Collaborative Problem-Solving in an Optimization Game

Isidora Jeknić
Saarland University
Saarbrücken, Germany
jeknic@lst.uni-saarland.de

Alex Duchnowski
Saarland University
Saarbrücken, Germany
aduchnowski@coli.uni-saarland.de

Alexander Koller
Saarland University
Saarbrücken, Germany
koller@coli.uni-saarland.de

Abstract

Dialogue agents that support human users in solving complex tasks have received much attention recently. Many such tasks are NP-hard optimization problems that require careful collaborative exploration of the solution space. We introduce a novel dialogue game in which the agents collaboratively solve a two-player Traveling Salesman problem, along with an agent that combines LLM prompting with symbolic mechanisms for memory, state tracking and problem-solving. Our best agent solves 45% of games optimally in self-play. It also demonstrates an ability to collaborate successfully with human users and generalize to unfamiliar graphs.

1 Introduction

Humans frequently face the challenge of solving hard combinatorial problems in their daily lives, from planning to scheduling to resource allocation, and they often struggle to solve these problems well (Hidalgo-Herrero et al., 2013). LLMs are surprisingly good at solving these problems (Fan et al., 2024), but only when the human user has full knowledge of the problem's contributing variables and constraints, and can spell it out in detail. One of the great promises of LLMs is that they can help humans solve such problems *collaboratively* (Gundawar et al., 2024), through a dialogue in which the human and the system take turns exploring the problem and proposing increasingly complete and correct solutions.

AI agents for collaborative problem-solving must have a number of fundamental skills to be effective. In addition to the ability to solve NP-hard problems, they also need to perform conversational grounding (tracking the details of the problem and the parts of the solution that have been agreed upon), remember what their partner wants and knows, and negotiate and revise partial solutions. All of these are established problems in the dialogue literature, and it is not obvious that LLMs will solve them easily.

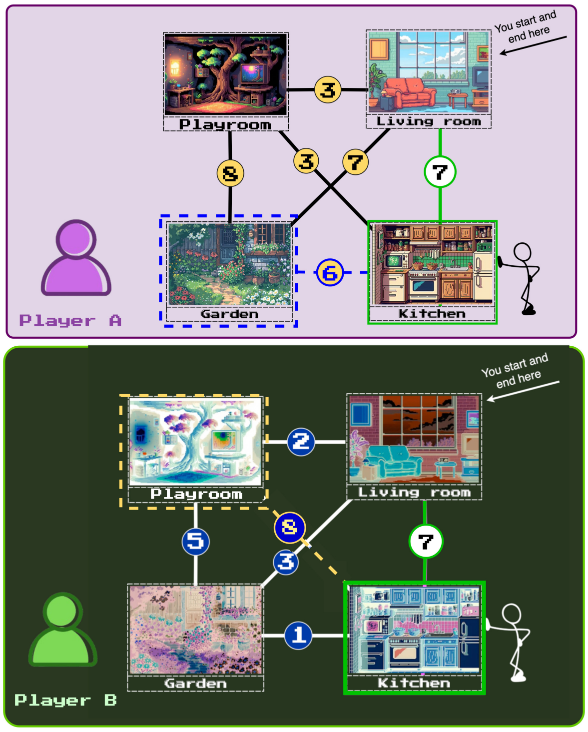


Figure 1: The two players' (A, B) perspectives of the same move, in their respective dimensions (light, ghost). Solid green lines represent the path already taken, whereas the dashed lines represent each player's local optimal next move - for Player A, it's the Garden, and for Player B it's the playroom.

In this paper, we make two contributions to the development of collaborative agents for solving complex problems.[1] First, we introduce TRAVELING ADVENTURERS, a game in which two agents collaboratively solve a Traveling Salesman problem. This game serves as a testbed for collaborative problem-solving, as each agent only has partial information about the problem initially and the agents must communicate and collaborate in or-

[1]Our code and data are available at https://github.com/coli-saar/collaborative-problem-solving.

Proceedings of the 26th Annual Meeting of SIGDIAL, pages 780–799
Aug 25–27, 2025. ©2025 SIGDIAL

der to negotiate an optimal solution. Furthermore, because it builds on an NP-hard optimization problem, TRAVELING ADVENTURERS is difficult enough that humans could not immediately find optimal solutions even with access to their partner's knowledge, necessitating the incremental construction of a solution that we expect in a collaborative problem-solving dialogue.

Second, we present an artificial agent that performs well on playing TRAVELING ADVENTURERS. We find that a purely LLM-based baseline struggles with tracking its partner's knowledge and conversational grounding; it is decisively outperformed by a neurosymbolic agent which combines the LLM with symbolic components for these skills and an exact symbolic optimizer. The neurosymbolic agent solves 98% of its games correctly and 45% optimally in self-play, and still achieves a high optimality score in 32% of games with human partners. We conclude with a discussion of conversational skills that would be required to push the optimality rate even higher.

2 Background

In order to facilitate collaboration, interlocutors must engage in dialogue so as to ensure a shared set of intentions (Dafoe et al., 2021). Collaboration in dialogue can take the form of a negotiation, carried out through a set of dialogue acts such as **presenting**, **accepting**, or **rejecting** a proposal (Georgila and Traum, 2011). A collaborative agent must successfully process and produce such acts, at least implicitly.

Task-oriented collaborative dialogue is primarily studied through *collaborative games* which foster negotiation by incentivizing goal-driven conversation (Schlangen, 2019). This often includes simple problem-solving tasks (Zarrieß et al., 2016; Lewis et al., 2017; Kim et al., 2019; Jeknic et al., 2024). These tasks stand in contrast to NP-hard optimization problems, which ensure that no player can easily come to an optimal solution alone, therefore prompting negotiation and collaboration.

Examples of more complex tasks include TRAINS (Allen et al., 1995) and Overcooked (Carroll et al., 2020; Ghost Town Games, 2016). The former involves collaboration between a human manager and an artificial assistant to regulate a railway and satisfy constraints, whereas in the latter, players must coordinate various steps of the cooking process in order to make dishes. We offer an environment of high complexity in terms of problem-solving and negotiation that is minimal enough to be addressed by prompting and crowdsourcing.

LLMs have shown promising results and further potential on very complex tasks (see Cheng et al. (2024); Mialon et al. (2023); Hartmann and Koller (2024) for an overview). However, previous work has shown that even one of the most advanced models (at the time, GPT-4) severely underperforms humans on strategic thinking tasks (Zhou et al., 2024). Similarly, when dealing with complex tasks, human input can be beneficial for agent performance and compensate for certain model-specific failures (Lin et al., 2024). Finally, most previous work investigating LLMs and NP-hard problems is focused on single-agent optimization (Duchnowski et al., 2025; Ramamonjison et al., 2023; Tang et al., 2025; Khan and Hamad, 2024).

3 The Game

We present TRAVELING ADVENTURERS, a two-player game environment for collecting task-oriented collaborative dialogue. Players A and B observe a fully connected, undirected graph G: the nodes (V) and edges (E) are identical for both players, whereas the weights w_A, w_B are unique and only known to the respective player. The goal is for the players to agree on a single, identical path that visits all nodes exactly once and returns to the starting node while maximizing the sum of A's and B's weights along the way. The players communicate exclusively through written text and have access to the entire chat history throughout the game.

Story. The players are told that an evil wizard imprisoned them in two parallel dimensions of the same house: the *light* and *ghost* dimensions (see Fig. 1 for illustration). They observe the same layout, where each room is connected to all the others via hallways. In order to break the curse, the players must perform a ritual: they must visit each room together and return to the starting point. Moreover, the players get to keep the loot (*light* and *ghost* coins) that they jointly collect as they traverse the hallways between the rooms.

Features. The game mechanics are based on a classic NP-hard optimization problem, the Traveling Salesman Problem (TSP), which we extend to two players and "dress up" (Duchnowski et al., 2025) as a coin-collecting game. In relation to

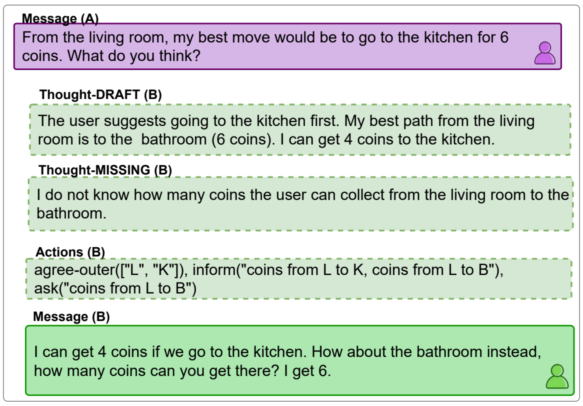


Figure 2: Player B's internal reasoning process (green dashed boxes) and output message, given Player A's message (purple box). Player B has the structure of the Baseline agent.

