. This expertise is important to Flight
Engineers and one of the reasons Anders was chosen for the previous space
mission. Anders did not disappoint. His ability to set up and coordinate science
projects allowed his crew to finish more than twice the number of projects than
anticipated."""},
    {"label": "AG4", "item": """Anders was one of 8 hikers to successfully summit
Mount Everest in 2005. Interviews with his comrades suggest Anders was the "glue"
that held the group together when things went wrong."""},
    {"label": "AG5", "item": """On previous missions, where lack of sleep was a
frequent occurrence, Anders was largely unaffected. Despite the sub-optimal
sleeping patterns, Anders' mental and physical health did not show signs of
suffering."""},
    {"label": "AG6", "item": """Data collected during training on simulated
emergencies shows that Anders is able to remain calm during stressful and
unexpected situations. He performs well under pressure."""},
    {"label": "AB1", "item": """Anders suffered a broken wrist 7 months ago
during a training simulation. The injury has healed, but will likely require
extra exercise to prevent forearm muscle atrophy in zero gravity. This may take
time away from him performing other team duties."""},
    {"label": "AB2", "item": """Most of Anders' space training experience has
been limited to simulations on Earth, with only one mission to the International
Space Station. Furthermore, this mission to Mars would be even longer than his
only long-distance flight experience--it is not clear if he will adapt well for a
long-duration mission."""},
    {"label": "AB3", "item": """Anders comes from a close-knit family. In diaries
collected during training, he reported feeling sad and frustrated when he was
unable to communicate with family members on a regular basis. This could be a
serious concern for his mental health on a mission to Mars because there will be
very little contact with Earth, and the little contact there is will be through
mission control."""},
    {"label": "AB4", "item": """Anders easily gets annoyed when explaining
concepts to others and becomes very short with them as a result."""},
    {"label": "AN1", "item": """Anders enjoys science fiction literature and has
self-published a novel about the start of a human colony in a distant galaxy. He
hopes to write a sequel when he retires."""},
    {"label": "AN2", "item": """As a child, Anders enjoyed attending Houstonli-
area sporting events with his parents. He was particularly fond of attending the
Texans football games."""},
    {"label": "AN3", "item": """Anders has a pet cat named Buzz, after Buzz
Aldrin. Anders met Buzz as a child around when he decided he wanted to be an
astronaut."""},
    {"label": "AN4", "item": """Anders used to own a Brown Labrador Retriever
named Bailey. When Anders participated in astronaut training and missions, Bailey
used to stay with his family."""},
    {"label": "BG1", "item": """On Expedition 31/32, where he helped refuel the
International Space Station (ISS), Bean encountered a breakdown in the
communication system between his vehicle and the ISS before arrival to the ISS.
Bean was credited for leading the extravehicular activity that identified and
fixed the issue, ultimately making the refuel possible."""},
    {"label": "BG2", "item": """While it is widely known that Bean holds a
bachelor’s degree in Aerospace Engineering and Cross-Cultural studies, few know
that he also pursued a minor in Counseling Psychology. Crewmembers have reported
that Bean is a great listener and often knows exactly what to say to make others
feel more comfortable."""},
    {"label": "BG3", "item": """Survey responses from Bean’s previous crew mates
reveal that he has a universal sense of humor. People always understand and enjoy
Bean’s jokes, so there is little risk of his humor being misinterpreted as being
negative. Crew members report that this really helps to de-escalate tense
situations."""},
    {"label": "BB1", "item": """Bean had a difficult time getting along with some
of his co-workers at Ames. He had asked his supervisor to intervene on several
occasions as they were not able to resolve the disagreements on their own."""},
    {"label": "BB2", "item": """Bean does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "BB3", "item": """Bean has a 2-year-old daughter, and colleagues
have noticed he is frustrated when he has to work late and cannot be home with
her. There is worry that being away for so long on this mission will
exponentially increase Bean's frustration."""},
    {"label": "BN1", "item": """Bean enjoys listening to the Beatles while
performing work on his own."""},
    {"label": "BN2", "item": """Bean is an avid fan of Shakespeare and often
quotes his work."""},
    {"label": "BN3", "item": """In college, Bean was on the squash team, where he
met his best friend Jack. Jack now lives in Providence, Rhode Island, but they
still talk on a weekly basis."""},
    {"label": "BN4", "item": """Astronaut Bean enjoys traveling and his favorite
place to vacation to is in the south of France."""},
    {"label": "BN5", "item": """Bean has five siblings: one older brother, one
older sister, and three younger sisters. He is closest to his younger sister,
Rebecca, who is just one year younger than he is."""},
    {"label": "BN6", "item": """If chosen for this mission, Bean would be the
shortest crew member at 5-feet, 5-inches tall."""},
    {"label": "CG1", "item": """Collins is accustomed to being isolated for long
periods of time. He successfully completed two treks to the Antarctic in which he
lived in tents and a small building for the Antarctic winter-over period. Living
in small spaces did not bother him."""},
    {"label": "CG2", "item": """Collins has a passion for languages. He is fluent
in English, Spanish, and French and proficient in Russian. Recently he started a
class on Mandarin and already he is nearly proficient. This ability will help him
interact and bond with international crew members."""},
    {"label": "CG3", "item": """Collins has participated in spacewalks to repair
damaged solar panels and ammonia pump module. His peers have ranked him as the
most competent crew member during their spacewalks."""},
    {"label": "CG4", "item": """Collins received the “Silver Snoopy” award for
his contributions on NASA’s STS-135 mission. This award is considered to be the
highest honor of all of the awards, given to the person who was the most
outstanding team member, creating a team environment focused on flight safety and
overall team success."""},
    {"label": "CB1", "item": """Surveys showed that Collins has snapped at
coworkers on multiple occasions. In one situation, a crewmember even went so far
as to say would not want to work with Collins on future missions."""},
    {"label": "CB2", "item": """Collins does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "CB3", "item": """Recent blood work indicated that Collins may be
developing above-average blood pressure. Doctors advised another assessment
before making any conclusive determinations. However, if Collins does have
heightened blood pressure, he would have to take extra time to rest while on the
mission. This would detract from his time working on experiments, as well as his
ability to do physically stressful activities."""},
    {"label": "CB4", "item": """Three weeks ago, Collins broke his ankle. As a
result, he has not been training as much and the current state of his
cardiovascular performance is behind compared to the other two candidates."""},
    {"label": "CB5", "item": """In one incident during a mock extravehicular
activity scenario, Collins refused to follow an instruction from mission control.
Fellow crew members were unsure of why he did this since the request from mission
control seemed straightforward and helpful. Crewmembers reported being conflicted
with how to respond, which slowed their progress on the extravehicular activity
scenario. Only half of the tasks needed were completed on the simulated
extravehicular activity."""},
    {"label": "CB6", "item": """An internal incident report indicates that
Collins made offensive jokes to a Chinese astronaut on a previous analog space
mission that made the Chinese astronaut and other crew members
uncomfortable."""},
    {"label": "CN1", "item": """Collins considers himself a coffee aficionado and
his favorite brand of coffee is Intelligentsia."""},
    {"label": "CN2", "item": """Collins’ childhood friend completed NASA’s
astronaut candidate training program but was never selected for a space
mission."""},
    {"label": "CN3", "item": """Collins enjoys completing Sudoku puzzles to
relax"""},
    {"label": "CN4", "item": """Collins has a fraternal twin who works in finance
in Houston, Texas."""}
]

#####AGENT 1: CMR######
cmr_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG6, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True}
]
cmr = CogAgent("CMR", cmr_KB, "high")
cog_cmr = CogChain(cmr)

#####AGENT 2: MS1#####
ms1_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CB3, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CB5, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": CN4, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN4, "frequency" : 0 ,"initial": True},
    {"item": AG1, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True}
]
ms1 = CogAgent("MS1", ms1_KB, "high")
cog_ms1 = CogChain(ms1)

#####AGENT 3: MS2#####
ms2_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
ms2 = CogAgent("MS2", ms2_KB, "low")
cog_ms2 = CogChain(ms2)

#####AGENT 4: FE######
fe_KB = [
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CB2, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN5, "frequency" : 0 ,"initial": True},
    {"item": BN6, "frequency" : 0 ,"initial": True},
    {"item": AG5, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
fe = CogAgent("FE", fe_KB, "low")
cog_fe = CogChain(fe)

