PPOAgent for ANAC 2025 SCML OneShot Track

Shoma Mizuno
Nagoya Institute of Technology
s.mizuno.974@stn.nitech.ac.jp
Japan

June 13, 2025

1 Introduction

PPOAgent is a reinforcement learning (RL) agent developed for the ANAC 2025 SCML OneShot Track (hereafter referred to as OneShot). With the release of the RL framework in 2025, we took on the challenge of building our agent using it.

2 Agent Design

2.1 Problem definition

In order to introduce RL, we formalize the OneShot environment as a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$, where $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ is the set of all possible actions, P is a transition probability function $P(s' \mid s, a) = \Pr[S_{t+1} = s' \mid S_t = s, A_t = a]$, and R is a reward function $R(s, a, s') = \mathbb{E}[R_t \mid S_t = s, A_t = a, S_{t+1} = s']$ returning the expected immediate reward when transitioning from state s to state s' under action a.

In RL, it is crucial to define the state, action, and reward precisely. We introduce these components below.

2.2 Training purpose

The purpose of training is to produce offers that can be accepted and maximize our own profit. The profit is the change in the account balance over a day, calculated based on sales profit, various costs, and shortage penalties.

2.3 State

PPOAgent receives observations such as the following: my needs, exogenous contracts, and other relevant information. It uses these observations to decide what offers to make.

- All current incoming offers
- The currently needed quantity
- The current factory level
- The current relative simulation time
- The current disposal cost (normalized to a value between 1 and 10)
- The current shortfall penalty (normalized to a value between 1 and 10)
- The current trading price
- The current exogenous contracts

The difference from the default code is that it does not observe past offers. Most rule-based agents tend to soften their demands as the negotiation progresses. In such cases, it is assumed that agents can still adjust their own proposals to gain profit, even without observing past offers.

2.4 Action

Each agent selects four pairs of numerical values, where $Q \in \{q \in \mathbb{Z} \mid 0 \leq q \leq 9\}$ and $V \in \{0, 1\}$, representing the quantity and price of a given offer, respectively. To allocate offers to agents whose number exceeds the four selected actions, a conversion process is applied to map each selected value pair to an actual number of partners. In the implementation, we followed the provided template without modification, creating four agent groups. The offers assigned to each group are then distributed among the agents within that group. We adopted an equal division strategy for this distribution process.

2.5 Reward

The default reward function is calculated as the difference in score between consecutive days, evaluated only at the end of each day. Since this reward cannot be computed during ongoing negotiation rounds, it appears as a sparse reward function from the perspective of the reinforcement learning agent. This reward sparsity and delay make learning difficult. To address this issue, we modified the reward function used by the PPO agent to provide feedback at every action taken within a negotiation round. Specifically, we used the following equation:

$$R = \alpha \underbrace{\tanh\big(\epsilon(\mathrm{tp}(\rho, d)\Delta N_{\mathrm{my}} - C_{\mathrm{step}})\big)}_{\text{Dense reward}} + \beta \underbrace{\tanh(\eta \Delta B)}_{\text{Sparse reward}}$$

Here, $\mathrm{tp}(\rho, d)$ denotes the trading price of product ρ on the current day d, ΔN_{my} represents the reduction in the agent's own needs, i.e., the quantity successfully traded through agreements, C_{step} is the cost incurred per step (i.e., per negotiation round), and ΔB denotes the change in the agent's account balance. The parameters $\alpha, \beta \in \mathbb{R}^+$ are weighting coefficients that control the relative importance of each term, while $\epsilon, \eta \in \mathbb{R}^+$ are scaling factors used to adjust the influence of outliers.

The first term provides an immediate reward at the moment a deal is concluded, consisting of the profit from selling and a step cost based on the agent's remaining needs. The second term provides a sparse reward based on the change in the agent's account balance. This second term is only computed at the end of the negotiation day and remains zero during ongoing negotiations.

For normalization purposes, the first and second terms are passed through separate tanh functions, ensuring that the overall reward R satisfies $-(\alpha+\beta) < R < \alpha+\beta$.

2.6 Algorithm

By default, A2C was used. Although A2C enables fast learning, the learning curve showed somewhat unstable behavior. Therefore, we adopted Proximal Policy Optimization (PPO), which is also implemented in the same library, stable-baselines3 [1]. While PPO requires more time to train, it exhibited a more stable learning curve.

3 Evaluation

3.1 Training results

Training was conducted under the following settings, covering two different contexts. The number of negotiation days was randomly selected between 50 and 200 for each context. The training opponents consisted solely of DistRedistAgents. The number of training steps was 5.1×10^6 for the supplier side and 3.2×10^6 for the consumer side. The parameters of the reward function, as shown below, were kept identical across both contexts.

