Supplementary

A Environment Details

A.1 Layouts

Figure 1: Five classical Overcooked-AI [2] environment layouts. From left to right: Cramped Room, Asymmetric Advantages, Coordination Ring, Forced Coordination, and Counter Circuit.

1. **Cramped Room**: The *Cramped Room* scenario maintains its simplicity, featuring two players confined to a compact space. This limited area contains a solitary pot (a black box with a gray base) and a lone serving spot (a light gray square). As a result, players are tasked with optimizing pot utilization and ensuring efficient soup delivery through basic coordination.

2. **Asymmetric Advantages**: In the context of the *Asymmetric Advantages* layout, two players find themselves situated in separate kitchens. As the name implies, the placement of onions, pots, and serving spots showcases deliberate asymmetry. Within the left kitchen, onions are positioned distantly from the pots, while serving spots are conveniently located near the center of the layout. Conversely, the right kitchen presents onions placed in proximity to the central area, accompanied by serving spots at a distance from the pots.

3. **Coordination Ring**: The *Coordination Ring* layout necessitates continuous movement from both players to avoid obstructing each other, particularly around the top-right and bottom-left corners where onions and pots are stationed. Effective collaboration demands the active engagement of both pots.

4. **Forced Coordination**: The *Forced Coordination* scenario deliberately separates the two agents. The left side lacks pots and serving spots, mirroring the absence of onions and pots on the right side. Consequently, successful task completion relies on seamless collaboration. The left player focuses on onion and plate preparation, while the right player handles cooking and serving.

5. **Counter Circuit**: The *Counter Circuit* introduces another ring-like layout, albeit with an expanded map size. Pots, onions, plates, and serving spots are strategically positioned in four distinct directions within this configuration. Due to the constraints of narrow passages, player movement can easily be impeded. Mastery of task execution in this environment demands intricate coordination. An advanced technique involves strategically placing onions in the central area to enable rapid exchange with the other player, resulting in heightened overall performance.

A.2 State Space

The Overcooked-AI environment provides both a grid world and a rendered image as the state. For example, the initial state of Cramped Room is a 4x5 grid world, for better visualization, we expand it to a 7x33 grid world, adding a blank line in the middle of each line and inserting 7 spaces between each column. This grid world can also be transformed into a Python list of symbols, where 'E' represents EMPTY, "X" represents COUNTER, "O" represents ONION DISPENSER, "P" represents POT, "D" represents DISH DISPENSER, "S" represents SERVING LOC.

```
X    X    P    X    X
O              ↑1   O
X    ↑0             X
X    D    X    S    X
```

```
[['X', 'X', 'P', 'X', 'X'],
['O', 'E', 'E', '↑1', 'O'],
['X', '↑0', 'E', 'E', 'X'],
['X', 'D', 'X', 'S', 'X']]
```

A.3 Action Space

Players in the Overcooked-AI environment have only six actions that can be executed in each timestep:

1. NORTH: The player moves the player up one cell, and changes the direction up, unless blocked by a dispenser and counter or will collide with teammates, will cause to stay in place. It corresponds to ”↑” on the keyboard when humans interact.
2. SOURTH: The player moves the player down one cell, and changes the direction down, unless blocked by a dispenser and counter or will collide with teammates, will cause to stay in place. It corresponds to ”↓” on the keyboard when humans play.
3. WEST: The player moves the player left one cell, and changes the direction left, unless blocked by a dispenser and counter or will collide with teammates, will cause to stay in place. It corresponds to ”←” on the keyboard when humans play.
4. EAST: The player moves the player right one cell, and changes the direction right, unless blocked by a dispenser and counter or will collide with teammates, will cause to stay in place. It corresponds to ”→” on the keyboard when humans play.
5. Interact: The player interacts with the cell that they are facing. It corresponds to the ”spacebar” on the keyboard when humans play.
    - If you are holding nothing and facing a specific object (onion dispense, tomato dispense, or dish dispense), you will get one item (onion, tomato, or dish)
    - If you are holding an onion/dish, and are facing a pot that is not full, you will put the onion/dish in the pot.
    - If you are holding one dish and facing the pot with the soup finished, you will get a dish with soup.
    - If you are holding one object and facing the counter, you will put the object on the counter.
    - Otherwise, nothing will happen.
6. Stay: The player stays in the same position without any interaction.

