One Page Extended Abstract

Motivation Social deduction games like Mafia, Werewolf, and Among Us challenge players to infer hidden roles using limited information, deception, and social reasoning. These games provide a rich environment for studying reinforcement learning in multi-agent settings involving uncertainty, collaboration, and strategic misdirection. Our work explores how RL agents can learn and adapt in such environments by reasoning about incentives, payoffs, and uncertainty. This has broader applications in domains like negotiation, where agents manage conflicting goals, and information security, where detecting deception is critical.

Method and Implementation We implemented a social deduction reinforcement learning pipeline using PettingZoo. Our agents, using PPO and MADDPG, were trained in the games of Mafia and Spyfall. Below we summarize our policy implementations and environment setups. PPO (Proximal Policy Optimization) optimizes the clipped surrogate objective:

$$\mathcal{L}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 \pm \epsilon)\hat{A}_t \right) \right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, $\hat{A}_t$ is the estimated advantage, and ϵ is the clipping parameter. We compute returns backward through the trajectory and subtract value estimates from the model.MADDPG (Multi-Agent DDPG) uses centralized critics $Q_i(x, a_1, \ldots, a_N)$ and decentralized actors $\mu_i(o_i)$. The gradient for agent i is:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{x,a\sim\mathcal{D}} \left[\nabla_{\theta_i}\mu_i(o_i) \nabla_{a_i} Q_i(x, a_1, \ldots, a_N) \right]$$

For our environments: Mafia is a hidden-role game where Mafia eliminate villagers at night, and players vote during the day. Players can take special actions (e.g., shielding) in our variations of the game. The action space is a one-hot vector over n players and m additional actions. Observations include vote history and public actions. Spyfall is a game where all players except one (the spy) are given a location, and players take turn asking each other questions to determine who the spy is, all while not giving away too much information to reveal the true location, which the spy is trying to guess. We trained both PPO and MADDPG agents on the Mafia and Spyfall environments across varying numbers of players. Agents were evaluated in tournament-style settings against opponents with different training durations, including untrained (random) agents. Most experiments were run for 50,000 training steps. This value, inspired by earlier assignments, was sufficient for convergence based on reward stability and loss behavior. We did not extensively tune this hyperparameter. Learning rates were selected via a basic grid search over powers of ten, and no exhaustive search was conducted for architectural parameters such as network size. Smaller networks and shorter runs were used during debugging, while larger networks were reserved for final experiments. Since we were not optimizing model size or epoch count, we allowed variation during early development phases without compromising experimental validity. For evaluation, we tested trained agents against random baselines to assess learning progress. We also tested them in environments populated by other trained agents to analyze behavioral adaptation in the presence of skilled opponents. To better qualitatively analyse learned policies, we logged and inspected agent action histories.

Results Our MADDPG and PPO agents exhibited higher winrates than other less trained agents, and achieved stable, convergent loss on all datasets. MADDPG slightly outperformed PPO, and qualitative analysis shows some degree of intelligent gameplay. See results section for more.

Discussion, Conclusion, and Next Steps Overall, our PPO and MADDPG approaches seemed to yield strong results, showing significant winrate and reward improvements over time in each of the game environments, with solid results against untrained agents and other trained agents, as well as qualitatively intelligent strategies. Trained agent vs. trained agent tournaments and agent vs. random choice policy tournaments both showed learning, and convergence of both winrate and loss. One improvement that can be made for future work is to add hindsight relabeling on top of making the available vocabulary magnitudes of order larger, especially since these environments have generally sparse reward structures, and it could find some more signal to learn off of. A natural future direction of this research would be to incorporate LLMs into the decision-making process to guide certain aspects of communication and deduction, and to measure against and train against human competition and to leverage human feedback. See relevant sections for more.