Action	Description
`suggest(x)`	Agent suggests x
`agree-inner(x)`	Agent agrees with own suggestion x
`agree-outer(x)`	Agent agrees with other agent's suggestion x
`reject(x)`	Agent rejects suggestion x
`ask(y)`	Ask other agent the questions y
`inform(y)`	Tell other agent the information (y)
`visit(v)`	Both agents agree to visit node v
`solve(x)`	`Thought-DRAFT` produced a successful solution (only Baseline and Coin-Tracking agents)
`end(x)`	The agents are ready to submit a solution (x = final path)

Table 1: The agents' action space. `x` denotes a list of nodes, `v` denotes a single node, and `y` denotes a list of strings in communicative actions.

the TSP, the rooms represent cities and the coins distances (Fig. 1), with the coin maximization analogous to distance minimization. The curse-breaking ritual follows the rules of the TSP, and the coin reward in the end measures the overall optimality of the selected path. Because `TRAVELING ADVENTURERS` inherits the NP-hardness of TSP, the optimal path for a sufficiently large graph will not be immediately obvious to humans even if they have perfect information about both players' edge weights. We therefore expect that `TRAVELING ADVENTURERS` promotes dynamic negotiations in which the agents propose subpaths and revise their proposals as they discover more optimal alternatives.

The game is symmetric, as both players are given the same amount and type of information: their own edge weights are fully known, and the other agent's edge weights are initially unknown. This means that no player can solve the task optimally by themselves, which generates a need for negotiation about the optimal path.

4 Game-Playing Agents

We introduce a series of agents for playing `TRAVELING ADVENTURERS`. We start with a simple baseline agent which performs a variant of few-shot chain-of-thought (CoT) reasoning (Wei et al., 2022) and then extend it with symbolic modules for partner coin tracking, state tracking, and problem solving. We evaluate the agents on self-play (Section 5) and on gameplay with human partners (Section 6).

4.1 The Agent Interface and Actions

In each turn of the game, the agent receives a natural-language message from the other agent and generates a natural-language response as output to be sent back to the other agent (Fig. 2). Each variant of the agent maintains some internal symbolic state (see Fig. 3), whose contents are added to an LLM prompt along with the input message.

Before generating the output message, the LLM in each agent may generate a sequence of *actions* from a fixed list of action types (see Table 1), as well as free-form *thoughts*. The actions fall into three broad categories: *negotiation*, *conversation*, and *state update* actions. Note that the actions and thoughts are part of an agent's internal reasoning process and are not shared with the other agent; their primary role is for the LLM to condition the output message on them.

The *negotiation actions* represent collaborative dialogue acts: `suggest`, `agree`, and `reject`. An agent chooses to propose a subpath through the `suggest` action; the other two represent the acceptance and refusal of such a proposal. We distinguish between *inner* agreement (of an agent with its own suggestion) and *outer* agreement (with the other agent's suggestion); this allows us to track whether a suggestion has been accepted by both agents. We do not make such a distinction for rejection, since both parties need to agree on a proposal to make it valid, whereas a single rejection invalidates it. The LLM is instructed through its prompt to always generate an `agree-inner` action when it generates a `suggest` action.

The *conversation actions* represent dialogue acts that facilitate the exchange of information between

	Baseline	Coin-Tracking	State-Tracking	Problem-Solving
Agent's Coins	✓	✓	✓	✓
Partner's Coins		✓	✓	✓
Action History	✓	✓		
Visited			✓	✓
Remaining			✓	✓
IBP				✓

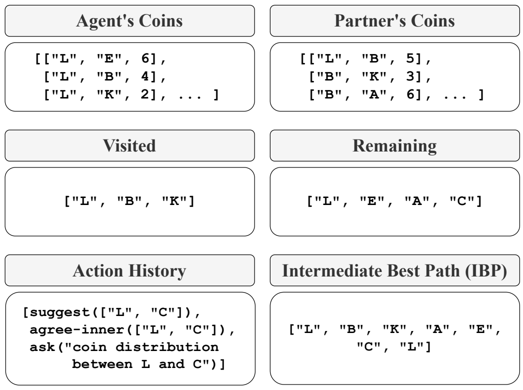


Figure 3: The table on the left shows the input structure for each version of the agent, with the rows representing the components and the columns showing agent versions. The figure on the right shows a real example of each component, where the letters correspond to graph nodes, i.e., rooms, while the numbers show the weights of the edges between them, i.e., coins.

the agents: `inform` and `ask`. Generating an `inform` action will typically cause the LLM to generate a statement encoding a piece of information as part of the output message, whereas `ask` generates questions. In contrast to the negotiation actions, the arguments of conversation actions are strings that will be spelled out as proper English by the LLM (see e.g. Fig. 2).

Finally, the *state update* actions directly affect and modify the agent's internal symbolic state. We will discuss these as we introduce the state representations of the individual agents below.

4.2 Baseline Agent

The Baseline agent prompts an LLM to generate actions, thoughts, and output messages using a modified `ReACT` prompting strategy (Yao et al., 2023). We present the LLM with a description of the task and environment, followed by a complete example dialogue between two players solving the task.

The agent prompts the LLM to generate *thoughts* of one of two possible types. In a `Thought-DRAFT`, it attempts to generate a solution to the game by proposing a full or partial path. The agent then either confirms the feasibility of this solution through the `solve` action explained below, or it generates a `Thought-MISSING`, which specifies that the agent needs more information (e.g., the other agent's coins) to determine a good path. A typical `Thought-MISSING` thought will then trigger an `ask` action to gather this information.

The LLM can also generate *actions* in addition to these thoughts. These include all the negotiation and conversation actions explained above. Furthermore, the `end` action finishes the game and submits a path; with this action, the agent claims that it has found the optimal path and agreed on it with the other agent. Finally, the `solve` action indicates that the agent considers the solution generated in `Thought-DRAFT` as successful, and that a `Thought-MISSING` is not needed.

A key difference between thoughts and actions is that the Baseline agent stores all generated actions in symbolic memory and includes this *action history* in the prompt in the next turn. The only symbolic information the Baseline agent uses in addition to the action history is a representation of its own world state (`Agent's Coins` in Fig. 3), i.e., the graph and the agent's own edge weights.

4.3 Coin-Tracking Agent

We observe several kinds of mistakes in the Baseline agent (see examples in Appendix D). The agent often **forgets the information** its partner (or it) has **already shared**. For example, in the agent's `Thought-DRAFT`, the agent might claim to have already shared its coin distribution when in fact it has not.

In order to improve the agent's ability to keep track of its partner's knowledge about their world, we extend the Baseline agent with additional symbolic memory that captures what the agent knows about the other agent's edge weights (`Partner's Coins`). The `Partner's Coins` structure gets updated progressively throughout the game after each message the partner sends. The necessary updates are extracted by an LLM-based module tasked with isolating the new information about the partner's edge weights. To do this, it processes each message with the additional context of the current `Partner's Coins`. To further constrain the module and aid with referential expression parsing, we include information about the world that the partner is in (*light*/*ghost*). This way, if a message contains claims about both the "ghost" and "light" coins, the module can disentangle it and identify which part of the utterance is relevant for the partner. The module employs CoT reasoning, first generating a single *thought*, followed by the new `Partner's Coins` information. The rest of the agent is identical to the Baseline (see second column in Fig. 3).

4.4 State-Tracking Agent

A frequent mistake that the Coin-Tracking agent makes is **forgetting which rooms** have yet to be visited or can be visited from a given node, resulting in incomplete paths. To address the inability to construct a path including all nodes, we introduce an agent augmented with state-tracking components: two dynamically updated structures keeping track of the agreed upon nodes (`Visited`) and the pool of remaining nodes (`Remaining`).

We replace the *action history* with the two state-tracking modules, which are updated whenever a `visit` action is generated (adding the agreed-upon node to `Visited`, and removing it from `Remaining`). This is in contrast to the previous two agents, where we preserve the actions in the memory by repeating a progressively longer list of generated actions. By executing the `visit` action, we reduce the input size, while still preserving the key negotiation actions in the memory. This addition enables us to augment the module for updating the `Partner's Coins` with the agent's last agreed-upon node, i.e., the last element in `Visited`, in order to aid the module in co-reference resolution.