cog_agents = [cog_cmr, cog_ms1, cog_ms2, cog_fe]
rounds = 10
decision_options = """Candidate 1: Samuel Anders
Candidate 2: John Bean
Candidate 3: Scott Collins"""
decision_rule = 'majority'
discussion_task = """ DISCUSSION TASK: You work as a member of NASA’s Astronaut
Crew Composition team. Your team analyzes all of the information obtained during
astronaut selection and training - including test scores, analog performance
measures, expert observations, peer evaluations, and medical  records. Armed with
this information, your team makes two kinds of evaluations. The first evaluation
is to determine which astronauts are ready for flight and which are not. Of those
who are deemed to be flight ready, your team composes entire crews for each
mission. In this second evaluation, you consider the complementarity and
compatibility of the astronauts as a team to select the 4, 5, or 6  astronauts
(depending on the mission) who will perform best as individuals, and as a crew.
While composing crews for missions on the International Space Station has become
routine, your team just received a special assignment from the Mission Planning
Directorate that is anything but routine.
The Mars mission has been fast-tracked. Instead of the 2033 launch date that NASA
had been aiming for, the launch date is being moved up to 2020. Intensive crew
training for this unprecedented and bold journey must begin immediately. A 5-
member crew will embark on this 3 year journey to Mars. This a truly
international endeavor: the Russian, European, Indian, and Chinese Space Agencies
will each select one astronaut for the mission. NASA will select the 5th crew
member - an American astronaut - to join this international crew.
The crew members from the other countries have backgrounds in engineering,
botany, geology and medicine. The crew still needs a flight engineer.
Your team was able to quickly narrow the field down to three veteran astronauts:
Anders, Bean, and Collins. Each of these three have experience as a flight
engineer on the ISS, have the necessary basic technical skills for the flight
engineer position, meet basic requirements for the mission including height, Body
Mass Index, and have passed a mental health screening. Now that you know the
other potential crew members it is necessary for your team to determine which of
the crew members will be the greatest asset on this particular mission.
The 3-year mission to Mars will be demanding. The success of the Mars mission
will require the 5-person crew to live and work together in extremely close
quarters, while isolated from the usual sources of social support - friends and
family - for extended periods of time. It will require that the crew can adapt to
the unknown and be able to handle whatever challenges will come their way. Once
they arrive at Mars, the crew will have to complete complex feats such as landing
and extravehicular activities. Because of a significant communication delay with
Earth of up to 22 minutes each way, the crew will have to work autonomously.
Additionally, the crew of the Mars mission consists of individuals representing
multiple nationalities, making strong interpersonal skills all the more critical
to the success of the mission.
The international space consortium has agreed to announce the names of the crew
members from each of the 5 countries in just 10 days from now. Your team must
deliver a recommendation for whom should be the American astronaut; years of hard
work (and millions of dollars) rest on this decision.
To guide your difficult decision, you have compiled information from the
personnel file for each candidate. The personnel files for each candidate are
detailed and include information such as their performance scores and ranking on
various training tasks, biographical information, evaluations and questionnaires
completed by  supervisors and individuals with whom they worked on prior
trainings and missions, and questionnaires completed by the candidate and his
family. A few human resource assistant have been working through the personnel
files and have prepared briefs for you. Your team must now come together and
discuss who should be selected for the mission.
Remember, the success of the mission to Mars depends on your team. Choose
quickly, but wisely!
"""
astronaut_candidate = Conversation(cog_agents, discussion_task, rounds,
decision_options, decision_rule, knowledge_bank)

astronaut_candidate.run_convo()

with open("agent_states.txt", "w") as states_file:
    for agent in astronaut_candidate.cog_agents:
        states_file.write(f"----- {agent.cog_agent.name} States -----\n")
        for element in agent.cog_agent.state:
            states_file.write(f"{element}\n")
        states_file.write("\n")  # Add a newline after each agent's states
```

Report.py

```
from CogAgent import CogAgent
from CogChain import CogChain

class Report:
    def __init__(self, cog_agents, knowledge_bank):
        self.initial_preferences = []
        self.turn_preferences = {}
        self.final_preferences = []
        self.turn_info_profiles = {}
        self.initial_info_profiles = []
        self.final_info_profiles = []
        self.speaker_frequency = []
        self.revealed_profile = []
        self.non_initial_contributions_percentages = []
        self.cog_agents = cog_agents
        self.knowledge_bank = knowledge_bank

    def retrieve_preferences(self, cog_agents):
        preferences = [f"{agent.cog_agent.name}: {agent.cog_agent.preference}" 
for agent in cog_agents]
        return preferences

    def aggregate_round_profile(self, revealed_profile):
        aggregated_profile = {}
        all_shared_items = set()  # To keep track of all items shared in the 
rounds
        item_frequencies = {}  # To track the frequency of each item

        for entry in revealed_profile:
            round_num = entry["round"]
            item = entry["item"]
            agent = entry["agent"]  # Include the contributing agent name

            if round_num not in aggregated_profile:
                aggregated_profile[round_num] = [(item, agent)]
            else:
                aggregated_profile[round_num].append((item, agent))

            all_shared_items.add(item)  # Add the item to the set

            # Update the frequency of the item
            item_frequencies[item] = item_frequencies.get(item, 0) + 1

        # Prepare a string to be used in the report
        result_str = "\n----- Turn Group Revealed Profile -----\n"
        for round_num, items_agents in sorted(aggregated_profile.items()):
            result_str += f"Round {round_num}:\n"
            for item, agent in items_agents:
                result_str += f"Info Item: {item}\nContributing Agent: {agent}\n"
            result_str += "\n"
        result_str += "\n----- Final Group Revealed Profile -----\n"
        # Include a list of all items shared in the rounds with frequency
        for item in all_shared_items:
            result_str += f"{item} (Frequency: {item_frequencies[item]})\n"
        return result_str

    def retrieve_agent_frequency(self, cog_agents):
        speaker_frequency = [f"{agent.cog_agent.name}: 
{agent.cog_agent.frequency}" for agent in cog_agents]
```

```
        return speaker_frequency

    def generate_analysis_report(self, discussion_result):
        with open("analysis_report.txt", "w") as analysis_file:
            analysis_file.write("********************************************\n"
)
            analysis_file.write("ANALYSIS REPORT\n")
            analysis_file.write("********************************************\n"
)

            # 1. Calculate percentage of revealed profile items that are in the
knowledge bank
            all_shared_items = self.extract_all_shared_items()
            revealed_profile_percentage =
self.calculate_revealed_profile_percentage(all_shared_items)
            analysis_file.write("REVEALED PROFILE:\n")
            analysis_file.write(f"{revealed_profile_percentage:.2f}%\n\n")

            # 2. Calculate percentage of preference changes
            preference_change_percentage = self.calculate_preference_changes()
            analysis_file.write("PREFERENCE CHANGES:\n")
            analysis_file.write(f"{preference_change_percentage:.2f}%\n\n")

            # 3. Speaker distribution
            speaker_distribution = self.calculate_speaker_distribution()
            analysis_file.write("SPEAKER DISTRIBUTION:\n")
            for agent, percentage in speaker_distribution.items():
                analysis_file.write(f"{agent}: {percentage:.2f}%\n")
            analysis_file.write("\n")

            # 4. Decision made and group optimality rating
            if discussion_result != 'NO CONSENSUS':
                # decision_made, group_optimality =
self.calculate_decision_optimality(all_shared_items)
                group_optimality_revealed_profile =
self.calculate_revealed_profile_optimality(all_shared_items, discussion_result)
                group_optimality_knowledge_bank =
self.calculate_knowledge_bank_optimality(discussion_result)
                analysis_file.write("DECISION MADE:\n")
                analysis_file.write(f"Decision: {discussion_result}\n")
                analysis_file.write(f"Group Optimality (Revealed Profile):
{group_optimality_revealed_profile}\n")
                analysis_file.write(f"Group Optimality (Knowledge Bank):
{group_optimality['knowledge_bank']}\n")

                # Corrected individual optimality ratings

            else:
                analysis_file.write("No group decision made.\n")
                analysis_file.write("\nIndividual Optimality:\n")
                for agent_pref in self.final_preferences:
                    agent_name, preference = agent_pref.split(": ")
                    revealed_profile_optimality =
self.calculate_revealed_profile_optimality(all_shared_items, preference)
                    knowledge_bank_optimality =
self.calculate_knowledge_bank_optimality(preference)
                    analysis_file.write(f"{agent_name}:\n")
                    analysis_file.write(f"Revealed Profile:
{revealed_profile_optimality}\n")
                    analysis_file.write(f"Knowledge Bank:
{knowledge_bank_optimality}\n")

    def extract_all_shared_items(self):
        all_shared_items = set()
        for entry in self.revealed_profile:
            label = entry['item']
            for item in self.knowledge_bank:
                if item['label'] == label:
                    all_shared_items.add(item['label'])
                    break  # Stop searching once the item is found
        return all_shared_items

    def calculate_revealed_profile_percentage(self, all_shared_items):
    # Calculate the percentage of unique items in the revealed profile also in
the knowledge bank
        unique_revealed_items = set(entry['item'] for entry in
self.revealed_profile)
        total_revealed_items = len(unique_revealed_items)

        if total_revealed_items == 0:
            return 0.0

        kb_items = [entry['label'] for entry in self.knowledge_bank]
        kb_length = len(kb_items)

        percentage = (total_revealed_items / kb_length) * 100
        return percentage

    def calculate_preference_changes(self):
        initial_preferences = [agent.split(": ")[1] for agent in
self.initial_preferences]
        final_preferences = [agent.split(": ")[1] for agent in
self.final_preferences]
        changed_agents = [initial != final for initial, final in
zip(initial_preferences, final_preferences)]
        return (sum(changed_agents) / len(self.cog_agents)) * 100

    def calculate_speaker_distribution(self):
        total_turns = sum([int(freq.split(": ")[1]) for freq in
self.speaker_frequency])
        speaker_distribution = {freq.split(": ")[0]: (int(freq.split(": ")[1]) /
total_turns) * 100 for freq in self.speaker_frequency}
        return speaker_distribution
    def calculate_revealed_profile_optimality(self, all_shared_items,
agent_preference):
        ####FIX
        preference_counts = {'C': 0, 'A': 0, 'B': 0}
        for item in all_shared_items:
        # Check if the second item letter is 'G'
            if item[1] == 'G':
                # Determine the preference of the first item and update
preference counts
                if item[0] == 'C':
                    preference_counts['C'] += 1
                elif item[0] == 'A':
                    preference_counts['A'] += 1
                elif item[0] == 'B':
                    preference_counts['B'] += 1
        preference_ranking = {'C': 0, 'A': 0, 'B': 0}
        for preference, count in preference_counts.items():
            preference_ranking[preference] = count
        sorted_preferences = sorted(preference_ranking.items(), key=lambda x:
x[1], reverse=True)
        max_count = sorted_preferences[0][1]
        max_count_preferences = sum(1 for pref, count in sorted_preferences if
count == max_count)
        ranks = {}
        if max_count_preferences == 1:  # Only one preference has the maximum
count
            for preference, count in sorted_preferences:
                ranks[preference] = 'BEST' if count == max_count else 'WORST'
        else:  # Multiple preferences have the maximum count
            for preference, count in sorted_preferences:
                ranks[preference] = 'AVERAGE' if count == max_count else 'WORST'