- The number of context: 2 (supllier/consumer)
  - The number of total day: (50,200)
  - Training opponents: DistRedistAgent
- The number of training steps: (supllier/consumer)=$(5.1 \times 10^6/3.2 \times 10^6)$
- The parameters of the reward function: $(\alpha, \beta, \epsilon, \eta) = (\frac{1}{3}, \frac{2}{3}, 50, 60)$

The training results are shown in Figure 1 and Figure 2. Figure 1 indicates that the learning process has converged, while Figure 2 shows that the mean number of negotiation rounds is higher for the supplier side compared to the consumer side. The reason why the episode length varies across different contexts remains unclear.

Furthermore, while the supplier side exhibits instability in the reward signal, the consumer side shows early convergence. This may be attributed to the inclusion of a step cost in the reward function, which causes the total reward to vary in proportion to the length of each episode.

3.2 Test results

3.2.1 vs 2023 RLAgent

We conducted matches against a reinforcement learning agent proposed in 2023. The tournament was executed with the parameter settings shown below.

- The number of different world configurations: (`n_configs`)=100
- The number of simulation days: (`n_steps`)={50, 100, 200}
- The number of runs per world: (`n_runs_per_world`)=1

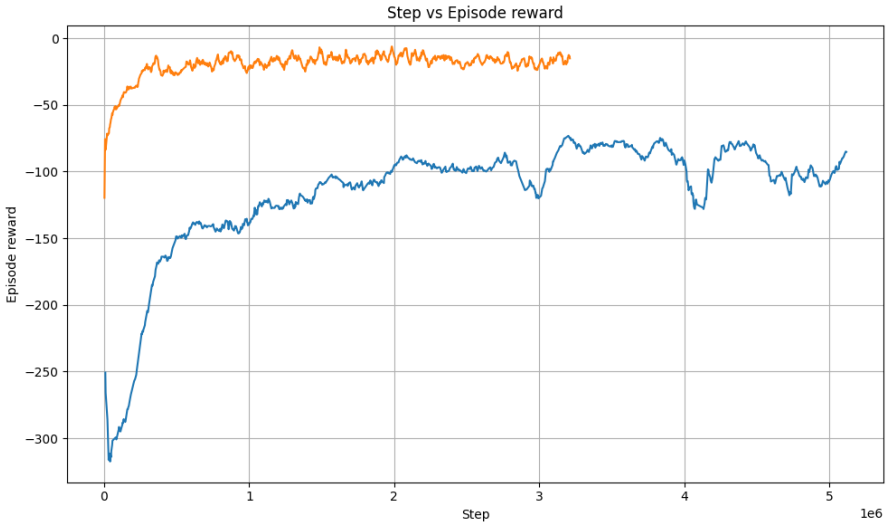

Figure1　Change in mean episode reward(Blue line is supplier. Orange line is consumer.)

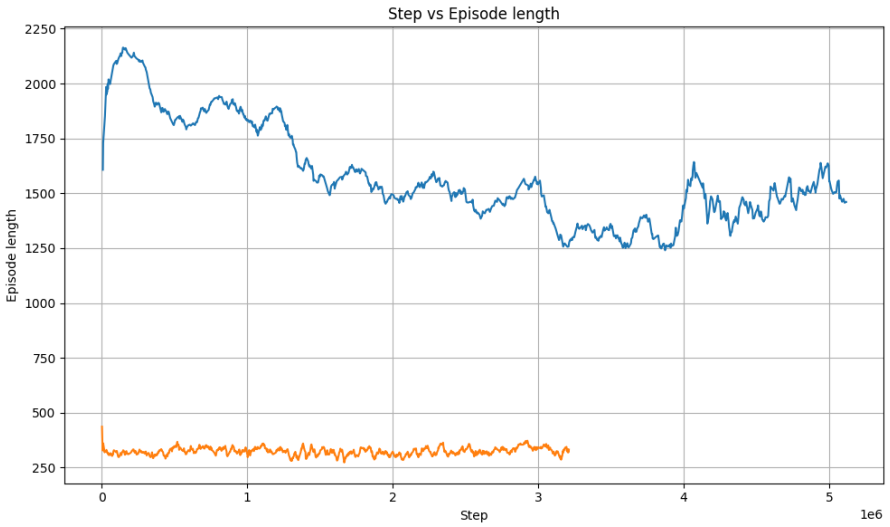

Figure2　Change in mean episode length(Blue line is supplier. Orange line is consumer.)

- The number of competitors per world:
  (`n_competitors_per_wordld`)$\in \{m \in \mathbb{N} \mid 2 \leq m \leq \min(4, n)\}$, n is the number of competitors.
- Competing agents: {PPOAgent, RLIndAgent[*1]}[*2]

From these results, it can be observed that PPOAgent slightly outperforms the previously proposed reinforcement learning agent. However, both agents tend to incur losses in most cases.