Additionally, we observe that the Coin-Tracking agent rarely uses the `Thought-MISSING`, instead including its presumed contents at the end of the `Thought-DRAFT`. Thus, we collapse the thoughts (`Thought-MISSING` and `Thought-DRAFT`) into a single `Thought` and remove the `solve` action. The rest of the output format remains the same.

4.5 Problem-Solving Agent

The State-Tracking agent is quite competent at generating correct paths, but it still struggles to generate optimal paths. We therefore define a final agent which uses a symbolic linear program solver (Gurobi Optimization, LLC, 2024) to compute optimal paths, rather than having the LLM compute them itself.

At the start of each turn, the Problem-Solving agent converts the agent's knowledge into an Integer Linear Program (ILP) for the Traveling Salesman Problem. The ILP further enforces that the optimal path must start with the `Visited` path on which the two agents have agreed so far. Unknown edge weights are assigned a value of zero. The agent then calls the Gurobi solver to compute the optimal path that is consistent with this input. We call this optimal path the "intermediate best path" (`IBP`), emphasizing that the optimal path may change in response to the `Visited` list and the agent's knowledge about the partner's edge weights.

The agent then inserts the IBP into the prompt for the LLM call generating the thoughts and actions. Note that the observation that symbolic solvers outperform LLMs in solving NP-hard optimization problems is not new (Duchnowski et al., 2025); in contrast to this related work, however, the linear program in our agent is generated programmatically from the `Visited` list and the edge weights, rather than by an LLM.

5 Agent-Agent self-play

5.1 Setup

In order to assess each agent from Section 4, we test pairs of agents on `TRAVELING ADVENTURERS` in self-play. We use GPT-4o (`gpt-4o-2024-08-06`) as the LLM base of the agents. We evaluate agents using four random seeds on a batch of 25 games, producing 100 games per agent. Both agents in a pair instantiate the same agent structure. We refer to the agents as `BOT` and `USER`, and set the `USER` agent to always make the first move. Additionally, the `BOT` is always in the *ghost* world, and the `USER` is always in the *light* world. The agents take turns sending messages until the game ends with a solution or a timeout.

We manually created six 6-node graph pairs (see Appendix F for details). In each game, we randomly pick one of these graph pairs and assign them to the `USER` and `BOT`. The rooms included in the self-play setup are: a living room (abbreviation: L), a bathroom (B), a kitchen (K), an empty room (E), and a children's room (C). The first room is

Agent	Identical	Correct	Optimal
Baseline	99	71	28
Coin-Tracking	**100**	65	25
State-Tracking	99	86	17
Problem-Solving	98	**98**	**45**

Table 2: The results of the self-play experiments per agent. The rows represent the agents described in Section 4. The columns represent binary metrics described in Section 5.1. The figures illustrate the number of games in which each binary metric occurs, and are expressed in percent points, representing the mean across four seeds.

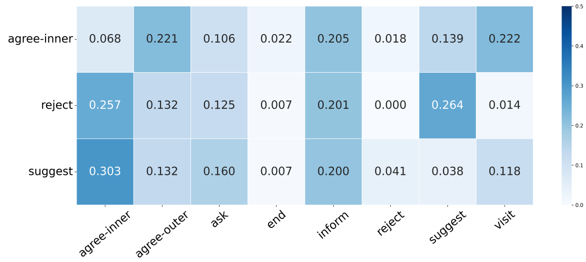


Figure 4: The abridged co-occurrence matrix illustrating the conditional probability of the actions on the y-axis co-occurring with actions on the x-axis ($P(x|y)$). The full matrix can be found in Appendix G.

always the living room.

The agents' goal is to agree on and submit a path that visits each room once and returns to the start. If the agents fail to converge on a solution within 15 turns each, the game is terminated and marked as a timeout. Once a path is submitted by both agents, we evaluate it in terms of three binary metrics: whether the agents submitted an **Identical** solution, whether the solution is **Correct**, i.e., whether it includes each room once and returns to the living room, and whether it is **Optimal**. Note that **Identical** $\leftarrow$ **Correct** $\leftarrow$ **Optimal**.

5.2 Results

Overview Table 2 breaks down the results by agent. Across all setups, most games ended with agents successfully producing identical solutions. Beyond that, the Problem-Solving agent decisively outperforms all other agents on correctness and optimality. Moreover, the Coin-Tracking agent trails the Baseline in both correctness and optimality. The State-Tracking agent achieves a high correctness score, but a much lower optimality score. This illustrates the distinct impacts of certain modules on scores, and trade-offs between the metrics, particularly optimality in favor of correctness.

Despite the constrained action space and limited number of possible moves at each node, the results suggest that the game environment creates a reasonably challenging task for the agent, even with the support of a symbolic problem-solving module.

Module impacts on scores We interpret the Coin-Tracking agent's dip in correctness and optimality relative to the Baseline as a result of the `Partner's Coins` module improving capacities that didn't have a direct effect on the two metrics.

The State-Tracking agent stores consistent partial solutions, which has a positive effect on correctness, but a negative effect on the optimality. This is in line with the expected functions of the added symbolic modules (`Visited` and `Remaining`): they are designed to aid the agent in keeping track of the selected subpath and nodes that remain to be selected. However, the modules do not contribute to forming an optimal path.

Lastly, we observe that the agent equipped with a symbolic problem-solving module is our best performing and most balanced version. The solver helps steer the model in the right direction (highest optimality and correctness) without negatively impacting conversation management and efficiency.

Storing actions The results further suggest that the way in which actions are stored in the agent's memory impacts correctness. Storing only the currently relevant actions (State-Tracking and Problem-Solving) improves the correctness, compared to simply including a list containing all generated actions up to the current turn without any pruning (Baseline and Coin-Tracking).

5.3 Collaboration analysis

To further assess how the agents collaborate, we conduct qualitative analyses by calculating co-occurrence probabilities across the defined action space. This allows us to get a better understanding of the contributing factors to the results in Table 2. Since these findings are consistent across agents and agents (`BOT` and `USER`), we focus on the Problem-Solving `USER` agent as an example.

Node-by-node strategy is preferred to full path generation Fig. 4 shows an abridged co-occurrence matrix of actions being generated in the same turn. Note that the `suggest` action typically occurs once per conversation turn, as indicated by a low probability of `suggest` occurring with itself (0.038). This would imply that the node-by-node approach is the favored strategy. The agents are not

Agent	Setup	Node-by-node	Full path
Baseline	SP	95.6	0.3
Coin-Tracking	SP	96.0	0.6
State-tracking	SP	99.6	0.0
Problem-solving	SP	96.2	0.3
Baseline	HA	65.5	23.1
Problem-solving	HA	91.6	4.2

Table 3: Collaboration strategy overview by agent and by setup (**SP**: BOT in self-play, **HA**: human-agent). The figures illustrate the percentage of total games using the two most prevalent strategies: **Node-by-node** and **Full path**. Full tables can be found in Appendix C.

explicitly instructed to generate actions in this manner, though this is the strategy used in the prompt example (Appendix E).

This observation is further supported by the length of argument passed to the `suggest` action.[2] Table 3 shows the percentage of total games using the two most prevalent strategies: **Node-by-node** (2 arguments) and **Full path** (7 arguments), further illustrating each agent's preference for the the node-by-node strategy, i.e., incremental greedy approach. We find that suggestions were overwhelmingly used to add only one node to the joint path, with the next most popular uses being suggesting 2 nodes or suggesting a full path.

Rejection is followed by a counterproposal Rejection is another important aspect of collaboration. While the agents rarely outright `rejected` proposals (only 0.1% of total actions), we notice that `rejecting` is typically done in tandem with a counter-proposal (`suggest` and `agree-inner` actions, see Fig. 4). This is desirable behavior, and future work could get the agent to reject more (where appropriate).

6 Human-Agent Experiments

We complement the self-play evaluation from Section 5 with an evaluation in a more natural environment: one in which the agent plays TRAVELING ADVENTURERS with a human. We test the Baseline and Problem-Solving agents.

6.1 Setup

The pairs complete one round of the game with a 6-node graph where the starting position is always fixed. The pair must communicate through a chat box and select an identical, agreed-upon path before clicking the **submit** button. If they try to submit a solution before both agents have agreed, the game framework will send both agents a notification informing that their selected paths must be identical. This was done to prevent participants from cheating by submitting a random sequence of nodes without any communication or agreement from their partner.