Towards a General Social Deduction Reinforcement Learning Model

Isaac Zhao
Department of Computer Science
Stanford University
ikezhao@stanford.edu

Wesley Tjangnaka
Department of Computer Science
Stanford University
wesleytj@stanford.edu

Abstract

Social deduction games (such as Mafia, Werewolf, or Among Us) are multiplayer games where potentially adversarial participants must use limited information, social cues, and strategy to determine hidden roles within the group. We investigate how reinforcement learning agents can learn to act and adapt in these games, where deception, communication, and collaboration are key components. We developed modified versions of the popular social deduction games Mafia and Spyfall, and we implement and evaluate PPO and MADDPG agents in these environments. Our agents exhibit improved win rates across various player counts, especially in environments with other trained agents. MADDPG consistently outperforms PPO against random agents by leveraging centralized critics during training, although both approaches demonstrate improvement over structured random agents. In a direct head-to-head environment pitting PPO and MADDPG agents against each other, we see that MADDPG continues to outperform PPO. Our findings highlight the ability of multi-agent RL to model complex interactive behaviors, and lay the groundwork for potential extensions with dynamic language-based communication in these socially grounded environments.

1 Introduction and Related Work

Social deduction games (such as Mafia, Werewolf, or Among Us) are multiplayer settings in which participants must infer hidden roles based on limited information, social cues, and strategic reasoning. These environments present unique challenges involving deception, collaboration, and natural language communication, making them compelling testbeds for research in reinforcement learning and multi-agent systems.

Our work explores how reinforcement learning can be applied to both learning and decision-making in social deduction games. These environments require agents not only to reason under uncertainty but also to engage in dynamic social interactions involving trust, influence, and strategic misdirection. Understanding how agents can adapt their strategies and reason about incentives in such settings has potential applications across several domains. In automated negotiation, for instance, agents must reconcile partial information with conflicting goals—much like players in a social deduction scenario. In information security, the modeling of adversarial behavior and insider threats mirrors the need to detect and counter deception. And in the study of group dynamics, such games offer a framework for analyzing how influence, trust, and strategies evolve in multi-agent settings over time.

This line of research bridges sequential decision-making, multi-agent reinforcement learning, and natural language reasoning, offering a foundation for building agents that interact in complex, human-like social environments.

1.1 Related Work

Recent work in RL has explored various environments where agents must reason about partial information, but few have directly tackled the challenge of social deduction games. Traditional RL applications focus more on games like chess, Go, or even multi-agent environments like competitive games (e.g., StarCraft II). However, social deduction games introduce a unique complexity: the need for agents to infer hidden information, deal with uncertainty, and collaborate or deceive. Previous attempts at simulating social deduction (such as using supervised learning or rule-based strategies) have been limited by the inability of agents to autonomously learn to interact with human-like uncertainty.

An relevant early paper that discussed RL in the context of multiplayer, imperfect information games was Heinrich et al. (2015) in which they developed a Q-learning algorithm to estimate the Nash equilibrium of a simplified games of hold'em poker Heinrich and Silver (2016). This technique was powerful enough to reliably approach the Nash equilibrium for solved versions (like Leduc hold'em), as well as perform at the caliber of top poker engines for unsolved versions like limit Texas hold'em. However, this did not cover the social aspect of deduction, as the deductive steps in poker are less verbal and more analytic due to an understanding of seen and unseen information.

Focused more specifically in the domain of social deduction, Sarkar et al. (2025) trains a RWKV language model to take more effective actions in the deduction game of Among Us, and employs proximal policy optimization (PPO) to help the bot learn effective "listening" to discern the imposter, and effective "speaking" to sway the opinion towards the correct imposter Sarkar et al. (2025). By adding this additional step to learn the speaking and listening specifically, this policy gives a win rate that is double that of the traditional RL, in which it attempts to only optimize winning or losing a game. These results demonstrate the power of RL for improving natural language communication in social deduction, serving as a potential point of comparison.

More generally, Lowe et al. (2020) proposed MADDPG, a novel multi-agent policy gradient algorithm of learning centralized critics, or critics based on the observations and actions of all agents, which enhances its ability to make strong cooperative and competitive decisions Lowe et al. (2020). This allows agents to perform strongly on complex communication tasks, such holding a "coded" conversation with a rival listening, attempting to decode it. The applications of this apply directly into building our social deduction bot, as it needs to optimize both deceptive and coopoerative communication.

Together, these lines of work suggest that social deduction games offer a rich environment for studying the intersection of reinforcement learning, communication, and theory of mind. Our goal is to push this frontier by developing agents that can learn to interact, deceive, and infer intent in open-ended, linguistically grounded settings.

2 Method

We followed our project proposal's technical outline and implemented a social deduction RL pipeline using PettingZoo. We have implemented a PPO agent and an MADDPG agent for the games of Mafia and Spyfall. We will explain the way we implemented the policies, as well as the rules of the games and environments.

