Multi-Agent RL for Unidentified Adversaries and Collaborators

Aditya Nair[*]
EECS Undergraduate Student
University of California, Berkeley
ads.nair@gmail.com

Mrunal Puram[†]
EECS Undergraduate Student
University of California, Berkeley
mpuram@berkeley.edu

Abstract

Often collaboration and adversarial behaviors are observed in isolation with known identities, whereas quite a few real world and social deduction game environments involve unknown adversaries and both collaborative and adversarial behavior. We introduce *Bayesian Democracy*, which involves using Bayesian Inference to better understand what agents are adversaries and what agents are collaborators. Using this method, we were able to find a better than chance way to provide information to a collaborator about potential adversaries by only observing actions taken in a particular radius around the collaborator.

1 Extended Abstract

We explored the field of Multi-Agent Reinforcement Learning for this project, and applied it to the various aspects of the game Among Us. We built upon prior work in this area, especially work relating to multi-agent cooperation done by Kleiman-Weiner et al. [2016], methods similar to Bayesian Delegation used to study cooperative games such as Overcooked explored by Wang et al. [2020], and work on adversarial behavior as done by Gleave et al. [2020].

Initially, we wanted to deal with a "hide and seek" reduction of the game: Have hiding agents and a seeking agent that play a variation to the game where the impostor must kill all the crewmates before they finish their tasks, but the impostor has low vision (low observation radius) and the crewmates have high vision. Our initial plan was to build off of methods outlined in the Overcooked implementation, and test out the effectiveness of Bayesian Delegation. However, the website and code had been suddenly removed from public access.

We figured out that there were more options for environments and algorithm implementations that better suited our goal of exploring multiagent RL for an Among Us-style game. Environments we attempted to explore included gym-minigrid, gym-multigrid, as well as more generic OpenAI environments like multiagent-particle-envs and multiagent-emergence. We decided to use OpenAI's MADDPG [3] as the training algorithm, and were able to get it working with two different environments.

For the first part of our project, we used a combination of MADDPG and multiagent-particle-env. For this environment, we achieved seven iterations. First, a baseline that tested if we could apply the maddpg algorithms to the multiagent-particle-env environment. Since the reward function only accounted for distance to goal landmarks, we modified it so that the distance between agent and adversary was taken into account. We modified the environment/agents to allow kills, one single unique goal landmark for each agent, and a reward that better reflected survival and landmark count. Our third iteration added multiple unique goals for each agent, a terminal state, and one-time rewards that didn't propagate through the rest of the episode. While the third iteration included a mix of

[*] github.com/AdityaNair99
[†] github.com/MrunalPuram
[3] https://github.com/openai/maddpg

one-time and persistent rewards, our fourth iteration explored performance when the rewards are either all persistent or all one-time only. For the fifth iteration, we explored whether scaling rewards, or penalties, for distance to landmarks would improve performance. For the sixth iteration, we decided to explore the differences in performance when using DDPG vs MADDPG for training either the adversary or the agents. For the seventh iteration, we implemented a kill cooldown for the adversary to better mimic the mechanics of the game.

Our experiments in the multiagent-particle-env gave us quite a strong starting point to launch the gridworld exploration, by showing us not only what mechanics we should continue to explore but also what mechanics we should and should not keep in an implementation.

For the second part of our project, we created our own gridworld style environment for Among Us, using the PredatorPrey implementation in ma_gym as a starting template (our code can be found here). We made significant changes and rewrote major portions to create an environment that implemented the game mechanics of Among Us: crewmates that attempt to find the task locations, and impostors that attempt to kill crewmates. An important change we introduced is that the crewmates do not know which of the other agents is the impostor, a significant difference from the previous environment. Furthermore, this environment is only partially observable. We also added mechanics like a kill cooldown and revamped the rewards for finishing tasks or killing crewmates to account for either being a one-time lump sum reward or a persistent reward. For the baseline, we observed that the crewmates ended up winning most of the games after passing 5000 iterations. We ran similar experiments to the previous section, such as comparing the performance of random initialization, using persistent rewards vs one-time rewards, and using DDPG instead of MADDPG to train.

Our experiments in the gridworld environment provided us more insight into different reward mechanics and further validated the result that MADDPG performs better for multiple cooperating agents.

For the third part of our project, we attempted to implement the voting mechanic from Among Us, where crewmates can decide to kill the impostor if they reach a consensus. To do so, we used Bayesian Inference by using a simple two Flipout layer network to estimate the posterior representing the probability an agent believes another agent is an adversary. We began with De Facto leader voting to test if were performing Bayesian Inference properly, and ended with a Majority Rules style voting, with both showing results that exceeded pure chance, a method that we called *Bayesian Democracy*.