We prepend this round with a 4-node graph tutorial that the human player completes on their own. This is meant to familiarize the player with the mechanics and goals of the game, in tandem with the written instructions they read before accepting the task. The weights of the graph range from 1 to 10 and sum to a preset amount (see Appendix B for more details).

The interface for the human participants consists of the chat box and a visual representation of the rooms, as in Fig. 1.[3] The human needs to click on the rooms in the agreed-upon order and, once finished, they need to click the **submit** button.

We collected 50 games per agent tested (100 in total) by recruiting participants through Prolific (www.prolific.com), where they were paid £12/hour and an additional £0.5 bonus for finding an optimal solution. The task took on average 12 minutes to complete, and the participants were not informed that their partner was an LLM agent. We deployed the game using Slurk (Götze et al., 2022), an out-of-the-box dialogue collection tool with a robust logging schema.

In order to make the game more rewarding for the participants, we present them with a score between 0 and 100. We use the percentile rank of the solution as a representative proxy of the submitted path's quality in relation to all other possible solutions. For example, a solution very good solution might be better than 98% of possible solutions, its percentile rank is 98, and the pair would get a score of 98. For further evaluation, we retain the **Optimality** metric from Section 5.1. Because of the changes we make to circumvent cheating, the round can only end in a **Completed** submission if both the human and agent select an identical final path at the time of submission. For this reason, we track the games which achieve a suboptimal, yet high score (over 90) as an indicator of successful collaboration. If the agents fail to reach a final agreement or if the human player does not submit the solution, the game framework terminates and

[2]For example, if an agent suggests moving from the kitchen to the attic, the subpath will be `["K", "A"]`.

[3]For an overview of visuals, see Appendix A.

	Comp.	S≥90	Opt.	AMT	HMT
Baseline	**88**	26	10	10.1	8.2
Problem-solving	76	**32**	**20**	9.4	7.8

Table 4: The results of the human-agent experiments by agent version, using **Comp**leted, **S**core $\geq$ **90**, and **Opt**imal metrics, as described in Section 6.1 (these metrics are percentages of all collected games). **AMT** and **HMT** refer to the **m**ean number of **t**urns per game for the **a**gent and **h**uman participants, respectively.

the round ends in a timeout.

6.2 Results

Overview The results can be found in Table 4. While the number of completed games is higher with the Baseline agent, the number of games ending with an optimal or high score is higher in the Problem-Solving agent. This shows the agents' ability to navigate and complete a complex, symmetric collaborative task and highlights the Problem-Solving agent's ability to steer dialogue and suggest optimal or near-optimal solutions.

User error causes incomplete games The majority of terminated games stem from user errors where the participant did not click the submit button (even when a full identical path was selected) or communicated very little with the agent. This is a confusing outcome that should be further looked into in future work.

Node-by-node strategy is preferred We conduct the analysis about path strategy as we did for self-play. The bottom two rows of Table 3 show the distribution of the two most dominant strategies, as indicated by the length of argument passed to the `suggest` action, illustrating the number of steps: node-by-node (argument length 2) and full-path (argument length 7). This matches the results in Section 5.3. The Problem-Solving agent is more consistent, whereas the Baseline agent shows more variability in its approach. This can be attributed to the Baseline agent being more susceptible to suboptimal suggestions, as, compared to the Problem-Solving agent, it does not have an IBP to which it can compare the human's suggestion.

It takes more turns to reach a solution In contrast to self-play experiments, in which the interlocutors consistently take relatively few turns to reach an agreement ($n_t \leq |V|$), the human-agent experiment results in a mean 10.1 turns for the Baseline agent and 8.2 turns for the Problem-Solving agent. The human player turns are slightly lower (9.4 and 7.8, respectively) due to the reactive system design: the agent can take a turn if it receives a message from the human or the game framework.

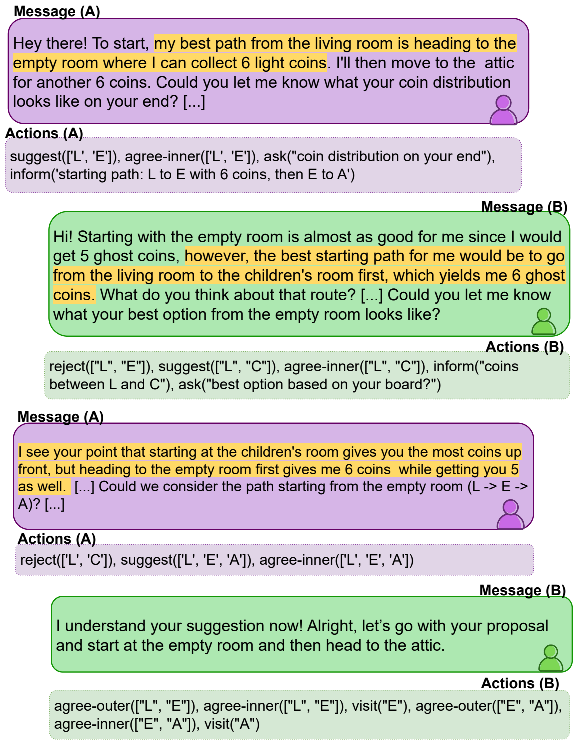


Figure 5: An example interaction of two Problem-Solving agents (A, B) in self-play. The dialogue illustrates a proposal rejection sequence, ultimately ending in a suggestion acceptance. We highlight the crucial parts of the dialogue.

7 Qualitative examples

We illustrate the abilities and limitations of our agents with a few qualitative examples. Fig. 5 shows a multi-turn negotiation example from self-play of the Problem-Solving agent, in which B (purple) rejects A (green)'s suggested path and offers its local optimal move as a counterproposal. In the subsequent turn, A manages to persuade B to reconsider their decision in favor of the joint optimum, which B accepts. This shows that the agent is capable of nontrivial negotiations about the path.

The most prominent remaining issue is that the agents with symbolic elements have no backtracking ability. This is illustrated in Fig. 6. Because of the current design of the symbolic state-tracking structures (`Visited`, `Remaining`) and the actions for updating them, the Problem-Solving agent (purple) cannot process the human's (blue) attempt to

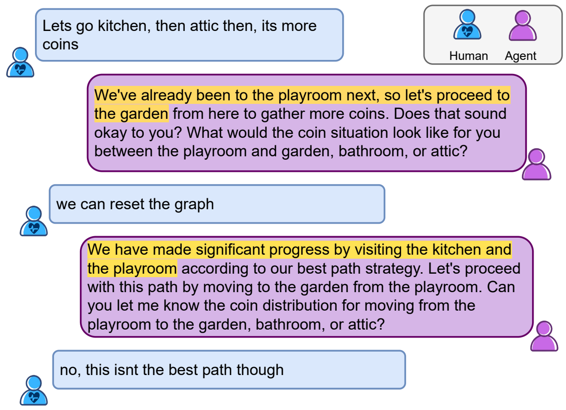


Figure 6: An example interaction between human (blue) and agent (purple). The dialogue illustrates the agent's inability to backtrack and revise the agreed-upon path. The agent has the structure of the Problem-Solving agent described in Section 4.

undo some of the recent steps in the path. The `Visited` data structure correctly represents a path that was already mutually agreed upon, but agents in an effective negotiation sometimes withdraw prior agreements in order to explore alternatives. Maintaining and updating an exact record of the agreed-upon path in such a flexible way remains a question for future research.

8 Conclusion and Future Work

We have presented `TRAVELING ADVENTURERS`, a novel two-player collaborative game environment based on the TSP, where players' individual optima might not be identical to one another, or the joint optimum, encouraging negotiation and collaboration. We presented an agent that plays the game and find that combining an LLM with neurosymbolic modules improves the performance compared to a Baseline in self-play. We also tested the agents with human participants and find that, while the performance is lower than in self-play, the neurosymbolic agent is able to foster successful cooperation through dialogue and negotiation, outperforming the Baseline agent.

In the future, it would be informative to see if the agent structure from Section 4 generalizes well to other problems, and whether it is sufficiently robust to handle larger graphs. Another possible avenue for future work is investigating the agents' social reasoning. For example, whether, when presented with a discrepancy between own, partner's and joint optima, the agent sticks to its own plan and tries to convince its partner, or vice versa.

Ethics Statement

We do not see any particular ethical challenges with the research reported here.

Acknowledgments

We thank Florian Kandra for his help with setting up the Slurk interface.