2.1 PPO and MADDPG Policies

1. **PPO Agent:** Proximal Policy Optimisation Schulman et al. (2017) seeks to optimise policies by maximising the expected value of the minimum of the clipped and unclipped versions of the probability ratio between the new and old policy after weighting by the advantage. We calculate the advantage by taking the returns through the trajectory (computed backward) and subtracting the value estimates (output by the model). The objective is therefore:

$$L(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)\hat{A}_t, 1 \pm \epsilon))]$$

The various components of this are:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio,

- $\hat{A}_t$ is the advantage estimate, which we are using a simple formula for (more complex choices like GAE exist).
- ϵ controls clipping to limit policy updates.

2. **MADDPG Agent:** Multi-Agent Deep Deterministic Policy Gradient trains critics that are shared across agents, and these critics use the joint observations of all agents. Each agents actions and their joint observations are used to calculate a "Q-value": $Q_i(x, a_1, a_2, ...)$ conditioned on this joint state. The objective is below:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{x,a\sim D}\left[\nabla_{\theta_i}\mu_i(o_i)\cdot\nabla_{a_i}Q_i(x, a_1, \ldots, a_N)\Big|_{a_i=\mu_i(o_i)}\right]$$

Breaking this down into its different components:

- $\nabla_{\theta_i} J(\theta_i)$: Gradient of agent i's policy objective w.r.t. actor parameters.
- $\mathbb{E}_{x,a\sim D}[\cdot]$: Expectation over joint states x and actions a from replay buffer D.
- $\mu_i(o_i)$: Agent i's deterministic policy (actor) given its observation o_i.
- $\nabla_{\theta_i}\mu_i(o_i)$: How the actor's output changes with its parameters.
- $\nabla_{a_i}Q_i(x, a_1, \ldots, a_N)$: Sensitivity of the centralized critic's Q-value to agent i's action.
- $a_i = \mu_i(o_i)$: Actions taken by agent i according to its current policy.

2.2 Environments

1. **Mafia:** A game of hidden roles where a small group (the Mafia) attempts to eliminate other players (the villagers) at night, while during the day all players vote to eliminate suspected Mafia. There are some additional actions that can be taken, such as using a shield, which can block being eliminated by the Mafia, or survive an elimination vote. The action space is a one-hot encoded vector of size $n + m$, where n corresponds to the number of players to be voted out and m represents the additional actions (such as shielding) that the players can take. The observation space includes voting history, and actions that are public (like shielding as opposed to Mafia eliminations).

2. **Spyfall:** All players are given a location (e.g., "Library") except one (the Spy) who does not know the location. Players ask each other questions to identify the Spy, while the Spy must deduce the location without revealing themselves. We use a modification of this game, where each player goes around saying a word relating to the location rather than asking questions. A word bank is provided for computational tractability. The action space is discrete with size $w + n + l$, where w is the number of words that can be spoken, $m$the number of players to vote for, l the number of locations the spy can guess. The observation space includes current location (unknown to the spy), round and phase details, words spoken by each player (as a matrix), suspicion scores (agent-learned or environment-based), location likelihood estimates for the spy, voting history, and a word quality scores for suspicion learning/location inference.

3 Experimental Setup

For each of the environments, we trained both the PPO and MADDPG agents across both the Mafia and Spyfall environments with varying number of players in the game ($n = 3, 5, 7$). We first trained the agents were trained in tournaments against other untrained (ie. random) opponents.

For these experiments training against random opponents, we ran the training for 50,000 steps (we did not attempt to optimize on this hyperparameter particularly much, and the choice was in line with our homework, where 50,000 steps was used as a guideline). For our PPO agent, we used a learning rate of $\eta = 3 \times 10^{-4}$, and for our MADDPG agent, we used a learning rate of $\eta = 1 \times 10^{-3}$. For MADDPG, we also used set the target network update rate $\tau = 0.01$ and the discount factor $\gamma = 0.95$. We observed generally that at 50,000 steps the results were generally stable, and that the loss curves and rewards seemed to have converged respectively, so this seemed like a choice that did not have to be changed. Similarly, we did not attempt to optimize on the learning rates beyond a simple grid search across rates increased or diminished by powers of 10, nor did we focus our efforts on exploring other architectural hyperparameters such as the size of our networks, save for the fact that, rather naturally, we used larger networks and full training steps for our final results and smaller networks for debugging and sanity checking during the quarter, as is usual. For the other parameters on our MADDPG agent, we followed the suggested values from Lowe et al. (2020).

