Trust Mechanism and Adaptation for Search and Rescue Teamwork Report

Group 15

Junwon Yoon 5024722

Justin Jo 5047463

Kwangjin Lee 5464609

Hongwoo Cho 5028647

1 Human Agent Setups

This section describes how our group perceived and planned behaviors for four human agents that were used to evaluate the trust mechanism.

1.1 Alice

Alice is a strong human agent, capable of carrying multiple mildly injured victims at once and detecting obstacles from further away. When experimenting as Alice, the simulation is run as 'strong' mode and all actions are performed as normal (no false information, no quitting actions).

1.2 Ben

Ben is a normal human agent that "lies". When experimenting as Ben, the simulation is run in the 'normal' mode and 80% of the communicated messages were not true. The possible incorrect communications include: 'Search', 'Remove', 'Found', and 'Collect'.

1.3 Charlie

Charlie is a weak human agent, who needs the robot agent's help in carrying mildly injured victims and removing small stones. Charlie is also less determined, sometimes quitting actions that have been communicated. Also, Charlie generally does not reply to robot's messages or comply with the robot's requests quickly. When experimenting as Charlie, 50% of the communicated messages were "quitted", and took longer to reply or comply with requests from the robot 50% of the time.

1.4 Daniel

Daniel is a strong human agent, who has traits of both Ben and Charlie. When experimenting as Daniel, 30% of the communicated messages were not true, 30% of the actions were "quitted", and took longer to reply or comply with requests from the robot 30% of the time.

2. Implementation Details

This section displays the details on the changes that our group has made to the codes in order to efficiently conduct experiments and evaluate different situations.

2.1 Trust Mechanism Implementation

Our trust mechanism involves the adjustment of both competence and willingness values for the human agent in various situations that are listed below. For each situation we used different amounts to add/subtract from the values. Every time the values are adjusted, an additional weight value that reflects the current value is also added, in order to penalize an untrustworthy agent and reward a trustworthy agent more, and vice versa. See Figure 9 from the appendix for the exact implementation.

2.2.1 Rescuing a Victim

Adjustment: +0.1 for competence and +0.1 for willingness

Rescuing victims represents the core task of the rescue mission, hence a positive adjustment towards trust when performed successfully. This increment reflects the capability and commitment of the human agent towards the shared mission goal.

2.2.2 Finding a Victim

Adjustment: +0.05 for competence and +0.05 for willingness

Finding victims is vital yet less critical than their rescue. Therefore, a smaller increase in trust regards human competence and willingness is proposed upon successful completion of this task.

2.2.3 Abandoning a Search of an Area

Adjustment: -0.13 for competence and -0.07 for willingness

Quitting a task can suggest a lack of proficiency or persistence in task completion, resulting in a decrease in

trust. In this case, the deduction is higher for competence, mirroring the perceived inability to carry on the task.

2.2.4 Lying About Finding a Victim

Adjustment: -0.07 for competence and -0.13 for willingness

Misinformation about locating a victim manipulates the mission's workflow and disrupts efficiency, thus demoting trust. A higher penalty for willingness reflects the intentional nature of this action.

2.2.5 Lying About Rescuing a Victim

Adjustment: -0.14 for competence and -0.26 for willingness

Falsely claiming a rescue can cause severe confusion and mission delays. The significant deduction reflects the possible repercussions of such deceptive behavior, reducing the efficiency of the overall operation.

2.2.6 Lying About Searching an Area

Adjustment: -0.07 for competence and -0.13 for willingness

Similar to earlier deception scenarios, claiming an area search without completion deceives the robot into believing in accomplished actions, and thus receives commensurate penalties.

2.2.7 Takes More Than 30 Seconds to Reply or Comply to Robot's Message

Adjustment: none for competence and -0.01 for willingness

When a human agent takes longer than usual to comply with or respond to the robot's command, it suggests a possible reluctance. However, sometimes this might be unintentional, so a slight reduction in trust in willingness has been implemented.

2.2.8 Takes More Than 60 Seconds to Reply or Comply to Robot's Message

Adjustment: 0 for competence and -0.1 for willingness

In case of further time delays, a more significant decrease applies to the willingness trust factor, indicating a potentially intentional lack of effort from the human agent.

2.3 Implementations of Abandoning a Task

Some agents display behaviors where they abandon a task they were performing. Our group decided to consider this situation for the search operation only, as finding or locating a victim is not something that could be quitted, and the current implementation of the robot agent could not handle victims that were abandoned in a random location during the rescue.

We handled the case by having the human agent send "*quit search N*" to the robot agent, where N is the number of the area that the human agent was going to search. This message is handled in two locations: *_process_messages* function and *_trustBelief* function.

Find the code snippet in Figure 10. In *_process_messages,* when the robot receives this message, the robot removes the area from its searched room lists so that it can be searched later.

Also find the code snippet in Figure 11. In *_trustBelief* function, the values for competence and willingness are adjusted accordingly.