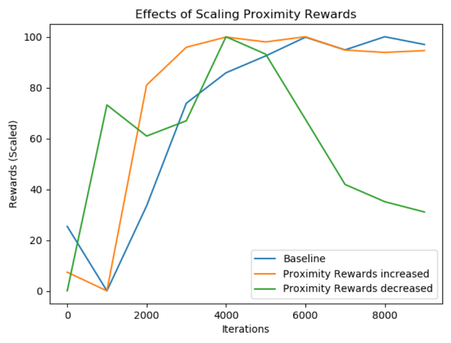
Along with our contributions towards an Among Us-style environment that can be used for future experiments, we felt we made valuable attempts at integrating Bayesian Inference with MADDPG and at better understanding the impact of different reward systems on training speed and quality.

2 Introduction

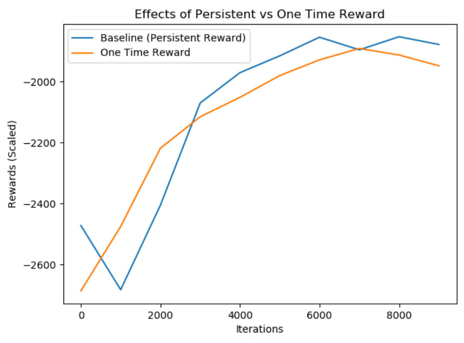
The inspiration for this project comes from the current viral game called Among Us, a game where players are randomly assigned to be either a regular “crewmate” or an "impostor". The identities of each player are kept secret, and the goal of the impostor is to kill off all the crewmates, while the goal of the crew-mates is to deduce who amongst them is the impostor while also completing tasks. In our experiments, crewmates "finished tasks" by going to the same location as a task. Similarly, impostors were able to kill crewmates if they were able to go near the crewmates.

For our training algorithm, we decided to use OpenAI’s MADDPG implementation, which is the multi-agent version of the Deep Deterministic Policy Gradient (DDPG) method detailed by Lowe et al. [2020]. The difference between simply extending single-agent algorithms for multi-agent environments is that single-agent algorithms would be independently updating policies rather than taking into account the policy of the other cooperative or adversarial agents. As mentioned in the above paper, MADDPG allows for training an agent’s actor using its own observations, but the critic is trained using the actions and policies of all agents. We decided to see the effects of using a single-agent algorithm like DDPG and compare it to the performance of MADDPG for our Among Us replication.

While our primary goal was to adapt a multi-agent algorithm to an Among Us-style environment, we also explored how limiting perception and tweaking the reward function would affect performance. We were curious to explore the effects of using one-time vs persistent rewards when awarding benefits



(a) Scaling agent proximity rewards



(b) One time rewards vs Persistent rewards

Figure 1: Two types of rewards

for finishing a task versus killing a crewmate. The first two sections explore our changes and the results in performance for a particle environment and gridworld environment.

Lastly, we also implemented a voting mechanism to allow crewmates to vote out an agent that they think is an impostor. The last section details how we used Bayesian Inference in our final implementation.

3 Using the Multi Agent Particle Environment

With this environment, we implemented a hide and seek game where agents would know who the adversary was, and the adversary's only goal is to "kill" agents by catching up to them.

3.1 Modifying the Environment

The Multi-Agent Particle Environment repository [4] provided environments with collaborative agents that sought out landmarks and adversaries that would aim to be near agents and landmarks. Agents received rewards for being near specific "goal" landmarks, and adversaries received rewards for being near agents and for finding the "goal" landmarks and being near them. The adversary reward would be negative, and would count against the total episode reward, so a dip in rewards would indicate better adversary performance or poorer agent performance. Agents would learn to split off and distract an adversary while the others collected the reward for being near the goal landmark.

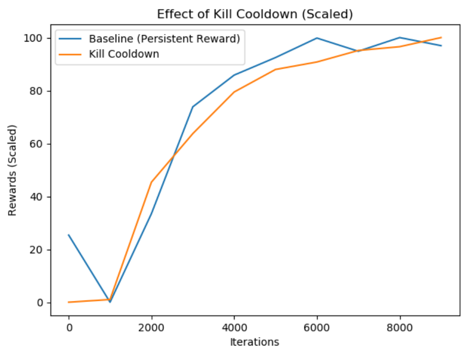
We started off with the available simple adversary scenario and modified it in various ways described below. With this environment, we wanted to better understand the impact of changing variables, internal mechanisms, and algorithms on the rewards that agents could achieve.