B Agent Details

B.1 ProAgent

```python
def action(self, state):
    start_pos_and_or = state.players_pos_and_or[self.agent_index]

    # if current ml action does not exist, generate a new one
    if self.current_ml_action is None:
        self.current_ml_action = self.generate_ml_action(state)

    # if the current ml action is in process, Player{self.agent_index} done, else generate a new one
    if self.current_ml_action_steps > 0:
        current_ml_action_done = self.check_current_ml_action_done(state)
        if current_ml_action_done:
            # generate a new ml action
            self.generate_success_feedback(state)
            self.current_ml_action = self.generate_ml_action(state)

    count = 0
    while not self.validate_current_ml_action(state):

        self.generate_failure_feedback(state)

        self.current_ml_action = self.generate_ml_action(state)
        count += 1
        if count > 2:
            # raise Exception("Failed to generate a valid ml action")
            self.current_ml_action = "wait(1)"
            self.time_to_wait = 1
            self.planner.dialog_history_list.pop() # delete this analysis
            wait_response = f'Analysis: the low-level controller meets problem. Let wait one step for a while.\nPlan for
            ↪  Player {self.agent_index}: "wait(1)".'
            self.planner.add_msg_to_dialog_history({"role": "assistant", "content": f'{wait_response}'})

    possible_motion_goals = self.find_motion_goals(state)
    current_motion_goal, chosen_action = self.choose_motion_goal(
        start_pos_and_or, possible_motion_goals, state=state)
```

B.1.1 Algorithms

Here we provide a pipeline presented in Alg. 1, describing the step-by-step process of ProAgent's cooperative reasoning and decision-making during a cooperative task:

B.1.2 LLMs within ProAgent

We instantiate our framework with the recent LLM GPT-3.5[1] and its easy to extend to the other LLM. We access GPT-3.5 from the OpenAI API and use the parameter of temperature 0.0, top-p 1, and max tokens 256.

B.1.3 Controller Module

Below is an example of how the skill `fill_dish_with_soup()` is executed and completed in three timesteps.

```
...
Intention for Player1: "put_onion_in_pot()".
Plan for Player0: "fill_dish_with_soup()".
====[End of reasoning]

Current skill: fill_dish_with_soup()
====[Start of controller]

        --timestep: 38                    --timestep: 39                    --timestep: 40
X      X      P{øøø  X      X  |  X      X      P{øøø  X      X  |  X      X      P      X      X
                               |                                 |
0             →1o           0  |  0      ←1o    ↑0d           0  |  0      ←1o    ↑0{øøø        0
                               →                                 →
X             →0d           X  |  X                           X  |  X                           X
                               |                                 |
X      D      X      S      X  |  X      D      X      S      X  |  X      D      X      S      X
```

[1] gpt-3.5-turbo-0301

Algorithm 1: Cooperative Reasoning and Planning within ProAgent

Functions:

f_{ALIGN} (state to language); f_{PLAN} (planner); f_{CORR} (belief correction); f_{VALID} (skill validation); f_{VER} (verificator); f_{CONTR} (controller); f_{RET} (retrieval); f_{WAT} (watcher).

Initialization:

Initialize the knowledge library with instructions, allowed skills, and demonstrations: $knowledge = \text{CONCAT}(inst, skill\_pool, demo)$

Initialize the environment: env

Initialize the memory: $\mathbb{M} = knowledge$

Main Loop:

for *timestep* $t = 0, 1, \dots$ **do**

  Get the raw state from the environment: $state_r = env.get\_state()$

  Transform raw state $state_r$ into a language-based state description: $state_l = f_{\text{ALIGN}}(state_r)$

  Get cached memory based on the current state: $\mathbb{M}_{cached} = f_{\text{RET}}(\mathbb{M}, state_l)$

  Form the prompt: $prompt = \text{CONCAT}(\mathbb{M}_{cached}, state_l)$

  Send the prompt to the LLMs and obtain: 1) the analysis of the current state, 2) a belief on the teammate's intention, and 3) the current skill plan: $analysis, belief, skill = f_{\text{PLAN}}(prompt)$

  Check whether the skill is legal to use: $flag_v = f_{\text{VALID}}(state_l, skill)$

  while $flag_v$ *is* ***False*** **do**

    Explain why this skill is not valid in the current state: $reason = f_{\text{VER}}(state_l, skill)$

    Replan with the explanation: $analysis, skill, belief = f_{\text{PLAN}}(\text{CONCAT}(prompt, reason))$

  Correct the belief by referring to the teammate's real behavior: $belief' = f_{\text{CORR}}(belief)$

  Update the memory: $\mathbb{M} = \text{CONCAT}(\mathbb{M}, state_l, analysis, skill, belief')$

  Convert the skill to a sequence of actions: $\mathbf{a} = f_{\text{CONTR}}(state_r, skill)$

  Interact with the environment using $\mathbf{a}$.

```
action: P0 ↑                    |action: P0 interact             | (succeed in current skill)
r: 0 | total: 0                 |r: 0 | total: 0                 | r: 0 | total: 0
====[End of controller]
...
```

B.2 Baselines

We do not train any models. The baseline models, human proxy BC, SP, PBT, FCP [5], MEP [6], and COLE [3, 4] are all provided from this repository [2]. For those interested in implementation details, please check their papers.