This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID KO 2916/3-1. We further acknowledge the stimulating research environment of the GRK 2853/1 "Neuroexplicit Models of Language, Vision, and Action", funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project number 471607914.

References

James F. Allen, Lenhart K. Schubert, George Ferguson, and 1 others. 1995. The TRAINS project: a case study in building a conversational planning agent. *JETAI*, 7(1):7–48.

Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. 2020. On the utility of learning about humans for human-ai coordination. *Preprint*, arXiv:1910.05789.

Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, and Xiuqiang He. 2024. Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects. *Preprint*, arXiv:2401.03428.

Allan Dafoe, Yoram Bachrach, Gillian Hadfield, Eric Horvitz, Kate Larson, and Thore Graepel. 2021. Cooperative AI: machines must learn to find common ground. *Nature*, 593(7857):33–36.

Alex Duchnowski, Ellie Pavlick, and Alexander Koller. 2025. A Knapsack by Any Other Name: Presentation impacts LLM performance on NP-hard problems. *Preprint*, arXiv:2502.13776.

Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. 2024. NPHardEval: Dynamic benchmark on reasoning ability of large language models via complexity classes. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4092–4114, Bangkok, Thailand. Association for Computational Linguistics.

Kallirroi Georgila and David Traum. 2011. Learning Culture-Specific Dialogue Models from Non Culture-Specific Data. In *International Conference on Universal Access in Human-Computer Interaction*, pages 440–449. Springer.

Ghost Town Games. 2016. Overcooked. https://www.ghosttowngames.com/overcooked. Video game developed by Ghost Town Games, published by Team17.

Jana Götze, Maike Paetzel-Prüsmann, Wencke Liermann, Tim Diekmann, and David Schlangen. 2022. The slurk Interaction Server Framework: Better Data for Better Dialog Models. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 4069–4078, Marseille, France. European Language Resources Association.

Atharva Gundawar, Mudit Verma, Lin Guan, Karthik Valmeekam, Siddhant Bhambri, and Subbarao Kambhampati. 2024. Robust planning with llm-modulo framework: Case study in travel planning. *Preprint*, arXiv:2405.20625.

Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual.

Mareike Hartmann and Alexander Koller. 2024. A Survey on Complex Tasks for Goal-Directed Interactive Agents. *Preprint*, arXiv:2409.18538.

Mercedes Hidalgo-Herrero, Pablo Rabanal, Ismael Rodriguez, and Fernando Rubio. 2013. Comparing Problem Solving Strategies for NP-hard Optimization Problems. *Fundamenta Informaticae*, 124:1–25.

Isidora Jeknic, David Schlangen, and Alexander Koller. 2024. A Dialogue Game for Eliciting Balanced Collaboration. In *Proceedings of the 25th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 477–489, Kyoto, Japan. Association for Computational Linguistics.

Muhammad Asif Khan and Layth Hamad. 2024. On the capability of LLMs in combinatorial optimization. *Authorea Preprints*.

Jin-Hwa Kim, Nikita Kitaev, Xinlei Chen, Marcus Rohrbach, Byoung-Tak Zhang, Yuandong Tian, Dhruv Batra, and Devi Parikh. 2019. CoDraw: Collaborative Drawing as a Testbed for Grounded Goal-driven Communication. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6495–6513, Florence, Italy. Association for Computational Linguistics.

Mike Lewis, Denis Yarats, Yann Dauphin, Devi Parikh, and Dhruv Batra. 2017. Deal or No Deal? End-to-End Learning of Negotiation Dialogues. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2443–2453, Copenhagen, Denmark. Association for Computational Linguistics.

Jessy Lin, Nicholas Tomlin, Jacob Andreas, and Jason Eisner. 2024. Decision-Oriented Dialogue for Human-AI Collaboration. *Transactions of the Association for Computational Linguistics*, 12:892–911.

Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented Language Models: a Survey. *Transactions on Machine Learning Research*. Survey Certification.

Rindranirina Ramamonjison, Timothy T. Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. 2023. NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. *Preprint*, arXiv:2303.08233.

David Schlangen. 2019. Grounded Agreement Games: Emphasizing Conversational Grounding in Visual Dialogue Settings. *Preprint*, arXiv:1908.11279.

Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. 2025. GraphArena: Evaluating and Exploring Large Language Models on Graph Computation. *Preprint*, arXiv:2407.00379.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

Sina Zarrieß, Julian Hough, Casey Kennington, Ramesh Manuvinakurike, David DeVault, Raquel Fernández, and David Schlangen. 2016. PentoRef: A Corpus of Spoken References in Task-oriented Dialogues. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 125–131, Portorož, Slovenia. European Language Resources Association (ELRA).

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. 2024. SOTOPIA: Interactive Evaluation for Social Intelligence in Language Agents. *Preprint*, arXiv:2310.11667.

A Visuals

Fig. 7 illustrates the visual representations of all rooms in the human-agent experiment setup. These were displayed to the human player as graphs as seen in Fig. 1. The pixel art room images were obtained from Google Images.

Figure 7: An overview of rooms shown in the human-agent experiment setup.

B Graph generation details

We generate the graphs in the human-agent experiment setup using a process aiming to maintain weight consistency. All weights range from 1 to 10, and we make sure the weights sum to a previously calculated total W_{total}. In order to obtain this figure, we sum the minimum ($w_{\text{min}} = 1$) and maximum ($w_{\text{max}} = 10$) possible weight, multiply it by the number of edges $|E|$ and divide it by 2, which balances the different graphs that get generated. In the 6-node case, this amounts to a sum of 82.

To assign weights, we compute a target mean for edge e_{ij}, $\mu_{e_{ij}}$. The $\mu_{e_{ij}}$ is defined as

$$\mu_{e_{ij}} = \left\lfloor \frac{W_{\text{total}} - W_{\text{current}}}{E_{\text{remaining}}} \right\rfloor,$$

whereby $w_{\text{min}} \leq \mu_{e_{ij}} \leq w_{\text{max}}$. We also compute a target deviation, $\text{dev} = \min(w_{\text{max}} - \mu_{e_{ij}}, \mu_{e_{ij}} - w_{\text{min}})$, i.e., the allowed distance from the target weight while still staying within bounds, constraining how random the weights can be.

We use this information to compute a dynamic, edge-specific interval, assuring that the value we assign still leaves room to generate the remaining weights below the allowed threshold. Lastly, we randomly sample an integer within the dynamically computed bounds and assign it to the given edge.

C Number of arguments to `suggest` action

Tables 5 and 6 show the distribution of number of arguments passed to the `suggest` action, indicating the number of nodes in a proposed path.

D Agent errors

Fig. 8 shows each error listed in Section 4.2, as well as an accompanying example.

E Prompts

Here we illustrate the structure of the system prompts we use for each agent presented in Section 4. Fig. 9 shows the general outline, with the task description, which is identical across all agents, and placeholders for agent-specific elements depicting the input and output components. These are expanded in Fig. 10 and Fig. 11, respectively. Lastly, Fig. 12 shows the actions breakdown, including agent-specific rules. Figures 13 and 14 depict examples used for each of the four agents. The examples represent entire instances of conversation, but have been shortened in the figures for the purposes of clearer visual representation.

F Pre-set boards

Table 7 illustrates the six preset boards setups that we use during self-play experiments, highlighting the difference between weights assigned to identical edges for the two players, `USER` and `BOT`.

G Full co-occurrence matrix

Fig. 15 illustrates the complete co-occurrence matrix depicting the probabilities of each generated action along the y-axis occurring with each action along the x-axis. Values are calculated for the Problem-Solving `USER` agent.