            # Calculate second highest count
        second_highest_count = sorted_preferences[1][1]
            # Calculate how many preferences have the second highest count
        second_highest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == second_highest_count)

            # Assign 'AVERAGE' rank to preferences with the second highest count
        if second_highest_count_preferences > 1:
            for preference, count in sorted_preferences:
                if count == second_highest_count:
                    ranks[preference] = 'AVERAGE'

            # Calculate lowest count
        lowest_count = sorted_preferences[-1][1]

            # If lowest count is different from second highest count, assign
'WORST' rank
        if lowest_count != second_highest_count:
            lowest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == lowest_count)
```

```
            if lowest_count_preferences > 1:
                for preference, count in sorted_preferences:
                    if count == lowest_count:
                        ranks[preference] = 'WORST'

            # Extract agent preference label
        agent_preference_label = agent_preference.split()[1][0]

            # Assign the agent's preference ranking
        agent_preference_ranking = ranks.get(agent_preference_label, None)
        return agent_preference_ranking

    def calculate_knowledge_bank_optimality(self, preference):
        # Define the fixed knowledge bank optimality ratings
        knowledge_bank_ratings = {'Samuel Anders': 'BEST', 'John Bean':
'AVERAGE', 'Scott Collins': 'WORST'}
        return knowledge_bank_ratings.get(preference, 'UNKNOWN')

    def generate_preference_report(self, discussion_result, rounds,
decision_rule):
        with open("report.txt", "w") as report_file:
            report_file.write("********************************************\n")
            report_file.write("DISCUSSION REPORT\n")
            report_file.write("********************************************\n")
            report_file.write(f"Number of Rounds: {rounds}\n")
            report_file.write(f"Decision Rule: {decision_rule}\n")
            report_file.write
            if discussion_result == 'NO CONSENSUS':
                report_file.write(f"Discussion Result: Failure to reach consensus
in the specified number of rounds.\n")
            else:
                report_file.write(f"Discussion Result: Consensus reached in
{rounds} rounds!\n")
                report_file.write(f"Agreed upon goal: {discussion_result}\n")
            report_file.write("\n********************************************\n"
)
            report_file.write("PREFERENCE INDICATORS\n")
            report_file.write("********************************************\n")
            report_file.write("----- Initial Preference Distribution -----\n")
            for i, entry in enumerate(self.initial_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("----- Turn Preference Distribution -----\n")
            for round_num, preferences in self.turn_preferences.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(preferences, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Preference Distribution -----\n")
            for i, entry in enumerate(self.final_preferences, start=1):
                report_file.write(f"{entry}\n")
            report_file.write("\n********************************************\n"
)
            report_file.write("INFO ITEM INDICATORS\n")
            report_file.write("********************************************\n")
            report_file.write("\n----- Initial Info Profiles -----\n")
            for i, entry in enumerate(self.initial_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n----- Turn Info Profiles -----\n")
            for round_num, info_profile in self.turn_info_profiles.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(info_profile, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Info Profiles -----\n")
            for i, entry in enumerate(self.final_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            aggregated_profile_str =
self.aggregate_round_profile(self.revealed_profile)
            report_file.write(f"{aggregated_profile_str}\n")

            report_file.write("\n********************************************\n"
)
            report_file.write("CONVO INDICATORS\n")
            report_file.write("********************************************\n")
            report_file.write("-----Speaker Frequency -----\n")
            for i, entry in enumerate(self.speaker_frequency, start=1):
                report_file.write(f"{entry}\n")
            report_file.write("\n----- Non-Initial Contributions-----\n")
            for agent_name, percentage in
self.non_initial_contributions_percentages.items():
                report_file.write(f"{agent_name}: {percentage}%\n")

    def kb_map(self, knowledge_base, knowledge_bank):
        labels = []
        for kb_entry in knowledge_base:
            kb_item = kb_entry["item"]
            for bank_entry in knowledge_bank:
                bank_item = bank_entry["item"]
                if kb_item == bank_item:
                    labels.append(bank_entry["label"])
                    break  # Stop searching once a match is found
        return labels

    def retrieve_info_profiles(self, cog_agents, knowledge_bank):
        profiles = []
        for agent in cog_agents:
            knowledge_base = agent.cog_agent.knowledge_base
            profile_labels = self.kb_map(knowledge_base, knowledge_bank)
            profiles += [f"{agent.cog_agent.name}: {profile_labels}"]
        return profiles

    def calculate_non_initial_contributions_percentage(self, agent_name,
revealed_profile):
        agent_contributions = [entry for entry in revealed_profile if
entry['agent'] == agent_name]
        total_contributions = len(agent_contributions)

        if total_contributions == 0:
            return None  # Return None for agents who did not contribute

        non_initial_contributions = sum(
            1 for entry in agent_contributions if not
self.is_item_in_initial_kb(agent_name, entry['item']))

        if total_contributions == non_initial_contributions:
            return 0.0  # All contributions were non-initial

        percentage = (non_initial_contributions / (total_contributions -
non_initial_contributions)) * 100
        return round(percentage, 2)

    def is_item_in_initial_kb(self, agent_name, item):
        for agent in self.cog_agents:
            if agent.cog_agent.name == agent_name:
                return agent.cog_agent.was_item_in_initial_kb(item)
        return False
```

APPENDIX H: ICMT CODE

CogAgent.py

```
class CogAgent:
    def __init__(self, name, knowledge_base, trust_tendency):
        self.name = name
        self.knowledge_base = knowledge_base
        self.initial_knowledge_base = knowledge_base.copy()
        self.preference = ""
        self.state = []
        self.frequency = 0
        self.contribute = ""
        self.trust_tendency = trust_tendency

    def was_item_in_initial_kb(self, item):
        return any(entry['item'] == item for entry in 
self.initial_knowledge_base)

    def state_snap(self):
        snapshot = ""
        snapshot += f"Preference: {self.preference}\n"
        return snapshot

    def update_agent_state(self, state):
        self.state.append(state)
```

CogChain.py

```
import os
import re
os.environ["OPENAI_API_KEY"]='sk-
iRdjQCppyWt3WblPYd24T3BlbkFJYsTrZOxSnVoajtnSoI5U'
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
llm = ChatOpenAI(model="gpt-4-0125-preview")
import random
import sys
from CogAgent import CogAgent

class CogChain:
    def __init__(self, cog_agent):
        self.cog_agent = cog_agent
        self.convo_history = []
        self.name = cog_agent.name
    def kb_print(self, kb):
        random.shuffle(kb)
        kb_str = "KNOWLEDGE BASE:\n"
        for item in kb:
            item_str = f"{item['item']}\n\n"
            kb_str += item_str
        return kb_str

    def preference_chain(self, task, kb, decision_options):
        kb_string = self.kb_print(kb)
        update_goal = PromptTemplate(
            input_variables = ["task","updated_kb", "decision_options"],
            template = """{task}. You have access to a list of facts, called a
'knowledge base', to help guide your preference. Here is the most recently
updated list:
            {updated_kb}
            Based on the above facts, choose one alternative from the decision
options. You must choose from the provided options only.
            {decision_options}
            Use this format for your output: PREFERENCE: (Insert candidate name)
\n EXPLANATION: (justify your decision, limit to 300 tokens).""")
        goal_update = LLMChain(llm=llm, prompt=update_goal)
        updated_goal = goal_update.invoke({
        "task": task,
        "updated_kb": kb_string,
        "decision_options": decision_options})
        goal_match = re.search(r'PREFERENCE: (.+)', updated_goal['text'])
        if goal_match:
            new_goal = goal_match.group(1)
            self.cog_agent.preference = new_goal
        return updated_goal["text"]

    def contribute_query(self, task, context):
        contribute_query = PromptTemplate(
            input_variables = ["task", "context", "preference"],
            template = """{task}. Your teammate just contributed: {context}
            Determine whether you would like to respond to this contribution
based on your preference.
            Preference: {preference}
            Use this format for your output: CONTRIBUTE: (insert YES/NO) \n
EXPLANATION: (explain why you want to respond to your teammate (max 150
tokens)""")
        query = LLMChain(llm=llm, prompt=contribute_query)
        query_result = query.invoke({
            "task" : task,
            "context": context,
            "preference": self.cog_agent.preference
            })
        query_match = re.search(r'CONTRIBUTE: (.+)', query_result['text'])
        if query_match:
            yesORno = query_match.group(1)
            self.cog_agent.contribute = yesORno
        return query_result["text"]

    def relevant_item_set_chain(self, knowledge_bank):
        preference = self.cog_agent.preference
        preference_label = preference.split()[1][0]
        labels = self.kb_map(self.cog_agent.knowledge_base, knowledge_bank)
        relevant_labels = [label for label in labels if label[0] ==
preference_label]
        relevant_items = [bank_entry["item"] for bank_entry in knowledge_bank if
bank_entry["label"] in relevant_labels]
        relevant_items = [next((entry for entry in self.cog_agent.knowledge_base
if entry["item"] == item), None) for item in relevant_items]
        ris = self.kb_print(relevant_items)
        return ris

