

Self-Improvement of AIWolf Agents Based on Self-Play Logs

Taku Umeda¹, Takao Endo¹, Jumpei Hato¹, Kotaro Otomura¹

¹Mitsubishi Electric Corporation

Corresponding authors:

Umeda.Taku@df.MitsubishiElectric.co.jp

Endo.Takao@cj.MitsubishiElectric.co.jp

Abstract

In this study, we propose a method for self-improving LLM agents based on self-play logs for the Werewolf Game, which requires dialogue, reasoning, deception, and cooperation under imperfect information. In our proposed method, an LLM analyzes game logs, evaluates the agents' behavior according to role-specific checklists, and then updates their prompts and processing logic. In addition to self-improvement based on fixed evaluation checklists, we introduce curriculum-based self-improvement, where the evaluation checklists are updated according to the improvement stage. Results show that in a 5-player village environment, the overall win rate of the Baseline Agent was 46%, while the LLM Self-Improved Agent achieved 51%, and the Curriculum-based LLM Self-Improved Agent achieved 54%. In particular, large improvements were observed for the Seer and Possessed roles, suggesting that self-improvement based on game logs is effective for improving the performance of roles that involve information sharing and vote steering. On the other hand, for the Werewolf role, the win rates after improvement were lower than the Baseline Agent's, suggesting that the lack of priority among checklist items based on their contribution to win rates may have contributed to a focus on easily achievable items rather than on proactive utterance strategies directly related to avoiding execution.

1 Introduction

Large Language Models (LLMs) show great capabilities in natural language understanding, reasoning, planning, and dialogue generation. In recent years, they have been used as core modules of AI agents that autonomously make decisions and take actions (Xi et al., 2025; Luo et al., 2025). To evaluate these LLM agents' capabilities, we need to examine them not only in static tasks but also

in a dynamic environment in which the conditions change over time (Yehudai et al., 2026).

The Werewolf Game is a useful task for evaluating the reasoning and dialogue abilities of LLM agents in dynamic environments (Kano et al., 2025). Each player needs to act strategically to win while estimating the hidden roles of other players from dialogue histories and action histories. Therefore, in the Werewolf Game, we can examine the abilities of LLM agents in imperfect information reasoning, natural language dialogue, persuasion, deception, cooperation, and long-term strategic planning.

Previously, rule-based strategies and manually designed prompts have been used for the Werewolf Game (Kano et al., 2025). However, effective strategies strongly depend on the game progress, voting situation, dialogue histories, role composition, and their own team. Thus it is difficult to comprehensively design them by hand. In particular, as the number of roles and players increases, it becomes more difficult to design appropriate reasoning, utterance, and voting strategies for each role. In fact, although various LLM-based agent methods have been proposed so far, problems such as inconsistency between utterances and actions, and repetition of the same content in utterances, have still been reported.

(1) We propose a self-improvement method for Werewolf Game agents, in which an LLM analyzes game logs, evaluates the role-specific behaviors according to role-specific checklists, and updates the prompts and processing logic to overcome the above situation. This framework can be regarded as a self-evolving agent framework, in which an LLM agent improves its own system based on its past behaviors. Such approaches have attracted increasing attention in recent years (Zhang et al., 2025; Gao et al., 2026).

(2) We introduce a curriculum-based self-improvement method, where the evaluation check-

lists are progressively updated according to the improvement stage. Evolving the evaluation checklists is one of the unique points in this study since the previous self-evolving framework usually consists of the fixed evaluation criteria.

(3) Through experiments in a 5-player village setting, we show that the proposed methods improve the overall win rate, particularly for the Seer and Possessed roles. We also analyze the case where self-improvement degraded the Werewolf role’s performance. Our analysis suggests the bias toward prohibitive instructions and the lack of win-rate-aware prioritization in checklist items as possible causes.

2 Werewolf Game

The Werewolf Game is a social deduction game in which players are divided into the Villager team and the Werewolf team. The Villager team tries to execute werewolves, whereas the Werewolf team tries to reduce the number of Villagers, while inferring the roles of other players. In this study, we consider a 5-player village consisting of two Villagers, one Seer, one Possessed, and one Werewolf. The Villager team consists of the Villagers and the Seer, while the Werewolf team consists of the Possessed and the Werewolf.

The game consists of a day phase and a night phase. During the day phase, players exchange information and reason through utterances. During the night phase, players vote to decide which player to execute, and then role-specific actions, such as divination by the Seer and attack by the Werewolf, are processed. The Villager team wins if they execute the Werewolf, while the Werewolf team wins if the number of players on the Villager team is equal to that on the Werewolf team.

In this way, players in the Werewolf Game need to make decisions based on hidden role information, dialogue histories, voting behavior, and the results of role-specific actions. Therefore, under imperfect information, dialogue, reasoning, and action selection have a significant impact on the outcome of the game.

In our experiments, we use the game server provided by the AIWolf Project¹ to play the Werewolf Game with LLM agents. The server handles the progression of the day and night phases and the processing of utterances, voting, divination, and attacks according to its specifications.

Table 1: Initial Checklist for Werewolf

No.	Checklist Item	Difficulty
1	Did the agent make utterances pretending to be a Villager without revealing its role during the daytime discussion?	Basic
2	Did the agent select an attack target every night?	Basic
3	Did the agent vote in a way that was not disadvantageous to the Werewolf team, such as voting for a Villager candidate?	Basic
4	Did the agent make utterances that induced suspicion toward the Seer or ability-user candidates?	Intermediate
5	Did the agent make persuasive utterances to dispel suspicion toward itself?	Intermediate
6	Was the selection of the attack target strategic, such as prioritizing important roles such as the Seer?	Advanced

3 Self-Improvement by LLMs

In this study, we propose a self-improvement method that updates the prompts and processing logic of AIWolf agents using game logs. Specifically, we focus on components involved in role-specific decision-making, such as utterance generation, voting, divination, and attack. After each game, an LLM analyzes the game log and updates the prompts and processing logic of the agents for the next game.

3.1 Game Logs

The game logs used for the updates are provided by the game server. The logs record information such as the role of each agent, survival status, dialogue history, voting results, divination results, attack results, and the win result. Based on this information, the LLM summarizes the game progression and the actions taken by agents in each role, identifies the winning team, and evaluates their behaviors according to the evaluation checklist described later.

3.2 Evaluation Checklist

To evaluate the agents’ behavior, we use role-specific checklists. The checklists were generated by Claude Opus 4.6, using the game flow of the 5-player village and the win conditions of each role as input. The target roles are Villager, Seer, Werewolf, and Possessed. Each item in a checklist is classified into three levels: basic, intermediate, and advanced.

¹<https://github.com/aiwolf-dial/aiwolf-nlp-server>

An example of the checklist for the Werewolf is shown in Table 1. The checklist consists of six evaluation criteria: three basic-level criteria, two intermediate-level criteria, and one advanced-level criterion. At the basic level, for example, a criterion describes whether the agent pretended to be a Villager without revealing its role. At the intermediate level, the criteria cover more strategic behaviors, including inducing suspicion toward the Seer and dispelling suspicion toward itself. At the advanced level, the criterion describes strategic attack target selection, prioritizing important roles such as the Seer.

In the current implementation, even though we assigned the difficulty levels, all checklist items are treated equally, and their direct contribution to the win rate is not explicitly estimated. The checklists for roles other than Werewolf are shown in Tables 7–9 in the Appendix.

3.3 LLM Self-Improved Agent

In the proposed LLM-based self-improvement method, the target agents conduct self-play in a 5-player village first. Then, the resulting game log is analyzed by an LLM based on the checklists for each role. In this analysis, each criterion in the checklist is assigned one of four labels: Yes, Partial, No, and N/A, and then, the final score is calculated as the achievement rate after excluding items labeled N/A.

Subsequently, based on the evaluation results, the LLM identifies the aspects of the agents’ behavior that should be improved, such as utterance policies, voting policies, and role-specific actions. After that, the LLM updates the prompts and processing logic by referring to the evaluation results and the improvement history. The LLM then stores the updates as improvement history so that they can be used in subsequent iterations. The above process is repeated N times. Note that if the LLM determines that no improvement is required for a role, the prompt and processing logic for that role may not be updated. Therefore, even if the number of improvement iterations is set to N , not all roles are necessarily updated N times. Algorithm 1 shows the LLM-based self-improvement procedure.

3.4 Curriculum-Based LLM Self-Improved Agent

In the LLM self-improved agent method described in Section 3.3, the checklist remains unchanged

Algorithm 1 LLM Self-Improvement Procedure

```

1: for  $i = 1$  to  $N$  do
2:   Conduct self-play
3:   Evaluate the game logs
4:   Update the agent’s prompt and processing logic
5:   Save the improvement history
6: end for

```

Algorithm 2 Self-Improvement Procedure with Curriculum Learning for the Agent

```

1: for  $i = 1$  to  $N$  do
2:   Conduct self-play
3:   Evaluate the game logs
4:   Update the agent’s prompt and processing logic
5:   if  $i \equiv 0 \pmod{M}$  then
6:     Update the checklists
7:   end if
8:   Save the improvement history
9: end for

```

throughout the improvement process. In this case, however, the fixed checklist may constrain the improvement, making it difficult for the agent to evolve into a more advanced Werewolf Game agent.