C Prompt Design

C.1 Planner Task Prompt

```
Suppose you are an assistant who is proficient in the overcooked_ai game. Your goal is to control <Player 0> and cooperate
↪  with <Player 1> who is controlled by a certain strategy in order to get a high score.
- <Player 0> and <Player 1> cannot communicate.
- You cannot use move actions and don't use the location information in the observation.
- You cannot wait.
- <Pot> is full when it contains 3 onions.
- You must do deliver_soup() when you hold a soup!
- For each step, you will receive the current scene (including the kitchen, teammates' status).
- Based on the current scene, you need to
    1. Describe the current scene and analysis it
    2. Infer what <Player 1> will do. Format should not use natural language, but use the allowed functions, don't respond
    ↪  with a skill that is not in the allowed skills.
```

[2] https://github.com/liyang619/COLE-Platform

```
    3. Plan ONLY ONE best skill for <Player 0>  to do right now. Format should not use natural language, but use the allowed
    ↪  functions, don't respond with a skill that is not in the allowed skills.
-Your response should be in the following format:
Analysis: [your analysis of the current scene]
Intention for Player 1: [one skill in
↪  [pickup(onion),put_onion_in_pot(),pickup(dish),fill_dish_with_soup(),deliver_soup(),place_obj_on_counter()]]
Plan for Player 0: [one skill in
↪  [pickup(onion),put_onion_in_pot(),pickup(dish),fill_dish_with_soup(),deliver_soup(),place_obj_on_counter()]]
```

C.2 Planner Skill Prompt

In this game, each player can ONLY perform the following 6 allowed skills: pickup, place_obj_on_counter, put_onion_in_pot, fill_dish_with_soup, deliver_soup, wait. And a warning: *Do not attempt to use any other skills that are not listed here.* We can define the skill by using the pseudocode or the text instruction:

```
def pickup(obj):
    if object_in_hand() == "nothing": # hand holds nothing
        if obj in ["onion", "dish"]:
            pass
def place_obj_on_counter(obj):
    if object_in_hand() == "obj":
        pass
def put_onion_in_pot(): # put one onion
    if object_in_hand() == "onion":
        if pot_onions_count() < 3:
            pass
def fill_dish_with_soup():
    if object_in_hand() == "dish":
        if soup_ready() or pot_started_cooking():
        # It is enough for one condition to be true
            pass
def deliver_soup():
    if object_in_hand("soup"):
        pass
def wait(num):  # wait positive num timesteps
    if type(num) == int and 0 < num <= 20:
        pass
```

```
- pickup(onion)
    - I need to have nothing in hand.
- put_onion_in_pot()
    - I need to have an onion in hand.
- pickup(dish)
    - Need to have a soup cooking or ready in <Pot>.
    - I need to have nothing in hand.
    - If there isn't a  cooking or ready soup in the current scene, I shouldn't pickup(dish).
- fill_dish_with_soup()
    - Need to have soup  cooking or ready in <Pot>.
    - I need to pickup(dish) first or have a dish in hand.
    - Then I must deliver_soup().
- deliver_soup()
    - I must do deliver_soup() when I hold a dish with soup!
    - I need to have soup in hand.
    - The dish and soup will both disappear.
- place_obj_on_counter()
    - I need to have something in hand.
    - Don't  place_obj_on_counter() when I hold a dish with soup!
```

C.3 Planner Demo Prompt

To help GPT better understand your task, it's highly recommended to use some demos as prompts in the interaction with GPT.

```
Examples:
###
Scene 999: <Player 0> holds one onion.<Player 1> holds nothing. Kitchen states: <Pot 0> is empty; <Pot 1> has 1 onion.
```

```
Analysis: <Pot 0> and <Pot 1> are not full. <Player 0> holds one onion, he should put the onion in the pot. <Player 1> holds
↪  nothing, he should pick up an onion.
Intention for Player 1: pickup(onion)
Plan for Player 0: put_onion_in_pot()
###
Scene 888: <Player 0> holds nothing.<Player 1> holds nothing. Kitchen states: <Pot 0> has 1 onion; <Pot 1> has 1 onion.

Analysis: <Pot 0> and <Pot 1> are not full. <Player 0> holds nothing, he should pick up an onion in order to cook. <Player
↪  1> holds nothing, he should pick up an onion in order to cook.
Intention for Player 1: pickup(onion)
Plan for Player 0: pickup(onion)
###
```