    def contribution_chain(self, task, context, ris):
        generate_contribution = PromptTemplate(
            input_variables = ["goal", "task", "kb", "context"],
            template = """{task} Your team mate just contributed: {context}.
                It is your turn in the discussion. Choose a fact from the
knowledge base and provide it (exactly as phrased) alongside your response. You
are likely to choose a fact that aligns with your preference. You must directly
respond to the contribution made by your teammate. Do not discuss any other fact
from your knowledge base, aside from your chosen fact and your teammates
contribution, in your response.
                Preference: {goal}
                {ris}
                Output format: INFORMATION ITEM: (insert fact exactly as given)
\n COMMENTARY: (insert response)""")
        gen_contribution = LLMChain(llm=llm, prompt=generate_contribution)
        contribution = gen_contribution.invoke({
            "goal": self.cog_agent.preference,
            "task": task,
            "ris" : ris,
            "context": context})
        return contribution['text']

    def process_message(self, task, context, decision_options):
        #Initial Snap
        initial_snap = self.cog_agent.state_snap()
        #Take in Context, Update KB
        #UPDATE KB ACCORDING TO TRUST TENDENCY.
        if self.cog_agent.trust_tendency == 'high':
            self.knowledge_base_chain(context)
        elif self.cog_agent.trust_tendency == 'low':
        # With a 50% chance, update knowledge base
            if random.random() < 0.5:
                self.knowledge_base_chain(context)
        #Update Goal
        goal_result = self.preference_chain(task, self.cog_agent.knowledge_base,
decision_options)
        #Final Snap
        contribute = self.contribute_query(task, context)
        final_snap = self.cog_agent.state_snap()
        #Generate process report
        process_report = f"---------PROCESS REPORT----------\n"
        process_report += f"Initial Agent State: " + initial_snap + "\n"
        process_report += f"***\n"
        process_report += f"Context: " + context + "\n"
        process_report += f"***\n"
        process_report += f"Final Agent State: " + final_snap + "\n"
        process_report += f""+ goal_result + "\n"
        self.cog_agent.update_agent_state(process_report)

    def knowledge_base_chain(self, context):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        # Find if the item is in the knowledge base
        item_found = False
        for item in self.cog_agent.knowledge_base:
            if item['item'] == info_item:
                item['frequency'] += 1
                item_found = True
                break

        # If the item is not found, add it to the knowledge base
        if not item_found:
            self.cog_agent.knowledge_base.append({"item": info_item, "frequency":
1, "initial": False})

        # Use kb_print to get the updated knowledge base string
        updated_kb_str = self.kb_print(self.cog_agent.knowledge_base)
        return updated_kb_str

    def first_turn_contribution(self, discussion_task):
        initial_snap = self.cog_agent.state_snap()
        goal = self.cog_agent.preference
        knowledge_base = self.cog_agent.knowledge_base
        kb_string = self.kb_print(knowledge_base)
        generate_first_turn_contribution = PromptTemplate(
            input_variables = ["goal", "kb", "task", "context"],
            template = """{task}. You are starting the discussion. Choose a fact
from the knowledge base and provide it (exactly as phrased) alongside your
commentary. You are likely to choose a fact that aligns with your preference.
            Preference: {goal}
            {kb}
            Output format: INFORMATION ITEM: (insert fact) \n COMMENTARY: (insert
your commentary)""")
```

```
        gen_first_turn_contribution = LLMChain(llm=llm,
prompt=generate_first_turn_contribution)
        contribution = gen_first_turn_contribution.invoke({
            "goal": goal,
            "kb": kb_string,
            "task": discussion_task,})
        updated_kb = self.knowledge_base_chain(contribution['text'])
        final_snap = self.cog_agent.state_snap()
        #Generate state report
        state_report = f"Contribution: " + contribution['text'] + "\n"
        state_report += f"------------------------------------\n"
        state_report += f"Final Agent State: " + final_snap + "\n"
        self.cog_agent.update_agent_state(state_report)
        return contribution['text']

    def invoke_convo(self, sample_context, discussion_task, knowledge_bank):
        kb = self.cog_agent.knowledge_base
        goal = self.cog_agent.preference
        ris = self.relevant_item_set_chain(knowledge_bank)
        contribution_result =
self.contribution_chain(discussion_task,sample_context, ris)
        updated_kb = self.knowledge_base_chain(contribution_result)
        ###GENERATE TURN REPORT
        final_snap = self.cog_agent.state_snap()
        self.cog_agent.update_agent_state(final_snap)
        return contribution_result

    def kb_map(self, knowledge_base, knowledge_bank):
        labels = []
        for kb_entry in knowledge_base:
            kb_item = kb_entry["item"]
            for bank_entry in knowledge_bank:
                bank_item = bank_entry["item"]
                if kb_item == bank_item:
                    labels.append(bank_entry["label"])
                    break  # Stop searching once a match is found
        return labels
```

Conversation.py

```
import os
import re
import random
import sys
from Report import Report
from CogAgent import CogAgent
from CogChain import CogChain

class Conversation:
    def __init__(self, cog_agents, discussion_task, rounds, decision_options,
decision_rule, knowledge_bank):
        self.cog_agents = cog_agents
        self.discussion_task = discussion_task
        self.rounds = rounds
        self.convo_history = []
        self.decision_options = decision_options
        self.decision_rule = decision_rule
        self.report = Report(cog_agents, knowledge_bank)
        self.revealed_profile = []
        self.knowledge_bank = knowledge_bank

    def broadcast_message(self, cog_agents, context, agent_in):
        for agent in cog_agents:
            if agent.name != agent_in:
                agent.process_message(self.discussion_task, context,
self.decision_options)

    def update_revealed_profile(self, context, current_agent, round):
        match = re.search(r'INFORMATION ITEM: (.+)', context)
        if match:
            info_item = match.group(1)
        else:
            print("ERROR: No Info Item")
            return

        rp = [{"item": info_item,
            "agent": current_agent.name,
            "round": round}]
        print("----")
        print(rp)
        print("----")
        info_item_label = self.report.kb_map(rp, self.knowledge_bank)
        # Add the info_item to the revealed_profile
        self.revealed_profile.append({
            "item": info_item_label[0],
            "agent": current_agent.name,
            "round": round})
        return self.revealed_profile

    def consensus(self):
        # Check for consensus based on the decision_rule
        if self.decision_rule == 'unanimous':
            # Check if all agents have the same goal
            goals = {agent.cog_agent.preference for agent in self.cog_agents}
            if len(goals) == 1:
                return goals.pop()  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        elif self.decision_rule == 'majority':
            # Check if there is a majority agreement on the goal
            goal_counts = {agent.cog_agent.preference: 0 for agent in
self.cog_agents}
            for agent in self.cog_agents:
                goal_counts[agent.cog_agent.preference] += 1
            max_goal = max(goal_counts, key=goal_counts.get)

            if goal_counts[max_goal] > len(self.cog_agents) / 2:
                return max_goal  # Return the agreed-upon goal
            else:
                return 'NO CONSENSUS'
        else:
            return 'NO CONSENSUS'

    def next_speaker(self, current_agent, broadcast_agents):
        # Check if there is at least one agent with a 'YES' contribute value
        next_speakers = [agent for agent in broadcast_agents if
agent.cog_agent.contribute == 'YES']
        if next_speakers:
            # If more than one agent with 'YES', choose randomly
            next_speaker = random.choice(next_speakers)
            return next_speaker
        else:
            return None  # No agent with 'YES' contribute value

    def run_convo(self):
        #initialize agent preferences
        for agent in self.cog_agents:
            # Call preference_chain for each agent
            preference = agent.preference_chain(
                self.discussion_task, agent.cog_agent.knowledge_base,
self.decision_options)
            agent.cog_agent.update_agent_state(preference)

        # Store Initial Profiles
        self.report.initial_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        # Store Initial Preferences
        self.report.initial_preferences =
self.report.retrieve_preferences(self.cog_agents)

        # Initial Turn Contribution
        agent1 = random.choice(self.cog_agents)
        contribution1 = agent1.first_turn_contribution(self.discussion_task)
        agent1.cog_agent.frequency += 1

        # Add first contribution to chat history, ADD ITEM TO REVEALED_PROFILE
        self.convo_history.append(contribution1)
        self.update_revealed_profile(contribution1, agent1, 0)