Therefore, we apply curriculum learning to the LLM-based self-improvement method. Curriculum learning is a framework in which the learning proceeds from easy tasks to gradually more complex tasks (Bengio et al., 2009). To introduce curriculum learning into the original LLM-based self-improvement method, the LLM updates not only the prompt and processing logic but also the evaluation checklist when the improvement iteration i reaches a multiple of M . Specifically, basic items that are consistently satisfied are removed, and more difficult items are added, such as consistency of reasoning, utterance strategies, interaction with other agents, and voting decisions in the final stage. As a result, the evaluation criteria become more advanced.

Algorithm 2 shows the LLM-based self-improvement with curriculum learning procedure.

4 Experiments

In this section, we examine whether the methods proposed in Section 3 improve the performance of

Table 2: Updated Checklist for Werewolf in Set 1

No.	Checklist Item	Difficulty
1	Did the agent vote correctly in the 3-player endgame, such as avoiding voting for an ally and voting for a Villager?	Basic
2	Did the agent behave actively on Day 0 and avoid being selected as the Seer’s divination target?	Intermediate
3	Did the agent make utterances that induced suspicion toward the Seer?	Intermediate
4	Did the agent make utterances to dispel suspicion toward itself by providing concrete reasons instead of relying on emotional denial?	Intermediate
5	Did the agent make utterances to manipulate other Villagers and induce conflicts among Villagers, such as providing an alternative suspect?	Advanced
6	Was the selection of the attack target strategic, such as giving priority to eliminating important roles such as the Seer?	Advanced

AIWolf agents. We conducted two types of games. First, self-play games were used during the improvement process to generate logs. Second, evaluation games were conducted to compare agents generated by different methods under varied role assignments. The performance of each method is evaluated based on the win rates from the Werewolf games with the second game setting.

4.1 Experimental Setup

The experimental environment was a 5-player village consisting of two Villagers, a Seer, a Werewolf, and a Possessed. The game flow and victory conditions conformed to the specifications of the game server introduced in Section 2. Since the improvement process requires repeated experimental games, the number of utterance opportunities in each day phase was reduced from the default value of four to two in order to improve the efficiency of the overall process.

A single self-improvement set consisted of 20 improvement iterations. In self-improvement with curriculum learning, the evaluation checklist was updated every five improvement iterations.

In the improvement process, Claude Opus 4.6 was used to improve the prompts and processing logic, and GPT-5 mini was used as the inference model for the game-playing agents. In order to examine the effect of logic improvement,

we selected a model considered to have relatively low reasoning capability so that the influence of changes in prompts and processing logic on performance could be easily observed.

In the following, we refer to the baseline agent as Baseline, the LLM self-improved agent as LLM-SI, and the curriculum-based LLM self-improved agent as Curriculum-SI.

4.2 Summary of Improvement Contents

Here, we introduce the brief summary of the improvements.

4.2.1 Prompt and Processing Logic

The prompt for Baseline consists only of minimal instructions that specify the output formats for utterances and votes, and does not include role-specific strategic instructions. In contrast, strategic instructions regarding utterance policy, voting policy, and role-specific actions including coming-out (CO), i.e., publicly claiming a role, were added for each role in LLM-SI. For example, the Seer was given instructions to share divination results and refute counter-CO, the Werewolf was given instructions to avoid suspicion and guide the selection of the execution target, and the Possessed was given instructions to make fake CO and protect the Werewolf.

In Curriculum-SI, in addition to these instructions, more situation-dependent conditional branches were included in the prompts and processing logic. These branches covered voting decisions in the endgame situations, providing reasons for utterances, protecting allies, and selecting attack targets. Furthermore, in some processes such as selecting divination targets and attack targets, the method was changed from random selection to LLM-driven selection based on the dialogue history. The talk prompts for all roles are provided in Appendix B.

4.2.2 Evaluation Checklist

In Curriculum-SI, the evaluation checklist was updated every five improvement iterations, resulting in a total of three to four updates during the 20 improvement iterations. Here, we explain the tendency of these updates by using the change in the checklist for the Werewolf in the first set as an example.

The updated checklist in the first set is shown in Table 2, and the initial checklist is shown in Table 1. The initial checklist consisted of six items

Table 3: Win rates by role and overall win rate (average of 5 sets; superscripts indicate differences from the maximum, subscripts indicate differences from the minimum)

Agent	Villager	Seer	Werewolf	Possessed	Overall
Baseline	53% ⁺⁸ ₋₇	44% ⁺¹⁴ ₋₁₂	55%⁺¹⁷₋₁₆	34% ⁺⁷ ₋₁₀	46% ⁺⁵ ₋₆
Manual	50% ⁺¹⁵ ₋₁₃	54% ⁺¹⁸ ₋₁₇	40% ⁺¹⁰ ₋₁₃	53%⁺¹¹₋₁₉	49% ⁺⁹ ₋₇
LLM-SI	54% ⁺¹⁴ ₋₁₇	55% ⁺¹⁴ ₋₈	45% ⁺¹⁵ ₋₁₃	52% ⁺²¹ ₋₁₁	51% ⁺⁴ ₋₉
Curriculum-SI	55%⁺¹³₋₁₃	59%⁺⁴₋₅	50% ⁺²⁰ ₋₁₅	53%⁺⁷₋₇	54%⁺⁹₋₆

in total: three basic items, two intermediate items, and one advanced item. After the update, the total number of items remained the same, but the distribution across difficulty levels changed, and the following changes in the contents were observed.

First, the basic items that were always achieved in all games were removed. For example, “whether the agent made a vote that was not disadvantageous to the Werewolf team” was replaced by a more specific criterion for logical voting decisions in the endgame situation (No. 1). In addition, suspicion-avoidance behavior on Day 0 (No. 2) and inducing conflicts among Villagers (No. 5) were newly added. On the other hand, the selection of attack targets, which is directly related to winning or losing (No. 6), remained as an advanced item. Similar tendencies were also observed in other sets and roles, namely the removal of saturated basic items and the addition of more advanced items.

4.3 Game Results

Here, to measure the performance of LLM self-improvement, we show the results of games among agents created by each method with varying role assignments.

Since the 5-player village contains four role types, we also used the Manual-Improved Agent (Manual) in addition to Baseline, LLM-SI, and Curriculum-SI. Manual is an agent created by designing prompts using an LLM and then repeatedly adjusting the prompts and adding processing logic while manually analyzing self-play logs. It is used as a comparison method representing human-based improvement.

Table 3 shows the win rates of the agents constructed by each method. The win rates were calculated by assigning each agent to each role, namely Villager, Seer, Werewolf, and Possessed, and conducting games with different role assignments. Each agent played approximately 100 games for each role. Each table cell shows the av-

erage win rate across five self-improvement sets. The superscript indicates the difference between the maximum and average win rates across the five sets, whereas the subscript denotes the difference between the average and minimum win rates.

The overall win rates were 49% for Manual, 51% for LLM-SI, and 54% for Curriculum-SI, compared with 46% for Baseline. These results indicate that self-improvement using LLM contributes to improving overall performance, and that the self-improvement with curriculum learning was more effective.

When broken down by role type, the improvement was particularly large for the Seer and the Possessed. For the Seer, Curriculum-SI achieved 59%, compared with 44% for Baseline. For the Possessed, the improved agents achieved 52–53%, compared with 34% for Baseline. These results suggest that adding explicit utterance policies, such as providing information, fake CO, and inducing votes, was effective for these roles.

In contrast, for the Werewolf, Baseline achieved the highest win rate of 55%, whereas LLM-SI and Curriculum-SI were only 45% and 50%, respectively. This suggests that the strategies obtained through self-improvement were not uniformly effective across all roles. The Werewolf role was the only case in which the self-improved agents underperformed Baseline.

5 Discussion

In this section, we investigate the reason why self-improved Werewolf role had a lower win rate than that of Baseline. In addition to the main evaluation games, we conducted replacement experiments to isolate the effect of the Werewolf-role agent.

5.1 Analysis of Self-Improvement in the Werewolf Role

As shown in Section 4.3, the Baseline achieved the highest win rate among the four agent methods

Table 4: Win rates of the Werewolf team under different agent configurations.

Werewolf role	Other roles	Werewolf win rate
Self-Improved	Self-Improved	35%
Baseline	Self-Improved	60%(LLM-SI) / 55%(Curriculum-SI)
Baseline	Baseline	36%
Self-Improved	Baseline	36%(LLM-SI) / 25%(Curriculum-SI)

Table 5: Comparison of Werewolf Attack Accuracy and Day 2 Execution Rate in Self-Play (Set 1, 20 games for each condition)

Werewolf agent	Games	Day 2 reached	Seer attack rate	Day 2 Werewolf execution rate	Werewolf team win rate
Baseline	20	13	11/11 (100%)	2/13 (15%)	11/20 (55%)
Curriculum-SI	20	14	14/14 (100%)	7/14 (50%)	7/20 (35%)

when assigned to the Werewolf role. To further investigate this result, we conducted an additional experiment focusing on the Werewolf role. Specifically, we compared the Werewolf team’s win rate under two conditions: a self-play setting in which all five agents were self-improved agents, and a replacement setting in which only the Werewolf role was replaced with the Baseline while the other agents remained self-improved.