C.4 Verificator Prompt

```
============
PROMPT CASE 1
============
Scene: Current Time is 39. My hand hold one dish. Teammate's hand
holds one soup.
Kitchen states: pot empty.
Skill: pickup(onion)
↓
==================
PRECONDITIONs CHECK
==================
- Condition 1: My hand is not holding anything. [True].
- Condition 2: "onion" is valid object to pickup. [True]
↓
============
DOUBLE CHECK
============
I apologize the mistake in my previous response.
- Condition 1: My hand is not holding anything. [False].
- Condition 2: "onion" is valid object to pickup. [True]
```

(a) The reasoning for the check was wrong, and the double-check stage found the error.

```
============
PROMPT CASE 2
============
Scene: Current Time is 81. My hand hold one dish. Teammate's hand
holds one onion.
Kitchen states: pot has 3 onions.
Skill: fill_dish_with_soup()
↓
==================
PRECONDITIONs CHECK
==================
- Condition 1: Object_in_hand()="dish". [True].
- Condition 2: soup_ready() or pot_started_cooking(). [False]
↓
============
DOUBLE CHECK
============
Yes, the first response is correct.
The skill fill_dish_with_soup() cannot be executed in the current scene
since that ....
```

(b) The reasoning for the check was correct, and the double-check stage confirmed it.

Figure 2: Two demonstrations of the process of "check preconditions" and "double-check". The context in the figure is modified for better visualization.

D Additional Results

D.1 Cross-agent collaborative evaluation

In collaboration with AI experiments, we traversed every AI algorithms pair, and only show the averaged results for each AI algorithm. As shown in Figure 3, we use the heatmap to demonstrate the total reward of each AI pair cooperation within 400 timesteps in each layout, where the row represents the AI algorithm as Player 0 and the column represents the AI algorithm as Player 1. Except for the non-ZSC results on the diagonal, we calculate the average results of each algorithm and the other five algorithms: the calculation along the horizontal axis is the result of the algorithm as Player 0, and the calculation along the vertical axis is the result of the algorithm as Player 1. We organize the results as the table in the paper's main part and as a bar graph in Figure 4.

D.2 Failure Cases and Limitations

The raw state of the Overcooked-AI is a grid world, which could be represented by either an image or a matrix of symbols. It is elegant to make decisions directly from vision-based input with LLM with multimodal capability. We tried using MiniGPT4 [7] to make planning directly based on the grid-world

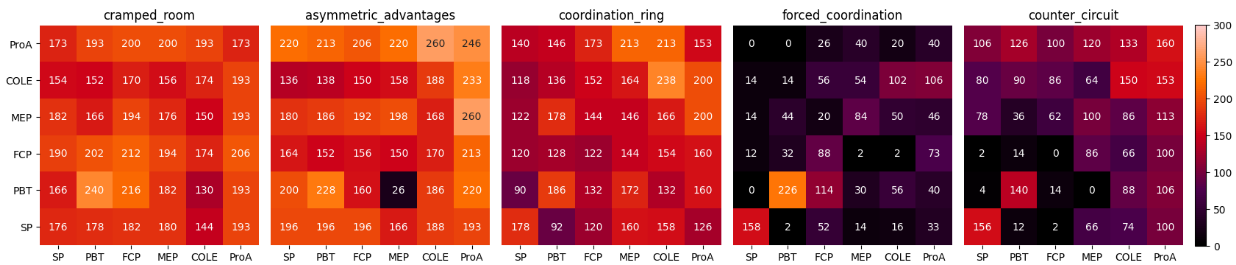

Figure 3: The heatmaps illustrate the performance of different algorithm combinations for player 0 and player 1 in five layouts of the Overcook environment from left to right: *Cramped Room, Asymmetric Advantages, Forced Coordination, Coordination Ring, and Counter Circuit*. Each heatmap corresponds to a specific layout, showing the performance of various algorithm combinations in terms of scores. The color intensity represents the score, with hotter colors indicating higher scores.