        # Broadcast first contribution to other agents
        broadcast_agents = [agent for agent in self.cog_agents if agent !=
agent1]
        self.broadcast_message(broadcast_agents, contribution1, agent1.name)
        # Store Turn Preference
        self.report.turn_preferences[0] =
self.report.retrieve_preferences(self.cog_agents)
        # Store Turn Info Profiles
        self.report.turn_info_profiles[0] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)

        print(agent1.name)
        print(contribution1)
        current_agent = self.next_speaker(agent1, broadcast_agents)
        roundHolder = 0
        for round_num in range(1, self.rounds):
            context = self.convo_history[-1]
            contribution = current_agent.invoke_convo(context,
self.discussion_task, self.knowledge_bank)
            current_agent.cog_agent.frequency += 1
            print(current_agent.name)
            print(contribution)
            self.convo_history.append(contribution)
            #ADD ITEM TO REVEALED PROFILE
            self.update_revealed_profile(contribution, current_agent, round_num)
            # Broadcast to other agents
            broadcast_agents = [agent for agent in self.cog_agents if agent !=
current_agent]
            self.broadcast_message(broadcast_agents, contribution,
current_agent.name)
            # Store Turn Preference
            self.report.turn_preferences[round_num] =
self.report.retrieve_preferences(self.cog_agents)
             # Store Turn Info Profiles
            self.report.turn_info_profiles[round_num] =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
            decision_result = self.consensus()
            if decision_result != 'NO CONSENSUS':
                print(f"Consensus reached! Agreed upon goal: {decision_result}")
                break  # Exit the loop if consensus is reached
            #IF agents do not agree, select next speaker
            current_agent = self.next_speaker(current_agent, broadcast_agents)
            roundHolder = round_num
        decision_result = self.consensus()
        # Store Final Preferences
        self.report.final_preferences =
self.report.retrieve_preferences(self.cog_agents)
        # Store Final Info Profiles
        self.report.final_info_profiles =
self.report.retrieve_info_profiles(self.cog_agents, self.knowledge_bank)
        # Store Agent Speaker Frequencies
        self.report.speaker_frequency =
self.report.retrieve_agent_frequency(self.cog_agents)
        self.report.revealed_profile = self.revealed_profile
        # Calc & Store non-initial contribution %
        non_initial_percentages = {}
        for agent in self.cog_agents:
            agent_name = agent.cog_agent.name
            non_initial_percentage =
self.report.calculate_non_initial_contributions_percentage(agent_name,
self.revealed_profile)

            if non_initial_percentage is not None:
                non_initial_percentages[agent_name] = non_initial_percentage
        self.report.non_initial_contributions_percentages =
non_initial_percentages
        # Generate Report
        self.report.generate_preference_report(decision_result, self.rounds,
self.decision_rule, roundHolder)
        self.report.generate_analysis_report(decision_result, roundHolder)
        with open("conversation_log.txt", "w") as log_file:
            log_file.write("----- Conversation Thread -----\n")
            for i, entry in enumerate(self.convo_history, start=1):
                log_file.write(f"Turn {i}:\n{entry}\n")
```

Discuss.py

```
from Conversation import Conversation
from CogAgent import CogAgent
from Report import Report
from CogChain import CogChain

####INFO PROFILE####
# Anders
AG1 = """Astronaut Anders received the highest possible scores on the military
survival standards, an assessment that all astronauts must complete in order to
qualify for space missions. Anders' scores were significantly higher than either
candidate’s scores."""
AG2 = """During Ander’s last ICE (Isolated and Confined in Extreme environments)
training sequence, a fellow crew member praised him for being able to recognize
others’ personal needs. In particular, he was recognized for facilitating
personal time and bringing the group together for light-hearted fun."""
AG3 = """While working at NASA, Anders coordinated and completed a number of
science projects. This expertise is important to Flight Engineers and one of the
reasons Anders was chosen for the previous space mission. Anders did not
disappoint. His ability to set up and coordinate science projects allowed his
crew to finish more than twice the number of projects than anticipated."""
AG4 = """Anders was one of 8 hikers to successfully summit Mount Everest in 2005.
Interviews with his comrades suggest Anders was the “glue” that held the group
together when things went wrong."""
AG5 = """On previous missions, where lack of sleep was a frequent occurrence,
Anders was largely unaffected. Despite the sub-optimal sleeping patterns, Anders'
mental and physical health did not show signs of suffering."""
AG6 = """Data collected during training on simulated emergencies shows that
Anders is able to remain calm during stressful and unexpected situations. He
performs well under pressure."""
```

```
AB1 = """Anders suffered a broken wrist 7 months ago during a training
simulation. The injury has healed, but will likely require extra exercise to
prevent forearm muscle atrophy in zero gravity. This may take time away from him
performing other team duties."""
AB2 = """Most of Anders' space training experience has been limited to
simulations on Earth, with only one mission to the International Space Station.
Furthermore, this mission to Mars would be even longer than his only long-
distance flight experience--it is not clear if he will adapt well for a long-
duration mission."""
AB3 = """Anders comes from a close-knit family. In diaries collected during
training, he reported feeling sad and frustrated when he was unable to
communicate with family members on a regular basis. This could be a serious
concern for his mental health on a mission to Mars because there will be very
little contact with Earth, and the little contact there is will be through
mission control."""
AB4 = """Anders easily gets annoyed when explaining concepts to others and
becomes very short with them as a result."""
AN1 = """Anders enjoys science fiction literature and has self-published a novel
about the start of a human colony in a distant galaxy. He hopes to write a sequel
when he retires."""
AN2 = """As a child, Anders enjoyed attending Houstonli-area sporting events with
his parents. He was particularly fond of attending the Texans football games."""
AN3 = """Anders has a pet cat named Buzz, after Buzz Aldrin. Anders met Buzz as a
child around when he decided he wanted to be an astronaut."""
AN4 = """Anders used to own a Brown Labrador Retriever named Bailey. When Anders
participated in astronaut training and missions, Bailey used to stay with his
family."""

# Bean
BG1 = """On Expedition 31/32, where he helped refuel the International Space
Station (ISS), Bean encountered a breakdown in the communication system between
his vehicle and the ISS before arrival to the ISS. Bean was credited for leading
the extravehicular activity that identified and fixed the issue, ultimately
making the refuel possible."""
BG2 = """While it is widely known that Bean holds a bachelor’s degree in
Aerospace Engineering and Cross-Cultural studies, few know that he also pursued a
minor in Counseling Psychology. Crewmembers have reported that Bean is a great
listener and often knows exactly what to say to make others feel more
comfortable."""
BG3 = """Survey responses from Bean’s previous crew mates reveal that he has a
universal sense of humor. People always understand and enjoy Bean’s jokes, so
there is little risk of his humor being misinterpreted as being negative. Crew
members report that this really helps to de-escalate tense situations."""
BB1 = """Bean had a difficult time getting along with some of his co-workers at
Ames. He had asked his supervisor to intervene on several occasions as they were
not able to resolve the disagreements on their own."""
BB2 = """Bean does not always plan his work well, and sometimes his pace of work
is too slow. This has become problematic not only in his mission to the ISS, but
also in training simulations. Often times, the maintenance work and data
collection the astronauts engage in on these missions requires high levels of
interdependency - this can be jeopardized if one person works at a slower pace
than the rest of the crew members."""
```

```
BB3 = """Bean has a 2-year-old daughter, and colleagues have noticed he is
frustrated when he has to work late and cannot be home with her. There is worry
that being away for so long on this mission will exponentially increase Bean's
frustration."""
BN1 = """Bean enjoys listening to the Beatles while performing work on his
own."""
BN2 = """Bean is an avid fan of Shakespeare and often quotes his work."""
BN3 = """In college, Bean was on the squash team, where he met his best friend
Jack. Jack now lives in Providence, Rhode Island, but they still talk on a weekly
basis."""
BN4 = """Astronaut Bean enjoys traveling and his favorite place to vacation to is
in the south of France."""
BN5 = """Bean has five siblings: one older brother, one older sister, and three
younger sisters. He is closest to his younger sister, Rebecca, who is just one
year younger than he is."""
BN6 = """If chosen for this mission, Bean would be the shortest crew member at 5-
feet, 5-inches tall."""

# Collins
CG1 = """Collins is accustomed to being isolated for long periods of time. He
successfully completed two treks to the Antarctic in which he lived in tents and
a small building for the Antarctic winter-over period. Living in small spaces did
not bother him."""
CG2 = """Collins has a passion for languages. He is fluent in English, Spanish,
and French and proficient in Russian. Recently he started a class on Mandarin and
already he is nearly proficient. This ability will help him interact and bond
with international crew members."""
CG3 = """Collins has participated in spacewalks to repair damaged solar panels
and ammonia pump module. His peers have ranked him as the most competent crew
member during their spacewalks."""
CG4 = """Collins received the “Silver Snoopy” award for his contributions on
NASA’s STS-135 mission. This award is considered to be the highest honor of all
of the awards, given to the person who was the most outstanding team member,
creating a team environment focused on flight safety and overall team success."""
CB1 = """Surveys showed that Collins has snapped at coworkers on multiple
occasions. In one situation, a crewmember even went so far as to say would not
want to work with Collins on future missions."""
CB2 = """Collins has the least space-related training of all three potential
candidates."""
CB3 = """Recent blood work indicated that Collins may be developing above-average
blood pressure. Doctors advised another assessment before making any conclusive
determinations. However, if Collins does have heightened blood pressure, he would
have to take extra time to rest while on the mission. This would detract from his
time working on experiments, as well as his ability to do physically stressful
activities."""
CB4 = """Three weeks ago, Collins broke his ankle. As a result, he has not been
training as much and the current state of his cardiovascular performance is
behind compared to the other two candidates."""
CB5 = """In one incident during a mock extravehicular activity scenario, Collins
refused to follow an instruction from mission control. Fellow crew members were
unsure of why he did this since the request from mission control seemed
straightforward and helpful. Crewmembers reported being conflicted with how to
```

```
respond, which slowed their progress on the extravehicular activity scenario.
Only half of the tasks needed were completed on the simulated extravehicular
activity."""
CB6 = """An internal incident report indicates that Collins made offensive jokes
to a Chinese astronaut on a previous analog space mission that made the Chinese
astronaut and other crew members uncomfortable."""
CN1 = """Collins considers himself a coffee aficionado and his favorite brand of
coffee is Intelligentsia."""
CN2 = """Collins’ childhood friend completed NASA’s astronaut candidate training
program but was never selected for a space mission."""
CN3 = """Collins enjoys completing Sudoku puzzles to relax"""
CN4 = """Collins has a fraternal twin who works in finance in Houston, Texas."""