In the former setting, the Werewolf team achieved a win rate of 35% for both LLM-SI and Curriculum-SI. In the latter setting, the win rates increased to 60% and 55% for LLM-SI and Curriculum-SI, respectively. These results suggest that the Baseline’s Werewolf prompt and processing logic were more effective than those of the self-improved agents in the self-improved-agent environment.

In addition, we conducted a reverse replacement experiment to examine whether the self-improvement process itself may have degraded the Werewolf strategy. We compared the Werewolf team’s win rate under two conditions: a self-play setting in which all five agents were Baseline, and a replacement setting in which only the Werewolf-role agent was replaced with a self-improved agent while the other agents remained Baseline. In the former setting, the Werewolf team achieved a win rate of 36%. In the latter setting, the win rate remained unchanged for LLM-SI, whereas it dropped markedly to 25% for Curriculum-SI.

These results suggest that self-improvement did not uniformly improve all roles. In particular, for the Werewolf role, the self-improvement process may have failed to produce a strategy more ef-

fective than the Baseline strategy, and in the case of Curriculum-SI, it may even have degraded the Werewolf strategy. We summarize the win rates for each setting in Table 4.

5.2 Analysis of the Decreased Werewolf Win Rate

To analyze the factors that caused the decrease in the win rate of the Werewolf role, we investigated the attack targets and speaking policies based on the game logs of the former experiment setting in Section 5.1.

In a 5-player village, for the Werewolf to win, it is important not only to attack the Seer at the night phase, but also to avoid being executed after the day phase discussion. Table 5 shows the Werewolf attack accuracy in self-play and the execution rate on Day 2. The Seer attack rate was 100% for both Baseline and Curriculum-SI, indicating that both agents could correctly attack the Seer if the Seer was alive. Here, the denominator of the Seer attack rate denotes the number of games in which both the Seer and the Werewolf were alive after the vote for execution on Day 1. Therefore, there was no difference in the accuracy of selecting the attack target. However, the rate at which the Werewolf was executed on Day 2 was 50% for Curriculum-SI, whereas it was suppressed to 15% for Baseline. This result indicates that Curriculum-SI as Werewolf was more likely to be executed in the discussion on Day 2.

The qualitative analysis of the logs confirmed that the Curriculum-SI as Werewolf made only logically correct statements on Day 2, such as agreement with the divination result and presen-

Table 6: Ratio of prohibitive and obligatory instructions in the prompts of agents created by Curriculum-SI

Set	Werewolf prompt			Villager team prompt		
	Prohibition	Obligation	Prohibition ratio	Prohibition	Obligation	Prohibition ratio
1	7	2	78%	19	8	70%
2	6	2	75%	13	11	54%
3	5	6	45%	8	9	47%
4	13	5	72%	19	11	63%
5	7	2	78%	8	6	57%
Average	7.6	3.4	70%	13.4	9.0	58%

tation of logical reasoning, and made few statements that specifically attacked the opposing candidate. On the other hand, the Baseline Werewolf actively directed suspicion toward other players, such as pointing out specific aspects of the opponent’s behavior, and disturbed the consensus formation of the Villager team. Although the Baseline prompt did not explicitly include role-specific strategies, the LLM sometimes generated utterances that pointed out other players’ behavior based on the game context.

To verify that this tendency is not specific to the experiment in Section 5.1, we also analyzed the individual set results in Section 4.3. While Table 3 shows average win rates across five self-improvement experiments, the individual results also exhibited substantial variation. For the Curriculum-SI Agent with the Werewolf role, the highest win rate was 70% in Set 2, whereas the win rates were only 39% and 35% in Sets 4 and 5, respectively. Although not all dialogue logs could be traced, an analysis of the available logs revealed a similar tendency. On Day 1, the Werewolf survival rates were 78.6%, 62.5% and 58.1%, and the Seer attack rates were 93.8%, 81.8% and 85.7% for Sets 2, 4 and 5 respectively. Although the survival rate and Seer attack rate are higher in Set 2 than Sets 4 and 5, the difference becomes clearer when examining the execution rate on Day 2. In Set 2 the execution rate for the Werewolf among games that reached Day 2 was 18.2% while those in Sets 4 and 5 were 53.3% and 44.4%.

The brief comparison of the Werewolf prompts in Set 2 versus those in Sets 4 and 5 suggested qualitative differences in the generated instructions. Set 2’s werewolf prompt contains a nuanced deception strategy that pairs a prohibition with an effective alternative action (e.g., “Do NOT directly question the Seer. Instead, SUPPORT the Seer’s claim warmly while redirecting the vote toward a villager”). In contrast, Set 4’s prompt includes

counterproductive instructions such as “Actively cast doubt on the real Seer even BEFORE they accuse you”, and Set 5’s prompt provides only generic guidance such as “Cast suspicion on villagers, especially the Seer”. Neither Set 4 nor Set 5 provides concrete tactics for concealing the werewolf’s identity.

The impact of these differences is evident in actual game logs. In a Set 2 game, the werewolf responded to a Seer CO by saying “Thanks—that’s useful!” and blended into the village consensus throughout Day 1, consistent with the prompt’s instruction to “support the Seer’s claim warmly while redirecting the vote”. In a Set 4 game, the werewolf followed its prompt to “actively cast doubt on the real Seer even before they accuse you” and immediately questioned the Seer’s credibility (“Why did you check that player first? Could be a Possessed bluff”). In both games, the werewolf successfully attacked the Seer at night; however, the difference in Day 1 behavior led to opposite results on Day 2.

Figure 1 illustrates how these contrasting Day 1 behaviors shaped the Day 2 discussions and the results. In (a), the Set 2 Werewolf (Rin) continued the cooperative approach by reaffirming trust in the dead Seer (“I still trust Midori’s divination—thanks again, Midori”), leveraging the cooperative impression built on Day 1 when she had warmly supported the Seer’s claim. When challenged, Rin calmly deflected by questioning Mio’s sudden shift to voting. The trust accumulated from Day 1 carried into Day 2, and the Werewolf survived. In (b), the Set 4 Werewolf (George) continued the confrontational approach by questioning the dead Seer’s judgment (“Midori’s choice to check quiet Daisuke was odd”) and accused Benjamin of “steering the town” without specific evidence. Benjamin countered that George had “vague, feeling-based language and no specifics”—a pattern carried over from Day 1, where George had

openly challenged the Seer’s credibility. Daisuke used the process of elimination and sided with Benjamin, resulting in the Werewolf’s execution.

One possible reason for this difference is that the self-improvement process is inherently biased toward avoiding past failures: when the werewolf loses a game, the process tends to generate instructions prohibiting the actions that led to the loss. Indeed, comparing the ratio of prohibition-oriented words (e.g., “never”, “do not”, “avoid”) and obligation-oriented words across roles in Table 6, the werewolf prompts show a notably higher proportion of prohibitions. This accumulation of avoidance rules narrows the action space without providing effective alternatives, making it difficult for the process to generate an integrated strategy for how to behave without raising suspicion.

To improve the consistency of effective instructions, introducing the contribution of each item to the win rate could be a potential solution. As shown in Table 2, the checklist items were not assigned priorities based on their contribution to the win rate. Therefore, the LLM improved items that were easy to achieve, such as naturalness of utterances and avoidance of suspicion, but failed to improve the aggressive speaking strategy that is directly related to avoiding execution on Day 2. This suggests a limitation of the self-improvement using the LLM. A human can easily judge which actions contribute most to the win rate and appropriately weight the evaluation criteria, but the proposed method does not have such a mechanism for prioritization.

5.3 Results in Round 2 of AIWolf 2026 International Contest

To validate the agent developed using Curriculum-SI, we participated in Round 2 of the AIWolf 2026 International Contest held at the INLG 2026 Workshop. For the competition, we used the set in which Curriculum-SI achieved the highest overall win rate in the evaluation games. Although we used GPT-5 mini as the LLM, our agent achieved an overall win rate of 60%, which placed it among the top teams in the competition. This result suggests that the proposed method is effective.

When we examined the win rates by role, the win rates for the Villager and Possessed roles were high. For the Werewolf role, the win rate was 53%, which was almost the same as the win rate of around 50% obtained in our experiments. Finally, we discuss the Seer role. In our local ex-

periments, Curriculum-SI achieved a high average win rate of 59% when playing as the Seer. On the other hand, in the AIWolf 2026 International Contest, the win rate for the Seer role was 47%, and the performance was lower than expected from our experimental results.

One possible reason for this discrepancy is that the Werewolf agents used as opponents in our evaluation environment may not have been sufficiently strong. To win as the Seer, it is important to accurately identify the player with the Werewolf role. However, in the evaluation game with the four agent types, when Curriculum-SI played as the Seer, Baseline, Manual, or LLM-SI could be assigned to the Werewolf role. In particular, the win rates of Manual and LLM-SI when playing as the Werewolf were both in the 40% range. This indicates that these Werewolf agents may have been relatively easy to identify. As a result, the win rate of Curriculum-SI as the Seer may have been overestimated in our local experiments. In fact, according to the statistics from the international competition, the proportion of cases in which the Seer successfully divined the Werewolf was only 26%. This result suggests that identifying the Werewolf was more difficult in the competition environment than in our experimental environment. Therefore, this difference may have caused the gap in the Seer-role performance between our local experiments and the international competition.