Specifically for the Spyfall environment, we also trained both the PPO and MADDPG agents to play in an environment against a mix of itself as well as random opponents. We set up multiple environments with $n = 4, 5, 7$ players, and we evaluated this both in its same environment, as well as head to head in environments that mixed both agents. For these agents, we trained them in the environments that contained $\lfloor \frac{n}{2} \rfloor$ trained agents, and the rest were trained agents. We ran the training for 100,000 steps, and we used the same learning rates, target network rate, and discount factor as the previous experiments.

For evaluation, we first ran evaluations on the performance of the two algorithms trained against random agents, which we performed on both the Mafia and Spyfall environments, as shown in Table 1. Then, to explore more advanced learning, we focused on just the Spyfall environment for the remainder of the experiments, due to the fact that there are more actions and complex decisions versus the Mafia environment. To explore how the agents performed against themselves and learn against other "skilled" players, we trained them in environments with multiple other trained agents of the same algorithm, while still mixing in a few random opponents (Table 2). Finally, we put them in an environment with a mix of different types of trained agents and random opponents (Table 3). Specifically, we would put equal numbers of PPO and MADDPG agents in the environment, and also roughly the same number of random agents as total trained agents. At various points, we tried to observe the patterns of play by the policies by examining the record of actions undertaken by the agents, which we had logged during runs.

4 Results

4.1 Quantitative Evaluation

Environment	Random	PPO	MADDPG
Mafia ($n = 3$)	45.6%	**52.0**%	51.4%
Mafia ($n = 5$)	42.1%	46.3%	**46.5**%
Mafia ($n = 7$)	34.6%	38.0%	**38.3**%
Spyfall ($n = 3$)	47.8%	52.5%	**53.3%**
Spyfall ($n = 5$)	42.8%	46.2%	**46.5%**
Spyfall ($n = 7$)	46.1%	**48.8%**	48.7%

Table 1: Winrates of agents on Mafia and Spyfall environments versus $n - 1$ random opponents. Trained against random agents. $100,000$ iterations

In 1, when trained and evaluated against all random opponents, we see that PPO and MADDPG both outperform the random policy, though only by around 5%. In general, PPO and MADDPG perform roughly the same, with MADDPG performing better by fractions of a percentage point.

Environment	Avg. Rew/Win (random)	Avg. Rew/Win(agent)
Spyfall PPO ($n = 4, n_a = 2$)	-0.34 / 31%	0.14 / 47%
Spyfall MADDPG ($n = 4, n_a = 2$)	-0.76 / 35%	0.38 / 57%
Spyfall PPO ($n = 5, n_a = 2$)	-0.3467/36.66%	0.3/46%
Spyfall MADDPG ($n = 5, n_a = 2$)	-0.3 / 41%	0.23 / 50%
Spyfall PPO ($n = 7, n_a = 3$)	0.05 / 47.5%	0.16 / 55%
Spyfall MADDPG ($n = 7, n_a = 3$)	-0.45 / 33.75%	0.48 / 43.33%

Table 2: Average Reward (rew) and Winrate (win) of training PPO/MADDPG on Spyfall environments with n_a agents and $n - n_a$ random opponents, run for $100,000$ iterations.

To see more significant learning, we pitted these trained agents against other trained agents. Evidently, both the PPO and MADDPG, when training with each other, now significantly outperform the random opponents by around 10%, and generate higher average returns when training too. The difference between MADDPG and PPO is more noticeable, with MADDPG performing much better than PPO in the all environments except the one with $n = 7$ players.

Environment	**Random**	**PPO**	**MADDPG**
Spyfall ($n = 4, n_{\text{PPO}} = 1, n_{\text{MADDPG}} = 1$)	34.7%	50.2%	**50.3%**
Spyfall ($n = 5, n_{\text{PPO}} = 1, n_{\text{MADDPG}} = 1$)	40.7%	35.9%	**51.3%**
Spyfall ($n = 7, n_{\text{PPO}} = 2, n_{\text{MADDPG}} = 2$)	28%	31.7%	**39.6%**

Table 3: Winrates of a mix of PPO, MADDPG, and random agents in Spyfall environment. Averaged across $10,000$ iterations.