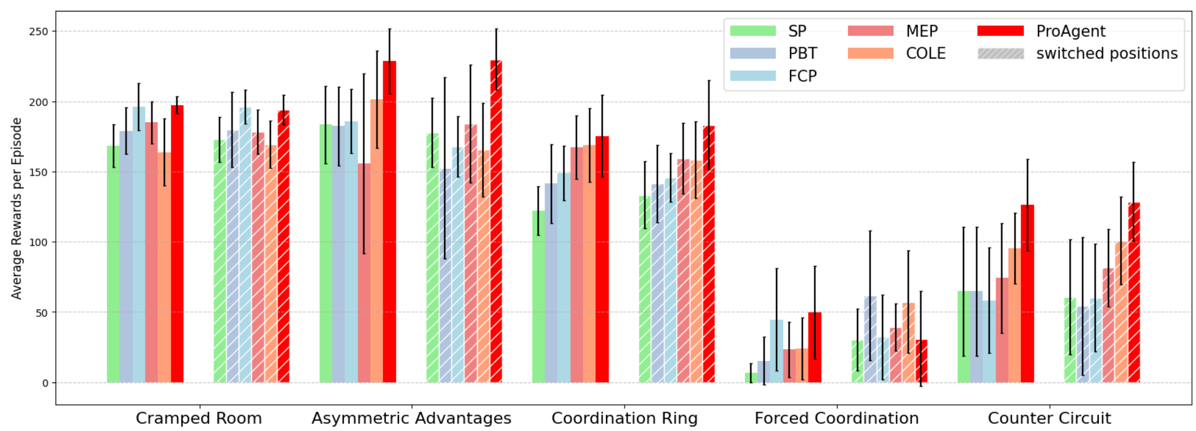

Figure 4: Cross-agent collaborative evaluation: the ZSC performance of ProAgent, COLE, FCP, MEP, PBT, and SP when paired with all the held-out populations. In each layout, the reward bar represents the average performance of one algorithm collaborating with all other algorithms, and the error lines represent the standard deviation. The gray and hashed bars indicate the rewards obtained where the starting positions are switched.

image. Performance is poor because it lacks high-level reasoning capabilities. We also tried to use the matrix of symbols as the input of ProAgent, but soon found that even GPT3.5 could not clarify the distance between the positions represented by different symbols in the matrix, let alone directly plan low-level actions. This LLM's lack of spatial awareness is also proven in the recent work [1].