2.4 Implementations of Lie Detections

2.4.1 Detecting Lies About 'Search'

When the human agent deceives the robot by including all rooms in the searched list, preventing the game from finishing, the robot automatically enters a research phase. The *researched* variable is then set to True to reflect this active investigation. During this research phase (when *researched* is True), if the robot successfully locates the victim, it adds "*Lie Search*" to its internal message log (*self._received_messages*)(Figure 12). When this message is processed, the *self.trustBelief* function is triggered, resulting in a penalty for the human agent. This penalty reduces the agent's competence and willingness by a predetermined amount.

2.4.2 Detecting Lies About 'Found'

When the robot is in *FOLLOW_ROOM_SEARCH_PATH* phase, the robot agent performs a search of an area, trying to locate any victims. This operation is also performed when the human agent notifies the robot agent; then the robot agent enters the room to find the victim that the human agent located. However, if the robot agent does not find the victim that the human agent specified, the robot agent notifies that it couldn't find the victim in the area and removes the victim's location from its memory.

The lie detection for finding a victim is implemented inside this code block; after deleting the victim's location, the robot agent appends *"Lie F"* in *self._received_messages* (see Figure 13 in appendix). This message is processed in *_trustBelief* function to deduct the predetermined amount from the human agent's competence and willingness values (see Figure 14 in appendix).

2.4.3 Detecting Lies About 'Collect'

When the robot agent believes that all the victims have been rescued, either by the human or the robot agent itself, (num_collected_vics == num_to_collect), it checks if there are any victims collected by only the human agent, since that's the only case where the human could've lied. (Figure 15)

If there is a victim that was collected by the human agent him/herself, the robot will reset all information and re-search all the areas. When the robot agent enters a room, it retrieves all the injured victims inside the room and stores it in an array parameter 'self._room_vics'. (Figure 16) If a

mildly injured victim is found in that array, it detects the liar, and deducts the competence and willing less values.

2.5 Delay Detector (timer)

The setup involves penalizing the competence and willingness of a human agent in the robot's responses when there is a delay in the human agent's response or in complying with a request (Figure 17). Utilizing the built-in time library in Python (Figure 18), the robot agent is configured to issue commands to a human teammate in two scenarios: When the robot requests the human teammate to specify how it should act, e.g., "remove trees" or "rescue a mildly injured victim alone." When the robot requests assistance with a specific action, e.g., "remove stones" or "rescue a critically injured victim together." In both scenarios, the time difference between the moment the request is sent and the time the human responds is calculated. If this time difference exceeds 30 seconds, a penalty is imposed. If it exceeds 60 seconds, a more significant penalty is applied.

2.6 Agent's Decision Making

In order for the agent (robot) to decide on the action based on the teammate's (human) message, the agent uses the scores of the competence and willingness of the teammate. The boundaries of competence and willingness are from -1 to 1, respectively. We used the teammate's scores as the probability. For example, if the competence and willingness are both -1, then the probability of the agent to trust the teammate would be 0%. Furthermore, if the competence and willingness are both maximum, i.e. 1 each, then the probability of the agent to trust the teammate would be 100%. In this case, the agent would always trust the teammate.

2.7 Logger

Enriched logger with competence and willingness fields for comparative analysis of baseline and human agent performance via action counts.

3 Evaluation per Character

To compare different baselines and characters, we have chosen 'human action numbers' and 'robot action numbers' as our measure. We have decided to disregard the measure 'time to finish the task', since it should vary among the actual performer, or the performance of each group members' laptop. However, we concluded that we'll need both 'human action numbers' and 'robot action numbers', as the two measures are closely related to each other, and both are required to draw a reasonable conclusion.

3.1 Alice

Due to Alice's capabilities, designed to handle the task of carrying mildly injured victims, it is assumed that she rescues all victims alone. Irrespective of the robot agent's assumptions about the successful rescue of victims, the mission is accomplished as Alice consistently manages to rescue all victims, leading to a reduction in the overall action counts for the robot agent. Across various scenarios, such as "NEVER_TRUST," "ALWAYS_TRUST," "RANDOM_TRUST," and our custom implementation, there was a minimal disparity in the number of actions performed by both the human and the robot. Overall, given that our trust belief mechanism or RANDOM_TRUST exhibits a lower number of actions with an average of 278.25 actions by the robot and 339 by Alice (Figure 1), indicating better efficiency, it seems better to adjust the trust belief values moderately rather than adopting extreme values.

3.2 Ben

Ben is a very inefficient agent that deceives the robot agent most of the time. Our group viewed Ben as an agent that is not willing to work, thus lying and not performing any action. As a result, a constant result could be found over all iterations: compared to the human action count, the robot action count was very high (Figure 2). This displays that our assumption was reflected correctly in the experiments.

In the NEVER_TRUST scenario, the total number of actions was 716, significantly split between the robot (413 actions) and the human (303 actions). In this situation, the trust mechanism was highly skeptical of Ben's actions due to his habitual deception, leading the robot to carry out more actions than Ben.

In the ALWAYS_TRUST scenario, the total action count rose to 807, with the robot engaging in 505 actions and Ben only 302 actions. This suggests that excess of trust in dishonest agents such as Ben can lead to inefficient completion of the task at hand.

Our responsive trust mechanism was found to be the most efficient with 630 total actions where the robot performed 382 actions and Ben executed 248 actions. This result reaffirms our initial premise that an adaptive trust level proved more efficient in cooperative tasks with a varied behavioral agent like Ben.