[*1] Unlike PPOAgent, RLIndAgent is an asynchronous agent. However, it was selected for evaluation since it outperformed RLSyncAgent (developed by the same team) in their official report.

[*2] In accordance with the provided codebase, the remaining agents are randomly selected from the following three: GreedyOneShotAgent, EqualDistOneShotAgent, and RandDistOneShotAgent.

Table1 Score results against reinforcement learning agents (n_step=50)

Agent name	mean	std	min	25%	median	75%	max
PPOAgent (ours)	**0.80494**	0.19369	**0.22642**	0.64301	**0.87170**	**0.96526**	1.09833
RLIndAgent	0.79873	0.19065	0.21704	0.64455	0.84718	0.95537	1.10145

Table2 Score results against reinforcement learning agents (n_step=100)

Agent name	mean	std	min	25%	median	75%	max
PPOAgent (ours)	**0.84633**	0.18296	0.27998	**0.76324**	**0.91537**	**0.98290**	1.09718
RLIndAgent	0.84043	0.18221	0.28405	0.76169	0.88823	0.97136	1.13941

Table3 Score results against reinforcement learning agents (n_step=200)

Agent name	mean	std	min	25%	median	75%	max
PPOAgent (ours)	**0.83522**	0.18487	**0.20061**	**0.71411**	**0.90097**	**0.97779**	**1.12471**
RLIndAgent	0.82396	0.17976	0.19408	0.71184	0.87762	0.95902	1.11548

3.2.2 vs 2024 3rd Agent

We conducted matches against DistRedistAgent, which ranked third in last year's competition. The tournament was executed with the following parameter settings.

- The number of different world configurations (`n_configs`): 10
- The simulation days (`n_steps`): 100
- The number of runs per world (`n_runs_per_world`): 1
- The number of competitors per world:
  (`n_competitors_per_wordld`)$\in \{m \in \mathbb{N} \mid 2 \leq m \leq \min(4, n)\}$, n is the number of competitors.
- Competing agents: {PPOAgent, RLIndAgent, GreedyOneShotAgent, DistRedistAgent } [*2]

Table4 Score Results Against Last Year's Third-Place Agent(n_step=100)

Agent name	mean	std	min	25%	median	75%	max
PPOAgent (ours)	0.82882	0.19283	0.34221	0.67960	0.89810	0.99040	1.09539
RLIndAgent	0.82558	0.18866	0.33986	0.67907	0.89516	0.98227	1.07348
GreedyOneShotAgent	0.77497	0.18957	0.25116	0.66037	0.83845	0.93534	1.03259
DistRedistAgent	1.06712	0.05505	0.92459	1.04044	1.07480	1.10133	1.18803

From these results, it can be observed that the reinforcement learning agents exhibit a score distribution similar to that of the GreedyOneShotAgent. Although PPOAgent achieves a higher mean score than GreedyOneShotAgent, its mean score is lower compared to DistRedistAgent. In terms of standard deviation, PPOAgent shows the highest variability, with a value more than three times greater than that of DistRedistAgent. Moreover, the minimum score of PPOAgent is significantly lower than that of DistRe-

distAgent. These findings indicate that, while PPOAgent shows some improvement over simpler agents, there remain challenges in both improving and stabilizing performance compared to DistRedistAgent.

4 Lessons and Conclusions

PPOAgent outperformed both RLIndAgent and GreedyOneShotAgent in terms of mean score by improving the observation space and reward structure from the default settings of the reinforcement learning framework. However, its mean score remains lower than that of DistRedistAgent, one of the top-performing agents in the 2024 competition. Similar to the findings in [2], these results highlight remaining challenges for agents based on reinforcement learning.

One potential cause of this performance gap is the perceptual aliasing problem, as well as the tendency of reward-driven agents to get stuck in local optima. In the OneShot environment, agents cannot observe the demand of their negotiation partners or the negotiation history between other agents, which easily leads to partial observability. To address this, incorporating separate modules for estimating the opponent's state or strategy—outside of the reinforcement learning process—and providing those estimates as part of the observations may lead to improved performance.

Moreover, given the dense reward setting and the on-policy nature of PPO, it may be difficult for agents to optimize their long-term account balance. To mitigate this issue, approaches such as integrating rule-based decision-making or utilizing imitation learning may offer potential improvements.

References

[1] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[2] Isaac Brown, Kevin Zhou, Soichiro Sato, and Soto Hisakawa. Automated negotiation for supply chain management: Final report. g-RIPS Sendai 2023, 2023.