3.2 Experiments and Results

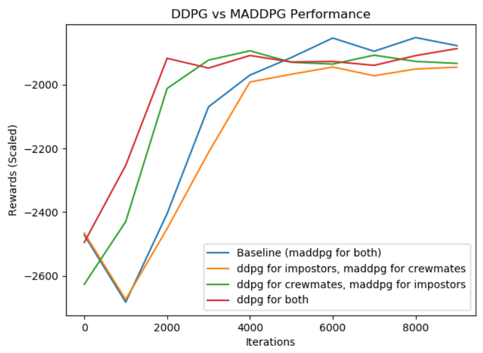
Our baseline included modified proximity rewards, both one time and persistent rewards, kill cooldowns, ghosts (so agents could continue performing tasks after death with a large reward reduction), variable landmarks, and unique landmark sequences assigned to each agent

Our first post-milestone experiment involved tweaking proximity rewards. We define a proximity reward as a persistent reward granted to adversaries for being near (alive) agents and to agents for being near (unvisited) goal landmarks. When we scaled the agents' proximity rewards down to reduce the impact of path finding on decision making, we found a large increase in adversary performance in later episodes, while scaling the proximity rewards up by a factor led to reaching the same early peak in agent performance, with the same plateau as the baseline. This implies that using lower values for proximity rewards may decrease collaborative performance in similar environments, as the agents would not learn to distract an adversary.

[4]https://github.com/openai/multiagent-particle-envs



(a) Kill Cooldown functionality



(b) MADDPG vs DDPG

Figure 2: Kill Cooldown analysis and Algorithm Usage analysis

We also tested the effect of one time rewards for killing agents vs persistent rewards for an action. So, when an adversary killed an agent, we would either provide a lump sum reward or a consistent reward spread out over the remaining time steps. With 10000 timesteps, we found that a one time reward led to slightly higher adversary performance. This may be because the trainer was not able to distinguish between proximity rewards and persistent kill rewards. It also became clear that with one time rewards, the average adversary reward was smaller in earlier stages of training, but became higher in the later stages

As a sanity check, we added a kill cooldown functionality to see if the adversaries would perform worse as the agents learned to optimize pathfinding better. As expected, we noticed no significant changes earlier in training, but as the training progressed, the agents steadily began to outperform the adversary. Even though the adversary would be reward for being near the agent, they would lose 4-5% of the final reward in the last thousand iterations (as seen in Figure 2a) because it couldn't kill the agent, while the agent recouped its baseline losses by staying alive and continuing to gain rewards unimpeded.

Finally, while modifying which policy the adversary and agents used, we tested the effect of MADDPG and DDPG. When both agents and adversaries used the same policy, the rewards tended towards the same value at 10000 iterations. When agents used DDPG, they performed worse, but when a single adversary used DDPG, it performed better than the baseline. This may be because Deep Deterministic Policy Gradient is not well suited to multi agent, collaborative scenarios, while MADDPG is.

3.3 Takeaways

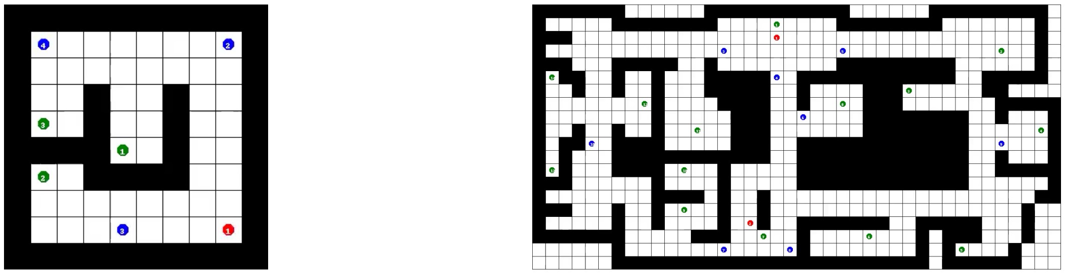
We were able to determine the benefit of using MADDPG instead of DDPG, and vice versa, the impact of kill cooldown and scaled proximity rewards, as well as the effect of switching between one time rewards and persistent rewards. Some of these determinations were significant enough to lead us to using kill cooldowns, proximity rewards, and multiple landmarks.