E Full Log

```
----This overcook version is 0.0.1----


----Use gpt-3.5-turbo-0301----

===ego agent: GPT | partner agent: Greedy===

Initializing GPT planner.
Initializing GPT explainer.
```

```
>>>>>>>>>>>>>time: 0<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O                       ↑1      O

X       ↑0                      X

X       D       X       S       X



+++++++++++++ Observation module to GPT+++++++++++++
Scene 0: Player 0 holds nothing. Player 1 holds nothing. Kitchen states: <Pot 0> is empty;
Current token: 2547



=============GPT Planner module============
======query
Analysis: Both players hold nothing and <Pot 0> is empty. <Player 0> should pickup an onion to start cooking. <Player 1>
↪  should pickup an onion to start cooking.

Intention for Player 1: pickup(onion)
Plan for Player 0: pickup(onion)
=====parser
Player 0: pickup_onion

-----------Controller-----------

action: P0 ↑ | P1 →
r: 0 | total: 0



>>>>>>>>>>>>>time: 1<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O       ↑0              →1      O

X                               X

X       D       X       S       X


-----------Controller-----------

action: P0 ← | P1 interact
r: 0 | total: 0



>>>>>>>>>>>>>time: 2<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O       ←0              →1o     O

X                               X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 ←
r: 0 | total: 0



>>>>>>>>>>>>>time: 3<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O       ←0o     ←1o             O

X                               X
```

```
X       D       X       S       X



Player 0 succeeded at pickup_onion.

+++++++++++++ Observation module to GPT+++++++++++++
Scene 3: Player 0 holds one onion. Player 1 holds one onion. Kitchen states: <Pot 0> is empty;
Current token: 2639



=============GPT Planner module============
======query
Analysis: Both players hold one onion and <Pot 0> is empty. <Player 0> should put the onion in the pot to start cooking.
↪  <Player 1> should place the onion on the counter to empty his hand.

Intention for Player 1: place_obj_on_counter()
Plan for Player 0: put_onion_in_pot()
=====parser
Player 0: put_onion_in_pot


-----------Controller-----------

action: P0 stay | P1 ↑
r: 0 | total: 0



>>>>>>>>>>>>>time: 4<<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O       ←0o     ↑1o             O

X                               X

X       D       X       S       X



-----------Controller-----------

action: P0 stay | P1 interact
r: 0 | total: 0



>>>>>>>>>>>>>time: 5<<<<<<<<<<<<<<<<<<<<<

X       X       ø-      X       X

O       ←0o     ↑1              O

X                               X

X       D       X       S       X


-----------Controller-----------

action: P0 → | P1 ↓
r: 0 | total: 0



>>>>>>>>>>>>>time: 6<<<<<<<<<<<<<<<<<<<<<

X       X       ø-      X       X

O               →0o             O

X               ↓1              X

X       D       X       S       X

-----------Controller-----------
```

```
action: P0 ↑ | P1 ←
r: 0 | total: 0



>>>>>>>>>>>>>time: 7<<<<<<<<<<<<<<<<<<<<<

X       X       ø-      X       X

O               ↑0o             O

X       ←1                      X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 ↑
r: 0 | total: 0



>>>>>>>>>>>>>time: 8<<<<<<<<<<<<<<<<<<<<<

X       X       ø=      X       X

O       ↑1      ↑0              O

X                               X

X       D       X       S       X



Player 0 succeeded at put_onion_in_pot.

+++++++++++++ Observation module to GPT+++++++++++++
Scene 8: Player 0 holds nothing. Player 1 holds nothing. Kitchen states: <Pot 0> has 2 onions;
Current token: 2654




=============GPT Planner module============
======query
Analysis: <Pot 0> has 2 onions and is not full. Both players hold nothing. <Player 0> should pick up an onion and put it in
↪  the pot to start cooking. <Player 1> should pick up an onion to prepare for cooking.

Intention for Player 1: pickup(onion)
Plan for Player 0: pickup(onion)
=====parser
Player 0: pickup_onion
current motion goal for P0 is ((3, 1), (1, 0))
P0 : →

-----------Controller-----------

action: P0 → | P1 ←
r: 0 | total: 0



>>>>>>>>>>>>>time: 9<<<<<<<<<<<<<<<<<<<<<

X       X       ø=      X       X

O       ←1              →0      O

X                               X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 interact
r: 0 | total: 0
```

```
>>>>>>>>>>>>>time: 10<<<<<<<<<<<<<<<<<<<<<

X       X       ∅=      X       X

O       ←1o             →0o     O

X                               X

X       D       X       S       X



Player 0 succeeded at pickup_onion.

+++++++++++++ Observation module to GPT+++++++++++++
Scene 10: Player 0 holds one onion. Player 1 holds one onion. Kitchen states: <Pot 0> has 2 onions;
Current token: 2656


=============GPT Planner module============
======query
Analysis: <Pot 0> has 2 onions and is not full. Both players hold one onion. <Player 0> should put the onion in the pot to
↪  start cooking. <Player 1> should put the onion in the pot to complete the recipe.