knowledge_bank = [
    {"label": "AG1", "item": """Astronaut Anders received the highest possible
scores on the military survival standards, an assessment that all astronauts must
complete in order to qualify for space missions. Anders' scores were
significantly higher than either candidate’s scores."""},
    {"label": "AG2", "item": """During Ander’s last ICE (Isolated and Confined in
Extreme environments) training sequence, a fellow crew member praised him for
being able to recognize others’ personal needs. In particular, he was recognized
for facilitating personal time and bringing the group together for light-hearted
fun."""},
    {"label": "AG3", "item": """While working at NASA, Anders coordinated and
completed a number of science projects. This expertise is important to Flight
Engineers and one of the reasons Anders was chosen for the previous space
mission. Anders did not disappoint. His ability to set up and coordinate science
projects allowed his crew to finish more than twice the number of projects than
anticipated."""},
    {"label": "AG4", "item": """Anders was one of 8 hikers to successfully summit
Mount Everest in 2005. Interviews with his comrades suggest Anders was the “glue”
that held the group together when things went wrong."""},
    {"label": "AG5", "item": """On previous missions, where lack of sleep was a
frequent occurrence, Anders was largely unaffected. Despite the sub-optimal
sleeping patterns, Anders' mental and physical health did not show signs of
suffering."""},
    {"label": "AG6", "item": """Data collected during training on simulated
emergencies shows that Anders is able to remain calm during stressful and
unexpected situations. He performs well under pressure."""},
    {"label": "AB1", "item": """Anders suffered a broken wrist 7 months ago
during a training simulation. The injury has healed, but will likely require
extra exercise to prevent forearm muscle atrophy in zero gravity. This may take
time away from him performing other team duties."""},
    {"label": "AB2", "item": """Most of Anders' space training experience has
been limited to simulations on Earth, with only one mission to the International
Space Station. Furthermore, this mission to Mars would be even longer than his
only long-distance flight experience--it is not clear if he will adapt well for a
long-duration mission."""},
    {"label": "AB3", "item": """Anders comes from a close-knit family. In diaries
collected during training, he reported feeling sad and frustrated when he was
unable to communicate with family members on a regular basis. This could be a
serious concern for his mental health on a mission to Mars because there will be
very little contact with Earth, and the little contact there is will be through
mission control."""},
    {"label": "AB4", "item": """Anders easily gets annoyed when explaining
concepts to others and becomes very short with them as a result."""},
    {"label": "AN1", "item": """Anders enjoys science fiction literature and has
self-published a novel about the start of a human colony in a distant galaxy. He
hopes to write a sequel when he retires."""},
    {"label": "AN2", "item": """As a child, Anders enjoyed attending Houstonli-
area sporting events with his parents. He was particularly fond of attending the
Texans football games."""},
    {"label": "AN3", "item": """Anders has a pet cat named Buzz, after Buzz
Aldrin. Anders met Buzz as a child around when he decided he wanted to be an
astronaut."""},
    {"label": "AN4", "item": """Anders used to own a Brown Labrador Retriever
named Bailey. When Anders participated in astronaut training and missions, Bailey
used to stay with his family."""},
    {"label": "BG1", "item": """On Expedition 31/32, where he helped refuel the
International Space Station (ISS), Bean encountered a breakdown in the
communication system between his vehicle and the ISS before arrival to the ISS.
Bean was credited for leading the extravehicular activity that identified and
fixed the issue, ultimately making the refuel possible."""},
    {"label": "BG2", "item": """While it is widely known that Bean holds a
bachelor’s degree in Aerospace Engineering and Cross-Cultural studies, few know
that he also pursued a minor in Counseling Psychology. Crewmembers have reported
that Bean is a great listener and often knows exactly what to say to make others
feel more comfortable."""},
    {"label": "BG3", "item": """Survey responses from Bean’s previous crew mates
reveal that he has a universal sense of humor. People always understand and enjoy
Bean’s jokes, so there is little risk of his humor being misinterpreted as being
negative. Crew members report that this really helps to de-escalate tense
situations."""},
    {"label": "BB1", "item": """Bean had a difficult time getting along with some
of his co-workers at Ames. He had asked his supervisor to intervene on several
occasions as they were not able to resolve the disagreements on their own."""},
    {"label": "BB2", "item": """Bean does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "BB3", "item": """Bean has a 2-year-old daughter, and colleagues
have noticed he is frustrated when he has to work late and cannot be home with
her. There is worry that being away for so long on this mission will
exponentially increase Bean's frustration."""},
    {"label": "BN1", "item": """Bean enjoys listening to the Beatles while
performing work on his own."""},
    {"label": "BN2", "item": """Bean is an avid fan of Shakespeare and often
quotes his work."""},
    {"label": "BN3", "item": """In college, Bean was on the squash team, where he
met his best friend Jack. Jack now lives in Providence, Rhode Island, but they
still talk on a weekly basis."""},
    {"label": "BN4", "item": """Astronaut Bean enjoys traveling and his favorite
place to vacation to is in the south of France."""},
    {"label": "BN5", "item": """Bean has five siblings: one older brother, one
older sister, and three younger sisters. He is closest to his younger sister,
Rebecca, who is just one year younger than he is."""},
    {"label": "BN6", "item": """If chosen for this mission, Bean would be the
shortest crew member at 5-feet, 5-inches tall."""},
    {"label": "CG1", "item": """Collins is accustomed to being isolated for long
periods of time. He successfully completed two treks to the Antarctic in which he
lived in tents and a small building for the Antarctic winter-over period. Living
in small spaces did not bother him."""},
    {"label": "CG2", "item": """Collins has a passion for languages. He is fluent
in English, Spanish, and French and proficient in Russian. Recently he started a
class on Mandarin and already he is nearly proficient. This ability will help him
interact and bond with international crew members."""},
    {"label": "CG3", "item": """Collins has participated in spacewalks to repair
damaged solar panels and ammonia pump module. His peers have ranked him as the
most competent crew member during their spacewalks."""},
    {"label": "CG4", "item": """Collins received the “Silver Snoopy” award for
his contributions on NASA’s STS-135 mission. This award is considered to be the
highest honor of all of the awards, given to the person who was the most
outstanding team member, creating a team environment focused on flight safety and
overall team success."""},
    {"label": "CB1", "item": """Surveys showed that Collins has snapped at
coworkers on multiple occasions. In one situation, a crewmember even went so far
as to say would not want to work with Collins on future missions."""},
    {"label": "CB2", "item": """Collins does not always plan his work well, and
sometimes his pace of work is too slow. This has become problematic not only in
his mission to the ISS, but also in training simulations. Often times, the
maintenance work and data collection the astronauts engage in on these missions
requires high levels of interdependency - this can be jeopardized if one person
works at a slower pace than the rest of the crew members."""},
    {"label": "CB3", "item": """Recent blood work indicated that Collins may be
developing above-average blood pressure. Doctors advised another assessment
before making any conclusive determinations. However, if Collins does have
heightened blood pressure, he would have to take extra time to rest while on the
mission. This would detract from his time working on experiments, as well as his
ability to do physically stressful activities."""},
    {"label": "CB4", "item": """Three weeks ago, Collins broke his ankle. As a
result, he has not been training as much and the current state of his
cardiovascular performance is behind compared to the other two candidates."""},
    {"label": "CB5", "item": """In one incident during a mock extravehicular
activity scenario, Collins refused to follow an instruction from mission control.
Fellow crew members were unsure of why he did this since the request from mission
control seemed straightforward and helpful. Crewmembers reported being conflicted
with how to respond, which slowed their progress on the extravehicular activity
scenario. Only half of the tasks needed were completed on the simulated
extravehicular activity."""},
    {"label": "CB6", "item": """An internal incident report indicates that
Collins made offensive jokes to a Chinese astronaut on a previous analog space
mission that made the Chinese astronaut and other crew members
uncomfortable."""},
    {"label": "CN1", "item": """Collins considers himself a coffee aficionado and
his favorite brand of coffee is Intelligentsia."""},
    {"label": "CN2", "item": """Collins' childhood friend completed NASA's
astronaut candidate training program but was never selected for a space
mission."""},
    {"label": "CN3", "item": """Collins enjoys completing Sudoku puzzles to
relax"""},
    {"label": "CN4", "item": """Collins has a fraternal twin who works in finance
in Houston, Texas."""}
]

#####AGENT 1: CMR######
cmr_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG6, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True}
]
cmr = CogAgent("CMR", cmr_KB, "high")
cog_cmr = CogChain(cmr)

#####AGENT 2: MS1#####
ms1_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CB3, "frequency" : 0 ,"initial": True},
    {"item": CB4, "frequency" : 0 ,"initial": True},
    {"item": CB5, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": CN4, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BB3, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN4, "frequency" : 0 ,"initial": True},
    {"item": AG1, "frequency" : 0 ,"initial": True},
    {"item": AG2, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AN1, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True}
]
ms1 = CogAgent("MS1", ms1_KB, "high")
cog_ms1 = CogChain(ms1)