The particle environment was proving quite difficult to convert to a discrete grid-world environment, and even adding elements such as walls were proving way too difficult. We also ran into difficulties with rerunning and displaying experiments, which made analysis an issue. This is why we decided to switch over to a grid-world environment, explored in depth in the next section.

4 Grid-world approximation of Among Us

4.1 Introduction

While the particle environment yielded interesting results, we were ultimately hindered in our attempts to transform that environment to create a better representation of the Among Us game. We used



(a) Medium Grid

(b) Skeld Grid

Figure 3: Among Us style gridworld environment. The black tiles represent walls. The green circles are stationary and represent the task locations. The blue circles represent the crewmates and their goal is to go to every task location. The red circle represents the impostors and their goal is to go to every crewmate and kill them.

ma_gym [5], a publicly available grid-world environment, and made modifications to it to create a better representation of the Among Us game. Just like the previous section, this section implemented a Hide and Seek variant, where "crewmates" seek to go to task locations, while "impostors" seek to kill crewmates. Crucially, unlike the previous section on particle environments, the crewmates do not know the identity of the impostor among all agents, and the environment is only partially observable for each agent.

4.2 Modifying the Environment

Ultimately, one of our significant areas of work ended up being the contributions made towards creating an environment compatible with openai gym that best simulates the mechanics of the Among Us game. Building off of the PredatorPrey environment provided in ma_gym, we rewrote many portions to better accommodate to the movement and reward dynamics of the game. We created custom maps, and while most of our experiments used the Medium grid of size 10x10, we also ran a trial on a map that resembles the "Skeld" map in Among Us (as seen in Figure 3). In addition to keeping track of tasks and lives, we added a kill functionality as well as a kill cooldown for the impostor. Unlike the previous particle environment, this gridworld environment provided for an explicit kill action for the impostor as opposed to automatically killing crewmates when they are in the vicinity of an impostor. We added the ability to specify persistent or one-time rewards for crewmates finishing tasks and impostors killing crewmates. We modified the partially observable space to include tasks and all agents. We ensured that an impostor can differentiate between fellow impostors and crewmates but a crewmate's observation does not. Each episode terminates when either the impostor kills all the crewmates, or the crewmates collectively go to all task locations.

4.3 Experiments and Results

For the experiments described below, we used the 10x10 grid Medium grid with fixed starting positions (see Figure 3a) for 3 crewmates, 1 impostor, and 3 tasks. By default, persistent rewards were turned off, the kill cooldown was 5. The crewmates would get a reward of 500 for finishing a task, the impostor would get a reward of 500 for killing a crewmate, and all agents would get a reward of 0.5 for staying alive every step. We found that the best value for max episode length was 200, along with training for 10000 episodes using a learning rate of 0.01 for the training code.

To obtain our baseline we ran the code using the above the default parameters. It was observed that after 5000 iterations, the crewmates figure out how to navigate the grid and learn to finish their tasks before being killed by the impostor. The impostor learns to kill crewmates as soon as possible, but is only able to kill one or two of them before all the tasks are achieved. As a result, the crewmates continue to win the game towards the end of the training iterations. The plot (see Figure 4) depicts the average reward for crewmates and the impostor, and the difference between the two is referred to