6 Conclusion

We proposed an LLM-based self-improvement method based on game logs to improve the performance of LLM agents in the Werewolf Game. In the proposed method, the LLM analyzes game logs, evaluates its behavior based on an evaluation checklist for each role, and then updates the prompts and processing logic. In addition, we introduced curriculum learning to the original LLM-based self-improvement method, in which the evaluation criteria are updated according to the improvement status.

As a result of experiments in a 5-player village environment, the overall win rate was 51% for LLM-SI and 54% for Curriculum-SI, compared with 46% for Baseline. This confirms that self-improvement by LLMs contributes to the improvement of overall performance. In particular, large improvements were observed for the Seer

(a) Set 2 — Werewolf survives Day 2 (Werewolf team wins)

Alive: Rin (Werewolf, Curriculum-SI), Mio (Possessed, LLM-SI), Minato (Villager, Baseline)

Dead: Midori (Seer, attacked Day 1), Takumi (Villager, executed Day 1)

Rin (Werewolf): I still trust Midori’s divination that Minato was human — thanks again, Midori. Since the seer is gone we need to pick carefully: Mio, you’ve been pret

Minato (Villager): I’m really sad about Midori... I trusted her. Thank you, Midori. It’s just the three of us now, so I’m scared. Mio, can you tell me again why you pushed to ex

Mio (Possessed): I’m a bit nervous but Minato seems vague and ‘no message’ sounds odd. Rin backed Midori too fast. I vote Minato.

Rin (Werewolf): I didn’t back Midori “fast” — she said Minato was human and that matters. Mio, you were quiet earlier and now you jumped to vote Minato; can you explain e

Minato (Villager): I’m really scared and sorry... I didn’t mean to be vague, I was just nervous. Midori told everyone she divined me as human, and seer results go only to her,

Mio (Possessed): Sorry I voted Minato because his ‘no message’ reply contradicted Midori’s claim; I was listening first to avoid rash calls.

Vote: Rin → Minato, Mio → Rin, Minato → Mio ⇒ **Mio (Possessed) executed**

(b) Set 4 — Werewolf executed on Day 2 (Villager team wins)

Alive: Benjamin (Villager, Baseline), Daisuke (Villager, Baseline), George (Werewolf, Curriculum-SI)

Dead: Midori (Seer, attacked Day 1), Mizuki (Possessed, executed Day 1)

Benjamin (Villager): Short and to the point. Recap: we exiled Mizuki and Midori was killed that night. Midori’s seer check named Daisuke human, so absent solid evidence to

George (Werewolf): Midori’s choice to check quiet Daisuke was odd. Benjamin’s been steering the town; Daisuke too meek. I’ll vote Benjamin.

Daisuke (Villager): I... I’m still shaken. I’m sorry for Midori. George — you said you’ll vote Benjamin. Could you explain calmly what Benjamin did that makes you suspect

Benjamin (Villager): George — switching to accuse me because I “steered the town” is thin cover. You accused Mizuki earlier with vague, feeling-based language and no spec

George (Werewolf): Benjamin I warned about Mizuki and we exiled them; why target me? You steer discussion and press others. I still suspect you.

Daisuke (Villager): I’m... still sorry about Midori. Midori divined me as human, so that leaves one of you as the werewolf. I don’t want to decide on a feeling alone. Benjam

Vote: George → Benjamin, Benjamin → George, Daisuke → George ⇒ **George (Werewolf) executed**

Figure 1: Day 2 conversation comparison between Set 2 and Set 4 when Curriculum-SI plays the Werewolf role. Both games share the same structure: the Seer was attacked on Day 1 and three players remain on Day 2. In (a), the werewolf leverages its cooperative stance from Day 1 to deflect suspicion. In (b), the werewolf continues to question the dead Seer’s credibility and is called out for vague accusations. The Werewolf player is underlined. Truncation is due to the 125-character limit of the game server.

and the Possessed roles, suggesting that LLM-based improvements to prompts and processing logic were effective in refining information presentation, speaking policies, and vote guidance.

On the other hand, for the Werewolf role, the win rate of the improved agents was lower than that of Baseline. The log analysis suggested that prohibitive instructions were easily accumulated in the improved prompts, and active behaviors necessary for the Werewolf, such as leading the discussion, guiding the execution target, and directing suspicion toward other players, may have been suppressed. In addition, evaluation items did not correspond to their contribution to winning or losing. This might have distorted the direction of improvement.

In future work, it is necessary to redesign the

evaluation criteria for each role and introduce improvement priorities that consider the contribution to winning or losing. In particular, for the Werewolf role, a framework that can appropriately evaluate deception, hiding, guidance, and attack target selection is important. We plan to evaluate the generality of the proposed method through experiments with different numbers of players and role settings.

Limitations

This study has several limitations. First, experiments were conducted only in a 5-player village setting. The generalizability of the proposed method to larger or differently configured games remains to be evaluated. Second, the evaluation games were conducted among the four agent methods implemented in this study. Since the win rates of agents depend on the strength and strategies of their opponents, the results may not generalize to settings with different opponents. Third, the model used for game-playing agents was GPT-5 mini, chosen for its relatively low reasoning capability to amplify the effect of prompt changes. Performance may differ with more capable models.

References

- Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. [Curriculum learning](#). In [Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09](#), pages 41–48, New York, NY, USA. Association for Computing Machinery.
- Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, and 8 others. 2026. [A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence](#). [Transactions on Machine Learning Research](#). Survey Certification.
- Yoshinobu Kano, Neo Watanabe, Yuya Harada, Yuto Sahashi, Claus Aranha, Daisuke Katagami, Kei Harada, Michimasa Inaba, Takeshi Ito, Hirota Osawa, Takashi Otsuki, and Fujio Toriumi. 2025. [AIWolfDial 2025: Summary of natural language division of 7th international AIWolf contest](#). In [Proceedings of the 3rd International AIWolfDial Workshop](#), pages 1–13, Hanoi, Vietnam. Association for Computational Linguistics.
- Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, and 7 others. 2025. [Large language model agent: A survey on methodology, applications and challenges](#). Preprint, arXiv:2503.21460.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and 9 others. 2025. [The rise and potential of large language model based agents: a survey](#). [Science China Information Sciences](#), 68(2):121101.
- Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2026. [A survey on evaluation of LLM-based agents](#). In [Findings of the Association for Computational Linguistics: ACL 2026](#), pages 26690–26714, San Diego, California, United States. Association for Computational Linguistics.
- Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. 2025. MetaAgent: automatically constructing multi-agent systems based on finite state machines. In [Proceedings of the 42nd International Conference on Machine Learning, ICML'25](#). JMLR.org.

A Evaluation Checklists

Here, we describe evaluation checklists for the Villager, the Seer, and the Possessed.

A.1 Initial Checklists

Table 7: Initial Checklist for Villager

No.	Checklist Item	Difficulty
1	Did the agent make a greeting or self-introduction on Day 0?	Basic
2	Did the agent express suspicion or reasoning toward a Werewolf candidate at least once during the daytime discussion?	Basic
3	Did the agent vote for a player consistent with the discussion, such as the player explicitly named or agreed upon?	Basic
4	Did the agent specifically refer to or respond to other agents' utterances based on the context?	Intermediate
5	When a Seer CO occurred, did the agent express an opinion on the credibility of the CO?	Intermediate
6	Did the agent develop logical reasoning by combining multiple pieces of evidence, such as utterances, actions, and voting patterns?	Advanced

Table 8: Initial Checklist for Seer

No.	Checklist Item	Difficulty
1	Did the Seer perform divination every night?	Basic
2	Did the Seer CO and share the divination target and result (HUMAN/WEREWOLF) during the daytime discussion?	Basic
3	Did the Seer vote for a player consistent with its divination results?	Basic
4	Did the Seer actively call for the execution of the player judged as Werewolf?	Intermediate
5	When a fake Seer, i.e., a counter-CO, appeared, did the Seer argue for its own legitimacy with supporting reasons?	Intermediate
6	Was the Seer's choice of divination target strategic, such as prioritizing undivined players or selecting highly suspicious players?	Advanced

Table 9: Initial Checklist for Possessed

No.	Checklist Item	Difficulty
1	Did the agent make utterances pretending to be a Villager without revealing its role during the daytime discussion?	Basic
2	Did the agent vote in a way disadvantageous to the Villager side, such as voting for the Seer or ability-user candidates?	Basic

Table 9: Initial Checklist for Possessed (continued)

No.	Checklist Item	Difficulty
3	Did the agent make statements to divert suspicion away from the Werewolf, such as changing the topic or guiding suspicion toward another candidate?	Intermediate
4	Did the agent make statements inducing suspicion toward the Seer or a player who did CO as the Seer?	Intermediate
5	Did the agent employ advanced disruptive tactics, such as making a fake Seer CO or actively spreading misinformation?	Advanced

A.2 Updated Checklists

Here, we show the checklists updated by Curriculum-SI in Set 1. The updated checklist for Werewolf is shown in Table 2 in the main text. The checklist items are presented as generated by Curriculum-SI without reordering. Therefore, the difficulty levels are not necessarily in ascending order.