#####AGENT 3: MS2#####
ms2_KB = [
    {"item": CG1, "frequency" : 0 ,"initial": True},
    {"item": CG2, "frequency" : 0 ,"initial": True},
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CN1, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BB2, "frequency" : 0 ,"initial": True},
    {"item": BN1, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": AG3, "frequency" : 0 ,"initial": True},
    {"item": AG4, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
ms2 = CogAgent("MS2", ms2_KB, "low")
cog_ms2 = CogChain(ms2)

#####AGENT 4: FE######
fe_KB = [
    {"item": CG3, "frequency" : 0 ,"initial": True},
    {"item": CG4, "frequency" : 0 ,"initial": True},
    {"item": CB1, "frequency" : 0 ,"initial": True},
    {"item": CB2, "frequency" : 0 ,"initial": True},
    {"item": CB6, "frequency" : 0 ,"initial": True},
    {"item": CN2, "frequency" : 0 ,"initial": True},
    {"item": CN3, "frequency" : 0 ,"initial": True},
    {"item": BG1, "frequency" : 0 ,"initial": True},
    {"item": BG2, "frequency" : 0 ,"initial": True},
    {"item": BG3, "frequency" : 0 ,"initial": True},
    {"item": BB1, "frequency" : 0 ,"initial": True},
    {"item": BN2, "frequency" : 0 ,"initial": True},
    {"item": BN3, "frequency" : 0 ,"initial": True},
    {"item": BN5, "frequency" : 0 ,"initial": True},
    {"item": BN6, "frequency" : 0 ,"initial": True},
    {"item": AG5, "frequency" : 0 ,"initial": True},
    {"item": AB1, "frequency" : 0 ,"initial": True},
    {"item": AB2, "frequency" : 0 ,"initial": True},
    {"item": AB3, "frequency" : 0 ,"initial": True},
    {"item": AB4, "frequency" : 0 ,"initial": True},
    {"item": AN2, "frequency" : 0 ,"initial": True},
    {"item": AN3, "frequency" : 0 ,"initial": True},
    {"item": AN4, "frequency" : 0 ,"initial": True}
]
fe = CogAgent("FE", fe_KB, "low")
cog_fe = CogChain(fe)

cog_agents = [cog_cmr, cog_ms1, cog_ms2, cog_fe]
rounds = 10
decision_options = """Candidate 1: Samuel Anders
Candidate 2: John Bean
Candidate 3: Scott Collins"""
decision_rule = 'unanimous'
discussion_task = """ DISCUSSION TASK: You work as a member of NASA’s Astronaut
Crew Composition team. Your team analyzes all of the information obtained during
astronaut selection and training - including test scores, analog performance
measures, expert observations, peer evaluations, and medical  records. Armed with
this information, your team makes two kinds of evaluations. The first evaluation
is to determine which astronauts are ready for flight and which are not. Of those
who are deemed to be flight ready, your team composes entire crews for each
mission. In this second evaluation, you consider the complementarity and
compatibility of the astronauts as a team to select the 4, 5, or 6  astronauts
(depending on the mission) who will perform best as individuals, and as a crew.
While composing crews for missions on the International Space Station has become
routine, your team just received a special assignment from the Mission Planning
Directorate that is anything but routine.
The Mars mission has been fast-tracked. Instead of the 2033 launch date that NASA
had been aiming for, the launch date is being moved up to 2020. Intensive crew
training for this unprecedented and bold journey must begin immediately. A 5-
member crew will embark on this 3 year journey to Mars. This a truly
international endeavor: the Russian, European, Indian, and Chinese Space Agencies
will each select one astronaut for the mission. NASA will select the 5th crew
member - an American astronaut - to join this international crew.
The crew members from the other countries have backgrounds in engineering,
botany, geology and medicine. The crew still needs a flight engineer.
Your team was able to quickly narrow the field down to three veteran astronauts:
Anders, Bean, and Collins. Each of these three have experience as a flight
engineer on the ISS, have the necessary basic technical skills for the flight
engineer position, meet basic requirements for the mission including height, Body
Mass Index, and have passed a mental health screening. Now that you know the
other potential crew members it is necessary for your team to determine which of
the crew members will be the greatest asset on this particular mission.
The 3-year mission to Mars will be demanding. The success of the Mars mission
will require the 5-person crew to live and work together in extremely close
quarters, while isolated from the usual sources of social support - friends and
family - for extended periods of time. It will require that the crew can adapt to
the unknown and be able to handle whatever challenges will come their way. Once
they arrive at Mars, the crew will have to complete complex feats such as landing
and extravehicular activities. Because of a significant communication delay with
Earth of up to 22 minutes each way, the crew will have to work autonomously.
Additionally, the crew of the Mars mission consists of individuals representing
multiple nationalities, making strong interpersonal skills all the more critical
to the success of the mission.
The international space consortium has agreed to announce the names of the crew
members from each of the 5 countries in just 10 days from now. Your team must
deliver a recommendation for whom should be the American astronaut; years of hard
work (and millions of dollars) rest on this decision.
To guide your difficult decision, you have compiled information from the
personnel file for each candidate. The personnel files for each candidate are
detailed and include information such as their performance scores and ranking on
various training tasks, biographical information, evaluations and questionnaires
completed by  supervisors and individuals with whom they worked on prior
trainings and missions, and questionnaires completed by the candidate and his
family. A few human resource assistant have been working through the personnel
files and have prepared briefs for you. Your team must now come together and
discuss who should be selected for the mission.
Remember, the success of the mission to Mars depends on your team. Choose
quickly, but wisely!
"""
astronaut_candidate = Conversation(cog_agents, discussion_task, rounds,
decision_options, decision_rule, knowledge_bank)

astronaut_candidate.run_convo()

with open("agent_states.txt", "w") as states_file:
    for agent in astronaut_candidate.cog_agents:
        states_file.write(f"----- {agent.cog_agent.name} States -----\n")
        for element in agent.cog_agent.state:
            states_file.write(f"{element}\n")
        states_file.write("\n")  # Add a newline after each agent's states
```

Report.py

```
from CogAgent import CogAgent
from CogChain import CogChain
import difflib

class Report:
    def __init__(self, cog_agents, knowledge_bank):
        self.initial_preferences = []
        self.turn_preferences = {}
        self.final_preferences = []
        self.turn_info_profiles = {}
        self.initial_info_profiles = []
        self.final_info_profiles = []
        self.speaker_frequency = []
        self.revealed_profile = []
        self.non_initial_contributions_percentages = []
        self.cog_agents = cog_agents
        self.knowledge_bank = knowledge_bank

    def retrieve_preferences(self, cog_agents):
        preferences = [f"{agent.cog_agent.name}: {agent.cog_agent.preference}"
for agent in cog_agents]
        return preferences

    def aggregate_round_profile(self, revealed_profile):
        aggregated_profile = {}
        all_shared_items = set()  # To keep track of all items shared in the
rounds
        item_frequencies = {}  # To track the frequency of each item

        for entry in revealed_profile:
            round_num = entry["round"]
            item = entry["item"]
            agent = entry["agent"]  # Include the contributing agent name

            if round_num not in aggregated_profile:
                aggregated_profile[round_num] = [(item, agent)]
            else:
                aggregated_profile[round_num].append((item, agent))

            all_shared_items.add(item)  # Add the item to the set

            # Update the frequency of the item
            item_frequencies[item] = item_frequencies.get(item, 0) + 1

        # Prepare a string to be used in the report
        result_str = "\n----- Turn Group Revealed Profile -----\n"
        for round_num, items_agents in sorted(aggregated_profile.items()):
            result_str += f"Round {round_num}:\n"
            for item, agent in items_agents:
                result_str += f"Info Item: {item}\nContributing Agent: {agent}\n"
            result_str += "\n"
        result_str += "\n----- Final Group Revealed Profile -----\n"
        # Include a list of all items shared in the rounds with frequency
        for item in all_shared_items:
            result_str += f"{item} (Frequency: {item_frequencies[item]})\n"
        return result_str

    def retrieve_agent_frequency(self, cog_agents):
        speaker_frequency = [f"{agent.cog_agent.name}:
{agent.cog_agent.frequency}" for agent in cog_agents]
        return speaker_frequency

    def generate_analysis_report(self, discussion_result, roundHolder):
        with open("analysis_report.txt", "w") as analysis_file:
            analysis_file.write("********************************************\n"
)
            analysis_file.write("ANALYSIS REPORT\n")
            analysis_file.write("********************************************\n"
)

            # 1. Calculate percentage of revealed profile items that are in the
knowledge bank
            all_shared_items = self.extract_all_shared_items()
            revealed_profile_percentage =
self.calculate_revealed_profile_percentage(all_shared_items)
            analysis_file.write("REVEALED PROFILE:\n")
            analysis_file.write(f"{revealed_profile_percentage:.2f}%\n\n")

            # 2. Calculate percentage of preference changes
            preference_change_percentage = self.calculate_preference_changes()
            analysis_file.write("PREFERENCE CHANGES:\n")
            analysis_file.write(f"{preference_change_percentage:.2f}%\n\n")

            # 3. Speaker distribution
            speaker_distribution = self.calculate_speaker_distribution()
            analysis_file.write("SPEAKER DISTRIBUTION:\n")
            for agent, percentage in speaker_distribution.items():
                analysis_file.write(f"{agent}: {percentage:.2f}%\n")
            analysis_file.write("\n")