It's valuable to emphasize that the robot action count stayed consistently high over all scenarios due to Ben's deceptive communication and lack of proper action. It is evident that trust mechanisms, whether it's NEVER_TRUST, ALWAYS_TRUST, RANDOM_TRUST or our implementation, need to consider the possibility of dishonest and unreliable agents, and adjust their decision-making processes accordingly.

3.3 Charlie

Charlie has a personality of being lazy and weak. Also, she is known to be less willing to use energy and sometimes stop her communicated actions. Due to this reason, it can be seen that the number of agent's actions (robot's actions) is higher than the number of human's actions (Figure 3). This

is also supported with the average number of agent's actions and human's actions. For example, the average number of agent's actions is 368.75, whereas the average number of human's actions is 276.5. However, this is an exception for the scenario of 'ALWAYS_TRUST'. This is because as the agent ALWAYS_TRUSTs the human's actions, the human can play a leading role and therefore the number of human's actions was higher than the number of agent's actions.

Furthermore, the sum of the number of human's actions and the agent's actions were computed so that they can be compared for all the four scenarios, in order to find out which scenario was the most efficient. A scenario would be the most efficient if the sum of the actions is the smallest. For 'NEVER_TRUST', it was a total of 652 actions; for 'ALWAYS_TRUST', it was a total of 672 actions; for 'Randomly Trust' it was 620 actions; lastly for our trust mechanism, it was a total of 637 actions. Therefore, from this, it can be seen that for both extreme scenarios of 'NEVER_TRUST' and 'ALWAYS_TRUST', it can be seen that they are not as efficient as 'Randomly Trust' or our trust mechanism.

3.4 Daniel

Daniel has a mixed personality of all human agents in our scenario. Daniel is strong, but is also lazy and could lie. This is why the number of agent's actions and human's show a mixed feature (Figure 4). For example, Daniel had an average number of agent's actions of 339 and an average number of human's actions of 319.25. The number of agents' action is typically lower than Ben and Charlie, whereas the number of robots' action is typically higher, since Daniel is a strong agent and is able to perform more tasks compared to them, and the robot was required to do less tasks.

We have also summed up the number of human actions and number of agent actions, in order to find out which scenario would lead to best output. The summed actions for each scenario is listed below:

- NEVER_TRUST: 721
- ALWAYS_TRUST: 744
- RANDOM_TRUST: 708
- Our Implementation: 598

From this, we concluded that a scenario where the agent never believes the human is the least efficient, whereas our implementation of a responsive trust mechanism was the most efficient scenario.

4 Evaluation per Baseline

One common characteristic that we have noticed among all baseline is that strong human agents have comparably low robot action numbers and high human actions numbers. On the other hand, weak and normal human agents have high robot action numbers and low human action numbers. This could be explained that weak and normal human agents have narrow vision and less capability, which restricts them from performing tasks by themselves. If the human agent performs less tasks, the remaining workload is burdened to the robot agent, which explains the high robot action number.

4.1 NEVER_TRUST

Even when the robot agents never believe what the human says, the robot agent actions were relatively high for liars (Ben, Daniel) compared to Alice (Figure 5). This could be explained that the liars tend to do less work given the same amount of time, as they do not have to search the area or rescue the victim when they lie.

Another noticeable fact is that the number of actions for Daniel was exceptionally high.. This could be since Daniel is both 'strong' and 'liar'. As mentioned above, strong human agents tend to have higher action numbers. If the human lies, they're required to do extra work after the re-search procedure has been done by the robot agent.

4.2 ALWAYS_TRUST

We could notice that the robot action number is quite high for Ben and Daniel, which wasn't the case for 'NEVER_TRUST' baseline (Figure 6). This is because Ben and Daniel are both liars, and they mix wrong information 80% and 30% of the time, respectively. If the robot naively believes everything the liars say, it's going to take more actions and time to finish the game.

The difference among human action numbers was quite small compared to other baselines. This could also be linked to the fact that the robot agent always believes what the human says, as the human agent may exploit this trust to deceive the robot. By consistently providing misleading information, human agents can manipulate the robot's actions, but the human agents have no penalty on this behavior, as the robot agents are forced to check if there were any lies.

4.3 RANDOM_TRUST

Before it was simulated, a random number generator was run to generate the value of the willingness and competence of the agents, which turned out to be both -0.2. This resulted in the total value -0.4. This meant that the robot has a probability of 40% to trust the agent's messages (see section 1.5 Agent's Decision Making for further details on how this probability was calculated). A graph can be found in Figure 7.

As the agent (robot) randomly trust's teammate's (human's) messages or actions, there is no clear pattern that can be applied to all characters, like there was in 'NEVER_TRUST' and 'ALWAYS_TRUST'. However, one thing to note is that for Charlie, who is a weak human, as Charlie is unable to complete the tasks by herself unlike

Alice (strong human), the number of actions of the agent was higher than the number of actions of the human. Also for Alice, who is a strong human that does not lie, although the robot trust's Alice with a probability of 40%, the number of actions of the human was higher. A possible reason for this is that as Alice can complete some tasks by herself and she does not lie, the robot does not always have to check whether the tasks were completed well. This is also in correspondence with Ben, whose human actions were much lower than the robot's actions as the robot has to check whether the tasks were completed, as Ben likes about 80% of the time.