[5]https://github.com/koulanurag/ma-gym

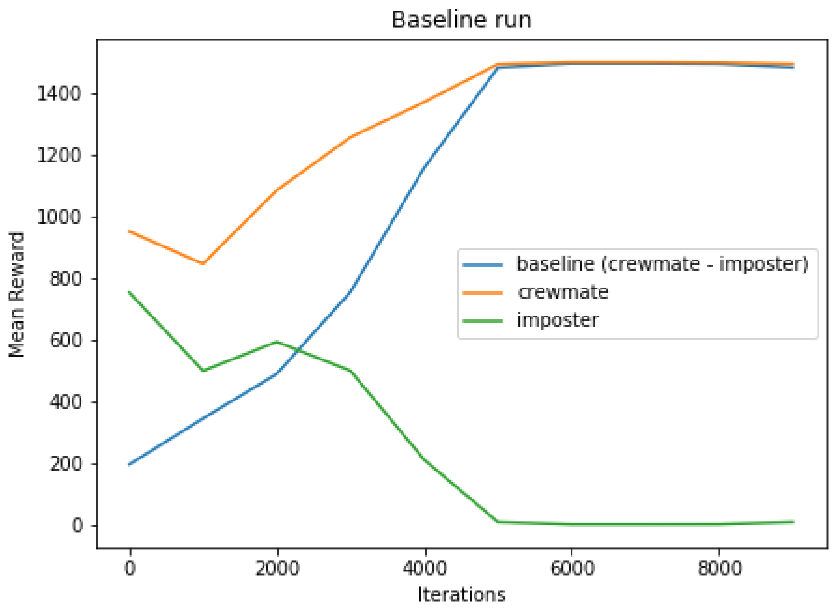


Figure 4: Plot for baseline run using default parameters

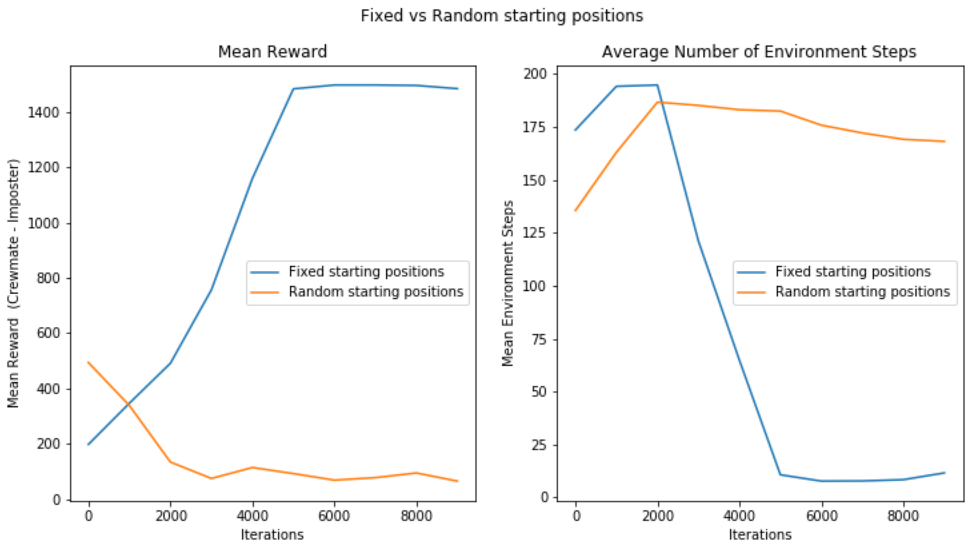


Figure 5: Performance comparison for fixed vs random starting location initialization. The left plot with the rewards (average crewmate minus average impostor) shows that the fixed setting leads to greater rewards for crewmates. The right plot for number of environment steps show that the random setting takes more steps on average, indicating that neither side is consistently winning a majority of games.

as the baseline. For latter plots, the reward curves plotted will depict the average crewmate reward minus the average impostor reward.

For the first experiment, we explored the effect of randomly initializing the starting positions for the tasks, crewmates, and impostor as compared to having them start in fixed positions at the start of each episode. We theorized that training with fixed starting positions would lead to better performance

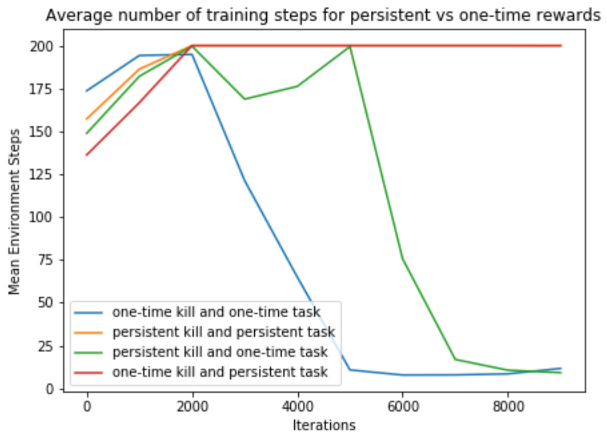


Figure 6: Performance comparison for persistent vs one-time rewards for killing crewmates (impostor's reward) and completing tasks (crewmates' reward). The plot depicts the average number of training steps taken, with a lower number indicating that the episode finished earlier since one side finished all their goals. The baseline (one-time rewards for both killing and tasks) usually ends in a crewmate win, so the plot suggests that using persistent rewards may not be yielding much improvement.

as the agents would learn to navigate the environment better, which is why we used it as the default for the baseline and for the latter experiments. As seen by Figure 5, using random starting positions resulted in lower rewards. Furthermore, the number of training steps taken shows that the random initialization took longer to run, indicating that neither side was able to settle on an unbeatable strategy. In the vein of the classic explore-vs-exploit analogy, we would think that using random starting positions would lead to better generalization for different maps, but using fixed starting positions would be a better representation of the game.