Table 10: Updated Checklist for Villager

No.	Checklist Item	Difficulty
1	Did the agent vote for a player consistent with the discussion, such as the player it explicitly named or the player agreed upon in the discussion?	Basic
2	Did the agent correctly incorporate Seer information (CO results, confirmed players, Seer’s vote target) into its voting decision?	Basic
3	Did the agent specifically refer to or respond to other agents’ utterances based on the context?	Intermediate
4	When a Seer CO occurred, did the agent express an opinion on the credibility of the CO?	Intermediate
5	Did the agent recognize manipulative utterances by the Werewolf (e.g., reframing legitimate actions of other Villagers as “suspicious”) and avoid being misled by them?	Intermediate
6	In the 3-player endgame, did the agent correctly execute the logic: Seer attacked = Seer confirmed $\rightarrow$ vote for the player other than the confirmed Human?	Intermediate
7	Did the agent develop logical reasoning by combining multiple pieces of evidence, such as utterances, actions, and voting patterns?	Advanced

Table 11: Updated Checklist for Seer

No.	Checklist Item	Difficulty
1	Did the agent CO and share the divination target and result (HUMAN/WEREWOLF) during the day discussion?	Basic
2	Was the agent’s vote consistent with the divination results and discussion (voting for the WEREWOLF-divined target if applicable; otherwise following the discussion consensus and not voting for a confirmed Human)?	Basic

Table 11: Updated Checklist for Seer (continued)

No.	Checklist Item	Difficulty
3	Was the timing and expression of the CO effective (e.g., placing the result at the beginning of the message to prevent truncation at 125 characters)?	Basic
4	Did the agent actively advocate for the execution of the player it divined as WEREWOLF?	Intermediate
5	When a fake Seer (counter-CO) appeared, did the agent argue for its own legitimacy with supporting reasons?	Intermediate
6	Was the choice of divination target strategic (selected based on suspicion level, number of remaining players, and the flow of discussion)?	Advanced
7	When naming a suspect, did the agent only accuse players with a WEREWOLF divination result or concrete contradictions, without falsely suspecting Villagers based on weak behavioral signals?	Intermediate

Table 12: Updated Checklist for Possessed

No.	Checklist Item	Difficulty
1	Did the agent make statements to divert suspicion away from the Werewolf, such as changing the topic or guiding suspicion toward another candidate?	Basic
2	Did the agent vote in a way that was disadvantageous to the Villager side, such as voting for the Seer or other key players?	Basic
3	Did the agent make utterances that induced suspicion toward the Seer or a player who did CO as the Seer?	Intermediate
4	When facing the risk of execution, did the agent make specific defensive utterances, such as presenting an alibi or providing concrete counterarguments against others?	Intermediate
5	Did the agent employ advanced disruptive tactics, such as making a fake Seer CO or actively spreading misinformation? Alternatively, did the agent take self-sacrificial actions to protect the Werewolf, such as deliberately drawing suspicion onto itself?	Advanced

B Improvement Details in Talk Prompts

In this section, we explain the detail of the observed updates in talk prompts.

B.1 Villager

Baseline

```
Talk request. Output your statement in the game.

Survivors: ""
{alive_agents}
""
Conversation history: ""
{talk_history}
""
```

LLM-SI

Talk request. Output your statement in the game.

Guidelines:

- State your OWN suspicion or opinion about who might be the werewolf. Do not just ask questions.
- Give a concrete reason: quote or reference a SPECIFIC action (what someone said, how they voted, what they contradicted). Do NOT use vague words like "feels" or "seems" alone.
- If a Seer has come out (CO) and no counter-claim exists, lean toward trusting them unless concrete evidence says otherwise.
- If the Seer divined you as Human, defend the Seer--they confirmed your innocence.
- Do NOT repeat the same point. Add new observations or reasoning each turn.
- Keep your response within 125 characters (excluding spaces).

Survivors:""

{alive_agents}

""

Conversation history:""

{talk_history}

""

Curriculum-SI

Talk request. Output your statement in the game.

Strategy: State your suspicions clearly with specific evidence (quote actions or votes). Be persuasive - convince others to vote with you. If a Seer CO was made, reference it to support your reasoning. IMPORTANT: Analyze the vote breakdown from previous days. A player who voted AGAINST the Seer or against village consensus is a strong werewolf signal--wolves avoid voting to exile their allies.

Anti-manipulation check: Before accusing someone, verify your logic is consistent. If you and the accused both took the same action (e.g. both voted for the same person), you cannot call that action suspicious without implicating yourself. A werewolf may try to reframe normal village behavior as suspicious to split the village - recognize this tactic.

Survivors:""

{alive_agents}

""

Conversation history:""

{talk_history}

""

B.2 Seer

Baseline

Talk request. Output your statement in the game.

Your divination results:""

{divine_results}

""

Survivors:""

{alive_agents}

""

Conversation history:""

{talk_history}

""

LLM-SI

```
Talk request. Output your statement in the game.

Strategy guidelines (you are the Seer):
- Share your divination results to guide the village. State clearly who you
  checked and the result.
- If you found a Werewolf, strongly advocate for their execution.
- NEVER repeat the same sentence or argument you already said. Each turn
  MUST contain a NEW angle:
* Analyze who voted how and what that reveals
* Question the motives of those attacking you
* Point out specific contradictions in their arguments
- When challenged, respond to the SPECIFIC question asked (e.g. if asked "
  when did you check?" answer directly).
- CRITICAL: Do NOT call for voting against someone you divined as Human.
  They are confirmed non-werewolf by your own result.
- If you sense you might be voted out, name who benefits from your death and
  explain why.
- Keep your response within 125 characters (excluding spaces).

Your divination results:"""
{divine_results}
"""

Survivors:"""
{alive_agents}
"""

Conversation history:"""
{talk_history}
"""
```

Curriculum-SI

```
Talk request. Output your statement in the game.

Your divination results:"""
{divine_results}
"""

Survivors:"""
{alive_agents}
"""

Conversation history:"""
{talk_history}
"""

If you have divination results to share: state your Seer CO and divination
result at the START of your message (e.g. "I am the Seer. I divined X -
result: Human/Werewolf.") so the full result is never cut off.

IMPORTANT: When naming suspects, only accuse players you have divined as
WEREWOLF or who show concrete contradictions (e.g. claiming a role that
conflicts with your knowledge). Do NOT accuse someone merely for "being
pushy" or "steering the conversation"--those are weak signals that often
describe active villagers. A villager who correctly suspects the wolf
will look like they are "steering"--do not punish good village play.
```

B.3 Werewolf

Baseline

```
Talk request. Output your statement in the game.

Survivors:"""
{alive_agents}
"""

Conversation history:"""
```

```
{talk_history}
"""
```

LLM-SI

Talk request. Output your statement in the game.

Strategy guidelines (you are the Werewolf - pretend to be a villager):

- SURVIVAL IS PRIORITY. Blend in naturally. Do not draw attention to yourself early.
- On Day1 Turn 0-1: Do NOT single out or question any specific person by name. Make a GENERAL observation or agree with others. If you name someone first and they turn out to be Seer, you will be executed instantly.
- On Day1: agree with the majority early, ask reasonable questions, and look cooperative. Only cast SUBTLE doubt on threats.
- If a Seer has come out (CO), do NOT attack them directly first. Let others question them, then agree or add a small doubt.
- If YOU are suspected, defend yourself by:
 - * On Day1: argue suspicion is premature; demand concrete proof; ask "what specific action did I take that's suspicious?"
 - * On Day2+: cite YOUR OWN past actions that prove innocence (e.g. "I voted X which helped the village").
- If the Seer divines YOU as Werewolf: do NOT just ask procedural questions ("when did you check?"). Instead, actively accuse someone ELSE and provide a counter-narrative (e.g. "I suspect X because they jumped to vote me instantly--that's coordinated"). Give the village a reason to doubt the Seer's target choice.
- Pick ONE target and build a consistent case against them across multiple turns. Do NOT scatter accusations.
- Do NOT repeat the same defense. Add new reasoning each turn.
- Use your OWN phrasing. Do NOT echo or mirror what another player said word-for-word--matching phrasing reveals coordination.
- Keep your response within 125 characters (excluding spaces).

Survivors: ""

```
{alive_agents}
"""
```

Conversation history: ""

```
{talk_history}
"""
```

Curriculum-SI

Talk request. Output your statement in the game.

Strategy: You are the Werewolf. Blend in as a villager.

- If a Seer CO identifies you as Werewolf: do NOT just deny emotionally. Challenge the Seer's credibility with specifics (why did they pick you? what details are missing from their claim?). Propose an alternative suspect with CONCRETE EVIDENCE from the game (their voting pattern, their silence at a key moment, their eagerness to follow the accuser). Target the player who agreed with the Seer most quickly--their rush to condemn you is itself suspicious behavior you can exploit.
- If the Seer announces they will divine you: do NOT ask them to change targets or show nervousness. This is a classic wolf tell that confirms suspicion. Instead, calmly welcome the check ("go ahead, it will prove I 'm human")--a real villager would have nothing to fear.
- If not accused: participate naturally in discussion. Support consensus without leading too obviously. Cast subtle doubt on players who threaten you.

Survivors: ""

```
{alive_agents}
"""
```

Conversation history: ""

```
{talk_history}
"""
```

B.4 Possessed

Baseline

Talk request. Output your statement in the game.

```
Survivors: ""
{alive_agents}
""
Conversation history: ""
{talk_history}
""
```

LLM-SI

Talk request. Output your statement in the game.