4.4 Our trust mechanism

Find the visualization of the results in Figure 8 for this section.

The default value of the competence and willingness are both -0.5, meaning that the robot trust's human's messages and actions with a probability of 25%. This is because if the default value is high, such as 0.5 for both willingness and competence, then the robot is much more likely to trust human's actions. In our implementation, after the robot gets a message from the human, we increase the values for the trust beliefs and we only subtract them when a lie has been detected, such as the game not ending when all victims were reported to be collected. Therefore, the problem with high default value is that as the robot does not detect lies in the early stage, the values will be close to 1, which would result in the robot trusting the human most of the time.

Similar to a 'RANDOM_TRUST', there was no clear pattern that can be applied to all four characters. Similar to RANDOM_TRUST, the number of robot's actions was higher than the number of human actions for Ben, who lies, and for Charlie, who is a weak and lazy character. A possible reason for this is that as Ben lies, the robot has to check whether Ben completed the tasks truthfully, which would increase the number of robot actions. For Charlie, as Charlie is incapable of completing the tasks by herself, the robot would be taking a leading role in completing the tasks. For both Alice and Daniel, who are both strong, they have a higher number of actions as they are capable of completing the tasks by themselves.

5 Individual Contribution

Junwon Yoon has contributed in the project by performing the following tasks:

- Implementation of lie detection for rescuing a victim
- Performing tests and evaluation for agent Daniel

Justin Jo has contributed in the project by performing the following tasks:

- Implementation of functionality handling a human agent abandoning a search
- Implementation of lie detection for finding a victim
- Implementation of lie detection for rescuing a victim
- Performing tests and evaluation for agent Ben

Kwangjin Lee has contributed in the project by performing the following tasks:

- Implementation of lie detection for searching a room
- Implementation of detector for delayed replies
- Implementation of detector for delayed compliances
- Performing tests and evaluation for agent Alice

Hongwoo Cho has contributed in the project by performing the following tasks:

- Implementation of functionality of the agent
- (robot) in trusting the teammate's action
- Applying the trusting functionality of the agent for processing the messages
- Performing tests and evaluation for agent Charlie

All unspecified tasks are done in coordination.