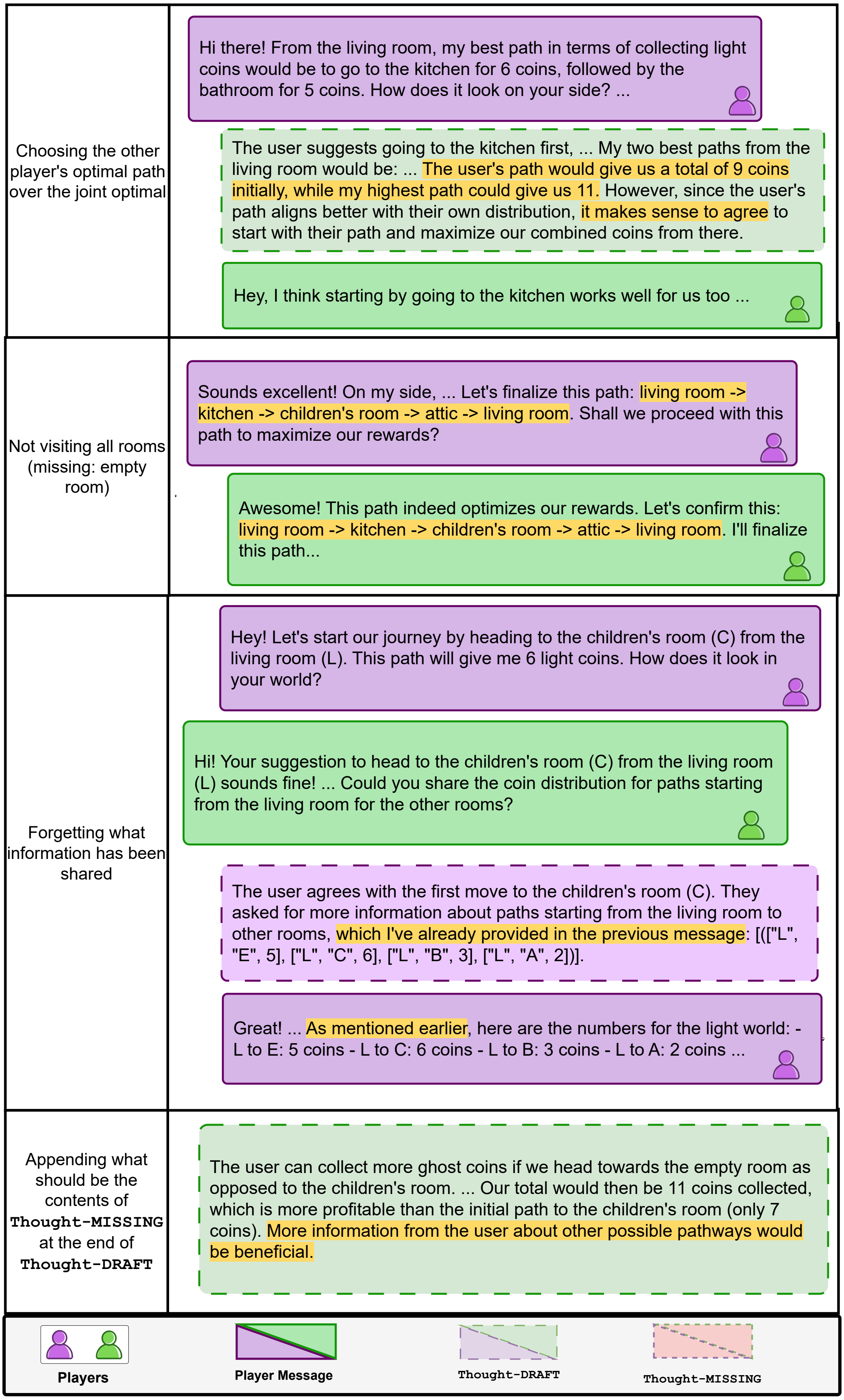


Figure 8: An overview of error examples found in the Baseline and Coin-Tracking agents, as described in Section 4.

```
You are playing a game with a partner where you each control a
magical character. In the game, an evil wizard separated you and
your partner into two different dimensions: both of you find
yourselves in the same house, but you are in the ghost world, while
your partner is in the light world. You are able to communicate
through written text only. The scene for both of you consists of 4
rooms all connected to each other by hallways in which you will find
various amounts of coins. In order to break the curse and escape,
you and your partner have to visit each of the 4 rooms once and
return to the room where you started. For this to work, you must
travel in parallel and visit the same rooms at the same time. On
your way, you are both individually collecting coins: you collect
ghost coins, and your partner collects light coins. Because of the
wizard's curse, the ghost and light coins only show up if you're
both simultaneously going through the same hallway, so make sure to
coordinate your paths. Your score is the sum of both your and your
partner's collected coins - you must work together to maximize this.
Once you've escaped the split dimension, you and your partner will
each be rewarded the full sum of your total coin collection. Your
goals are to escape and collect as big of a reward as possible. Chat
with your partner to decide on the best way to proceed.
```

You will get an input containing the following:
[World-state-own] - the state of your own world, including the coin distribution
[Observation] - the Message that the user sent.
Other agent-specific input components description

You will generate an output containing different kinds of thought, actions and a message to be sent to the user. The elements of your reply will be indicated by the flag in square brackets ([Flag]).

Agent specific reasoning step description

Action description

[Message] - each output must conclude with a [Message] to be shown to the user; it should be adapted to the information generated under [Thought]s and [Action].
Make sure to ALWAYS format the output in this exact way.

Agent-specific example

Figure 9: The structure of the base system prompt used for all agents described in Section 4.

Baseline

[History] - what Actions you decided to take previously. This should inform your choice of output.
[Observation] - the Message that the user sent.

Coin-tracking

[World-state-user] - the most recent state of the other user's world, as reported by the user
[History] - what Actions you decided to take in the previous turns. This should inform your choice of output.

State-tracking

[World-state-user] - the most recent state of the other user's world, as reported by the user.
[Visited] - a list of rooms that you and your partner have agreed upon.
[Remaining] - a list of rooms that you and your partner have yet to visit.

Problem-solving

[World-state-user] - the most recent state of the other user's world, as reported by the user.
[Visited] - current subpath; a list of rooms that have been agreed upon.
[Remaining] - a list of rooms that you and your partner have yet to visit.
[IBP] - intermediate best path starting with the subpath in [Visited]; the current best path given the known information about your board and the user's board. Note that the path might change when you learn new information about the user's coin distribution. You should ALWAYS use this as a reference point when making a suggestion.

Figure 10: An overview of agent-specific inputs included in the base system prompt from Fig. 9.

	2 (node-by-node)	3	4	5	6	7 (full path)	8
Baseline	**65.5**	4.5	1.7	0.3	4.5	*23.1*	0.3
Problem-Solving	**91.6**	2.6	0.6	0.3	0.6	*4.2*	0.0

Table 5: Distribution of the number of arguments to the suggest action per agent per version in the human-agent setup. Figures expressed in percentages. Bold figures indicate highest percentage, whereas italic figures indicate second highest percentage.

		2 (node-by-node)	3	4	5	6	7 (full path)
Baseline	USER	**95.8**	*2.1*	0.0	0.3	0.3	1.4
	BOT	**95.6**	3.4	0.5	0.3	0.0	0.3
Coin-Tracking	USER	**88.5**	*6.0*	0.9	0.0	0.0	4.6
	BOT	**96.0**	*2.9*	0.0	0.3	0.0	0.6
State-Tracking	USER	**91.3**	*5.4*	1.2	0.4	0.0	1.7
	BOT	**99.6**	0.4	0.0	0.0	0.0	0.0
Problem-Solving	USER	**73.5**	4.0	0.4	1.8	1.8	*18.4*
	BOT	**96.2**	*1.9*	0.5	0.5	0.5	0.3

Table 6: Distribution of the arguments (columns) to the suggest action per agent per version in self-play. Figures expressed in percentages. Bold figures indicate highest percentage, whereas italic figures indicate second highest percentage.

Baseline & Coin-tracking

```
The following are allowed kinds of thought:
- [Thought-DRAFT] - This flag signifies part of the output in which you
try to find the best path with the information you are given, while
making sure to visit all rooms only once and return to the starting
room. You should always start with this thought. If the [Thought-DRAFT]
is successful, you should indicate this by generating a solve(x)
Action, whereby x is a Python list of strings referring to the rooms in
the order to be visited. If the [Thought-DRAFT] was not successful,
meaning that a path that corresponds to the constraints has not been
generated, you will generate another thought: [Thought-MISSING].
- [Thought-MISSING] - If you could not find a valid path in [Thought-
DRAFT], you will generate this kind of thought in which you identify
why you couldn't solve the puzzle in the previous step.
```

State-tracking & Problem-solving

```
- [Thought] - This flag signifies part of the output in which you try
to find the best path with the information you are given (your own coin
distribution, given to you in [World-state-own], and the user's
currently known distribution, given in [World-state-user]), while
making sure to visit all rooms only once and return to the starting
room (check that against [Remaining]). You should always start with
this thought.
```