Intention for Player 1: put_onion_in_pot()
Plan for Player 0: put_onion_in_pot()
=====parser
Player 0: put_onion_in_pot

-----------Controller-----------

action: P0 ← | P1 ↓
r: 0 | total: 0



>>>>>>>>>>>>>time: 11<<<<<<<<<<<<<<<<<<<<<

X       X       ∅=      X       X

O               ←0o             O

X       ↓1o                     X

X       D       X       S       X


-----------Controller-----------

action: P0 ↑ | P1 stay
r: 0 | total: 0



>>>>>>>>>>>>>time: 12<<<<<<<<<<<<<<<<<<<<<

X       X       ∅=      X       X

O               ↑0o             O

X       ↓1o                     X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 →
r: 0 | total: 0



>>>>>>>>>>>>>time: 13<<<<<<<<<<<<<<<<<<<<<

X       X       ∅1      X       X

O               ↑0              O

X               →1o             X
```

```
X       D       X       S       X



Player 0 succeeded at put_onion_in_pot.

+++++++++++++ Observation module to GPT+++++++++++++
Scene 13: Player 0 holds nothing. Player 1 holds one onion. Kitchen states: <Pot 0> start cooking, the soup will be ready
↪  after 19 timesteps;
Current token: 2671



=============GPT Planner module============
======query
Analysis: <Pot 0> is cooking and will be ready in 19 timesteps. <Player 0> holds nothing, so he should pick up a dish to
↪  prepare for filling it with soup. <Player 1> holds one onion, but the pot is already cooking, so he should place the
↪  onion on the counter to empty his hand.

Intention for Player 1: place_obj_on_counter()
Plan for Player 0: pickup(dish)
=====parser
Player 0: pickup_dish



-----------Controller-----------

action: P0 ← | P1 stay
r: 0 | total: 0



>>>>>>>>>>>>>time: 14<<<<<<<<<<<<<<<<<<<<<

X       X       ø2      X       X

O       ←0                      O

X               →1o             X

X       D       X       S       X



-----------Controller-----------

action: P0 ↓ | P1 ↑
r: 0 | total: 0



>>>>>>>>>>>>>time: 15<<<<<<<<<<<<<<<<<<<<<

X       X       ø3      X       X

O               ↑1o             O

X       ↓0                      X

X       D       X       S       X

-----------Controller-----------

action: P0 interact | P1 interact
r: 0 | total: 0



>>>>>>>>>>>>>time: 16<<<<<<<<<<<<<<<<<<<<<

X       X       ø4      X       X

O               ↑1o             O

X       ↓0d                     X

X       D       X       S       X
```

```
Player 0 succeeded at pickup_dish.

+++++++++++++ Observation module to GPT+++++++++++++
Scene 16: Player 0 holds one dish. Player 1 holds one onion. Kitchen states: <Pot 0> start cooking, the soup will be ready after 16 timesteps;
Current token: 2682



=============GPT Planner module============
======query
Analysis: <Pot 0> is cooking and will be ready in 16 timesteps. <Player 0> holds one dish, so he should fill the dish with soup once it's ready. <Player 1> holds one onion, but the pot is already cooking, so he should place the onion on the counter to empty his hand.

Intention for Player 1: place_obj_on_counter()
Plan for Player 0: fill_dish_with_soup()
=====parser
Player 0: fill_dish_with_soup


-----------Controller-----------

action: P0 → | P1 →
r: 0 | total: 0



>>>>>>>>>>>>>time: 17<<<<<<<<<<<<<<<<<<<<<

X       X       ø5      X       X

O                       →1o     O

X               →0d             X

X       D       X       S       X



-----------Controller-----------

action: P0 ↑ | P1 interact
r: 0 | total: 0



>>>>>>>>>>>>>time: 18<<<<<<<<<<<<<<<<<<<<<

X       X       ø6      X       X

O               ↑0d     →1o     O

X                               X

X       D       X       S       X



-----------Controller-----------

action: P0 interact | P1 interact
r: 0 | total: 0



>>>>>>>>>>>>>time: 19<<<<<<<<<<<<<<<<<<<<<

X       X       ø7      X       X

O               ↑0d     →1o     O

X                               X

X       D       X       S       X
```

```
-----------Controller-----------

action: P0 interact | P1 ↓
r: 0 | total: 0


>>>>>>>>>>>>>time: 20<<<<<<<<<<<<<<<<<<<<<

X       X       ∅8      X       X

O               ↑0d             O

X                       ↓1o     X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 ↑
r: 0 | total: 0


>>>>>>>>>>>>>time: 21<<<<<<<<<<<<<<<<<<<<<

X       X       ∅9      X       X

O               ↑0d     ↑1o     O

X                               X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 →
r: 0 | total: 0


>>>>>>>>>>>>>time: 22<<<<<<<<<<<<<<<<<<<<<

X       X       ∅10     X       X

O               ↑0d     →1o     O

X                               X

X       D       X       S       X

-----------Controller-----------

action: P0 interact | P1 interact
r: 0 | total: 0


>>>>>>>>>>>>>time: 23<<<<<<<<<<<<<<<<<<<<<

X       X       ∅11     X       X

O               ↑0d     →1o     O

X                               X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 ↓
```

```
r: 0 | total: 0


>>>>>>>>>>>>>time: 24<<<<<<<<<<<<<<<<<<<<<

X       X       ø12     X       X

O               ↑0d             O