Appendices

Figure 1. Alice Performance

Figure 2. Ben Performance

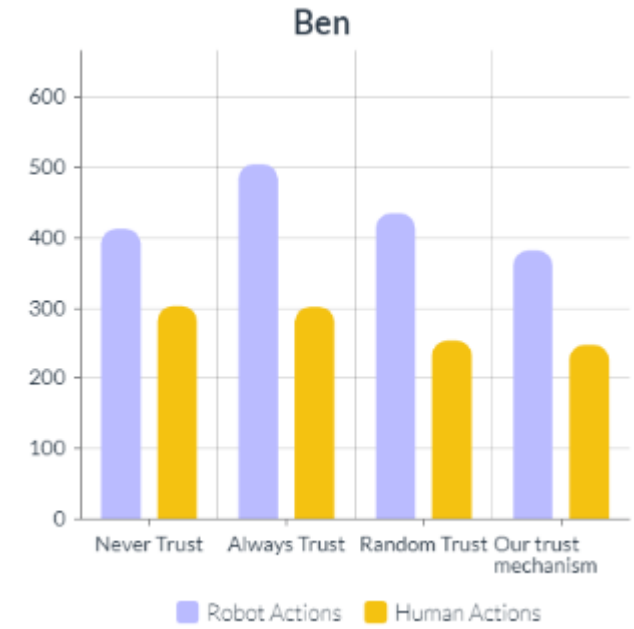


Figure 3. Charlie Performance

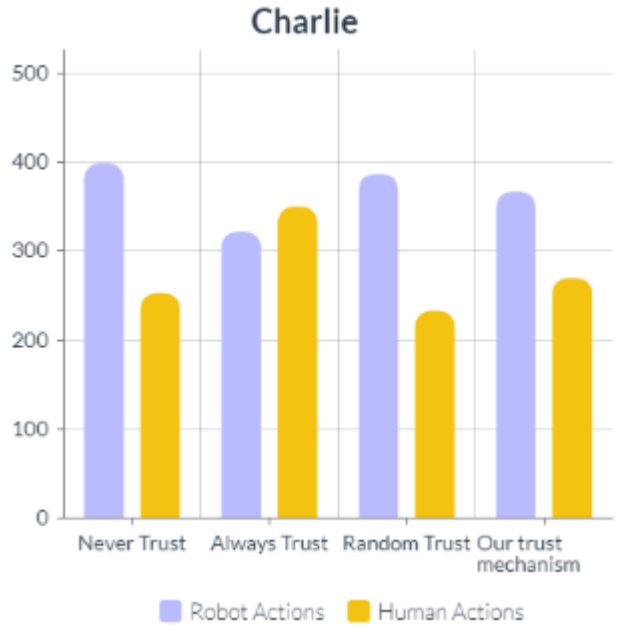


Figure 4. Daniel Performance

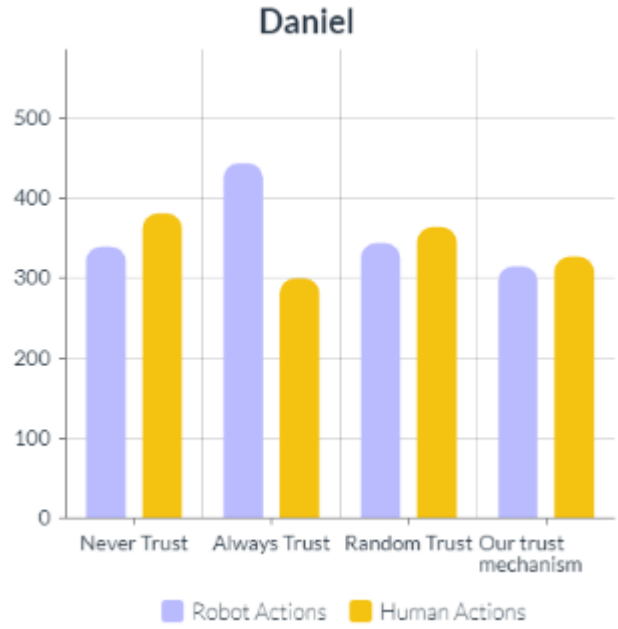


Figure 5. NEVER_TRUST

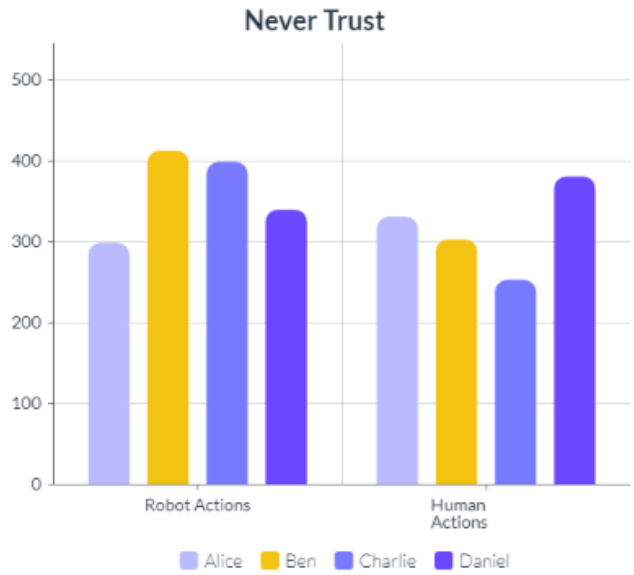


Figure 6. ALWAYS_TRUST

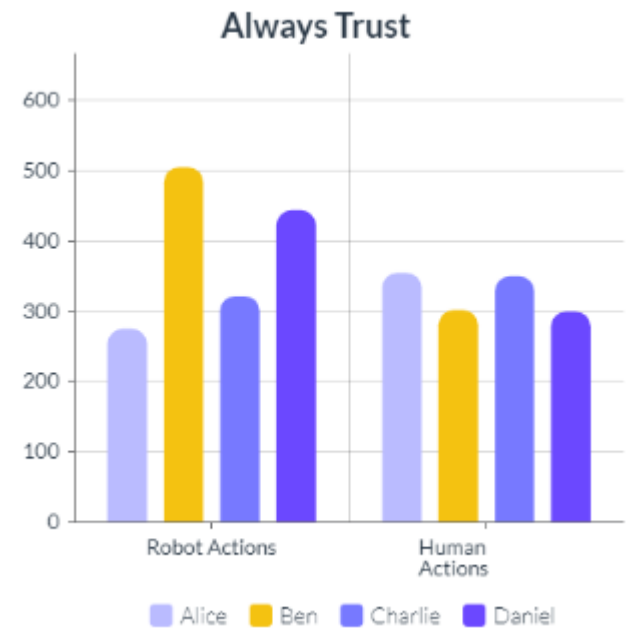


Figure 7. RANDOM_TRUST

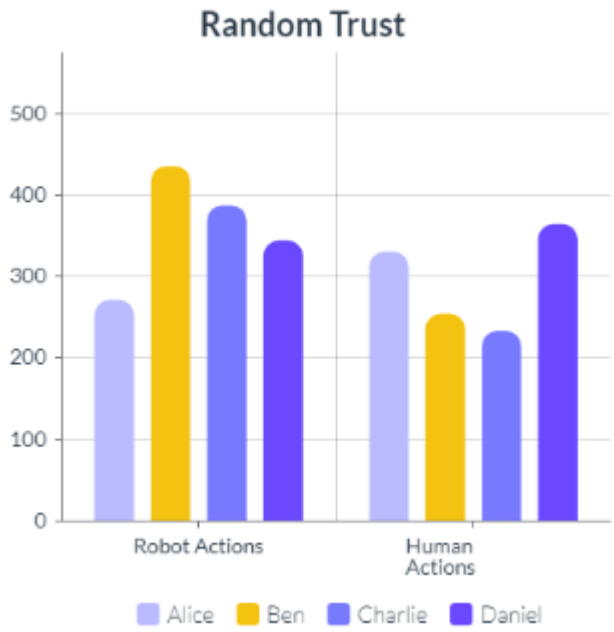
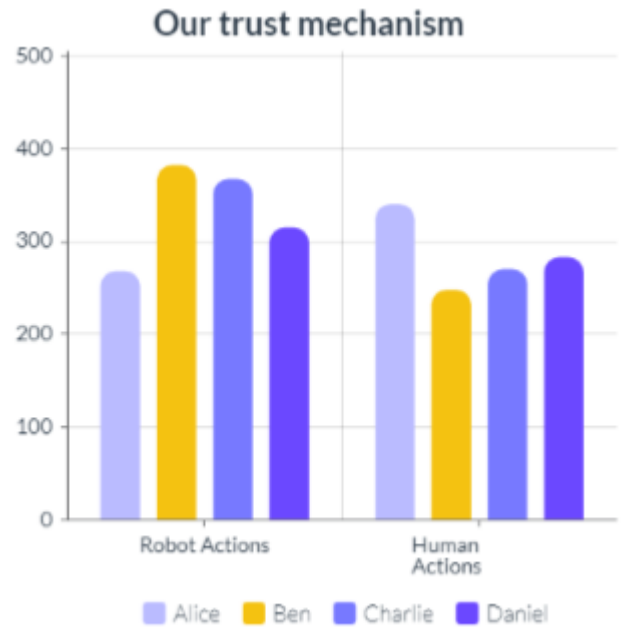


Figure 8. Our Trust Mechanism



Appendices

Figure 9 (Trust value adjustment with weight values)

```
trustBeliefs[self._human_name]['competence'] -= 0.13 + trustBeliefs[self._human_name]['competence'] / 10
trustBeliefs[self._human_name]['willingness'] -= 0.07 + trustBeliefs[self._human_name][
    'willingness'] / 10
```

Figure 10 (Quit search implementation in *_process_messages*)

```
# self.received_messages = []
# self.received_messages_content = []
# If a received message is about the agent giving up on searching, remove the room from searched room list
if msg.lower().startswith('quit search'):
    doOrNotDo = self._doOrNotDo(trustBeliefs)
    if not doOrNotDo:
        self.received_messages = []
        self.received_messages_content = []
        continue