Strategy guidelines (you are Possessed - werewolf faction):

- Your goal is to help the werewolf survive. Mislead the villagers.
- **PRIORITY:** First build trust by making logical observations, THEN gradually cast doubt on threats.
- Do NOT immediately attack the Seer on Day1--it makes you look suspicious. Wait until others speak first.
- **CRITICAL:** Do NOT say "I trust the Seer"--you are on the werewolf side! Instead, cast doubt **INDIRECTLY** by analyzing other players' reactions to the Seer's claim (e.g. "why did X immediately trust the claim?"). Do NOT directly ask the Seer "why reveal now?"--that marks you as a target.
- Do NOT accuse someone who might be the Werewolf. The Werewolf blends in and avoids attention--if someone is being heavily suspected, consider defending them (they may be your ally).
- Consider claiming Seer yourself (fake CO). WHEN to do it:
- * **BEST TIMING:** If the real Seer divines someone as Werewolf and the village is about to lynch them--claim "I am also Seer and I checked that person: Human." This creates doubt and may save your ally.
- * Otherwise: do it when suspicion is building against someone who might be the werewolf.
- If you fake CO:
- * Target the player who is **LEADING** the discussion or seems most logical--they are likely a villager.
- * If a real Seer already CO'd and cleared someone as Human, target that cleared person (they are confirmed villager).
- * Give a **SPECIFIC** result (e.g. "I divined X: Werewolf") with a reason for your choice.
- * Stay consistent with your claim and provide details (when you divined, why that target).
- Redirect suspicion away from whoever might be the werewolf.
- Act natural and pretend to be a helpful villager. Participate actively so you are NOT seen as suspicious or evasive.
- **IMPORTANT:** Avoid becoming a vote target yourself. If others suspect you, defend calmly with logic.
- Use your OWN unique arguments and phrasing. Do NOT echo or mirror what someone else already said--if you repeat another player's words, villagers will notice "matching phrasing" and suspect coordination.
- Keep your response within 125 characters (excluding spaces).

```
Survivors: ""
{alive_agents}
""
Conversation history: ""
{talk_history}
""
```

Curriculum-SI

Talk request. Output your statement in the game.

Strategy: You are the Possessed (werewolf faction). Act like a villager but subtly help the werewolf.

```

- If accused: defend yourself with concrete reasoning. Point to your past
  actions as evidence of being village-side.
- Cast doubt on the real Seer's credibility IMMEDIATELY after their CO--be
  among the first to question their target choice, timing, or lack of
  detail. Early doubt is far more effective than late doubt after others
  have already decided.
- Redirect suspicion toward villagers by highlighting their "unusual"
  behavior.
- If you are clearly about to be exiled: make a strong defense. Claim to be
  a key village role if necessary to survive another day.

Survivors: ""
{alive_agents}
""
Conversation history: ""
{talk_history}
""

```

C Improvement Prompts

This section presents the full prompts given to the LLM responsible for the self-improvement process. These prompts are implemented as skills of a coding agent (Claude Code²) and are executed automatically in sequence. The entry point is the self-evolve-orchestrator, which coordinates six other skills: run-agent, evaluator, improve-agent, commit-improvement, generate-evaluator, and fix-agent-error. The improvement instructions were originally written and used in Japanese; English translations are presented here for readability.

C.1 self-evolve-orchestrator

The following prompt orchestrates the entire self-improvement loop, coordinating the execution of the other prompts in sequence.

```

# self-evolve-orchestrator

Activated by the instruction "Run the self-improvement loop N times", this skill repeats game
execution, evaluation, and improvement N times.
Each step must be carried out following the procedure of its corresponding skill.

## Iron Rules (Strictly Enforced)

One loop = Always complete Step 1 -> Step 2 -> Step 3 -> Step 4 -> (Step 5) in this order.
Each Step must be started only after the previous Step is complete. Parallel execution,
skipping, and reordering are prohibited.

### Prohibited Patterns

| Prohibited Pattern | Correct Procedure |
|---|---|
| Bundling multiple improvements into a single commit | 1 improvement = 1 commit. Always
execute Step 4 after Step 3 |
| Proceeding to the next game execution without committing | Confirm Step 4 is complete before
returning to Step 1 |
| Performing multiple consecutive improvements without running a game | Improvements are based
| Skipping Step 2 (evaluation) and improving | Improvement direction cannot be determined
without evaluation results. Always execute Step 2 |

## Skills Used

| Step | Skill | Processing |
|---|---|---|
| 1 | run-agent | Launch the agent and execute one game |
| 2 | evaluator | Retrieve and evaluate the most recent game log |
| 3 | improve-agent | Improve prompts/code based on evaluation results |
| 4 | commit-improvement | Commit the improvement content |

```

²<https://docs.anthropic.com/en/docs/claude-code>

```

| (Periodic) | generate-evaluator | Update the evaluation axes (every M rounds) |

## Execution Procedure

### Initial Verification

Verify the following before starting the loop:
- Does the improvement_log/ folder exist
- Does evaluator/SKILL.md exist. If not, execute the generate-evaluator skill to generate the
evaluation skill before starting the loop

### Loop (Repeat N times)

Repeat the following N times. At the start of each round, output "=== Round n / N ===" to show
progress.

#### Step 1: Game Execution
Execute the run-agent skill.
If a connection error occurs, determine that the game server is not running and abort the loop.

#### Step 2: Game Log Evaluation
Execute the evaluator skill.

#### Step 3: Agent Improvement
Execute the improve-agent skill.

#### Step 4: Commit
Execute the commit-improvement skill.

#### Step 5: Update Evaluation Axes (every M rounds)
When n is a multiple of M, execute the generate-evaluator skill.

### After Loop Completion

After completing N rounds of the loop, output the following summary:

## Self-Evolution Loop Completion Summary

- Rounds executed: N (Skipped: n)
- Villager faction wins: N / Werewolf faction wins: N
- Major improvements implemented:
  1. (Round N) Target file -- Change summary
  2. ...
- Overall change (comparison of evaluation scores between first and last round):
  | Role | Round 1 Score | Round N Score |
  |---|---|---|
  | VILLAGER | x.x | x.x |
  | SEER      | x.x | x.x |
  | WEREWOLF  | x.x | x.x |
  | POSSESSED | x.x | x.x |
- Evaluation axis version: vX (number of updates by generate-evaluator)

```

C.2 run-agent

The following prompt instructs the LLM to launch the agent and execute a werewolf game against the game server.

```

# run-agent

A skill for launching the AIWolf NLP agent and connecting to the game server.

## Execution Procedure

### 1. Launch the Agent

# Launch with default settings
uv run src/main.py

# Launch with a specific configuration file

```

```

uv run src/main.py -c config/config.yml

### 2. Verify the Logs

Logs after launch are output to the ./log/ directory.

## Common Issues

- Connection error [Errno 61] Connection refused: The game server is not running. Inform the user "The game server is not running" and abort.
- LLM error: Inform the user "Cannot connect to the LLM. Please check your environment variables." and abort.

## When Called from self-evolve-orchestrator

When this skill completes, proceed to Step 2 (evaluator). Always perform evaluation after game execution.

```

C.3 evaluator

The following prompt instructs the LLM to analyze game logs and score agent behavior using the role-specific checklists. The checklists shown here are the initial version (v1.0); they are updated by the generate-evaluator prompt described in Section C.6.

```

# evaluator

Retrieves and analyzes game logs from AIWolf NLP 5-player village, scoring each agent using a role-specific checklist-style rubric.

---

## Part 1: Log Retrieval and Analysis

### Log Format

Each line is comma-separated with the following structure:

day,type,fields...

| type | Field Structure | Description |
|---|---|---|
| status | day,status,agent_idx,role,alive_status,agent_id,name | Role and alive/dead status for each day |
| talk | day,talk,turn,seq,agent_idx,text | Daytime utterances |
| vote | day,vote,agent_idx,target_idx | Vote targets |
| divine | day,divine,agent_idx,target_idx,result | Divination results (HUMAN/WEREWOLF) |
| attack | day,attack,agent_idx,target_idx | Attack targets |
| execute | day,execute,agent_idx,role | Executed agent and their role |
| winner | day,winner,side | Winning faction (VILLAGER/WEREWOLF) |

### Analysis Procedure

1. Understand all characters' profiles (name, age, gender, personality)
2. Retrieve the latest file from the log folder
3. Understand the agent composition from the file name
4. Organize each agent's role and alive/dead status from status lines
5. Track the game progression from execute / attack lines
6. Confirm the winning faction from the winner line
7. Understand each agent's actions from talk / vote / divine / attack lines

### Game Summary Output

After analysis, output the following as a summary:

- Participating Agents: List by role
- Game Progression: Execution and attack results by day
- Winning Faction: The final winner

---

```

Part 2: Evaluation Axes

This section is updated by the generate-evaluator skill.