Finally, we had the agents play against each other in different environments, which was where we saw the MADDPG agents shine the most. These agents significantly outperformed both the random and PPO agents in the larger settings ($n = 5, 7$), and the MADDPG and PPO performed similarly in the $n = 3$ setting. Interestingly, we see that though random performs the worst by a fairly significant margin in most environments, it actually slightly outperforms the PPO in the $n = 5$ environment.

Note that for all experiments, winrates do not add up to 100%. This is due to the fact that in both Spyfall and Mafia, there is only 1 Spy/Mafia and $n - 1$ non-spies and non-Mafia. At the end of each game, if the Spy/Mafia win, they records 1 win while everyone else records a loss; similarly, when non-spies/non-Mafia win, $n - 1$ wins are recorded.

4.2 Qualitative Analysis

In Figure 1, we see the steady rising of the agent rewards in the Mafia environment for different size games, steadily rising throughout the duration of the training. In contrast, Figure 2 shows a quick plateau of rewards at 0 for the Spyfall environment, with a generally steady win rate.

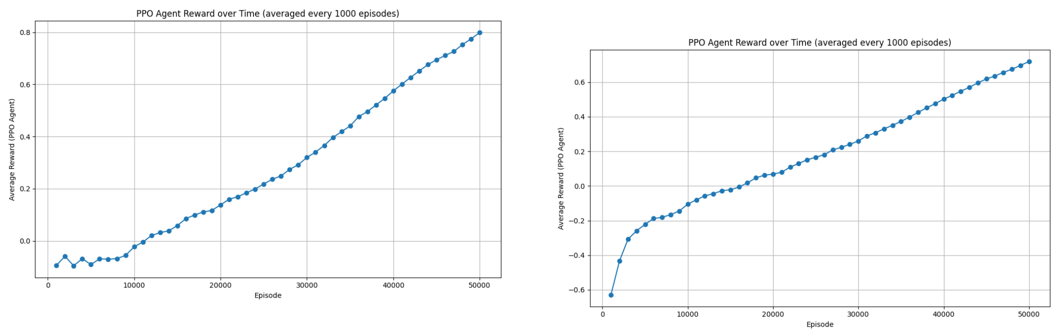

Figure 1: PPO Agent Training Average Reward (Mafia, $n = 3, 5$)

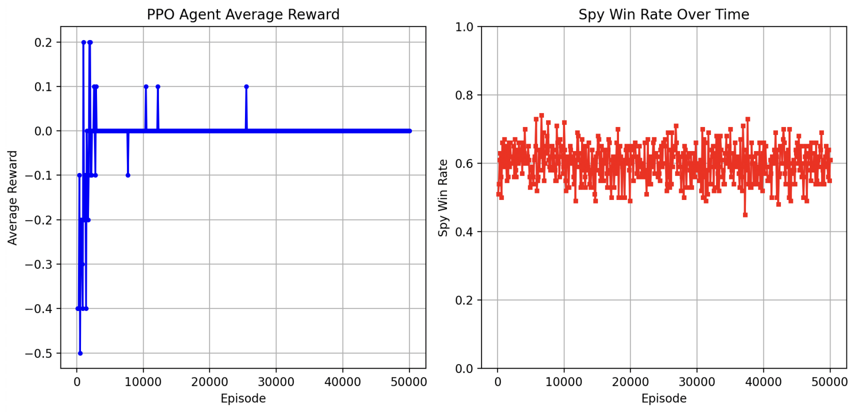

Figure 2: PPO Agent Training Average Reward and Win Rate (Spyfall, $n = 5$).