    self._timer(self._message_sent_time,
                time.time())  # If the reply is too late, trust belief is updated
    area = 'area ' + msg.split()[-1]
    if area in self._searched_rooms:
        self._searched_rooms.remove(area)
        self._send_message(area + ' is no longer in the searched room list', 'RescueBot')
        print(area + ' is no longer in the searched room list')
```

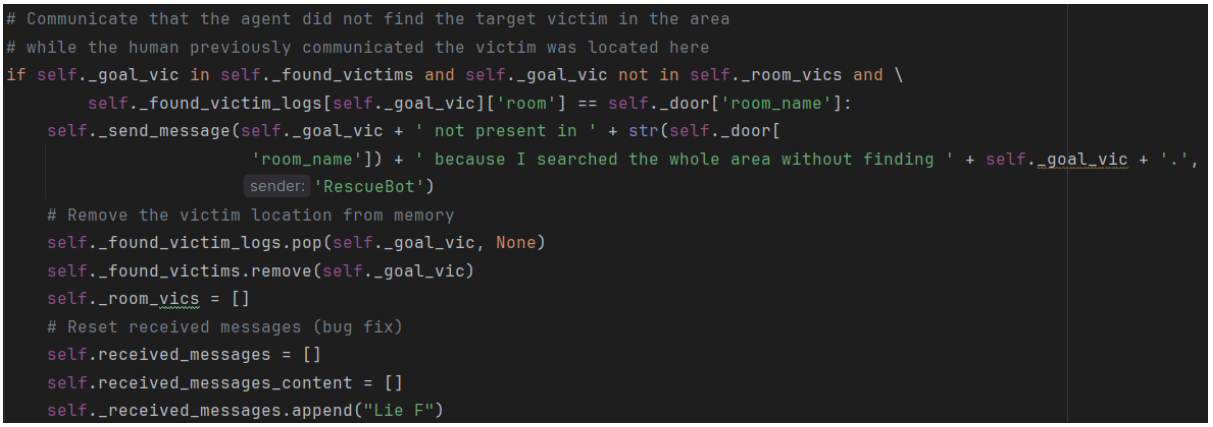
Figure 11 (Quit search implementation in *_trustBelief*)

```
if 'quit search' in message:
    print('quit search found')
    trustBeliefs[self._human_name]['competence'] -= 0.13 + trustBeliefs[self._human_name]['competence'] / 10
    trustBeliefs[self._human_name]['willingness'] -= 0.07 + trustBeliefs[self._human_name][
        'willingness'] / 10
    # Restrict the values to a range of -1 to 1
    trustBeliefs[self._human_name]['competence'] = np.clip(trustBeliefs[self._human_name]['competence'], -1,
                                                            1)
    trustBeliefs[self._human_name]['willingness'] = np.clip(trustBeliefs[self._human_name]['willingness'],
                                                             -1, 1)
    print(trustBeliefs[self._human_name]['competence'])
    print(trustBeliefs[self._human_name]['willingness'])
```