Figure 11: An overview of agent-specific inputs included in the base system prompt from Fig. 9.

```
- [Action] - Under this flag, generate the actions you are taking. Here is a
list of allowed actions and their descriptions:
  * suggest(x) - indicates your suggestion of a partial or complete path;
whereby x is an executable python list of strings referring to the rooms in
the order to be visited.
  * agree-inner(x) - indicates your agreement with the proposed partial path
x, whereby x is an executable Python lists of strings referring to the rooms
in the order to be visited.
  * agree-outer(x) - indicates the other player's agreement with the proposed
partial path x, displayed in [Observation], whereby x is an executable Python
lists of strings referring to the rooms in the order to be visited.
  * reject(x) - indicates that path (x) has been rejected either in the
generated [Thought] or by the other user in [Observation]; x is an executable
Python list of strings referring to the rooms in the order to be visited.
  * visit(x) - indicates that both you and the other player agree to visit
room x next; x is a single-letter string, corresponding to the room to be
visited.
  * ask(x) - generate this action to indicate what information to seek from
the user; x is a string.
  * inform(x) - generate this action to indicate what information about your
own world you are sharing with the user; x is a string.
  * end(x) - generate this action when you are ready to end the game and
submit the final solution (x), whereby x is a Python list of strings (of
length 1) referring to the rooms in the order to be visited. This action must
be included in order to end the game.
Here are the most important rules to keep in mind when generating these
actions:
  - "suggest(x)" must be accompanied by an "agree-inner(x)" action
  - "agree-inner(x)" should never be generated if you have previously agreed
on path x
  - "visit(x)" should only be generated in you generate an "agree-inner(x)"
with an "agree-outer(x)" OR when you generate an "agree-inner(x)" to a
previous user suggestion OR if you generate an "agree-outer(x)" to your
previous suggestion
  - "inform(x)" should always be based on the coin distribution in [World-
state-own] (when asked for coin distribution)
```

Agent-specific rules

State-tracking

```
  - DO NOT generate a visit(x) action if x is in [Visited]
```

Problem-solving

```
  - "ask(x)" and "suggest(x)" actions should always be generated based on the
best path given in [IBP]
  - DO NOT generate a visit(x) action if x is in [Visited]
  - "inform(x)" should always be based on the coin distribution in [World-
state-own] (when asked for coin distribution) or [IBP] (when asked for best
path from node)
```

Figure 12: An overview of the actions description included in the base system prompt from Fig. 9.

A.

Here is an example conversation:

[...]

[World-state-own] [["L", "E", 6], ["L", "B", 4], ["L", "K", 2], ["L", "C", 1], ["L", "A", 5], ["E", "B", 3], ["E", "K", 1], ["E", "C", 2], ["E", "A", 6], ["B", "K", 5], ["B", "C", 4], ["B", "A", 3], ["K", "C", 6], ["K", "A", 2], ["C", "A", 1]]
[History] [suggest(["L", "E"]), agree-inner(["L", "E"]), inform("coins between L and B, coins between L and A, coins between L and K, coins between L and C"), ask("coins between L and E, coins between L and B, coins between L and K, coins between L and A, coins between L and C"), [...] agree-inner(["A", "B"]), suggest(["B", "K"]), agree-inner(["B", "K"]), inform("coins between E and K"), ask("coins between A and B, coins between B and K")]
[Observation] that would be perfect for me because i can get 6 light coins between the bathroom and attic! from the bathroom, i can get 2 coins to the kitchen. From the kitchen, we would go to the children's room and then back to the living room, right? The first half of that path would give me 3 more light coins, and the second half 6 more light coins.

[Thought-DRAFT]: The user agrees with my suggestion and says they can get 6 light coins going to the bathroom. The user agrees to then go into the kitchen, where they can collect 2 coins, followed by the children's room, with 3 more light coins, and finally back to the living room, to collect 6 more light coins.
[Action]: agree-outer(["A", "B"]), visit("B"), agree-outer(["B", "K"]), visit("K"), agree-outer(["K", "C"]), agree-outer(["C", "L"]), agree-inner(["K", "C"]), visit("C"), agree-inner(["C", "L"]), visit("C"), solve(["L", "E", "A", "B", "K", "C", "L"]), end(["L", "E", "A", "B", "K", "C", "L"])
[Message]: Okay great, that works well! Then it's settled and we are done - the path we decided on is living room --> empty room --> attic --> bathroom --> kitchen --> children's room --> living room. If my calculations are correct, we should be getting 52 coins each as our reward, which seems pretty decent!

B.

Here is an example conversation:

[World-state-own]: [["L", "E", 5], ["L", "B", 3], ["L", "K", 1], ["L", "C", 6], ["L", "A", 2], ["E", "B", 4], ["E", "K", 6], ["E", "C", 1], ["E", "A", 3], ["B", "K", 2], ["B", "C", 5], ["B", "A", 6], ["K", "C", 3], ["K", "A", 4], ["C", "A", 1]]
[World-state-user]: [["L", "E", 6], ["L", "A", 5], ["L", "B", 4]]
[History]: []
[Observation]: Hello there! Let’s get started! It seems that from the living room, it would be best if I went to the empty room as I could get 6 ghost coins. The other paths that have a similar amount of coins would be to the attic, where I could get 5 ghost coins, or to the bathroom, where I'd collect 4 ghost coins. What do you think of this? How many coins could you collect in this way?

[Thought-DRAFT]: This is the start of the game. I know the light coin distribution. The user suggests going to the empty room, to get 6 ghost coins. This would be good in the light world as well, because I would collect 5 light coins, for a total of 11 coins. My best move from the living room would be through the children's room, but the user didn't suggest that.
[Action]: agree-outer(["L", "E"]), agree-inner(["L", "E"]), visit("E"), inform("coins between L and E, coins between L and C, coins between L and B, coins between L and A")
[Message]: Oh, I see! Okay, that would be quite good for me as well because going to the empty room gives me 5 light coins. I would get the most coins if we go to the children's room, i can get 6 light coins there. Regarding the bathroom, I can only get 3 light coins, and to the attic I can get only 2 light coins. Thus, if we go to the empty room, our total would be 11 coins.

[...]

Figure 13: An abridged overview of the examples included in the Baseline (A.) and Coin-Tracking (B.) agents’ base system prompts from Fig. 9.

```
Here is an example conversation:                                          C.

## [ … ]

[World-state-own]: [["L", "E", 6], ["L", "B", 4], ["L", "K", 2], ["L", "C",
1], ["L", "A", 5], ["E", "B", 3], ["E", "K", 1], ["E", "C", 2], ["E", "A", 6],
["B", "K", 5], ["B", "C", 4], ["B", "A", 3], ["K", "C", 6], ["K", "A", 2],
["C", "A", 1]]
[World-state-user]: [["L", "E", 5], ["L", "B", 3], ["L", "C", 6], ["L", "A",
2], ["E", "A", 3], ["E", "K", 6], ["B", "A", 6], ["B", "E", 4], ["B", "K", 2],
["B", "C", 5], ["C", "K", 3], ["K", "L", 1]]
[Visited]: ["L", "E", "A", "B", "C"]
[Remaining]: ["L", "K"]
[Observation]: cool yep that works - i get 3 coins to the kitchen and then 1
coin back to the living room.

[Thought]: The user agrees with my suggestion and says they can get 3 light
coins going to the kitchen and then 1 coin going to the living room. Thus, we
have agreed to visit K and then L.  This means that we are done and can submit
our solution - our final path is ["L", "E", "A", "B", "C", "K", "L"].
[Action]: agree-outer(["C", "K"]), visit("K"), agree-outer(["K", "L"]),
visit(["L"]), end(["L", "E", "A", "B", "C", "K", "L"])
[Message]: Cool, then it's settled and we are done - the path we decided on is
living room --> empty room --> attic --> bathroom --> children's room -->
kitchen --> living room.
```

```
Here is an example conversation:                                          D.

[World-state-own]: [["L", "E", 6], ["L", "B", 4], ["L", "K", 2], ["L", "C", 1], ["L", "A",
5], ["E", "B", 3], ["E", "K", 1], ["E", "C", 2], ["E", "A", 6], ["B", "K", 5], ["B", "C",
4], ["B", "A", 3], ["K", "C", 6], ["K", "A", 2], ["C", "A", 1]]
[World-state-user]: []
[Remaining]: ["L", "E", "K", "B", "C", "A"]
[Visited]: ["L"]
[IBP]: ["L", "E", "A", "K", "C", "B", "L"]
[Observation]: hi there! what should we do?

[Thought]: This is the start of the game. I know the ghost coin distribution and my
current best path. According to my [IBP], I should start by going from the living room to
the empty room. This way I will get 6 coins. I will suggest this (suggest, agree-
inner(["L", "E"])) to the user. I don't know anything about the user's coin distribution
or best path, so I should ask. I will also declare the two alternative paths that have a
large amount of coins: going to the attic (where I get 5 coins), and going to the bathroom
(where I get 4 coins).
[Action]: suggest(["L", "E"]), agree-inner(["L", "E"]), inform("coins between L and E,
coins between L and A, coins between L and B"), ask("coins between L and E, best next step
from L")
[Message]: Hello there! Let’s get started! From the living room, it would be best if I
went to the empty room, as I could get 6 ghost coins. What do you think of this? How many
coins could you collect in this way, and what would be your best starting move?
Alternatively, going to the attic yields 5 ghost coins and going to the bathroom 4 ghost
goins.

## [ … ]
```

Figure 14: An abridged overview of the examples included in the State-Tracking (C.) and Problem-Solving (D.) agents base system prompts from Fig. 9.