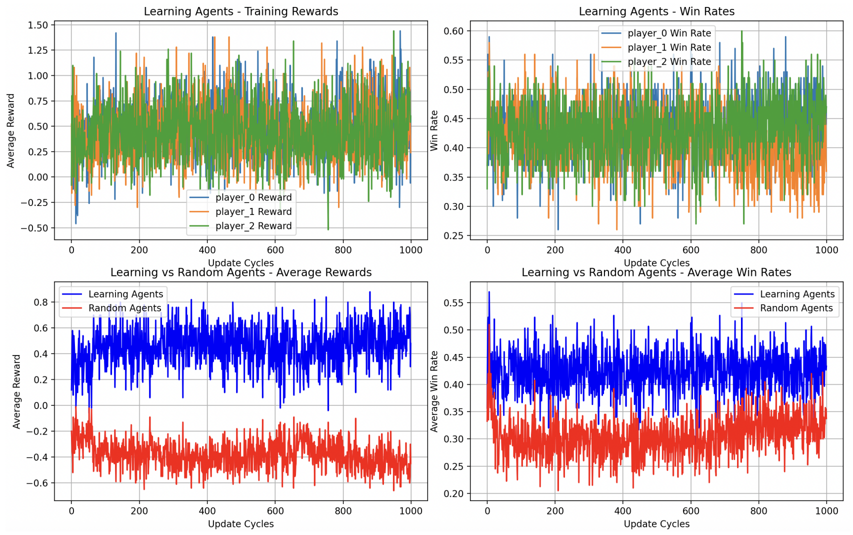

Figure 3: Training $n_m = 3$ MADDPG agents with $n = 7$ total players on Spyfall environment.

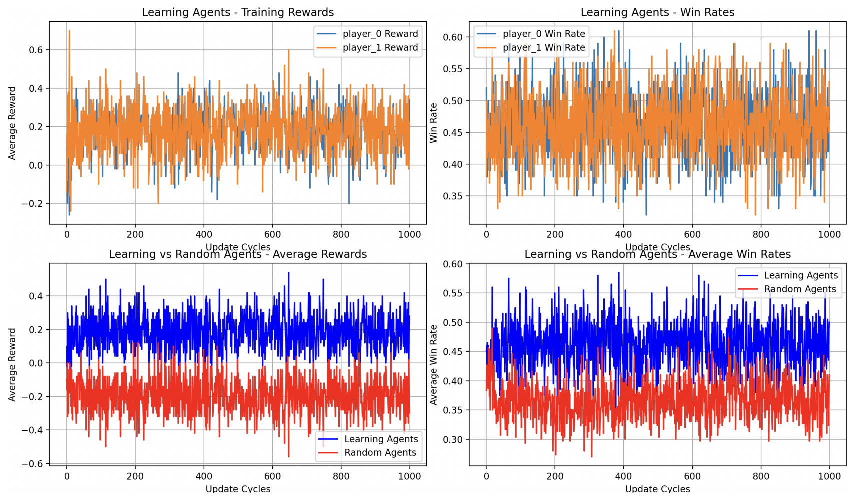

Figure 4: Training $n_m = 2$ MADDPG agents with $n = 5$ total players on Spyfall environment.

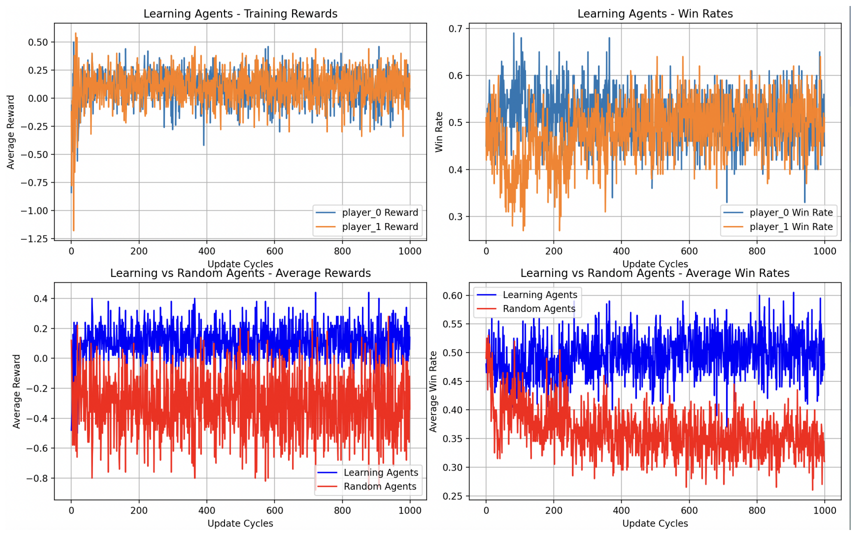

Figure 5: Training $n_m = 2$ PPO agents with $n = 4$ total players on Spyfall environment.