For the second experiment, we explored the affect of using persistent rewards instead of one-time rewards when an impostor kills a crewmate or a crewmate finishes a task. For the plot (see Figure 6), we didn't plot the raw rewards since they were scaled at different values. Like the previous experiment, we felt that looking at the average number of environment steps gives us an indication of the level of performance either side achieves. Since the baseline (using one-time rewards for both kills and tasks) resulted in crewmate dominance toward the end, its notable that both runs with persistent task rewards resulted in longer training periods and poorer crewmate dominance. While we cannot outright dismiss persistent rewards purely looking at number of training steps, this method provides one way of comparing performance.

For the last experiment, we compared the effect of using MADDPG vs DDPG. Looking at Figure 7, we see that for the three crewmates, using DDPG performs worse than when using MADDPG. However, for the single impostor, using DDPG performed better than MADDPG. It would be interesting to see whether DDPG performs better when there are multiple impostors as well.

4.4 Takeaways

We were able to create a gym-compatible environment that mimics the game mechanics of the game Among Us. Through various experiments on the environment, we were able to successfully train a model where the crewmates consistently win. We were able to get better insight into the effects of random starting point initialization, the use of persistent rewards, and how MADDPG benefits multiple agents compared to DDPG.

While we got some good insight into how the various game mechanics play affect model performance, in the next section we explore implementing what we consider the most differentiating aspect of the Among Us game, having the crewmates learn to identify and vote out who they think the impostor is.

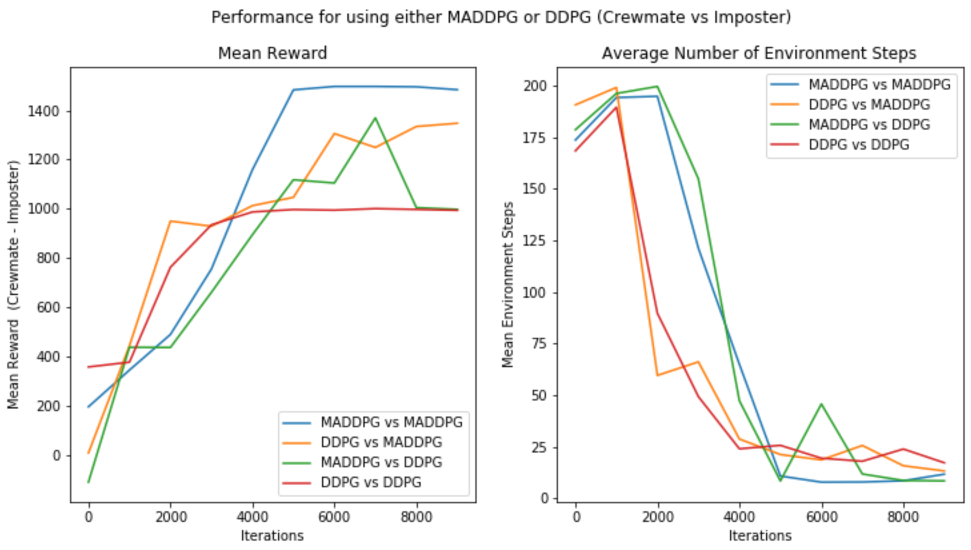


Figure 7: Performance comparison for using MADDPG or DDPG. Given the same algorithm for the impostor, we notice that the crewmates perform better when using MADDPG (on this plot higher reward is better for crewmates). Similarly, when comparing against the same algorithm for the crewmates, the impostor seems to perform better when using DDPG than MADDPG.

5 Using Bayesian Inference to implement Impostor Voting

5.1 Bayesian Inference

We first interacted with Bayesian Inference while exploring the overcooked paper during our initial project search phase. Each agent used Bayesian Inference to determine what other collaborative agents were going to do in order to maximize their collaborative output. They would manually collect, and update, priors and likelihoods in order to then estimate a posterior.

For our implementation of Bayesian Inference, we chose to use a two DenseFlipout layer network combined with Monte Carlo estimations and KL divergence to better match the actual probability distributions of certain actions implying that a player is an impostor. The Flipout layer was initially coined and explored by Wen et al. [2018]

5.2 Impostor Voting

We assume c crewmates and i impostors for future runs with Bayesian Inference implemented.