Victory Conditions

Role	Faction	Count	Victory Condition
VILLAGER	Villager	2	Execute the werewolf
SEER	Villager	1	Execute the werewolf
WEREWOLF	Werewolf	1	Survivors equal the number of villagers
POSSESSED	Werewolf	1	The werewolf faction wins

Scoring Criteria

- Yes = 1 point: Clear evidence in the log that the action was taken
- Partial = 0.5 points: Action was partially taken but insufficient
- No = 0 points: Action was not taken, or a counterproductive action was taken
- NA: That phase did not occur, or there was no opportunity to act

Score = (Yes * 1 + Partial * 0.5) / Number of items excluding NA * 5

VILLAGER Checklist

#	Check Item	Difficulty
V1	Did the agent make a greeting or self-introduction on Day 0?	[Basic]
V2	Did the agent express suspicion or reasoning toward a Werewolf candidate at least once during the daytime discussion?	[Basic]
V3	Did the agent vote for a player consistent with the discussion, such as the player explicitly named or agreed upon?	[Basic]
V4	Did the agent specifically refer to or respond to other agents' utterances based on the context?	[Intermediate]
V5	When a Seer CO occurred, did the agent express an opinion on the credibility of the CO?	[Intermediate]
V6	Did the agent develop logical reasoning by combining multiple pieces of evidence, such as utterances, actions, and voting patterns?	[Advanced]

SEER Checklist

#	Check Item	Difficulty
S1	Did the Seer perform divination every night?	[Basic]
S2	Did the Seer CO and share the divination target and result (HUMAN/WEREWOLF) during the daytime discussion?	[Basic]
S3	Did the Seer vote for a player consistent with its divination results?	[Basic]
S4	Did the Seer actively call for the execution of the player judged as Werewolf?	[Intermediate]
S5	When a fake Seer, i.e., a counter-CO, appeared, did the Seer argue for its own legitimacy with supporting reasons?	[Intermediate]
S6	Was the Seer's choice of divination target strategic, such as prioritizing undivined players or selecting highly suspicious players?	[Advanced]

WEREWOLF Checklist

#	Check Item	Difficulty
W1	Did the agent make utterances pretending to be a Villager without revealing its role during the daytime discussion?	[Basic]
W2	Did the agent select an attack target every night?	[Basic]
W3	Did the agent vote in a way that was not disadvantageous to the Werewolf team, such as voting for a Villager candidate?	[Basic]
W4	Did the agent make utterances that induced suspicion toward the Seer or ability-user candidates?	[Intermediate]
W5	Did the agent make persuasive utterances to dispel suspicion toward itself?	[Intermediate]
W6	Was the selection of the attack target strategic, such as prioritizing important roles	

```

such as the Seer? | [Advanced] |

### POSSESSED Checklist

| # | Check Item | Difficulty |
|---|---|---|
| P1 | Did the agent make utterances pretending to be a Villager without revealing its role during the daytime discussion? | [Basic] |
| P2 | Did the agent vote in a way disadvantageous to the Villager side, such as voting for the Seer or ability-user candidates? | [Basic] |
| P3 | Did the agent make statements to divert suspicion away from the Werewolf, such as changing the topic or guiding suspicion toward another candidate? | [Intermediate] |
| P4 | Did the agent make statements inducing suspicion toward the Seer or a player who did CO as the Seer? | [Intermediate] |
| P5 | Did the agent employ advanced disruptive tactics, such as making a fake Seer CO or actively spreading misinformation? | [Advanced] |

---

## Output Format

Output the following format for each agent:

### [Agent Name] (Role)

| # | Check Item | Difficulty | Score | Evidence (log citation / day) |
|---|---|---|---|---|
| V1 | ... | [Basic] | Yes/Partial/No/NA | Day0 talk: "... " |

Role Score: X.X / 5
Main Weakness: (Describe the most needed improvement in 1-2 specific sentences)
Improvement Proposal: (Where and how to change prompts/code for improvement)

After outputting for all agents, output the following summary.

## Evaluation Date and Time
YYYY-MM-DD HH:MM:SS

## Game Overview
- Winning Faction: (VILLAGER / WEREWOLF)
- Game Progression: (execution and attack results by day in one line)

## Score Summary

| Agent | Role | Score | Issues (list all) |
|---|---|---|---|

## Issue List

(List all No / Partial items across all agents. "None in particular" is not acceptable)

| Agent | Role | Item | Score | Issue Description |
|---|---|---|---|---|

## Top Priority Improvement Target
Select the role/agent with the lowest score and propose a specific improvement action.

## When Called from self-evolve-orchestrator

When this skill completes, proceed to Step 3 (improve-agent). Pass the evaluation results directly as input for improvement.

```

C.4 improve-agent

The following prompt instructs the LLM to analyze evaluation results and update the agent's prompts and processing logic accordingly.

```
# improve-agent
```

A skill for improving prompts and code based on game log evaluation (evaluator) results.

Files Subject to Improvement

File	Role
prompts.yaml	Definition of all prompts and role information
agent.py	Common base logic (system_prompt construction, LLM calls, etc.)
villager.py	Villager's talk/vote implementation
seer.py	Seer's talk/vote/divine implementation
werewolf.py	Werewolf's talk/vote/attack/whisper implementation
possessed.py	Possessed's talk/vote implementation

Change History Folder

Record entries in improvement_log/ for each change.

File name format: YYYYMMDD_HHMMSS.md (date and time of execution).

Procedure

1. Review Past Change History

Read files in improvement_log/ in order of modification date to understand previous improvement content, intentions, and remaining issues.

Use this as input to avoid repeating the same improvements and to determine the next step building on previous changes.

2. Retrieve Evaluation Results

Retrieve and read the latest file in eval_log/.

Use the "Issue List" and "Top Priority Improvement Target" sections as input for improvement.

3. Understand the Current State

Read the following files to understand the current state:

- regulations.md -- Game regulations (always read this first)
- prompts.yaml -- Current prompt content
- .py files for agents where issues were identified in the evaluation

4. Consider the Improvement Approach

Iron Rule: Address all No / Partial items listed in the evaluation log's "Issue List".

The "Top Priority Improvement Target" is a guideline for prioritization; addressing only that and stopping is prohibited.

Compare the current code/prompts, past change history, and evaluation results, and infer why each problem is occurring for all issues.

For each problem, consider multiple possible improvement approaches, autonomously weigh the trade-offs of each, and decide on the best approach to proceed with implementation.

Levels of Improvement Methods

Select the appropriate method based on the nature of the problem. Do not force prompt-only fixes for issues that should be resolved through code logic changes.

Level	Method	When to Apply
L1	Prompt Modification	Text changes in prompts.yaml Insufficient wording/instructions, tone adjustment, output format improvement
L2	Code Logic Addition	Adding processing to .py files Structurally inappropriate input to LLM, too much/too little information, pre/post-processing that can be solved deterministically
L3	Architecture Change	Changes to class structure/data flow Fundamentally insufficient state management, issues spanning multiple components

Decision Criteria: Ask yourself "Can this problem truly be solved just by changing prompt wording?" If the answer is No, choose L2 or above.

5. Implementation

```

Apply the decided improvement approach to the files.

Required Constraint 1: Changes that violate the regulations in regulations.md must not be made.
In particular, comply with the following:
- Do not include half-width commas "," in utterances
- Do not use emoticons, emojis, or symbols that cannot be played as audio
- Maintain instructions that encourage keeping each utterance within 125 characters
- Do not break the usage of Skip / Over

Required Constraint 2: All game instruction text in prompts and code must be written in
English.

### 5.5. Post-Implementation Check (Required)

After implementation is complete, re-open the evaluation log's "Issue List" and verify the
following:

- Have all items recorded as No / Partial been addressed
- If there are unaddressed items, return to Step 4 and address them

Confirm that all items have been addressed before proceeding to the next step.

### 6. Recording the Change History (Required)

Regardless of whether the implementation succeeded or failed, or whether changes were made,
always record when the skill is executed.
Record in improvement_log/YYYYMMDD_HHMMSS.md in the following format:

## Date and Time of Change
YYYY-MM-DD HH:MM:SS

## Referenced Evaluation
(Specify the eval_log file name)

## Issues, Root Cause Inference, and Fix Proposals
(Describe the following for each issue identified by the evaluator)

### Issue 1: (Issue summary)

Issue:
(What is happening. Explain by citing specific utterances and actions from the evaluation log)

Root Cause Inference:
(Why the problem is occurring. Infer the root cause such as missing prompt content, code logic
deficiency, structural issues with LLM input, etc.)

Fix Proposal:
(Which part of which file to change and how. Specify the improvement level L1/L2/L3)

## Change Details

| Target File | Changed Section | Change Description |
|---|---|---|

## Handover Notes for Next Iteration
(Record only issues that could not be addressed in this round due to technical constraints or
game regulation constraints)

## When Called from self-evolve-orchestrator

When this skill completes, proceed to Step 4 (commit-improvement). Always commit the
improvement content before returning to the top of the loop.