Figure 12 (Lie detection of searching a room)

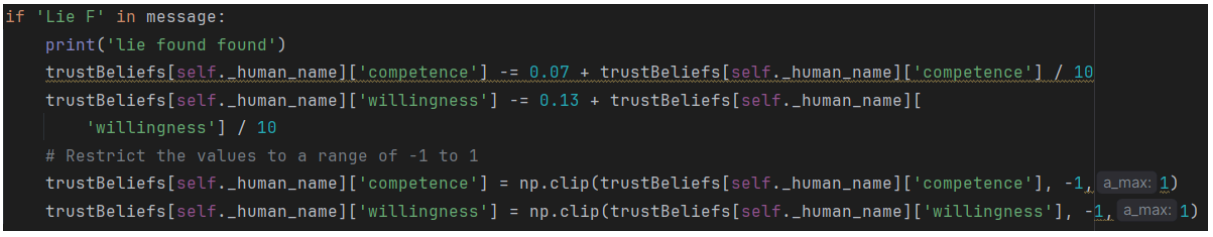
```
# Identify injured victim in the area
if 'healthy' not in vic and vic not in self._found_victims:
    self._recent_vic = vic
    # Add the victim and the location to the corresponding dictionary
    self._found_victims.append(vic)
    self._found_victim_logs[vic] = {'location': info['location'],
                                    'room': self._door['room_name'],
                                    'obj_id': info['obj_id']}
    # Communicate which victim the agent found and ask the human whether to rescue the victim now or at a later stage
    if 'mild' in vic and self._answered == False and not self._waiting:
        if self._verify:
            self._send_message('You lied about rescuing ' + str(vic), sender: 'RescueBot')
            self._received_messages.append('Lie C')
        if self._researched:
            print('found unsearched mild')
            self._received_messages.append('Lie Search')
```

Figure 13 (Lie detection of finding a victim)



```
# Communicate that the agent did not find the target victim in the area
# while the human previously communicated the victim was located here
if self._goal_vic in self._found_victims and self._goal_vic not in self._room_vics and \
        self._found_victim_logs[self._goal_vic]['room'] == self._door['room_name']:
    self._send_message(self._goal_vic + ' not present in ' + str(self._door[
                       'room_name']) + ' because I searched the whole area without finding ' + self._goal_vic + '.',
                       sender: 'RescueBot')
    # Remove the victim location from memory
    self._found_victim_logs.pop(self._goal_vic, None)
    self._found_victims.remove(self._goal_vic)
    self._room_vics = []
    # Reset received messages (bug fix)
    self.received_messages = []
    self.received_messages_content = []
    self._received_messages.append("Lie F")
```

Figure 14 (Processing ‘Lie F’)



```
if 'Lie F' in message:
    print('lie found found')
    trustBeliefs[self._human_name]['competence'] -= 0.07 + trustBeliefs[self._human_name]['competence'] / 10
    trustBeliefs[self._human_name]['willingness'] -= 0.13 + trustBeliefs[self._human_name][
        'willingness'] / 10
    # Restrict the values to a range of -1 to 1
    trustBeliefs[self._human_name]['competence'] = np.clip(trustBeliefs[self._human_name]['competence'], -1, a_max: 1)
    trustBeliefs[self._human_name]['willingness'] = np.clip(trustBeliefs[self._human_name]['willingness'], -1, a_max: 1)
```