As MADDPG initializes, we create c Inference models, and assign each one to one of the c crewmates. At every step, we add the returned (c+i-1) x (c+i-1) array to a training batch. For each batch, we assign a unique shuffle to ensure we don't overfit to the environment's preset format of automatically labeling the first i agents as impostors and the remaining agents as crewmates. After a predetermined amount of iterations of MADDPG, the algorithm then trains the model, and begins the monte carlo estimation. This is done by predicting a posterior distribution for each crewmate using $imposter_{x,y}$ as the event that agent x sees agent y as an imposter and $actions_{x,y}$ as the observed actions agent y has taken in agent x's observation radius

$$P = (imposter_{x,y}|actions_{x,y})$$

m times, and then averaging out those m predictions. Once done, we approached the final voting scheme one of two ways.

5.2.1 Single Crewmate Voting

To initially test our implementation, we used the final crewmate as a de facto leader and only used their votes to assess performance. They would cast a final vote accusing a crewmate of being an

m	Accuracy	Threshold	m	Accuracy	Threshold
	16.5%	0.0		17.9%	0.0
	16.5%	0.05		17.9%	0.05
	16.5%	0.1		17.9%	0.1
	16.5%	0.15		17.9%	0.15
	16.5%	0.2		17.9%	0.2
	16.2%	0.25		16.2%	0.25
10	15.6%	0.30	50	19.9%	0.30
	14.4%	0.35		15.9%	0.35
	12.4%	0.4		12.8%	0.4
	7.8%	0.45		30.8%	0.45
	8.3%	0.5		50%	0.5
	8.3%	0.55		33%	0.55
	12.5%	0.6		0%	0.6

Table 1: Guessing accuracy for De Facto leader voting format. As we increase the value of m, we notice an increase in performance when asking for higher confidence votes, which indicates we move closer to the actual posterior distribution

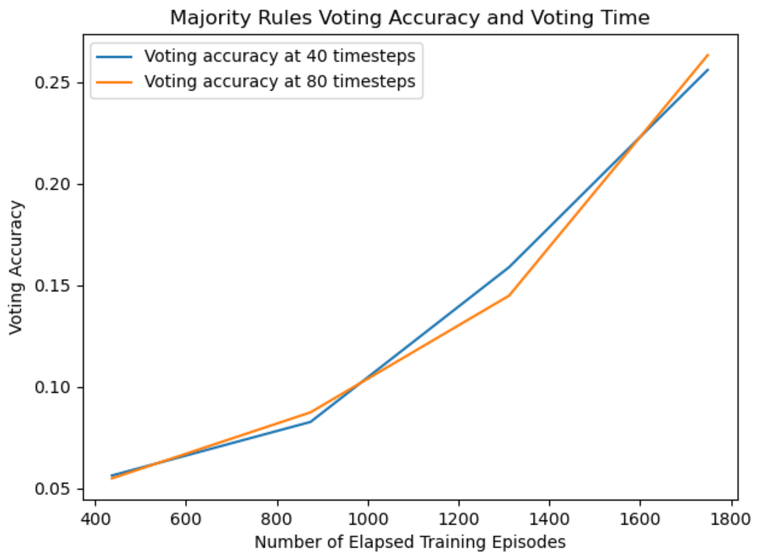
impostor if their confidence exceeded a certain threshold. This was to assess the impact of a single player who can influence a lobby to vote for a particular person if they are confident enough. We use the concept of a threshold to better understand the confidence of a particular vote, so if the maximum value of a distribution exceeds a threshold, we assume higher confidence in that vote and cast it. We found 0.3-0.5 to provide a higher accuracy than random guessing with c = 5 and i = 1, as shown in Table 1