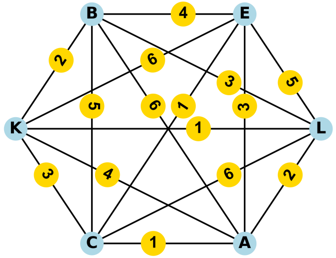
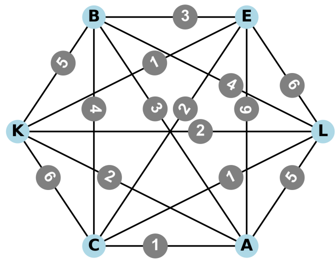
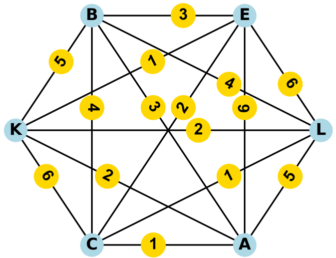
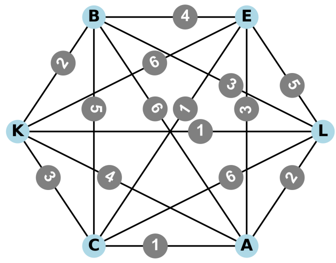
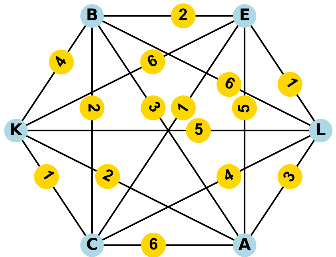
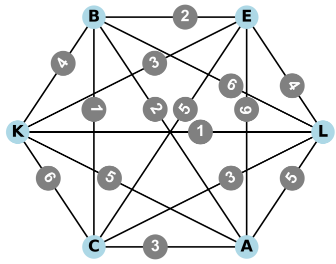
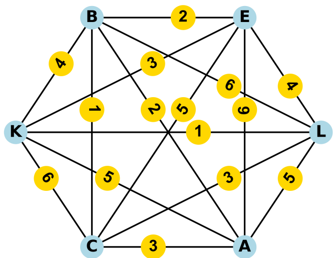
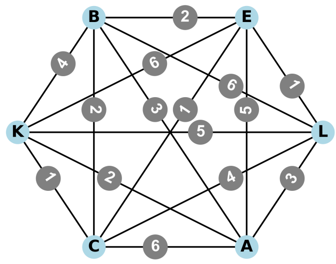
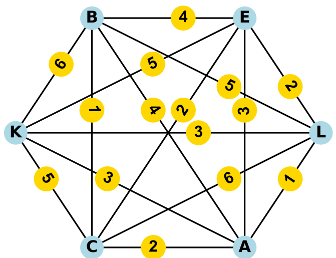
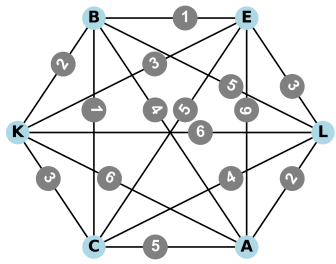
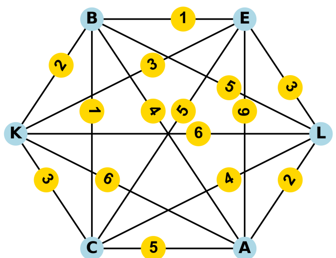
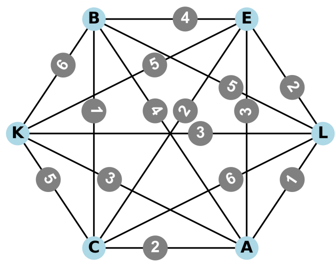
	USER	BOT
Board Setup 1		
Board Setup 2		
Board Setup 3		
Board Setup 4		
Board Setup 5		
Board Setup 6		

Table 7: The six board setups used in the experiments.

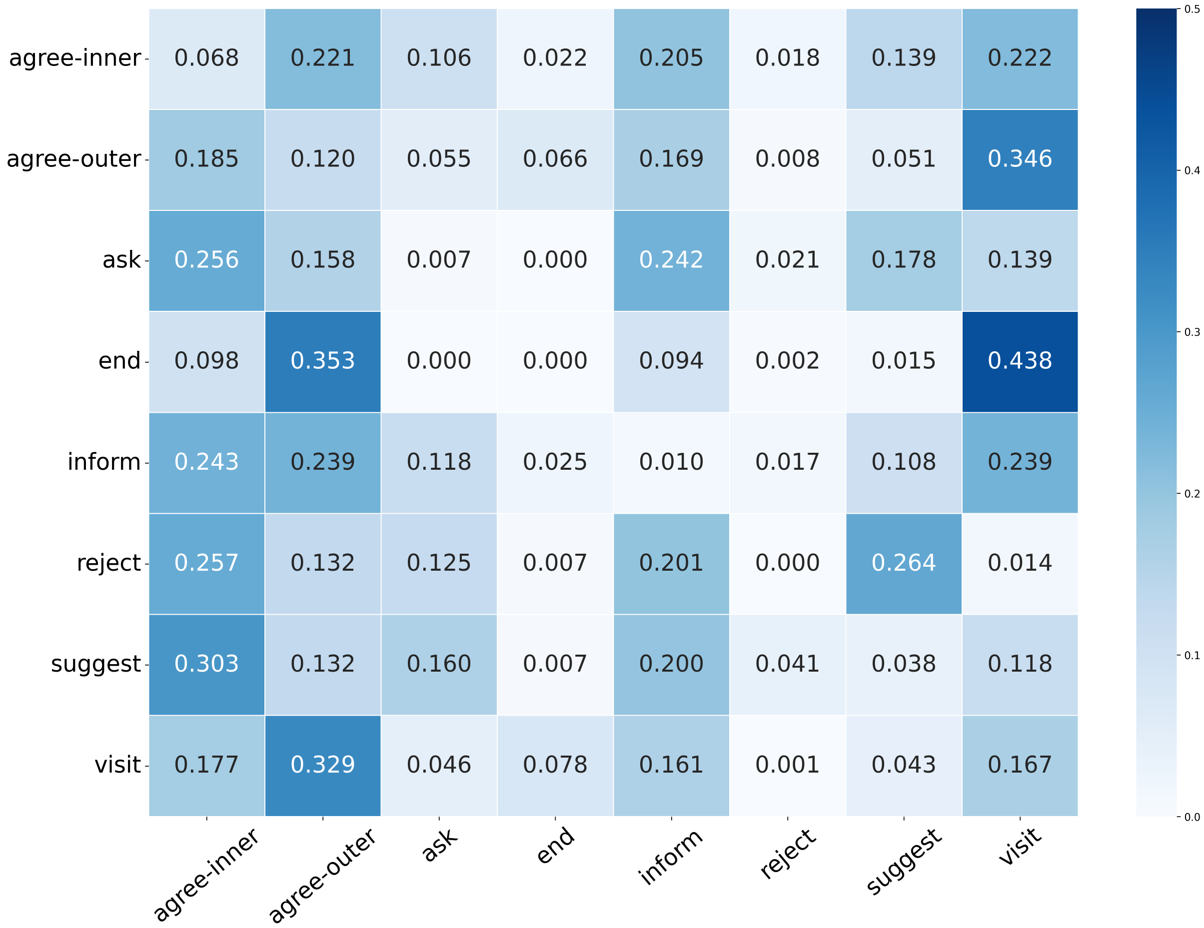


Figure 15: The full co-occurrence matrix illustrating the probabilities of each action along the y axis occurring with each action along the x axis ($P(x|y)$); Problem-Solving USER agent.