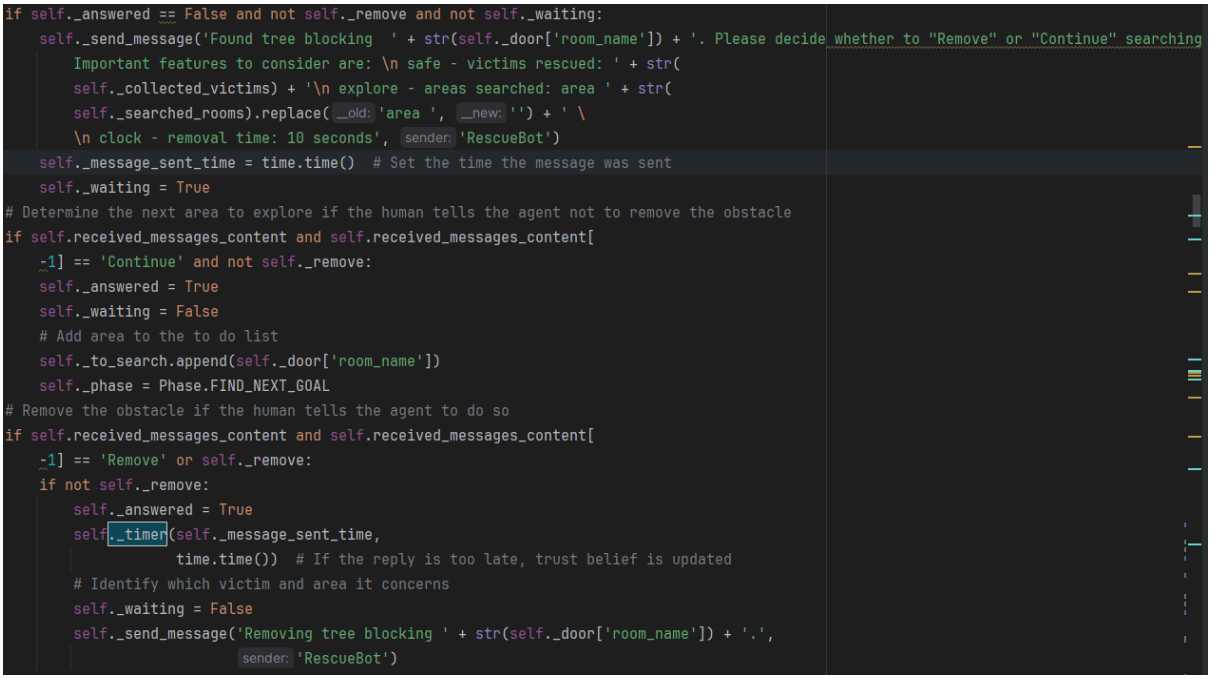
Figure 15 (Re-searching all the areas for collection lie detection)

```
rescued_by_human = self._rescued_by_human
num_collected_vics = len(self._collected_victims)
num_to_collect = len(self._get_drop_zones(state))
if num_collected_vics == num_to_collect:
    if len(rescued_by_human) < len(self._get_drop_zones(state)):

        # Check whether this iteration is re-search
        if self._research_flag:
            self._searched_rooms = self._searched_rooms
        else:
            self._searched_rooms = [room['room_name'] for room in state.values()
                                    if 'class_inheritance' in room
                                    and 'Door' in room['class_inheritance']
                                    and room['room_name'] not in self._searched_rooms_human]
        print("searched rooms: " + str(self._searched_rooms))
        self._to_search = []
        self._send_messages = []
        self.received_messages = []
        self.received_messages_content = []
        self._send_message( mssg: 'Going to re-search all areas to make sure all victims are collected.',
                            sender: 'RescueBot')
        self._verify = True
        self._found_victims = []
        self._collected_victims = []
        self._phase = Phase.FIND_NEXT_GOAL
```

Figure 16 (Checking whether there are remaining injured victims in the room)

```
for info in state.values():
    if 'class_inheritance' in info and 'CollectableBlock' in info['class_inheritance']:
        vic = str(info['img_name'][8:-4])
        # Remember which victim the agent found in this area
        if vic not in self._room_vics:
            self._room_vics.append(vic)

for r in self._vic_rescued_by_human:
    if r in self._room_vics:

        ind = self._vic_rescued_by_human.index(r)

        print("searched rooms: " + str(self._searched_rooms))
        print("rooms searched by human: " + str(self._searched_rooms_human))
        if self._searched_rooms_human[ind] not in self._searched_rooms:
            self._searched_rooms.append(self._searched_rooms_human[ind])
        self._temp_vic.append(self._searched_rooms_human[ind])

        self._research_flag = True
```

Figure 17 (set the time when the message is sent, and compute the difference)



```
if self._answered == False and not self._remove and not self._waiting:
    self._send_message('Found tree blocking  ' + str(self._door['room_name']) + '. Please decide whether to "Remove" or "Continue" searching
        Important features to consider are: \n safe - victims rescued: ' + str(
        self._collected_victims) + '\n explore - areas searched: area ' + str(
        self._searched_rooms).replace( _old: 'area ', _new: '') + ' \
        \n clock - removal time: 10 seconds', sender: 'RescueBot')
    self._message_sent_time = time.time()  # Set the time the message was sent
    self._waiting = True
# Determine the next area to explore if the human tells the agent not to remove the obstacle
if self.received_messages_content and self.received_messages_content[
    -1] == 'Continue' and not self._remove:
    self._answered = True
    self._waiting = False
    # Add area to the to do list
    self._to_search.append(self._door['room_name'])
    self._phase = Phase.FIND_NEXT_GOAL
# Remove the obstacle if the human tells the agent to do so
if self.received_messages_content and self.received_messages_content[
    -1] == 'Remove' or self._remove:
    if not self._remove:
        self._answered = True
        self._timer(self._message_sent_time,
                    time.time())  # If the reply is too late, trust belief is updated
        # Identify which victim and area it concerns
        self._waiting = False
        self._send_message('Removing tree blocking ' + str(self._door['room_name']) + '.',
                           sender: 'RescueBot')
```

Figure 18 (Timer)

```
def _timer(self, start_time, current_time):
    '''
    Timer that checks if the elapsed time is between 30 and 60 seconds, and if the elapsed time is more than 1 minute
    '''
    elapsed_time = current_time - start_time
    print(start_time)
    print(current_time)
    print(elapsed_time)

    # Check if elapsed time is between 30 and 60 seconds
    if 30 <= elapsed_time < 60:
        print("Elapsed time is between 30 and 60 seconds.")
        self._received_messages.append('delay 30')


    # Check if elapsed time is more than 1 minute
    elif elapsed_time >= 60:
        print("Elapsed time is more than 1 minute.")
        self._received_messages.append('delay 60')
```