5.2.2 Majority Rules Voting

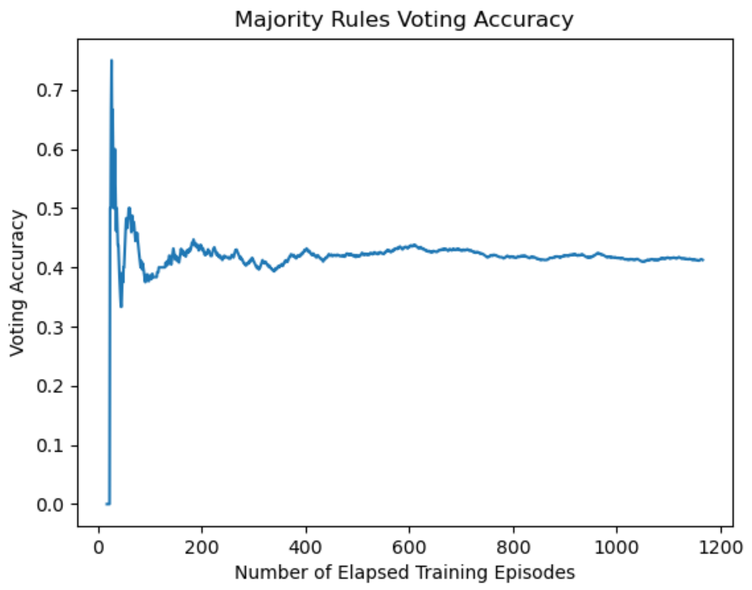
As the implementation evolved, we shifted to a majority voting format - if enough crewmates were confident enough about the same crewmate, they would cast a vote to accuse that crewmate of being an impostor. We evaluated this method in a few different ways. The first of which was a simple timestep analysis: we compared accuracy of voting after 40 timesteps to the accuracy of voting at 80 timesteps across different stages of training, as seen in Figure 8a. Here we see that in each quarter of training, the accuracy of the majority voting went up. While this was promising, success and failure in guessing was not impacting the agent's growth as much as it possible could, so we moved to a more integrated approach.

The second approach was as follows: we would flip a flag in the environment on a majority vote, and provide a massive reward for a correct majority and a large penalty for an incorrect majority. We decided to replace the need for a threshold with the cushion of a majority, and noticed immediate results. Because this was quite resource heavy, we did have to reduce c to 3 and i to 1. Doing this, we noticed that the accuracy would converge to about $42 \pm 1.5\%$, as shown in Figure 8b

In the end, with the combination of MADDPG and Bayesian Inference, we were able to show that, even with limited observability and minimal data pass through to a Bayesian net, we can produce successful estimates of impostor-like (adversarial) behavior, and positively influence voting behavior in crewmates (agents).



(a) Majority Rules Voting (not integrated), $c = 5$, $i = 1$



(b) Majority Rules Voting (integrated), $c = 3$, $i = 1$

Figure 8: Exploring Majority Rewards around the Context of Environment Integration

6 Conclusion

We made forks of three different repositories as detailed in the previous sections. They can be found here:

- https://github.com/285-Project-Multi-Agent/milestone
- https://github.com/285-Project-Multi-Agent/multiagent-particle-envs
- https://github.com/285-Project-Multi-Agent/ma-gym

6.1 Group Contributions

For this project, Aditya worked on building and running experiments on the Multi-Agent Particle Env implementation used in the first section, as well as the Bayesian Inference ideation, integration, and experimentation.

Mrunal worked on building Multi-Agent Particle Env implementation, as well as building and running experiments on the Grid World implementation in the second section.

6.2 Challenges

- Our initial idea and propopsal was initially based on an overcooked implementation from a paper by Wang et al. [2020], but the referenced github with the base code was taken down, which heavily impacted our progress
- Multi Agent Particle Environment was quite difficult initially to customize, and had a few hardcoded limitations
- Colab was a very useful, but often buggy and uncooperative, resource. We didn't have access to local computing resources, and we had to work around various Colab-related challenges, such as being able to render environments. The bugs related to rendering prevented us from proceeding forward with many environments, including multi-agent-emergence-environments [6], gym-minigrid [7] and gym-multigrid [8].

6.3 Future extensions

Given more time to explore our findings, we would want to increase the size of the Bayesian Inference networks, add further game mechanics like venting and faking tasks, further increase the number of agents, as well as improve the internal logging mechanisms to better account for fully remote research.

As undergraduates without prior research experience, we found this course to be greatly rewarding and a valuable experience. We are grateful for all the assistance provided by the professor and the course staff of CS 285.

[6]https://github.com/openai/multi-agent-emergence-environments
[7]https://github.com/maximecb/gym-minigrid
[8]https://github.com/ArnaudFickinger/gym-multigrid

References

Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning, 2020.

Max Kleiman-Weiner, M. K. Ho, J. L. Austerweil, M. L. Littman, and Joshua B. Tenenbaum. Coordinate to cooperate or compete: Abstract goals and joint intentions in social interaction. In *Proceedings of the 38th Annual Conference of the Cognitive Science Society*, 2016.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020.

Rose E. Wang, Sarah A. Wu, James A. Evans, Joshua B. Tenenbaum, David C. Parkes, and Max Kleiman-Weiner. Too many cooks: Bayesian inference for coordinating multi-agent collaboration, 2020.

Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches. *arXiv e-prints*, art. arXiv:1803.04386, March 2018.