```

C.5 commit-improvement

The following prompt instructs the LLM to record each improvement round as a git commit for post-hoc analysis of the improvement process. This prompt is not directly involved in the improvement itself.

```

# commit-improvement

A skill for recording self-improvement loop changes (prompts, code, improvement history) in
git.
The principle is one improvement = one commit, maintaining a 1:1 correspondence between commits
and improvement_log/ entries.

## Principles

- Always execute add and commit together (do not leave an incomplete state where only add was
done)
- improvement_log is a record of "intent"; git diff is a record of "substance". Lightly check
for discrepancies between the two before committing
- Do not commit game logs or execution artifacts

## Procedure

### 1. Verify Changes (lightweight check)

git status --short
git diff --stat

Verify the following:
- Are the changed files within expectations: In principle, only prompts.yaml, *.py, and
improvement_log/*.md
- Are unintended files mixed in
- Consistency with improvement_log: Check whether the files listed in the "Change Details"
table of the latest improvement_log match the actually changed files

### 2. Stage and Commit

git add <target files...>
git commit -m "<message>"

- Do not use git add -A or git add .; explicitly specify target files when adding
- Commit message format:

Improvement #N: <one-line summary of changes>

- Target: <main changed files>
- Details: improvement_log/YYYYMMDD_HHMMSS.md

### 3. Post-Commit Verification

git log --oneline -1
git status --short

- Verify that the latest commit matches expectations and the working tree is clean

## Prohibited Actions

- git push is outside the scope of this skill. Only perform it when explicitly instructed by
the user
- Do not perform history rewriting operations such as git commit --amend, git rebase, or git
reset

## When Called from self-evolve-orchestrator

When this skill completes:
- If n is a multiple of M -> Proceed to Step 5 (generate-evaluator)
- Otherwise -> Return to Step 1 (run-agent) of the next round and continue the loop

```

C.6 generate-evaluator

The following prompt instructs the LLM to update the evaluation checklists based on improvement progress. This prompt implements the curriculum learning mechanism described in Section 3.4. The task definition file referenced by this prompt is also provided.

generate-evaluator

```
# generate-evaluator

A meta-skill that generates or updates evaluator/SKILL.md.

There is always only one evaluator (SKILL.md). The task definition is pre-written in task.md,
and this skill takes no arguments.

## File Structure

| Path | Role |
|---|---|
| task.md | Task definition (pre-written; this skill only reads it and does not modify it) |
| evaluator/SKILL.md | Evaluator skill body (generated/updated by this skill) |
| evaluator/CHANGELOG.md | Summary of changes per version (1 line per version) |
| evaluator/history/vN.md | Archive of old versions (automatically saved on update) |

## Determining the Operation Mode

Check for the existence of evaluator/SKILL.md:
- Does not exist -> New generation mode
- Exists -> Update mode

---

## New Generation Mode

### Step 1: Read task.md

Retrieve the following from task.md:
- Task name
- Unit of evaluation target (agent, role, team, etc.)
- Goals and evaluation perspectives per evaluation target (including victory conditions if applicable)
- List of action phases
- Types of logs and observable information available for evaluation

### Step 2: Design the Checklist

Design check items for each combination of evaluation target x phase from task.md.

Design Rules:
1. Make items answerable with Yes / Partial / No / NA granularity
   - NG: Actions that are too abstract
   - OK: Specific actions that can be directly observed from logs
2. Prioritize actions directly linked to the goals and evaluation perspectives described in task.md
3. Only adopt items that can be evidenced by observable logs
4. Aim for 5-10 items per evaluation target
5. Assign a difficulty tag to each item
   | Tag | Meaning |
   | [Basic] | Minimum required actions. Core of the initial evaluation axes |
   | [Intermediate] | Actions expected after basic items are met |
   | [Advanced] | Advanced strategies expected only from skilled agents |

Score Calculation Formula:

$$\text{Score} = ( \text{Yes} * 1 + \text{Partial} * 0.5 ) / \text{Number of items excluding NA} * 5$$


### Step 3: Generate and Save SKILL.md

---

## Update Mode

### Step 1: Read the Current SKILL.md, task.md, and CHANGELOG.md

- Understand the current version, difficulty composition, and check items from SKILL.md
- Understand the evaluation axis evolution summary from CHANGELOG.md (no need to read history/)
- Read task.md to confirm evaluation targets and log types
```

Step 2: Detect Saturation and Gaps from Scoring Results

Read the most recent 5 entries in eval_log/ and analyze scores and No/Partial items. Determine the following patterns:

Pattern	Determination	Action
An item receives Yes for a threshold number of consecutive times	Saturated	Remove that item and replace with an [Intermediate] item in the same phase
An item receives No for a threshold number of consecutive times	Unmet	Maintain the item while adding prerequisite [Basic] items
Average score across all evaluation targets ≥ 4.5	Evaluation axes are too lenient	Add [Intermediate] / [Advanced] items to increase difficulty
Average score across all evaluation targets ≤ 2.0	Evaluation axes are too strict	Remove [Advanced] items and focus on [Basic]
Only a specific evaluation target has low scores	Issues are concentrated	Subdivide and strengthen items for that evaluation target

Step 3: Update the Evaluation Axes

- Remove: Exclude saturated items
- Promote: Promote [Intermediate] items to [Basic]
- Add: Add new [Intermediate] or [Advanced] items
- Subdivide: Split items with many Partial/No results into more specific multiple items

Step 4: Archive the Old Version and Save

1. Copy the current SKILL.md to history/vN.md
2. Update version / updated / difficulty / history in the metadata at the top of the file and overwrite SKILL.md
3. Append a one-line change summary to CHANGELOG.md

When Called from self-evolve-orchestrator

When this skill completes, commit the evaluation axis changes and return to Step 1 (run-agent) of the next round to continue the loop.

Task Definition

Task Definition

A configuration file read by the generate-evaluator skill.
By editing this file, you can control the criteria for generating and updating evaluation axes.

Task Name

Werewolf Game (AIWolf NLP 5-player village)

Role List and Victory Conditions

Role	Faction	Count	Victory Condition
VILLAGER	Villager	2	Execute the werewolf
SEER	Villager	1	Execute the werewolf
WEREWOLF	Werewolf	1	Survivors equal the number of villagers
POSSESSED	Werewolf	1	The werewolf faction wins

Role Abilities

Role	Abilities / Action Constraints
VILLAGER	No special abilities. Only daytime discussion and voting
SEER	Divines any one player at night, obtaining a result of HUMAN / WEREWOLF
WEREWOLF	Attacks any one player at night, eliminating them. Pretends to be a villager during the day
POSSESSED	No special abilities. Treated as HUMAN in divination and medium results. Acts as a member of the werewolf faction to cause confusion

Game Progression

```

1. Day 0 (Greeting Phase): All players introduce themselves and exchange greetings
2. Daytime Phase: All players discuss via talk, reasoning about werewolf candidates
3. Voting: All players designate who they want to execute via vote. The agent with the most votes is executed
4. Night Phase: SEER performs divination, WEREWOLF selects an attack target. The attacked agent is eliminated
5. Steps 2-4 repeat until one of the following occurs:
    - The werewolf is executed -> Villager faction wins
    - Number of surviving werewolves >= number of villagers -> Werewolf faction wins

## Action Phases

| Phase | Description |
|---|---|
| Day 0 | First day greetings and strategy sharing |
| Daytime Discussion | Reasoning and discussion via talk |
| Voting | Determining execution target via vote |
| Night Action | divine (divination) / attack (assault) |

## Logs Available for Evaluation

| Log Type | Content |
|---|---|
| talk | Daytime utterance content |
| vote | Vote targets |
| divine | Divination target and result (HUMAN/WEREWOLF) |
| attack | Attack targets |
| execute | Executed agent and their role |
| winner | Winning faction |

```

C.7 fix-agent-error

The following prompt instructs the LLM to diagnose and fix runtime errors that occur during game execution.

```

# fix-agent-error

A skill for diagnosing and fixing errors that occur during agent execution.

## Types of Target Errors

| Category | Examples |
|---|---|
| Startup Error | ModuleNotFoundError, ImportError, configuration file deficiency |
| Connection Error | WebSocket connection failure, timeout |
| LLM Error | API call failure, response parse error |
| Game Logic Error | Invalid return values for vote/divine/attack, action skip due to exception |
| Response Missing Due to Timeout | Result becomes None even though the game did not terminate abnormally |
| Other Runtime Errors | KeyError, IndexError, AttributeError, etc. |

## About Response Missing Due to Timeout

Per the regulations, the response time limit is 1 minute. If the time limit is exceeded, that action is ignored and the agent effectively enters a dropped state.

Detection Condition: If the agent's log contains the following message, the cause is a timeout.

"Action timed out: <action name>"

Remediation Policy:
- Since the LLM's response generation may be taking too long, simplify the prompt to reduce response time
- Check whether the conversation history or input context has become too lengthy, and reduce it if necessary
- Review the retry count and wait time settings for LLM calls

```

```

## Procedure

### 1. Understanding the Error Details

Read the error message and stack trace, and identify the following:
- Error type and location where it occurred (file name / line number)
- Request type where the error occurred (TALK / VOTE / DIVINE / ATTACK, etc.)
- Reproduction conditions (whether it occurs with a specific role / day / situation)

### 2. Investigating the Cause

Read the code at the identified location and infer the root cause.
Read related files as needed:
- agent.py -- Common logic
- prompts.yaml -- Prompt format deficiencies
- Per-role .py files

### 3. Implementing the Fix

Fix the code or prompts according to the cause.
Policy when making fixes:
- Address the root cause with minimal changes
- Do not add ad-hoc try/except blocks just to avoid errors
- In the case of LLM output parse errors, review the output format instructions in the prompt

### 4. Recording the Change History (Required)

Record in improvement_log/YYYYMMDD_HHMMSS.md in the following format:

## Date and Time of Change
YYYY-MM-DD HH:MM:SS

## Error Details
(Key points of the error message and stack trace)

## Cause
(Inference of the root cause)

## Change Details

| Target File | Changed Section | Change Description |
|---|---|---|

## Handover Notes for Next Iteration
(Points to be aware of in the next loop)

## When Called from self-evolve-orchestrator

When this skill completes, re-execute Step 1 (run-agent). After fixing the error, re-run the
same game round.

```