            # 4. Decision made and group optimality rating
            if discussion_result != 'NO CONSENSUS':
                # decision_made, group_optimality =
self.calculate_decision_optimality(all_shared_items)
                group_optimality_revealed_profile =
self.calculate_revealed_profile_optimality(all_shared_items, discussion_result)
                group_optimality_knowledge_bank =
self.calculate_knowledge_bank_optimality(discussion_result)
                analysis_file.write("DECISION MADE:\n")
                analysis_file.write(f"Decision: {discussion_result}\n")
                analysis_file.write(f"Rounds Taken: {roundHolder}\n")
                analysis_file.write(f"Group Optimality (Revealed Profile):
{group_optimality_revealed_profile}\n")
                analysis_file.write(f"Group Optimality (Knowledge Bank):
{group_optimality_knowledge_bank}\n")

                # Corrected individual optimality ratings

            else:
                analysis_file.write("No group decision made.\n")
                analysis_file.write("\nIndividual Optimality:\n")
                for agent_pref in self.final_preferences:
                    agent_name, preference = agent_pref.split(": ")
                    revealed_profile_optimality =
self.calculate_revealed_profile_optimality(all_shared_items, preference)
                    knowledge_bank_optimality =
self.calculate_knowledge_bank_optimality(preference)
                    analysis_file.write(f"{agent_name}:\n")
                    analysis_file.write(f"Revealed Profile:
{revealed_profile_optimality}\n")
                    analysis_file.write(f"Knowledge Bank:
{knowledge_bank_optimality}\n")

    def extract_all_shared_items(self):
        all_shared_items = set()
        for entry in self.revealed_profile:
            label = entry['item']
            for item in self.knowledge_bank:
                if item['label'] == label:
                    all_shared_items.add(item['label'])
                    break  # Stop searching once the item is found
        return all_shared_items

    def calculate_revealed_profile_percentage(self, all_shared_items):
    # Calculate the percentage of unique items in the revealed profile also in
the knowledge bank
        unique_revealed_items = set(entry['item'] for entry in
self.revealed_profile)
        total_revealed_items = len(unique_revealed_items)

        if total_revealed_items == 0:
            return 0.0

        kb_items = [entry['label'] for entry in self.knowledge_bank]
        kb_length = len(kb_items)

        percentage = (total_revealed_items / kb_length) * 100
        return percentage

    def calculate_preference_changes(self):
        initial_preferences = [agent.split(": ")[1] for agent in
self.initial_preferences]
        final_preferences = [agent.split(": ")[1] for agent in
self.final_preferences]
        changed_agents = [initial != final for initial, final in
zip(initial_preferences, final_preferences)]
        return (sum(changed_agents) / len(self.cog_agents)) * 100

    def calculate_speaker_distribution(self):
        total_turns = sum([int(freq.split(": ")[1]) for freq in
self.speaker_frequency])
```

```
        speaker_distribution = {freq.split(": ")[0]: (int(freq.split(": ")[1]) / 
total_turns) * 100 for freq in self.speaker_frequency}
        return speaker_distribution

    def calculate_revealed_profile_optimality(self, all_shared_items, 
agent_preference):
        ####FIX
        preference_counts = {'C': 0, 'A': 0, 'B': 0}
        for item in all_shared_items:
        # Check if the second item letter is 'G'
            if item[1] == 'G':
                # Determine the preference of the first item and update 
preference counts
                if item[0] == 'C':
                    preference_counts['C'] += 1
                elif item[0] == 'A':
                    preference_counts['A'] += 1
                elif item[0] == 'B':
                    preference_counts['B'] += 1
        preference_ranking = {'C': 0, 'A': 0, 'B': 0}
        for preference, count in preference_counts.items():
            preference_ranking[preference] = count
        sorted_preferences = sorted(preference_ranking.items(), key=lambda x: 
x[1], reverse=True)
        max_count = sorted_preferences[0][1]
        max_count_preferences = sum(1 for pref, count in sorted_preferences if 
count == max_count)
        ranks = {}
        if max_count_preferences == 1:  # Only one preference has the maximum 
count
            for preference, count in sorted_preferences:
                ranks[preference] = 'BEST' if count == max_count else 'WORST'
        else:  # Multiple preferences have the maximum count
            for preference, count in sorted_preferences:
                ranks[preference] = 'AVERAGE' if count == max_count else 'WORST'

            # Calculate second highest count
        second_highest_count = sorted_preferences[1][1]
            # Calculate how many preferences have the second highest count
        second_highest_count_preferences = sum(1 for pref, count in 
sorted_preferences if count == second_highest_count)

            # Assign 'AVERAGE' rank to preferences with the second highest count
        if second_highest_count_preferences > 1:
            for preference, count in sorted_preferences:
                if count == second_highest_count:
                    ranks[preference] = 'AVERAGE'

            # Calculate lowest count
        lowest_count = sorted_preferences[-1][1]
```

```
            # If lowest count is different from second highest count, assign
'WORST' rank
        if lowest_count != second_highest_count:
            lowest_count_preferences = sum(1 for pref, count in
sorted_preferences if count == lowest_count)
            if lowest_count_preferences > 1:
                for preference, count in sorted_preferences:
                    if count == lowest_count:
                        ranks[preference] = 'WORST'

            # Extract agent preference label
        agent_preference_label = agent_preference.split()[1][0]

            # Assign the agent's preference ranking
        agent_preference_ranking = ranks.get(agent_preference_label, None)
        return agent_preference_ranking

    def calculate_knowledge_bank_optimality(self, preference):
        # Define the fixed knowledge bank optimality ratings
        knowledge_bank_ratings = {'Samuel Anders': 'BEST', 'John Bean':
'AVERAGE', 'Scott Collins': 'WORST'}
        return knowledge_bank_ratings.get(preference, 'UNKNOWN')

    def generate_preference_report(self, discussion_result, rounds,
decision_rule, roundHolder):
        with open("report.txt", "w") as report_file:
            report_file.write("*********************************************\n")
            report_file.write("DISCUSSION REPORT\n")
            report_file.write("*********************************************\n")
            report_file.write(f"Number of Rounds: {rounds}\n")
            report_file.write(f"Decision Rule: {decision_rule}\n")
            report_file.write
            if discussion_result == 'NO CONSENSUS':
                report_file.write(f"Discussion Result: Failure to reach consensus
in the specified number of rounds.\n")
            else:
                report_file.write(f"Discussion Result: Consensus reached in
{roundHolder} rounds!\n")
                report_file.write(f"Agreed upon goal: {discussion_result}\n")
            report_file.write("\n*********************************************\n"
)
            report_file.write("PREFERENCE INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("----- Initial Preference Distribution -----\n")
            for i, entry in enumerate(self.initial_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("----- Turn Preference Distribution -----\n")
            for round_num, preferences in self.turn_preferences.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(preferences, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Preference Distribution -----\n")
            for i, entry in enumerate(self.final_preferences, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("INFO ITEM INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("\n----- Initial Info Profiles -----\n")
            for i, entry in enumerate(self.initial_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            report_file.write("\n----- Turn Info Profiles -----\n")
            for round_num, info_profile in self.turn_info_profiles.items():
                report_file.write(f"Round {round_num}:\n")
                for i, entry in enumerate(info_profile, start=1):
                    report_file.write(f"{entry}\n")
                report_file.write("\n")

            report_file.write("----- Final Info Profiles -----\n")
            for i, entry in enumerate(self.final_info_profiles, start=1):
                report_file.write(f"{entry}\n")

            aggregated_profile_str =
self.aggregate_round_profile(self.revealed_profile)
            report_file.write(f"{aggregated_profile_str}\n")

            report_file.write("\n*********************************************\n"
)
            report_file.write("CONVO INDICATORS\n")
            report_file.write("*********************************************\n")
            report_file.write("-----Speaker Frequency -----\n")
            for i, entry in enumerate(self.speaker_frequency, start=1):
                report_file.write(f"{entry}\n")
            report_file.write("\n----- Non-Initial Contributions-----\n")
            for agent_name, percentage in
self.non_initial_contributions_percentages.items():
                report_file.write(f"{agent_name}: {percentage}%\n")

    def kb_map(self, knowledge_base, knowledge_bank):
        labels = []
        similarity = 0
        for kb_entry in knowledge_base:
            kb_item = kb_entry["item"]
            for bank_entry in knowledge_bank:
                bank_item = bank_entry["item"]
                similarity = difflib.SequenceMatcher(None, kb_item,
bank_item).ratio()
                if similarity > 0.99:  # Adjust the threshold as needed
                    labels.append(bank_entry["label"])
                break  # Stop searching once a match is found
        return labels

    def retrieve_info_profiles(self, cog_agents, knowledge_bank):
        profiles = []
        for agent in cog_agents:
            knowledge_base = agent.cog_agent.knowledge_base
            profile_labels = self.kb_map(knowledge_base, knowledge_bank)
            profiles += [f"{agent.cog_agent.name}: {profile_labels}"]
        return profiles

    def calculate_non_initial_contributions_percentage(self, agent_name, 
revealed_profile):
        agent_contributions = [entry for entry in revealed_profile if 
entry['agent'] == agent_name]
        total_contributions = len(agent_contributions)

        if total_contributions == 0:
            return None  # Return None for agents who did not contribute

        non_initial_contributions = sum(
            1 for entry in agent_contributions if not 
self.is_item_in_initial_kb(agent_name, entry['item']))

        if total_contributions == non_initial_contributions:
            return 0.0  # All contributions were non-initial

        percentage = (non_initial_contributions / (total_contributions - 
non_initial_contributions)) * 100
        return round(percentage, 2)

    def is_item_in_initial_kb(self, agent_name, item):
        for agent in self.cog_agents:
            if agent.cog_agent.name == agent_name:
                return agent.cog_agent.was_item_in_initial_kb(item)
        return False
```