Overall, we see that the Spyfall rewards generally plateau quickly, even in different cases. As shown in Figures 4, 3 and 5, it is evident that though the rewards stay generally stagnant, there is a very noticeable increase in winrate for the trained agent versus the random, and the difference is quite significant as well.

Examining the gameplay of the agents, we also recognize signs of "intelligent" gameplay. For MADDPG and PPO agents, we see that they tend to choose less obvious words at the beginning - for instance, if the location was "hospital," they would start off with a word like "busy," which is general and could apply to other locations. From there, they would slowly use all of the ambiguous words until forced to choose a word that is more obvious. As a spy, they followed the same policy, but would guess the location as soon as a word with more signal was revealed, such as "emergency."

5 Discussion

Overall, we notice that both PPO and MADDPG provide improvement over a random policy, and though MADDPG performs better than PPO in most cases, the difference is very marginal. We believe that this is due to the fact that MADDPG can leverage the joint observations of all agents for its critic, which gives it a particular advantage in a multi-agent setting. However, since both agents use decentralized policies during execution, this advantage does not always translate into a significant advantage at test time. This may explain why the performance difference is positive but marginal despite MADDPG's theoretically more informative signal. These results could also be likely due to the sparsity in the environment: it can be hard for both agents to find significant signal to latch onto, though it still learns and improves its reward gradually, and so even agents with an edge may struggle to pull ahead of other agents, due to the difficulty of the task. In addition, we observed that from a qualitative perspective, agents learn more theoretically optimal strategies against other agents than against the fixed random opponents, where there is less of a learning signal for complex strategies to develop.

At various points, we tried to observe the patterns of play by the policies. Qualitatively, it is hard to summarize the general improvements of the learned policies as a direct description of strategy for the Mafia agents, save for the fact that overall it seems that the best performed trained agents would perform most of the (generally beneficial) 'bonus actions' as frequently as were permitted, and they would learn to not vote unintelligently (for instance, they would learn quickly not to vote themselves out). Other, more complex voting patterns were hard to find, and we ran out of time during the quarter to more seriously undertake an analysis of this aspect of the game. The use of bonus actions was generally well done - for instance, shielding earlier in the game, as there is no guarantee of surviving later into the game.

It is interesting that in the Spyfall environment, the agents all quickly converge to a reward early on, which is somewhat expected given the simplifications we made to make the environment trainable. Though having an additional language model to play this game and training it would have been too computationally intensive, it seems that our simplification of giving a word bank made it such that it required an early random assignment to perform well, since it could use the less obvious words before being taken by the opposing group. As a result, though we see improvements with PPO and MADDPG, it is not very large.

6 Conclusion

Overall, our PPO and MADDPG approaches seemed to yield strong results, showing significant winrate and reward improvements over time in each of the game environments, with solid results against untrained agents and other trained agents, as well as qualitatively intelligent strategies. One improvement that can be made for future work is to add hindsight relabeling on top of making the available vocabulary magnitudes of order larger, especially since these environments have generally sparse reward structures, and it could find some more signal to learn off of. A natural future direction of this research would be to incorporate LLMs into the decision-making process to guide certain aspects of communication and deduction, and to measure against and train against human competition and to leverage human feedback.

7 Team Contributions

- **Isaac:** Worked on creating the PPO agent and MADDPG agent for Mafia, and setup the infrastructure. Equal contribution.
- **Wesley:** Worked on creating the PPO agent and MADDPG agent for Spyfall, and setup the infrastructure. Equal contribution.

Changes from Proposal: No significant changes - but we were originally a team of three and we did lose one team member in Arnav Mehta (due to an unfortunate injury), and he had to drop the class early on. As a result, we did not do the LLM stuff he was originally excited to work on.

References

Johannes Heinrich and David Silver. 2016. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. (2016). arXiv:1603.01121 [cs.LG] `https://arxiv.org/abs/1603.01121`

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2020. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. arXiv:1706.02275 [cs.LG] `https://arxiv.org/abs/1706.02275`

Bidipta Sarkar, Warren Xia, C. Karen Liu, and Dorsa Sadigh. 2025. Training Language Models for Social Deduction with Multi-Agent Reinforcement Learning. arXiv:2502.06060 [cs.AI] `https://arxiv.org/abs/2502.06060`

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] `https://arxiv.org/abs/1707.06347`