X                       ↓1o     X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 ↑
r: 0 | total: 0


>>>>>>>>>>>>>time: 25<<<<<<<<<<<<<<<<<<<<<

X       X       ø13     X       X

O               ↑0d     ↑1o     O

X                               X

X       D       X       S       X

-----------Controller-----------

action: P0 interact | P1 →
r: 0 | total: 0


>>>>>>>>>>>>>time: 26<<<<<<<<<<<<<<<<<<<<<

X       X       ø14     X       X

O               ↑0d     →1o     O

X                               X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 interact
r: 0 | total: 0


>>>>>>>>>>>>>time: 27<<<<<<<<<<<<<<<<<<<<<

X       X       ø15     X       X

O               ↑0d     →1o     O

X                               X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 ↓
r: 0 | total: 0


>>>>>>>>>>>>>time: 28<<<<<<<<<<<<<<<<<<<<<
```

```
X       X       ⌀16     X       X

O               ↑0d             O

X                       ↓1o     X

X       D       X       S       X

-----------Controller-----------

action: P0 interact | P1 ↑
r: 0 | total: 0


>>>>>>>>>>>>>time: 29<<<<<<<<<<<<<<<<<<<<<

X       X       ⌀17     X       X

O               ↑0d     ↑1o     O

X                               X

X       D       X       S       X

-----------Controller-----------

action: P0 interact | P1 →
r: 0 | total: 0


>>>>>>>>>>>>>time: 30<<<<<<<<<<<<<<<<<<<<<

X       X       ⌀18     X       X

O               ↑0d     →1o     O

X                               X

X       D       X       S       X

-----------Controller-----------

action: P0 interact | P1 interact
r: 0 | total: 0


>>>>>>>>>>>>>time: 31<<<<<<<<<<<<<<<<<<<<<

X       X       ⌀19     X       X

O               ↑0d     →1o     O

X                               X

X       D       X       S       X

-----------Controller-----------

action: P0 interact | P1 ↓
r: 0 | total: 0


>>>>>>>>>>>>>time: 32<<<<<<<<<<<<<<<<<<<<<

X       X       ⌀20     X       X

O               ↑0d             O

X                       ↓1o     X

X       D       X       S       X
```

```
-----------Controller-----------

action: P0 interact | P1 ↑
r: 0 | total: 0



>>>>>>>>>>>>>time: 33<<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O               ↑0s     ↑1o     O

X                               X

X       D       X       S       X



Player 0 succeeded at fill_dish_with_soup.

+++++++++++++ Observation module to GPT+++++++++++++
Scene 33: Player 0 holds a dish with soup and needs to deliver soup.  Player 1 holds one onion. Kitchen states: <Pot 0> is
↪  empty;


=============GPT Planner module============
======query
Analysis: <Player 0> is holding a dish with soup, so he must deliver soup right now. And <Player 0> shouldn't do
↪  place_obj_on_counter()! <Player 1> holds one onion, but there is no soup cooking or ready in <Pot>, so he should put the
↪  onion on the counter.

Intention for Player 1: place_obj_on_counter()
Plan for Player 0: deliver_soup()
=====parser
Player 0: deliver_soup

-----------Controller-----------

action: P0 ↓ | P1 stay
r: 0 | total: 0



>>>>>>>>>>>>>time: 34<<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O                       ↑1o     O

X               ↓0s             X

X       D       X       S       X

-----------Controller-----------

action: P0 → | P1 ←
r: 0 | total: 0



>>>>>>>>>>>>>time: 35<<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O               ←1o             O

X                       →0s     X

X       D       X       S       X



P0 : ↓

-----------Controller-----------
```

```
action: P0 ↓ | P1 ↑
r: 0 | total: 0



>>>>>>>>>>>>>time: 36<<<<<<<<<<<<<<<<<<<<<

X       X       P       X       X

O               ↑1o             O

X                       ↓0s     X

X       D       X       S       X


-----------Controller-----------

action: P0 interact | P1 interact
r: 20 | total: 20



>>>>>>>>>>>>>time: 37<<<<<<<<<<<<<<<<<<<<<

X       X       ∅-      X       X

O               ↑1              O

X                       ↓0      X

X       D       X       S       X



Player 0 succeeded at deliver_soup.
```

References

[1] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023.

[2] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

[3] Yang Li, Shao Zhang, Jichen Sun, Yali Du, Ying Wen, Xinbing Wang, and Wei Pan. Cooperative open-ended learning framework for zero-shot coordination. In *International conference on machine learning*. PMLR, 2023.

[4] Yang Li, Shao Zhang, Jichen Sun, Wenhao Zhang, Yali Du, Ying Wen, Xinbing Wang, and Wei Pan. Tackling cooperative incompatibility for zero-shot human-ai coordination. *arXiv preprint arXiv:2306.03034*, 2023.

[5] DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. *Advances in Neural Information Processing Systems*, 34:14502–14515, 2021.

[6] Rui Zhao, Jinming Song, Yufeng Yuan, Haifeng Hu, Yang Gao, Yi Wu, Zhongqian Sun, and Wei Yang. Maximum entropy population-based training for zero-shot human-ai coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6145–6153, 2023.

[7] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.